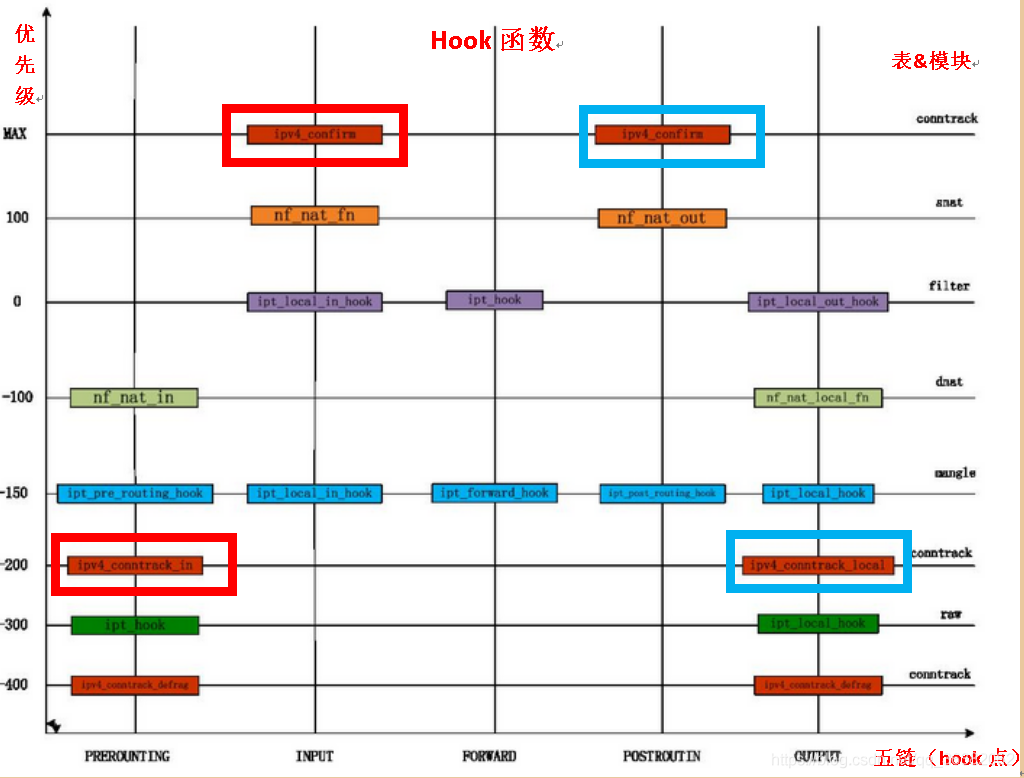
SNAT并钩包封装发送后，回包不能DNAT的解决记录

1. 连接跟踪相关函数在Linux内核的钩子点。这张图是老版本的Linux内核钩子函数，v4.15的nf\_nat\_out已变为 iptable\_nat\_ipv4\_out。



1. SNAT位置:

SNAT发生在POSTROUTING钩子函数的优先级为NF\_IP\_PRI\_NAT\_SRC = 100的位置。也就是说，如果别的PC数据包经过本服务器转发，在POSTROUTING的NF\_IP\_PRI\_FIRST = INT\_MIN钩包，源IP为别的PC的IP；在NF\_IP\_PRI\_LAST = INT\_MAX位置钩包，源IP为本服务器的IP。

Linux内核对NAT转换的钩子注册结构体如下，在本文中我们重点关注iptable\_nat\_ipv4\_out，该函数将别的PC的源IP转换为本地网卡的源IP。

static const struct nf\_hook\_ops nf\_nat\_ipv4\_ops[] = {

/\* Before packet filtering, change destination \*/

{

.hook = iptable\_nat\_ipv4\_in,

.pf = NFPROTO\_IPV4,

.hooknum = NF\_INET\_PRE\_ROUTING,

.priority = NF\_IP\_PRI\_NAT\_DST,

},

/\* After packet filtering, change source \*/

{

.hook = iptable\_nat\_ipv4\_out,

.pf = NFPROTO\_IPV4,

.hooknum = NF\_INET\_POST\_ROUTING,

.priority = NF\_IP\_PRI\_NAT\_SRC,

},

/\* Before packet filtering, change destination \*/

{

.hook = iptable\_nat\_ipv4\_local\_fn,

.pf = NFPROTO\_IPV4,

.hooknum = NF\_INET\_LOCAL\_OUT,

.priority = NF\_IP\_PRI\_NAT\_DST,

},

/\* After packet filtering, change source \*/

{

.hook = iptable\_nat\_ipv4\_fn,

.pf = NFPROTO\_IPV4,

.hooknum = NF\_INET\_LOCAL\_IN,

.priority = NF\_IP\_PRI\_NAT\_SRC,

},

};

1. 现在的问题：
2. 没有myfunc()钩子点时包能顺利发出，返回时也能被正确的被NAT转换。
3. 有myfunc()且返回NF\_STOLEN时，对端返回的包解封后不能被NAT转换回来。在PRE\_ROUTING的最高优先级和最低优先级钩包，目的IP均是192.168.1.7。（理论上在最低优先级钩包，目的IP应为192.168.2.2）

Linux内核行为的表现给人的感觉是，一个数据包虽然在钩子表的中间位置进行了转换，但必须要完整地在协议栈走一遍，并发送出去，转换才会被记录。

//封包之后返回NF\_ACCEPT会是什么结果

尝试在二层钩包，利用ebtables的相关钩子点，但发现勾不到。原因是只有网桥转发的数据包才会经过这个钩子点，三层转发的数据包不会经过ebtables。



1. 解决办法

查阅大量资料，发现Linux内核有一个被称为nf\_conntrack\_count的模块。该模块内部维护着四张表：

Confirm表：系统正式连接追踪表，即/proc/net/nf\_conntrack看到的列表

Expired表：期望值表（暂没有深入了解）

Unconfirm表：记录已经进入连接追踪系统，但还没有加入正式追踪表的连接

Dying表：记录已经在正式追踪表超时删除，等待系统最终删除的连接块

我们的问题就出现在，经过iptable\_nat\_ipv4\_out()函数的NAT转换后，转换记录只是加入了Unconfirm表。被钩包后，该数据包没有继续在内核协议栈流动，导致没有进入Confirm表。

//myfunc也是最低优先级并且比ipv4\_confirm()函数后注册为什么没有被confirm？

i = 0;

nhooks = 0;

while (i < old\_entries) {

if (orig\_ops[i] == &dummy\_ops) {

++i;

continue;

}

//待插入的钩子优先级大于当前循环到的优先级，注意是大于号

//若是相同优先级会插入到之前钩子的前面

if (inserted || reg->priority > orig\_ops[i]->priority) {

new\_ops[nhooks] = (void \*)orig\_ops[i];

new->hooks[nhooks] = old->hooks[i];

i++;

} else {

new\_ops[nhooks] = (void \*)reg;

new->hooks[nhooks].hook = reg->hook;

new->hooks[nhooks].priv = reg->priv;

inserted = true;

}

nhooks++;

}

根据钩子函数注册源码可以看出在相同优先级情况下，后注册的钩子反而在先注册钩子的前面。因此myfunc()函数在ipv4\_confirm()函数前面，stolen之后发送跳过了ipv4\_confirm()函数

继续查阅资料，发现confirm的过程发生在ipv4\_confirm()函数里，该函数挂载在POSTROUTING的最低优先级。

{

.hook = ipv4\_confirm,

.pf = NFPROTO\_IPV4,

.hooknum = NF\_INET\_POST\_ROUTING,

.priority = NF\_IP\_PRI\_CONNTRACK\_CONFIRM,

},

该函数的调用树与主要功能如下图所示。



由于我们的模块在ipv4\_confirm()执行之前就返回NF\_STOLEN，导致无法执行确认，没有进入Confirm表，因此回包不能执行DNAT。

Ipv4\_confirm()的优先级为NF\_IP\_PRI\_CONNTRACK\_CONFIRM = INT\_MAX，已无法找到比它更低的优先级，因此我们在钩包后，在代码里直接调用ipv4\_confirm()，再返回NF\_STOLEN。

但这样还有一个问题，ipv4\_confirm()定义在net/ipv4/netfilter/nf\_conntrack\_l3proto\_ipv4.c，但是没有EXPORT\_SYMBOL导出该函数，也没有头文件对它进行声明。故不能直接调用该函数。我们将函数从内核源码里直接复制到模块代码里，解决了这一问题。继续观察该函数调用的子函数，都是定义在头文件或用了EXPORT\_SYMBOL修饰的，因此子函数不需要复制到模块代码里，只需要包含以下几个头文件：

#include <net/netfilter/nf\_conntrack.h>

#include <net/netfilter/nf\_conntrack\_core.h>

#include <net/netfilter/nf\_conntrack\_seqadj.h>

#include <net/netfilter/nf\_conntrack\_ecache.h>

1. 总结

综上所述，只要在postrouting()钩子函数里，将要返回NF\_STOLEN之前，调用ipv4\_confirm()函数，即可将NAT转换记录添加进Confirm表，回包即可实现DNAT转换。

参考https://blog.csdn.net/qq\_36382062/article/details/105959555