第一章 snort 简介

snort 有三种工作模式：**嗅探器、数据包记录器、网络入侵检测系统**。嗅探器模式仅仅是从网络上读取数据包并作为连续不断的流显示在终端上。数据包记录器模式把数据包记录到硬盘上。网路入侵检测模式是最复杂的，而且是可配置的。可以让 snort 分析网络数据流以匹配用户定义的一些规则，并根据检测结果采取一定的动作。

嗅探器

所谓的嗅探器模式就是 snort 从网络上读出数据包然后显示在控制台上。如果只要把 TCP/IP 包头信息打印在屏幕上，只需要输入下面的命令：

```
./snort -v
```

使用这个命令将使 snort 只输出 IP 和 TCP/UDP/ICMP 的包头信息。如果要看到应用层的数据，可以使用：

```
./snort -vd
```

这条命令使 snort 在输出包头信息的同时显示包的数据信息。如果还要显示数据链路层的信息，就使用下面的命令：

```
./snort -vde
```

注意这些选项开关还可以分开写或者任意结合在一块。例如：下面的命令就和上面最后的一条命令等价：

```
./snort -d -v –e
```

数据包记录器

如果要把所有的包记录到硬盘上，需要指定一个日志目录，snort 就会自动记录数据包：

```
./snort -dev -l ./log
```

当然，./log 目录必须存在，否则 snort 就会报告错误信息并退出。当 snort 在这种模式下运行，它会记录所有看到的包将其放到一个目录中，这个目录以数据包目的 IP 地址命名，例如：192.168.10.1

如果只指定了-l 命令开关，而没有设置目录名，snort 有时会使用远程主机的 IP 地址作为目录，有时会使用本地主机 IP 地址作为目录名。为了只对本地网络进行日志，需要给出本地网络：

```
./snort -dev -l ./log -h 192.168.1.0/24
```

这个命令告诉 snort 把进入 C 类网络 192.168.1 的所有包的数据链路、TCP/IP 以及应用层的数据记录到目录./log 中。

如果网络速度很快，或者想使日志更加紧凑以便以后的分析，那么应该使用二进制的日志文件格式。所谓的二进制日志文件格式就是 tcpdump 程序使用的格式。使用下面的命令可以把所有的包记录到一个单一的二进制文件中：

```
./snort -l ./log -b
```

注意此处的命令行和上面的有很大的不同。勿需指定本地网络，因为所有的东西都被记录到一个单一的文件。不必冗余模式或者使用-d、-e 功能选项，因为数据包中的所有内容都会被记录到日志文件中。

可以使用任何支持 tcpdump 二进制格式的嗅探器程序从这个文件中读出数据包，例如：tcpdump 或者 Ethereal。使用-r 功能开关，也能使 snort 读出包的数据。snort 在所有运行模式下都能够处理 tcpdump 格式的文件。例如：如果在嗅探器模式下把一个 tcpdump 格式的二进制文件中的包打印到屏幕上，可以输入下面的命令：

```
./snort -dv -r packet.log
```

在日志包和入侵检测模式下，通过 BPF(BSD Packet Filter)接口，可以使用许多方式维护日志文件中的数据。例如，只想从日志文件中提取 ICMP 包，只需要输入下面的命令行：

```
./snort -dvr packet.log icmp
```

网络入侵检测系统

snort 最重要的用途还是作为网络入侵检测系统(NIDS)，使用下面命令行可以启动这种模式：

```
./snort -dev -l ./log -h 192.168.1.0/24 -c snort.conf
```

snort.conf 是规则集文件。snort 会对每个包和规则集进行匹配，发现这样的包就采取相应的行动。如果不指定输出目录，snort 就输出到/var/log/snort 目录。

注意：如果想长期使用 snort 作为自己的入侵检测系统，最好不要使用-v 选项。因为使用这个选项，使 snort 向屏幕上输出一些信息，会大大降低 snort 的处理速度，从而在向显示器输出的过程中丢弃一些包。

此外，在绝大多数情况下，也没有必要记录数据链路层的包头，所以-e 选项也可以不用：

```
./snort -d -h 192.168.1.0/24 -l ./log -c snort.conf
```

这是使用 snort 作为网络入侵检测系统最基本的形式，日志符合规则的包，以 ASCII 形式保存在有层次的目录结构中。

网络入侵检测模式下的输出选项

在 NIDS 模式下，有很多的方式来配置 snort 的输出。在默认情况下，snort 以 ASCII 格式记录日志，使用 full 报警机制。如果使用 full 报警机制，snort 会在包头之后打印报警消息。如果不需要日志包，可以使用-N 选项。

snort 有 6 种报警机制：full、fast、socket、syslog、smb(winpopup)和 none。其中有 4 个可以在命令行状态下使用-A 选项设置。这 4 个是：

- -A fast：报警信息包括：一个时间戳(timestamp)、报警消息、源/目的 IP 地址和端口。
  -A full：是默认的报警模式。
  -A unsock：把报警发送到一个 UNIX 套接字，需要有一个程序进行监听，这样可以实现实时报警。
  -A none：关闭报警机制。

使用-s 选项可以使 snort 把报警消息发送到 syslog，默认的设备是 LOG_AUTHPRIV 和 LOG_ALERT。可以修改 snort.conf 文件修改其配置。

snort 还可以使用 SMB 报警机制，通过 SAMBA 把报警消息发送到 Windows 主机。为了使用这个报警机制，在运行./configure 脚本时，必须使用--enable-smbalerts 选项。

下面是一些输出配置的例子：

使用默认的日志方式(以解码的 ASCII 格式)并且把报警发给 syslog：

```
./snort -c snort.conf -l ./log -s -h 192.168.1.0/24
```

使用二进制日志格式和 SMB 报警机制：

```
./snort -c snort.conf -b -M WORKSTATIONS
```

第二章 编写 snort 规则

基础

snort 使用一种简单的，轻量级的规则描述语言，这种语言灵活而强大。在开发 snort 规则时要记住几个简单的原则。

第一， 大多数 snort 规则都写在一个单行上，或者在多行之间的行尾用/分隔。Snort 规则被分成两个逻辑部分：规则头和规则选项。规则头包含规则的动作，协议，源和目标 ip 地址与网络掩码,以及源和目标端口信息;规则选项部分包含报警消息内容和要检查的包的具体部分。

下面是一个规则范例：

```
alert tcp any any -> 192.168.1.0/24 111 (content:\"|00 01 86 a5|\"; msg: \"mountd access\";)
```

第一个括号前的部分是规则头（rule header），包含的括号内的部分是规则选项（rule options）。规则选项部分中冒号前的单词称为选项关键字（option keywords）。注意，不是所有规则都必须包含规则选项部分，选项部分只是为了使对要收集或报警，或丢弃的包的定义更加严格。组成一个规则的所有元素对于指定的要采取的行动都必须是真的。当多个元素放在一起时，可以认为它们组成了一个逻辑与（AND）语句。同时，snort 规则库文件中的不同规则可以认为组成了一个大的逻辑或（OR）语句。

规则高级概念

Includes:

include 允许由命令行指定的规则文件包含其他的规则文件。

格式：

include: <include file path/name>

注意在该行结尾处没有分号。被包含的文件会把任何预先定义的变量值替换为自己的变量引用。参见变量（Variables）一节以获取关于在 SNORT 规则文件中定义和使用变量的更多信息。

Variables :

变量可能在 snort 中定义。

格式：

var: <name> <value>

例子：

```
var MY_NET 192.168.1.0/24
alert tcp any any -> $MY_NET any (flags: S; msg: \"SYN packet\";)
```

规则变量名可以用多种方法修改。可以在\"$\"操作符之后定义变量。\"?\" 和 \"-\"可用于变量修改操作符。

$var - 定义变量。
$(var) - 用变量\"var\"的值替换。

$(var:-default) - 用变量\"var\"的值替换，如果\"var\"没有定义用\"default\"替换。
$(var:?message) - 用变量\"var\"的值替换或打印出错误消息\"message\"然后退出。

例子：

```
var MY_NET $(MY_NET:-192.168.1.0/24)
log tcp any any -> $(MY_NET:?MY_NET is undefined!) 23
```

Config

Snort 的很多配置和命令行选项都可以在配置文件中设置。

格式：

config <directive> [: <value>]

Directives

- order 改变规则的顺序(snort -o)
- alertfile 设置报警输出文件，例如：config alertfile: alerts
- classification 创建规则分类。
- decode_arp 开启 arp 解码功能。(snort -a)
- dump_chars_only 开启字符倾卸功能。(snort -C)
- dump_payload 倾卸应用层数据。(snort -d)
- decode_data_link 解码第二层数据包头。(snort -e)
- bpf_file 指定 BPF 过滤器(snort -F)。例如：config bpf_file: filename.bpf
- set_gid 改变 GID (snort -g)。例如：config set_gid: snort_group
- daemon 以后台进程运行。(snort -D)
- reference_net 设置本地网络。 (snort -h). 例如：config reference_net:192.168.1.0/24
- interface 设置网络接口(snort –i)。例如：config interface: xl0
- alert_with_interface_name 报警时附加上接口信息。(snort -I)
- logdir 设置记录目录 (snort -l)。例如：config logdir: /var/log/snort
- umask 设置 snort 输出文件的权限位。(snort -m). Example: config umask: 022
- pkt_count 处理 n 个数据包后就退出。(snort -n). Example: config pkt_count: 13
- nolog 关闭记录功能，报警仍然有效。 (snort -N)
- obfuscate 使 IP 地址混乱 (snort -O)
- no_promisc 关闭混杂模式。(snort -p)
- quiet 安静模式，不显示标志和状态报告。(snort -q)
- checksum_mode 计算校验和的协议类型。类型值：none, noip, notcp, noicmp, noudp, all
- utc 在时间戳上用 UTC 时间代替本地时间。 (snort -U)
- verbose 将详细记录信息打印到标准输出。 (snort -v)
- dump_payload_verbose 倾卸数据链路层的原始数据包 (snort -X)

- show_year 在时间戳上显示年份。(snort -y)
- stateful 为 stream4 设置保证模式。
- min_ttl 设置一个 snort 内部的 ttl 值以忽略所有的流量。
- disable_decode_alerts 关闭解码时发出的报警。
- disable_tcpopt_experimental_alerts 关闭 tcp 实验选项所发出的报警。
- disable_tcpopt_obsolete_alerts 关闭 tcp 过时选项所发出的报警。
- disable_tcpopt_ttcp_alerts 关闭 ttcp 选项所发出的报警。
- disable_tcpopt_alerts 关闭选项长度确认报警。
- disable_ipopt_alerts 关闭 IP 选项长度确认报警。
- detection 配置检测引擎。(例如：search-method lowmem)
- reference 给 snort 加入一个新的参考系统。

规则头

规则动作：

规则的头包含了定义一个包的 who，where 和 what 信息，以及当满足规则定义的所有属性的包出现时要采取的行动。规则的第一项是\"规则动作\"（rule action），\"规则动作\"告诉 snort 在发现匹配规则的包时要干什么。在 snort 中有五种动作：alert、log、pass、activate 和 dynamic.

1、Alert-使用选择的报警方法生成一个警报，然后记录（log）这个包。
2、Log-记录这个包。
3、Pass-丢弃（忽略）这个包。
4、activate-报警并且激活另一条 dynamic 规则。
5、dynamic-保持空闲直到被一条 activate 规则激活，被激活后就作为一条 log 规则执行。
可以定义自己的规则类型并且附加一条或者更多的输出模块给它，然后就可以使用这些规则类型作为 snort 规则的一个动作。

下面这个例子创建一条规则，记录到 tcpdump。

```
ruletype suspicious
{
type log output
log_tcpdump: suspicious.log
}
```

下面这个例子创建一条规则，记录到系统日志和 MySQL 数据库

```
ruletype redalert
{
type alert output
alert_syslog: LOG_AUTH LOG_ALERT
output database: log, mysql, user=snort dbname=snort host=localhost
}
```

协议

规则的下一部分是协议。Snort 当前分析可疑包的 ip 协议有四种：tcp 、udp、icmp 和 ip。将来可能会更多，例如 ARP、IGRP、GRE、OSPF、RIP、IPX 等。

Ip 地址

规则头的下一个部分处理一个给定规则的 ip 地址和端口号信息。关键字\"any\"可以被用来定义任何地址。Snort 没有提供根据 ip 地址查询域名的机制。地址就是由直接的数字型 ip 地址和一个 cidr 块组成的。Cidr 块指示作用在规则地址和需要检查的进入的任何包的网络掩码。/24 表示 c 类网络，/16 表示 b 类网络，/32 表示一个特定的机器的地址。例如，192.168.1.0/24 代表从 192.168.1.1 到 192.168.1.255 的地址块。在这个地址范围的任何地址都匹配使用这个 192.168.1.0/24 标志的规则。这种记法给我们提供了一个很好的方法来表示一个很大的地址空间。

有一个操作符可以应用在 ip 地址上，它是否定运算符（negation operator）。这个操作符告诉 snort 匹配除了列出的 ip 地址以外的所有 ip 地址。否定操作符用"! "表示。下面这条规则对任何来自本地网络以外的流都进行报警。

```
alert tcp !192.168.1.0/24 any -> 192.168.1.0/24 111 (content: \"|00 01
86 a5|\"; msg: \"external mountd access\";)
```

这个规则的 ip 地址代表"任何源 ip 地址不是来自内部网络而目标地址是内部网络的 tcp 包"。

可以指定 ip 地址列表，一个 ip 地址列表由逗号分割的 ip 地址和 CIDR 块组成，并且要放在方括号内“[”，“]”。此时，ip 列表可以不包含空格在 ip 地址之间。下面是一个包含 ip 地址列表的规则的例子。

```
alert tcp ![192.168.1.0/24,10.1.1.0/24] any ->
[192.168.1.0/24,10.1.1.0/24] 111 (content: \"|00 01 86 a5|\"; msg:
\"external mountd access\";)
```

端口号

端口号可以用几种方法表示，包括"any"端口、静态端口定义、范围、以及通过否定操作符。"any"端口是一个通配符，表示任何端口。静态端口定义表示一个单个端口号，例如 111 表示 portmapper，23 表示 telnet，80 表示 http 等等。端口范围用范围操作符"："表示。范围操作符可以有数种使用方法，如下所示：

```
log udp any any -> 192.168.1.0/24 1:1024
```

记录来自任何端口的，目标端口范围在 1 到 1024 的 udp 流

log tcp any any -> 192.168.1.0/24 :6000

记录来自任何端口，目标端口小于等于 6000 的 tcp 流

log tcp any :1024 -> 192.168.1.0/24 500:

记录来自任何小于等于 1024 的特权端口，目标端口大于等于 500 的 tcp 流

端口否定操作符用"！"表示。它可以用于任何规则类型（除了 any，这表示没有，呵呵）。例如，由于某个古怪的原因需要记录除 x windows 端口以外的所有一切，可以使用类似下面的规则：

log tcp any any -> 192.168.1.0/24 !6000:6010

方向操作符

方向操作符"->"表示规则所施加的流的方向。方向操作符左边的 ip 地址和端口号被认为是流来自的源主机，方向操作符右边的 ip 地址和端口信息是目标主机，还有一个双向操作符"<>"。它告诉 snort 把地址/端口号对既作为源，又作为目标来考虑。这对于记录/分析双向对话很方便，例如 telnet 或者 pop3 会话。用来记录一个 telnet 会话的两侧的流的范例如下：

log !192.168.1.0/24 any <> 192.168.1.0/24 23

Activate 和 dynamic 规则：

注：Activate 和 dynamic 规则将被 tagging 所代替。在 snort 的将来版本，Activate 和 dynamic 规则将完全被功能增强的 tagging 所代替。

Activate 和 dynamic 规则对给了 snort 更强大的能力。你现在可以用一条规则来激活另一条规则，当这条规则适用于一些数据包时。在一些情况下这是非常有用的，例如你想设置一条规则：当一条规则结束后来完成记录。Activate 规则除了包含一个选择域：activates 外就和一条 alert 规则一样。Dynamic 规则除了包含一个不同的选择域：activated_by 外就和 log 规则一样，dynamic 规则还包含一个 count 域。

Actevate 规则除了类似一条 alert 规则外，当一个特定的网络事件发生时还能告诉 snort 加载一条规则。Dynamic 规则和 log 规则类似，但它是当一个 activate 规则发生后被动态加载的。把他们放在一起如下图所示：

```
activate tcp !$HOME_NET any -> $HOME_NET 143 (flags: PA;
content: "|E8C0FFFFFF|/bin"; activates: 1; msg: "IMAP buffer
overflow!\";)
dynamic tcp !$HOME_NET any -> $HOME_NET 143 (activated_by:
1; count: 50;)
```

规则选项

规则选项组成了 snort 入侵检测引擎的核心，既易用又强大还灵活。所有的 snort 规则选项用分号"；"隔开。规则选项关键字和它们的参数用冒号"："分开。按照这种写法，snort 中有 42 个规则选项关键字。

msg - 在报警和包日志中打印一个消息。
logto - 把包记录到用户指定的文件中而不是记录到标准输出。
ttl - 检查 ip 头的 ttl 的值。
tos 检查 IP 头中 TOS 字段的值。
id - 检查 ip 头的分片 id 值。
ipoption 查看 IP 选项字段的特定编码。
fragbits 检查 IP 头的分段位。
dsize - 检查包的净荷尺寸的值 。
flags -检查 tcp flags 的值。
seq - 检查 tcp 顺序号的值。
ack - 检查 tcp 应答（acknowledgement）的值。
window 测试 TCP 窗口域的特殊值。
itype - 检查 icmp type 的值。
icode - 检查 icmp code 的值。
icmp_id - 检查 ICMP ECHO ID 的值。
icmp_seq - 检查 ICMP ECHO 顺序号的值。
content - 在包的净荷中搜索指定的样式。
content-list 在数据包载荷中搜索一个模式集合。
offset - content 选项的修饰符，设定开始搜索的位置 。
depth - content 选项的修饰符，设定搜索的最大深度。
nocase - 指定对 content 字符串大小写不敏感。
session - 记录指定会话的应用层信息的内容。
rpc - 监视特定应用/进程调用的 RPC 服务。
resp - 主动反应（切断连接等）。
react - 响应动作（阻塞 web 站点）。
reference - 外部攻击参考 ids。
sid - snort 规则 id。
rev - 规则版本号。
classtype - 规则类别标识。
priority - 规则优先级标识号。

uricontent – 在数据包的 URI 部分搜索一个内容。
tag – 规则的高级记录行为。
ip_proto – IP 头的协议字段值。
sameip – 判定源 IP 和目的 IP 是否相等。
stateless – 忽略刘状态的有效性。
regex – 通配符模式匹配。
distance – 强迫关系模式匹配所跳过的距离。
within – 强迫关系模式匹配所在的范围。
byte_test – 数字模式匹配。
byte_jump – 数字模式测试和偏移量调整。

msg

msg 规则选项告诉记录和报警引擎,记录或报警一个包的内容的同时打印的消息。它是一个简单的文本字符串，转义符是"\"。
格式:

```
msg: "<message text>";
```

logto

logto 选项告诉 snort 把触发该规则的所有的包记录到一个指定的输出日志文件中。这在把来自诸如 nmap 活动，http cgi 扫描等等的数据组合到一起时很方便。需要指出的是当 snort 工作在二进制记录模式下时这个选项不起作用。
格式:

```
logto:"filename";
```

ttl

这个规则选项用于设置一个要检查的存活期的值。只有确切地匹配时它所进行的检查才成功。这个选项关键字用于检测 traceroute。

格式:

```
ttl:<number>;
```

TOS

tos 关键字允许你验证 IP 头中 TOS 字段为一个特殊的值。只有匹配时才执行成功。
格式:

```
tos: <number>;
```

id

这个选项关键字用于检测 ip 头的分片 id 的值。有些黑客工具（以及别的程序）为了各种目的设置这个域的值，例如一些黑客常使用 31337。用一个简单的规则检查这个值就可以对付他们。
格式:

id: <number>;

Ipoption

如果数据包中使用了 IP 选项，Ipoption 选项会查找使用中的某个特别 IP 选项，比如源路由。这个选项的合法参数如下：

rr - Record route（记录路由）
eol - End of list （列表结尾）
nop - No op （无所作为）
ts - Time Stamp （时间戳）
sec - IP security option （IP 安全选项）
lsrr - Loose source routing （松散源路由）
ssrr - Strict source routing （严格源路由）
satid - Stream identifier （流标示符）

松散和严格源路由是 IP 选项中最经常被检查的内容，但是它们并没有被用在任何广泛使用的 Internet 应用中。每一个特定的规则只能用这个选项一次。

格式：
ipoption: option;

Fragbits

这条规则检测 IP 头中的分段和保留位字段的值，共有三个位能被检测，保留位 RB(Reserved Bit)，更多分段位 MF（More Fragments ），和不分段位 DF（Don’t Fragment）。这些位可以结合在一起来检测。使用下面的值来代表这些位，R-RB，M-MF，D-DF。你也可以使用修饰语对特殊的位来指出合理的匹配标准：* + 所有标记匹配特殊位外加任何其他*；*-任何标记匹配如果任何位被设置为*；！如果指定位没有设置就没有标记匹配。

格式：
fragbits: <bitvalues>;
例子：
alert tcp !$HOME_NET any -> $HOME_NET any (fragbits: R+; msg: "Rerserved bit set!";)

dsize

dsize 选项用于检查包的净荷的大小。它可以设置成任意值，可以使用大于/小于符号来指定范围。例如，如果你知道某个特定的服务有一个特定大小的缓冲区，你可以设定这个选项来监视缓冲区溢出的企图。它在检查缓冲区溢出时比检查净荷内容的方法要快得多。

格式：
dsize: [<>]<number>[<><number>];

说明：“> <”号是可选的。

content

content 关键字是 snort 中比较重要的一个。它允许用户设置规则在包的净荷中搜索指定的内容并根据内容触发响应。当进行 content 选项模式匹配时，Boyer-Moore 模式匹配函数被调用，并且对包的内容进行检查（很花费计算能力）。如果包的净荷中包含的数据确切地匹配了参数的内容，这个检查成功并且该规则选项的其他部分被执行。注意这个检查是大小写敏感的。

Content 关键字的选项数据比较复杂;它可以包含混合的文本和二进制数据。二进制数据一般包含在管道符号中（"|"），表示为字节码（bytecode）。字节码把二进制数据表示为 16 进制数字，是描述复杂二进制数据的好方法。下面是包含了一个混合数据的 snort 规则范例。

格式：

```
content: [!] "<content string>";
```

例子：

```
alert tcp any any -> 192.168.1.0/24 143 (content: "|90C8 C0FF FFFF|/bin/sh"; msg: "IMAP buffer overflow!";)
```

注：多内容的规则可以放在一条规则中，还有（: ; / “）不能出现在 content 规则中。如果一条规则前面有一个“！”。那么那些不包含这些内容的数据包将触发报警。这对于关注那些不包含一定内容的数据包是有用的。

offset

offset 规则选项被用作使用 content 规则选项关键字的规则的修饰符。这个关键字修饰符指定模式匹配函数从包净荷开始处开始搜索的偏移量。它对于 cgi 扫描检测规则很有用，cgi 扫描的内容搜索字符串不会在净荷的前 4 个字节中出现。小心不要把这个偏移量设置的太严格了，会有可能漏掉攻击！这个规则选项关键字必须和 content 规则选项一起使用。

格式：

```
offset: <number>;
```

depth

depth 也是一个 content 规则选项修饰符。它设置了内容模式匹配函数从他搜索的区域的起始位置搜索的最大深度。它对于限制模式匹配函数超出搜索区域指定范围而造成无效搜索很有用。（也就是说，如果你在一个 web 包中搜索 "cgi-bin/phf"，你可能不需要浪费时间搜索超过净荷的头 20 个字节）。

格式：

```
depth: <number>;
```

例子：

```
alert tcp any any -> 192.168.1.0/24 80 (content: "cgi-bin/phf"; offset: 3; depth: 22; msg: "CGI-PHF access";)
```

nocase

nocase 选项用于取消 content 规则中的大小写敏感性。它在规则中指定后，任何与包净荷进行比较的 ascii 字符都被既作为大写又作为小写对待。

格式：
nocase；
例子：
```
alert tcp any any -> 192.168.1.0/24 21 (content: "USER root";
nocase; msg: "FTP root user access attempt";)
```

flags

这个规则检查 tcp 标志。在 snort 中有 9 个标志变量：

F - FIN (LSB in TCP Flags byte)
S - SYN
R - RST
P - PSH
A - ACK
U - URG
2 - Reserved bit 2
1 - Reserved bit 1 (MSB in TCP Flags byte)
0 - No TCP Flags Set

在这些标志之间还可以使用逻辑操作符：
+ ALL flag, 匹配所有的指定的标志外加一个标志。
* ANY flag, 匹配指定的任何一个标志。
! NOT flag, 如果指定的标志不在这个数据包中就匹配成功。

保留位可以用来检测不正常行为，例如 IP 栈指纹攻击或者其他可疑的行为。

格式：
```
flags: <flag values>[,mask value];
```
例子：
```
alert any any -> 192.168.1.0/24 any (flags: SF,12; msg:
"Possible SYN FIN scan";)
```

seq

这个规则选项引用 tcp 顺序号（sequence number）。基本上，它探测一个包是否有一个静态的顺序号集，因此很少用。它是为了完整性而包含进来的。
格式：
```
seq: <number>;
```

ack

ack 规则选项关键字引用 tcp 头的确认（acknowledge）部分。这个规则的一个实用的目的是：检查 nmap tcp ping，nmap tcp ping 把这个域设置为 0，然后发送一个 tcp ack flag 置位的包来确定一个网络主机是否活着。
格式：
```
ack: <number>;
```
例子：

```
alert any any -> 192.168.1.0/24 any (flags: A; ack: 0; msg:
"NMAP TCP ping";)
```

Window

这条规则选项指向 TCP 窗口大小。这个选项检查静态窗口大小，此外别无他用。包括它只是为了完整性。

格式：

```
window:[!]<number>;
```

Itype

这条规则测试 ICMP 的 type 字段的值。它被设置为使用这个字段的数字值。要得到所有可能取值的列表，可以参见 Snort 包中自带的 decode.h 文件，任何 ICMP 的参考资料中也可以得到。应该注意的是，type 字段的取值可以超过正常范围，这样可以检查用于拒绝服务或 flooding 攻击的非法 type 值的 ICMP 包。

格式：

```
itype: <number>;
```

Icode

Icode 规则选项关键字和 itype 规则非常接近，在这里指定一个数值，Snort 会探测使用该值作为 code 值的 ICMP 包。超出正常范围的数值可用于探测可疑的流量。

格式：

```
icode: <number>;
```

Session

Session 关键字用于从 TCP 会话中抽取用户数据。要检查用户在 telnet，rlogin，ftp 或 web sessions 中的用户输入，这个规则选项特别有用。Session 规则选项有两个可用的关键字作为参数：printable 或 all。Printable 关键字仅仅打印用户可以理解或者可以键入的数据。All 关键字使用 16 进制值来表示不可打印的字符。该功能会显著地降低 Snort 的性能，所以不能用于重负载环境。它适合于对二进制（tcpdump 格式）log 文件进行事后处理。

格式：

```
session: [printable|all];
```

例子

```
log tcp any any <> 192.168.1.0/24 23 (session: printable;)
```

Icmp_id

Icmp_id 选项检查 ICMP ECHO 数据包中 ICMP ID 数值是否是指定值。许多秘密通道（covert channel）程序使用静态 ICMP 字段通讯，所以该选项在检查这种流量时非常有用。这个特别的插件用于增强由 Max Vision 编写的 stacheldraht 探测规则，但是在探测一些潜在攻击时确实有效。
格式：
```
icmp_id: <number>;
```

Icmp_seq

Icmp_seq 选项检查 ICMP ECHO 数据包中 ICMP sequence 字段数值是否是指定值。许多秘密通道（covert channel）程序使用静态 ICMP 字段通讯，所以该选项在检查这种流量时非常有用。这个特别的插件用于增强由 Max Vision 编写的 stacheldraht 探测规则，但是在探测一些潜在攻击时确实有效。（我知道该字段的信息和 icmp_id 的描述几乎完全相同，实际上它们就是同样的东西！）
格式：
```
icmp_seq: <number>;
```

Rpc

这个选项查看 RPC 请求，并自动将应用（Application）、过程（procedure）和程序版本（program version）译码，如果所有三个值都匹配的话，该规则就显示成功。这个选项的格式为"应用、过程、版本"。在过程和版本域中可以使用通配符"*"。

格式：
```
rpc: <number, [number|*], [number|*]>;
```
例子
```
alert tcp any any -> 192.168.1.0/24 111 (rpc: 100000,*,3;
msg:"RPC getport (TCP)";)
alert udp any any -> 192.168.1.0/24 111 (rpc: 100000,*,3;
msg:"RPC getport (UDP)";)
alert udp any any -> 192.168.1.0/24 111 (rpc: 100083,*,*;
msg:"RPC ttdb";)
```
Resp

Resp 关键字可以对匹配一条 Snort 规则的流量进行灵活的反应（flexible reponse -FlexResp）。FlexResp 代码允许 Snort 主动地关闭恶意的连接。该插件合法的参数如下：

rst_snd - 向发送方发送 TCP-RST 数据包
rst_rcv - 向接受方发送 TCP-RST 数据包
rst_all - 向收发双方发送 TCP_RST 数据包
icmp_net - 向发送方发送 ICMP_NET_UNREACH
icmp_host - 向发送方发送 ICMP_HOST_UNREACH
icmp_port - 向发送方发送 ICMP_PORT_UNREACH

icmp_all - 向发送方发送上述所有的 ICMP 数据包

在向目标主机发送多种响应数据包时，这些选项组合使用。多个参数之间使用逗号分隔。

格式：

```
resp: <resp_modifier[, resp_modifier...];
```

使用 resp 选项时要小心，因为很容易就会使 snort 陷入无限循环中，例如如下规则：

```
alert tcp any any -> 192.168.1.1/24 any (msg: "aiee!"; resp: rst_all;)
```

content_list

content_list 关键字允许多内容字符串被放在一个单独的内容匹配选项中，被匹配的字符串被存放在指定的文件中，而且每个字符串要单独占用一行。否则他们就等同于一个 content 字符串。这个选项是 react 关键字的基础。

格式；

```
content-list: <file_name>;
```

下面是一个文件的内容：

```
# adult sites
"porn"
"porn"
"adults"
"hard core"
"www.pornsite.com"
```

React

注意，使用这个功能很容易使网络流量陷入回路。React 关键字以匹配一个规则时所作出的灵活的反应为基础。基本的反应是阻塞一些引人注意的站点的用户的访问。响应代码允许 snort 积极的关掉有冒犯行为的访问和/或发送一个通知给浏览者。这个通知可以包含你自己的注释。这个选项包括如下的基本修饰词：

block——关闭连接并且发送一个通知

warm——发送明显的警告信息

基本修饰词可以和如下的附加修饰词组合使用：

msg——把 msg 选项的内容包含进阻塞通知信息中

proxy<port_nr>——使用代理端口发送通知信息

大量的附加修饰词由逗号隔开，react 关键字将被放在选项的最后一项。

格式：

```
react: <react_basic_modifier[,
react_additional_modifier]>;
```

例子：

```
alert tcp any any <> 192.168.1.0/24 80 (content: "bad.htm";
msg: "Not for children!"; react: block, msg;)
```

reference

这个关键字允许规则包含一个外面的攻击识别系统。这个插件目前支持几种特定的系统，它和支持唯一的 URL 一样好。这些插件被输出插件用来提供一个关于产生报警的额外信息的连接。
确信先看一看如下地方：
http://www.snort.org/snort-db
格式：

```
reference: <id system>,<id>;
```

例子：

```
alert tcp any any -> any 7070 (msg: "IDS411/dos-realaudio";
flags: AP; content: "|fff4 fffd 06|"; reference:
arachNIDS,IDS411;)
alert tcp any any -> any 21 (msg:
"IDS287/ftp-wuftp260-venglin-linux"; flags: AP; content:
"|31c031db 31c9b046 cd80 31c031db|"; reference:
arachNIDS,IDS287; reference: bugtraq,1387; reference:
cve,CAN-2000-1574; )
```

Sid

这个关键字被用来识别 snort 规则的唯一性。这个信息允许输出插件很容易的识别规则的 ID 号。

sid 的范围是如下分配的：
<100 保留做将来使用
100-1000,000 包含在 snort 发布包中
>1000,000 作为本地规则使用

文件 sid-msg.map 包含一个从 msg 标签到 snort 规则 ID 的映射。这将被 post-processing 输出模块用来映射一个 ID 到一个报警信息。

格式：

```
sid: <snort rules id>;
```

rev

这个关键字是被用来识别规则修改的。修改，随同 snort 规则 ID，允许签名和描述被较新的信息替换。
格式：

```
rev: <revision integer>
```

Classtype

这个关键字把报警分成不同的攻击类。通过使用这个关键字和使用优先级，用户可以指定规则类中每个类型所具有的优先级。具有 classification 的规则有一个缺省的优先级。

格式：

```
classtype: <class name>;
```

在文件 classification.config 中定义规则类。这个配置文件使用如下的语法：

```
config classification: <class name>,<class
description>,<default priority>
```

Priority

这个关键字给每条规则赋予一个优先级。一个 classtype 规则具有一个缺省的优先级，但这个优先级是可以被一条 priority 规则重载的。

格式：

```
priority: <priority integer>;
```

Uricontent

这个关键字允许只在一个请求的 URI（URL）部分进行搜索匹配。它允许一条规则只搜索请求部分的攻击，这样将避免服务数据流的错误报警。关于这个关键字的参数的描述可以参考 content 关键字部分。这个选项将和 HTTP 解析器一起工作。（只能搜索第一个“/”后面的内容）。

格式：

```
uricontent:[!]<content string>;
```

Tag

这个关键字允许规则记录不仅仅是触发这条规则的那个数据包。一旦一条规则被触发，来自这个主机的数据包将被贴上“标签”。被贴上标签的数据流将被记录用于随后的响应代码和提交攻击流量的分析。

格式：

```
tag: <type>, <count>, <metric>, [direction]
```

type

session 记录触发这条规则的会话的数据包

host 记录激活 tag 规则的主机的所有数据包（这里将使用[direction]修饰词

count Count 指定一个单位的数量。这个单位由<metric>给出。

metric

packets 标记主机/会话的<count>个数据包。
seconds 标记主机/会话的<count>秒。

例子：

```
alert tcp !$HOME_NET any -> $HOME_NET 143 (flags: A+; content: "|e8 c0ff ffff|/bin/sh"; tag: host, 300, packets, src; msg: "IMAP Buffer overflow, tagging!";)
alert tcp !$HOME_NET any -> $HOME_NET 23 (flags: S; tag: session, 10, seconds; msg: "incoming telnet session";)
```

Ip_proto

Ip_proto 关键字允许检测 IP 协议头。这些协议可以是由名字标识的，参考/etc/protocols 文件。在规则中要谨慎使用 ip_protocol 关键字。

格式：

```
ip_proto:[!] <name or number>;
```

例子：

```
alert ip !$HOME_NET any -> $HOME_NET any (msg: "IGMP traffic detected"; ip_proto: igmp;)
```

SameIP

Sameip 关键字允许规则检测源 IP 和目的 IP 是否相等。

格式：

```
sameip;
```

例子：

```
alert ip $HOME_NET any -> $HOME_NET any (msg: "SRC IP == DST IP"; sameip;)
```

Regex

这个模块现在还正在开发，所以在当前的产品规则集中还不能使用。如果使用的话，它将触发一个错误信息。

Flow

这个选项要和 TCP 流重建联合使用。它允许规则只应用到流量流的某个方向上。这将允许规则只应用到客户端或者服务器端。这将能把内网客户端流览 web 页面的数据包和内网服务器所发送的数据包区分开来。这个确定的关键字能够代替标志：A+ 这个标志在显示已建立的 TCP 连接时都将被使用。

选项：

to_client 触发服务器上从 A 到 B 的响应。
to_server 触发客户端上从 A 到 B 的请求。
from_client 触发客户端上从 A 到 B 的请求。
from_server 触发服务器上从 A 到 B 的响应。
established 只触发已经建立的 TCP 连接。

stateless 不管流处理器的状态都触发（这对处理那些能引起机器崩溃的数据包很有用。

no_stream 不在重建的流数据包上触发（对 dsize 和 stream4 有用。

only_stream 只在重建的流数据包上触发。

格式：

```
flow:[to_client|to_server|from_client|from_server|established|stateless|no_stream|only_stream]}
```

例子：

```
alert tcp !$HOME_NET any -> $HOME_NET 21 (flow: from_client; content: "CWD incoming"; nocase; msg: "cd incoming detected"; )
alert tcp !$HOME_NET 0 -> $HOME_NET 0 (msg: "Port 0 TCP traffic"; flow: stateless;)
```

Fragoffset

这个关键字允许把 IP 分段偏移值和一个十进制数相比较。为了抓到一个 IP 会话的第一个分段,你可以使用这个 fragbits 关键字并且和 fragoffset:0 选项一起查看更多的分段选项。

格式：

```
fragoffset:[<|>]<number>
```

例子：

```
alert ip any any -> any any (msg: "First Fragment"; fragbits: M; fragoffset: 0;)
```

Rawbytes

Rawbytes 关键字允许规则查看 telnet 解码数据来处理不常见的数据。这将使得 telnet 协议代码独立于预处理程序来检测。这是对前面的 content 的一个修饰。

格式：

```
rawbytes;
```

例子：

```
alert tcp any any -> any any (msg: "Telnet NOP"; content: "|FF F1|"; rawbytes;)
```

distance

distance 关键字是 content 关键字的一个修饰词,确信在使用 content 时模式匹配间至少有 N 个字节存在。它被设计成在规则选项中和其他选项联合使用。

格式：

```
distance: <byte count>;
```

例子：

```
alert tcp any any -> any any (content: "2 Patterns"; content: "ABCDE"; content: "EFGH"; distance: 1;)
```

Within

Winthin 关键字是 content 关键字的一个修饰词，确保在使用 content 时模式匹配间至多有 N 个字节存在。它被设计成在规则选项中和 distance 选项联合使用。

格式：

```
within: <byte count>;
```

例子：

```
alert tcp any any -> any any (content: "2 Patterns"; content: "ABCDE"; content: "EFGH"; within: 10;)
```

Byte_Test

测试一个字节的域为特定的值。能够测试二进制值或者把字节字符串转换成二进制后再测试。

格式：

```
byte_test: <bytes_to_convert>, <operator>, <value>, <offset> [[relative],[big],[little],[string],[hex],[dec],[oct]]
```

bytes_to_convert 从数据包取得的字节数。

operator 对检测执行的操作（<,>,=,!）。

value 和转换后的值相测试的值。

offset 开始处理的字节在负载中的偏移量。

relative 使用一个相对于上次模式匹配的相对的偏移量。

big 以网络字节顺序处理数据（缺省）。

little 以主机字节顺序处理数据。

string 数据包中的数据以字符串形式存储。

hex 把字符串数据转换成十六进制数形式。

dec 把字符串数据转换成十进制数形式。

oct 把字符串数据转换成八进制数形式。

例子：

```
alert udp $EXTERNAL_NET any -> $HOME_NET any (msg:"AMD procedure 7 plog overflow "; content: "|00 04 93 F3|"; content: "|00 00 00 07|"; distance: 4; within: 4; byte_test: 4,>, 1000, 20, relative;)

alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"AMD procedure 7 plog overflow "; content: "|00 04 93 F3|"; content: "|00 00 00 07|"; distance: 4; within: 4; byte_test: 4, >,1000, 20, relative;)
```

Byte_Jump

Byte_jump 选项用来取得一定数量的字节，并把它们转换成数字形式，跳过一些字节以进一步进行模式匹配。这就允许相对模式匹配在网络数据中进行数字值匹配。

格式：

```
byte_jump: <bytes_to_convert>, <offset>
[[relative],[big],[little],[string],[hex],[dec],[oct],[al
ign]]
```

bytes_to_convert 从数据包中选出的字节数。
offset 开始处理的字节在负载中的偏移量。
relative 使用一个相对于上次模式匹配的相对的偏移量。
big 以网络字节顺序处理数据（缺省）。
little 以主机字节顺序处理数据。
string 数据包中的数据以字符串形式存储。
hex 把字符串数据转换成十六进制数形式。
dec 把字符串数据转换成十进制数形式。
oct 把字符串数据转换成八进制数形式。
align 以 32 位为边界对转换的字节数对齐，即转换的字节数为 4 的倍数。

例子：

```
alert udp any any -> any 32770:34000 (content: "|00 01 86 B8|";
content: "|00 00 00 01|"; distance: 4; within: 4; byte_jump:
4, 12, relative, align; byte_test: 4, >, 900, 20, relative;
msg: "statd format string buffer overflow";)
```

第三章 预处理程序

预处理程序从 Snort 版本 1.5 开始引入，使得 Snort 的功能可以很容易地扩展，用户和程序员能够将模块化的插件方便地融入 Snort 之中。预处理程序代码在探测引擎被调用之前运行，但在数据包译码之后。通过这个机制，数据包可以通过额外的方法被修改或分析。使用 preprocessor 关键字加载和配置预处理程序。在 Snort 规则文件中的 preprocessor 指令格式如下：

```
preprocessor <name>: <options>
```

例子：

```
preprocessor minfrag: 128
```

HTTP Decode

HTTP Decode 用于处理 HTTP URI 字符串并且将串中的数据转化为可读的 ASCII 字串。HTTP 对于一些特性定义了一个十六进制编码方法，例如字符串%20 被解释成一个空格。Web 服务器被设计成能够处理无数的客户端并且支持多种不同的标准。

格式：

http_decode:<port list> [unicode] [iis_alt_unicode] [double_encode] [iis_flip_slash] [full_whitespace]
例子：
```
preprocessor http_decode: 80 8080 unicode iis_flip_slash iis_alt_unicode
```

Portscan Detector

Snort Portscan 预处理程序的用处：

向标准记录设备中记录从一个源 IP 地址来的端口扫描的开始和结束。

如果指定了一个记录文件，在记录扫描类型的同时也记录目的 IP 地址和端口。

端口扫描定义为在时间 T（秒）之内向超过 P 个端口进行 TCP 连接尝试，或者在时间 T（秒）之内向超过 P 个端口发送 UDP 数据包。端口扫描可以是对任一 IP 地址的多个端口，也可以是对多个 IP 地址的同一端口进行。现在这个版本可以处理一对一和一对多方式的端口扫描，下一个完全版本将可以处理分布式的端口扫描（多对一或多对多）。端口扫描也包括单一的秘密扫描（stealth scan）数据包，比如 NULL，FIN，SYNFIN，XMAS 等。如果包括秘密扫描的话，端口扫描模块会对每一个扫描数据包告警。为避免这种情况，可以在 Snort 标准发行版中的 scan-lib 文件里把有关秘密扫描数据包的小节注释掉，这样对每次扫描就只记录一次。如果使用外部记录特性，可以在记录文件中看到（端口扫描的？）技术和类型。该模块的参数如下：

network to monitor - 监视端口扫描的目标网络以 network/CIDR 表示。
number of ports - 在探测期间访问的端口数目。
detection period - 以秒计数的端口访问时间限制。
logdir/filename - 告警信息存放的目录/文件名，告警也可以写入标准的告警文件中。

格式：
```
portscan: <monitor network> <number of ports> <detection period> <file path>
```
例子：
```
preprocessor portscan: 192.168.1.0/24 5 7 /var/log/portscan.log
```

Portscan Ignorehosts

如果用户的服务器（比如 NTP，NFS 和 DNS 服务器）会妨碍端口扫描的探测，可以通知 portscan 模块忽略源自这些主机的 TCP SYN 和 UDP 端口扫描。该模块的参数为 IPs/CIDR 的列表。

格式：
```
portscan-ignorehosts: <host list>
```
例子：
```
preprocessor portscan-ignorehosts: 192.168.1.5/32
192.168.3.0/24
```

Frag2

Frag2 是一个新的 IP 碎片重组预处理器。Frag2 的内存使用和碎片时间超时选项是可配置的。不给出参数，frag2 将使用缺省的内存量（4MB）和时间超时值（60 秒）。这个时间值用来决定一个没有重组的分段将被丢弃的时间长度。
格式
```
preprocessor frag2: [memcap <xxx>], [timeout <xx>], [min_ttl
<xx>], [detect_state_problems], [ttl_limit <xx>]
```

timeout <seconds> 在状态表中保存一个不活跃的流的最大时间值，如果发现活动就重新刷新对话并且这个会话被自动拾起。缺省值是 30 秒。
memcap <bytes> 内存消耗的最大值，如果超出这个值，frag2 就强制削减那些不活跃的会话，缺省值是 4MB。detect_state_problems turns on alerts for events such as overlapping fragments
min_ttl <xx> 设置 frag2 接受的最小 ttl 值。
detect_state_problems 发现重叠分段时报警。
ttl_limit <xx> 设置 ttl 的极限值，它可以避免报警。(初始化段 TTL +/- TTL Limit)
例子：
```
preprocessor frag2: memcap 16777216, timeout 30
```

Stream4

Stream4 模块使 snort 具有 TCP 流从新组装和状态分析能力。强壮的流重组能力使得 snort 能够忽视无“状态”攻击，例如，stick 粘滞位攻击。Stream4 也能够给大量用户提供超过 256 个 TCP 同步连接。Stream4 缺省配置时能够处理 32768 个 TCP 同步连接。Stream4 有两个可配置的模块，stream4 preprocessor 和相关的 stream4_reassemble 插件。stream4_reassemble 有如下选项：
Stream4 格式：
```
preprocessor stream4: [noinspect], keepstats
[machine|binary], [timeout <seconds>], [memcap <bytes>],
[detect_scans], [detect_state_problems],
[disable_evasion_alerts], [ttl_limit <count>]
```

noinspect 关闭状态监测能力。
keepstats [machine|binary] 保持会话统计，如果是“machine”选项就从机器以平坦的模式读入，如果是“binary”选项就用统一的二进制模式输出。
timeout <seconds> 在状态表中保存一个不活跃的流的最大时间值，如果发现活动就重新刷新对话并且这个会话被自动拾起。缺省值是 30 秒。
memcap <bytes> 内存消耗的最大值，如果超出这个值，frag2 就强制削减那些不活跃的会话，缺省值是 8MB。
detect_scans 打开 portscan 的报警能力。
detect_state_problems 打开流事件报警能力，例如，没有 RST 的数据包、带有数据的 SYN 包和超出窗口序列号的包。
disable_evasion_alerts 关闭事件报警能力，例如，TCP 重叠。
ttl_limit 设置 ttl 的极限值。

Stream4_Reassemble 格式：

```
preprocessor stream4_reassemble: [clientonly],
[serveronly],[noalerts], [ports <portlist>]
```

clientonly 对一个连接的客户端提供重组
serveronly 对一个连接的服务器端提供重组
noalerts 对于插入和逃避攻击事件不发出报警
ports <portlist> - 一个空格分隔的执行重组的端口列表，all 将对所有的端口进行重组。缺省对如下端口重组： 21 23 25 53 80 110 111 143 和 513

注： 在配置文件中仅仅设置 stream4 和 stream4_reassemble 命令而没有参数，它们将会使用缺省的参数配置。Stream4 引入了一个新的命令行参数：-z 。在 TCP 流量中，如果指定了 -z 参数，snort 将只对那些通过三次握手建立的流以及那些协作的双向活动的流（即，一些流量走一个方向而其他一些除了一个 RST 或 FIN 外走相反方向）检测报警。当设置了-z 选项后 snort 就完全忽略基于 TCP 的 stick/snot 攻击。

Conversation

Conversation 预处理器使 Snort 能够得到关于协议的基本的会话状态而不仅仅是由 *spp_stream4 处理的 TCP 状态。*

目前它使用和 stream4 相同的内存保护机制，所以它能保护自己免受 DOS 攻击。当它接收到一个你的网络不允许的协议的数据包时，它也能产生一个报警信息。要做到这一点，请在 IP 协议列表中设置你允许的 IP 协议，并且当它收到一个不允许的数据包时，它将报警并记录这个数据包。
格式：

```
preprocessor conversation: [allowed_ip_protocols
<protonumbers|all>], [timeout <sec>], [alert_odd_protocols],
[max_conversations <number>]
```

Portscan2

这个模块将检测端口扫描。它要求包含 Conversation 预处理器以便判定一个会话是什么时间开始的。它的目的是能够检测快速扫描，例如，快速的 nmap 扫描。
格式：

```
preprocessor portscan2: [scanners_max <num>], [targets_max <num>], [target_limit <num>], [port_limit <num>], [timeout <sec>]
```

scaners_max 一次所支持的扫描一个网络的主机数
targets_max 分配代表主机的节点的最大数
target_limit 在一个扫描触发前，一个扫描器所允许扫描的最大的主机数
port_limit 在一个扫描触发前，一个扫描器所允许扫描的最大的端口数
timeout 一个扫描行为被忘记的秒数

Telnet Decode

telnet_decode 预处理器使 snort 能够标准化 telnet 会话数据的控制协议字符。它把数据包规格和成单独的数据缓存，这样原始数据就能够通过 rawbytes content 修饰词来记录或者检验了。缺省情况下，它运行在 21, 23, 25, 和 119 端口.
格式：

```
preprocessor telnet_decode: <ports>
```

RPC Decode

Rpc_decode 预处理器将 RPC 的多个碎片记录组合成一个完整的记录。它是通过将数据包放在标准缓存中来做到这一点的。如果打开 stream4 预处理器功能。它将只处理客户端的流量。它缺省运行在 111 和 32771 端口。
格式：

```
preprocessor rpc_decode: <ports> [ alert_fragments ] [no_alert_multiple_requests] [no_alert_large_fragments] [no_alert_incomplete]
```

Perf Monitor

这个模块是用来评估 snort 各方面性能的一个工具。它的输出格式和参数格式都是变化的，在这里就不给出注释了。

Http Flow

使用这个模块可以忽略 HTTP 头后面的 HTTP 服务响应。

第四章 输出插件

输出插件使得 Snort 在向用户提供格式化输出时更加灵活。输出插件在 Snort 的告警和记录子系统被调用时运行，在预处理程序和探测引擎之后。规则文件中指令的格式非常类似于预处理程序。

注意：如果在运行时指定了命令行的输出开关，在 Snort 规则文件中指定的输出插件会被替代。例如，如果在规则文件中指定了 alert_syslog 插件，但在命令行中使用了"-A fast"选项，则 alert_syslog 插件会被禁用而使用命令行开关。多个输出插件是在 snort 的配置文件中指定的。当指定多个输出插件时，它们被压入栈并且在事件发生时按顺序调用。关于标准的记录和报警系统，输出模块缺省把数据发送到 /var/log/snort.或者通过使用-l 命令行参数输出到一个用户指定的目录。在规则文件中通过指定 output 关键字，使得在运行时加载输出模块。

格式：

```
output <name>: <options>
```

例子：

```
output alert_syslog: LOG_AUTH LOG_ALERT
```

Alert_syslog

该插件向 syslog 设备发送告警（很像命令行中的-s 开关）。该插件也允许用户指定记录设备，优先于 Snort 规则文件中的设定，从而在记录告警方面给用户更大的灵活性。

可用关键字：

选项（Options）

LOG_CONS
LOG_NDELAY
LOG_PERROR
LOG_PID

设备（Facilities）

LOG_AUTH
LOG_AUTHPRIV

LOG_DAEMON
LOG_LOCAL0
LOG_LOCAL1
LOG_LOCAL2
LOG_LOCAL3
LOG_LOCAL5
LOG_LOCAL6
LOG_LOCAL7
LOG_USER

优先级（**Priorities**）

LOG_EMERG
LOG_ALERT
LOG_CRIT
LOG_ERR
LOG_WARNING
LOG_NOTICE
LOG_INFO
LOG_DEBUG

格式：

```
alert_syslog: <facility> <priority> <options>
```

Alert_fast

将报警信息快速的打印在指定文件的一行里。它是一种快速的报警方法，因为不需要打印数据包头的所有信息。

格式：

```
alert_fast: <output filename>
```

例子：

```
output alert_fast: alert.fast
```

Alert_full

打印数据包头所有信息的报警。这些报警信息写到缺省的日志目录（/var/log/snort）或者写到命令行指定的目录。在日志目录内，每个 IP 都创建一个目录。产生报警的数据包被解码后写到这个目录下的文件里。这些文件的创建将大大降低 snort 的性能。所以这种输出方法对大多数不适用，但那些轻量级的网络环境还是可以使用的。

格式：

```
alert_full: <output filename>
```

例子：

```
output alert_full: alert.full
```

Alert_smb

这个插件将把 WinPopup 报警信息发送给 NETBIOS 命名的机器上的一个文件。并不鼓励使用这个插件，因为它以 snort 权限执行了一个外部可执行二进制程序，通常是 root 权限。那个工作站上接受报警信息的文件每行存放一条报警信息。
格式：
alert_smb: <alert workstation filename>
例子；
output alert_smb: workstation.list

Alert_unixsock

打开一个 UNIX 套接字，并且把报警信息发送到那里。外部的程序 / 进程会在这个套接字上侦听并实时接收这些报警数据。
格式：
alert_unixsock
例子：
output alert_unixsock

Log_tcpdump

log_tcpdump 插件将数据包记录到 tcpdump 格式的文件中。这便于使用已有的多种检查 tcpdump 格式文件的工具，来对收集到的流量数据进行后处理工作。该插件只接受一个参数，即输出文件名
格式：
log_tcpdump: <output filename>
例子：
output log_tcpdump: snort.log

database

该插件由 Jed Pickel 提供将 Snort 数据记录到 Postgres SQL 数据库中。更多的有关安装和配置该插件的信息可以在 Incident.org（http://www.incident.org/snortdb）找到。这个插件的参数是数据库名称和一个参数列表。参数由格式 parameter = argument 来指定。可用参数如下：
host - 连接主机。如果指定了一个非零字串，就使用 TCP/IP 通讯。如果不指定主机名，就会使用 Unix domain socket 连接。
port - 连接服务器主机的端口号，或者是 Unix-domain 连接的 socket 文件名扩展。
dbname - 数据库名。

user - 数据库中身份认证用的用户名。
password - 如果数据库要求口令认证，就使用这个口令。
sensor_name 为 snort 指定一个你自己的名字。如果你不指定，这里就自动产生一个。
encoding 因为数据包负载和选项都是二进制的，所以没有一个轻便简单的方法把它存储在数据库中。没有使用 BLOBS，因为它们在穿越数据库时不是那么轻便的。所以，我们提供了一个 encoding 选项给你。你可以从下面的选项中选择。它们有各自的优缺点。
hex (default) 把二进制数据表示成十六进制字符串
    storage requirements - 二进制的二倍容量
    searchability - 很好用
    human readability - 不是很好读除非你很滑稽，要求邮件处理。
base64 把二进制数据表示成以 64 为基的字符串。
    storage requirements 二进制的 1.3 倍容量。
    searchability - 没有邮件处理是不可能的。
    human readability - 不易读，要求邮件处理。
ascii 把二进制数据表示成 ascii 码字符串。这是唯一的可以释放数据的选项。非 ascii 码数据用… 代替。即使你选择了这个选项，ip 和 tcp 选项数据还将用十六进制表示，因为那些数据用 ascii 码标上没有任何意义。
    storage requirements - 稍微比二进制大，因为避免了一些字符(&,<,>)。
    searchability - 对于搜索文本字符串很好用，而搜索二进制串是不可能的。
    human readability - 很好用。
detail 你想存储多少细节数据，有如下选项：
    full （缺省值）记录一个引起报警数据包的所有的细节（包括 ip/tcp 选项和负载）。
    fast 只记录少量数据。如果选择了这个选项，你将削减了潜在的分析能力，但这仍是一些应用的最佳选项。这将记录下面的字段(timestamp, signature, source ip, destination ip, source port, destination port, tcp flags, and protocol)

此外，还必须定义一个记录方法和数据库类型。有两种记录方法，log 和 alert。

设置为 log 类型，将启动这个程序的数据库记录功能。如果你设置为 log 类型，

输出链表将调用这个插件。设置为 alert 类型，将启动这个程序的数据库报警

输出功能。

当前共有四种数据库类型：MySQL, PostgreSQL, Oracle, 和 unixODBC-

兼容数据库。
格式：
```
output database: log, mysql, dbname=snort user=snort
host=localhost password=xyz
```

CSV

CSV 输出插件可以将报警数据以一种方便的形式输出到一个数据库。这个插件要求两个参数，一个全路径文件名和输出模式选项。下面是模式选项列表。如果模式选项缺省，就按模式选项列表中的顺序输出。

timestamp
msg
proto
src
srcport
dst
dstport
ethsrc
ethdst
ethlen
tcpflags
tcpseq
tcpack
tcplen
tcpwindow
ttl
tos
id
dgmlen
iplen
icmptype
icmpcode
icmpid
icmpseq

格式：

```
output alert_CSV: <filename> <format>
```

例子：

```
output alert_CSV: /var/log/alert.csv default
output alert_CSV: /var/log/alert.csv timestamp, msg
```

Unified

Unified 输出插件被设计成尽可能快的事件记录方法。它记录一个事件到一个报警文件和一个数据包到一个日志文件。报警文件包含一个事件的主要信息（ips, protocol, port, message id）。日志文件包含数据包信息的细节（一个数据包考贝及相关的事件 ID）。

这两个文件都是以 spo_unified.h 文件中描述的二进制形式写的。以

unix 秒为单位的时间将附加到每个文件的后面写出。

格式

```
output alert_unified: <file name>
output log_unified: <file name>
```
例子：
```
output alert_unified: snort.alert
output log_unified: snort.log
```

Log Null

有时创建这样的规则是必要的，即在某些情况下能够发出报警而不记录数据包。当使用 log_null 插件时就相当于命令行的-N 选项，但这个插件可以工作在一个规则类型上。

格式：
```
output log_null

ruletype info {
   type alert
   output alert_fast: info.alert
   output log_null
}
```

自己动手编写好的规则

当编写 snort 规则时，首先考虑的是效率和速度。

好的规则要包含 content 选项。2.0 版本以后，snort 改变了检测引擎的工作方式，在第一阶段就作一个集合模式匹配。一个 content 选项越长，这个匹配就越精确。如果一条规则不包含 content 选项，它们将使整个系统慢下来。

当编写规则时，尽量要把目标定位在攻击的地方（例如，将目标定位在 1025 的偏移量等等）而不仅仅是泛泛的指定（如，在这匹配脚本代码）。Content 规则是大小写敏感的（除非你使用了 nocase 选项）。不要忘记 content 是大小写敏感的和大多数程序的命令都是大写字母。FTP 就是一个很好的例子。考虑如下的规则：
```
alert tcp any any -> 192.168.1.0/24 21 (content: "user root";
msg: "FTP root login";)
alert tcp any any -> 192.168.1.0/24 21 (content: "USER root";
msg: "FTP root login";)
```

上面的第二条规则能检测出大多数的自动以 root 登陆的尝试，而第一条规则就不行。Internet 守护进程在接受输入时是很随便的。在编写规则时，很好的理解协议规范将降低错过攻击的机会。

加速含有内容选项的规则

探测引擎运用规则的顺序和它们在规则中的书写顺序无关。内容规则选项总是最后一个被检验。利用这个事实，应该先运用别的快速规则选项，由这些选项决定是否需要检查数据包的内容。例如：在 TCP 会话建立起来后，从客户端发来的数据包，PSH 和 ACK 这两个 TCP 标志总是被置位的。如果想检验从客户端到服务器的有效载荷，利用这个事实，就可以先进行一次 TCP 标志检验，这比模式匹配算法（pattern match algorithm）在计算上节约许多。使用内容选项的规则要加速的一个简便方法就是也进行一次标志检验。基本思想是，如果 PSH 和 ACK 标志没有置位，就不需要对数据包的有效载荷进行检验。如果这些标志置位，检验标志而带来的计算能力消耗是可以忽略不计的。

```
alert tcp any any -> 192.168.1.0/24 80 (content: "cgi-bin/phf"; flags:
PA; msg: "CGI-PHF probe";)
```