Snort包处理流程

Snort的主函数是SnortMain（snort.c），前面一大堆都是准备过程，不详述，包处理过程从最后一句InterfaceThread(NULL)开始讲起。希望通过这篇报告能够使大家对snort包处理流程清晰一些。

注意：强烈推荐大家用source insight看snort源码，会使大家事半功倍。另外，下面的内容参照“流程图.vsd”,下面的所有内容都已经记录到了vsd文件中，双击任何一个函数就能看到。

SnortMain（） //属于文件Snort.c

{

准备过程一大堆…………

//下面这个函数就是开始包处理流程了

InterfaceThread(NULL);

}

void \*InterfaceThread(void \*arg) //属于文件Snort.c

{

1. 开始计时，记录一下抓包的开始时间
2. 调用pcap\_loop( pd, pv.pkt\_cnt, (pcap\_handler) ProcessPacket, NULL)

这个函数的作用是不断抓包，然后将抓到的包交给回调函数ProcessPacket处理

1. 记录一下抓包的结束时间，然后减去开始时间得出抓包的总时间。
2. 调用CleanExit（），这个函数还是比较复杂的，因为他内部又调用了其他函数，也用到了很多结构体，但是只是打印一下统计信息，对性能不会有太大影响

}

前面讲到抓到的包将会全部交给ProcessPacket处理，因为这个函数还是比较重要的

void ProcessPacket(char \*user, struct pcap\_pkthdr \* pkthdr, u\_char \* pkt) //来自Snort.c

{

初始化一些东西（略）

/\*调用解码引擎，将数据包拆分后，放在p当中\*/

(\*grinder) (&p, pkthdr, pkt);

*/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

\*\*刚开始时这个grinder找不到，后来发现其实是一个定义在Snort.h中的函数指针：

\*\*定义如下：

\*\*typedef void (\*grinder\_t)(Packet \*, struct pcap\_pkthdr \*, u\_char \*);

\*\*extern grinder\_t grinder;

\*\*原始定义在snort.c中 grinder\_t grinder;

\*\*流程是（snort.c中）

\*\*SnortMain-> SetPktProcessor，此处根据datalink的不同grinder会指向不同的函数

\*\*见下SetPktProcessor的分析

\*\*（此处假设我们处理的是以太网的协议，grinder = DecodeEthPkt）

\* \*grinder分析见下

*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*

判断运行模式

switch(runMode)

{

case MODE\_PACKET\_LOG: //包记录模式

CallLogPlugins(&p, NULL, NULL, NULL);

break;

case MODE\_IDS: //IDS模式

/\* start calling the detection processes \*/

Preprocess(&p); /\*下面我要重点介绍一下预处理\*/

break;

default:

break;

}

最后是清理

}

int SetPktProcessor() //设置解码引擎

{

1. 判断datalink值

（注意datalink的赋值在snort.c：SnortMain->OpenPcap(): datalink = pcap\_datalink(pd); ）

1. 例如：

如果case DLT\_EN10MB: /\* 即是以太网Ethernet \*/

grinder = DecodeEthPkt;

如果case DLT\_PPP: /\*PPP协议\*/

grinder = DecodePppPkt;

其他的以此类推，不详述

}

grinder = DecodeEthPkt /\*我们假设处理的是以太网的协议\*/

void DecodeEthPkt(Packet \* p, struct pcap\_pkthdr \* pkthdr, u\_int8\_t \* pkt) //隶属于Decode.c

{

这个函数还是比较简单的，顾名思义，就是将包拆解开，先判断以太网包头的ether\_type，即下层协议类型，举例如下：

switch(ntohs(p->eh->ether\_type))

case ETHERNET\_TYPE\_IP： /\*（0800）假设为IP协议，则调用DecodeIP \*/

DecodeIP(p->pkt + ETHERNET\_HEADER\_LEN,

cap\_len - ETHERNET\_HEADER\_LEN, p);

解码的过程继续往下，过程类似，进入DecodeIP（Decode.c），如果判断下层为TCP，调用DecodeTCP（Decode.c），判断为UDP则调用DecodeUDP（Decode.c），等等，报警信息也将在这里面输出，

例如在DecodeIP中，若IP头部长度异常，检测到之后会通过

if((runMode == MODE\_IDS) && pv.decoder\_flags.decode\_alerts) {

SnortEventqAdd(GENERATOR\_SNORT\_DECODE, DECODE\_IPV4\_DGRAM\_LT\_IPHDR,

1, DECODE\_CLASS, 3, DECODE\_IPV4\_DGRAM\_LT\_IPHDR\_STR, 0);

}

这段代码将输出一个警告信息："(snort\_decoder) WARNING: IP dgm len < IP Hdr len!"

（错误信息见Generators.h）

其他过程都类似了，不详述。

}

int Preprocess(Packet \* p) /\*预处理，来自Detect.c\*/

{

1. 预处理函数早已经以链表的形式初始化完成，预处理的第一步即调用这些函数一一对数据包进行处理

while(idx != NULL){

idx->func(p); //调用预处理函数

idx = idx->next; //取下一个预处理函数

}

1. Detect(p); //检测数据包，来自Detect.c

}

int Detect(Packet \* p) //检测函数,这个函数貌似也挺重要的，可是我没看懂！！

{

1. 调用fpEvalPacket
2. CallLogFuncs(p, "Tagged Packet", NULL, &event);

}

int fpEvalPacket(Packet \*p) //Fpdetect.c

{

//假设是一个TCP 数据包

Return fpEvalHeaderTcp(p);

}

static INLINE int fpEvalHeaderTcp(Packet \*p)

{

//分析不下去了，累死我了。。。先停在这里吧。}