

# 多平台抓包分析及防火墙操作实践

---

《计算机网络》研讨 20201014

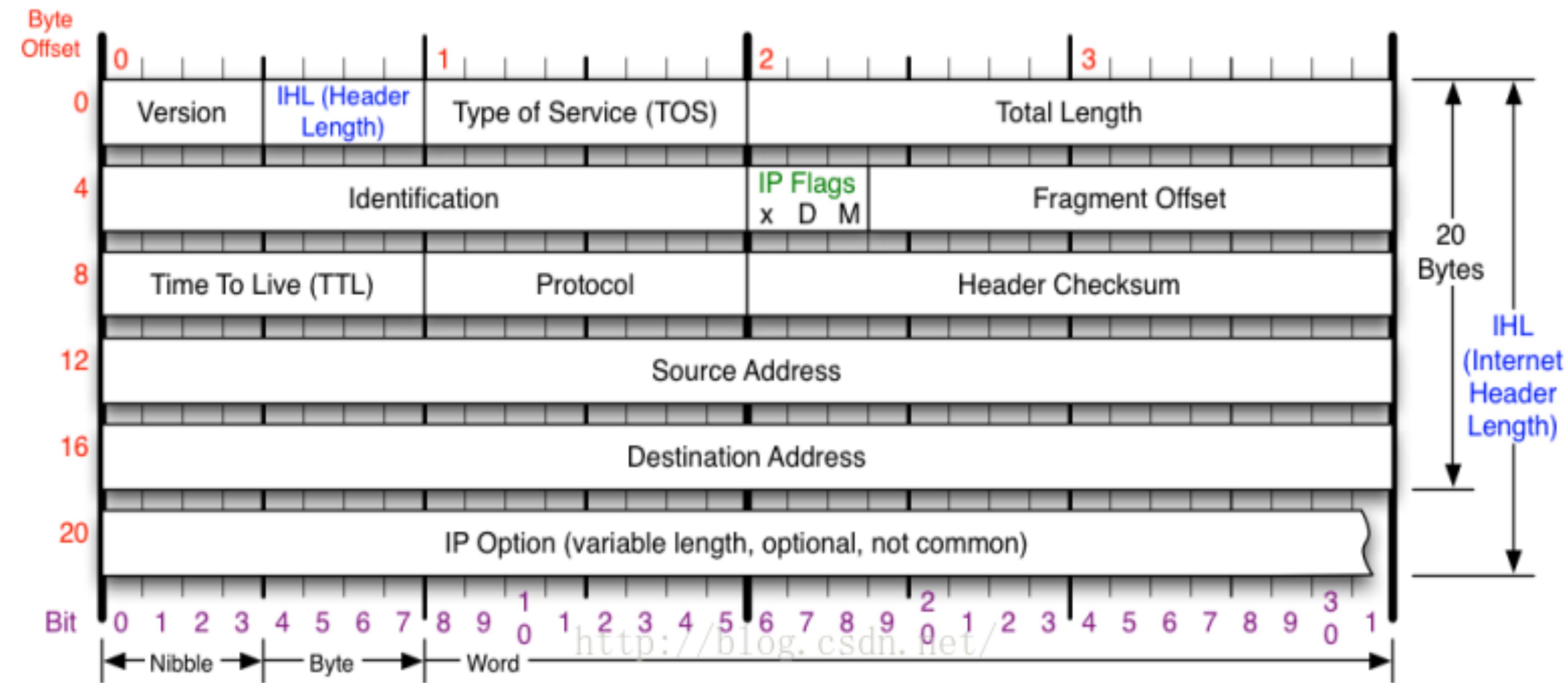
18120162 孙懿祺 18120172 王如嫣 18120189 林艺珺

# Ubuntu下 基于Python dpkt pcap的抓包分析

---

# Ipv4报文格式

- 可以从中看到版本号，标识符，源ip，目标ip等信息所在的报文位置



## Version

Version of IP Protocol. 4 and 6 are valid. This diagram represents version 4 structure only.

## Header Length

Number of 32-bit words in TCP header, minimum value of 5. Multiply by 4 to get byte count.

## Protocol

IP Protocol ID. Including (but not limited to):

1 ICMP	17 UDP	57 SKIP
2 IGMP	47 GRE	88 EIGRP
6 TCP	50 ESP	89 OSPF
9 IGRP	51 AH	115 L2TP

## Total Length

Total length of IP datagram, or IP fragment if fragmented. Measured in Bytes.

## Fragment Offset

Fragment offset from start of IP datagram. Measured in 8 byte (2 words, 64 bits) increments. If IP datagram is fragmented, fragment size (Total Length) must be a multiple of 8 bytes.

## Header Checksum

Checksum of entire IP header

## IP Flags

### x D M

x 0x80 reserved (evil bit)  
D 0x40 Do Not Fragment  
M 0x20 More Fragments follow

## RFC 791

Please refer to RFC 791 for the complete Internet Protocol (IP) Specification.

# Wireshark分析tcp三次握手

- 可以看到前三个数据包为tcp三次握手的过程。
- 客户端先向服务器端发送数据包，SYN=1，并且告诉服务器端当前发送序号为0。
- 服务器端随后进行ACK应答ACK=1，应答值为0+1表示确认，同时也发送SYN=1，告诉客户端发送序号为0
- 客户端收到ack=1后，客户端到服务端单向连接成功，状态为established，发送了ACK=1的应答包对于服务器端的SYN进行应答

760 36.754963	192.168.1.101	218.75.123.181	TCP	66 60950 → 80 [SYN] Seq=0 Win=64240 Len=0
762 36.764000	218.75.123.181	192.168.1.101	TCP	66 80 → 60950 [SYN, ACK] Seq=0 Ack=1 Win=132 Len=0
763 36.764160	192.168.1.101	218.75.123.181	TCP	54 60950 → 80 [ACK] Seq=1 Ack=1 Win=132 Len=0

# Wireshark分析tcp挥手

- 由这次抓包的情况看，这次tcp连接没有进行标准的四次挥手，而是省略了服务端第一个挥手应答。
- 客户端发送数据包中Fin=1
- 随后服务器回应ACK=1，同时发送FIN=1给服务器端
- 最后客户端回应服务器ACK=1的数据包

1839	180.959491	192.168.1.101	218.75.123.181	TCP	54 60950 → 80 [FIN, ACK] Seq=1163 Ack=6344 Wi
1840	180.970007	218.75.123.181	192.168.1.101	TCP	54 80 → 60950 [FIN, ACK] Seq=6344 Ack=1164 Wi
1841	180.970069	192.168.1.101	218.75.123.181	TCP	54 60950 → 80 [ACK] Seq=1164 Ack=6345 Win=131

# 用python实现pcap文件分析

```
Frame 3: 54 bytes on wire (432 bits), 54 bytes captured (432 bits) on interface \Device\NPF_{FA00DF3C-C62D-4A  
Ethernet II, Src: RivetNet_ba:6f:8f (9c:b6:d0:ba:6f:8f), Dst: Tp-LinkT_2c:e5:08 (48:0e:ec:2c:e5:08)  
Internet Protocol Version 4, Src: 192.168.1.101, Dst: 218.75.123.181  
Transmission Control Protocol, Src Port: 60950, Dst Port: 80, Seq: 1, Ack: 1, Len: 0
```

- Pcap文件开始以太网部分数据链路层报文，其次是ipv4协议网络层报文，最后是tcp协议传输层报文。
- 代码实现的是从pcap报文中提取了
  - 源mac和目标mac地址
  - 源ip地址和目标ip地址
  - 发送端口和接受端口的信息。

# 用python实现pcap文件分析

```
ethheader = buf[0:14]

dstmac = ethheader[0:6]

srcmac = ethheader[6:12]

netlayer_type = ethheader[12:14]

fw.writelines("dstMAC:"+str(binascii.b2a_hex(dstmac))+"\tsrcMAC:"+str(binascii.b2a_hex(srcmac))+"\n")

pktheader = buf[14:34]

trans_type = pktheader[9]

srcip = pktheader[12:16]

dstip = pktheader[16:20]

fw.writelines("dstIP:"+addr2str(dstip)+"\tsrcIP:"+addr2str(srcip)+"\n")
```

# 用dpkt实现pcap结果

```
timestamp:1602601996.951464      packet len:66
dstMAC:b'480eec2ce508'      srcMAC:b'9cb6d0ba6f8f'
dstIP:218.75.123.181        srcIP:192.168.1.101
packet type:TCP
srcport:60950    dstport:80
```

```
timestamp:1602601996.960501      packet len:66
dstMAC:b'9cb6d0ba6f8f'      srcMAC:b'480eec2ce508'
dstIP:192.168.1.101        srcIP:218.75.123.181
packet type:TCP
srcport:80      dstport:60950
```

```
timestamp:1602601996.960661      packet len:54
dstMAC:b'480eec2ce508'      srcMAC:b'9cb6d0ba6f8f'
dstIP:218.75.123.181        srcIP:192.168.1.101
packet type:TCP
```

```
srcport:60950    dstport:80
```

192.168.1.101	218.75.123.181	TCP	66	60950 → 80
218.75.123.181	192.168.1.101	TCP	66	80 → 60950
192.168.1.101	218.75.123.181	TCP	54	60950 → 80

# Windows 10下 基于WinPcap的抓包分析

---

## 开发环境：

Windows10系统上基于WinPcap开发

Visual Studio 2019

C++

时间	长度	源MAC地址	目的MAC地址	协议	源IP地址	目的IP地址
20/10/14 ...	105	48-0E-EC-2C-E...	28-C6-3F-D2-E...	TCP	49.234.153.138	192.168.1.104
20/10/14 ...	54	28-C6-3F-D2-E...	48-0E-EC-2C-E...	TCP	192.168.1.104	49.234.153.138
20/10/14 ...	105	48-0E-EC-2C-E...	28-C6-3F-D2-E...	TCP	81.69.172.214	192.168.1.104
20/10/14 ...	54	28-C6-3F-D2-E...	48-0E-EC-2C-E...	TCP	192.168.1.104	81.69.172.214
20/10/14 ...	105	48-0E-EC-2C-E...	28-C6-3F-D2-E...	TCP	49.234.153.138	192.168.1.104
20/10/14 ...	54	28-C6-3F-D2-E...	48-0E-EC-2C-E...	TCP	192.168.1.104	49.234.153.138
20/10/14 ...	105	48-0E-EC-2C-E...	28-C6-3F-D2-E...	TCP	81.69.172.214	192.168.1.104
20/10/14 ...	54	28-C6-3F-D2-E...	48-0E-EC-2C-E...	TCP	192.168.1.104	81.69.172.214
20/10/14 ...	42	48-0E-EC-2C-E...	FF-FF-FF-FF-F...	ARP	192.168.1.1	192.168.1.100

```
typedef struct iphdr
{
#if defined(LITTLE_ENDIAN)
    u_char ihl:4;
    u_char version:4;
#elif defined(BIG_ENDIAN)
    u_char version:4;
    u_char ihl:4;
#endif
    u_char tos;
    u_short tlen;
    u_short id;
    u_short frag_off;
    u_char ttl;
    u_char proto;
    u_short check;
    u_int saddr;
    u_int daddr;
    u_int     op_pad;
};
```

# VinPcap开发

```
#include "pcap.h"

main()
{
pcap_if_t *alldevs;
pcap_if_t *d;
int inum;
int i=0;
pcap_t *adhandle;
int res;
char errbuf[PCAP_ERRBUF_SIZE];
struct tm *ltime;
char timestr[16];
struct pcap_pkthdr *header;
const u_char *pkt_data;
time_t local_tv_sec;

/* 获取本机设备列表 */
if (pcap.findalldevs_ex(PCAP_SRC_IF_STRING, NULL, &alldevs, errbuf) == -1)
{
    fprintf(stderr,"Error in pcap.findalldevs: %s\n", errbuf);
    exit(1);
}

/* 打印列表 */
for(d=alldevs; d; d=d->next)
{
    printf("%d. %s", ++i, d->name);
    if (d->description)
        printf(" (%s)\n", d->description);
    else
        printf(" (No description available)\n");
}

if(i==0)
{
    printf("\nNo interfaces found! Make sure WinPcap is installed.\n");
    return -1;
}
```

目的IP地址
192.168.1.104
49.234.153.138
192.168.1.104
81.69.172.214
192.168.1.104
49.234.153.138
192.168.1.104
81.69.172.214
192.168.1.100

```
typedef struct iphdr
{
#if defined(LITTLE_ENDIAN)
    u_char ihl:4;
    u_char version:4;
#elif defined(BIG_ENDIAN)
    u_char version:4;
    u_char ihl:4;
#endif
    u_char tos;
    u_short tlen;
    u_short id;
    u_short frag_off;
    u_char ttl;
    u_char proto;
    u_short check;
    u_int saddr;
    u_int daddr;
    u_int     op_pad;
};
```

# Windows防火牆配置

---

# 配置环境：

GUI

腾讯云服务器 CVM

```
C:\Users\Administrator>ping 172.17.0.16
```

正在 Ping 172.17.0.16 具有 32 字节的数据:

来自 172.17.0.16 的回复: 字节=32 时间<1ms TTL=128  
来自 172.17.0.16 的回复: 字节=32 时间<1ms TTL=128  
来自 172.17.0.16 的回复: 字节=32 时间<1ms TTL=128  
来自 172.17.0.16 的回复: 字节=32 时间<1ms TTL=128

172.17.0.16 的 Ping 统计信息:

数据包: 已发送 = 4, 已接收 = 4, 丢失 = 0 (0% 丢失),

往返行程的估计时间(以毫秒为单位):

最短 = 0ms, 最长 = 0ms, 平均 = 0ms

```
C:\Users\Administrator>ping 172.17.0.16
```

正在 Ping 172.17.0.16 具有 32 字节的数据:

来自 172.17.0.16 的回复: 字节=32 时间<1ms TTL=128  
来自 172.17.0.16 的回复: 字节=32 时间<1ms TTL=128  
来自 172.17.0.16 的回复: 字节=32 时间<1ms TTL=128  
来自 172.17.0.16 的回复: 字节=32 时间<1ms TTL=128

```
C:\Users\Administrator>ping 172.17.0.9
```

正在 Ping 172.17.0.9 具有 32 字节的数据:

来自 172.17.0.9 的回复: 字节=32 时间<1ms TTL=128  
来自 172.17.0.9 的回复: 字节=32 时间<1ms TTL=128  
来自 172.17.0.9 的回复: 字节=32 时间<1ms TTL=128  
来自 172.17.0.9 的回复: 字节=32 时间<1ms TTL=128

172.17.0.9 的 Ping 统计信息:

数据包: 已发送 = 4, 已接收 = 4, 丢失 = 0 (0% 丢失),

往返行程的估计时间(以毫秒为单位):

最短 = 0ms, 最长 = 0ms, 平均 = 0ms

```
C:\Users\Administrator>
```

```
C:\Users\Administrator>ping 172.17.0.9
```

正在 Ping 172.17.0.9 具有 32 字节的数据:

请求超时。  
请求超时。  
请求超时。  
请求超时。

## 高级安全 Windows 防火墙

文件(F) 操作(A) 查看(V) 帮助(H)



## 本地计算机 上的高级安全 Win

- 入站规则
- 出站规则
- 连接安全规则
- 监视
  - 防火墙
  - 连接安全规则
- 安全关联

## 防火墙

名称	配置文件	操作	替代	方向	程序	本地地址
Cortana (小娜)	所有	允许	否	入站	任何	任何
Windows 远程管理(HTTP-In)	公用	允许	否	入站	Syst...	任何
放通PING--入站	所有	允许	否	入站	任何	任何
核心网络 - Internet 组管理协议(IGMP-In)	所有	允许	否	入站	Syst...	任何
核心网络 - IPv6 (IPv6-In)	所有	允许	否	入站	Syst...	任何
核心网络 - IPv6 的动态主机配置协议(DHCPv6-In)	所有	允许	否	入站	C:\W...	任何
核心网络 - 参数问题(ICMPv6-In)	所有	允许	否	入站	Syst...	任何
核心网络 - 超时(ICMPv6-In)	所有	允许	否	入站	Syst...	任何
核心网络 - 动态主机配置协议(DHCP-In)	所有	允许	否	入站	C:\W...	任何
核心网络 - 多播侦听程序报告 v2 (ICMPv6-In)	所有	允许	否	入站	Syst...	任何
核心网络 - 多播侦听程序报告(ICMPv6-In)	所有	允许	否	入站	Syst...	任何
核心网络 - 多播侦听程序查询(ICMPv6-In)	所有	允许	否	入站	Syst...	任何
核心网络 - 多播侦听程序完成(ICMPv6-In)	所有	允许	否	入站	Syst...	任何
核心网络 - 邻居发现广播(ICMPv6-In)	所有	允许	否	入站	Syst...	任何
核心网络 - 邻居发现请求(ICMPv6-In)	所有	允许	否	入站	Syst...	任何
核心网络 - 路由器广播(ICMPv6-In)	所有	允许	否	入站	Syst...	任何
核心网络 - 路由器请求(ICMPv6-In)	所有	允许	否	入站	Syst...	任何
核心网络 - 目标不可访问(ICMPv6-In)	所有	允许	否	入站	Syst...	任何
核心网络 - 数据包太大(ICMPv6-In)	所有	允许	否	入站	任何	任何
核心网络 - 需要目标不可访问的碎片(ICMPv6-In)	所有	允许	否	入站	Syst...	任何
远程桌面 - 用户模式(TCP-In)	所有	允许	否	入站	C:\W...	任何
远程桌面 - 用户模式(UDP-In)	所有	允许	否	入站	C:\W...	任何
远程桌面 - 远程监控(TCP-In)	所有	允许	否	入站	C:\W...	任何

## 操作

防火墙

查看

刷新

导出列表...

帮助

放通PING--入站

属性

帮助

```
C:\Users\Administrator>ping 172.17.0.16
```

正在 Ping 172.17.0.16 具有 32 字节的数据:

来自 172.17.0.16 的回复: 字节=32 时间<1ms TTL=128  
来自 172.17.0.16 的回复: 字节=32 时间<1ms TTL=128  
来自 172.17.0.16 的回复: 字节=32 时间<1ms TTL=128  
来自 172.17.0.16 的回复: 字节=32 时间<1ms TTL=128

```
C:\Users\Administrator>ping 172.17.0.9
```

正在 Ping 172.17.0.9 具有 32 字节的数据:

来自 172.17.0.9 的回复: 字节=32 时间<1ms TTL=128  
来自 172.17.0.9 的回复: 字节=32 时间<1ms TTL=128

高级安全 Windows 防火墙

文件(F) 操作(A) 查看(V) 帮助(H)

本地计算机 上的高级安全 Win

防火墙

名称	配置文件	操作	替代	方向	程序	本地地
Cortana (小娜)	所有	允许	否	入站	任何	任何
Windows 远程管理(HTTP-In)	公用	允许	否	入站	Syst...	任何
放通PING--入站	所有	允许	否	入站	任何	任何
核心网络 - Internet 组管理协议(IGMP-In)	所有	允许	否	入站	Syst...	任何
核心网络 - IPv6 (IPv6-In)	所有	允许	否	入站	Syst...	任何
核心网络 - IPv6 的动态主机配置协议(D...	所有	允许	否	入站	C:\W...	任何
核心网络 - 参数问题(ICMPv6-In)	所有	允许	否	入站	Syst...	任何
核心网络 - 超时(ICMPv6-In)	所有	允许	否	入站	Syst...	任何
核心网络 - 动态主机配置协议(DHCP-In)	所有	允许	否	入站	C:\W...	任何
核心网络 - 多播侦听程序报告 v2 (ICMP...	所有	允许	否	入站	Syst...	任何
核心网络 - 多播侦听程序报告(ICMPv6-In)	所有	允许	否	入站	Syst...	任何
核心网络 - 多播侦听程序查询(ICMPv6-In)	所有	允许	否	入站	Syst...	任何
核心网络 - 多播侦听程序完成(ICMPv6-In)	所有	允许	否	入站	Syst...	任何
核心网络 - 邻居发现播发(ICMPv6-In)	所有	允许	否	入站	Syst...	任何
核心网络 - 邻居发现请求(ICMPv6-In)	所有	允许	否	入站	Syst...	任何
核心网络 - 路由器播发(ICMPv6-In)	所有	允许	否	入站	Syst...	任何
核心网络 - 路由器请求(ICMPv6-In)	所有	允许	否	入站	Syst...	任何
核心网络 - 目标不可访问(ICMPv6-In)	所有	允许	否	入站	Syst...	任何
核心网络 - 数据包太大(ICMPv6-In)	所有	允许	否	入站	任何	任何
核心网络 - 需要目标不可访问的碎片(IC...	所有	允许	否	入站	Syst...	任何
远程桌面 - 用户模式(TCP-In)	所有	允许	否	入站	C:\W...	任何
远程桌面 - 用户模式(UDP-In)	所有	允许	否	入站	C:\W...	任何
远程桌面 - 远程监控(TCP-In)	所有	允许	否	入站	C:\W...	任何
阻止PING规则--出站	所有	阻止	否	出站	任何	任何

操作

防火墙

查看

刷新

导出列表...

帮助

放通PING--入站

属性

帮助

```
C:\Users\Administrator>ping 172.17.0.16
正在 Ping 172.17.0.16 具有 32 字节的数据:
一般故障。
一般故障。
一般故障。
一般故障。
一般故障。
172.17.0.16 的 Ping 统计信息:
    数据包: 已发送 = 4, 已接收 = 0, 丢失 = 4 (100% 丢失),
```

```
C:\Users\Administrator>ping 172.17.0.9
正在 Ping 172.17.0.9 具有 32 字节的数据:
来自 172.17.0.9 的回复: 字节=32 时间<1ms TTL=128
172.17.0.9 的 Ping 统计信息:
    数据包: 已发送 = 4, 已接收 = 4, 丢失 = 0 (0% 丢失),
往返行程的估计时间(以毫秒为单位):
    最短 = 0ms, 最长 = 0ms, 平均 = 0ms
```

请输入用户名和密码: *yiqiSun,123456*  
该IP不允许登录!

允许登陆的账号密码 {'yiqiSun': '123456', 'ruyanWang': '123456', 'liyiJun': '123456'}  
禁止的IP: ['192.168.80.1']  
ip: ('192.168.80.1', 61132)  
*yiqiSun,123456*

# CentOS 8下 基于C语言 使用libpcap 对IPv4抓包实现

---

---

自用腾讯云服务器

CentOS Linux release 8.0.1905 (Core)

libpcap-1.9.1

# 操作环境

---

安装gcc

安装flex、bison

*yum -y install flex*

*yum -y install bison*

wget下载libpcap

*wget -c http://www.tcpdump.org/release/libpcap-1.9.1.tar.gz*

tar解压并安装

*tar libpcap-1.9.1.tar.gz*

*cd libpcap-1.9.1*

*./configure*

*make*

*make install*

# 准备工作

---

Setting the device

Opening the device for sniffing

BPF Filtering

Save .pcap for Wireshark

代码演示

---

# CentOS 8下 防火墙FirewallD操作实践 及netfilter简介

---

---

自用腾讯云服务器

CentOS Linux release 8.0.1905 (Core)

FirewallD 0.6.3

# 操作环境

---

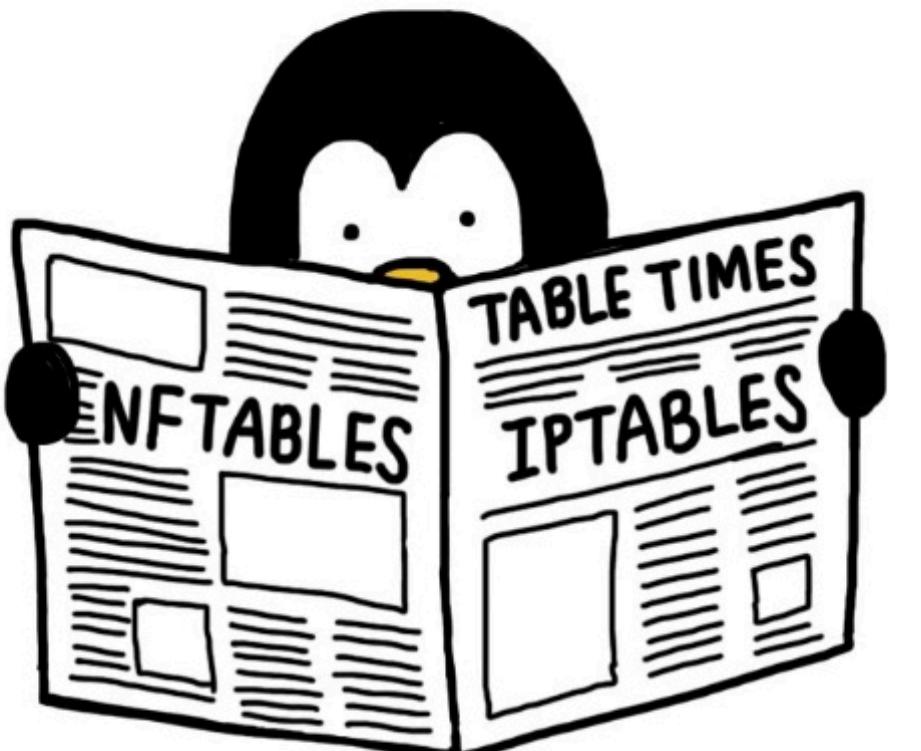
firewalld不基于iptables而是nftables

了解FirewallD命令使用方法

查看firewalld状态

开启或关闭firewalld

查看端口情况



ipv6onlyhosting.com

## 准备工作

---

```
[root@VM-0-15-centos /]# firewall-cmd -h
```

```
[root@VM-0-15-centos /]# firewall-cmd --help
```

# 了解FirewallD命令使用方法

---

准备工作

了解FirewallD命令使用方法

查看firewalld状态

开启或关闭firewalld

查看端口情况

```
[root@VM-0-15-centos /]# systemctl status firewalld
```

# 查看firewalld状态

---

准备工作

了解FirewallD命令使用方法

查看firewalld状态

开启或关闭firewalld

查看端口情况

```
[root@VM-0-15-centos /]# systemctl start firewalld.service
```

```
[root@VM-0-15-centos /]# systemctl stop firewalld.service
```

# 开启或关闭firewalld

---

准备工作

了解FirewallD命令使用方法

查看firewalld状态

开启或关闭firewalld

查看端口情况

```
[root@VM-0-15-centos /]# netstat -tunlp
```

## 查看端口情况

---

准备工作

了解FirewallD命令使用方法

查看firewalld状态

开启或关闭firewalld

查看端口情况

开放单个端口

对特定ip地址开放特定端口

开放https服务

## 防火墙配置

---

关闭防火墙的情况下，即status为inactive时，可以正常访问由nginx配置的443端口，即https访问，如下图。



## 开放单个端口

### 防火墙配置

```
reneelin — root@VM-0-15-centos:~ — ssh root@122.51.65.184 — 110x16
[root@VM-0-15-centos ~]# systemctl stop firewalld.service
[root@VM-0-15-centos ~]# systemctl status firewalld
● firewalld.service - firewalld - dynamic firewall daemon
  Loaded: loaded (/usr/lib/systemd/system/firewalld.service; disabled; vendor preset: enabled)
  Active: inactive (dead)
    Docs: man:firewalld(1)

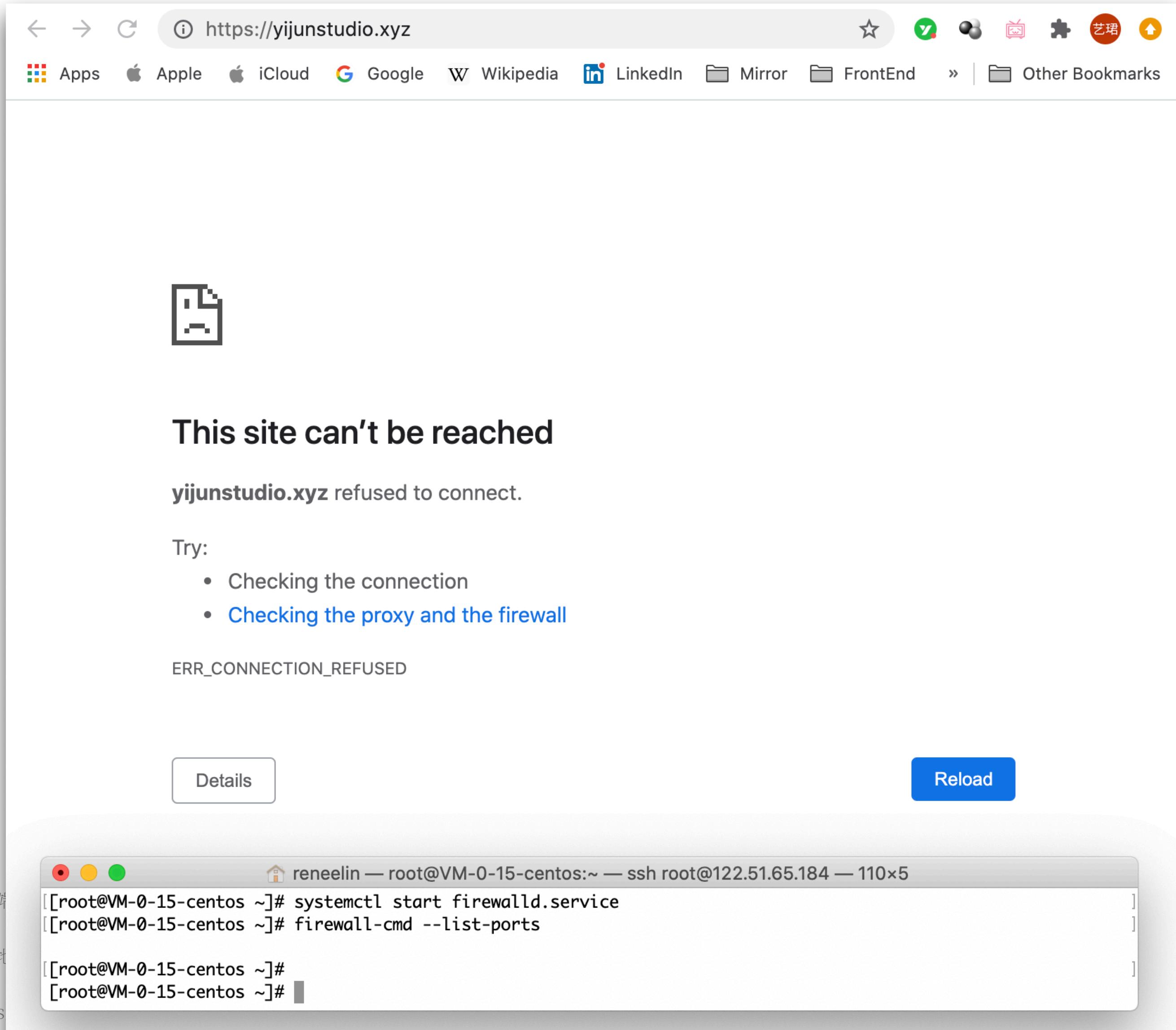
Oct 13 15:16:39 VM-0-15-centos systemd[1]: Starting firewalld - dynamic firewall daemon...
Oct 13 15:16:39 VM-0-15-centos systemd[1]: Started firewalld - dynamic firewall daemon.
Oct 13 15:24:09 VM-0-15-centos systemd[1]: Stopping firewalld - dynamic firewall daemon...
Oct 13 15:24:10 VM-0-15-centos systemd[1]: Stopped firewalld - dynamic firewall daemon.
```

开放单个端口

对特定ip地

开放https

打开防火墙， 默认没有端口可以被访问， 因此进行https访问时被拒绝， 如下图。



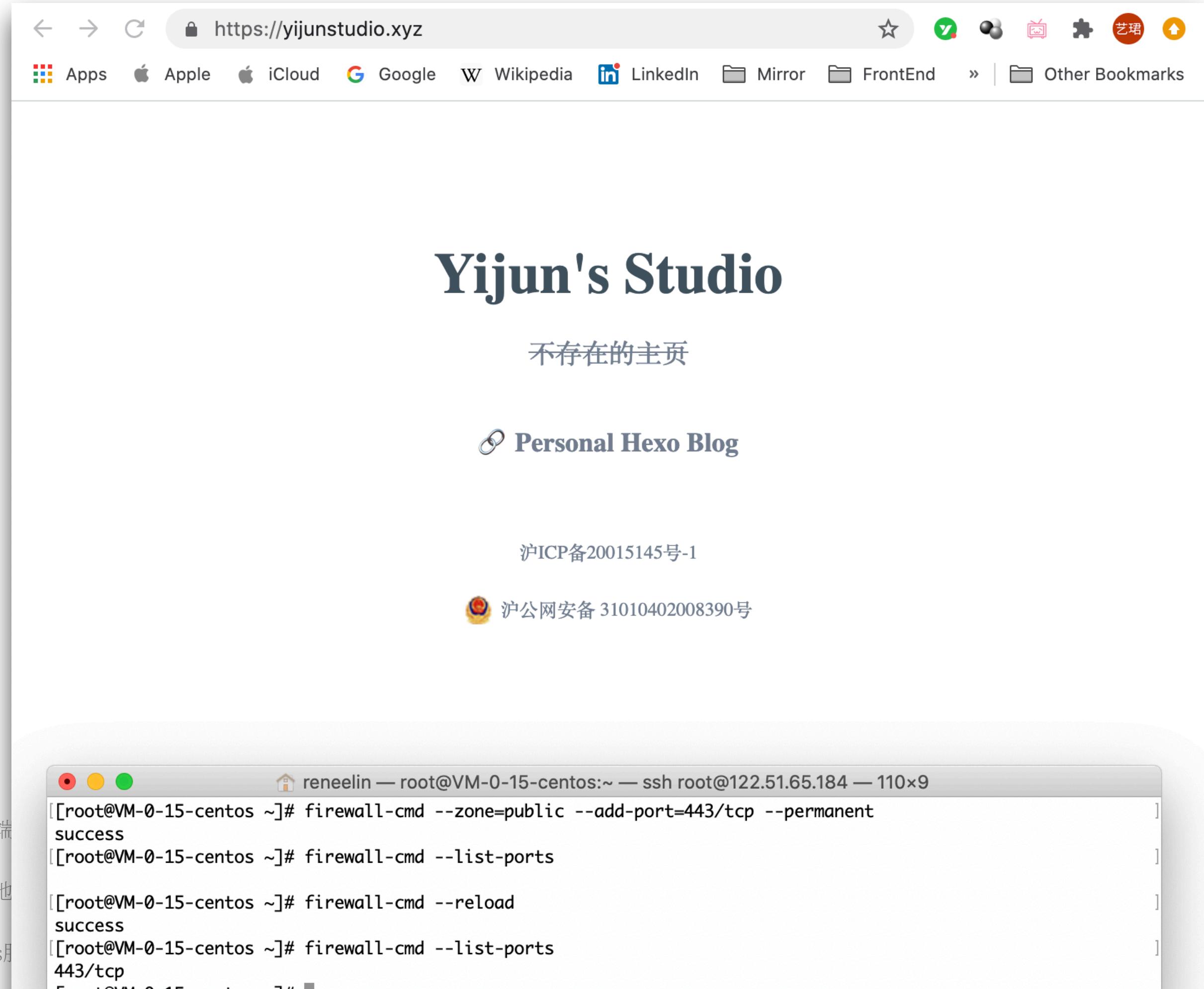
# 开放单个端口

## 防火墙配置

添加443端口的永久tcp连接。此处永久指重启不失效。

使用`--list-ports`查看现在的端口，依然没有443/tcp，网页也依然无法访问。因为需要`reload`配置，激活新添加的端口。

`reload`后，重新查看端口，可以看到已经添加。网页也可以被访问。



# 开放单个端口

防火墙配置

开放单个端口  
对特定ip地  
开放https用

首先删除之前开放的443端口，网页无法访问。

# 对特定ip地址开放特定端口

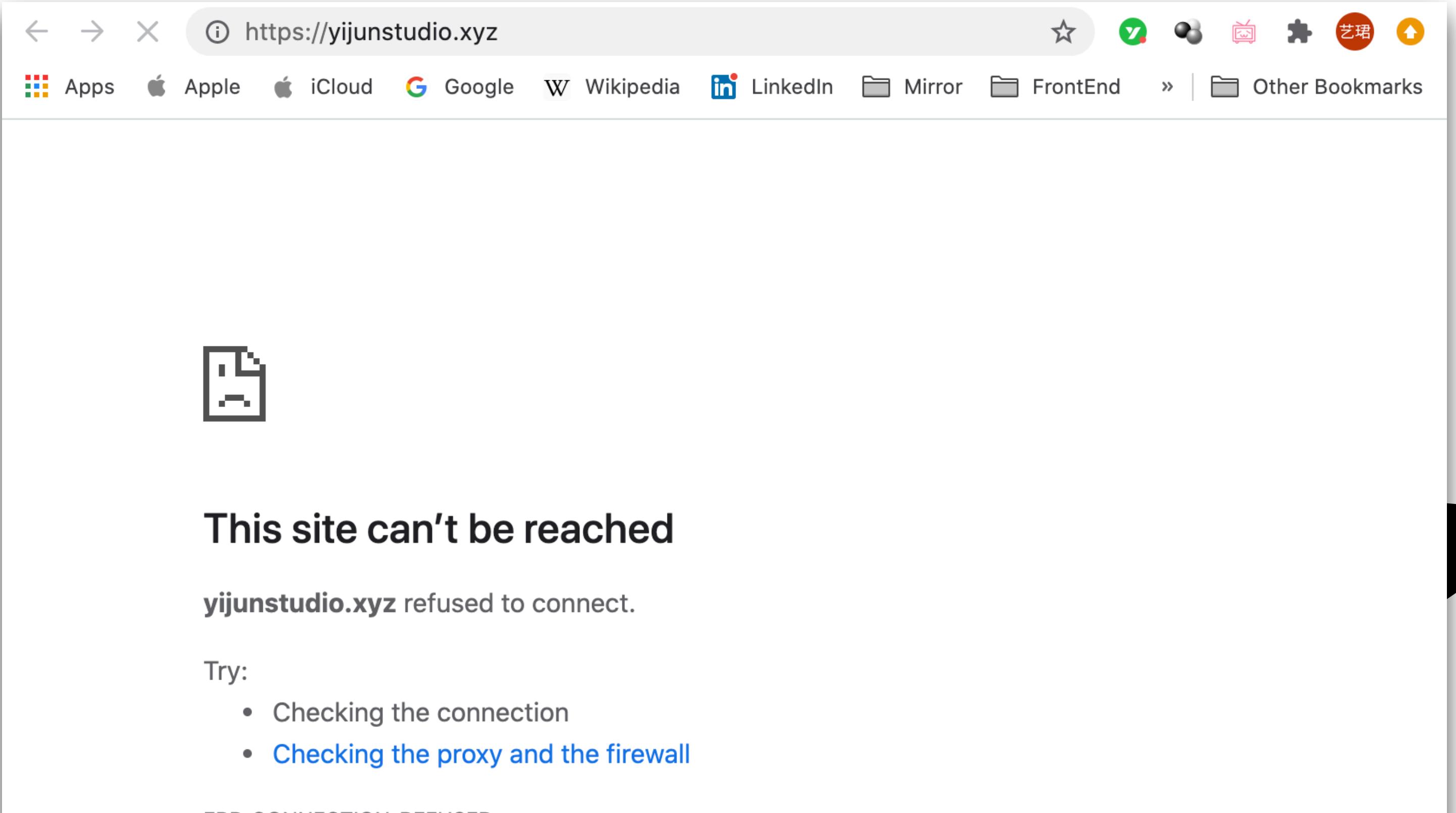
防火墙配置

开放单个端口

对特定ip地址开放特定端口

开放https服务

首先删除之前开放的443端口，网页无法访问。



# 址开放特定端口

## 防火墙配置

```
[root@VM-0-15-centos ~]# firewall-cmd --list-ports
443/tcp
[root@VM-0-15-centos ~]# firewall-cmd --zone=public --remove-port=443/tcp --permanent
success
[root@VM-0-15-centos ~]# firewall-cmd --reload
success
[root@VM-0-15-centos ~]# firewall-cmd --list-ports
[root@VM-0-15-centos ~]#
```

开放单个端口  
对特定ip地址  
开放https服务

对10.95.69.248开放443端口。

# 对特定ip地址开放特定端口

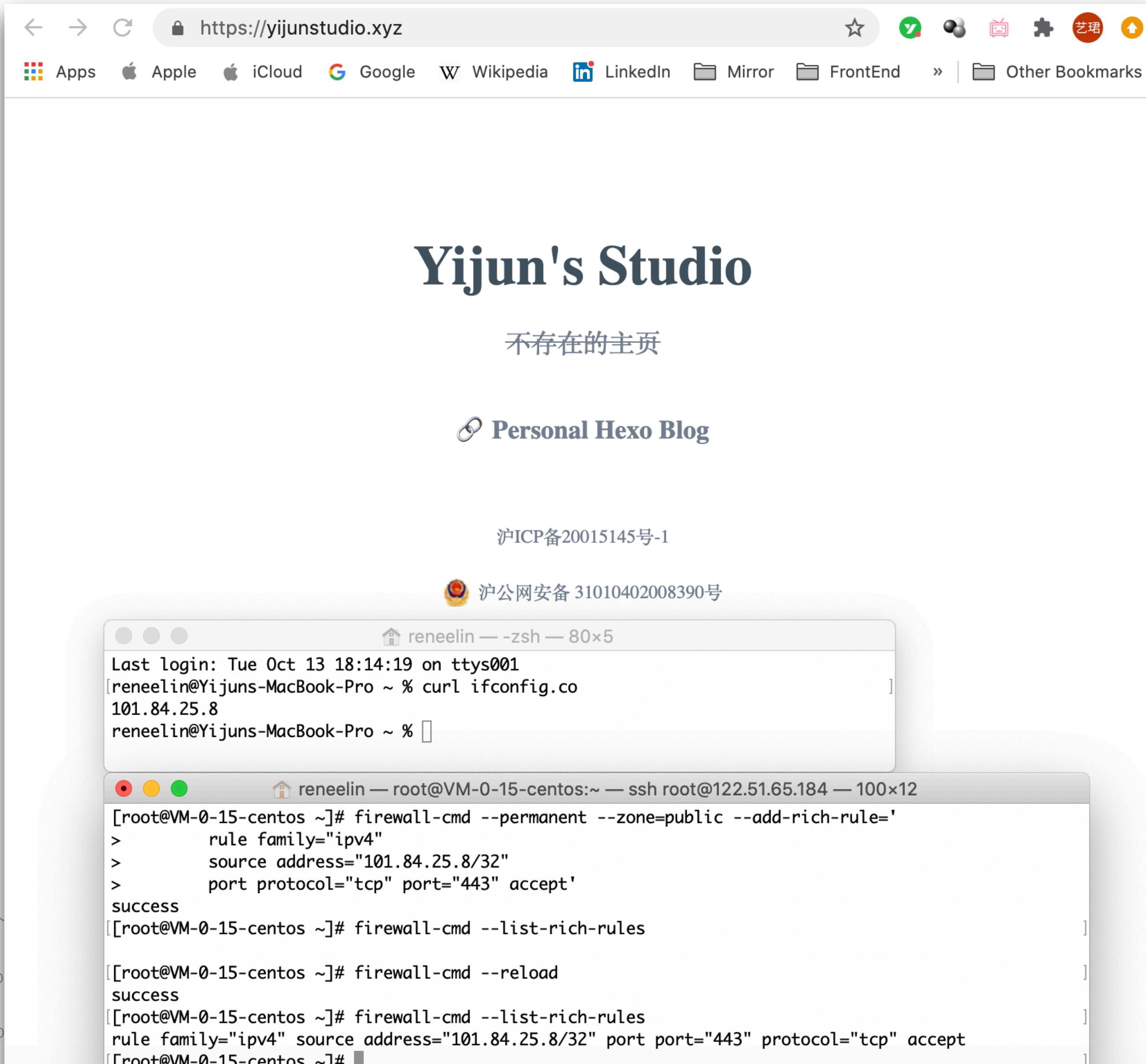
防火墙配置

开放单个端口

对特定ip地址开放特定端口

开放https服务

对10.95.69.248开放443端口。



# 地址开放特定端口

防火墙配置

再拒绝10.95.69.248对于443端口的请求。

# 对特定ip地址开放特定端口

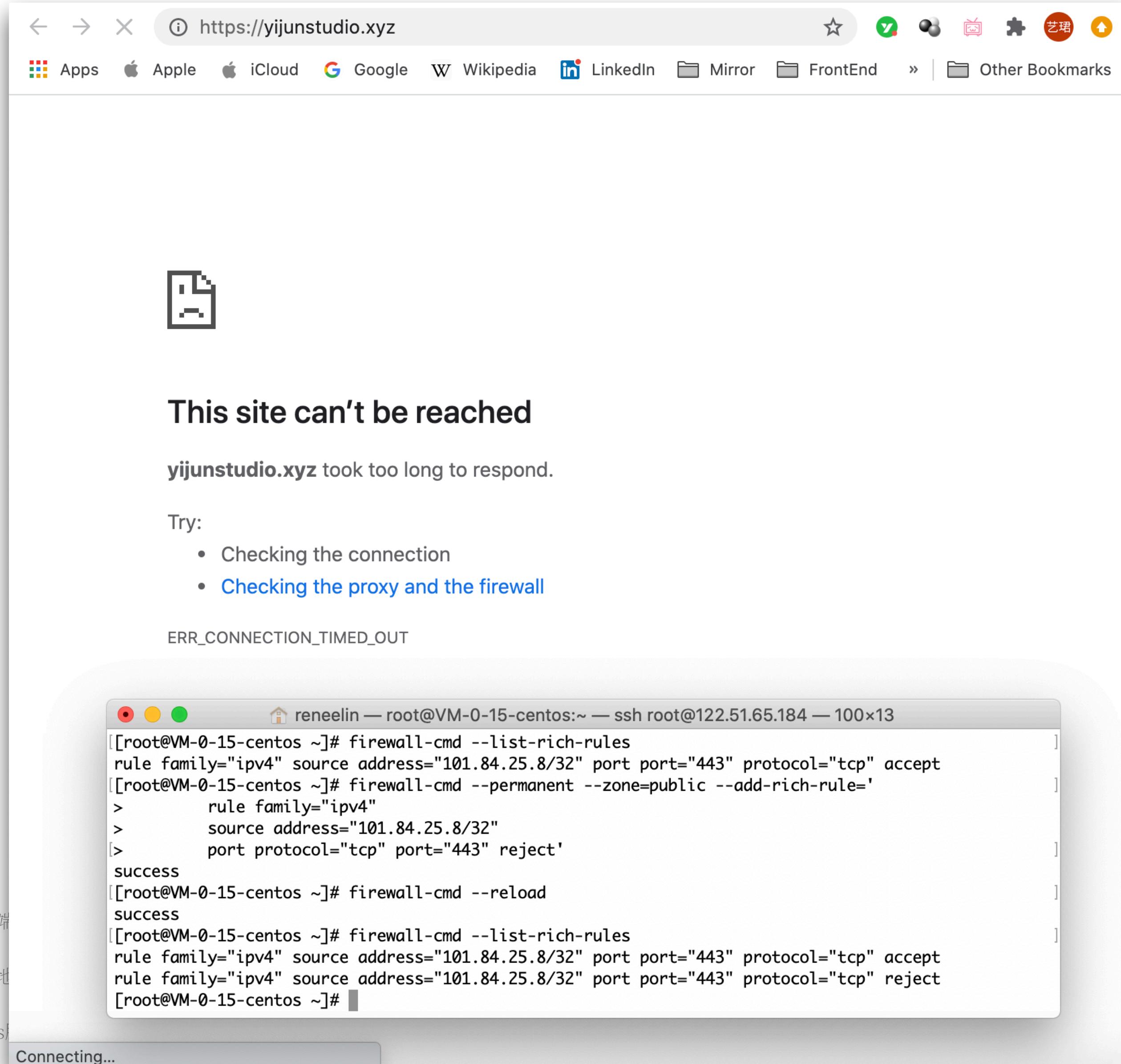
防火墙配置

开放单个端口

对特定ip地址开放特定端口

开放https服务

再拒绝10.95.69.248对于443端口的请求。



# 地址开放特定端口

## 防火墙配置

```
reneelin — root@VM-0-15-centos:~ — ssh root@122.51.65.184 — 100x13
[[root@VM-0-15-centos ~]# firewall-cmd --list-rich-rules
rule family="ipv4" source address="101.84.25.8/32" port port="443" protocol="tcp" accept
[[root@VM-0-15-centos ~]# firewall-cmd --permanent --zone=public --add-rich-rule='
>     rule family="ipv4"
>     source address="101.84.25.8/32"
>     port protocol="tcp" port="443" reject'
success
[[root@VM-0-15-centos ~]# firewall-cmd --reload
success
[[root@VM-0-15-centos ~]# firewall-cmd --list-rich-rules
rule family="ipv4" source address="101.84.25.8/32" port port="443" protocol="tcp" accept
rule family="ipv4" source address="101.84.25.8/32" port port="443" protocol="tcp" reject
[[root@VM-0-15-centos ~]# ]
```

删除reject规则，可正常访问。

# 对特定ip地址开放特定端口

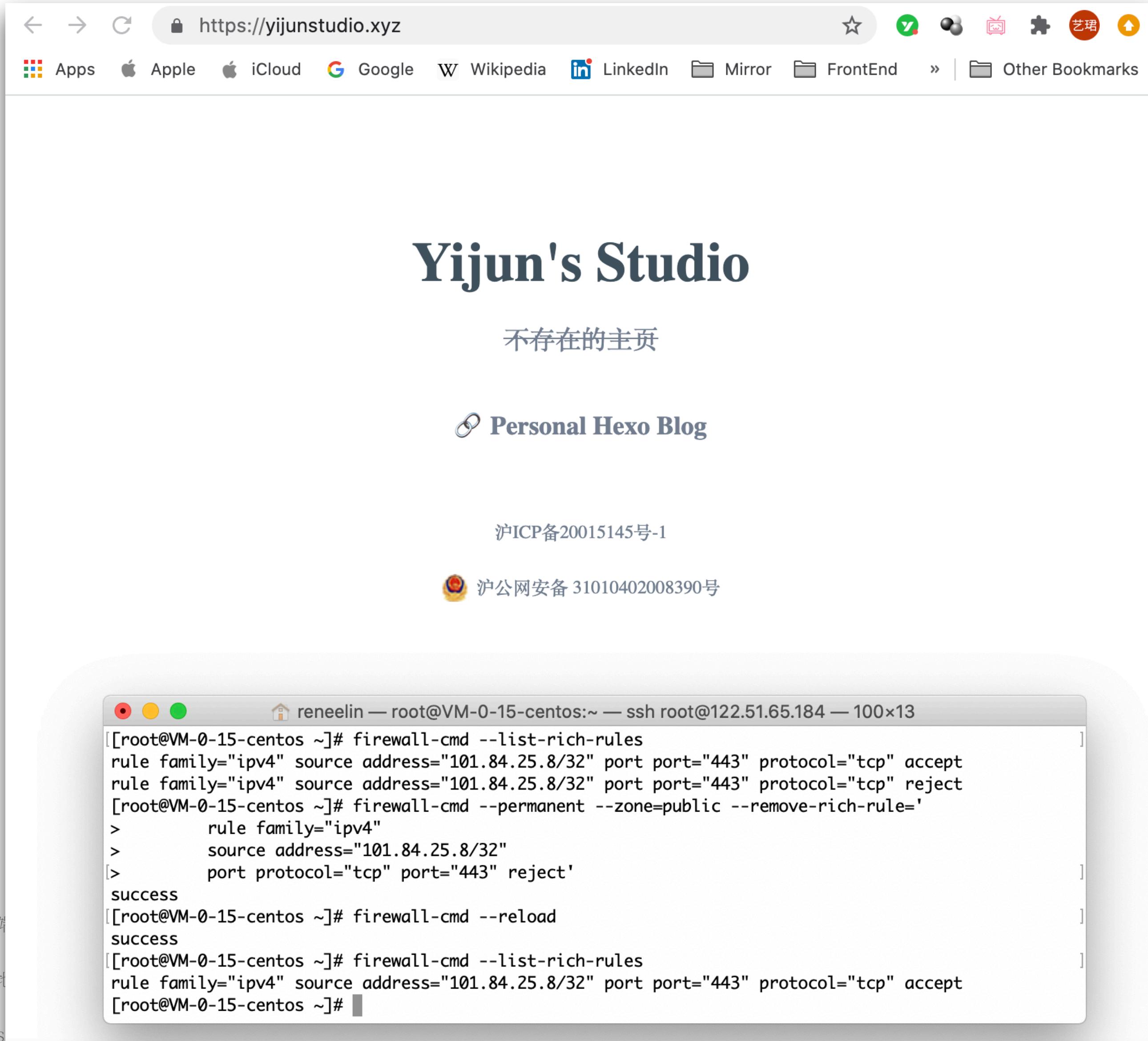
防火墙配置

开放单个端口

对特定ip地址开放特定端口

开放https服务

删除reject规则，可正常访问。

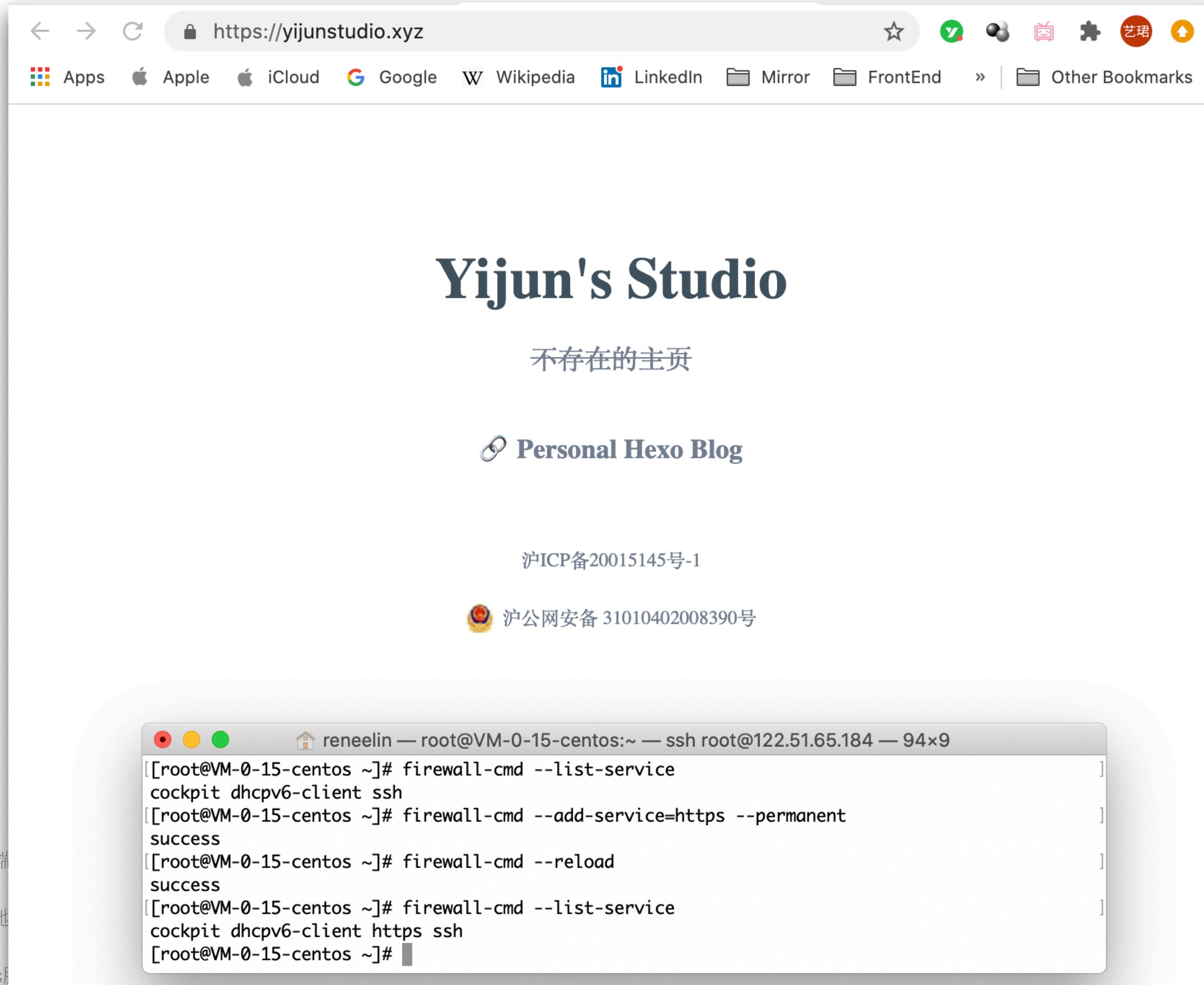


# 地址开放特定端口

## 防火墙配置

查看所有支持的服务。-get-service

显示现在已经启动的服务。随后添加https服务， reload后重新查看已启动的服务，网页可以访问。



# 开放https服务

防火墙配置

开放单个端口  
对特定ip地址  
开放https服务

netfilter.org is home to the software of the packet filtering framework inside the Linux 2.4.x and later kernel series. Software commonly associated with netfilter.org is iptables.

stateless packet filtering (IPv4 and IPv6)

stateful packet filtering (IPv4 and IPv6)

all kinds of network address and port translation, e.g. NAT/NAPT (IPv4 and IPv6)

flexible and extensible infrastructure

multiple layers of API's for 3rd party extensions

# netfilter

---

build internet firewalls based on stateless and stateful packet filtering

deploy highly available stateless and stateful firewall clusters

use NAT and masquerading for sharing internet access if you don't have enough public IP addresses

use NAT to implement transparent proxies

aid the tc and iproute2 systems used to build sophisticated QoS and policy routers

do further packet manipulation (mangling) like altering the TOS/DSCP/ECN bits of the IP header

netfilter.org is home to the software of the packet filtering framework inside the Linux 2.4.x and later kernel

The screenshot shows the netfilter.org website with a blue header bar containing links for www, ftp, git, lists, bugzilla, people, and patchwork. Below the header is the netfilter logo with the tagline "firewalling, NAT, and packet mangling for linux". The main content area features a large graphic of a network mesh. A dark blue sidebar on the left contains sections for About (Coreteam, Contributors, History, License, Thanks, PGP key) and Projects (iptables, nftables, libnftnl, libnfnetlink). The main content area has a dark blue header with the text "The netfilter.org project" and a sub-section "What is netfilter.org?". The text explains that netfilter.org is the home of the packet filtering framework in Linux kernels, mentioning iptables and its history as a successor to ipchains and ipf. It also describes the framework's capabilities like packet filtering, NAT, and mangling.

[www](#) | [ftp](#) | [git](#) | [lists](#) | [bugzilla](#) | [people](#) | [patchwork](#)

**netfilter**  
firewalling, NAT, and packet mangling for linux

**About**

[Coreteam](#)  
[Contributors](#)  
[History](#)  
[License](#)  
[Thanks](#)  
[PGP key](#)

**Projects**

[iptables](#)  
[nftables](#)  
[libnftnl](#)  
[libnfnetlink](#)

**The netfilter.org project**

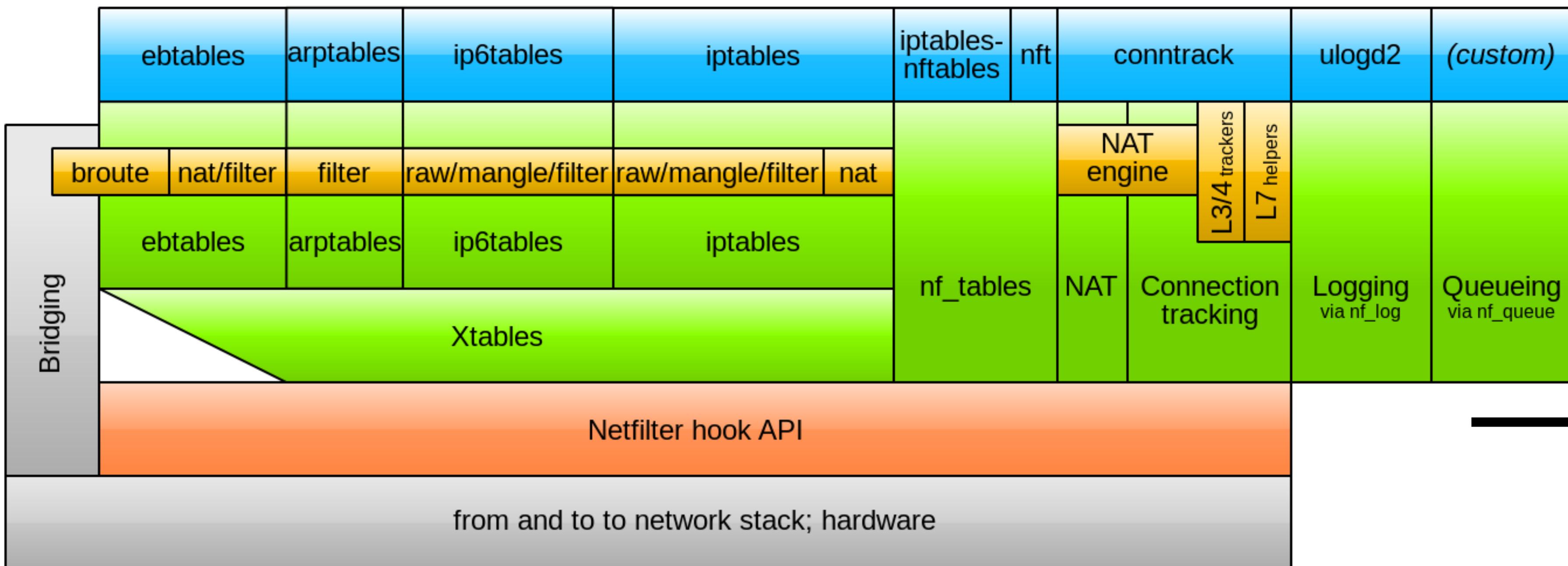
## What is netfilter.org?

netfilter.org is home to the software of the packet filtering framework inside the [Linux](#) 2.4.x and later kernel series. Software commonly associated with netfilter.org is [iptables](#).

Software inside this framework enables packet filtering, network address [and port] translation (NA[P]T) and other packet mangling. It is the re-designed and heavily improved successor of the previous Linux 2.2.x [ipchains](#) and Linux 2.0.x [ipf](#). It can do further packet manipulation (mangling) like altering the TOS/DSCP/ECN bits of the IP header

# **Netfilter components**

Jan Engelhardt, last updated 2014-02-28 (initial: 2008-06-17)



- █ Userspace tools
- Netfilter kernel components
- other networking components

# netfilter

Windows防火墙配置 - <https://cloud.tencent.com/edu/learning/course-2186-41300>

WinPcap源码全解 - <http://www.ferrisxu.com/WinPcap/html/modules.html>

Linux下Libpcap源码分析和包过滤机制 - CSDN

Programming with pcap - <http://www.tcpdump.org/pcap.html>

Netfilter - <https://www.netfilter.org/>

Netfilter Tutorial - <http://pages.cs.wisc.edu/~chalpin/project/netfilter.html>

Project: Firewall Based on Netfilter - <https://github.com/mryuan0428/Firewall-Based-on-Netfilter>

Compare: Firewalld / Iptables / Nftables / Netfilter - MEDIUM

谢谢

---

Thanks.