我这里的重定向环境是对客户端的请求直接截断重定向到其他页面。

以太头和Ip头想应数据对调。

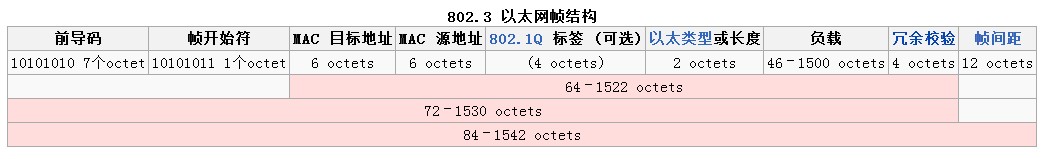
Tcp头flags标记和序号问题。主要问题

数据内容主要的就两条：

"HTTP/1.1 302 Moved Temporarily"

"Location: 路径"

**一、以太头结构**



在帧结构里我用到的有ＭＡＣ源和目的地址，以太类型。至于802.1Ｑ标签，如果加了，对以太类型想对地址要向生移４个字节。源来的地址值则为0x8100。

用到的以太类型是　0x800 表以后一个负载段的内容是一个IP包。这个宏定义在if\_ether.h头文件里。

#define ETH\_P\_PUPAT 0x0201 /\* Xerox PUP Addr Trans packet \*/

#define ETH\_P\_IP 0x0800 /\* Internet Protocol packet \*/

#define ETH\_P\_X25 0x0805 /\* CCITT X.25 \*/

#define ETH\_P\_ARP 0x0806 /\* Address Resolution packet \*/

先看是不是IP包

是：对换MAC地址，复制以太头结构到buffer

不是：这个就直接转发的转发，丢包的丢包了不管了。

unsigned short \*port = NULL;

unsigned short h\_poto = 0;

if(pkt->eth\_hdr\_ptr->h\_proto == htons(ETH\_P\_8021Q))

{

h\_poto = \*((unsigned short \*) pkt ->eth\_hdr\_ptr + 8);

}

else

{

h\_poto = pkt ->eth\_hdr\_ptr->h\_proto;

}

if ( h\_poto != htons(ETH\_P\_IP) )

{

return ERROR\_FAIL;

}

port = (unsigned short \*) pkt->l4\_hdr\_ptr;

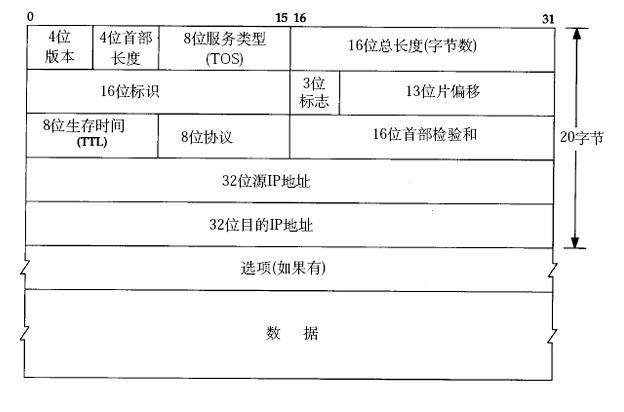
if(port == NULL)

{

return ERROR\_FAIL;

}

IP报文头结构的话



struct iphdr {

#if defined(\_\_LITTLE\_ENDIAN\_BITFIELD)

\_\_u8 ihl:4,

version:4;

#elif defined (\_\_BIG\_ENDIAN\_BITFIELD)

\_\_u8 version:4,

ihl:4;

#else

#error "Please fix <asm/byteorder.h>"

#endif

\_\_u8 tos;

\_\_be16 tot\_len;

\_\_be16 id;

\_\_be16 frag\_off;

\_\_u8 ttl;

\_\_u8 protocol;

\_\_sum16 check;

\_\_be32 saddr;

\_\_be32 daddr;

/\*The options start here. \*/

};

Version = 4;

Ihl 一般情况下都是5。我在重定向的时候直接 new\_ipv4\_hdr->ihl = old\_ipv4\_hdr->ihl;因为长度使用收到报文头长度，所以如果后还有选项的话是直接复制到新的头结构中。

Tos两样复制接收到的报文数据。现在都不怎么用这个字段了。

Tot\_len是指从IP头结构到数据的总长度，自己算。我在算你在算，在校验之前算好。

Id我比较简单直接用0。这个会用于分片和重新组装数据报。我也没用到只是发一个302重定向报文。

Frag\_off直接0。我也没有分片，用不到偏移。

Ttl一般不是32就是64。

Protoco 根据自己的实际情况为6。

Check 去网上找个现成的就可以。也可以找找lib库接口函数

Saddr daddr 直接对换了就可以。

要是有大小端的情况下，在这些头结构中如果字段是32位的就用 htonl 或是 ntohl；要是16位的话就用 htons 或是 ntohs。这个问题当时还纠结了好几天。唉

int reply\_packet\_iphdr\_data(pkt\_info\_s \*pkt, struct iphdr \*ipv4\_hdr)

{

//其他没有填写字段的内容需要在算校验之前填好

ipv4\_hdr->id=0;

ipv4\_hdr->version=4;

ipv4\_hdr->ihl=pkt->ipv4\_hdr\_ptr->ihl;

ipv4\_hdr->ttl=64;

ipv4\_hdr->protocol=6; //tcp 6

ipv4\_hdr->frag\_off=ntohs(0x4000);

ipv4\_hdr->tos= pkt ->ipv4\_hdr\_ptr->tos;

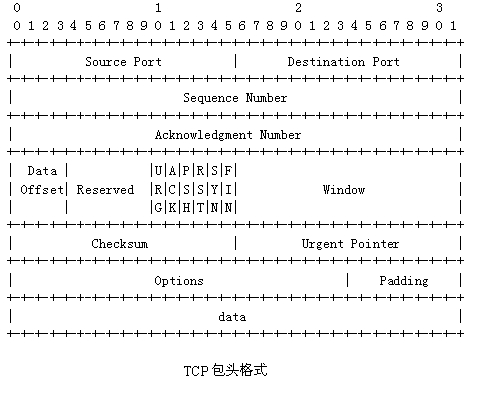
ipv4\_hdr->daddr= pkt ->ipv4\_hdr\_ptr->saddr;

ipv4\_hdr->saddr= pkt ->ipv4\_hdr\_ptr->daddr;

ipv4\_hdr->check = 0;

return ERROR\_OK;

}



struct tcphdr {

\_\_be16 source;

\_\_be16 dest;

\_\_be32 seq;

\_\_be32 ack\_seq;

#if defined(\_\_LITTLE\_ENDIAN\_BITFIELD)

\_\_u16 res1:4,

doff:4,

fin:1,

syn:1,

rst:1,

psh:1,

ack:1,

urg:1,

ece:1,

cwr:1;

#elif defined(\_\_BIG\_ENDIAN\_BITFIELD)

\_\_u16 doff:4,

res1:4,

cwr:1,

ece:1,

urg:1,

ack:1,

psh:1,

rst:1,

syn:1,

fin:1;

#else

#error "Adjust your <asm/byteorder.h> defines"

#endif

\_\_be16 window;

\_\_sum16 check;

\_\_be16 urg\_ptr;

};

TCP协议头最少20个字节，和IP头到是一样

TCP源端口(Source Port)，TCP目的端口(Destination port) 对换

　　TCP序列号（序列码,Sequence Number）：32位

TCP应答号(Acknowledgment Number)：32位

数据偏移量(HLEN)：4位包括TCP头大小，指示何处数据开始。其实也是tcp头部长度。

　　保留(Reserved)：6位值域，这些位必须是0。为了将来定义新的用途所保留。

标志(Code Bits)：6位标志域：

1.       URG：紧急标志

紧急(The urgent pointer) 标志有效。紧急标志置位，

2.       ACK：确认标志

确认编号(Acknowledgement Number)栏有效。大多数情况下该标志位是置位的。TCP报头内的确认编号栏内包含的确认编号(w+1，Figure：1)为下一个预期的序列编号，同时提示远端系统已经成功接收所有数据。

3.       PSH：推标志

该标志置位时，接收端不将该数据进行队列处理，而是尽可能快将数据转由应用处理。在处理 telnet 或rlogin 等交互模式的连接时，该标志总是置位的。

4.       RST：复位标志

复位标志有效。用于复位相应的TCP连接。

5.       SYN：同步标志

同步序列编号(Synchronize Sequence Numbers)栏有效。该标志仅在三次握手建立TCP连接时有效。它提示TCP连接的服务端检查序列编号，该序列编号为TCP连接初始端(一般是客户端)的初始序列编号。在这里，可以把TCP序列编号看作是一个范围从0到4，294，967，295的32位计数器。通过TCP连接交换的数据中每一个字节都经过序列编号。在TCP报头中的序列编号栏包括了TCP分段中第一个字节的序列编号。

6.       FIN：结束标志

带有该标志置位的数据包用来结束一个TCP回话，但对应端口仍处于开放状态，准备接收后续数据。

　　窗口(Window)：16位，用来表示想收到的每个TCP数据段的大小。随便写个就好不要太小了。

　　校验位(Checksum)：16位TCP头。源机器基于数据内容计算一个数值，收信息机要与源机器数值 结果完全一样，从而证明数据的有效性。

　　优先指针（紧急,Urgent Pointer）：16位，指向后面是优先数据的字节，在URG标志设置了时才有效。如果URG标志没有被设置，紧急域作为填充。加快处理标示为紧急的数据段。

选项(Option)：长度不定，但长度必须以字节。如果 没有 选项就表示这个一字节的域等于0。　一般只用出到在syn包中

数据（Date）：应用程序的数据。

从三次握手开始。

C -------> S: syn 1; ack 0;客户端seq，ack\_seq会有值。可能是一个随机值，写一个就可以

C <------- S: syn 1; ack 1; 捎带ack。服务端回 seq = 随机值。Ack\_seq = old\_seq + 1

C -------> S: syn 0; ack 1; 端回 seq = old\_ack\_seq; Ack\_seq = old\_seq + 1

C -------> S: get psh 1; ack 1; 端回 seq Ack\_seq 和上一个ack包的一样。

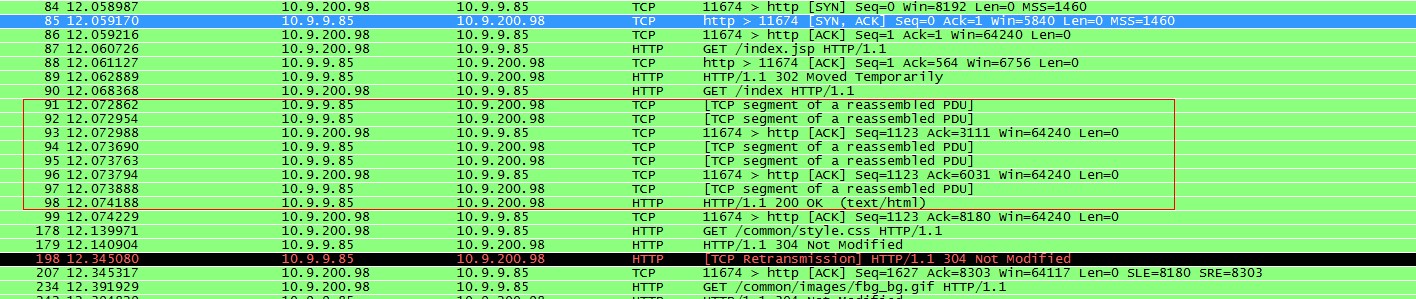
C <------- S: 302 fin 1; psh 0; ack 1; 端回 seq = old\_ack\_seq; Ack\_seq = old\_seq + data\_len

对于客户端发过来的seq和ack\_seq 值直接取就可以，为了给回包用。之后就直接丢了。

在头两个syn包里会有选项字段内容。要注意MSS

重定向主要就回了两个包：一个是syn\_ack包。另一个是http 302包。

注意get 报文可能会进行tcp分片：发两个ack数据包等一个ack确认回包。真到psh,ack包发之后才算是一个完整的get 报文。



//这些数据基本不用变。

tcp\_hdr->dest = pdesc->tcp\_hdr\_ptr->source;

tcp\_hdr->source = pdesc->tcp\_hdr\_ptr->dest;

tcp\_hdr->doff = pdesc->tcp\_hdr\_ptr->doff;

tcp\_hdr->window = pdesc->tcp\_hdr\_ptr->window;

tcp\_hdr->check = 0;

tcp\_hdr->seq = pdesc->tcp\_hdr\_ptr->ack\_seq;

tcp\_hdr->ack = 1;

// ack\_seq字段的两种情况

if(send\_falg != 0)

{

tcp\_hdr->ack\_seq = htonl(ntohl(pdesc->tcp\_hdr\_ptr->seq)+ data\_len);

}

else

{

tcp\_hdr->ack\_seq = htonl(ntohl(pdesc->tcp\_hdr\_ptr->seq)+ 1);

}