Snort3 规则开发文档

一．规则基本结构

Snort3 中,规则分为两个部分,分别是**规则标头**和**规则主体.**
规则标头:用来定义匹配的流量所采取的动作.
规则主体:定义规则的条件
规则示例:

```
alert tcp $EXTERNAL_NET 80 -> $HOME_NET any
(
    msg:"Attack attempt!";
    flow:to_client,established;
    file_data;
    content:"1337 hackz 1337",fast_pattern,nocase;
    service:http;
    sid:1;
)
```

规则注释:
Snort3 中可以给规则添加注释,#可以表示单行注释.**/*…*/**表示多行注释

```
# 单行注释
/*
多行注释
*/
```

二．规则标头

Snort3 中所有的规则都是由**规则标头**开始的,主要用于规定该规则需要执行的动作和检测的协议.

以下列**规则标头**为例,分别讲解每个部分所代表的含义.

```
alert tcp $EXTERNAL_NET 80 -> $HOME_NET any
```

alert:表示规则的动作
tcp:表示规则的协议
$EXTERNAL_NET、$HOME_NET:表示需要对哪些 IP 地址进行检测
80、any:表示需要对哪些源端口进行检测
->:表示流量的传输方向

2.1 规则动作

alert:告警
block:阻断当前以及该流后续的数据包
drop:丢弃当前数据包
log:记录日志
pass:放行
react:返回响应并终止会话
reject:终止会话
rewrite:覆盖数据包内容

2.2 协议

目前支持网络层,传输层以及应用层的协议.(一条规则只能够指定一个协议)
例如:
ip
udp
tcp
icmp
http
smtp

2.3 IP 地址

目前有四种方式可以指定 IP 地址:

1) 具体的 IP 地址或网段: 1.1.1.1, 2.2.2..0/24
2) 自定义的地址变量,例如: $HOME_NET
3) 任意地址,any
4) 地址列表,通过逗号隔开: [1.1.1.1, 2.2.2..0/24]

备注:

可以使用**!**表示**非**语法

例如: ![192.168.1.0/24,10.1.1.0/24]:

2.4 端口号

目前有 5 种方法可以表示端口号:

1) 任意端口:any
2) 指定端口:80,445
3) 自定义的端口变量,$HTTP_PORTS
4) 端口范围: 1:1024,500:
5) 端口列表:[1:1024,333,5555,$HTTP_PORTS]

备注:

如果指定的应用层协议与流量匹配,则规则中指定的端口号不生效.只有传统的四个协议可以指定端口生效(tcp udp).即如果指定 http 协议,则该规则的端口限定不起作用.

2.5 传输方向

目前有两种传输方向:

->:从左到右

<>:双向

三．Snort3 中新的规则特性

3.1 服务规则

即应用层协议,snort3 中新增了应用层协议的标识.可以在规则中指定应用层协议.
例如:

```
alert http
(
    msg:"SERVER-WEBAPP This rule only looks at HTTP traffic";
    flow:to_server,established;
    http_uri;
    content:"/admin.php",fast_pattern,nocase;
    content:"cmd=",nocase;
    pcre:"/[?&]cmd=[^&]*?\x3b/i";
    sid:1;
)
```

可以在 lua/snort_defaults.lua 中查看支持的协议

```
default_wizard =
{
    spells =
    {
        { service = 'ftp', proto = 'tcp',
          to_client = { '220*FTP', '220*FileZilla' } },

        { service = 'http', proto = 'tcp',
          to_server = http_methods, to_client = { 'HTTP/' } },

        { service = 'imap', proto = 'tcp',
          to_client = { '** OK', '** BYE', '** PREAUTH' } },

        { service = 'pop3', proto = 'tcp',
          to_client = { '+OK', '-ERR' } },

        { service = 'sip',
          to_server = sip_requests, to_client = { 'SIP/' } },

        { service = 'smtp', proto = 'tcp',
          to_server = { 'HELO', 'EHLO' },
          to_client = { '220*SMTP', '220*MAIL' } },

        { service = 'ssh', proto = 'tcp',
          to_server = { 'SSH-' }, to_client = { 'SSH-' } },

        { service = 'dce_http_server', proto = 'tcp',
          to_client = { 'ncacn_http' } },

        { service = 'dce_http_proxy', proto = 'tcp',
          to_server = { 'RPC_CONNECT' } },

    },
    hexes =
    {
        { service = 'dnp3', proto = 'tcp',
          to_server = { '|05 64|' }, to_client = { '|05 64|' } },

        { service = 'netflow', proto = 'udp',
          to_server = netflow_versions },

        { service = 'http2', proto = 'tcp',
          to_client = { '???|04 00 00 00 00 00|' },
          to_server = { '|50 52 49 20 2a 20 48 54 54 50 2f 32 2e 30 0d 0a 0d
0a 53 4d 0d 0a 0d 0a|' } },

--[[
        { service = 'modbus', proto = 'tcp',
          to_server = { '??|0 0|' } },
        { service = 'rpc', proto = 'tcp',
          to_server = { '????|0 0 0 0 0 0 0 1|' },
          to_client = { '????|0 0 0 0 0 0 0 1|' } },
--]]

        { service = 'ssl', proto = 'tcp',
          to_server = { '|16 03|' }, to_client = { '|16 03|' } },

        { service = 'telnet', proto = 'tcp',
          to_server = telnet_commands, to_client = telnet_commands },
    },

    curses = {'dce_udp', 'dce_tcp', 'dce_smb', 'mms', 's7commplus', 'sslv2'}

}
```

或在/src/service_inspectors/wizard/curses.cc 中的 `curse_map` 查看支持的协议

```
// map between service and curse details
static vector<CurseDetails> curse_map
{
    // name          service        alg                 is_tcp
    { "dce_udp"    , "dcerpc"     , dce_udp_curse    , false },
    { "dce_tcp"    , "dcerpc"     , dce_tcp_curse    , true  },
    { "mms"        , "mms"        , mms_curse        , true  },
    { "s7commplus", "s7commplus" , s7commplus_curse, true  },
    { "dce_smb"    , "netbios-ssn", dce_smb_curse    , true  },
    { "sslv2"      , "ssl"        , ssl_v2_curse     , true  }
};
```

3.2 文件规则

支持匹配文件中的特征
目前支持的协议:

- HTTP
- SMTP
- POP3
- IMAP
- SMB
- FTP

使用该特性需要指定 file_data 字段
示例:

```
#http 和 imap 中传输到客户端的文件中出现特征就产生告警
alert tcp $EXTERNAL_NET [80,143] -> $HOME_NET any
(
    msg:"MALWARE-OTHER Win.Ransomware.Agent payload download attempt";
    flow:to_client,established;
    file_data; content:"secret_encryption_key",fast_pattern,nocase;
    service:http, imap;
    classtype:trojan-activity;
    sid:1;
)
#smtp 中传输到 smtp 服务端的文件出现特征就告警
alert tcp $EXTERNAL_NET any -> $SMTP_SERVERS 25
(
    msg:"MALWARE-OTHER Win.Ransomware.Agent payload download attempt";
    flow:to_server,established;
    file_data; content:"secret_encryption_key",fast_pattern,nocase;
    service:smtp;
    classtype:trojan-activity;
    sid:2;
)
  #任何文件中出现特征字符串均产生告警
alert file
(
    msg:"MALWARE-OTHER Win.Ransomware.Agent payload download attempt";
    file_data;
    content:"secret_encryption_key",fast_pattern,nocase;
    classtype:trojan-activity;
    sid:3;
)
```

3.3 文件类型规则

该特征支持通过 file_id 和 file_meta 来识别文件类型.并在其他文件中通过 file_type 来调用.该特性默认启用.

例如:

识别 windows 可执行文件.

```
file_id (
    msg:"Windows/DOS executable file";
    file_meta:type MSEXE, id 21, category "Executables,Dynamic Analysis Capable,Local Malware Analysis Capable";
    file_data;
    content:"| 4D 5A |", depth 2, offset 0;
    gid:4;
    sid:16;
    rev:1;
)
```

识别 PDF 文件

```
file_id (
    msg:"PDF file";
    file_meta:type PDF, id 282, category "PDF files,Dynamic Analysis Capable,Local Malware Analysis Capable", version "1.0";
    file_data;
    content:"| 25 50 44 46 2D 31 2E 30 |", depth 8, offset 0;
    gid:4;
    sid:158;
    rev:1;
)
```

四．规则可选项

4.1 语法关键字

这部分内容可以帮助理解该文档中的部分规则.

1. 斜体字
   表示占位符,需要根据实际情况填充字符
2. 方括号
   方括号内的值表示任意可选
3. 大括号
   大括号内的值表示有且只能有一个
4. 省略号
   省略号表示可以通过逗号分隔添加任意多个.

4.2 一般规则

这部分规则关键字不会影响规则的具体检测逻辑,但建议规则开发者在规则中添加对应的描述信息.

- msg:设置命中规则时需要打印的信息
- reference:参考信息,通常用来设置 url 和 cve
- gid:特定 snort 组件 id
- sid:规则唯一 id
- rev:规则修订号
- classtype:攻击类型
- priority:规则优先级
- metadata:以键值对的形式给规则添加额外的信息
- service:关联的应用层协议
- rem:规则住体中添加注释
- file_meta:用于给文件类型标识规则添加元数据

msg

示例:

```
msg:"SERVER-WEBAPP /etc/inetd.conf file access attempt";
msg:"Malicious file download attempt";
```

reference

示例:

```
reference:url,www.example.com;
reference:cve,2020-1234;
```

gid

表示生成器 id,表示规则的分类分组。默认为 1
例如:
  Gid 1 表示标准文本规则
  Gid 3 表示共享对象规则
  Gid 大于 100 表示内置规则
通过 snort –list-gids 查看对应信息

sid

规则的唯一 id,范围 0-999999

rev

表示规则的修订版本号,默认为 1,修改后递增 1

classtype

表示攻击的分类,在 snort_defaults.lua 中可以看到详细的规则分类表
示例:

```
classtype:web-application-attack;
classtype:attempted-user;
```

priority

攻击的优先级,范围 1-2147483647,1 为最高级

metadata

添加任意键值对信息,键值对以空格分隔,多个键值对以逗号分隔.

service

指定应用层协议,该条件与规则标头中的端口号为**或**的关系.如果在规则标头中已经指定服务或文件识别规则,则不应该使用该字段。

例如:

该规则会检测 tcp 端口为 8000 或 http 协议的流量

```
alert tcp any any -> any any 8000 (
    msg:"HTTP traffic or dst port 8000 please";
    service:http;
    sid:1000000;
)
```

rem

支持在规则主体中创建注释信息.

示例:

```
rem:"check for a malicious URI string";
http_uri;
content:"/php_backdoor.php";
```

file_meta

用来创建文件类型识别规则.使用该字段需要提供两个必要的参数:

type *type*	文件类型的命令
id *type_id*	文件类型对应的 id

还可以添加一些可选参数:

category *type_category* 文件类别的详细描述
group *type_group* 与特定文件类型关联的字符串
version *type_version* 文件版本

示例:

```
# defines the parameters for Windows/DOS executable files
file_meta:type MSEXE, id 21, category "Executables,Dynamic Analysis Capable"
;
file_meta:type PDF, id 282, category "PDF files", version "1.0";
file_meta:type MOV, id 4, category "Multimedia", group "video";
```

4.3 载荷检测规则

4.3.1 关键字快查表

keyword	**description**
content	匹配字符串或 16 进制值
fast_pattern	用于修饰 content 字段,若声明且未命中该特征时,不对后续规则子条件进行检测
nocase	用于修饰 content 字段,忽略大小写
offset	用于修饰 content 字段,表示从缓冲区的第 n 位才开始匹配
depth	用于修饰 content 字段,表示从 offset 开始到第 n 位进行匹配
distance	用于修改 content 字段,表示前一个 content 与下一个 content 之间需要跳过前后个字节
within	用于修饰 content 字段,表示相对与前一个匹配的特征结束的位置前后 n 位进行匹配
HTTP buffers	用于声明各个 http 解码字段
bufferlen	检查指定缓冲区的长度
isdataat	检查特征是否存在于指定位置
dsize	检查数据包的大小
pcre	语法简洁,更快
regex	语法复杂,速度较慢
pkt_data	将检测指针移动到规范数据包开头
raw_data	将检测指针移动到原始数据包开头
file_data	将检测指针移动到文件数据开头
js_data	将检测指针移动到 javascript 数据开头
vba_data	将检测指针移动到 vba 宏代码开头
base64_decode	解码数据包中的 base64 数据
base64_data	将检测指针移动到 base64 解码缓冲区的开头

byte_extract	从缓冲区中读取一些字节并存储到一个命名变量中
byte_test	根据指定的运算符测试数据包中的一个或多个字节
byte_math	从缓冲区中提取一些字节,并进行运算,将结果存储在一个新变量中
byte_jump	
ber_data and ber_skip	处理 ber 编码数据
ssl_state and ssl_version	处理 ssl/tls 协议
DCE Specific Options	处理 dcerpc 协议
SIP Specific Options	处理 SIP 协议
sd_pattern	检测敏感数据
asn1	解码并检测 asn.1 类型,长度和数据
cvs	查找特定的攻击类型
md5, sha256, and sha512	根据指定的哈希值检查缓冲区数据
GTP Specific Options	处理 GTP 协议
DNP3 Specific Options	处理 DNP3 协议
CIP Specific Options	处理 CIP 协议
IEC 104 Specific Options	处理 iec 104 协议
Modbus Specific Options	处理 modbus 协议
S7comm Specific Options	处理 s7comm 协议

4.3.2 content

```
# 检查字符串
content:"USER root";
# 检查字符串&16 进制值   使用||来限定 16 进制值
content:"PK|03 04|";
#修饰符 ,多个修饰符用逗号隔开
content:"pizza", nocase;
content:"cheese";
content:" pizza", within 6;
#以下字符需要使用反斜杠进行转义
;  (分号)
\  (反斜杠)
"  (双引号)
```

4.3.3 fast_pattern

适当地使用该字段可以提高规则的性能,理想的适用场景是命中该部分特征时,规则整体的命中率较高.

以下字段无法使用 fast_pattern 进行修饰:

- http_raw_cookie
- http_param
- http_raw_body
- http_version
- http_raw_request
- http_raw_status
- http_raw_trailer
- http_true_ip

```
content:"super_secret_encryption_key",fast_pattern;
#还可以通过 fast_pattern_offset 和 fast_pattern_length 指定作用域
content:"/index/not_a_cnc_endpoint.php",fast_pattern_offset 6,fast_pattern_length 23;
```

4.3.4 nocase

忽略大小写,同样适用于 16 进制值.

```
#匹配 A 和 a
content:"|41|",nocase;
```

4.3.5 offset/depth/distance/within

```
#表示数据/缓冲区匹配时的起始位置
offset {offset|variable_name}
content:"|FE|SMB", offset 4;

#表示需要匹配的长度
depth {depth|variable_name}
content:"|FE|SMB", depth 4, offset 4;
#未写 offset 时,默认为 0
content:"PK|03 04|", depth 4;
```

```

#表示从前一个特征匹配后跳过多少个字节进行下一次匹配
distance {distance|variable_name}
content:"ABC";
content:"EFG", distance 1;

#表示前一个 content 的结尾到后 n 位之间进行匹配.可以结合 distance 进行使用.默认
distance 为 0
within {within|variable_name}
content:"ABC";
content:"EFG", within 10;
content:"ABC";
content:"EFG", distance 1, within 3;
content:"DEF";
content:"GHI", within 3;
```

4.3.6 http buffers

http_uri

```
 #http_uri 可以分为 6 个部分
#path, query, fragment(url 中#号后面的部分), host, port,  scheme(协议名称)
#会自动执行 url 解码等规范化操作

http_uri[:{scheme|host|port|path|query|fragment}];

http_uri; content:"/basic/example/of/path?query=value",fast_pattern,nocase;

http_uri; content:"/basic/example/of/path",fast_pattern,nocase;
http_uri:query; content:"query=value",nocase;
```

http_raw_uri

不执行规范化操作，其他与 http_uri 一致

http_header

```
#提供请求头和响应头的规范化后的数据
#field 可以指定键值对中的键名
http_header[:field header_name];

http_header;
```

```
content:"User-Agent: abcip",fast_pattern,nocase;
content:"Accept-Language: en-us",nocase,distance 0;

http_header:field user-agent; content:"abcip";
```

http_raw_header

不执行规范化操作,其他与 http_header 一致

http_cookie

```
#会执行规范化操作.
#不包含键名的部分
http_cookie; content:"name=value",depth 10;

http_cookie;
content:"name=value",fast_pattern;
content:"name6=value6",distance 0;
```

http_raw_cookie

不执行规范化操作,其他与 http_cookie 一致

http_param

```
#http_param 会将请求中 uri 和 body 中的键值对解析后,放入该缓冲区
http_param:"param_name"[,nocase];

http_param:"favoriteFood",nocase;
content:"pizza",nocase;

http_uri;
content:"/food.php",fast_pattern,nocase;
http_param:"favoriteFood",nocase;
content:"pizza",nocase;
```

http_method

```
#GET, POST, OPTIONS, HEAD, DELETE, PUT, TRACE, and CONNECT.
http_method; content:"POST";
http_method; content:"GET",fast_pattern;
```

http_version

```
#http 协议版本
http_version; content:"HTTP/1.1";
http_version; content:"HTTP/1.0";
```

http_stat_code

```
#http 响应码
http_stat_code; content:"200";
http_stat_code; content:"403";
```

http_stat_msg

```
#http 响应消息内容
http_stat_msg; content:"OK";
http_stat_msg; content:"Forbidden";
```

http_raw_request

```
#http 请求头的第一行
http_raw_request; content:"GET /robots.txt HTTP/1.1";
http_raw_request; content:"POST /totally_not_vulnerable.php HTTP/1.1";
```

http_raw_status

```
#http 响应头的第一行
http_raw_status; content:"HTTP/1.1 200 OK";
http_raw_status; content:"HTTP/1.1 404 Not Found";
```

http_trailer

```
#http 分块传输的结尾行
```

```
http_trailer[:field field_name];
http_trailer; content:"Expires:";

http_trailer;
content:"Expires:";
content:"2015", within 30;
```

http_raw_trailer

文档中暂未描述,默认为非规范化的数据.

http_true_ip

```
#该部分数据来自请求头中的 X-Forwarded-For" "True-Client-IP" headers, 或其他自定义的字段"X-Forwarded-For-type
http_true_ip; content:"192.168.1.2";
http_true_ip; content:"150.172.238.178";
```

http_version_match

```
#检查协议的版本号
#目前包含以下版本:1.0, 1.1, 2.0, 3.0, 0.9, malformed, other.
#malformed 指协议格式错误的, 例如 1.a
#other 指上面五个版本以外的,例如 8.4
http_version_match:"version[ version]…"[,request];

http_version_match:"0.9 1.0 1.1";
http_version_match:"2.0 3.0";
http_version_match:"other";
http_version_match:"malformed";
```

http_num_headers

```
#该字段表示 http 请求头中有多少个 key
#单一值表达式
http_num_headers:[<|>|=|!|<=|>=]count[,request];
#范围值表达式
http_num_headers:min_count{<>|<=>}max_count[,request];

# 查找数量大于 100 的
```

```
http_num_headers:>100;
查找数量等于 100 的
http_num_headers:100;
# 查找数量在 50 到 100 之间的，不包括 50 和 100
http_num_headers:50<>100;
```

http_num_cookies

```
#用于判断 cookie 中有多少个键值对/值
#单一值表达式
http_num_cookies:[<|>|=|!|<=|>=]count[,request];
#范围值表达式
http_num_cookies:min_count{<>|<=>}max_count[,request];

#超过 100 个的
http_num_cookies:>100;
# 等于 100 个的
http_num_cookies:100;
# 50 到 100 之间的，不包括 50 和 100
http_num_cookies:50<>100;
```

http_header_test

```
#用于判断请求头中的某个值是否为数字，并与指定值做比较。或者判断该值是否存在
http_header:field header_name[,numeric {true|false}][,check range][,absent][
,request];
以下运算符可以用来做比较
[<|>|=|!|<=|>=]number
min_number{<>|<=>}max_number

# 判断 content-length 的值是数字，并且大于 40000000
http_header_test:field content-length,numeric true,check >40000000;
# 判断 content-length 的值不是数字
http_header_test:field content-length,numeric false;
# 判断 user-agent 不存在
http_header_test:field user-agent,absent;
```

http_trailer_test

```
#由于判断分块传输的最后一行,其余用法与 http_header_test 一致
http_trailer:field trailer_name[,numeric {true|false}][,check range][,absent
];
```

```

以下运算符可以用来做比较
[<|>|=|!|<=|>=]number
min_number{<>|<=>}max_number


# 判断 Expires 的值是否为数字
http_trailer_test:field expires,numeric true;
```

4.3.7 bufferlen

```
#该字段用于判断缓冲区的长度
#单一值表达式
bufferlen:[<|>|=|!|<=|>=]length[,relative];
#范围值表达式
bufferlen:min_length{<>|<=>}max_length[,relative];

#检查缓冲区长度大于 100
bufferlen:>100;

#检查缓冲区中是否包含特征，且长度等于 10
http_uri;
content:"/pizza.php?";
bufferlen:10,relative;

#检查 http_client_body 长度在[2,10]之间
http_client_body;
bufferlen:2<=>10;

#检查缓冲区长度在(2,10)之间
http_client_body;
bufferlen:2<>10;
```

4.3.8 isdataat

```
isdataat:[!]location[,relative];
isdataat:100;

#检测 USER 后至少还有 30 个字节的数据
content:"USER";
isdataat:29,relative;
content:!"|0a|", within 30;
```

4.3.9 dsize

```
#检查数据包有效载荷的长度
dsize:[<|>|=|!|<=|>=]size;
dsize:min_size{<>|<=>}max_size;

dsize:300<>400;
dsize:>10000;
dsize:<10;
```

4.3.10 pcre

pcre 的编写规则可以参考以下网址:
https://www.pcre.org/original/doc/html/pcrepattern.html

```
pcre:[!]"/pcre_string/[flag…]";
http_uri;
content:"/vulnerable_endpoint.php",fast_pattern,nocase;
pcre:"/[?&]interface=[\x60\x3b]/i";
```

注意事项：使用 pcre 时必须在前面使用 content 字段.

4.3.11 regex

regexp 的编写规则可以参考以下网址:
https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Global_Objects/RegExp

```
regex:"/regex_string/[flag…]"[,fast_pattern][,nocase];
http_uri;
regex:"/\x2fvulnerable_endpoint\x2ephp?interface=[\x60\x3b]/i",fast_pattern;
```

4.3.12 pkt_data

```
#检测经过规范化的原始数据
pkt_data;
content:"pizza", depth 5;

```

```
pkt_data;
content:"AAAAAA";
bufferlen:>1000;
```

注意:不能与其他经过解码的规则字段同时使用,例如 http_uri.

4.3.13 raw_data

```
#检测未经过规范化的原始数据,使用的较少,snort2 中是为了解决 telnet 的检测问题
raw_data;
content:"|FF F1|";
```

4.3.14 file_data

```
#检测网络流量中的文件内容
#目前支持的协议有:
http
pop3
imap
smtp
ftp-data
netbios-ssn
示例:
alert http (
  …
  flow:to_client,established;
  file_data;
  content:"<script>var aaaaaaa";
  …
)
alert file (
  …
  flow:to_client,established;
  file_data;
  content:"MZ",depth 2;
  …
)
alert tcp $EXTERNAL_NET any -> $HOME_NET 25 (
  …
  file_data;
  content:"decoded SMTP file here"
  …
)
alert tcp $EXTERNAL_NET any -> $HOME_NET 445 (
  …
  content:"|FE|SMB";
  content:"|08|", distance 8, within 1;
  file_data;
  content:"MALWARE";
  …
)
```

4.3.15 js_data

```
#该字段会将原始 js 数据经过"加强规范化"以后输出到缓冲区
[原始 js 数据]
<script>var o = {};
o.__defineGetter__('vuln');</script>

[js_data]
00000000  76 61 72 20 76 61 72 5F 30 30 30 30 3D 7B 7D 3B  var var_0000={};
00000010  76 61 72 5F 30 30 30 30 2E 5F 5F 64 65 66 69 6E  var_0000.__defin
00000020  65 47 65 74 74 65 72 5F 5F 28 27 76 75 6C 6E 27  eGetter__('vuln'
00000030  29 3B                                            );

示例:
js_data;
content:"=new Uint32Array(-1)|3B|";

js_data;
content:"var var_0000={}|3B|o.__defineGetter__(";

js_data;
content:"0xFFFFFFFF";
content:"-1";
bufferlen:<200;
```

4.3.16 vba_data

```
#检测 vba 脚本的内容

#使用该字段需要在配置文件中开启以下选项
http_inspect.decompress_zip = true
http_inspect.decompress_vba = true

```

```
示例:
vba_data;
content:"URLDownloadToFileA",nocase;
```

4.3.17 base64_decode&base64_data

```
#解码缓冲区的 base64 数据
base64_decode[:[bytes bytes][, offset offset][, relative]];

示例:
base64_decode;

base64_decode:relative;

http_header;
content:"Authorization:",nocase;
base64_decode:bytes 12, offset 6, relative;

#base64_data 将检测光标放在解码后的数据的首位
http_header;
content:"Authorization:",nocase;
base64_decode:bytes 12, offset 6, relative;
base64_data;
content:"NTLMSSP", within 8;
```

4.3.18 byte_extract

```
#该字段可以提取缓冲区中的数据放置到一个变量中，并在后续规则中调用
#该字段可选项较多,暂不展开，待后续补充示例
byte_extract:count, offset, name[, relative][, multiplier multiplier] \
             [, endian][, string[, {dec|hex|oct}]][, align align][, dce] \
             [, bitmask bitmask];
示例:
byte_extract:1, 0, str_offset;
byte_extract:1, 1, str_depth;
content:"bad stuff", offset str_offset, depth str_depth;

byte_extract:1, 0, multiplier_ex1, multiplier 8;
content:"AAAAA", within multiplier_ex1;
```

```
content:"MAGIC";


byte_extract:4, 0, field_sz, relative, little;
content:"next field", distance field_sz;
http_header;
content:"Content-Length: ";

byte_extract:4, 0, content_len, relative, string;
isdataat:!content_len;


byte_extract:4, 0, hex_string_var, string, hex;
content:"BBBBB", distance hex_string_var;
```

4.3.19 byte_test

```
#该字段主要是提取数据并与预置的数进行逻辑运算
byte_test:count, [!]operator, compare, offset[, relative][, endian] \
          [, string[, {dec|hex|oct}]][, dce][, bitmask bitmask];
示例:
content:"ABCD", depth 4;
byte_test:2, >, 0x7fff, 0, relative, little;

# grabs 2 bytes at offset 0, performs a
# bitwise AND with 0x3FF0, right-shifts the result by 4 bits,
# and compares the final result to 568 to see if they are equal
byte_test:2, =, 568, 0, bitmask 0x3FF0;

# grabs 4 bytes af offset 0, converts those bytes
# as a decimal string, and tests that the converted number
# is greater than 1234
byte_test:4, >, 1234, 0, string, dec;

content:"AAAA";
byte_extract:4, 0, a_sz, relative;
content:"BBBB";
# grabs 4 bytes right after "BBBB"
# and tests if those bytes are greater than
# the extracted bytes from earlier
byte_test:4, >, a_sz, 0, relative;
```

4.3.20 byte_math

```
#该字段使用较少,暂不赘述
byte_math:bytes count, offset offset, oper operator, rvalue rvalue, result result \
          [, relative][, endian endian][, string[, {dec|hex|oct}]][, dce] \
          [, bitmask bitmask];
示例:
# extracts 2 bytes at offset 0, multiplies the extracted number by 10,
# and stores the result in the variable "area", and then uses
# the resulting variable in a `byte_test` option
byte_math:bytes 2, offset 0, oper *, rvalue 10, result area;
byte_test:2, >, area, 16;

http_header;
content:"Content-Length: ",nocase;
# extracts 8 decimal string bytes immediately after "Content-Length: " (if they exist), multiplies
# the value by 10, and stores the result in "content_length"
byte_math:bytes 8, offset 0, oper *, rvalsue 10, result content_length, relative, string dec;

content:"ABCD";
# this is a valid byte_math option despite the odd ordering of the arguments
# extracts a single byte 10 bytes from the previous match, adds 10 to it,
# performs on it a bitwise AND with 0x12, and stores the result in "var"
byte_math:oper +, rvalue 10, offset 10, result var, bytes 1, bitmask 0x12, relative;
```

4.3.21 byte_jump

```
#该字段主要操控检测光标的位置跳转

# grab the 2 bytes at offset 0 and
# jump that many bytes forward
byte_jump:2,0;

content:"START";
byte_extract:1, 0, myvar, relative;
# grab a single byte 3 bytes after the previous
# byte_extract location, jump forward that number
# of bytes, and then adjust forward "myvar"
# number of bytes after the jump
byte_jump:1,3,relative,post_offset myvar;
content:"END", distance 6, within 3;

# grab 2 bytes at offset 1 from the current cursor location,
# bitmask AND the grabbed bytes by 0x03f0, jump the
# resulting number of bytes, and then adjust forward
# 2 number of bytes after the jump
byte_jump:2,1,relative,post_offset 2,bitmask 0x03f0;
byte_test:2,=,968,0,relative;

# this grabs 0 bytes so that it can
# jump backwards 6 bytes from the end of the
# current payload
byte_jump:0,0,from_end,post_offset -6;
content:"end..", distance 0, within 5;
```

4.3.22 ber_data & ber_skip

```
#通过指定 tlv 的数据的类型来检测对应的值中是否存在指定的特征
ber_data:type;

ber_data:0x02;
content:"|01 00 01|", within 3;


ber_data:0x30;
# 检查 TLV 中的 type 为 0x30 时,value 的第一个值是否为 0x02
ber_data:0x02;

#跳过指定类型的 tlv 数据，如果使用 optional 则该可以忽略类型
ber_skip:type[,optional];

ber_skip:0x02;
content:"|01 01 05|", within 3;

#可以忽略 type 不为 0x02 的情况,继续检测
ber_skip:0x02,optional;
content:"|01 01 05|", within 3;
```

4.3.23 ssl_state & ssl_version

```
#检测 ssl 协议所处的阶段
ssl_state:[!]{client_hello|server_hello|client_keyx|server_keyx|unknown}
          [,{client_hello|server_hello|client_keyx|server_keyx|unknown}]…;

ssl_state:client_hello;

ssl_state:client_keyx,server_keyx;

ssl_state:!server_hello;

#检测 ssl 协议版本号
ssl_version:[!]{sslv2|sslv3|tls1.0|tls1.1|tls1.2}
            [,{sslv2|sslv3|tls1.0|tls1.1|tls1.2}]…;
Examples
ssl_version:sslv3;

ssl_version:tls1.0,tls1.1,tls1.2;

ssl_version:!sslv2;
```

4.3.24 DCE/RPC

```
#以下字段和 DCE/RPC 协议中的标准名称类似,不过多赘述,详情可以查看 RFC 文档或 wireshark 对该协议的介绍

dce_iface:uuid uuid [, version [<|>|=|!|<=|>=]version][, any_frag];

dce_iface:uuid 4b324fc8-1670-01d3-1278-5a47bf6ee188;
dce_iface:uuid 4b324fc8-1670-01d3-1278-5a47bf6ee188, version <2;
dce_iface:uuid 4b324fc8-1670-01d3-1278-5a47bf6ee188, any_frag;
dce_iface:uuid 4b324fc8-1670-01d3-1278-5a47bf6ee188, version =1, any_frag;

dce_opnum:"{opnum|min_opnum-max_opnum}[ {opnum|min_opnum-max_opnum}]…";

dce_opnum:"15";
dce_opnum:"15-18";
dce_opnum:"15 18-20";
dce_opnum:"15 17 20-22";


dce_stub_data;
```

```

dce_stub_data;
byte_test:4,>,128,8,dce;
dce_stub_data;
pcre:"/^(\x00\x00\x00\x00|.{12})/s";
```

4.3.25 SIP

```
#sip 协议相关的内容
#请求方法:invite, cancel, ack, bye, register, options, refer, subscribe, update, join, info, message, notify, prack.

sip_method:[!]method[,method]…;

sip_method:invite, cancel;
sip_method:!invite;
sip_method:!invite;
sip_method:!bye;

#sip 的请求头和响应头
sip_header;

sip_header;
content:"CSeq";


#SIP 的所有有效载荷
sip_body;

sip_body;
content:"v=0|0D 0A|", within 5;

#sip 的响应状态码
sip_stat_code:stat_code[,stat_code]…;
#1-9 代表检测 1xx,2xx,3xx,4xx,等
#匹配 200
sip_stat_code:200;

# 匹配 2xx
sip_stat_code:2;

# 匹配 200 或 180
sip_stat_code:200, 180;
```

```

# match any 2xx SIP status codes or any 4xx SIP status codes
sip_stat_code:200, 180;
```

4.3.26 sd_pattern

```
#敏感数据检测
#内置三种
"credit_card",
"us_social"
"us_social_nodashes",

sd_pattern:"pattern"[, threshold count];
#匹配信用卡号
sd_pattern:"credit_card";
#匹配邮箱
sd_pattern:"\b\w+@ourdomain\.com\b";
#匹配三百次
sd_pattern:"This is a string literal", threshold 300;
```

4.3.27 asn1*

```
asn1:{bitstring_overflow|double_overflow|oversize_length length} \
     [, {bitstring_overflow|double_overflow|oversize_length length}]… \
     [, {absolute_offset|relative_offset} offset];

有效范围:
oversize_length: 0:4294967295
absolute_offset: 0:65535
relative_offset: -65535:65535

示例
asn1:oversize_length 10000, absolute_offset 0;
asn1:bitstring_overflow, double_overflow, relative_offset 7;
asn1:bitstring_overflow, double_overflow, oversize_length 499, relative_offset 7;
```

4.3.28 cvs

```
#目前 cvs 只有一种漏洞,因此该规则只有一种用法
cvs:option;

```

```
cvs:invalid-entry;
```

4.3.29 md5/sha256/sha512

```
#这三条规则用来检测某一段数据的哈希值与规则中所给的值是否匹配
md5
md5:[!]"|hash|", length length[, offset offset][, relative];
示例:
md5:"|7cf2db5ec261a0fa27a502d3196a6f60|", length 100, offset 0, relative;

sha256
sha256:[!]"|hash|", length length[, offset offset][, relative];
示例:
sha256:"|9ed1515819dec61fd361d5fdabb57f41ecce1a5fe1fe263b98c0d6943b9b232e|",
 \ length 100, offset 0, relative;

sha512
sha512:[!]"|hash|", length length[, offset offset][, relative];
示例:
sha512:"|d8fefb4255686e6bf365b0f4763fad983f624beb7cbbb59b617c745c346b8db51a8
70fe0a89cfba036cfbf2d011686b881acd8ab3278b318a304227ac2a99072|", \  length 1
00, offset 0, relative;
```

4.3.30 GTP*

```
#gtp 协议,通用分组无线服务隧道协议

#该值可以是一个整数,也可以是一个预定义的值,具体查看 snort_defaults.lua
gtp_info:info_element;
示例:
gtp_info:16;
gtp_info:packet_flow_id;

#用于检查特定消息类型的值
gtp_type:"type[ type]…";
示例:
gtp_type:"255 16";
gtp_type:"255 create_pdp_context_request";

#检查版本号
gtp_version:version_num;
示例:
gtp_version:1;
```

```
gtp_version:2;
```

4.3.31 DNP3*

```
#该选项用于检测 DNP3 中的函数映射值,可以使用整数或预定义的值具体可以查
看 dnp3_map.cc
dnp3_func:function_code;
示例
dnp3_func:0;
dnp3_func:confirm;

#检查 DNP3 的 flag
dnp3_ind:"indicator_flag[ indicator_flag ]…";
示例:
dnp3_ind:"config_corrupt event_buffer_overflow need_time";
dnp3_ind:"need_time";

#检查 DNP3 的对象标头,需要指定组号和变量号
dnp3_obj:group group_number, var varation_number;
Examples:
dnp3_obj:group 80, var 1;
dnp3_obj:group 60, var 2;

#检查 DNP3 的数据内容
dnp3_data;
示例:
dnp3_data;
content:"badstuff";
```

4.3.32 CIP*

```
#通用工业协议
#默认不开启该功能
#在 config 文件中添加:
-- enable the CIP inspector
cip = { }
-- add CIP binder entries
binder =
{
    { when = { proto = 'tcp', ports = '44818', role = 'server' }, use =
{ type = 'cip' } },
    { when = { proto = 'udp', ports = '22222', role = 'server' }, use =
{ type = 'cip' } },
#查找 CIP 属性
cip_attribute:[<|>|=|!|<=|>=]attribute;
cip_attribute:min_attribute{<>|<=>}max_attribute;
示例:
cip_attribute:5;
cip_attribute:>5;

#CIP 类型
cip_class:[<|>|=|!|<=|>=]class;
cip_class:min_class{<>|<=>}max_class;
示例:
cip_class:2;
cip_class:2<>5;

#查找 CIP 连接路径类型 0:65535
cip_conn_path_class:[<|>|=|!|<=|>=]conn_path_class;
cip_conn_path_class:min_conn_path_class{<>|<=>}max_conn_path_class;
示例:
cip_conn_path_class:10;
cip_conn_path_class:!0;

#查找 CIP 实例 0:4294967295
cip_instance:[<|>|=|!|<=|>=]cip_instance;
cip_instance:min_cip_instance{<>|<=>}max_cip_instance;
示例
cip_instance:33;
cip_instance:!33;

#查找 CIP 请求
cip_req;

#查找 CIP 响应
cip_rsp;

#查找 CIP 服务 0:127
cip_service:[<|>|=|!|<=|>=]service;
cip_service:min_service{<>|<=>}max_service;
示例:
cip_service:10;
cip_service:127;

#查找 CIP 状态码 0:255
cip_status:[<|>|=|!|<=|>=]status;
cip_status:min_status{<>|<=>}max_status;
```

```
示例:
cip_status:0;
cip_status:!0;

#查找 ENIP 命令 0:65535
enip_command:[<|>|=|!|<=|>=]enip_command;
enip_command:min_enip_command{<>|<=>}max_enip_command;
示例
enip_command:5<>100;
enip_command:<7;

#查找 ENIP 请求
enip_req;

#查找 ENIP 响应
enip_rsp;
```

4.3.33 IEC 104*

```
#默认不开启该功能
#config 文件中添加:
-- enable the IEC104 inspector
iec104 = { }
-- add the IEC104 binder entry
binder =
{
    { when = { proto = 'tcp', ports = '2404' }, use = { type = 'iec104' }, }
,
}

#查找 APCI 类型
iec104_apci_type:{u|U|unnumbered_control_function|s|S|numbered_supervisory_f
unction \
    |i|I|information_transfer_format};
示例
iec104_apci_type:unnumbered_control_function;
iec104_apci_type:S;
iec104_apci_type:i;

#查找函数名称
iec104_asdu_func:function_name;
示例
iec104_asdu_func:M_SP_NA_1;
iec104_asdu_func:m_ps_na_1;
```

4.3.34 Modbus*

```
# SCADA 网络中的常用协议
#默认不开启该功能:
-- enable the Modbus service inspector
modbus = {}
-- add the Modbus binder entry
binder =
{
    { when = { proto = 'tcp', ports = '502' }, use = { type = 'modbus' }, },
}

#匹配 modbus 数据
modbus_data;
示例
modbus_data;
content:"modbus stuff";

#匹配 modbus 函数
modbus_data;
示例:
modbus_data;
content:"modbus stuff";

#匹配 modbus 单元标识符
modbus_unit:modbus_unit_id;
示例:
modbus_unit:0;
modbus_unit:73;
```

4.3.35 S7*

```
#s7comm 和 S7commplus 是西门子中常用的 PLC 协议
#该功能默认不开启
-- enable the s7commplus service inspector
s7commplus = { }
-- add the s7commplus binder entry
binder =
{
    { when = { proto = 'tcp', ports = '102' }, use = { type = 's7commplus' }
, },
}
```

```

#查找内容
s7commplus_content;
示例
s7commplus_content;
content:"|01 02 03 04|";


#查找函数名/号
s7commplus_func:{function_code_name|function_code_number};
示例:
s7commplus_func:explore;
s7commplus_func:0x586;

#查找操作码
s7commplus_opcode:{opcode_code_name|opcode_code_number};
示例
s7commplus_opcode:request;
s7commplus_opcode:0x31;
```

4.4 无载荷检测规则

4.4.1 fragoffset

```
#查找 IP 层的段偏移值
fragoffset:[<|>|=|!|<=|>=]fragoffset;
fragoffset:fragoffset_min{<>|<=>}fragoffset_max;
示例:
fragoffset:0;
fragoffset:!0;
```

4.4.2 ttl

```
#查找 IP 层的 ttl 值
ttl:[<|>|=|!|<=|>=]ttl;
ttl:ttl_min{<>|<=>}ttl_max;
示例
ttl:64;
ttl:!64;
ttl:<3;
ttl:3<=>5;
ttl:=5;
```

4.4.3 tos

```
#查找 IP 层的 tos 值
tos:[<|>|=|!|<=|>=]tos;
tos:tos_min{<>|<=>}tos_max;
示例:
tos:!4;
tos:4;
```

4.4.4 id

```
#查找 IP 层的 id
id:[<|>|=|!|<=|>=]id;
id:id_min{<>|<=>}id_max;
示例
id:31337;
id:>31337;
```

4.4.5 ipopts

```
#查找 IP 层的可选项
#如下:
rr: Record Route
eol: End of Options List
nop: No Operation
ts: Time Stamp
sec: Security
esec: Extended Security
lsrr: Loose Source Routing
lsrre: Loose Source Routing (For MS99-038 and CVE-1999-0909)
ssrr: Strict Source Route
satid: Stream ID
any: Any IP options are set

ipopts:{rr|eol|nop|ts|sec|esec|lsrr|lsrre|ssrr|satid|any};
示例:
ipopts:rr;
ipopts:any;
```

4.4.6 fragbits

```
#查找 IP 层的 flags

M -> More Fragments
D -> Don't Fragment
R -> Reserved Bit
+ -> 至少存在一个
* ->  可有可无
! -> 不存在
Format:
fragbits:[modifier]fragbit…;
示例:
fragbits:M;
#M 和 D 标志位存在
fragbits:+MD;
```

4.4.7 ip_proto

```
#查看 IP 层所承载的协议
#参考: https://www.iana.org/assignments/protocol-numbers/protocol-numbers.xhtml
ip_proto:[!|>|<]{proto_number|proto_name};
示例:
ip_proto:igmp;
ip_proto:!tcp;
```

4.4.8 flags

```
#TCP 层的 flags
F -> FIN (Finish)
S -> SYN (Synchronize sequence numbers)
R -> RST (Reset the connection)
P -> PSH (Push buffered data)
A -> ACK (Acknowledgement)
U -> URG (Urgent pointer)
C -> CWR (Congestion window reduced)
E -> ECE (ECN-Echo)
0 -> No TCP flags set

flags:[modifier]test_flag…[,mask_flag…];
示例:
```

```
flags:S;
flags:SA;
flags:*SA;
# 检查 SYN 和 FIN 是否设置，忽略 CWR 和 ECN
flags:SF,CE;
```

4.4.9 flow

```
#判断流量的方向和状态等信息
to_client   发向客户端的数据
to_server   发向服务端的数据
from_client 从客户端发出的数据
from_server 从服务端发出的数据
established TCP:建立连接
not_established TCP:未建立连接
stateless   匹配任何状态的流
no_stream   匹配未重组的数据包
only_stream 匹配重组的数据包
no_frag 匹配分段的数据包
only_frag   匹配未分段的数据包

flow:[{established|not_established|stateless}] \
     [,{to_client|to_server|from_client|from_server}] \
     [,{no_stream|only_stream}] \
     [,{no_frag|only_frag}];
示例:
flow:to_server,established;
flow:to_client;
flow:to_server,established,no_stream;
flow:stateless;
```

4.4.10 flowbits

```
#设置流标志位

set 给当前流设定一个特定的状态
unset   取消特定的状态
isset    判断是否存在某个状态
isnotset    判断是否不存在某个状态
noalert 不告警

示例:
alert tcp any 143 -> any any (
    msg:"IMAP login";
    content:"OK LOGIN";
    flowbits:set,logged_in;
    flowbits:noalert;
)

#如果匹配"LIST"字符串并且 logged_in 标志为 1
alert tcp any any -> any 143 (
    msg:"IMAP LIST";
    content:"LIST";
    flowbits:isset,logged_in;
)


flowbits:isset,flag1&flag2;
flowbits:isset,flag1|flag2;
flowbits:set,flag1&flag2;
flowbits:unset,flag1&flag2;

#noalert,使规则不告警
flowbits:noalert;
```

4.4.11 file_type

```
#判断文件类型
file_type:"type_name[,type_version]…[ type_name[,type_version]…]…";

示例::
file_type:"PDF";
file_type:"PDF,1.6";
file_type:"PDF,1.6,1.7";
file_type:"MSEXE MSCAB MSOLE2";
```

4.4.12 seq

```
#判断 TCP 的 seq 序列号
seq:[<|>|=|!|<=|>=]seq;
seq:seq_min{<>|<=>}seq_max;
示例:
seq:0;
```

4.4.13 ack

```
#判断 TCP 的 ack 序列号
ack:[<|>|=|!|<=|>=]ack;
ack:ack_min{<>|<=>}ack_max;
示例:
ack:0;
ack:0<=>1000;
```

4.4.14 window

```
#判断 tcp 窗口的范围
window:[<|>|=|!|<=|>=]window;
window:window_min{<>|<=>}window_max;
示例:
window:55808;
window:0<>55808;
```

4.4.15 itype

```
#判断 icmp 协议的类型
itype:[<|>|=|!|<=|>=]itype;
itype:itype_min{<>|<=>}itype_max;
示例:
itype:>30;
```

4.4.16 icode

```
#判断 icmp 的码值
icode:[<|>|=|!|<=|>=]icode;
icode:icode_min{<>|<=>}icode_max;
示例:
icode:>30;
icode:0<=>30;
```

4.4.17 icmp_id

```
#判断 icmp 的 id 值
icmp_id:[<|>|=|!|<=|>=]icmp_id;
icmp_id:icmp_id_min{<>|<=>}icmp_id_max;
```

```
示例:
icmp_id:0;
```

4.4.18 icmp_seq

```
#判断 icmp 的 seq 序列号
icmp_seq:[<|>|=|!|<=|>=]icmp_seq;
icmp_seq:icmp_seq_min{<>|<=>}icmp_seq_max;
示例
icmp_seq:0;
```

4.4.19 rpc

```
#判断 rpc 协议等相关信息
rpc:application_number[,{version_number|*}][,{procedure_number|*}];
示例:
#应用号为 100000,版本号为任意,函数号为 3
rpc:100000, *, 3;
```

4.4.20 stream

```
#用于决定匹配中的流量是否要流重组
stream_reassemble:action {enable|disable}, direction {server|client|both} \

                    [, noalert][, fastpath];
示例:
#响应数据中存在 ABCDEF 时,禁用流重组,且不产生告警
flow:to_client,established;
content:"ABCDEF";
stream_reassemble:action disable, direction client, noalert;
```

4.4.21 stream_size

```
#检查流的大小
stream_size:[<|>|=|!|<=|>=]bytes[,{either|to_server|to_client|both}];
stream_size:min_bytes{<>|<=>}max_bytes[,{either|to_server|to_client|both}];

示例:
stream_size:=125,to_server;
stream_size:0<>100,both;
```

4.5 检测后规则

4.5.1 detection_filter

```
#判断生成告警生成事件的次数
detection_filter:track {by_src|by_dst}, count c, seconds s;
示例:
#同一个 SIP 在 60s 内登录 30 次 ssh,则产生告警
flow:established,to_server;
content:"SSH",nocase,offset 0,depth 4;
detection_filter:track by_src, count 30, seconds 60;
```

4.5.2 replace

```
#覆盖匹配中的内容
replace:"string";
示例:
content:"ABCD";
replace:"EFGH";
```

4.5.3 tag

```
tag:{session|host_src|host_dst}, {packets packets|seconds seconds|bytes bytes};
示例:
# 标记会话中的下面 10 个数据包
tag:session, packets 10;
# 标记源 IP 的下一个 4000 字节
tag:host_src, bytes 4000;
```

4.6 共享对象规则

```
#共享对象规则
共享对象规则用 C++编写，并由 Snort 在运行时加载。
以下是一个简单的 Snort3 SO 规则示例：

```

```
#include "main/snort_types.h"
#include "framework/so_rule.h"

using namespace snort;

static const char* rule_1337 = R"[Snort_SO_Rule](
alert tcp any any -> any any (
    msg:"SERVER-OTHER SO Example";
    soid:1337;
    flow:to_server,established;
    content:"foo",nocase;
    so:eval;
    gid:3;
    sid:1337;
)
)[Snort_SO_Rule]";

static const unsigned rule_1337_len = 0;

static IpsOption::EvalStatus rule_1337_eval(void*, Cursor&, Packet*)
{
    return IpsOption::MATCH;
}

static SoEvalFunc rule_1337_ctor(const char* /*so*/, void** pv)
{
    *pv = nullptr;
    return rule_1337_eval;
}

static const SoApi so_1337 =
{
    {
        PT_SO_RULE,
        sizeof(SoApi),
        SOAPI_VERSION,
        0, // version
        API_RESERVED,
        API_OPTIONS,
        "1337",
        "SERVER-OTHER SO Example",
        nullptr,
        nullptr
    },
    (uint8_t*)rule_1337,
    rule_1337_len,
    nullptr,         // pinit
    nullptr,         // pterm
    nullptr,         // tinit
    nullptr,         // tterm
    rule_1337_ctor, // ctor
    nullptr          // dtor
};

const BaseApi* pso_1337 = &so_1337.base;

SO_PUBLIC const BaseApi* snort_plugins[] =
{
    pso_1337,
    nullptr
};
```