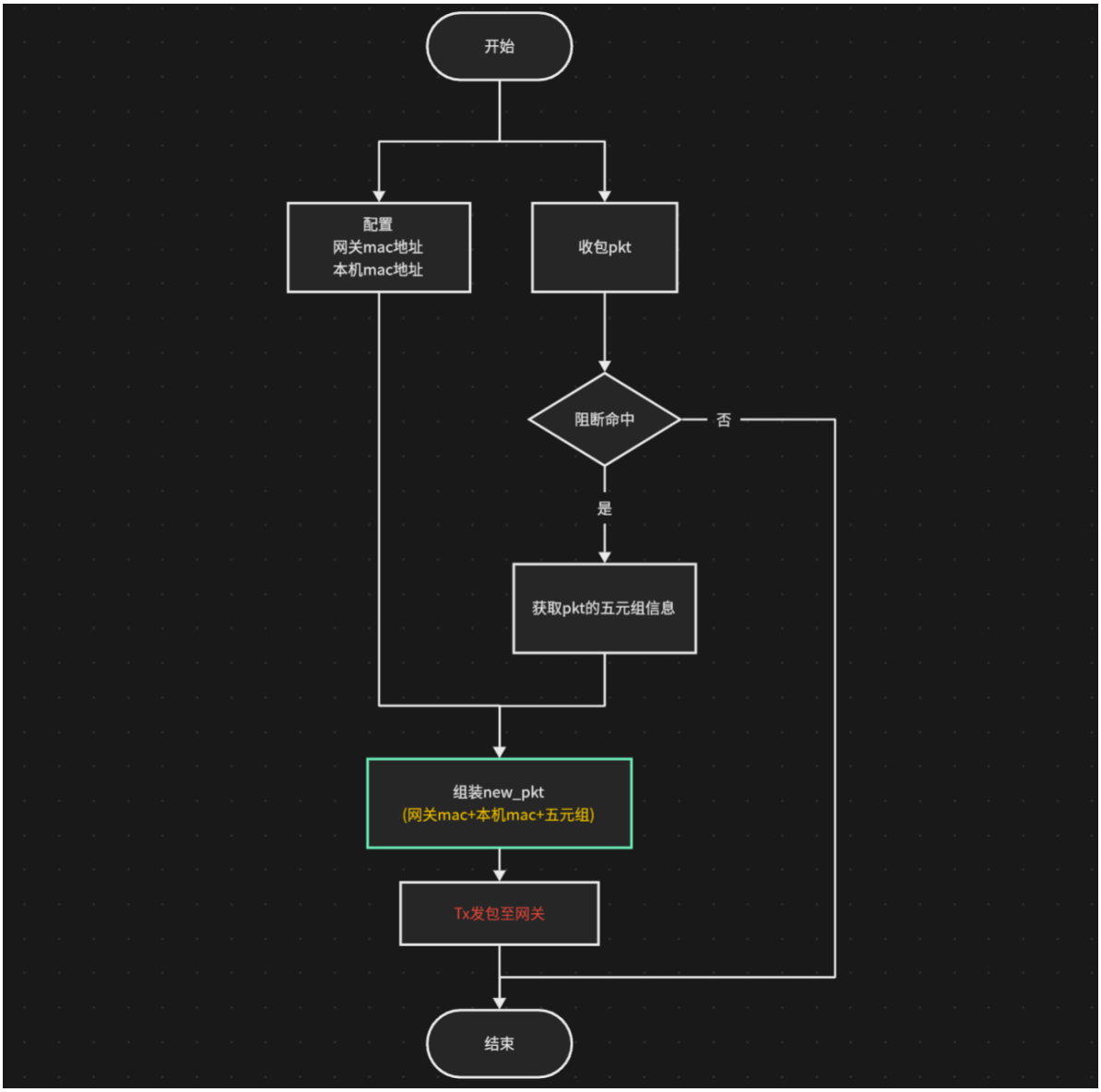
suricata阻断项目本记

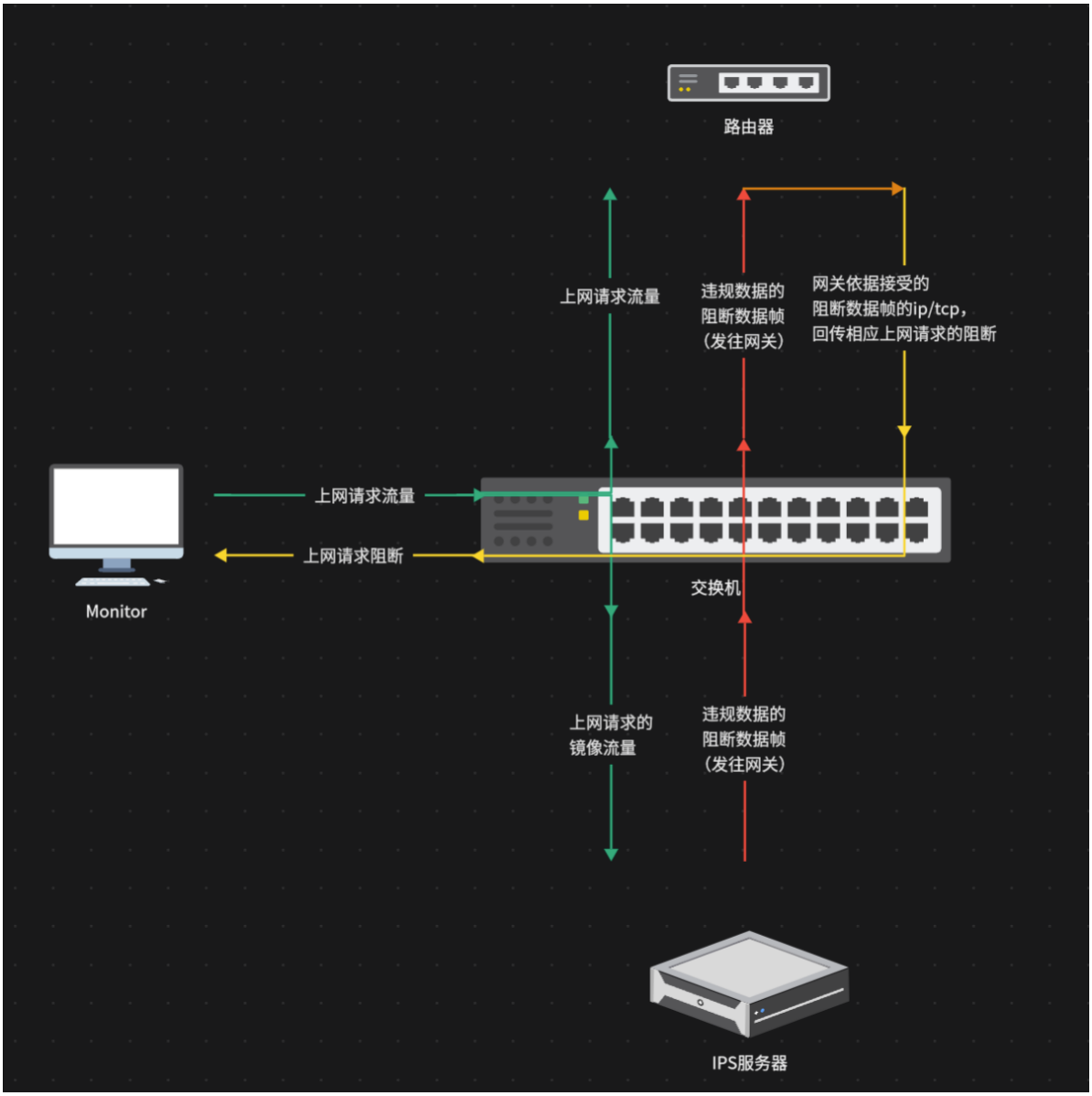
一、阻断逻辑流程



注意：

- **所使用的阻断数据包，需要依据所阻断的pkt数据帧，进行相应的Tcp层seq/ack值的计算；**
- **阻断数据包的mac地址，不使用原数据中两端的mac地址；**

二、架构网络拓扑图



注意：

- **主要的两条数据流：用户上网的请求数据帧、IPS设备的阻断数据帧。**
- **IPS设备的阻断数据帧：先发送网关，再由：网关-->路由器--->终端设备 实现阻断**

三、suricata的阻断使用

再suricata的使用中，要实现阻断的及时相应，有以下两种实践方法：

1.使用单包的tcp阻断规则

```
reject tcp-pkt any any -> any any (msg: "ATTACK [PTsecurity] Spring Core RCE aka Spring4Shell Attempt"; content: "news.cn"; reference: url, github.com/ptresearch/AttackDetection; reference: url, www.cyberkendra.com/2022/03/springshell-rce-0-day-vulnerability.html; classtype: attempted-admin; sid: 10007107; rev: 1;)
```

在suricata中，单包规则就会执行：来一个pkt，就对该包tcp的payload部分进行conten规则匹配，从而进行rst阻断可以更及时。

2.使用流stream的阻断规则

```
reject http any any -> any any (msg: "ATTACK [PTsecurity] Spring Core RCE aka Spring4Shell Attempt"; flow: established, to_server; http.host;content: "news.cn"; reference: url, github.com/ptresearch/AttackDetection; reference: url, www.cyberkendra.com/2022/03/springshell-rce-0-day-vulnerability.html; classtype: attempted-admin; sid: 10007107; rev: 1;)
reject tcp any any -> any any (msg: "ATTACK [PTsecurity] Spring Core RCE aka Spring4Shell Attempt"; content: "news.cn"; reference: url, github.com/ptresearch/AttackDetection; reference: url, www.cyberkendra.com/2022/03/springshell-rce-0-day-vulnerability.html; classtype: attempted-admin; sid: 10007107; rev: 1;)

```

第一步：开启stream处理inline模式：

```
stream:
  memcap: 40gb
  #memcap-policy: ignore
  checksum-validation: no      # reject incorrect csums
  midstream: true
  midstream-policy: auto
  inline: yes                  # auto will use inline mode in IPS mode, yes or no set it statically
  reassembly:
    memcap: 40gb
    #memcap-policy: ignore
    depth: 0                   # reassemble 1mb into a stream
    toserver-chunk-size: 2560
    toclient-chunk-size: 2560
    randomize-chunk-size: yes
    #randomize-chunk-range: 10
    #raw: yes
    #segment-prealloc: 2048
    #check-overlap-different-data: true

# Host table:
#
```

第二步：开启IPS

1、对于程序支持IPS使用，则在相应的网卡配置中指定：ips模式

```
  # IPS mode for Suricata works in 3 modes - none, tap, ips
  # - none: IDS mode only - disables IPS functionality (does not further forward packets)
  # - tap: forwards all packets and generates alerts (omits DROP action) This is not DPDK TAP
  # - ips: the same as tap mode but it also drops packets that are flagged by rules to be dropped
  copy-mode: none
```

2、程序不方便开启IPS模式，可以使用的一种

```
1、命令行指定强行使用IPS(该模式下)
    --simulate-ips
```

注意：

Suricata 的 IPS（入侵防御系统）模式和 simulate-ips 模式之间有几个关键区别：

1. **功能**：
   - **IPS 模式**：在此模式下，Suricata 直接插入网络流量路径中，能够实时检测并阻止恶意流量。它通常配置为在网络接口上接收和转发流量，对流量进行深度包检查，并根据规则做出响应。

- **simulate-ips 模式**：此模式用于测试和调试目的，Suricata 仍然会分析流量，但不会实际阻止或丢弃流量。它可以记录检测到的威胁并生成日志，但不会影响流量流动。

2. **性能**：
   - **IPS 模式**：由于需要实时处理流量并采取行动，因此在性能和延迟上有更高的要求，需要确保网络吞吐量和响应速度。
   - **simulate-ips 模式**：由于不对流量采取实际行动，因此对性能的影响较小，更适合用于开发、调试和测试场景。
3. **应用场景**：
   - **IPS 模式**：适用于生产环境，需要实时防护和威胁检测。
   - **simulate-ips 模式**：适用于测试和验证配置或规则，而不干扰实际流量。

总结来说，IPS 模式用于实时防护和流量控制，而 simulate-ips 模式主要用于测试和调试。

3. 为什么针对stream规则的阻断，就需要IPS模式呢

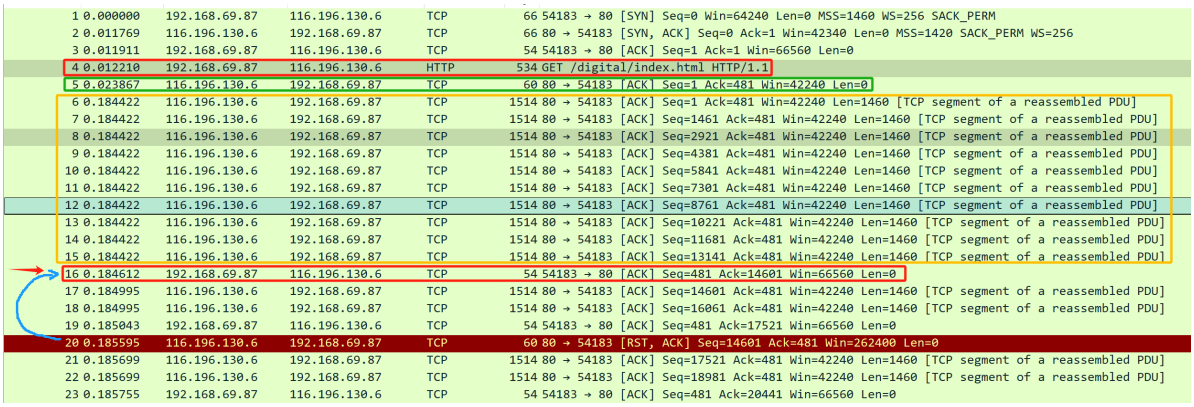
针对这个问题，我们分两种情况，也就是两种规则来讨论：

```
// 具体的应用层协议数据，http层的字段检测处理：
reject http any any -> any any (msg: "ATTACK [PTsecurity] Spring Core RCE aka Spring4Shell Attempt"; flow: established, to_server; http.host;content: "news.cn"; reference: url, github.com/ptresearch/AttackDetection; reference: url, www.cyberkendra.com/2022/03/springshell-rce-0-day-vulnerability.html; classtype: attempted-admin; sid: 10007107; rev: 1;)

// 不涉及应用层，进行tcp的流数据检测
reject tcp any any -> any any (msg: "ATTACK [PTsecurity] Spring Core RCE aka Spring4Shell Attempt"; content: "news.cn"; reference: url, github.com/ptresearch/AttackDetection; reference: url, www.cyberkendra.com/2022/03/springshell-rce-0-day-vulnerability.html; classtype: attempted-admin; sid: 10007107; rev: 1;)

```

我们查看：szjj-reset-stream40.pcapng

```
STREAMTCP_STREAM_FLAG_DEPTH_REACHED: // 达到了还原深度
  含义：当流的重组深度达到预设的最大值时，这个标志会被设置。这意味着流中已经重组的包达到了配置的限制，任何进一步的包将被忽略。
  用途：此状态可以防止内存使用过多或处理开销过大，确保系统能够在合理的资源范围内运行。它通常在处理大流量或高并发连接时非常重要。

STREAMTCP_STREAM_FLAG_TRIGGER_RAW: // 应用层解析过程中，发现达到了处理深度
  含义：这个标志用于指示下次需要进行“原始raw”数据处理时，触发重组。它表示流在当前阶段没有足够的信息来进行完整重组，但在下一次stream的数据到来时，就可以开始对数据进行原始处理。
  用途：此状态通常用于优化数据处理流程。当流的某些条件满足时（如接收到特定类型的数据包），系统会准备重新开始重组或处理数据。
```

包详情如下：



1	0.000000	192.168.69.87	116.196.130.6	TCP	66 54183 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
2	0.011769	116.196.130.6	192.168.69.87	TCP	66 80 → 54183 [SYN, ACK] Seq=0 Ack=1 Win=42340 Len=0 MSS=1420 SACK_PERM WS=256
3	0.011911	192.168.69.87	116.196.130.6	TCP	54 54183 → 80 [ACK] Seq=1 Ack=1 Win=66560 Len=0
4	0.012210	192.168.69.87	116.196.130.6	HTTP	534 GET /digital/index.html HTTP/1.1
5	0.023867	116.196.130.6	192.168.69.87	TCP	60 80 → 54183 [ACK] Seq=1 Ack=481 Win=42240 Len=0
6	0.184422	116.196.130.6	192.168.69.87	TCP	1514 80 → 54183 [ACK] Seq=1 Ack=481 Win=42240 Len=1460 [TCP segment of a reassembled PDU]
7	0.184422	116.196.130.6	192.168.69.87	TCP	1514 80 → 54183 [ACK] Seq=1461 Ack=481 Win=42240 Len=1460 [TCP segment of a reassembled PDU]
8	0.184422	116.196.130.6	192.168.69.87	TCP	1514 80 → 54183 [ACK] Seq=2921 Ack=481 Win=42240 Len=1460 [TCP segment of a reassembled PDU]
9	0.184422	116.196.130.6	192.168.69.87	TCP	1514 80 → 54183 [ACK] Seq=4381 Ack=481 Win=42240 Len=1460 [TCP segment of a reassembled PDU]
10	0.184422	116.196.130.6	192.168.69.87	TCP	1514 80 → 54183 [ACK] Seq=5841 Ack=481 Win=42240 Len=1460 [TCP segment of a reassembled PDU]
11	0.184422	116.196.130.6	192.168.69.87	TCP	1514 80 → 54183 [ACK] Seq=7301 Ack=481 Win=42240 Len=1460 [TCP segment of a reassembled PDU]
12	0.184422	116.196.130.6	192.168.69.87	TCP	1514 80 → 54183 [ACK] Seq=8761 Ack=481 Win=42240 Len=1460 [TCP segment of a reassembled PDU]
13	0.184422	116.196.130.6	192.168.69.87	TCP	1514 80 → 54183 [ACK] Seq=10221 Ack=481 Win=42240 Len=1460 [TCP segment of a reassembled PDU]
14	0.184422	116.196.130.6	192.168.69.87	TCP	1514 80 → 54183 [ACK] Seq=11681 Ack=481 Win=42240 Len=1460 [TCP segment of a reassembled PDU]
15	0.184422	116.196.130.6	192.168.69.87	TCP	1514 80 → 54183 [ACK] Seq=13141 Ack=481 Win=42240 Len=1460 [TCP segment of a reassembled PDU]
16	0.184612	192.168.69.87	116.196.130.6	TCP	54 54183 → 80 [ACK] Seq=481 Ack=14601 Win=66560 Len=0
17	0.184995	116.196.130.6	192.168.69.87	TCP	1514 80 → 54183 [ACK] Seq=14601 Ack=481 Win=42240 Len=1460 [TCP segment of a reassembled PDU]
18	0.184995	116.196.130.6	192.168.69.87	TCP	1514 80 → 54183 [ACK] Seq=16061 Ack=481 Win=42240 Len=1460 [TCP segment of a reassembled PDU]
19	0.185043	192.168.69.87	116.196.130.6	TCP	54 54183 → 80 [ACK] Seq=481 Ack=17521 Win=66560 Len=0
20	0.185595	116.196.130.6	192.168.69.87	TCP	60 80 → 54183 [RST, ACK] Seq=14601 Ack=481 Win=262400 Len=0
21	0.185699	116.196.130.6	192.168.69.87	TCP	1514 80 → 54183 [ACK] Seq=17521 Ack=481 Win=42240 Len=1460 [TCP segment of a reassembled PDU]
22	0.185699	116.196.130.6	192.168.69.87	TCP	1514 80 → 54183 [ACK] Seq=18981 Ack=481 Win=42240 Len=1460 [TCP segment of a reassembled PDU]
23	0.185755	192.168.69.87	116.196.130.6	TCP	54 54183 → 80 [ACK] Seq=481 Ack=20441 Win=66560 Len=0

第四帧数据包是我们所期待的命中数据包；下面，我用正常IDS模式下，对于TCP流规则命中的逻辑进行一个说明：

```
#下面我们用序列号表示对应数据帧
1-3  ：tcp连接信令数据帧；
4    ：开始进行协议检测,所以当时在detect的hs检测函数中打断点也可以走到，因为协议检测里面也会用到相应的匹配函数；
5    ：对应第四帧的反向ack数据帧到来，开始走相关的"对向(87-->6)stream"协议数据(也就是我们的'第四帧数据')解析，也就是第四帧的http协议数据解析；并且解析过程中，http调用了AppLayerParserTriggerRawStreamReassembly 函数，用来设置STREAMTCP_STREAM_FLAG_TRIGGER_RAW标志，以告诉"对向(87-->6)stream"下一次可以使用该流向的还原数据进行相关处理
6-15 ：同向ip(6-->87)的一个tcp重组还原；
16   ：对向(87-->6)stream的新数据帧到来，并且因为之前为该向stream设置的STREAMTCP_STREAM_FLAG_TRIGGER_RAW标志，满足了detect条件，此时suricata才开始进行raw原始流数据的规则检测，然后命中；
// 最后：我们依据16帧数据创造rst数据包，开始进行tcp阻断。
```

所以我们一共有三个问题：

1、tcp的规则关键字使用什么类型匹配函数；

使用预过滤PrefilterPktStream函数先过滤出"适用规则"，StreamMpmFunc即是适用流多模匹配函数；

```
    // 需要满足存在可用于检测的stream数据
    if (p->flags & PKT_DETECT_HAS_STREAMDATA) {
        struct StreamMpmData stream_mpm_data = { det_ctx, mpm_ctx };
        StreamReassembleRaw(p->flow->protoctx, p,
                StreamMpmFunc, &stream_mpm_data,
                &det_ctx->raw_stream_progress,
                false /* mpm doesn't use min inspect depth */);
    }
```

StreamReassembleRaw函数：

```
    TcpStream *stream;
    // 取的都是同向stream数据
    if (PKT_IS_TOSERVER(p)) {
        stream = &ssn->client;
    } else {
        stream = &ssn->server;
    }

```

```
    if ((stream->flags & (STREAMTCP_STREAM_FLAG_NOREASSEMBLY|STREAMTCP_STREAM_FLAG_DISABLE_RAW)) ||
        // 判断是否满足的检测深度
        StreamTcpReassembleRawCheckLimit(ssn, stream, p) == 0)
    {
        *progress_out = STREAM_RAW_PROGRESS(stream);
        return 0;
    }
```

2、这类匹配函数满足什么条件才开始匹配；

尤其对于流的匹配：PKT_DETECT_HAS_STREAMDATA 必须满足

```

DetectFlow
 DetectRun
  DetectRunSetup // 流类规则的检测，开始进行相关前置条件的设置

       if ((p->proto == IPPROTO_TCP && (p->flags & PKT_STREAM_EST)) ||
                (p->proto == IPPROTO_UDP) ||
                (p->proto == IPPROTO_SCTP && (p->flowflags & FLOW_PKT_ESTABLISHED)))
        {
            flow_flags = FlowGetDisruptionFlags(pflow, flow_flags);
            alproto = FlowGetAppProtocol(pflow);
            if (p->proto == IPPROTO_TCP && pflow->protoctx && // protoctx其实就是对应方向的stream
                    StreamReassembleRawHasDataReady(pflow->protoctx, p)) { // 下面详解
                p->flags |= PKT_DETECT_HAS_STREAMDATA;
            }
            SCLogDebug("alproto %u", alproto);
        }
```

StreamReassembleRawHasDataReady 函数：

```
    // 非inline模式(IDS)下，需要进行各类的限制判断
    if (StreamTcpInlineMode() == FALSE) {
        const uint64_t segs_re_abs =
                STREAM_BASE_OFFSET(stream) + stream->segs_right_edge - stream->base_seq;
        if (STREAM_RAW_PROGRESS(stream) == segs_re_abs) { // 如果不存在新的数据
            return false;
        }
        // 存在新的数据，开始进行一些限制判断
        if (StreamTcpReassembleRawCheckLimit(ssn, stream, p) == 1) {
            return true;
        }
    } else {
    // IPS的inline模式，则只需要满足是新数据帧存在即可
        if (p->payload_len > 0 && (p->flags & PKT_STREAM_ADD)) {
            return true;
        }
```

```
    }
```

3、这些条件需要在什么情况下达成；

StreamTcpReassembleRawCheckLimit判断：**以下只显示此次数据包所使用的判断部分**

```
#define STREAMTCP_STREAM_FLAG_FLUSH_FLAGS       \
        (   STREAMTCP_STREAM_FLAG_DEPTH_REACHED \
        |   STREAMTCP_STREAM_FLAG_TRIGGER_RAW   \
        |   STREAMTCP_STREAM_FLAG_NEW_RAW_DISABLED)

    if (stream->flags & STREAMTCP_STREAM_FLAG_FLUSH_FLAGS) {
        if (stream->flags & STREAMTCP_STREAM_FLAG_DEPTH_REACHED) { // 满足最大存储深度，开始检测；
            printf("                DEPTH_REACHED\n");
        }
        if (stream->flags & STREAMTCP_STREAM_FLAG_TRIGGER_RAW) {  // 多为应用层协议解析设置，表示满足检测条件
            printf("                TRIGGER_RAW\n");
        }
        if (stream->flags & STREAMTCP_STREAM_FLAG_NEW_RAW_DISABLED) { // 禁止对新段进行原始重组，即每次进来新数据，都进行检测
            printf("                RAW_DISABLED\n");
        }
        SCReturnInt(1);
    }
#undef STREAMTCP_STREAM_FLAG_FLUSH_FLAGS
```

比如我们次次遇到的，就是STREAMTCP_STREAM_FLAG_TRIGGER_RAW的设置，主要由上层协议解析时进行 AppLayerParserTriggerRawStreamReassembly 调用处理。

4、http的url封堵

4.1 确认网关地址

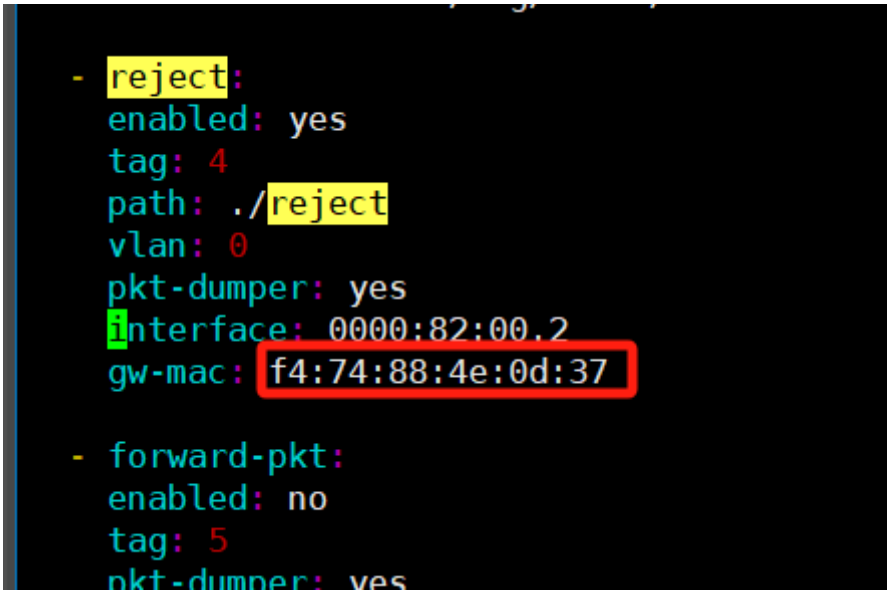
```
route -n
```

```
[root@localhost pcap]# route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         192.168.69.1    0.0.0.0         UG    100    0        0 em3
172.17.0.0      0.0.0.0         255.255.0.0     U     0      0        0 docker0
192.168.69.0    0.0.0.0         255.255.255.0   U     100    0        0 em3
[root@localhost pcap]#
```

```
arp -n
```

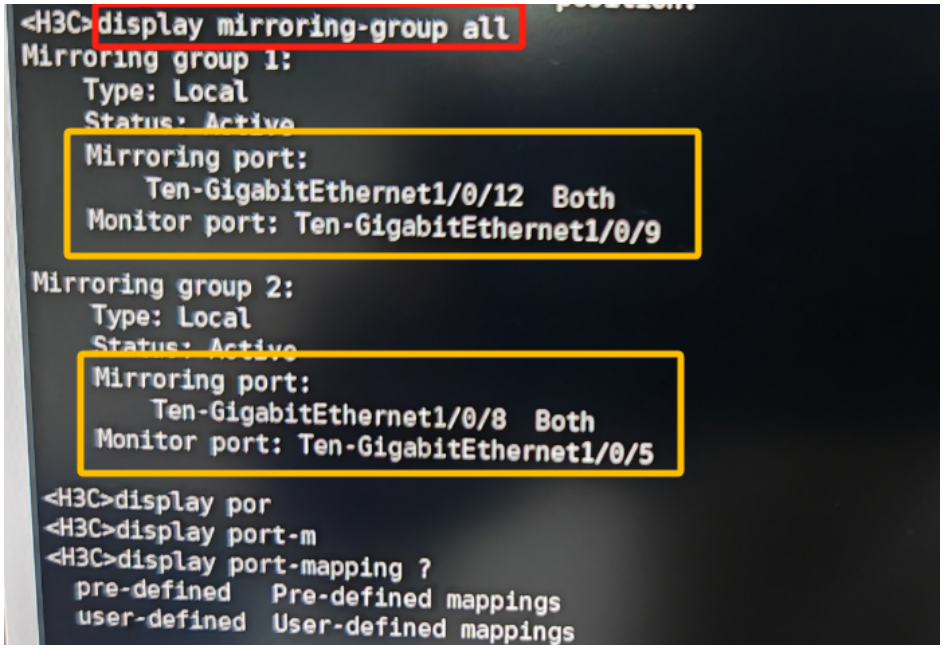
```
[root@localhost pcap]# arp -n
Address                  HWtype  HWaddress           Flags Mask            Iface
192.168.69.79            ether   16:ca:ee:b4:7b:a4   C                     em3
192.168.69.1             ether   f4:74:88:4e:0d:37   C                     em3
192.168.69.94            ether   32:61:45:f7:16:d7   C                     em3
192.168.69.78            ether   2a:be:66:a4:d1:e1   C                     em3
192.168.69.62            ether   20:57:9e:be:3c:b9   C                     em3
192.168.69.77            ether   8c:7a:3d:0a:7f:7a   C                     em3
192.168.69.30            ether   14:84:77:e1:85:b0   C                     em3
192.168.69.92            ether   62:59:32:9a:00:a1   C                     em3
192.168.69.161           ether   00:71:cc:96:2c:cb   C                     em3
192.168.69.76            ether   72:b0:fc:ab:5b:a1   C                     em3
192.168.69.68            ether   8a:f1:a2:99:43:ed   C                     em3
192.168.69.37            ether   ec:f4:bb:e6:a9:c4   C                     em3
192.168.69.91            ether   3e:e3:a2:88:51:c2   C                     em3
192.168.69.75            ether   3e:e3:a2:88:51:c2   C                     em3
192.168.69.67            ether   96:e9:22:76:60:58   C                     em3
192.168.69.82            ether   d4:d8:53:e9:ca:e1   C                     em3
192.168.69.74            ether   9c:da:3e:90:ce:19   C                     em3
192.168.69.97            ether   66:7a:ec:14:2d:4b   C                     em3
192.168.69.35            ether   30:d0:42:f5:bb:d9   C                     em3
192.168.69.96            ether   0e:91:da:a5:6e:d6   C                     em3
[root@localhost pcap]#
```

由此确认对应的网关mac地址，配置在yaml文件中：



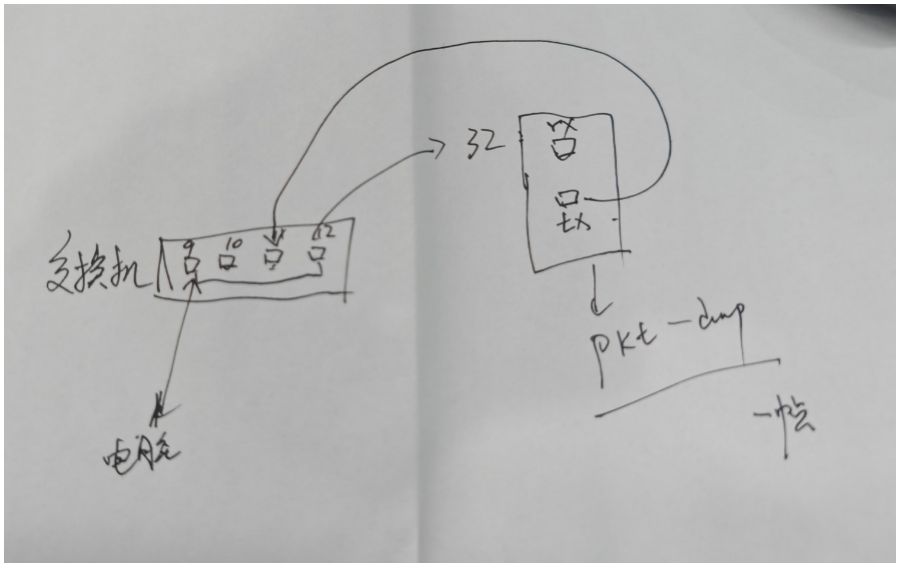
4.2 查看路由镜像拓扑，配置布线

交换机镜像拓扑：



其中我们使用group1中的配置：<12--9>，其中12口是被镜像口，连接我们的访问网络的机器，9口是镜像流量出来的口，连接我们的32中suricata-IDS收包口；

画出完整的拓扑图，并由此连线(tx口，连接到交换机空闲的11口即可)：



然后再在"reject"模块中，配置tx发包(302)的口，记得在eal的参数allow中，将rx/tx网卡的pci地址都加上，免得被屏蔽。

4.3 程序启动

记得一定要走IPS模式

```
./suricata -c ./cfg/suricata.yaml  -l ./log/ --dpdk --simulate-ips
```

之后访问新华网时，就会被baidu.com替换；

```
# 具体逻辑可以查看：
    suricata\src\message\message-reject.c
    suricata\src\block-detect\block-url.c
```


hao123_上网从这里开始
百度一下，你就知道
hao123.com
高爆打金
自动吸怪 0冲服
查看更多
hao123
设为首页
北京 切换
七日天气
今晴 良
明晴 14～24℃
10月11日 九月初九
星期五
星座运势
查IP
邮箱
Bai du 百度
网页
空客“超级大白鲸”运输机现身天津
男子不爱社交长期不说话致语言退化 热
日本首相石破茂不参加靖国神社大祭
郑钦文vs保利尼 冲击武网4强 热
俄方：五个核武器国家将在纽约开会
hao123推荐
人民网
新华网
央视网
国际在线
中国日报
中国网
中经网
光明网
百度
新浪·微博
搜狐·热点
腾讯
网易
百度地图
爱淘宝
京东
阿里1688
天猫国际
携程旅行网
好看视频
聚划算
天猫
东方财富
知乎
文心一言
12306