一、故障现象

931WII塞尔维亚版本在WAN侧抓到来自192.168.1.2的包。**解决故障，是对故障现象仔细收集并逐一解析的过程。**

故障现象：

1. 私网包都是FIN，RST包。

2. 出现私网地址泄漏前后，都会出现PPPoE的错包，查看是ppp层在包的最后补了6个0。

```
PPP-over-Ethernet Session
    0001 .... = Version: 1
    .... 0001 = Type: 1
    Code: Session Data (0x00)
    Session ID: 0x3682
  ⊞ Payload Length: 42 [incorrect, should be 48]
```

3. PC有发RST包，Modem收到后转发，SEQ和ACK编号都为1

4. PC和服务器正常交互后，过一分钟以上如果PC再发送FIN/RST包，就会出现泄漏。

13	5.831433	109.92.210.243	74.125.232.231	TCP	56185 > 80 [ACK] Seq=2709 Ack=991 Win=65788 Len=0
14	121.036691	192.168.1.2	74.125.232.231	TCP	56185 > 80 [FIN, ACK] Seq=1 Ack=1 Win=16447 Len=0

二、故障分析

怀疑是WAN侧先发送FIN包，之后加速器根据这个FIN包将连线跟踪条目**老化**掉，然后LAN侧回应FIN包。【在环境缺失的情况下，分析浏览器和服务器交互过程，会发现RST包和FIN包都是由浏览器发起】【想到了老化，怎么就没有查看Modem中各种条目的老化时间呢？不熟练，对故障原因想得不够】

1. WAN侧的FIN包能否到协议栈中来？还是由加速器转发？

2. WAN侧收到FIN包的时候，是否有刷新连线跟踪条目的操作？

3. WAN侧收到FIN+ACK和收到FIN效果相同吗？

在modem作限制，连线跟踪不存在，却收到FIN、RST报文，则直接丢弃这种包。

如果故障仍然存在，则：

1. 能否不通过镜像，在上网的PC上抓包？不过滤包。

2. 通过telnet关闭modem的加速器，验证故障是否存在。telnet登录后，cmfctl learn --disable; cmfctl disable

3. 使用xtm中的镜像工具，镜像抓包。 telnet登录后，echo 1 > /proc/driver/xtm/mirror/in; echo 1 > /proc/driver/xtm/mirror/mirror/out

4. 登录telnet，查看连线跟踪的各种信息。

    cat /proc/net/nf_conntrack

    cat /proc/sys/net/netfilter/nf_conntrack_count

基本思路：

1. LAN侧发送RST报文给modem，导致连接进入CLOSE状态，此时加速器不要刷新连线的老化时间。

2. WAN侧发送FIN报文给modem，此时加速器刷新连线跟踪，会删除加速条目，导致WAN侧回应的FIN报文进入到协议栈后透传到WAN侧，也就是private address is leaking to wan

```
#if defined(CONFIG_MIPS_BRCM) && defined(CONFIG_BLOG)
void blog__nf_ct_refresh_acct(struct nf_conn *ct,
    enum ip_conntrack_info ctinfo,
    const struct sk_buff *skb,
    unsigned long extra_jiffies,
    int do_acct)
{
#ifdef CFG_931WII_ATH_Serbia
struct tcphdr *th ;
if(ct->proto.tcp.state == TCP_CONNTRACK_CLOSE)
  return ;

th = skb->h.th ;
if(th && th->fin == 1)
  return ;
#endif
```

__nf_ct_refresh_acct(ct, ctinfo, skb, extra_jiffies, do_acct);

}

EXPORT_SYMBOL_GPL(blog__nf_ct_refresh_acct);

#endif

#if defined(CONFIG_MIPS_BRCM) && defined(CONFIG_BLOG)

blog_refresh_fn = (blog_refresh_t)blog__nf_ct_refresh_acct;

#endif

故障仍然存在！！！

----------10月17日测试----------

在出问题前后，有PPPOE的错包，如果把加速器打开，这样包就不会走协议栈，不会出现错包，是否还有私网地址泄漏？

如果只打开软加速呢？

只打开软加速，QoS功能能否生效？

实际验证，在只打开软加速的情况下，私网地址仍然可以过来。故障可能与PPPoE的错包没有关系。

1. 在ping包的时候，如果连线断开后再重新连上，则会出现私网地址泄漏到公网的问题。当时这个问题可以通过在重新建链后将连线跟踪老化一下。

2. 在出问题的时候，PC有发RST包，Modem收到后转发，SEQ和ACK编号都为1，这个是发出来的包的问题，还是modem修改的？modem修改的话，它在什么地方会修改这两个值？

SEQ和ASEQ为1，是wireshark的显示问题。wireshark显示ack和seq时，都是根据一条连线的第一个包计算出来的相对数值，因为soure为192.168.1.5，wireshark认为是一个新的连接，seq显示没意义。

3. 收到一个RST包，如果先走nf_conntrack_core将包删除，再走NAT，会发生什么事情？连线跟踪和nat的优先级如何？NAT的优先级为NF_IP_PRI_NAT_SRC(100)，confirm的优先级为NF_IP_PRI_CONNTRACK_CONFIRM(MAX)。

tcp_packet会检查包的状态确定是否删除连接。

4. 出错时的包间隔前一个正常的包偶比较长，大于60s

5. 出错时有打印：nf_ct_tcp: invalid new deleting.

1) 在find失败后，打印所有的连线跟踪。

2) 打印删除连线跟踪的信息。

6. 正常的tcp连接建立后，ipv4 2 tcp 6 18 ESTABLISHED，这里的老化时间只有不到30s，但是cat nf_conntrack_tcp_timeout_established却有1800

一个老化时间，有三个数值！！！

ipv4 2 tcp 6 28 ESTABLISHED src=192.168.1.5 dst=90.1.1.3 sport=3855 dport=80 src=90.1.1.3 dst=201.208.2.47 sport=80 dport=3855 [ASSURED] use=1

究竟是谁改了我的老化时间！！

tcp_conntracks: src=90.1.1.3:80 dst=201.208.2.99:4853 syn=0 ack=1 fin=0 rst=0 old=3 new=3

timeout=1800

__nf_ct_refresh_acct, state=3, timeout=1799

__nf_ct_refresh_acct, state=3, timeout=30

三、故障解决

故障与加速器有关系，加速器会将所有TCP连接的老化时间设置为30s。

浏览器在打开网页后，并不立即关闭TCP连接，过大概一分钟之后才发送FIN包。但此时modem中的TCP已经老化掉，没有办法做NAT。

由于我们没有办法修改加速器代码，所以在nf_conntrack_in中，如果连线跟踪创建失败，那么将这个包丢掉不转发，这样就不会泄漏了。

```
    ct = resolve_normal_ct(*pskb, dataoff, pf, protonum, l3proto, l4proto,
                           &set_reply, &ctinfo);
        if (!ct) {
                /* Not valid part of a connection */
                NF_CT_STAT_INC(invalid);
                return NF_DROP;
}
```

另外一种解决方法：修改加速器的TCP连接老化时间和linux内核的一致。

```
void blog__nf_ct_refresh_acct(struct nf_conn *ct,
     enum ip_conntrack_info ctinfo,
     const struct sk_buff *skb,
    unsigned long extra_jiffies,
     int do_acct)
{
  u_int8_t protonum  ;
#ifdef CFG_931WII_ATH_Serbia
struct tcphdr *th ;
if(ct->proto.tcp.state == TCP_CONNTRACK_CLOSE)
  return ;
#endif

protonum = ct->tuplehash[IP_CT_DIR_ORIGINAL].tuple.dst.protonum ;
if(IPPROTO_TCP == protonum && ct->proto.tcp.state == TCP_CONNTRACK_ESTABLISHED)
  extra_jiffies = nf_ct_tcp_timeout_established ;

DEBUGP("blog__nf_ct_refresh_acct, extra_jiffies=%d\n", extra_jiffies/HZ) ;
__nf_ct_refresh_acct(ct, ctinfo, skb, extra_jiffies, do_acct);
}
```

四、故障复盘

1. 为什么没有连线跟踪仍然可以走到NAT中。

在进入的nf_nat_core.c中，如果检查没有连线跟踪，则直接return NF_ACCEPT，并不丢弃包。

2. NF中，返回NF_DROP，何时丢包。

NF_HOOK的执行函数nf_hook_slow中，

```
verdict = nf_iterate(&nf_hooks[pf][hook], pskb, hook, indev,
        outdev, &elem, okfn, hook_thresh);
if (verdict == NF_ACCEPT || verdict == NF_STOP)
{
  ret = 1;
  goto unlock;
}
else if (verdict == NF_DROP)
{
  kfree_skb(*pskb);
  ret = -EPERM;
}
```