# Snort在Ubuntu上的编译安装

编译Snort之前，安装好所需的第三方函数库。

Snort有四个主要的必须的第三方函数库：

* pcap (libpcap-dev) 从Ubuntu仓库安装
* PCRE (libpcre3-dev) 从Ubuntu仓库安装
* Libdnet (libdumbnet-dev) 从Ubuntu仓库安装
* DAQ (<http://www.snort.org/downloads/>) 从源码编译

编译工具的安装：

sudo apt-get install -y build-essential

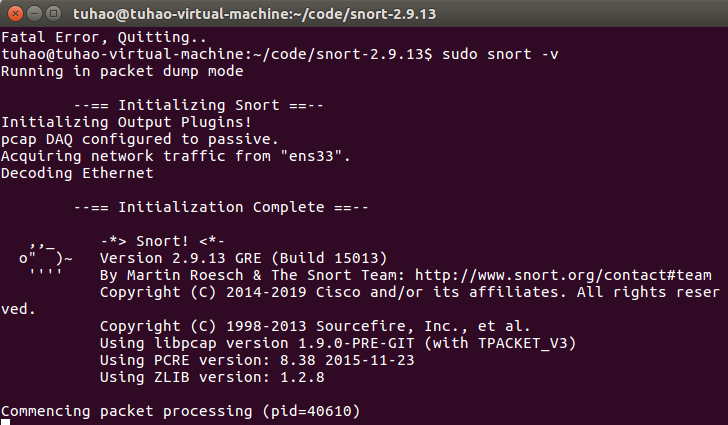
第三方函数库的安装：

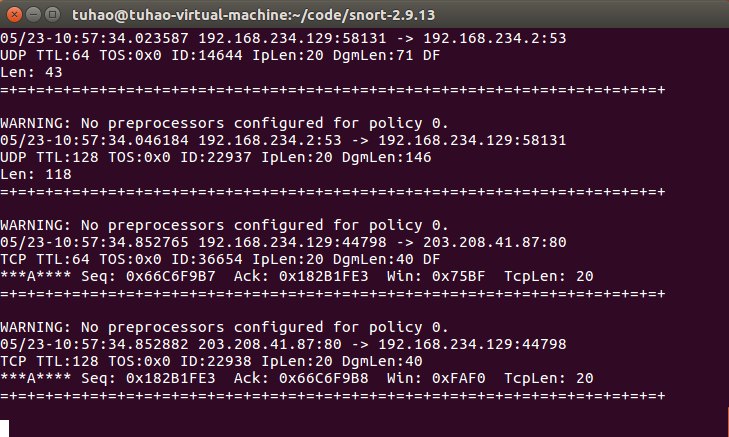
sudo apt-get install -y libpcap-dev libpcre3-dev libdumbnet-dev

sudo apt-get install -y bison flex （DAQ编译安装需要提前安装的库）

安装成功后：

/usr/local/bin/snort snort可执行文件存在目录





安装完Snort之后的配置工作：

创建存放配置文件的文件夹，修改用户权限

从Snort源文件拷贝配置文件和静态预处理器至/etc/snort

拷贝规则文件至/ect/snort/rules

编辑配置文件/etc/snort/snort.conf (对于NIDS模式至关重要)

windows和Linux的路径处理:

sudo sed -i "s/include \$RULE\\_PATH/#include \$RULE\\_PATH/" /etc/snort/snort.conf

允许使用本地规则文件

include $RULE\_PATH/local.rules

测试是否配置好Snort

sudo snort -T -i ens33 -c /etc/snort/snort.conf

**/etc/snort目录结构**

.

├── attribute\_table.dtd

├── classification.config

├── file\_magic.conf

├── gen-msg.map

├── preproc\_rules

├── reference.config

├── rules (规则目录，详细规则略)

│   ├── iplists

│   ├──black\_list.rules

│    └── white\_list.rules

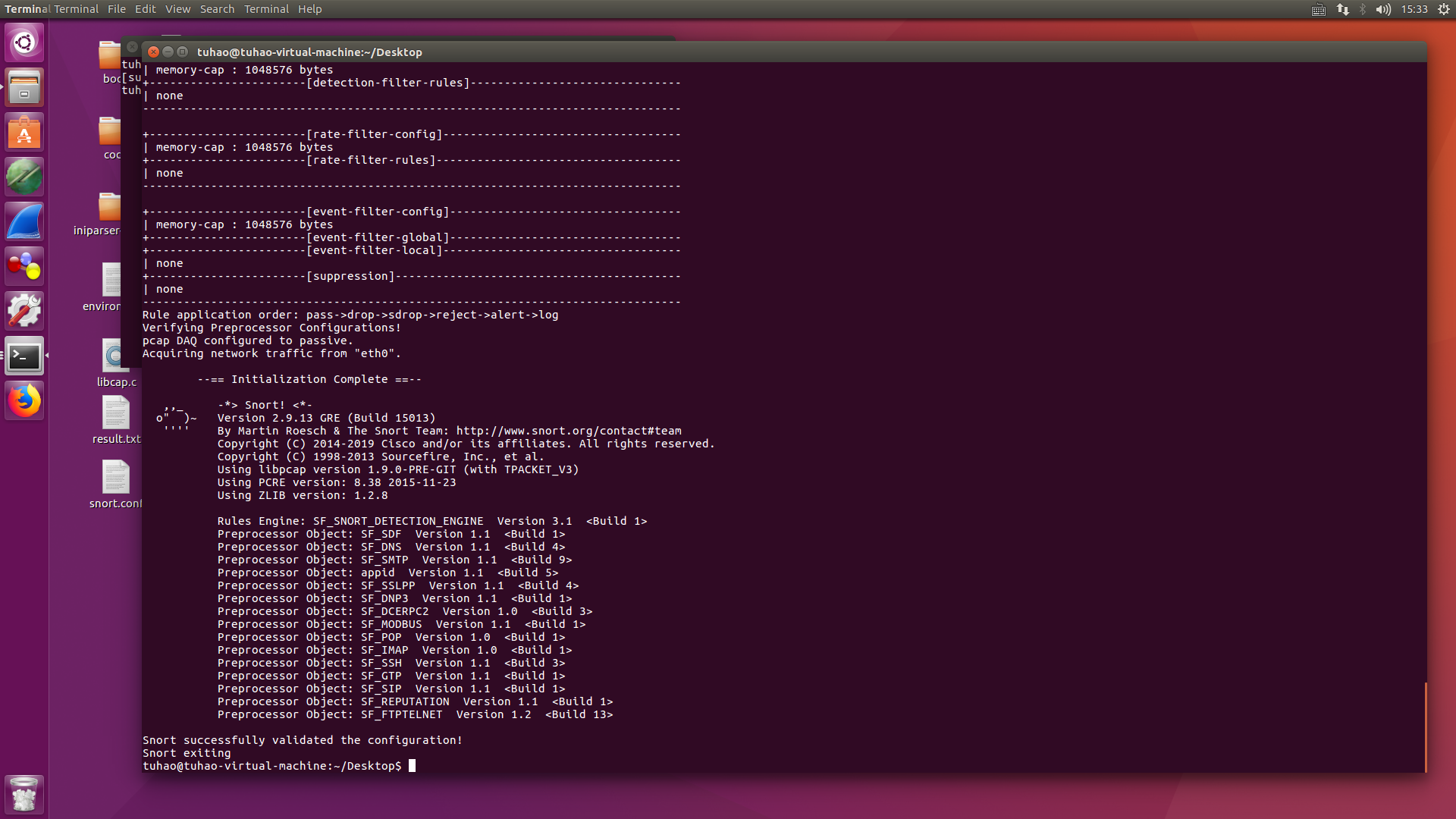
├── sid-msg.map

├── snort.conf

├── so\_rules

├── threshold.conf

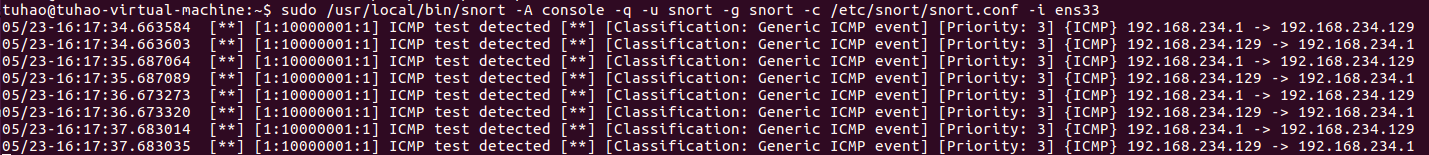
└── unicode.map



**编写简单规则**：/etc/snort/rules/local.rules

alert icmp any any -> $HOME\_NET any (msg:"ICMP test detected"; GID:1; sid:10000001; rev:001; classtype:icmp-event;)

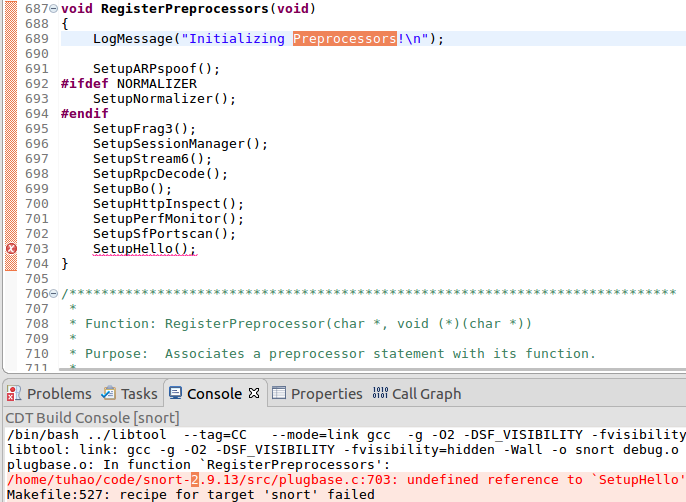
在snort.conf要包含local.rules使之生效

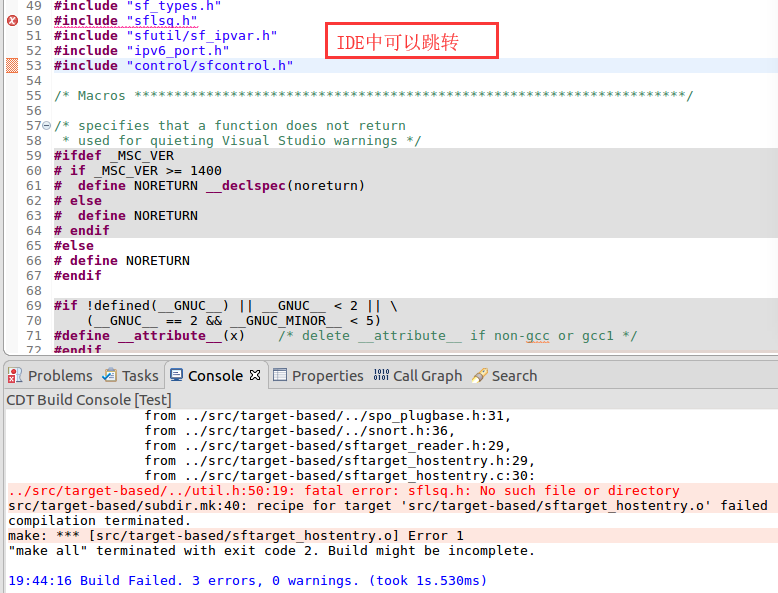


<https://blog.csdn.net/tao546377318/article/details/52047085?locationNum=12&fps=1>

IDE开发工具安装

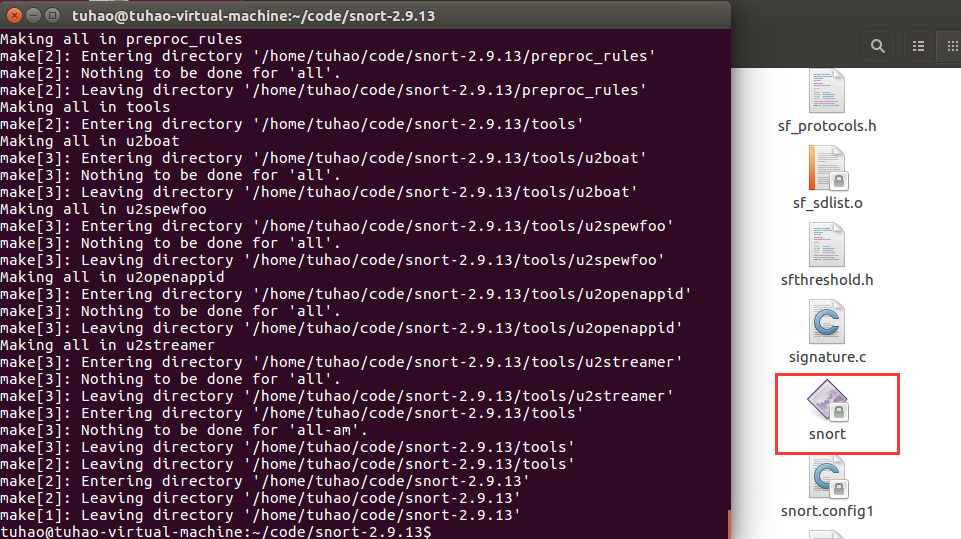
Qt，Code Blocks，Eclipse for C/C++



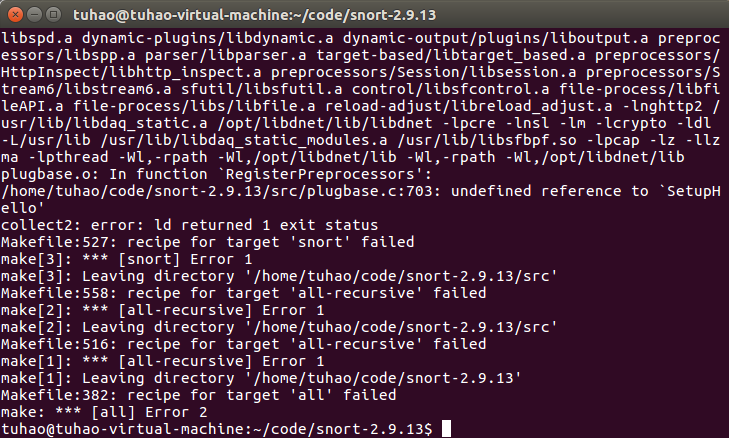


GCC

编译生成可执行文件



需要修改Makefile



# 简单预处理器的编写

参考的文献：

<https://wenku.baidu.com/view/bd9d309f26fff705cd170a08.html>

<http://www.cppblog.com/jerryma/archive/2012/05/22/175730.html>

Qt中导入Snort项目

<https://blog.csdn.net/u013511711/article/details/80600063>

## 官方文档

位置：/snort2.9.13/doc

Snort 1.5版引入了一个主要的新概念插件。有两种类型Snort目前可用的插件：检测插件和预处理器。检测插件检查数据包的单个方面以查找其中定义的值规则并确定分组数据是否符合其验收标准。对于例如，tcp标志检测插件检查TCP数据包的标志部分用于与特定规则中定义的标志组合的匹配。发现每个包具有不同的参数可以多次调用插件。

预处理器每个数据包只被称为一次，并且可能表现很高复杂的功能，如TCP流重组，IP碎片整理或HTTP请求规范化。他们可以直接操作分组数据，甚至可以调用检测引擎直接用他们修改过的数据。他们的表现可以减少复杂的任务，如统计数据收集或阈值监控。

以用户身份添加新插件到Snort：

现在，为Snort添加一个新插件很简单但需要你手动编辑两个文件。该插件应包含两个文件，

“sp\_something.c”/“sp\_something.h”用于检测插件，和“spp\_something.c”/“spp\_something.h”用于预处理程序。对于检测插件，将它与Snort集成有两个步骤：

添加预处理器同样简单。这个过程基本上就是与上述相同：

1）编辑plugbase.h并插入该行 #include“spp\_preproc.h”使用其他“#include”语句进入文件。保存并关闭文件。或者插入plugbase.c中

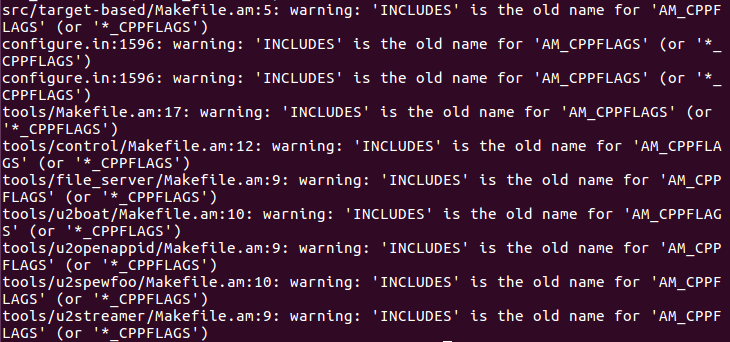
2）编辑plugbase.c文件并在InitPreprocessors（）函数中添加具有其他设置功能的列表的设置功能的名称。保存并关闭文件。

3）编辑Makefile.am并将两个文件的名称添加到列表中 “snort\_SOURCES”行上的名称。保存并退出该文件。跑 “automake”。

编写新的Snort插件作为开发人员：

这个过程也非常简单，也是寻找的最佳场所有关执行这些操作的信息位于“templates”目录中的文件中。sp\_ \*文件是为检测插件设置的，spp\_ \*文件是指预处理器文件。一旦你写完它就要记住的主要事情是将正确的包含和函数调用放入plugbase [.c | .h]。

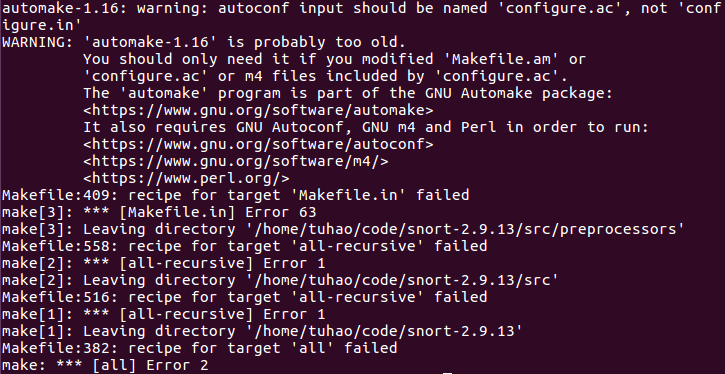
运行 automake 会出现警告



再运行make：编译失败

但是1.16和1.16.1是最新的两个版本

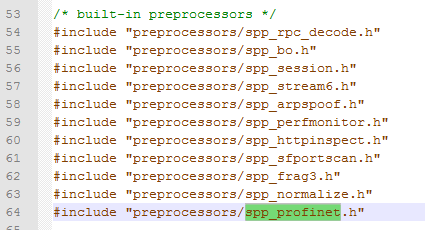
换1.15，1.13和1.11会发生版本不匹配的错误。



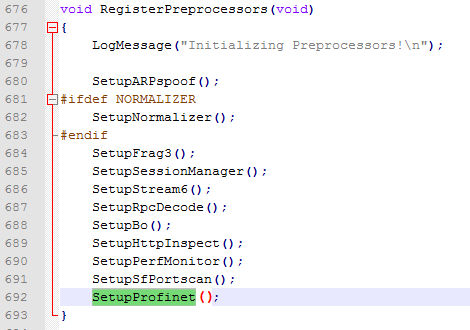
## **网上参考**

使用Snort版本为2.9.8.2

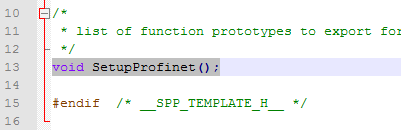
1. 复制snort-2.9.8.2/templates/spp\_template.c snort-2.9.8.2/templates/spp\_template.h到snort-2.9.8.2/src/preprocessors，重命名为spp\_profinet.c spp\_profinet.h （名字任意）
2. 修改snort-2.9.8.2/src/plugbase.c 文件，将插件的头文件spp\_profinet.h 包含到plugbase.c 中如图64行



1. 将插件的 SetupProfinet()函数插入plugbase.c 的 RegisterPreprocessors()函数中



1. 修改spp\_profinet.h文件，定义初始化函数原型void SetupProfinet();



1. 修改spp\_profinet.c文件
2. 修改引用的头文件 #inlcude “spp\_template.h”改为 #inlcude “spp\_profinet.h”
3. SetupProfinet()函数即为模板中的函数SetupTemplate()，用来调用注册预处理器函数RegisterPreprocessor，修改为

void SetupProfinet()

{

#ifndef SNORT\_RELOAD

RegisterPreprocessor("Profinet\_RT", ProfinetRTInit);

#else

RegisterPreprocessor("Profinet\_RT", ProfinetRTInit, ProfinetRTReloadFuction,NULL, NULL, NULL);

#endif

}

Profinet\_RT为预处理器名称

ProfinetRTReloadFuction这个函数可以暂时为NULL，本例中也只是声明了这个函数，并没有使用。

1. ProfinetRTInit()函数即为模板中TemplateInit()函数。用来初始化预处理器，调用AddFuncToPreprocList把此预处理器加入预处理器列表。

static void ProfinetRTInit(struct \_SnortConfig \*sc,u\_char \*args)

{

AddFuncToPreprocList(sc, ProfinetRTFunct,0x01, PP\_Profinet\_RT, PROTO\_BIT\_\_PROFINET);

printf("ProfinetRTInit is setup\n\n");

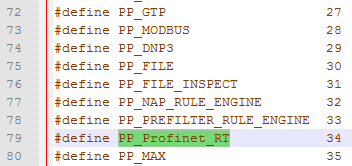
RegisterPreprocStats("Profinet\_RT", ProfinetPrintStats);

//AddFuncToCleanExitList(PreprocCleanExitFunction, NULL);

//AddFuncToRestartList(PreprocRestartFunction, NULL);

}

* ProfinetRTFunct为预处理器执行函数 0x01为预处理器优先级
* PP\_Profinet\_RT需在源文件 /src/preprocids.h中定义的最后添加(把PP\_MAX增加了1)



* PROTO\_BIT\_\_PROFINET为本项目中对新增的解码类型自己定义的标志。一般情况下想对IP数据进行预处理，使用标志PROTO\_BIT\_\_IP即可，这个数值定义在Decode.h文件中。**不要使用**PROTO\_BIT\_\_PROFINET**这个参数。**
* RegisterPreprocStats为Snort退出时预处理器输出总结信息的模块添加函数，即添加到此函数中的函数ProfinetPrintStats()会在程序结尾执行一次。不需要总结输出的话可以注释掉
* args为预处理器的配置信息

1. ProfinetRTFunct()函数，自由发挥。对每一个数据包的处理可以写在此函数中
2. PreprocCleanExitFunction PreprocRestartFunction未修改
3. 修改snort-2.9.8.2/src/preprocessors/Makefile.am文件

把此处理器的文件添加到libspp\_a\_SOURCES中去 spp\_profinet.c spp\_profinet.h \

1. 回到snort-2.9.8.2路径下automake自动修改makefile（src/preprocessors/路径下）
2. 重新make、 make install
3. 修改snort.conf文件，即启动snort IDS模式时-c 后面的那个文件

在Setp 5中添加

preprocessor Profinet\_RT

即启动Profinet\_RT预处理器

1. Over

备注：

1. 想要启动预处理器报警需修改/etc/snort/snort.conf文件

# include $PREPROC\_RULE\_PATH/preprocessor.rules 去掉前面的井号

1. 定制自己的预处理器报警需在/etc/snort/preproc\_rules/preprocessor.rules中添加报警规则，例如

alert ( msg: "Profinet Test 1"; sid:**1**; gid:146; rev: 1; metadata: rule-type preproc; classtype:protocol-command-decode; )

每一个预处理器使用一种gid

预处理器的每一种报警信息，使用一个sid

1. 预处理器中调用

SnortEventqAdd(146, **1**, 1, 0, 0,"aaaaaaaaaaaaaaaaaaaaaaaaaaaaa", 0);即可

1. 最好在/etc/snort/gen-msg中添加对应的预处理器与报警信息的对应信息，便于参考

![C:\Users\tuhao\AppData\Roaming\Tencent\Users\540544929\TIM\WinTemp\RichOle\9`I[]](2KLSZ[C93DY08CGY.png](data:image/png;base64,)

## 我的HelloSnort插件编写

编写过程简述：

1. 根据spp\_template.c里面的内容修改spp\_template.c和spp\_template.h
   1. 将这两个文件名称修改为spp\_hello.c和spp\_hello.h
2. 修改spp\_hello.c函数
   1. SetupHello
   2. HelloFunction
3. 修改spp\_hello.h函数
   1. 声明SetupHello
4. 修改plugbase.c函数
   1. #include spp\_hello.h
   2. RegistProcesser函数中调用SetupHello
5. 修改preprocids.h
   1. #define PP\_MAX 38
   2. #define PP\_HELLO 37
6. 修改Makefile.am,并在根目录运行automake命令
7. make, make install 直接覆盖已安装的snort
8. 修改/etc/snort/snort.conf
9. 测试：snort -dev -c /etc/snort/snort.conf

过程中出现的问题：

1. aumake版本的确定
   1. 查看源代码目录下的文件Makefile

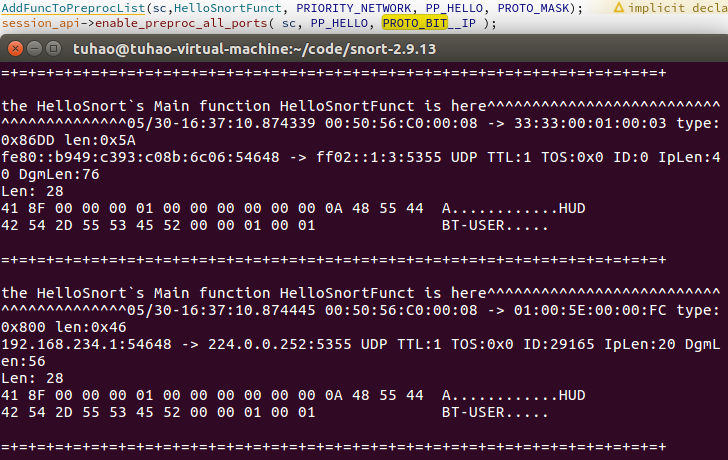
Makefile.in generated by automake 1.16.1 from Makefile.am

* 1. 版本为automake1.16.1

1. 预处理插件被加载而且初始化完成，但是预处理插件主函数不工作
   1. 原因是新版本的Snort新增加了

session\_api->enable\_preproc\_all\_ports

加入这一行代码后，预处理器运行正常



代码：spp\_hello.c

/\* $Id$ \*/

/\* Snort Preprocessor Plugin Source File Template \*/

/\*

\* Purpose:

\*

\* Preprocessors perform some function \*once\* for \*each\* packet. This is

\* different from detection plugins, which are accessed depending on the

\* standard rules. When adding a plugin to the system, be sure to

\* add the "Setup" function to the InitPreprocessors() function call in

\* plugbase.c!

\*

\* Arguments:

\*

\* This is the list of arguements that the plugin can take at the

\* "preprocessor" line in the rules file

\*

\* Effect:

\*

\* What the preprocessor does. Check out some of the default ones

\* (e.g. spp\_frag2) for a good example of this description.

\*

\* Comments:

\*

\* Any comments?

\*

\*/

#include <sys/types.h>

#include <stdlib.h>

#include <ctype.h>

#include <rpc/types.h>

#include "snort\_debug.h"

#include "session\_api.h"

/\*

\* If you're going to issue any alerts from this preproc you

\* should include generators.h and event\_wrapper.h

\*/

#include "generators.h"

#include "event\_wrapper.h"

#include "decode.h"

#include "util.h"

#include "plugbase.h"

#include "parser.h"

#ifdef HAVE\_CONFIG\_H

#include "config.h"

#endif

#include "snort.h"

/\*

\* put in other inculdes as necessary

\*/

/\*

\* your preprocessor header file goes here if necessary, don't forget

\* to include the header file in plugbase.h too!

\*/

#include "spp\_hello.h"

#define PROTO\_MASK 0x0001

/\*

\* define any needed data structs for things like configuration

\*/

typedef struct \_TemplateData

{

/\* Your struct members here \*/

} TemplateData;

/\*

\* If you need to instantiate the preprocessor's

\* data structure, do it here

\*/

TemplateData SomeData;

/\*

\* function prototypes go here

\*/

static void HelloSnortInit(struct \_SnortConfig \*sc,char \*);

static void ParseTemplateArgs(char \*);

static void HelloSnortFunct(Packet \*);

static void PreprocCleanExitFunction(int, void \*);

static void PreprocRestartFunction(int, void \*);

static void helloSnortreloadFuction(char \*args);

/\*

\* Function: SetupTemplate()

\*

\* Purpose: Registers the preprocessor keyword and initialization

\* function into the preprocessor list. This is the function that

\* gets called from InitPreprocessors() in plugbase.c.

\*

\* Arguments: None.

\*

\* Returns: void function

\*

\*/

void SetupHello()

{

/\*

\* link the preprocessor keyword to the init function in

\* the preproc list

\*/

#ifndef SNORT\_RELOAD

RegisterPreprocessor("Hello", HelloSnortInit);

#else

RegisterPreprocessor("Hello", HelloSnortInit,

helloSnortreloadFuction, NULL,NULL, NULL);

#endif

printf("now call the setupHelloSnort <><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>");

DEBUG\_WRAP(DebugMessage(DEBUG\_PLUGIN,"Preprocessor: HelloSnort is setup...\n"););

}

/\*

\* Function: TemplateInit(u\_char \*)

\*

\* Purpose: Calls the argument parsing function, performs final setup on data

\* structs, links the preproc function into the function list.

\*

\* Arguments: args => ptr to argument string

\*

\* Returns: void function

\*

\*/

static void HelloSnortInit(struct \_SnortConfig \*sc,char \*args)

{

DEBUG\_WRAP(DebugMessage(DEBUG\_PLUGIN,"Preprocessor: HelloSnortInit Initialized\n"););

printf("HelloSnortInit ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^is setup");

/\*

\* parse the argument list from the rules file

\*/

//ParseTemplateArgs(args);

/\*

\* perform any other initialization functions that are required here

\*/

/\*

\* Set the preprocessor function into the function list

\*/

//example: AddFuncToPreprocList(PreprocEvalFunc pp\_eval\_func, uint16\_t priority,uint32\_t preproc\_id, uint32\_t proto\_mask)

AddFuncToPreprocList(sc,HelloSnortFunct, PRIORITY\_NETWORK, PP\_HELLO, PROTO\_MASK);

session\_api->enable\_preproc\_all\_ports( sc, PP\_HELLO, PROTO\_BIT\_\_IP );

// AddFuncToCleanExitList(PreprocCleanExitFunction, NULL);

// AddFuncToRestartList(PreprocRestartFunction, NULL);

}

/\*

\* Function: ParseTemplateArgs(char \*)

\*

\* Purpose: Process the preprocessor arguements from the rules file and

\* initialize the preprocessor's data struct. This function doesn't

\* have to exist if it makes sense to parse the args in the init

\* function.

\*

\* Arguments: args => argument list

\*

\* Returns: void function

\*

\*/

static void ParseTemplateArgs(char \*args)

{

/\* your parsing function goes here, check out the other spp files

for examples \*/

}

/\*

\* Function: PreprocFunction(Packet \*)

\*

\* Purpose: Perform the preprocessor's intended function. This can be

\* simple (statistics collection) or complex (IP defragmentation)

\* as you like. Try not to destroy the performance of the whole

\* system by trying to do too much....

\*

\* Arguments: p => pointer to the current packet data struct

\*

\* Returns: void function

\*

\*/

static void HelloSnortFunct(Packet \*p)

{

/\* your preproc function goes here.... \*/

/\*

\* if you need to issue an alert from your preprocessor, check out

\* event\_wrapper.h, there are some useful helper functions there

\*/

printf("the HelloSnort`s Main function HelloSnortFunct is here^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^");

}

/\*

\* Function: PreprocCleanExitFunction(int, void \*)

\*

\* Purpose: This function gets called when Snort is exiting, if there's

\* any cleanup that needs to be performed (e.g. closing files)

\* it should be done here.

\*

\* Arguments: signal => the code of the signal that was issued to Snort

\* data => any arguments or data structs linked to this

\* functioin when it was registered, may be

\* needed to properly exit

\*

\* Returns: void function

\*/

static void PreprocCleanExitFunction(int signal, void \*data)

{

/\* clean exit code goes here \*/

}

static void helloSnortreloadFuction(char \*args)

{

printf("call the reload hellosnort");

}

/\*

\* Function: PreprocRestartFunction(int, void \*)

\*

\* Purpose: This function gets called when Snort is restarting on a SIGHUP,

\* if there's any initialization or cleanup that needs to happen

\* it should be done here.

\*

\* Arguments: signal => the code of the signal that was issued to Snort

\* data => any arguments or data structs linked to this

\* functioin when it was registered, may be

\* needed to properly exit

\*

\* Returns: void function

\*/

static void PreprocRestartFunction(int signal, void \*foo)

{

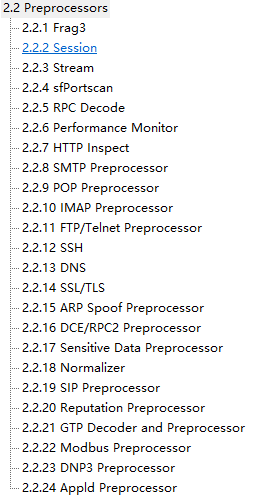
/\* restart code goes here \*/

}

# 预处理器DEMO

Snort是基于规则的入侵检测系统，规则更新比较快，预处理器相对固定。

以下预处理器，在安装的时候默认全都安装进去了。



其中有一个Modbus Prerocessor

Modbus预处理器是一个Snort模块，用于解码Modbus协议。它还提供了访问某些协议字段的规则选项。这允许一个用户编写Modbus数据包的规则而不用使用一系列的“内容”和“byte\_test”选项来解码协议

SID描述

1-Modbus标头中的长度与Modbus功能代码所需长度不匹配.

每个Modbus功能都有一个预期的请求和响应格式。

如果消息的长度与预期格式不匹配，则为此警报生成。

2-Modbus协议ID不为零。

协议ID字段用于其他协议与MODBUS的复用(multiplexing)。由于预处理器无法处理这些其他协议，而是生成此警报。

3保留使用的Modbus功能代码正在被使用

# 数据包结构体

可以向预处理器中的函数传递Packet指针，获取数据包信息，然后进行处理。

**typedef** struct \_Packet

**{**

const DAQ\_PktHdr\_t **\***pkth**;** // packet meta data

const uint8\_t **\***pkt**;** // raw packet data

//vvv------------------------------------------------

// TODO convenience stuff to be refactored for layers

//^^^------------------------------------------------

//vvv-----------------------------

EtherARP **\***ah**;**

const EtherHdr **\***eh**;** /\* standard TCP/IP/Ethernet/ARP headers \*/

const VlanTagHdr **\***vh**;**

EthLlc **\***ehllc**;**

EthLlcOther **\***ehllcother**;**

const PPPoEHdr **\***pppoeh**;** /\* Encapsulated PPP of Ether header \*/

const GREHdr **\***greh**;**

uint32\_t **\***mpls**;**

const CiscoMetaHdr **\***cmdh**;** /\* Cisco Metadata Header \*/

const IPHdr **\***iph**,** **\***orig\_iph**;**/\* and orig. headers for ICMP\_\*\_UNREACH family \*/

const IPHdr **\***inner\_iph**;** /\* if IP-in-IP, this will be the inner IP header \*/

const IPHdr **\***outer\_iph**;** /\* if IP-in-IP, this will be the outer IP header \*/

const TCPHdr **\***tcph**,** **\***orig\_tcph**;**

const UDPHdr **\***udph**,** **\***orig\_udph**;**

const UDPHdr **\***inner\_udph**;** /\* if Teredo + UDP, this will be the inner UDP header \*/

const UDPHdr **\***outer\_udph**;** /\* if Teredo + UDP, this will be the outer UDP header \*/

const ICMPHdr **\***icmph**,** **\***orig\_icmph**;**

const uint8\_t **\***data**;** /\* packet payload pointer \*/

const uint8\_t **\***ip\_data**;** /\* IP payload pointer \*/

const uint8\_t **\***outer\_ip\_data**;** /\* Outer IP payload pointer \*/

//^^^-----------------------------

void **\***ssnptr**;** /\* for tcp session tracking info... \*/

void **\***fragtracker**;** /\* for ip fragmentation tracking info... \*/

//vvv-----------------------------

IP4Hdr **\***ip4h**,** **\***orig\_ip4h**;**

IP6Hdr **\***ip6h**,** **\***orig\_ip6h**;**

ICMP6Hdr **\***icmp6h**,** **\***orig\_icmp6h**;**

IPH\_API**\*** iph\_api**;**

IPH\_API**\*** orig\_iph\_api**;**

IPH\_API**\*** outer\_iph\_api**;**

IPH\_API**\*** outer\_orig\_iph\_api**;**

int family**;**

int orig\_family**;**

int outer\_family**;**

//^^^-----------------------------

PreprocEnableMask preprocessor\_bits**;** /\* flags for preprocessors to check \*/

uint64\_t packet\_flags**;** /\* special flags for the packet \*/

uint32\_t xtradata\_mask**;**

uint16\_t proto\_bits**;**

//vvv-----------------------------

uint16\_t dsize**;** /\* packet payload size \*/

uint16\_t ip\_dsize**;** /\* IP payload size \*/

uint16\_t alt\_dsize**;** /\* the dsize of a packet before munging (used for log)\*/

uint16\_t actual\_ip\_len**;** /\* for logging truncated pkts (usually by small snaplen)\*/

uint16\_t outer\_ip\_dsize**;** /\* Outer IP payload size \*/

//^^^-----------------------------

uint16\_t frag\_offset**;** /\* fragment offset number \*/

uint16\_t ip\_frag\_len**;**

uint16\_t ip\_options\_len**;**

uint16\_t tcp\_options\_len**;**

//vvv-----------------------------

uint16\_t sp**;** /\* source port (TCP/UDP) \*/

uint16\_t dp**;** /\* dest port (TCP/UDP) \*/

uint16\_t orig\_sp**;** /\* source port (TCP/UDP) of original datagram \*/

uint16\_t orig\_dp**;** /\* dest port (TCP/UDP) of original datagram \*/

//^^^-----------------------------

// and so on ...

int16\_t application\_protocol\_ordinal**;**

uint8\_t frag\_flag**;** /\* flag to indicate a fragmented packet \*/

uint8\_t mf**;** /\* more fragments flag \*/

uint8\_t df**;** /\* don't fragment flag \*/

uint8\_t rf**;** /\* IP reserved bit \*/

uint8\_t ip\_option\_count**;** /\* number of options in this packet \*/

uint8\_t tcp\_option\_count**;**

uint8\_t ip6\_extension\_count**;**

uint8\_t ip6\_frag\_index**;**

uint8\_t error\_flags**;** /\* flags indicate checksum errors, bad TTLs, etc. \*/

uint8\_t encapsulated**;**

uint8\_t GTPencapsulated**;**

uint8\_t next\_layer**;** /\* index into layers for next encap \*/

#ifndef NO\_NON\_ETHER\_DECODER

const Fddi\_hdr **\***fddihdr**;** /\* FDDI support headers \*/

Fddi\_llc\_saps **\***fddisaps**;**

Fddi\_llc\_sna **\***fddisna**;**

Fddi\_llc\_iparp **\***fddiiparp**;**

Fddi\_llc\_other **\***fddiother**;**

const Trh\_hdr **\***trh**;** /\* Token Ring support headers \*/

Trh\_llc **\***trhllc**;**

Trh\_mr **\***trhmr**;**

Pflog1Hdr **\***pf1h**;** /\* OpenBSD pflog interface header - version 1 \*/

Pflog2Hdr **\***pf2h**;** /\* OpenBSD pflog interface header - version 2 \*/

Pflog3Hdr **\***pf3h**;** /\* OpenBSD pflog interface header - version 3 \*/

Pflog4Hdr **\***pf4h**;** /\* OpenBSD pflog interface header - version 4 \*/

#ifdef DLT\_LINUX\_SLL

const SLLHdr **\***sllh**;** /\* Linux cooked sockets header \*/

#endif

#ifdef DLT\_IEEE802\_11

const WifiHdr **\***wifih**;** /\* wireless LAN header \*/

#endif

const EtherEapol **\***eplh**;** /\* 802.1x EAPOL header \*/

const EAPHdr **\***eaph**;**

const uint8\_t **\***eaptype**;**

EapolKey **\***eapolk**;**

#endif

// nothing after this point is zeroed ...

Options ip\_options**[**IP\_OPTMAX**];** /\* ip options decode structure \*/

Options tcp\_options**[**TCP\_OPTLENMAX**];** /\* tcp options decode struct \*/

IP6Option **\***ip6\_extensions**;** /\* IPv6 Extension References \*/

CiscoMetaOpt **\***cmd\_options**;** /\* Cisco Metadata header options \*/

const uint8\_t **\***ip\_frag\_start**;**

const uint8\_t **\***ip\_options\_data**;**

const uint8\_t **\***tcp\_options\_data**;**

const IP6RawHdr**\*** raw\_ip6h**;** // innermost raw ip6 header

Layer layers**[**LAYER\_MAX**];** /\* decoded encapsulations \*/

IPAddresses inner\_ips**,** inner\_orig\_ips**;**

IP4Hdr inner\_ip4h**,** inner\_orig\_ip4h**;**

IP6Hdr inner\_ip6h**,** inner\_orig\_ip6h**;**

IPAddresses outer\_ips**,** outer\_orig\_ips**;**

IP4Hdr outer\_ip4h**,** outer\_orig\_ip4h**;**

IP6Hdr outer\_ip6h**,** outer\_orig\_ip6h**;**

MplsHdr mplsHdr**;**

H2Hdr **\***h2Hdr**;**

PseudoPacketType pseudo\_type**;** // valid only when PKT\_PSEUDO is set

uint16\_t max\_dsize**;**

/\*\*policyId provided in configuration file. Used for correlating configuration

\* with event output

\*/

uint16\_t configPolicyId**;**

uint32\_t iplist\_id**;**

unsigned char iprep\_layer**;**

uint8\_t ps\_proto**;** // Used for portscan and unified2 logging

uint8\_t ips\_os\_selected**;**

void **\***cur\_pp**;**

// Expected session created due to this packet.

struct \_ExpectNode**\*** expectedSession**;**

**}** Packet**;**

总结：

第一次接触大的工程，不熟悉Makefile，搭建Snort开发环境失败。

automake，autoconf等等软件。