

实验报告

PB15081576 蔡文韬

1. 功能设计

实现功能

- ☒ 不良网站拦截功能：在本地ip cache中找到对应ip为 '0.0.0.0'，返回域名不存在
- ☒ 服务器功能&中继功能：

具体实现可看Server.py中的class UDPHandler(socketserver.BaseRequestHandler)

```
class UDPHandler(socketserver.BaseRequestHandler):
    """
    https://docs.python.org/3.4/library/socketserver.html
    a udp handler for relay DNS
    """
    def handle(self):
        """
        :type local_cache: dict
        """

        data = self.request[0].strip()
        sock = self.request[1]
        analyzer = DNSAnalyzer(data) #对DNS报文进行分析

        if analyzer.question.qtype == 1:
            # 如果是request报文
            domain = analyzer.get_domain()
            if domain in local_cache:
                # 在本地
                ip = local_cache[domain]
                analyzer.set_ip(ip)
                if ip == "0.0.0.0":
                    # 此处打印相应信息
                else:
                    # 此处根据debug级别打印相应信息
            sock.sendto(analyzer.response(), self.client_address)
            # 将reply报文发送给client

        else:
            # relay
            public_request.append((sock, data.upper(),
            self.client_address))
            # 将请求报文压入栈，等待threading处理

        else:
            sock.sendto(data.upper(), self.client_address)
```

- ☒ 调试级别1: 输出ip和域名
- ☒ 调试级别2: 输入自选的public server和ip cache file (可以两个一起缺省, 使用默认值, 不能只缺省一个), 输出时间坐标, 序号, 域名和ip
- ☒ 调试级别3: 输入自选的public server (可以缺省), 输出接收和发送数据包的地址以及端口, 以及数据报的内容
- ☒ 多客户端并发&超时处理:
 - 实现方法: 使用了threading模块, 当handler接收到一个需要中继服务器的请求时, 就把它压入public_request栈中, 然后在Class Server的方法UDPThreading中pop出, 发送到中继服务器, 等待2s, 如果没有回复, 则timeout, 如果有, 则根据debug level输出需要的信息。id: 使用一个dictid来存放请求报文的id, 得到回复报文后, 将id从字典中取出, 打包好返回给client (否则会有id mismatch)

```
def UDPThreading(self):
    # start a loop to deal with task queue
    cur = 0
    while True:
        if len(public_request) > 0:
            if cur < MAXQUEUE: cur += 1
            else: cur = 0
            #队列

            sock, data, client_address = public_request[0]
            analyzer = dnsAnalyzer(data)
            id[cur] = analyzer.get_id()
            #id是一个dict, 给request分配一个cur, 将id存入

            self.sock.sendto(analyzer.request(cur),
self.public_server)

        try:
            reply, addr = self.sock.recvfrom(BUFSIZE)
        except socket.timeout:
            print("::: DNS timeout for 2 sec.\n")
            continue
            #超时
        else:
            reply_analyzer = DNSAnalyzer(reply)
            domain = reply_analyzer.get_domain()
            ip = reply_analyzer.get_ip(reply)
            #打印相应信息.....
            if testLev == 2:
                socTime = time.ctime()
                print("\n::: WHEN: {}".format(socTime))
                print("::: ID: {}".format(id[cur]))
            elif testLev == 3:
                -#打印相应信息.....
            rest = reply[2:]
            Id = id[cur]
```

```

        #将id从dict中取出
        reply = struct.pack("!H", Id) + rest
        sock.sendto(reply, client_address)

    public_request.pop(0)
    # FIFO

def run(self):
    """
    set up DNS relay server
    """
    Thread(target=self.UDPThreading).start()
    server = socketserver.UDPServer((self.ip, self.port), UDPHandler)
    server.serve_forever()

```

2. 模块划分

main.py: 提供prompt message, 根据用户输入信息进行判断, 进入Debug模块

Debug.py: 定义了三种测试等级(testL1(), testL2(), testL3()),

testL2后面的参数可以全部缺省, 表示使用public server和本地缓存, 但不可以只缺省一个

testL3的逻辑和testL2相同。提供ip valid check (ipv4), 用于二, 三级测试的情景。

将选定的public serve, 端口号和本地ip cache文件的地址提供给Server模块。

Server.py:

- **Class Server:**
 - **__init__:** 初始化
 - **loadCache:** 加载本地ip缓存文件
 - **run:** 启动服务器程序
 - **UDPThreading:** 多线程, 支持并发
- **Class UDPHandler:** 根据<https://docs.python.org/3.4/library/socketserver.html#socketserver-udpserver-example>上面的指导构建一个handler:
 - 在本地ip缓存中找到:
 - 找到且为'0.0.0.0': 报错
 - 找到且不为'0.0.0.0': 返回ip, 根据调试等级返回其他信息
 - 没有在本地图缓存中找到: 将当前的请求打包好, 加入队列中, 等待处理

DNSAnalyzer.py:

- **Class DNSAnalyzer:**
 - **__init__:** 用struct.unpack处理header (前12bytes)

```

def __init__(self, data):
    (self.Id, self.Flags, self.QdCount, self.AnCount, self.NsCount,

```

```
self.ArCount) = \
    struct.unpack('!HHHHHH', data[0: 12])
```

DNS HEADER

ID
FLAGS
QDCOUNT
ANCOUNT
NSCOUNT
ARCOUNT

并调用DNSQuestion分析请求部分

- `set_id`: 设置报文的id
- `get_id`: 返回header中的id
- `get_domain`: 返回DNS question中的域名
- `set_ip`: 填写header中其他部分
- `get_ip`: 返回回复报文中的ip
- `Class DNSQuestion`:
 - `__init__`: unpack question部分

```
class DNSQuestion:
    def __init__(self, data):
        i = 1
        self.domain = ''
        self.ip = ''
        while True:
            d = data[i]
            if d == 0:
                break
            elif d < 32:
                self.domain += '.'
            else:
                self.domain += chr(d)
            i += 1
        self.package = data[0: i + 1]
        (self.qtype, self.qclass) = struct.unpack('!HH', data[i + 1: i
+ 5])
        self.len = i + 5
```

DNS Question

QNAME
QTYPE
QCLASS

- `get_bytes`: 返回封装好的question部分
- `Class DNSRRF`:
 - `__init__`: 初始化Resource Record Format

```
class DNSRRF:
    def __init__(self, ip):
        self.name = 49164
        self.qtype = 1
        self.qclass = 1
        self.ttl = 200
        self.datalength = 4
        self.ip = ip
```

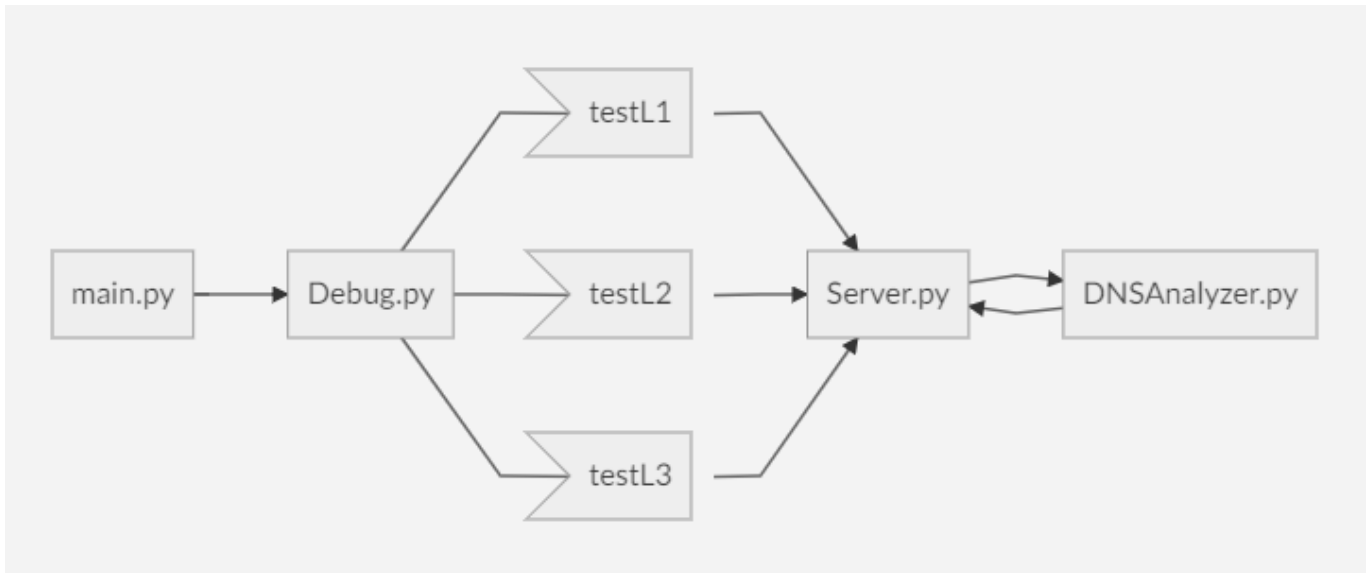
Resource Record Format

NAME
TYPE
CLASS
TTL
EDLENGTH
RDATA

rdata从第13个bytes开始。

- `get_bytes`: 返回封装好的rrf

3. 系统流程图



4. 程序运行说明

环境

linux, python3, 解释器: cpython

操作

在/src文件夹下

```
sudo python3 main.py
```

启动DNS程序

5. 测试用例及运行结果

简单测试

调试级别

- Debug level 1: 测试方法: 在一个terminal里运行`main.py`, 另一个terminal中 `dig domain @127.0.0.1` 黑名单:

```
rwxc2e@debian: ~/Projects/pythonGadgets/relay-dns/src
File Edit View Search Terminal Help
rwxc2e@debian:~/Projects/pythonGadgets/relay-dns/src$ sudo python3 main.py
:: This is a relay DNS
:: Enter: 'dnsrelay [-d][-dd] [dns-server-ipaddr] [filename]' to obtain ip
:: Enter: ctrl + c to stop program
> dnsrelay
-----
:: Error: domain 008.CN does not exist.
-----
-----
:: Error: domain 2QQ.CN does not exist.
-----
-----
:: Error: domain WWW.WIZISSOFT.COM does not exist.
-----
^[[^A
```

三个域名分别从ip cache（黑名单），ip cache和public server中得到。

```
rwxc2e@debian: ~/Projects/pythonGadgets/relay-dns/src
File Edit View Search Terminal Help
-----
:: ANSWER SECTION:
BAIDU.COM. 220.181.57.216
-----
-----
:: ANSWER SECTION:
USTC. 123.127.134.10
-----
-----
:: ANSWER SECTION:
WWW.SOHU.COM. 140.207.205.39
-----
```

- Debug level 2: 打印了时间和id

```
rwxc2e@debian: ~/Projects/pythonGadgets/relay-dns/src
File Edit View Search Terminal Help
rwxc2e@debian:~/Projects/pythonGadgets/relay-dns/src$ sudo python3 main.py
:: This is a relay DNS
:: Enter: 'dnsrelay [-d][-dd] [dns-server-ipaddr] [filename]' to obtain ip
:: Enter: ctrl + c to stop program
> dnsrelay -d 114.114.114.114 dnsrelay.txt
:: Entering debug level 2...
:: Selected public server: 114.114.114.114
:: Selected local cache: dnsrelay.txt
-----

:: ANSWER SECTION:

    USTC.      123.127.134.10

:: WHEN: Thu Dec 13 23:17:51 2018
:: ID: 44150
-----

:: ANSWER SECTION:

    BAIDU.COM.  220.181.57.216

:: WHEN: Thu Dec 13 23:18:06 2018
:: ID: 35390
-----
```

- Debug level 3: 打印了报文内容


```
rwxc2e@debian: ~/Projects/pythonGadgets/relay-dns/src
File Edit View Search Terminal Help
rwxc2e@debian:~/Projects/pythonGadgets/relay-dns/src$ sudo python3 main.py
:: This is a relay DNS
:: Enter: 'dnsrelay [-d][-dd] [dns-server-ipaddr] [filename]' to obtain ip
:: Enter: ctrl + c to stop program
> dnsrelay -dd
:: Entering debug level 3...
-----

:: Error: domain 2QQ.CN does not exist.
-----

:: ANSWER SECTION:

    TEST1.    11.111.11.111
:: SERVER: 127.0.0.1#53(127.0.0.1)
:: RAW DATA:b'.I\x81\x80\x00\x01\x00\x01\x00\x00\x01\x05test1\x00\x00\x01\x00\x01\x0c\x0c\x00\x01\x00\x01\x00\x00\x00\xbe\x00\x04\x0bo\x0bo'
-----

:: ANSWER SECTION:

    BING.COM.  13.107.21.200
:: SERVER: 127.0.0.1#53(127.0.0.1)
:: DATA:b'\x00\x01\x81\x80\x00\x01\x00\x02\x00\x00\x00\x00\x04BING\x03COM\x00\x00\x01\x00\x01\x0c\x0c\x00\x01\x00\x01\x00\x00\x03@\x00\x04rk\x15xc8xc0xc0\x01\x00\x01\x00\x00\x03@\x00\x04xcc0xc5xc8'
```

并发测试

可以使用/src下的脚本`dig.sh`

```
chmod +x dig.sh&&./dig.sh
```

用脚本`dig`一些网址，包含了ip cache中黑名单网址，其他网址和需要中继DNS的网址

`dig.sh`:

```
dig baidu.com @127.0.0.1
dig gin.ru @127.0.0.1
dig hotbar.com @127.0.0.1
dig www.9p.com @127.0.0.1
```

```
dig test1 @127.0.0.1
dig test2 @127.0.0.1
dig sohu.com @127.0.0.1
dig jd.com @127.0.0.1
dig sina.com.cn @127.0.0.1
dig weibo.com @127.0.0.1
```

运行dig.sh，相当于并发查询多个网站，处理结果良好。

```
rwxc2e@debian: ~/Projects/pythonGadgets/relay-dns/src
File Edit View Search Terminal Help

WWW.9P.COM.    199.59.242.151
-----

:: ANSWER SECTION:

TEST1.    11.111.11.111
-----

:: ANSWER SECTION:

TEST2.    22.22.222.222
-----

:: ANSWER SECTION:

SOHU.COM.    221.179.177.36
-----

:: ANSWER SECTION:

JD.COM.    120.52.148.118
-----

:: ANSWER SECTION:

SINA.COM.CN.    202.108.33.107
-----

:: ANSWER SECTION:

WEIBO.COM.    123.125.104.197
-----
```

替代本地DNS服务器

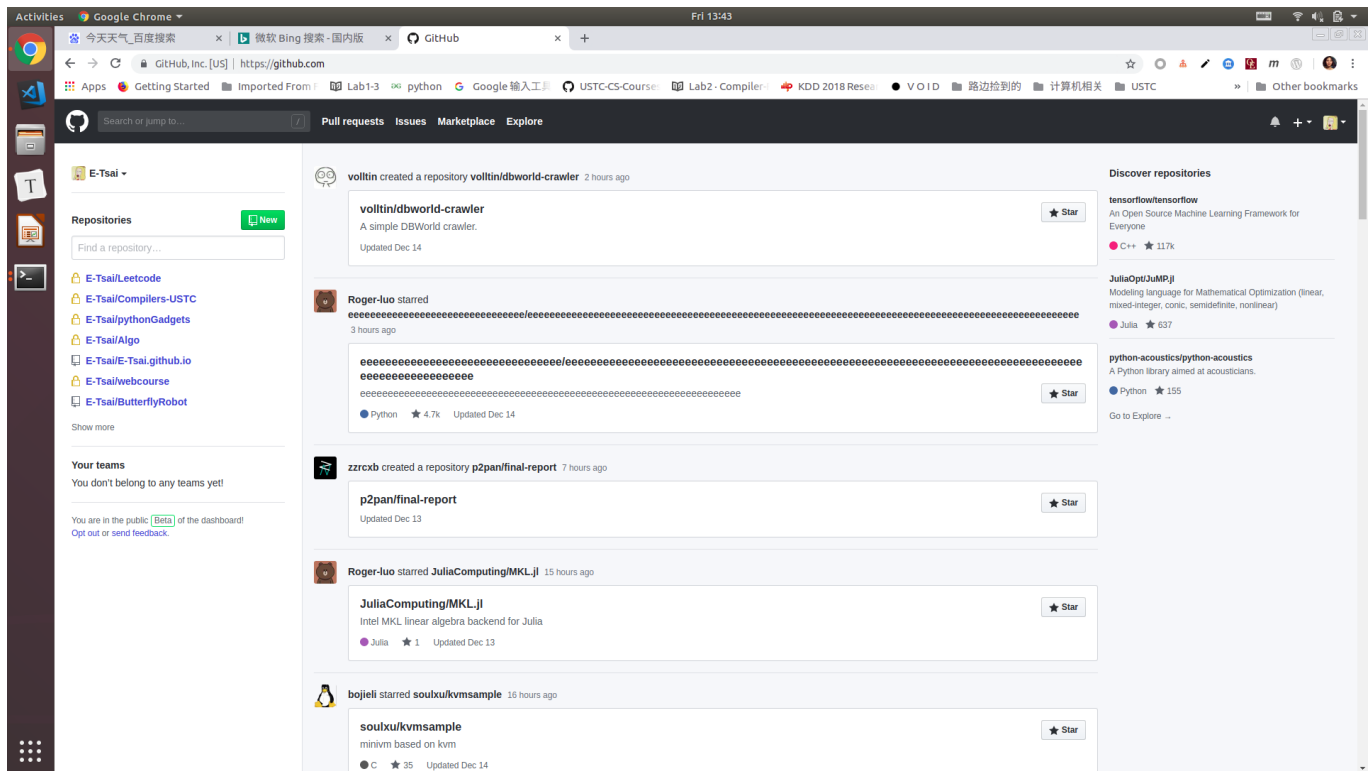
修改ubuntu中的dns文件(/etc/resolv.conf)

```
rwxpc2e@debian: /  
File Edit View Search Terminal Help  
rwxpc2e@debian:~$ cd /  
rwxpc2e@debian:/$ cat etc/resolv.conf  
# This file is managed by man:systemd-resolved(8). Do not edit.  
#  
# This is a dynamic resolv.conf file for connecting local clients to the  
# internal DNS stub resolver of systemd-resolved. This file lists all  
# configured search domains.  
#  
# Run "systemd-resolve --status" to see details about the uplink DNS servers  
# currently in use.  
#  
# Third party programs must not access this file directly, but only through the  
# symlink at /etc/resolv.conf. To manage man:resolv.conf(5) in a different way,  
# replace this symlink by a static file or a different symlink.  
#  
# See man:systemd-resolved.service(8) for details about the supported modes of  
# operation for /etc/resolv.conf.  
  
nameserver 127.0.0.53  
rwxpc2e@debian:/$ sudo vim etc/resolv.conf  
[sudo] password for rwxpc2e:
```

可以看到现在使用的是127.0.0.53，将它改成127.0.0.1 可以正常ping通：

```
rwxpc2e@debian: ~  
File Edit View Search Terminal Help  
rwxpc2e@debian:~$ ping baidu.com  
PING BAIDU.COM (220.181.57.216) 56(84) bytes of data.  
64 bytes from 220.181.57.216 (220.181.57.216): icmp_seq=1 ttl=48 time=290 ms  
64 bytes from 220.181.57.216 (220.181.57.216): icmp_seq=2 ttl=48 time=88.2 ms  
64 bytes from 220.181.57.216 (220.181.57.216): icmp_seq=3 ttl=48 time=75.0 ms  
64 bytes from 220.181.57.216 (220.181.57.216): icmp_seq=4 ttl=48 time=125 ms  
^C  
--- BAIDU.COM ping statistics ---  
4 packets transmitted, 4 received, 0% packet loss, time 3003ms  
rtt min/avg/max/mdev = 75.069/144.848/290.638/86.177 ms  
rwxpc2e@debian:~$ ping sohu.com  
PING SOHU.COM (221.179.177.36) 56(84) bytes of data.  
64 bytes from 221.179.177.36 (221.179.177.36): icmp_seq=1 ttl=54 time=305 ms  
64 bytes from 221.179.177.36 (221.179.177.36): icmp_seq=2 ttl=54 time=39.9 ms  
64 bytes from 221.179.177.36 (221.179.177.36): icmp_seq=3 ttl=54 time=44.5 ms  
64 bytes from 221.179.177.36 (221.179.177.36): icmp_seq=4 ttl=54 time=73.0 ms  
^C  
--- SOHU.COM ping statistics ---  
4 packets transmitted, 4 received, 0% packet loss, time 3003ms  
rtt min/avg/max/mdev = 39.957/115.751/305.521/110.294 ms  
rwxpc2e@debian:~$
```

正常使用浏览器：



6. 遇到的问题及心得体会

问题

dig baidu.com @127.0.0.1的时候遇到id mismatch 报错

```
dig baidu.com @127.0.0.1
;; Warning: ID mismatch: expected ID 26454, got 18262
;; Warning: ID mismatch: expected ID 26454, got 18262
;; Warning: ID mismatch: expected ID 26454, got 18262
```

dns程序返回:

```
-----
:: ANSWER SECTION:

BAIDU.COM.    123.125.115.110
-----
```

这个问题很奇怪，它好像是概率性地出现的：在我运行数十次时可能出现一次。我对id的处理原则是构建一个dict，将请求报文的id保存（得到回复报文后从字典中取出），它的key值是一个cur，从0到2047（同时支持2048个并发请求），之前的cur是局部变量，但是这样可能允许两个thread使用同样的key值，所以出现覆盖，id mismatch就出现了。将cur设定为全局变量后就没有这个问题了。

体会

我在写程序的过程中在以下网站学习了： socket编程: [python socket tutorial: https://realpython.com/python-sockets/](https://realpython.com/python-sockets/) Threading: [python threading:https://docs.python.org/3/library/threading.html](https://docs.python.org/3/library/threading.html) UDPHandler: <https://docs.python.org/3.4/library/socketserver.html#socketserver-udpserver-example>

通过这次实验，加深了我对DNS报文的理解，也对socket编程有了初步认识。