**同济大学计算机系**

**信息安全原理PROJ1报告**

****

**学 号 2152701**

**姓 名 陈玟桦**

**专 业 信息安全**

**授课老师 谭成翔**

1. 需求分析

实现一个demo系统，用笔记本电脑在**Windows**或**Linux**系统上**抓取**目标用户（此处为手机，也可以是其他对象）的l2tp请求并进行**拆解**分析，了解其中内容含义，能够**修改**特定l2tp包并将其**重发**给手机

本实验由2152699-陈俊良和2152701-陈玟桦共同完成，具体分工如下：

|  |  |
| --- | --- |
| 陈俊良 | 阿里云服务器的搭建，资料收集，运行环境搭建，代码展示部分之外的报告以及报告整合 |
| 陈玟桦 | 运行环境搭建，代码解析和代码实现，代码展示部分的报告 |

1. 环境搭建

为了实现这一需求，首先要在电脑上搭建能够捕获l2tp包的环境，并利用vpn技术让目标对象（手机）接入这一环境中，本程序在**Linux**系统下实现，环境的搭建采用的是阿里云ECS服务器。本次实验中，我尝试了softether server vpn、以及华为云、腾讯云、阿里云等云主机，以及win10和ubuntu两种不同的运行环境，最终作出了以上选择。

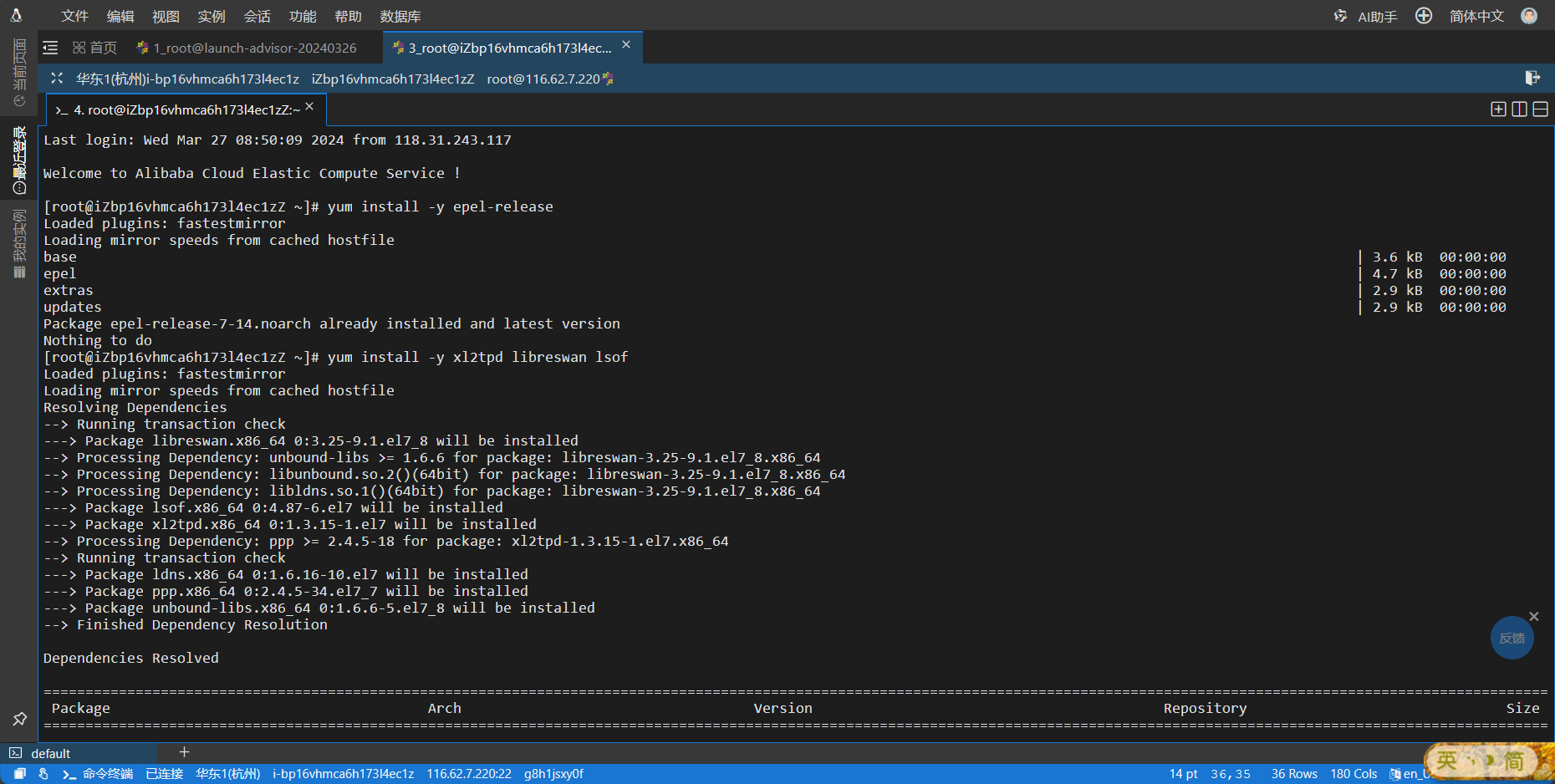
2.1 服务器环境搭建

参考网络教程，进入阿里云官网并选择相应的ECS服务器，并完成l2tp vpn配置。



通过workbench远程连接，执行以下命令：

1. 安装EPEL源：yum install -y epel-release
2. 安装xl2tpd和libreswan：yum install -y xl2tpd libreswan lsof



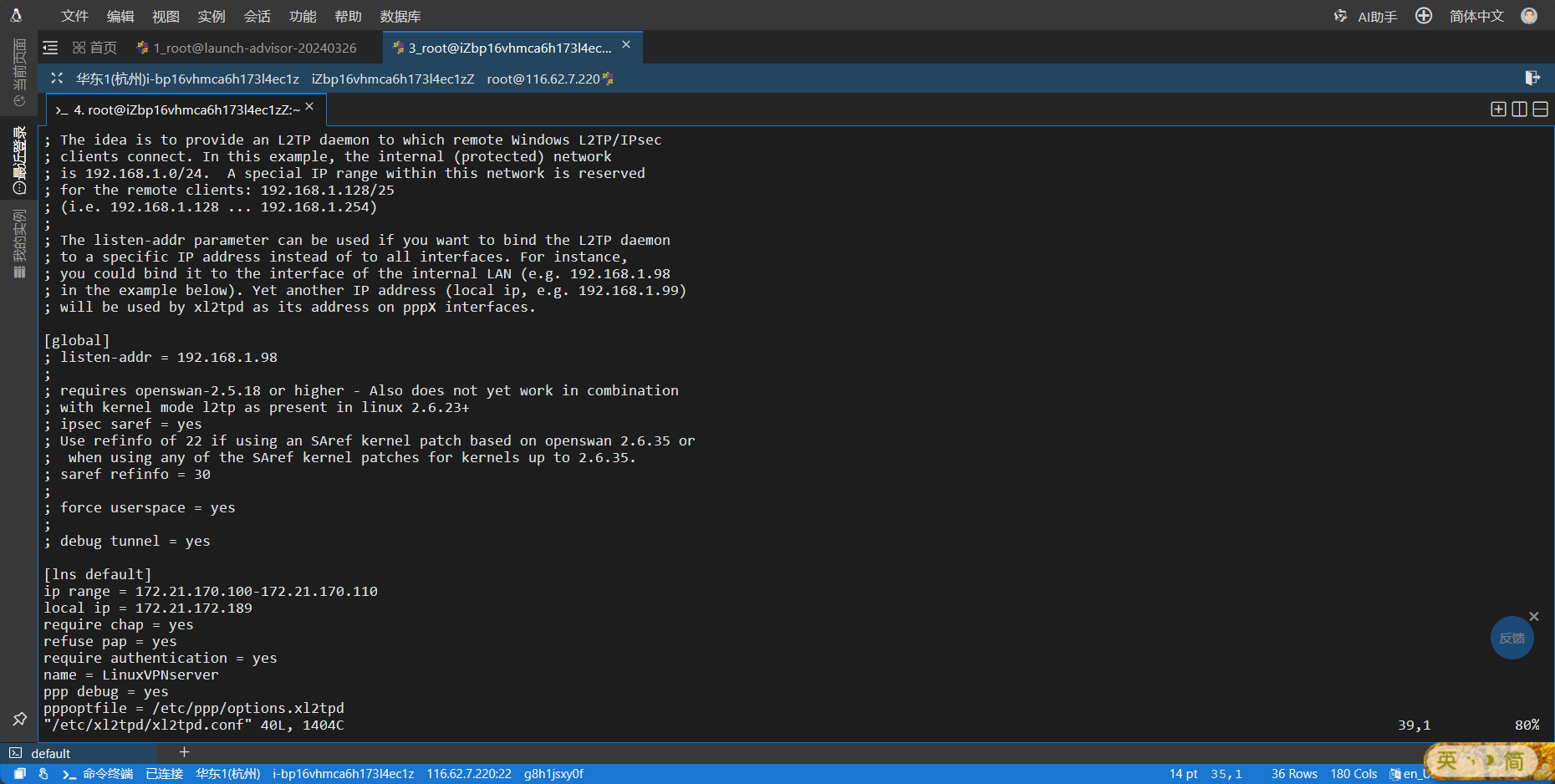
执行

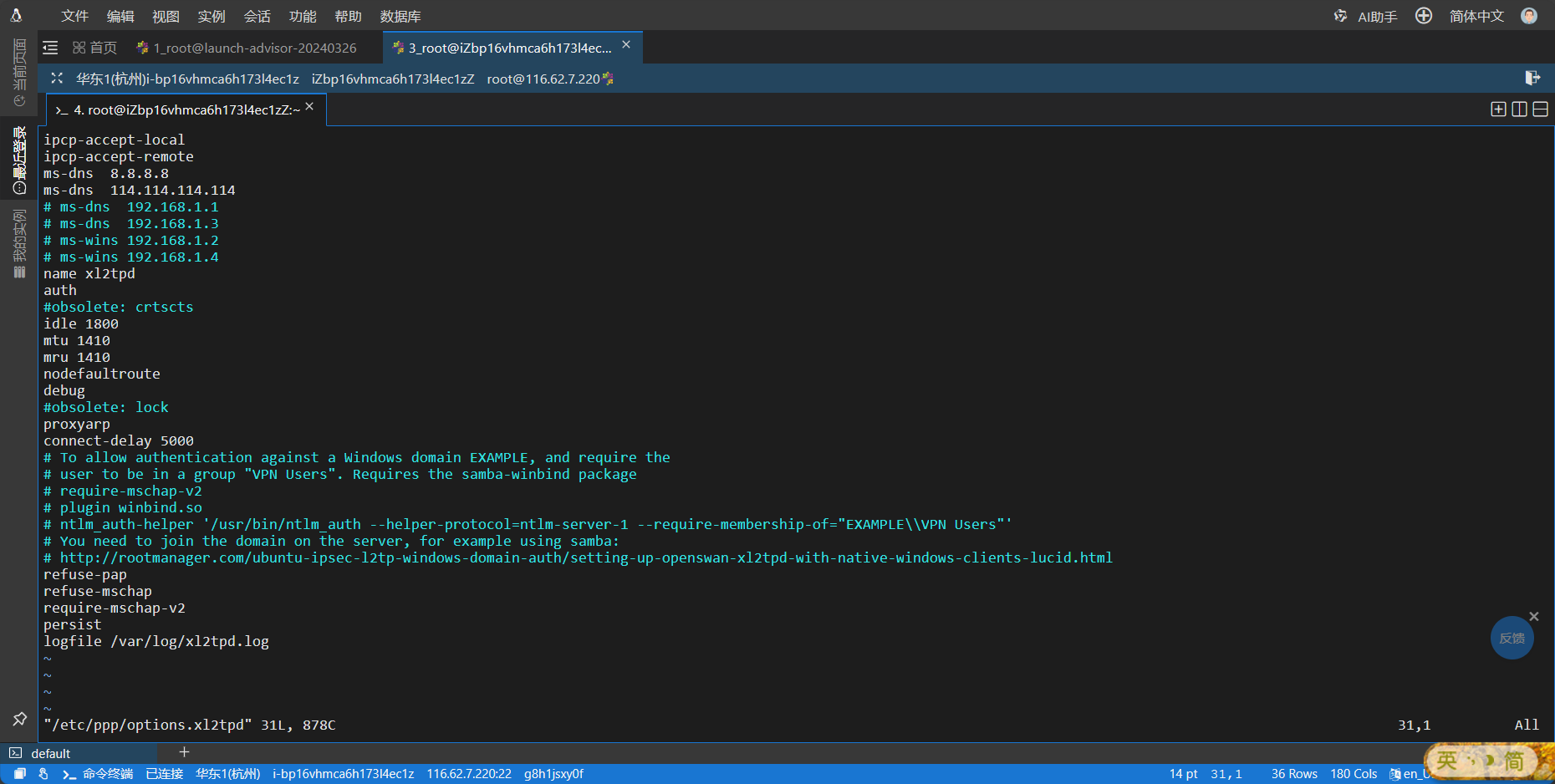
vim /etc/xl2tpd/xl2tpd.conf

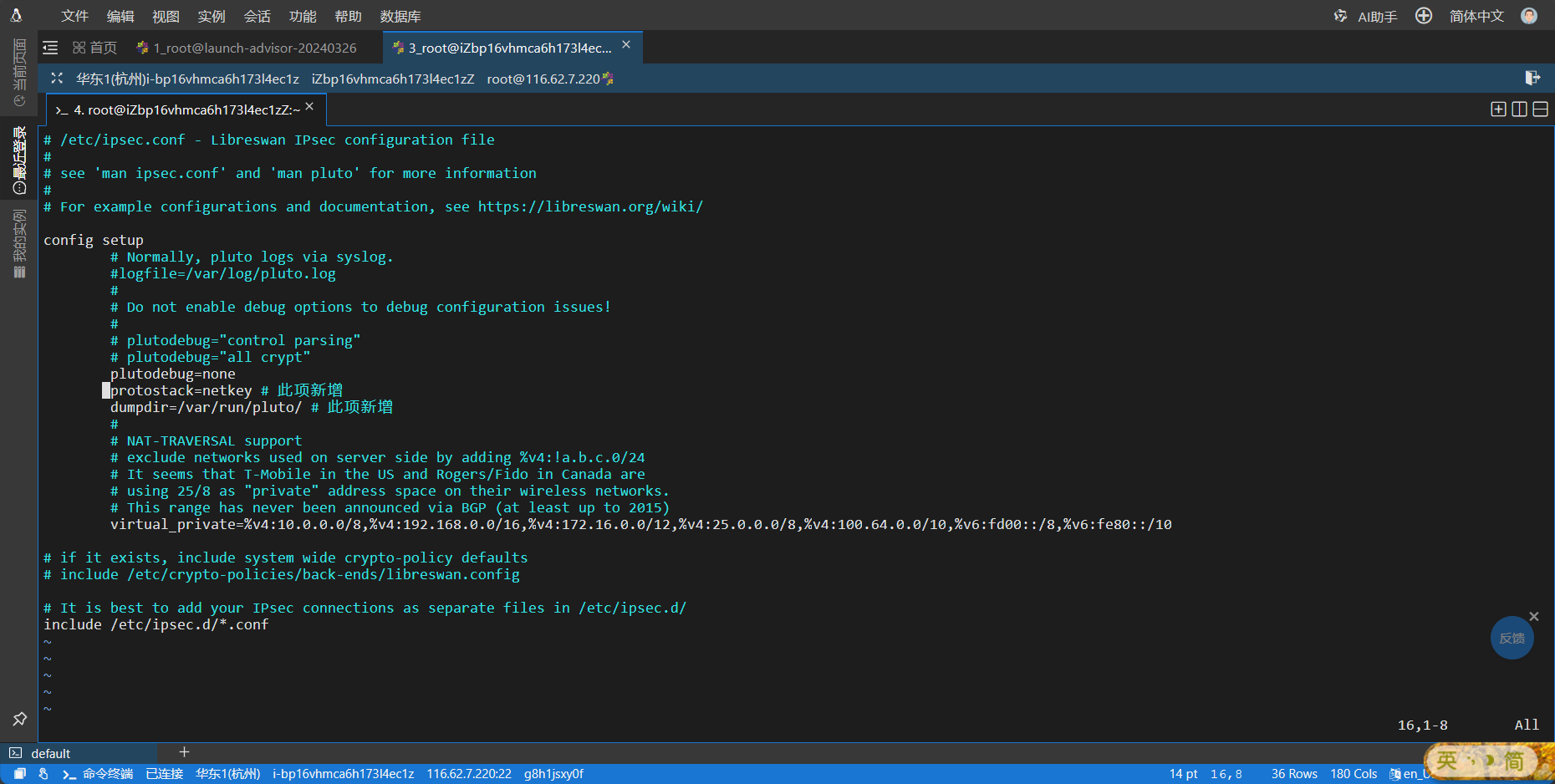
vim /etc/ppp/options.xl2tp

vim /etc/ipsec.conf

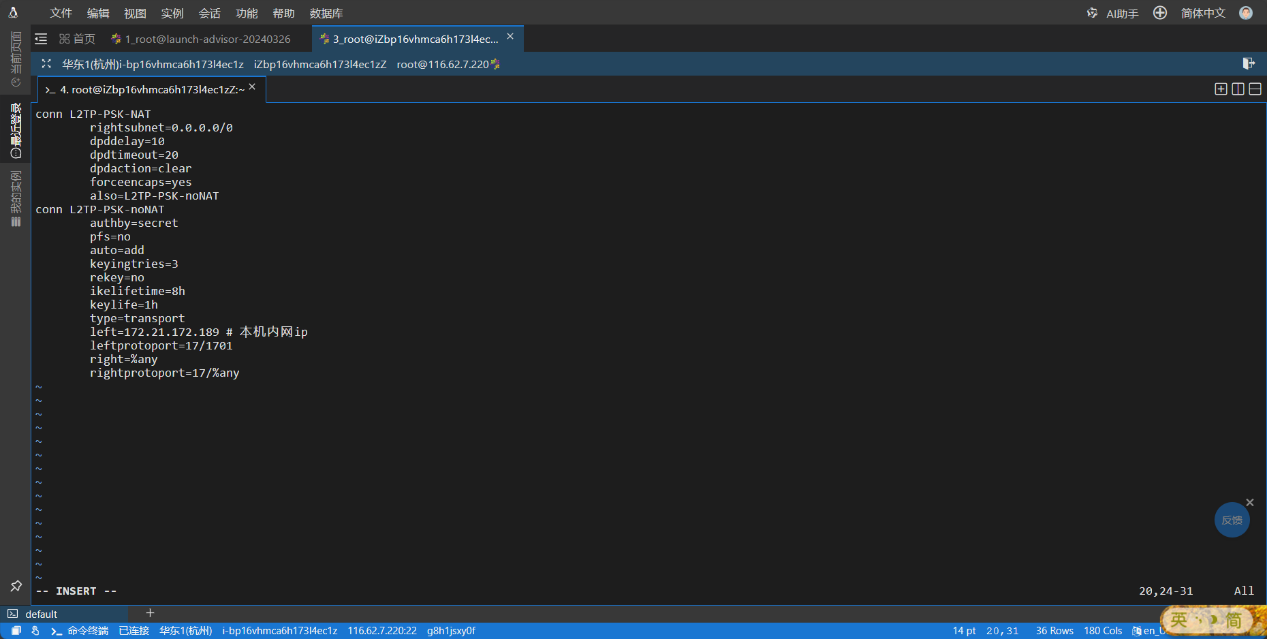
如下更改配置文件：



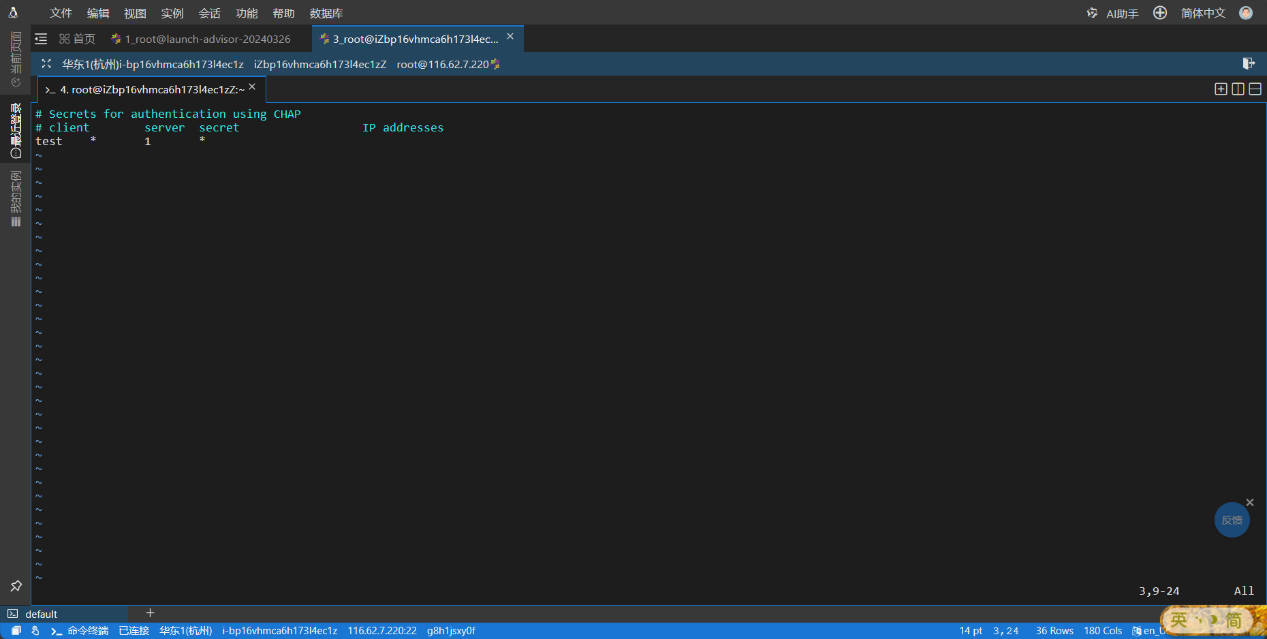




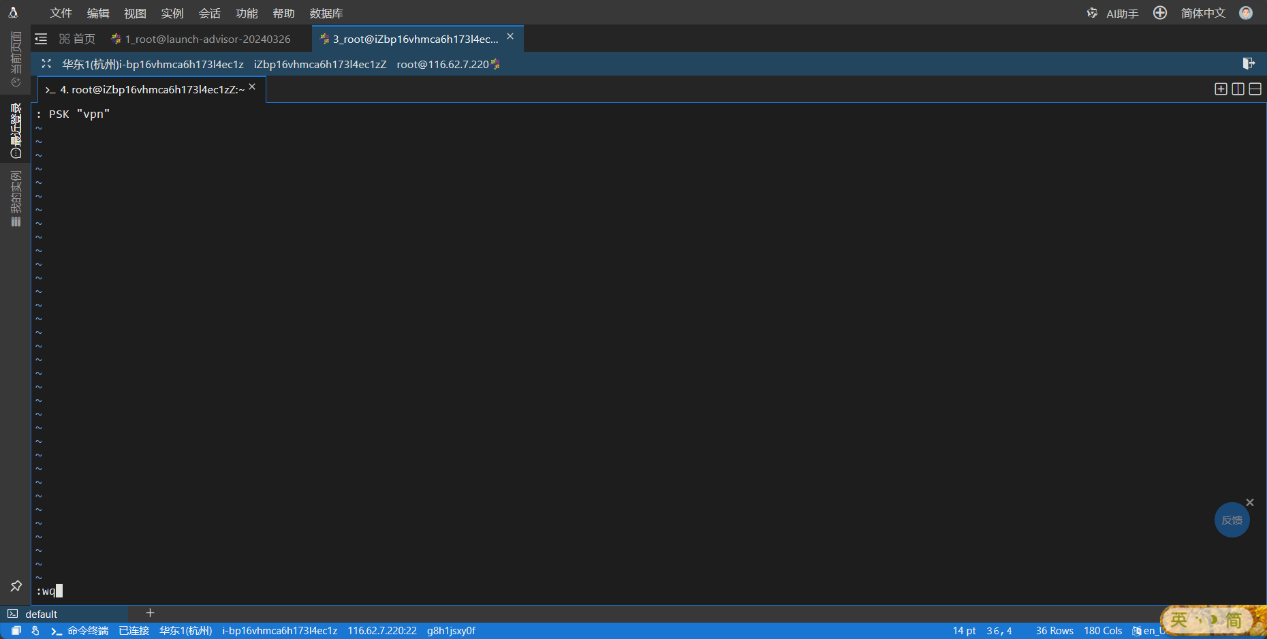
执行vim /etc/ipsec.d/l2tp-ipsec.conf，新建一个文件，并写入相关配置信息：



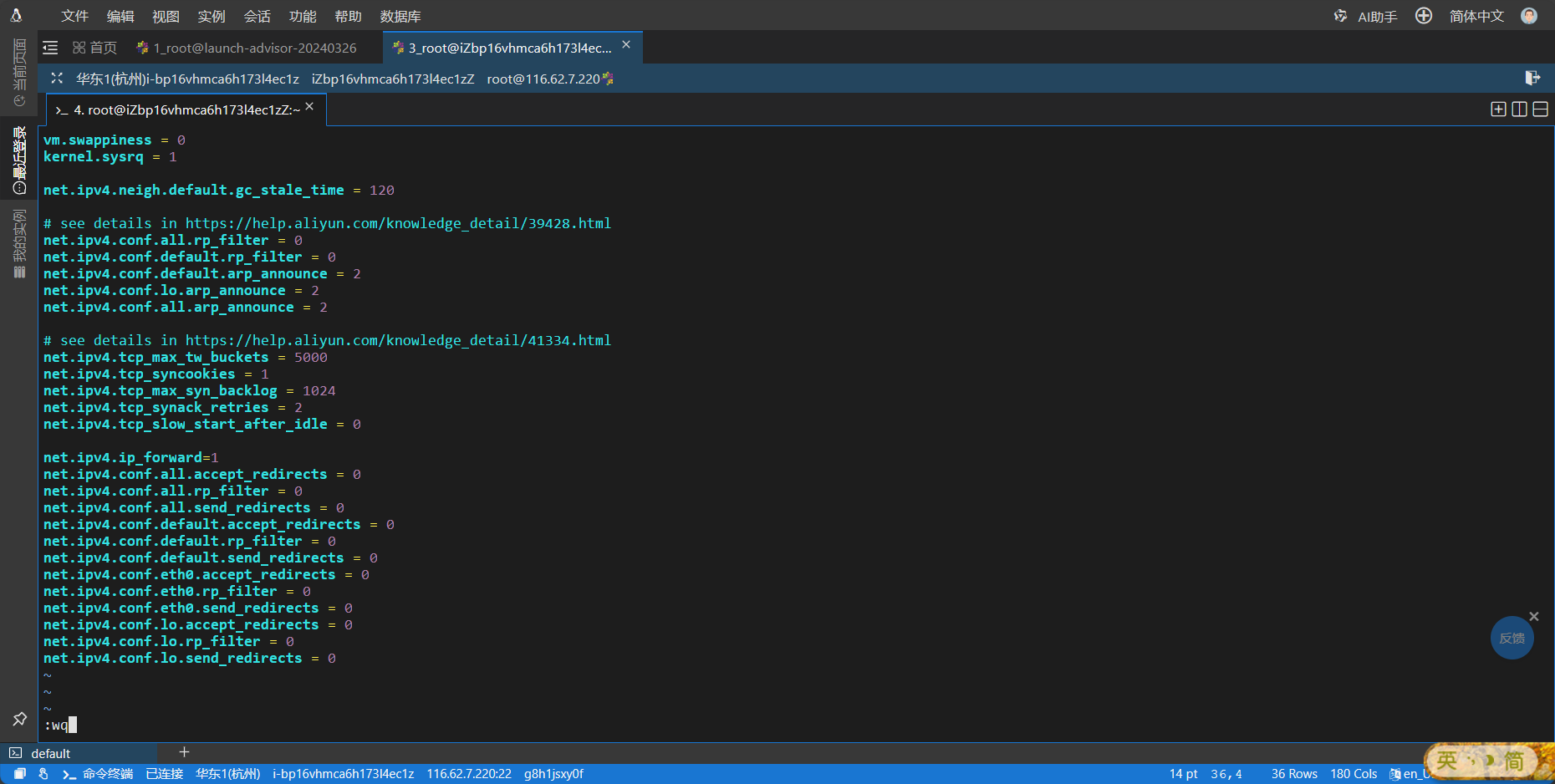
执行vim /etc/ppp/chap-secrets，在新建文件中添加用户名和密码：



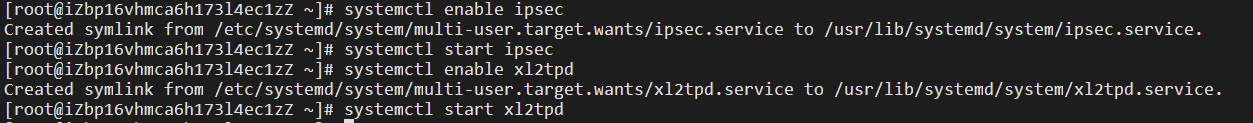
执行vim /etc/ipsec.d/default.secrets，在新建文件中添加预共享密钥：



执行vim /etc/sysctl.conf，追加相应参数配置内容：



最后启动ipsec和xl2tpd：

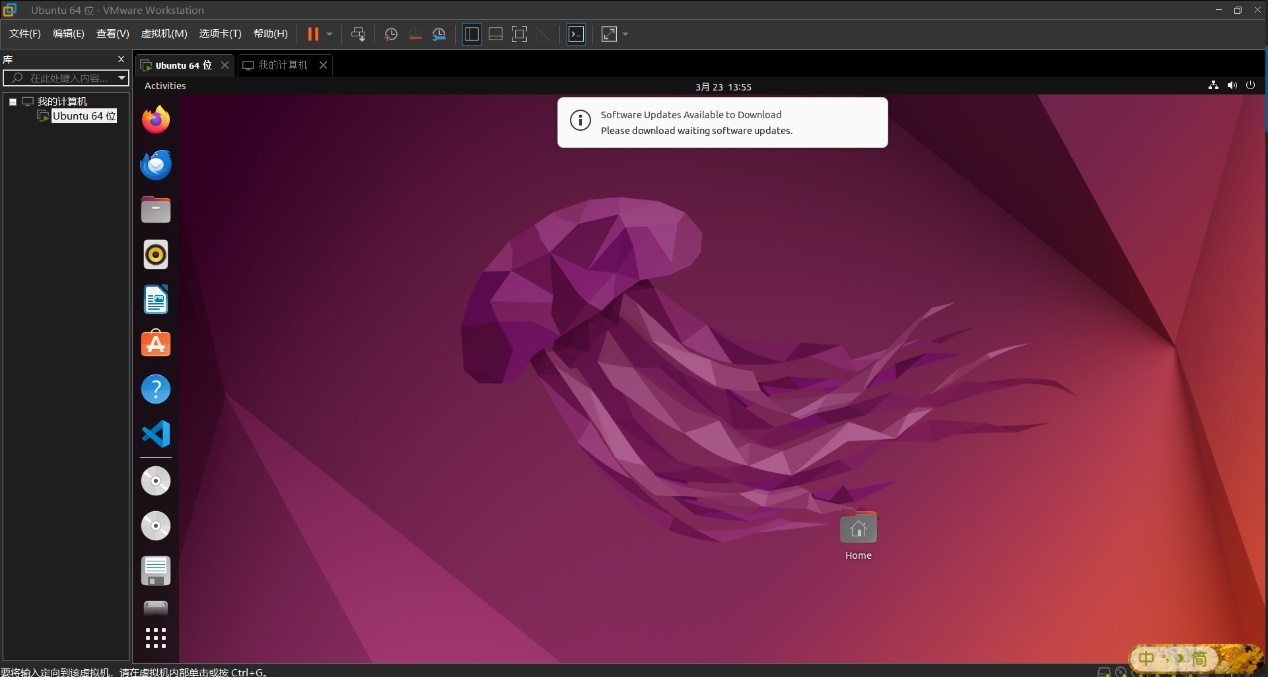


为了开放ipsec和l2tp需要的端口，还需要配置如下安全组，并将目标云主机加入到安全组中：

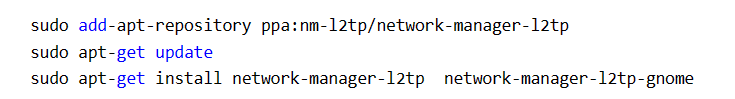


2.2 Linux环境搭建

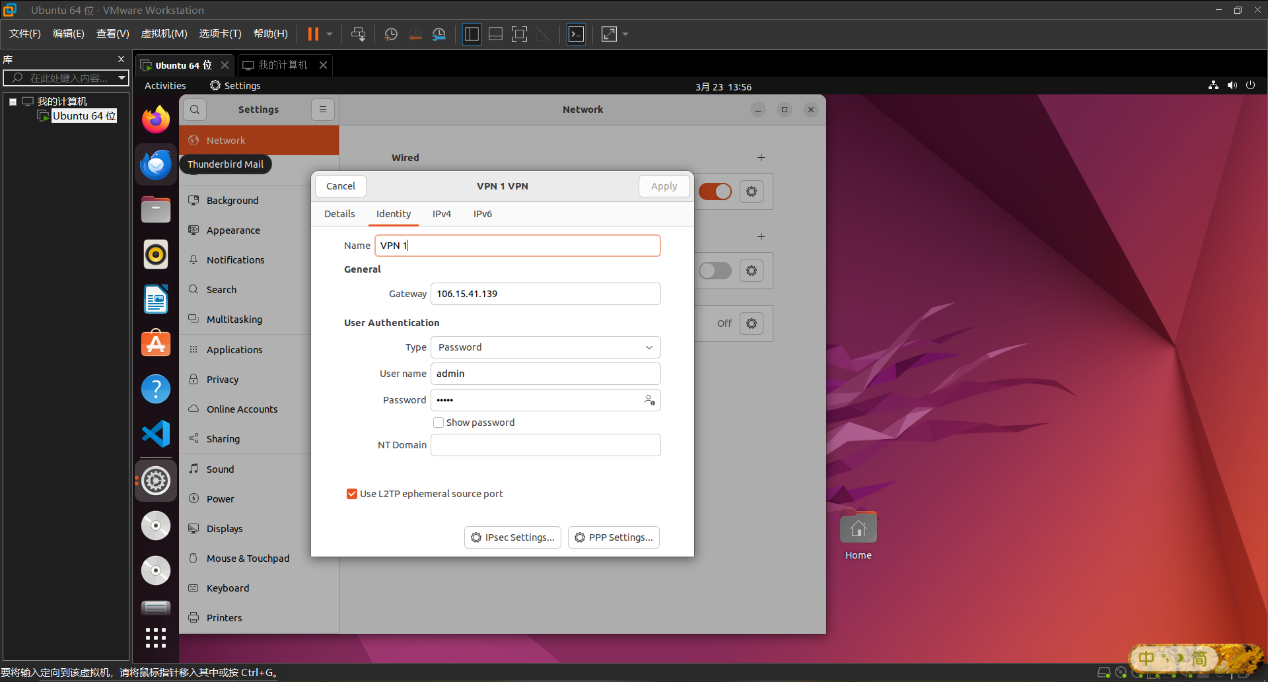
参考网络教程，使用VMware Workstation Pro配置Ubuntu虚拟机



使用相应的Linux命令安装l2tp模块，此后即可使用l2tp vpn



在虚拟机中配置相应服务器的vpn并完成连接，此时虚拟机可以相应监听到服务器中传输的数据包

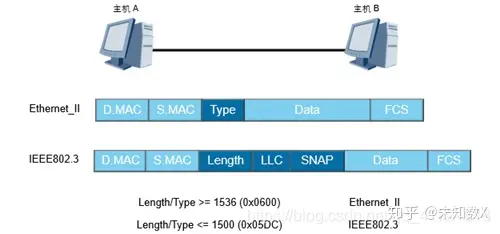


之后，只要在手机上连接到同一个vpn，笔记本电脑就可以通过相应的程序截获手机发出的l2tp包并进行分析

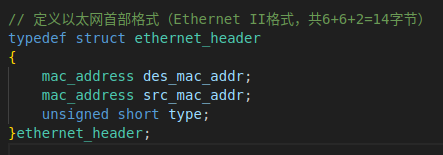
1. 协议分析

3.1 以太网首部分析

通行于数据链路层的以太网帧协议包含Ethernet II和IEEE 802.3两种，两种协议的差别在于Ethernet II包含了一个Type字段以标明该以太网帧被处理后应发往哪一个上层协议进行处理，而IEEE 802.3在相应位置则是一个长度字段



于是，基于Ethernet II以太网帧结构，解析其首部所用的结构体可以如下定义：



以太网帧开头还存在一个8字节的校验码，但是在程序捕获时该校验码已经在经过校验后去除，不需要额外考虑。

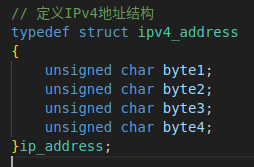
以太网首部中的des\_mac\_addr和src\_mac\_addr分别为目标的MAC地址和来源的MAC地址，MAC地址的结构体定义如下：



MAC地址是标识唯一网卡的地址，是一个48位共6字节长地址，通常表示为12个16进制数，所以用一个6个char型变量集合的结构体即可解析

3.2 IPV4分析

当ethernet\_header.type的解析结果为0x0800时，说明该以太网帧的Data部分使用的是IPV4协议，IPV4的地址结构体定义如图：



IPV4地址是一个32位共4字节的地址，与MAC地址相类似，但IP地址是一个基于逻辑的地址，不能对用户进行唯一标识，而MAC地址是基于物理的地址，能够与硬件一一对应。

在截获数据包并完成Ethernet II首部分析后，程序丢弃14字节的Ethernet II首部并继续对Data部分中的IPV4首部进行分析，IPV4首部是这样封装的：



首个byte包含IP版本和首部长度两个部分，各占4bit。因为是IPV4，所以IP版本应为4，即0b0100。首部长度是IP数据包头部的总长度，以4bit为1个单位，在无**选项字段**时长度为最少的20bit，否则将大于20bit，而最大是60bit，这是因为IP数据包头部以4字节为单位，而首部长度占4个字节，取值在0b0000-0b1111之间（因为头部长度不会为0，所以实际上的取值标准是在0b0101-0b1111之间），所以最长的首部长度最少是5\*4=20bit，最长是是15\*4=60bit。报头总长度必须是4bit的整数倍，如果不是将在选项字段的**填充字段**中填充到4bit的整数倍

第2个byte为区分服务字段，也称为TOS字段，该字段用来定义此IP包的所需要的服务类型，换句话说就是处理此包的优先级，通过此字段的设置可以将IP包的类别进行差异化的划分，使用户得到更好的服务。在没有对IP包的服务类型进行设置的时候，此字段默认为0。

第3、4两个byte是IP包总长度，与前文提到的报头总长度不同，此字段刻画的是整个IP包的长度。IP包总长度是以单个字节为单位的，字段占16bit，所以IP包允许的最大长度是65535bit。另外，这个总长度是包含报头在内的，但是与不在本层进行的协议封装无关，比如说不包含以太网首部的长度。

第5、6个byte是标识字段，IP协议服务在存储器中维护一个16bit整数计数器作为IP包标识，每次当数据超过数据链路层（层二）的MTU（最大传输单元）值而必须分段时，就会将计数器的值复制到此字段，然后计数器+1。因为IP只提供一种无连接、不可靠、尽力而为的服务，所以不存在数据包传输顺序的问题，标识字段并不会当作序号使用。

第7、8个byte是标志和片位移字段，标志占3bit，片位移占13bit。标志用来显示该IP包后是否还有数据包，而片位移显示的是该数据包在分段传输中的相对位置，如果是第一个数据包则为0。显然的，如果本次IP传输不分包，标志位一定显示为没有后续数据包，片位移一定是0。这两个字段用于在分包传输中标记本包。

第9个byte是生存时间字段，也称为TTL，最早单位是秒，但现在一般是跳。数据包具有一个初始的TTL，每经过路由器转发一次（一跳），TTL就-1，减到0路由器将不会处理此包的转发请求，也就是丢包了，显然TTL最多可以设置为255，一个IP包最多转发255次。这样的设计是为了防止IP包无限制循环传递占用带宽资源。

第10个byte是协议字段，标记此IP包在传输层使用的协议类型，以便主机知道使用何种服务。常用的传输层协议中，TCP是6，UDP是17。

第11、12个byte是首部校验和字段，因为每次转发生存时间等字段都会变化，每到达一个路由器就会对其重新计算一次。该校验和只对首部进行校验，与数据部分无关，通过校验和字段可以检查IP包首部是否有被篡改。

接下去的4个字节是源IP地址，再4个字节是目的IP地址，在整个传输过程中，无论经过多少跳这两个地址始终不变。

后面是选项字段，前面的部分刚好占用20个字节，而在首部长度字段也提到过没有选项字段的IP包首部就是20字节，而含选项字段的IP包最长可以到达60个字节。这一字段是供拓展使用的，用来支持排错、测量以及安全等措施，长度在1-40个字节不等，由拓展功能决定。如果选项字段使用的长度不是4的倍数，就需要填充0直到IP包首部为4的倍数为止。

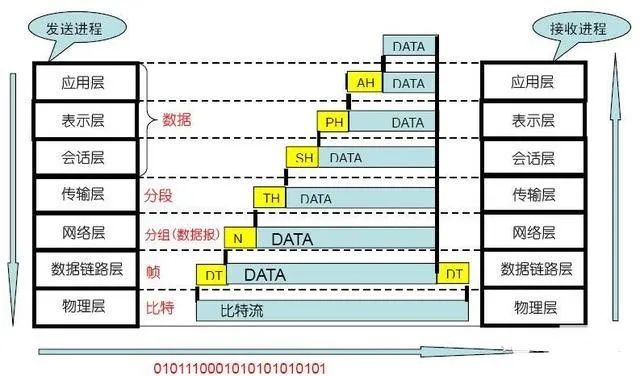
IP包首部到这里就结束了，再往下就是数据部分，程序用以解析IP包首部的结构体如下定义：



最后一个字段，因为不考虑具体的选项，所以仅设计了一个4字节的填补位

3.3 UDP分析

IP包首部结束，如果传输层使用UDP协议，那么接下来数据部分的开头就是UDP的首部，因为OSI七层模型使用如图所示的封装方式：



为了分析传输层协议，指针需要越过IP首部的长度，这一长度在IP首部的某个字段中已经有记录了。

程序只在传输层使用UDP协议的情况下进行下一步分析，即上文提到的IP包首部协议字段为17，因为L2TP在传输层对应的协议就是UDP。

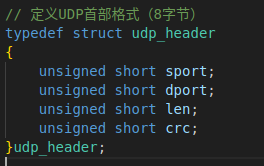
UDP协议封装规则如下：



源端口号和目的端口号都是2个字节；长度为包括UDP首部在内的UDP包总长度，占2个字节；校验和占2个字节，作用类似IP包首部的校验和。

可以发现UDP首部占8个字节，结构非常简单，那是因为UDP并不是面向连接的协议，这意味着它的可靠性、安全性相比面向连接的协议（如TCP）都比较低。

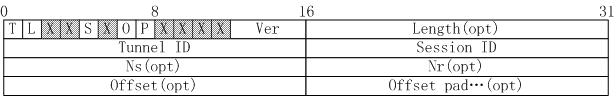
UDP首部结构体定义如下：



3.4 L2TP分析

L2TP是OSI七层中会话层的协议，这意味着它头部的位置在传输层协议UDP之后，因为我们要让指针再越过UDP首部的8个字节以分析L2TP首部的结构。

L2TP首部的封装格式如图：



头两个字节TLXXSXOPXXXXVer是一个多个标志位杂糅起来的字段，Ver占4个字节，是L2TP的版本号，而且一定是2（0B0010）。T是标识消息的类型，0表示是数据消息，1表示是控制消息，L2TP包含这两种不同类型的消息。L置1时，表示Length的值是存在的（有意义的），控制消息的L一定是1。S置1时，表示Ns和Nr的值是存在的（有意义的），控制消息的S一定是1。O置1时，表示Offset的值是存在的（有意义的），控制消息的O一定是1。P置1时，表示该消息在本队列和传输时应该优先处理，控制消息的P一定是0。其他的X都是保留位，均置0。

第2、3个byte为长度字段，标识整个L2TP报文（含首部）的长度

第4、5个byte为隧道标识符，只有本地意义

第5、6个byte为会话标识符，只有本地意义

第7、8个byte Ns标识发送数据或控制消息的序号，从0开始递增到216，步长为1，到达216后再次从0开始

第9、10个byte Nr标识下一个期望收到的控制消息，这设置为上一个接受到的控制消息的Ns+1，这个字段实际上是对接收到的控制消息的确认，对于数据消息来说是无意义的

第11、12个字段是偏移量，就是L2TP的数据区域（有效载荷）的相对位置

剩下的都是填充位，L2TP首部最终包含14个字节（含填充）。

L2TP首部解析的结构体定义如下：

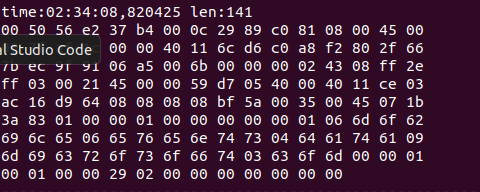


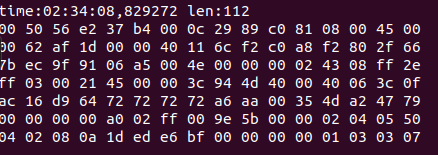
1. 代码展示

**4.1 抓取l2tp流量**

|  |
| --- |
| **// 获取本机设备列表**  **if (pcap\_findalldevs(&alldevs, errbuf) == -1){**  **fprintf(stderr, "Error in pcap\_findalldevs: %s\n", errbuf);**  **exit(1);**  **}**  **// 打印列表**  **for (i = 0, d = alldevs; d; d = d->next, i++){**  **ifprint(d, i + 1);**  **}**  **if (i == 0){**  **printf("No interfaces found! Make sure WinPcap is installed.\n");**  **return -1;**  **}**  **printf("Enter the interface number (1-%d):", i);**  **scanf("%d", &inum);**  **if (inum < 1 || inum > i){**  **printf("\nInterface number out of range.\n");**  **pcap\_freealldevs(alldevs); // 释放设备列表**  **return -1;**  **}**  **// 跳转到已选中的适配器**  **for (d = alldevs, i = 0; i < inum - 1; d = d->next, i++);**  **// 打开设备**  **if ((adhandle = pcap\_open\_live(d->name, // 设备名**  **BUFSIZ,**  **1,**  **1000,**  **errbuf // 错误缓冲池**  **)) == NULL){**  **fprintf(stderr, "\nUnable to open the adapter. %s is not supported by WinPcap\n", d->name);**  **pcap\_freealldevs(alldevs);**  **return -1;**  **}**  **// 检测链接层**  **if (pcap\_datalink(adhandle) != DLT\_EN10MB){**  **fprintf(stderr, "\n此程序只能运行在以太网上.\n");**  **pcap\_freealldevs(alldevs);**  **return -1;**  **}**  **// if (d->addresses != NULL) {**  **// netmask = ((struct sockaddr\_in\*)(d->addresses->netmask))->sin\_addr.s\_addr;// 获取接口第一个地址的掩码**  **// }**  **// else {**  **// netmask = 0xffffff; // 如果这个接口没有地址，那么我们假设这个接口在C类网络中**  **// }**  **if (pcap\_compile(adhandle, &fcode, packet\_filter, 1, netmask) >= 0){**  **// 设置过滤器**  **if (pcap\_setfilter(adhandle, &fcode) < 0){**  **fprintf(stderr, "\nError setting the filter.\n");**  **pcap\_freealldevs(alldevs);**  **return -1;**  **}**  **}**  **else {**  **fprintf(stderr, "\nError setting the filter.\n");**  **pcap\_freealldevs(alldevs);**  **return -1;**  **}**  **printf("\nlistening on %s...\n", d->description);**  **pcap\_freealldevs(alldevs);** |

**最终结果通过协议的时间的连续性，且实时输出，证明抓取的实时性，结果如下：**





**4.2 针对每个目标用户，实时监控其l2tp请求，并分析、还原、呈现其连接和业务载荷**

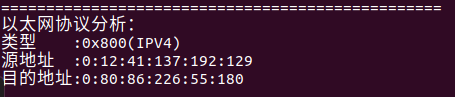
**先定义了各个协议相应的数据结构来接受相应分析结果**

|  |
| --- |
| **// 定义mac地址格式**  **typedef struct mac\_address**  **{**  **u\_char byte1;**  **u\_char byte2;**  **u\_char byte3;**  **u\_char byte4;**  **u\_char byte5;**  **u\_char byte6;**  **}mac\_address;**  **// 定义以太网首部格式**  **typedef struct ethernet\_header**  **{**  **mac\_address des\_mac\_addr;**  **mac\_address src\_mac\_addr;**  **u\_short type;**  **}ethernet\_header;**  **// 定义IPv4地址结构**  **typedef struct ipv4\_address**  **{**  **u\_char byte1;**  **u\_char byte2;**  **u\_char byte3;**  **u\_char byte4;**  **}ip\_address;**  **// 定义IP首部格式**  **typedef struct ipv4\_header**  **{**  **u\_char ver\_ihl; // 版本和首部长度**  **u\_char tos; // 服务类型**  **u\_short tlen; // 总长度**  **u\_short identification; // 标识号**  **u\_short flags\_fo; // 段偏移量**  **u\_char ttl; // 生存时间**  **u\_char proto; // 协议**  **u\_short crc; // 首部校验和**  **ipv4\_address saddr; // 源ip地址**  **ipv4\_address daddr; // 目的地址**  **u\_int op\_pad; // 选项和填补位**  **}ip\_header;**  **// 定义UDP首部格式**  **typedef struct udp\_header**  **{**  **u\_short sport; // 16bit源端口**  **u\_short dport; // 16bit目的端口**  **u\_short len; // 16bit长度**  **u\_short crc; // 16bit 校验和**  **}udp\_header;**  **// 定义l2tp首部格式**  **typedef struct l2tp\_header**  **{**  **u\_char tlxxsxop; // t类型（0数据1控制）**  **// l长度在位标志（控制必为1）**  **// s顺序字段在位标志（1存在ns nr控制必为1）**  **// o偏移值在位标志**  **// p优先级（用于数据消息 控制必为0）**  **u\_char xxxxver; // 版本号**  **u\_short length; // 消息总长度**  **u\_short tunnel\_id; // 隧道标识符 本地意义**  **u\_short session\_id; // 会话标识符 本地意义**  **u\_short ns; // 当前消息顺序号**  **u\_short nr; // 下一控制消息顺序号，数据消息为保留字段**  **u\_short offset; // 偏移值 指示载荷开始位置**  **u\_short offser\_pading; // 偏移量填充**  **}l2tp\_header;** |

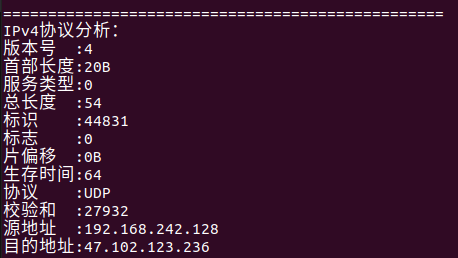
**再通过相应的解包函数完成对协议的分析：**

|  |
| --- |
| **// l2tp解包函数**  **int decode\_l2tp(char\* l2tpbuf)**  **{**  **struct l2tp\_header\* pl2tpheader;**  **pl2tpheader = (l2tp\_header\*)l2tpbuf;**  **u\_short t, l, s, o;**  **t = (pl2tpheader->tlxxsxop & 0x80) >> 7;**  **l = (pl2tpheader->tlxxsxop & 0x40) >> 6;**  **s = (pl2tpheader->tlxxsxop & 0x08) >> 3;**  **o = (pl2tpheader->tlxxsxop & 0x02) >> 1;**  **printf("\n=================================================\n");**  **printf("L2TP协议分析：\n");**  **printf("0x%x\n", pl2tpheader->tlxxsxop);**  **printf("类型 :%s\n", t ? "1(控制信息)" : "0(数据信息)");**  **printf("长度在位标志 :%d\n", l);**  **printf("顺序字段在位标志:%d\n", s);**  **printf("偏移值在位标志 :%d\n", o);**  **printf("优先级 :%d\n", pl2tpheader->tlxxsxop & 0x01);**  **printf("版本号 :%d\n", pl2tpheader->xxxxver & 0x0f);**  **if (l == 1) { // 长度在位标志为1**  **printf("消息总长度 :%d\n", ntohs(pl2tpheader->length));**  **}**  **printf("隧道标识符 :%d\n", ntohs(pl2tpheader->tunnel\_id));**  **printf("会话标识符 :%d\n", ntohs(pl2tpheader->session\_id));**  **if (s == 1) { // 顺序字段在位标志为1**  **printf("当前消息顺序号 :%d\n", ntohs(pl2tpheader->ns));**  **if (t == 1) { // 控制信息nr才有意义**  **printf("下一消息顺序号 :%d\n", ntohs(pl2tpheader->nr));**  **}**  **}**  **if (l == 1) { // 偏移值在位标志为1**  **printf("偏移量 :%d\n", ntohs(pl2tpheader->offset));**  **}**  **return true;**  **}**  **// UDP解包函数,l2tp用UDP协议**  **int decode\_udp(char\* udpbuf)**  **{**  **udp\_header\* pudpheader;**  **pudpheader = (udp\_header\*)udpbuf;**  **printf("\n=================================================\n");**  **printf("UDP协议分析：\n");**  **printf("源端口 :%d\n", ntohs(pudpheader->sport));**  **printf("目的端口:%d\n", ntohs(pudpheader->dport));**  **printf("数据长度:%d\n", ntohs(pudpheader->len));**  **printf("校验和 :%d\n", ntohs(pudpheader->crc));**  **if (ntohs(pudpheader->sport) == 4500 && ntohs(pudpheader->dport) == 4500) {**  **printf("L2tp over IpSec connection has been found.\n");**  **}**  **else if (ntohs(pudpheader->sport) == 1701 || ntohs(pudpheader->dport) == 1701) {**  **printf("L2tp connection has been found.\n");**  **decode\_l2tp((char\*)(udpbuf + 8));**  **}**  **return true;**  **}**  **// IP解包函数**  **int decode\_ipv4(char\* ipbuf)**  **{**  **ip\_header\* ih;**  **u\_int ip\_len;**  **printf("\n=================================================\n");**  **printf("IPv4协议分析：\n");**  **// 返回IP首部的位置**  **ih = (ip\_header\*)(ipbuf);**  **// IP首部长度**  **ip\_len = (ih->ver\_ihl & 0xf) \* 4;**  **printf("版本号 :%d\n", (ih->ver\_ihl & 0xf0) >> 4);**  **printf("首部长度:%dB\n", ip\_len);**  **printf("服务类型:%d\n", ih->tos);**  **printf("总长度 :%d\n", ntohs(ih->tlen));**  **printf("标识 :%d\n", ntohs(ih->identification));**  **printf("标志 :%d\n", (ih->flags\_fo & 0xe000) >> 12);**  **printf("片偏移 :%dB\n", (ih->flags\_fo & 0x1fff) \* 8);**  **printf("生存时间:%d\n", int(ih->ttl));**  **printf("协议 :%s\n", ih->proto == IPPROTO\_UDP ? "UDP" : "\*");**  **printf("校验和 :%d\n", ntohs(ih->crc));**  **printf("源地址 :%d.%d.%d.%d\n", ih->saddr.byte1, ih->saddr.byte2, ih->saddr.byte3, ih->saddr.byte4);**  **printf("目的地址:%d.%d.%d.%d\n", ih->daddr.byte1, ih->daddr.byte2, ih->daddr.byte3, ih->daddr.byte4);**    **// 选项和填补位**  **printf("\n");**  **if (ih->proto == IPPROTO\_UDP) {**  **decode\_udp((char\*)(ipbuf + ip\_len));**  **}**  **return true;**  **}**  **// 以太网分析**  **int decode\_ethernet(char\* etherbuf)**  **{**  **ethernet\_header\* peheader;**  **u\_short type;**  **peheader = (ethernet\_header\*)etherbuf;**  **printf("\n=================================================\n");**  **printf("以太网协议分析：\n");**  **type = ntohs(peheader->type);**  **printf("类型 :0x%x", type);**  **switch (type){**  **case 0x0800:**  **printf("(IPV4)\n");**  **break;**  **case 0x86DD:**  **printf("(IPV6)\n");**  **break;**  **case 0x0806:**  **printf("(ARP)\n");**  **break;**  **case 0x0835:**  **printf("(RARP)\n");**  **break;**  **default:**  **break;**  **}**  **printf("源地址 :%d:%d:%d:%d:%d:%d\n",**  **peheader->src\_mac\_addr.byte1,**  **peheader->src\_mac\_addr.byte2,**  **peheader->src\_mac\_addr.byte3,**  **peheader->src\_mac\_addr.byte4,**  **peheader->src\_mac\_addr.byte5,**  **peheader->src\_mac\_addr.byte6);**  **printf("目的地址:%d:%d:%d:%d:%d:%d\n",**  **peheader->des\_mac\_addr.byte1,**  **peheader->des\_mac\_addr.byte2,**  **peheader->des\_mac\_addr.byte3,**  **peheader->des\_mac\_addr.byte4,**  **peheader->des\_mac\_addr.byte5,**  **peheader->des\_mac\_addr.byte6);**  **printf("\n");**  **if (type == 0x0800) {**  **decode\_ipv4((char\*)(etherbuf + 14));**  **}**  **return true;**  **}** |

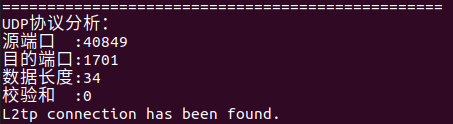
**最终得到：以太网协议分析结果：**



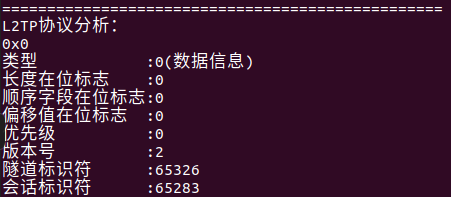
**IPv4协议分析结果：**



**UDP协议分析结果：**



**L2TP协议分析结果：**

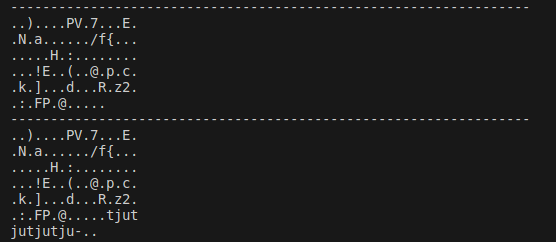


**4.3 支持对特定目标的净载荷替换**

**首先通过分析找到数据包的净载荷部分，并将特定数据包的净载荷替换为自己的data数据，替换完成后再按照L2TP的报文结构进行封装，产生一个新的数据包。然后再重新拆包，得到我们替代完成的新净载荷。代码部分如下：**

|  |
| --- |
| **char a[15] = "tjutjutjutju-";**  **// 将一个unsigned long型的IP转换为字符串类型的IP**  **#define IPTOSBUFFERS 12**  **char\* iptos(u\_long in)**  **{**  **static char output[IPTOSBUFFERS][3 \* 4 + 3 + 1];**  **static short which;**  **u\_char\* p;**  **p = (u\_char\*)&in;**  **which = (which + 1 == IPTOSBUFFERS ? 0 : which + 1);**    **// 格式化字符串**  **sprintf(output[which], "%d.%d.%d.%d", p[0], p[1], p[2], p[3]);**  **return output[which];**  }  **// 包处理回调函数，对于每个嗅探到的数据包**  **void packet\_handler(u\_char\* param, const struct pcap\_pkthdr\* header, const u\_char\* pkt\_data)**  **{**  **struct tm\* ltime;**  **char timestr[16];**  **time\_t local\_tv\_sec;**  **char mypkt[1000];**  **// 将时间戳转换成可识别的格式**  **local\_tv\_sec = header->ts.tv\_sec;**  **ltime = localtime(&local\_tv\_sec);**  **strftime(timestr, sizeof timestr, "time:%H:%M:%S", ltime);**  **printf("%s,%.6ld len:%d\n", timestr, header->ts.tv\_usec, header->len);**  **// 打印包**  **int len = header->caplen + 1;**  **for (int i = 1; i < len; i++)**  **{**  **if (i < 93)**  **mypkt[i - 1] = pkt\_data[i - 1];**  **printf("%.2x ", pkt\_data[i - 1]);**  **if ((i % 16) == 0)**  **printf("\n");**  **}**  **printf("\n-----------------------------------------------------------------\n");**  **for (int i = 1; i < len; i++)**  **{**  **char temp = mypkt[i - 1];**  **if (temp >= 32 && temp <= 126) {**  **printf("%c", temp);**  **}**  **else {**  **printf(".");**  **}**  **if ((i % 16) == 0)**  **printf("\n");**  **}**  **printf("\n-----------------------------------------------------------------\n");**  **for (int i = 93; i < 93 + 15; i++) {**  **mypkt[i - 1] = a[i - 93];**  **}**  **for (int i = 1; i < 93 + 15; i++)**  **{**  **char temp = mypkt[i - 1];**  **if (temp >= 32 && temp <= 126) {**  **printf("%c", temp);**  **}**  **else {**  **printf(".");**  **}**  **if ((i % 16) == 0)**  **printf("\n");**  **}**  **decode\_ethernet((char\*)pkt\_data);**  **printf("\n-----------------------------------------------------------------\n");**  **printf("\n\n");**  **}** |

**最终结果如下：**



1. 总结

通过本次实践，我们学习了云主机ECS服务器的搭建过程，以及在Linux下使用libpcap抓包并分析的过程，为将来的实践开拓了眼界，打下了重要的基础。

# 

# 参考文献

[1]ChestnutSilver. 同济大学信息安全原理课程作业[EB/OL].2023[2024.3.20]

https://github.com/ChestnutSilver/l2tp-analysis

[2]htttw. libpcap使用

https://blog.csdn.net/htttw/article/details/7521053

# [3]wangwuyy. 以太网帧，IP，TCP，UDP首部结构

https://blog.csdn.net/wangtingyao1990/article/details/52809065

# [4]hsb0917. L2TP协议笔记2---L2TP主要协议报文分析

https://blog.csdn.net/hsb0917/article/details/85239198

# [5] [薛凌康](https://blog.csdn.net/qq_35433926). 阿里云ECS部署L2TP/IPSEC，访问服务器内网

https://blog.csdn.net/qq\_35433926/article/details/103188830