## 内容

设计一个HTTP代理服务器，在指定端口监听来自被代理客户的请求，根据其中的URL内容，确定返回缓存还是请求原网页之后返回。

支持网站过滤：白名单和黑名单（以黑名单为主）

支持用户过滤：某些用户不允许访问某些网站（建立IP地址的映射关系）

网站引导：如果一个网站在引导列表中，将客户对某个网站的访问全部设置为对另一个网站的访问。

一个被代理的客户端的所有请求都会被发送到代理服务器上。

## 模型

## 基本的HTTP功能

1. 识别用户；
2. 服务器可以宕机，因此数据需要持久化，以便回到之前的状态；

## 类的设计（python）

基于HTTPServer

class ProxyHTTPServer:

成员变量:

whiteList

blackList //一个set，记录字符串形式的name，ip，数字形式的ip

usersBlackList //一个多重表结构

usersWhiteList

成员函数：

checkVisitAccess()

checkHasCache()

基于SimpleHTTPRequestHandler

class ProxyHTTPRequestHandler

成员变量：

serverAddress

httplib2.request serverRequester

优先顺序：

1.如果whiteList不为空，所有用户都仅能访问whitList中的网站；

否则，所有用户都不能访问blackList中的网站

2.检查当前用户，如果其whiteList不为空，则仅能访问其whiteList上的网站；否则，不能访问其blackList中的网站

处理过程：

checkVisitAccess(…)

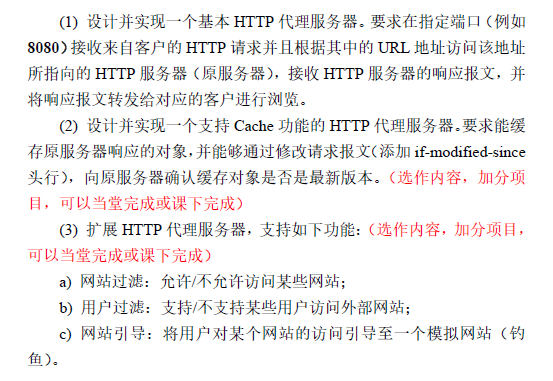
if(checkHasCache())

do\_request\_cache()

handle\_request()

整个流程实现网站过滤、用户过滤和网站引导的语义。

## 详细要求



## 代理功能实现

代理转发的请求是: "CONNECT www.oschina.net:443 HTTP/1.1"

两种风格的代理:CONNECT风格和完全中间人风格.

* CONNECT风格

处理流程: 重新定向到中间人风格

Proxy-Connection: keep-alive

处理头部Proxy-Connection

处理Host

CONNECT的处理逻辑: 一旦建立连接成功, 就让两个socket直接相互通信而不必经过代理服务器本身的处理.

RFC: <https://tools.ietf.org/id/draft-luotonen-web-proxy-tunneling-01.txt>

首先需要建立一个到远程服务器相应端口socket连接,获得一个socket类型

与本地已经建立的连接rfile,wfile进行数据交换

如何连接两个流?(进行流同步)

首先要检测流的数据, 每当监测到流的数据传输,规定在TIMEOUT时间内将数据传输完毕;然后将这个数据传输到另一方.

syncDoubleStream(instream,outstream):

检测数据发生:

bufsize=1024

while True:

s=instream.recv(1)

if len(s)==0:

break

outstream.sendall(s)

while True:

data,\_,\_,\_=instream.recvmsg(bufsize)

if len(data)==0:

break

outstream.sendall(data)

响应: HTTP/1.1 200 Connection Established

* 中间人风格

处理流程: 识别源端口,规定一个源端口号只能对应一次会话请求.

向目标主机发送请求,然后等待消息传回,再将内容传回主机,关闭一次会话.

需要用到:源端口号 目标主机 目标网址 所有的headers

* 防止递归

如果一个请求是指向代理服务器本身的,则这个请求不会被处理,发送404.或者使用非代理模式进行处理即可.

## 缓存

对于一个网址,它有可能被访问过而得到缓存.

* 代理服务器何时缓存

当从某一个目标服务器上收到Last-Modified头的时候, 它会进行缓存,然后再下一次发送的时候进行发送.

* 缓存如何被使用(仅当方法GET的时候使用)

利用相同的请求行, 和头信息,添加If-Modified-Since进行请求;如果收到304 Not Modified,就发送缓存的消息;

否则,更新缓存

* 代理服务器还应当处理来自客户端的If-Modified-Since

因为客户端可能同样实现了缓存机制, 如果代理端没有相应的缓存,就完整转发整个请求,其他部分不变.

* 缓存管理

缓存的唯一标识是 网址:端口/路径

使用host:port作为关键字, path作为次关键字,Last-Modified作为键值

应当从缓存文件夹中读取

* 关于Last-Modified

日期格式为

* 缓存服务器处理逻辑

处理过程:do\_ALL

递归访问? 返回本地的文件

是get方法

是条件get

当前缓存存在且条件日期大于缓存日期

返回304

结束

否则

保持 if-modified-since不变

否则不是条件get,如果当前缓存存在

在头部中添加if-modified-since头

否则既不是条件get, 也没有缓存

返回一般处理结果

#如果程序执行到这里,说明尚未处理完毕,但头部已经准备完毕,并且是一个条件get

调用do\_CONDITIONAL\_GET

结束

否则,不是get方法,返回一般的处理结果

do\_CONDITIONAL\_GET:

发送请求到目的主机

如果返回304

发送200

设置Last-Modified

读取本地文件返回给客户端

结束

如果返回200且有Last-Modified头部

更新本地文件的Last-Modified

更新内容

将服务器端响应发送给客户端

do\_GENERAL\_COMMAND()

请求服务器端连接

返回服务器端的所有结果

* 对于不支持缓存的服务器的处理

一个不支持缓存处理的服务器从来都不会发送304, 但是200的时候它有可能发送Last-Modified, 所以更新仅应当在真正需要的时候进行,不能被所有的Last-Modified要求强制更新

## 网站引导功能实现

设置一个两个dict数据:client\_map,server\_map, 以cache\_host作为键, 值类型有以下三种:

1. Not\_Allowed 不允许访问
2. Redirected 重定向, 其定向值在Redirect\_Map中

如果一个用户来自Not\_Allowed, 则直接返回404(黑名单方式实现)

如果一个服务器目标地址来自Not\_Allowed 也返回404

如果来自Redirected,则将请求中的Host替换掉

## API参考

判断IP相等:

socket.gethostbyname(hostname) ==

解析地址

处理Transfer-Encoding: chunked的办法: del it -- 旨在提升服务器的鲁棒性

os.path.join(‘a’,’/c’) -- 返回/c而不是a/c, 所以在路径的前面不要加上/

## GUI界面

# 附录：Python下Socket编程

## 客户端

import socket

s = socket.socket(socket.AF\_INET,socket.SOCK\_STREAM) //TCP

port = socket.getservbyname(‘http’,’tcp’)

s.connect((‘www.baidu.com’,port))

s.getsockname()

s.getpeername()

或者使用

httplib2这种高级的库

## 服务器端

s = socket…

s.bind((host,port)) //host=’’表示所有本地IP端口

s.listen(n)

while True:

clientsocket, clientaddr = s.accept()

…//new process

# 附录：Python下Web编程

类体系：(python 3.5)

http/server.py

HTTPServer(server\_addr,RequestHandlerClass) (derived from socketserver.TCPServer <--- socketserver.BaseServer)

serve\_forever() 🡪 runs forever

BaseHTTPRequestHandler(request,client\_addr,server)

成员变量：

client\_address

server(server instance)

close\_connection(Boolean)

requestline(request line without CRLF)

command(‘GET’)

path

request\_version

headers(不是dict类型,而是MessageClass)

rfile(input stream)

wfile

属性：

server\_version(格式：BaseHTTP/0.2)

sys\_version(python的版本，如果Python/3.6)

成员函数：

handle() call handle\_one\_request once

handle\_one\_request()

send\_error(code[,message])

send\_response(code[,message]) //Server和Date 从version\_string() date\_time\_string()中获取，也就是说除了头之外，还有额外的两条消息

send\_header(keyword,value)

send\_response\_only(code,message=None) //用于100 Continue这样的响应消息

end\_headers() adds a blank line to buffer and calls flush\_headers

flush\_headers()

log\_error() calls for log\_message

log\_message(format,…) 打印消息到sys.stderr

date\_time\_string([float\_time]) 返回GMT格式时间

address\_string()

SimpleHTTPRequestHandler

属性：

server\_version

extensions\_map 包含小写的MIME类型字符串

方法：

do\_GET() -- 自动映射到本地的资源文件

do\_POST()

do\_HEAD()

CGIHTTPRequestHandler

建立服务器监听端口：

import http.server

server = http.server.HTTPServer((‘’,8080),http.server.SimpleHTTPRequestHandler)

server.serve\_forver()

# 附录:Python3 Requests/cURL库的使用

<http://docs.python-requests.org/en/master/user/quickstart/#make-a-request>

requests.request(method,url.,\*\*kwargs)

kwargs : params 查询字符串 dict或者bytes

data

json body中

headers 头部,不一定是dict类型

cookies

files

stream False.则内容会被立即下载

* Response

成员变量:

content byte类型;

text unicode编码的content

raw 在stream=True时可用

* cURL
* subprocess

out=subprocess.run([‘ls’,’-l’],stdout=suporcess.PIPE).stdout

print(out.decode())

或者

p = subprocess.Popen(args,stdout=PIPE,stderr=PIPE)

p.stdout.read()

# 附录:Python下多线程模型

使用 concurrent.futures

<https://docs.python.org/3/library/concurrent.futures.html>

使用示例

**with** ThreadPoolExecutor(max\_workers=1) **as** executor:

future = executor.submit(pow, 323, 1235)

print(future.result())

**cancel**()

Attempt to cancel the call. If the call is currently being executed and cannot be cancelled then the method will return False, otherwise the call will be cancelled and the method will return True.

**cancelled**()

Return True if the call was successfully cancelled.

**running**()

Return True if the call is currently being executed and cannot be cancelled.

**done**()

Return True if the call was successfully cancelled or finished running.

**add\_done\_callback**(*fn*)

Attaches the callable *fn* to the future. *fn* will be called, with the future as its only argument, when the future is cancelled or finishes running.

Added callables are called in the order that they were added and are always called in a thread belonging to the process that added them. If the callable raises an [Exception](https://docs.python.org/3/library/exceptions.html#Exception) subclass, it will be logged and ignored. If the callable raises a [BaseException](https://docs.python.org/3/library/exceptions.html#BaseException) subclass, the behavior is undefined.

If the future has already completed or been cancelled, *fn* will be called immediately.

# 附录:实验

实验1、实验3、实验4、实验5、实验6、实验8.其他实验选作。

# 附录：Python内置数据类型持久化—json

import json

json.dumps()

json.loads()

# 附录：HTTP几种头信息详解

URL解析： 主机 端口 路径 附带参数

方法解析：POST GET

常用头

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 意义 | 取值 | 是否必须 | 代理服务器需用到 |
| Host | 主机名称 |  | 必须 | YES |
| Date | 请求的时间 | GMT格式 |  | YES |
| Server |  |  |  |  |
| User-Agent |  |  |  |  |
| Connection | 是否保持连接 | keep-alive  close |  |  |
| Content-Length |  | 数字 | 响应必须 |  |
| Content-Type |  |  |  |  |
| Last-Modified |  |  |  | YES |
| If-Modified-Since | 条件GET |  |  | YES |
| Set-Cookie |  |  |  |  |
| Cookie |  |  |  |  |
|  |  |  |  |  |

HTTP响应

|  |  |  |  |
| --- | --- | --- | --- |
| 响应码 | 消息 | 含义 | 代理服务器 |
| 200 | OK | 请求被正常处理 |  |
| 301 | Moved Permanently | 永久转移 |  |
| 302 |  | 资源被临时转移 |  |
| 304 | Not Modified | 可以使用缓存 | YES |
| 400 | Bad Reqeust | 通用出错代码，服务器不能处理该请求 |  |
| 404 | Not Found | 资源不在服务器上 |  |
| 505 | HTTP Version Not Supported | HTTP版本不支持 |  |

缓存和条件GET：

1. 必须使用GET方法
2. 请求报文包含If-Modified-Since

满足上述两条语句，就称做条件GET。

如果服务器发送的头部中包含Last-Modified，则接收方可以缓存这个文件并在之后的请求中使用条件GET方法（即满足缓存的条件）

缓存服务器缓存的信息是（Last-Modified, Path）, 发送请求时总是先查询缓存，如果存在，取出时间，用作条件GET的时间发送。

日期格式：

Tue, 09 Aug 2011 15:11:03 GMT

WEEKDAY, DATE MONTH YEAR HOUR:MINUTE:SECOND FORMAT

解析GMT格式的时间可以使用正则表达式 ^\s\*(\w+),\s(\d+)\s(\w+)\s(\d+)\s(\d+): (\d+): (\d+)\s(\w+)$,

依次提取其中的每一个组即可。

示例:

>>> import datetime

>>> datetime.datetime.strptime('Wed, 23 Sep 2009 22:15:29 GMT', '%a, %d %b %Y %H:%M:%S GMT')

datetime.datetime(2009, 9, 23, 22, 15, 29)

>>> import email.utils as eut

>>> eut.parsedate('Wed, 23 Sep 2009 22:15:29 GMT')

(2009, 9, 23, 22, 15, 29, 0, 1, -1)

If you want a datetime.datetime object, you can do:

def my\_parsedate(text):

return datetime.datetime(\*eut.parsedate(text)[:6])

# 附录：Scapy和其他工具的使用

## ls

进入scapy之后，使用

ls(TCP) , ls(IP) . . .可以得到所有协议包的字段和默认值

## help

## ip.show()

可以用来显示一个包的详情

## IP/TCP

可以用来组合包

## IP(str)

可以从原始数据到一个scapy包的转换

## packet.payload属性

这个属性的值仍然是一个payload,因此有可能使用 packet.payload.payload….chksum

如 p r i n t pk t . p a yl o a d . p a yl o a d . p a yl o a d . chksum

但是可以通过[]来访问，譬如

if UDP in packet:

print pkt[UDP].chksum

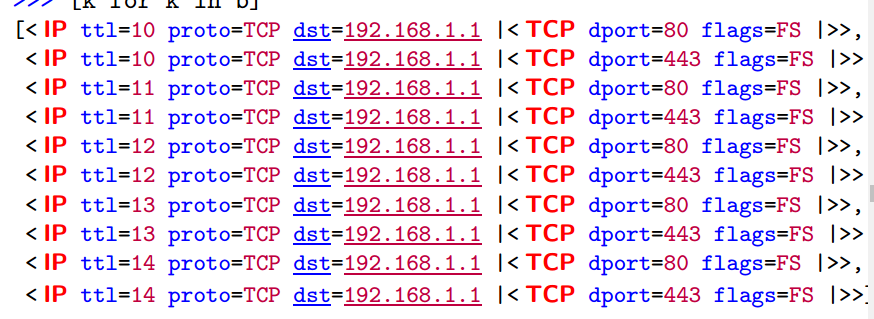
## 使用二元组设定多个值

packet.ttl=(10,14)

packet.payload.dport=(80,443)

这个时候packet成为一个迭代器，可以使用[]生成列表

[k for k in packet]



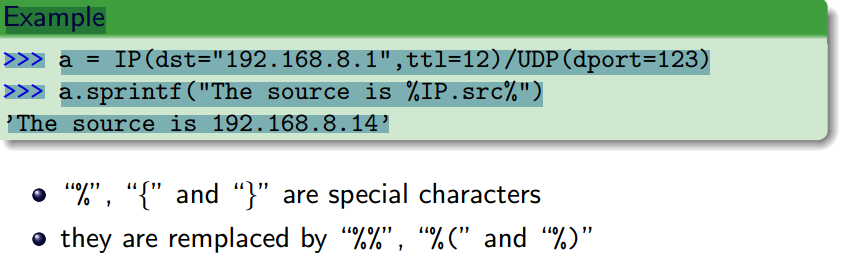
## 实用函数集

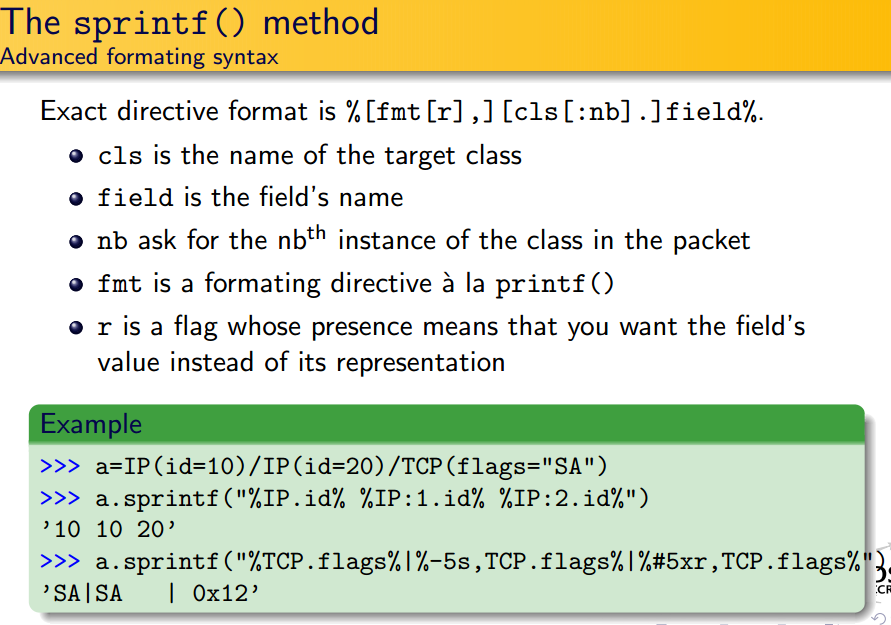
str()

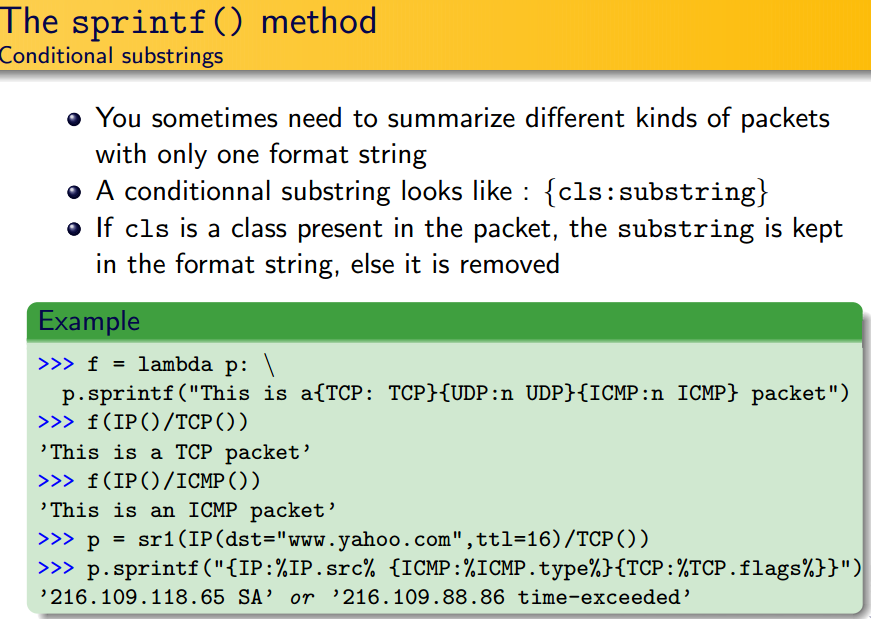
hexdump() ls() pkt.summary() pkt.show()（显示原始构造的包） pkt.show2()（显示出有效的构造包）

summary() 对包的概述

pkt.sprintf() 格式化打印属性







pkt.decode\_payload\_as() 改变承载数据的解释方式

pkt.haslayer() test if it has a layer

pkt.getlayer() return a given layer

## Configuration : conf

conf可以用来查看本地的配置情况

## 包的函数

send(pkt)

send([pkt]\*3) //重复发送

send(pkt,inter=0.1,loop=1) //循环发送直到按下Ctrl-C

sendp(“I’m using something at layer 2”,iface=”…”) //在数据链路层发送数据，其承载类型不定

ansed,notansed = sr(…) 发送和接收

RandShort() 随机产生一个short

## sniffer函数

sniff(count=5,filter=”tcp”) #filter 语法符合PCAP格式

可选参数有 prn 表示接收到一个包的动作 lambda x:x.summary()

filter

iface 选择网卡

常用sniff函数

summary

nsummary() 打印序号

conversations()

wrpcap(文件，a) 将a（包或者包的集合）存储到文件中

rdpcap(文件) 从文件读取

## 高级函数

tranceroute([[www.apple.com](http://www.apple.com),[www.microsoft.com](http://www.microsoft.com)])

# 使用virtualbox建立网络实验环境

## linux网卡操作

通过ubuntu的网络管理器可以很方便地建立一个虚拟网卡

## tcpdump

# 附录：网络操作的语法

## man pcap-filter

一个表达式返回一个bool值暗示对相应的包的测试条件的真假。

基本语素是： type dir proto id

id是所需要修饰的符号，type/dir/proto都是指定修饰的内容。

逻辑连接符号包括 and、or、not

相当于说明 id is type,id is dir and id is proto

前导修饰

type type qualifiers say what kind of thing the id name or number refers to. Possible types are host, net ,

port and portrange. E.g., `host foo', `net 128.3', `port 20', `portrange 6000-6008'. If there is no

type qualifier, host is assumed.

dir dir qualifiers specify a particular transfer direction to and/or from id. Possible directions are src,

dst, src or dst, src and dst, ra, ta, addr1, addr2, addr3, and addr4. E.g., `src foo', `dst net

128.3', `src or dst port ftp-data'. If there is no dir qualifier, src or dst is assumed. The ra, ta,

addr1, addr2, addr3, and addr4 qualifiers are only valid for IEEE 802.11 Wireless LAN link layers. For

some link layers, such as SLIP and the ``cooked'' Linux capture mode used for the ``any'' device and

for some other device types, the inbound and outbound qualifiers can be used to specify a desired

direction.

proto proto qualifiers restrict the match to a particular protocol. Possible protos are: ether, fddi, tr,

wlan, ip, ip6, arp, rarp, decnet, tcp and udp. E.g., `ether src foo', `arp net 128.3', `tcp port 21',

`udp portrange 7000-7009', `wlan addr2 0:2:3:4:5:6'. If there is no proto qualifier, all protocols

consistent with the type are assumed. E.g., `src foo' means `(ip or arp or rarp) src foo' (except the

latter is not legal syntax), `net bar' means `(ip or arp or rarp) net bar' and `port 53' means `(tcp or

udp) port 53'.

id可以缺少。

## 常用

port 80 -- 或者目的是80，或者来自80

host fulton – 如果主机是fulton

tcp udp icmp -- 如果包的协议是tcp/…

ip broadcast -- 如果包是广播的

ether multicast – 如果包是多播的

ifname/on [interface] – 如果包是来自于interface的

src/dst port [portnum] – 如果包来自于/目的地是portnum

net [net] -- 如果

gateway [gateway] -- 如果包使用gateway作为host

# 结束标志