باسمه تعالی

راهنمای نرم افزار Snort

نسخه 2.6.0

به همراه مقدمه ای پیرامون سیستم های کشف مزاحمت مبتنی بر شبکه

ابراهیم مولویان جزی

ebrahim.molavian@gmail.com

آزمایشگاه امنیت شبکه

دانشکده برق و کامپیوتر دانشگاه صنعتی اصفهان

فهرست مطالب

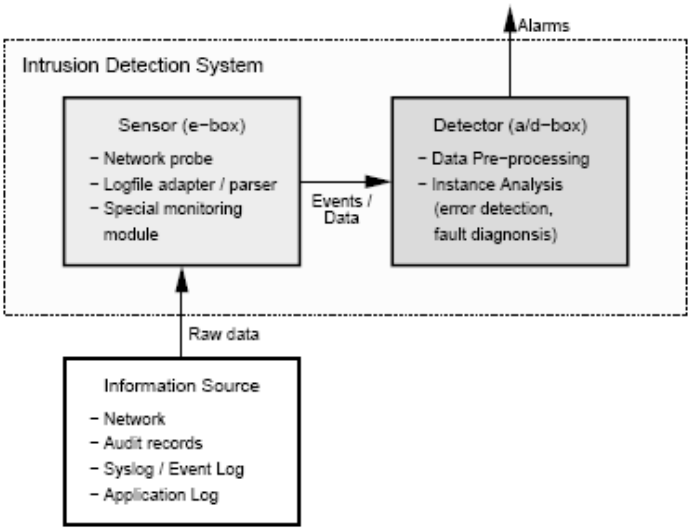
۱) آشنایی با سیستم های کشف مزاحمت

به طور کلی مزاحمت یا تهاجم به فعالیتی گفته می شود که از خارج یا داخل شبکه صورت گرفته و هدف آن نقض یکی از جنبه های امنیتی (CIA) در شبکه، یعنی محرمانگی (Confidentiality)، تمامیت (Integrity) و دسترس پذیری (Availablity) است. در این صورت عملیات کشف مزاحمت عبارتست از فرآیند نظارت و بررسی (monitoring) کامپیوترها و شبکه‌ها به منظور کشف نفوذها و فعالیتهای غیر مجاز و دستکاری در فایلها. بر این اساس، سیستم‌های کشف مزاحمت (Intrusion Detection Systems) یا IDS- ها یکی از مهمترین ابزارهای امنیتی مورد استفاده در شبکه‌ها و مکمل سایر فناوری های امنیتی هستند. دو کاربرد مهم سیستم های کشف مزاحمت عبارتند از:

- ❖ فراهم کردن اطلاعات مربوط به مزاحمت و ارائه آن به مدیر شبکه طوری که حملات پیش بینی شده یا پیش بینی نشده به وسیله ابزارهای امنیتی دیگر مثل Firewall یا Service wrapper کشف شده و از آنها جلوگیری شود.
- ❖ ثبت مزاحمت های ایجاد شده برای ماشین، طوری که این اطلاعات فرم حقوقی و قضایی به خود گرفته و امکان تشخیص و تعقیب مهاجمان برای سازمان ها فراهم شود.

سیستم‌های IDS بر اساس خط مشی‌های (policy) امنیتی تعریف شده، رخدادهای (event) غیر مجاز را تشخیص داده و بر اساس آن یک پیغام هشدار (alert) به مسئول شبکه یا مسئول سیستمِ IDS ارسال می کنند (مثلاً به صورت فراخوانی page، نامه الکترونیکی E-mail، SNMP trap و ...). باید دقت کرد که برخلاف firewall ها که وظیفه‌ی اصلی آنها ممانعت از نفوذگری(prevention) است، IDS ها بیشتر به کار شناسایی (detection) می‌پردازند. با این وجود، سیستم‌های IDS گاهی نسبت به مزاحمت‌ها واکنش(respond) هم نشان می‌دهند. عکس العمل‌های یک IDS مواردی نظیر قطع ارتباط کاربرِ مزاحم، غیر فعال کردن شناسه کاربری مهاجم و قرار دادن او در لیست سیاه، و "کشتن" مهاجم با ارسال بسته های ساختگی برای او خواهد بود.

یک چارچوب مشترک برای سیستم کشف مزاحمت، CIDF یا Command Intrusion Detection Framework است که به این صورت بیان می شود:



مدل کلی یک سیستم کشف مزاحمت (IDS)

منبع اطلاعات (Information Source) که مثلاً خطوط ارتباطی شبکه است، داده های خام (raw data) را در اختیار بخش Sensor یا حسگر (E-box =Event Generator) می گذارد و این بخش با یک بررسی (مثلاً مقایسه داده های خام با یک الگوی از پیش معلوم) موارد مشکوک را گزارش می کند. سپس این رخدادها (events) به بخش آشکارساز (Detector) می رود. این بخش خود شامل موتورهای تحلیل (A-box = Analysis Engine) و مکانیزم های ذخیره سازی (D-box = Storage Mechanisms) می باشد. در موتور تحلیل، داده ها بر اساس روش های آماری، تحلیلهای نموداری و حتی مدلهای ایمن سازی بیولوژیکی، مورد پردازش قرار می گیرند و حملات علیه سیستم تحت عنوان آژیر (alarm) اعلام می شود. به علاوه با کمک ذخیره سازی در D-box ، امکان بررسی یا گزارش آژیرها و موارد مشکوک در زمان های بعد نیز فراهم می شود. بلوک مقابله (C-box = Countermeasure) امکان قطع ارتباط TCP یا تغییر لیست مسیرها (router) را فراهم می کند تا پس از آشکارشدن حملات اولیه، حملات بعدیِ دیگری رخ ندهند.

IDS ها در یک طبقه بندی، به دو نوع "مبتنی بر میزبان" (Host-based IDS) یا HIDS و "مبتنی بر شبکه" (Network-based IDS) یا NIDS تقسیم می‌شوند. سیستم‌های HIDS با نظارت از نزدیک روی سیستم عامل، داده‌های موجود روی هر کامپیوتر را جداگانه بررسی کرده و اطلاعات صریح و مناسبی پیرامون اینکه "آیا یک فایل دستکاری شده است" یا اینکه "چه کسی در چه زمانی چه کاری را به چه مقصدی انجام داده است" و مواردی از این دست، در اختیار می‌گذارند. این اطلاعات و گزارش‌ها از طریق System-logها یا

audit-logها یا event-logهای سیستم عامل یا برنامه‌های کاربردی ارسال می‌شوند. برخی از HIDS های مورد استفاده عبارتند از:

Cyborcop ، POLYCENTER Security Intrasion Director، Kane Security Monitor Intruder Alert ، Computer Misuse Detection System ، Tripwire، Monitor

اما نوع سیستم های کشف مزاحمتی که نرم افزار Snort در آن دسته قرار می‌گیرد، نوع دوم IDSها یعنی سیستم‌های کشف مزاحمت مبتنی بر شبکه (NIDS) هستند. یک NIDS با کمک بخش ناظر خود (monitor یا Sniffer یا Sensor) به پویش شبکه و بررسی ترافیک آن پرداخته و با استفاده از تکنیک‌هایی مثل pocket Sniffing (کالبد شکافی بسته‌ها)، داده‌ها را از درون بسته‌های TCP/IP در حال رفت و آمد در شبکه استخراج می‌کند و سپس بخش دومِ NIDS یعنی نرم افزار عامل (agent) اطلاعات را به صورت بازخورد به ناظر ارسال می‌کند. (ممکن است در برخی NIDSها بخشی به نام Management Console پیش بینی شود که نقش آن، دریافت امن گزارش از ناظر و تبادل اطلاعات پیکر بندی است.) NIDS ها در برابرحمله DoS یا Denial of Service و دسترسی‌های غیر مجاز خارجی عملکرد خوبی دارند و چون در سطح بسته‌ها فعالیت می‌کنند، به نوع سیستم عامل بستگی ندارند. البته آنها در سرعت‌های انتقال بسته زیاد (100 Mbps و بالاتر)، بسته‌ها را از دست می‌دهند و در حالت وجود بسته‌های رمز شده، از شناسایی الگوهای حمله باز می‌مانند. از معروفترین انواع NIDS می‌توان به این موارد اشاره کرد :

Seosion Wall-3 – Cybercop Monitor – ISS RealSecure – Net - NFR - Shadow Ranger – Net Prowler Dragon – Snort

۴ تکنیک اصلی که IDS ها برای کشف مزاحمت استفاده می کنند، عبارتند از :

۱ ـ تشخیص نمونه‌های غیر متعارف (Anomaly Detection): یافتن رخدادهایی که تعداد دفعات وقوع آنها به طور غیر عادی زیاد باشد؛ مثلاً ۲۰ بار login کردن در یک شبانه روز.

۲ ـ کشف از روی نشانه (Signature Detection): یافتن رخدادها یا رفتارهای غیر مجاز بر اساس دسته‌ای از

الگوهای شناخته شده از رفتارهای مجاز (Signature) ؛ مثلاً اجرای یک FTP غیر عادی.

۳ ـ نظارت بر هدف (Target Monitoring): بررسی برای یافتن دستکاری‌های احتمالی در برخی فایل‌ها (مثلاً بر اساس تابع در هم hash function) و کنترل تصحیحی آنها (انجام معکوس عمل دستکاری).

۴ ـ کاوش‌های نهانی (Stealth probe): جمع آوری مقدار زیادی داده از سیستم و تلاش برای یافتن آثار یک حمله‌ی ساخت یافته و دارای اسلوب که در طول یک بازه زمانی طولانی (مثلاً چند ماه) انجام می‌گیرد.

۲) Snort چیست ؟

نرم افزار Snort به عنوان یک NIDS، نوشته ی آقای Martin Roesch در سال ۱۹۹۸ است و پس از افتتاح شرکت Sourcefire توسط او، این شرکت به طور رسمی پشتیبانی و به روز رسانیِ نرم افزار را به عهده گرفته است. (البته در یک تقسیم بندی دقیقتر، Snort متعلق به نسل سوم IDSها یعنی سیستم‌های کشف مزاحمت مبتنی بر منابع ناهمگون(مبتنی بر agent یا عامل) است که اطلاعات را هم از میزبان و هم از شبکه جمع آوری می‌کنند.) نسخه اصلی آن مبتنی بر Linux/Unix نوشته شده است ولی نسخه‌های مبتنی بر Win 32 آن نیز موجود است. از ویژگی‌های مهم Snort می‌توان به این موارد اشاره کرد :

- ❖ رایگان بودن
- ❖ log کردن بسته‌ها و آنالیز real-time (بلادرنگ) از ترافیک شبکه
- ❖ بررسی داده‌های اصلی و مهم یک بسته و بررسی عواقب احتمالی ورود مجموعه‌ای این داده‌ها به سیستم
- ❖ جستجو برای یافتن یک بخش خاص از یک بسته تحت پروتکل خاص(آنالیز پروتکل)
- ❖ شناسایی حملات مختلف (سرریز بافر - پویش مخفی پورت‌ها- حملات CGI - حمله به پروتکل SMB - حمله تشخیص نوع سیستم عامل - حمله منع سرویس)
- ❖ تشخیص اجزای کوچک یک بسته اطلاعاتی و کشف و ارائه آمار نمونه‌های مشکوک
- ❖ کار کردن با بسته های IP قطعه سازی شده و نیز امکان گردآوری مجدد و بازسازی رشته ی بسته های TCP
- ❖ ارسال پیام‌های هشدار به صورت بلادرنگ به Syslog یا پنجره pop-up یا هر فایل دیگر

❖ log کردن رخدادها به صورت XML یا پایگاه داده

❖ وجود یک زبان انعطاف پذیر برای تعریف قوانینِ ثبت رخدادها

❖ قابلیت انعطاف، گسترش و پذیرش الحاقات و pluginها

۳) معماری Snort

ساختار Snort از سه بخش اصلی تشکیل شده است :

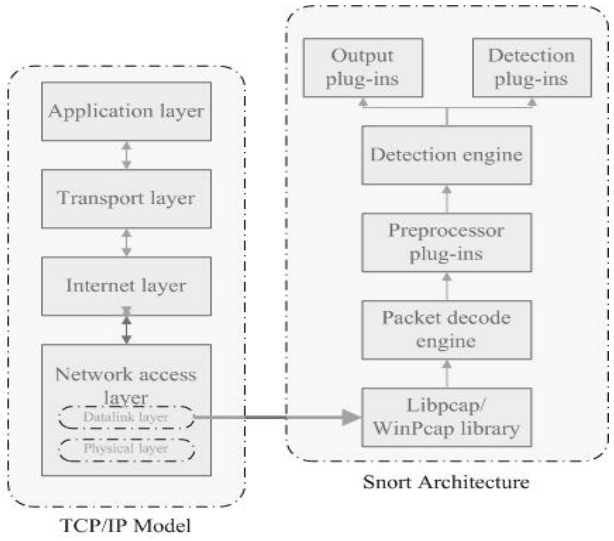
۱ـ Packet decoder یا کد بردار بسته: این بخش، اطلاعات هر بسته‌ی شبکه را به فرمت مناسبی که قابل تشخیص در موتور تشخیص باشد، تبدیل می‌کند. این فرمت‌ها عبارتند از: SLIP (استاندارد اتصال به اینترنت از طریق تلفن و مودم) Ethernet (استاندارد اتصال کامپیوترها در یک شبکه محلی) و ppp (استاندارد اینترنت برای اتصالات سریال)

۲ ـ preprocessor یا پیش پردازشگر: این بخش شامل یک سری از pluginهای قابل الحاق به نرم افزار است که رفتار آن را تغییر می دهند و از آنجا که عملیات پردازشی آنها پیش از کار موتور تشخیص اصلی انجام می شود، به آنها «پیش پردازشگر» گفته می شود. در واقع Snort در حالت پایه بی حالت(stateless) است و با اضافه شدن این پیش پردازشگرها، امکان شکل دهی به داده ها (data formatting) و حالت دهی به نرم افزار (making snort more stateful) فراهم می شود. در حال حاضر لیست پیش پردازشگرهای قابل الحاق به Snort به این شرح است: portscan detector (آشکارساز پویش پورت) – portscan ignorehosts – frag2 – stream4 – flow – flow-portscan – telnet decode (کدگشایی telnet) – rpc decode (کدگشایی rpc) – performance monitor (ناظر اجرا) – HTTP inspect (بازرسی HTTP) – back orifice detector (آشکارساز روزنه پشتی). توجه کنید که با ویرایش فایل snort.conf می توان هر یک از این پیش پردازشگرها را فعال یا غیر فعال کرد.

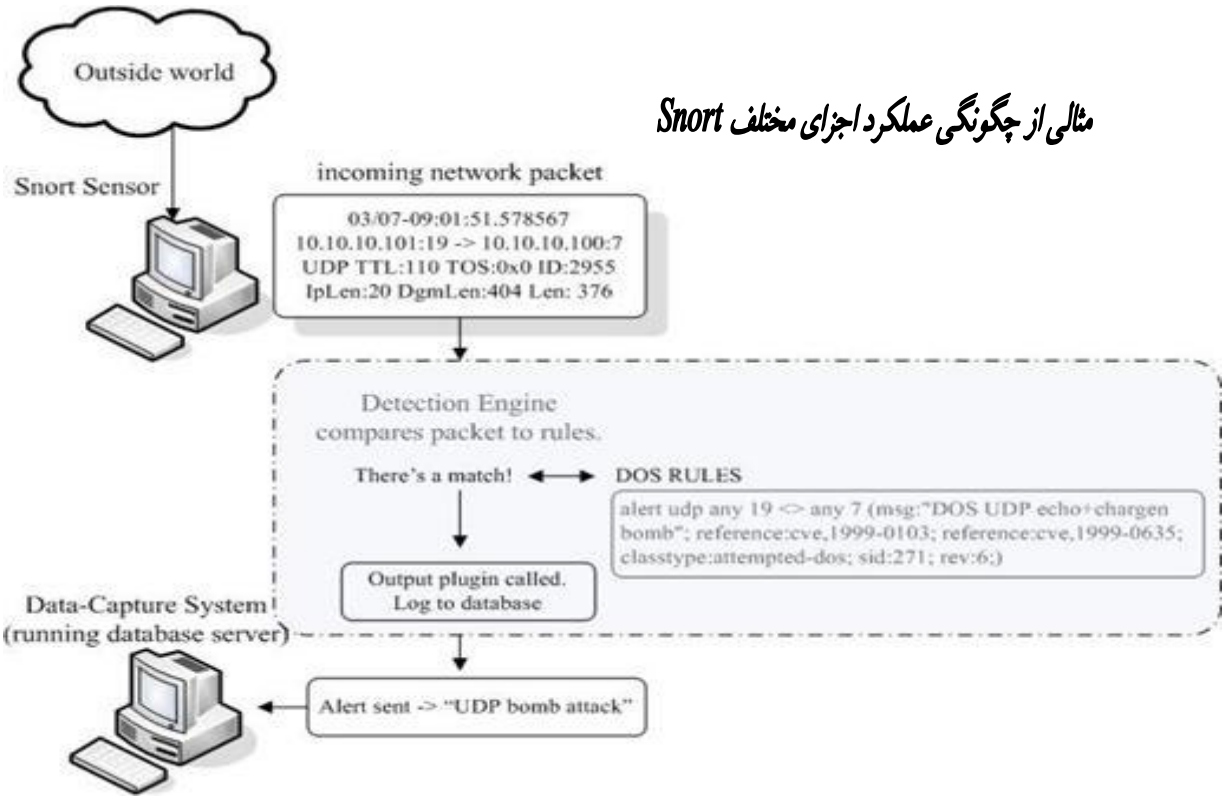
۳ـ detection engine یا موتور تشخیص: این بخش، هر بسته ورودی را با تمامی قوانین موجود در پایگاه داده Snort (که در هنگام اجرای نرم افزار در حافظه بارگذاری شده‌اند) تطبیق می‌کند و با یافتن اولین مورد تطبیق

وضعیت بسته با یک قانون (الگوی مزاحمت)، عمل ذکر شده در قانون مربوطه را اجرا می‌کند. موتور تشخیصِ Snort، قابلیت اضافه کردن pluginها و ماژول های جدید را دارد که قدرت تحلیل آن را افزایش می‌دهد.

۴ـ Logger یا رخداد نگار (Alerter یا هشدار دهنده): این بخش به ثبت رخدادها در آدرس پیش فرض var/log/snort/ و ثبت هشدارها در آدرس پیش فرض var/log/snort/alerts/ می پردازد و امکان نمایش اطلاعات ثبت شده از ترافیک شبکه را (با فرمت قابل فهم برای انسان یا فرمت tcpdump) فراهم می کند.



ساختار کامل Snort



مثالی از چگونگی عملکرد اجزای مختلف Snort

٤) دریافت و نصب Snort

نرم افزار Snort را می‌توان از سایت این نرم افزار www.snort.org دریافت کرد. آخرین نسخه ارائه شده از آن، در زمان نوشته شدن این متن، نسخه 2.6.0.2 می‌باشد. آنچه ذیلاً ارائه می شود، جهت نصب نرم افزار روی سیستم عامل Linux توزیع SuSE9.x و یا SuSE10 فراهم شده است.

۴-۱) بسته های پیش نیاز برای نصب

پس از ورود به سیستم به عنوان root (که در کلیه مراحل نصب لازم است) برای نصب بسته‌های پیش نیاز نصب، شاخه جدیدی ایجاد می کنیم و تمام بسته ها را در آن شاخه قرار می دهیم. سپس عملیات نصب را از آن‌جا اجرا می نماییم. به طور مثال فرض کنید بسته‌ها را در /usr/src/snort قرار داده‌ایم. در خط فرمان با دستور زیر وارد شاخه مربوطه می شویم:

```
#cd /usr/src/snort
```

برای کمپایل کردن و نصب صحیح نرم افزار و ایجاد تمام امکانات لازمِ بعدی، به کتابخانه ها و بسته های زیر احتیاج است: (اکثر فایل‌ها همراه توزیع SuSE در CDهای نصب SuSE وجود دارد. همچنین منظور از + که در انتهای نام بسته‌ها قرار دارد، نسخه ذکرشده یا نسخه‌های بالاتر بسته است.)

autoconf	automake	bison	flex
gcc/gcc++	gettext	glibc-devel	
libtool	make	libstd++-devel	

MySQL-4.0.x+	MySQL-devel-4.1.10a-3+	MySQL-shared4.0.x+
MySQL-client-4.0.x+	PHP-MySQL4.0.x+	MySQL-shared-compat-4.1.18
pcre-6.6+	Libnet-1.0.2a	libpcap 8.3.3+

ما جهت نصب Snort و بسته های پیش نیاز از Yast کمک خواهیم گرفت. Yast یک ابزار قدرتمند جهت مدیریت نصب بسته های نرم افزاری توزیع suse است که در گزینه system از منوی پانل قرار دارد. از طریق

Yast می‌توان بسته های موجود درCDهای suse و نیز هر بسته دلخواه RPM را نصب کرد. توجه کنید که برای اطمینان از وجود بسته های فوق در Yast به آیکون install and remove software رجوع کرده و در لیست filters گزینه selection را انتخاب کرده و سپس از لیست نمایش‌داده‌شده C/C++ Compilers and tools را انتخاب می کنیم.

چنانچه برخی از بسته‌های RPM فوق همراه توزیع SuSE نباشند، کافی است روی بسته RPM کلیک کرده و سپس install from Yast را کلیک نماییم. بدین ترتیب Yast بسته مورد نظر را طی مراحلی نصب می‌کند و به لیست بسته‌های نصب‌شده سیستم اضافه می‌نماید. توجه کنید که نصب بسته mysql-devel الزامی است. همچنین در صورتیکه نسخه mysql ، 4.0.x یا پایین‌تر باشد غیر از mysql-shared که همراه suse نصب می‌شود، mysql-shared-compat را نیز باید نصب کرد.

۲-۴) روند نصب و تنظیمات Snort

با مراجعه به صفحه اصلی سایت رسمی نرم افزار به آدرس www.snort.org می توان آخرین نسخه نرم افزار را (در اینجا 2.6.0.2) به علاوه ی حجم زیادی از راهنماهای مربوطه دریافت کرد. همچنین با کلیک کردن روی لینک مربوط به قوانین (RULES) می توان قوانین مربوط به آن نسخه را دریافت کرد. در زمان دریافت فایل قوانین، باید دقت کرد که نسخه قوانین دریافتی با نسخه ی در حال اجرا از نرم افزار Snort هماهنگ بوده و قوانین، فایل snort.conf متناسب را در بربگیرد. البته قوانین رایگانِ در دسترس، مربوط به نسخه های قبلی نرم افزار می باشد (که البته ممکن است به ندرت مشکل ساز نیز بشود). اما قوانین کاملِ مربوط به آخرین نسخه، تنها در اختیار مشترکین پولی سایت (subscribers) قرار می گیرد. همچنین با پر کردن یک فرم ثبت نام ساده، می توان علاوه بر امکان استفاده از گروه های خبری رسمی نرم افزار، به عنوان ثبت نام کننده (registered users)، یک نسخه پایه (ولی غیر کامل) از قوانین را دریافت کرد.

حال جهت نصب کردن Snort دستورات زیر را اجرا می کنیم :

```
# tar -zxvf snort-2.6.0.2.tar.gz
# cd snort
#./configure --enable-smbalerts --enable-flexrep --with-mysql
--with-snmp --with-openssl
# make
# make install
# cd ..
```

دقت کنید که برای نصب snort با mysql گزینه --with-mysql ضروری است. همچنین قرار دادن گزینه ی --enable-flexresp در صورتی میسر است که libnet قبلاً نصب شده باشد. به علاوه اگر تمام بسته‌های لازم mysql از جمله mysql-devel قبل از این مرحله نصب نشده باشند، در این مرحله با مشکل مواجه خواهیم شد.

پس از انجام عملیات نصب، باید تغییرات لازم را در فایل snort.conf که فایل تنظیمات snort است، اعمال کنیم. ابتدا یک گروه و کاربر مختص snort در سیستم ایجاد می کنیم:

```
# groupadd snort
# useradd -g snort snort
```

حال snort را برای استفاده از قوانین تنظیم می کنیم. تمام فایل‌های قوانین موردنظرتان را در شاخه‌ای با نام rules قرار می دهیم:

```
# mkdir /etc/snort
# cp ./rules /etc/snort
# cp /etc/snort.conf /etc/snort
# cp /etc/*.config /etc/snort
# cp /etc/unicode.map /etc/snort
# vi /etc/snort/snort.conf
```

در فایل /etc/snort/ snort.conf یک سری تغییرات اعمال می کنیم. ابتدا آدرس شبکه داخلی خود را مشخص می کنیم. به عنوان مثال:

```
Var HOME_NET 192.168.4.0/24
```

آدرس شاخه قوانین را مشخص می کنیم:

```
Var RULE_PATH ./rules
```

در نهایت نوع ماژول خروجی را تعیین می کنیم که در اینجا قصد داریم از پایگاه داده MySql استفاده کنیم:

```
Output database: log,mysql,user=snort
password=your_snortDB_pass dbname=snort
localhost=localhost
```

پایگاه داده و کاربر مربوطه را که در این مرحله "snort" قرار داده‌ایم، در مراحل بعد ایجاد خواهیم کرد.

۴-۳) تنظیمات جداول پایگاه داده

پس از نصب MySql لازم است تنظیماتی را برای استفاده از آن اعمال کرده واین سرویس را فعال کنیم:

```
#/usr/bin/mysql_install_db
# mysqld -u root
```

تنظیم کلمه عبور ریشه برای mysql :

```
#/usr/bin/mysqladmin -u root password 'mysql_root_pass'
```

برای این که هنگام بوت سیستم این سرویس فعال شود، فرمان زیر را اجرا می کنیم:

```
#chkconfig mysql on
```

حال برای **اجرای MySql** فرمان زیر را در خط فرمان اجرا می کنیم:

```
# mysql -u root -p
```

با این کار، کلمه رمز پرسیده خواهد شد که در اینجا همان کلمه رمزی را که با دستور mysqladmin ثبت کردیم، وارد می کنیم:

```
password: 'mysql_root_pass'
```

پس از وارد کردن کلمه رمز، اعلانِ Mysql ظاهر می‌شود. اکنون پایگاه داده snort را ایجاد می کنیم و دسترسی‌های لازم را تنظیم می کنیم:

```
mysql> create database snort;
mysql> grant insert,select on root.* to
    snort@localhost;
mysql> set password for snort@localhost =
     password('mysql_snortDB_pass ');
```

```
mysql> grant create,insert,select,delete,update on
         snort.* to snort@localhost
mysql> grant create,insert,select,delete,update on
     snort.* to snort
mysql> exit;
```

در این مرحله برای ایجاد جداول مربوط به snort از فایلی که همراه بسته snort ارائه می‌شود، استفاده می کنیم.

دستور زیر را در خط فرمان اجرا می کنیم:

```
# mysql -u root -p < ./schemas/create-mysql/snort
```

اکنون با ورود به MySql و اجرای فرمان‌های زیر، می‌توانیم جداول ایجادشده را مشاهده کنیم:

```
mysql> use snort;
mysql> show tables;
  +------------------+
  | Tables_in_snort  |
  +------------------+
  | data             |
  | detail           |
  | encoding         |
  | event            |
  | icmphdr          |
  | iphdr            |
  | opt              |
  | reference        |
  | reference_system |
  | schema           |
  | sensor           |
  | sig_class        |
  | sig_reference    |
  | signature        |
  | tcphdr           |
  | udphdr           |
  +------------------+
  16 rows in set (0.00 sec)
```

۵) مدهای کاری Snort و نحوه اجرای نرم افزار

نرم افزار Snort دارای چهار مد کاری شامل: Packet Sniffer (برای دریافت و رمز گشایی بسته‌های در حال عبور در شبکه و و نمایش روی console یا صفحه نمایش)، Packet logger (برای ثبت و ضبط تمام بسته‌ها در disk یا حافظه)، Full Intrusion Detection (برای تشخیص نفوذ کامل)، In line (برای عکس العمل به اقدامات مشکوک کشف شده) می باشد. هر یک از این مدهای کاری با یک سری دستور خاص فعال می شود.

۵-۱) مد کاری Packet Sniffer

Snort در این مد کاری تنها به کار مشاهده بسته های در حال رفت و آمد در شبکه محلی خود می پردازد و بدون آنکه عملکرد خاصی در مورد آنها داشته باشد، صرفاً این ترافیک را نمایش می دهد. فرمت خروجی در این حالت، متن عادیِ ASCII است. گزینه ها و دستورات مربوط به این مد کاری به شرح زیر است :

- v- برای نمایش سرآیندهای بسته‌های TCP/IP روی صفحه؛ با دستور: #snort -v

خروجی نرم افزار در این حالت به شکل زیر است:

```
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+

09/19-12:42:30.502747 172.16.2.157:137 -> 172.16.2.255:137
UDP TTL:128 TOS:0x0 ID:3177 IpLen:20 DgmLen:78
Len: 50
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+

09/19-12:43:16.507051 172.16.2.225:9565 -> 172.16.2.129:445
TCP TTL:64 TOS:0x0 ID:61610 IpLen:20 DgmLen:182 DF
***AP*** Seq: 0x714B23BE  Ack: 0xAA963DDC  Win: 0x7CC  TcpLen: 32
TCP Options (3) => NOP NOP TS: 1692111 156991
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+

09/19-12:46:27.345219 ARP who-has 172.16.2.114 tell 172.16.2.225

09/19-12:46:27.993287 172.16.2.225 -> 172.16.2.129
ICMP TTL:64 TOS:0x0 ID:1 IpLen:20 DgmLen:84 DF
Type:8  Code:0  ID:50459   Seq:2  ECHO
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
```

- d- برای نمایش داده‌های کاربردی بسته ها (payload) علاوه بر سرآیند بسته با دستور: #snort -vd

خروجی نرم افزار در این حالت به شکل زیر است:

```
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+

09/19-14:14:25.915957 172.16.2.225:16190 -> 172.16.2.129:445
TCP TTL:64 TOS:0x0 ID:30873 IpLen:20 DgmLen:182 DF
***AP*** Seq: 0xC89FD735  Ack: 0xE11D5928  Win: 0x6C0  TcpLen: 32
TCP Options (3) => NOP NOP TS: 3059378 211683
00 00 00 7E FF 53 4D 42 75 00 00 00 00 08 01 D8  ...~.SMBu.......
00 00 00 00 00 00 00 00 00 00 00 00 00 00 81 18  ................
01 A0 03 00 04 FF 00 00 00 00 00 01 00 53 00 00  .............S..
5C 00 5C 00 31 00 37 00 32 00 2E 00 31 00 36 00  \.\.1.7.2...1.6.
2E 00 32 00 2E 00 31 00 32 00 39 00 5C 00 48 00  ..2...1.2.9.\.H.
50 00 20 00 4C 00 41 00 53 00 45 00 52 00 4A 00  P. .L.A.S.E.R.J.
45 00 54 00 20 00 31 00 33 00 30 00 30 00 20 00  E.T. .1.3.0.0. .
50 00 43 00 4C 00 20 00 36 00 00 00 3F 3F 3F 3F  P.C.L. .6...????
3F 00                                            ?.

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+

09/19-14:14:27.966949 ARP who-has 172.16.2.131 tell 172.16.2.129

09/19-14:14:30.284662 ARP who-has 172.16.2.142 tell 172.16.2.182

09/19-14:14:32.700785 172.16.2.195:138 -> 172.16.2.255:138
UDP TTL:128 TOS:0x0 ID:4096 IpLen:20 DgmLen:229
Len: 201
11 02 80 23 AC 10 02 C3 00 8A 00 BB 00 00 20 45  ...#.......... E
50 45 4E 45 50 45 50 45 4E 45 4A 43 41 43 41 43  PENEPEPENEJCACAC
41 43 41 43 41 43 41 43 41 43 41 43 41 43 41 00  ACACACACACACACA.
20 45 46 45 44 45 46 45 45 46 41 46 45 43 41 43   EFEDEFEEFAFECAC
41 43 41 43 41 43 41 43 41 43 41 43 41 43 41 42  ACACACACACACACAB
4E 00 FF 53 4D 42 25 00 00 00 00 00 00 00 00 00  N..SMB%.........
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 00 11 00 00 21 00 00 00 00 00 00 00 00 00 E8  .....!..........
03 00 00 00 00 00 00 00 00 21 00 56 00 03 00 01  .........!.V....
00 00 00 02 00 32 00 5C 4D 41 49 4C 53 4C 4F 54  .....2.\MAILSLOT
5C 42 52 4F 57 53 45 00 01 00 80 FC 0A 00 4F 4D  \BROWSE.......OM
4F 4F 4D 49 00 00 2D 00 73 00 65 00 72 00 05 01  OOMI..-.s.e.r...
03 10 01 00 0F 01 55 AA 00                       ......U..

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+

09/19-14:20:51.743889 172.16.2.181 -> 224.0.0.22
IGMP TTL:1 TOS:0x0 ID:431 IpLen:24 DgmLen:40
IP Options (1) => Opt 148: 00 00
22 00 EB 03 00 00 00 01 03 00 00 00 EF FF FF FA  "...............

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
```

❖ e- برای نمایش سرآیندهای لایه data-link شبکه و دریافت اطلاعات MAC مربوطه؛ با دستور:

```
#snort -vde
```

دقت داشته باشید که می توانیم چند سوییچ مختلف یک دستور را جداگانه بنویسیم؛ مثلاً به جای #snort -vde می توانیم #snort -v -d -e را نیز به کار ببریم.

۵-۲) مد کاری Packet logger

Snort در این مد کاری به ثبت (log) کردن بسته های جاری در شبکه تحت حفاظت، داخل حافظه می پردازد تا امکان بازبینی و پردازش های بعدی فراهم شود. محل پیش فرض برای ذخیره سازی بسته ها var/log/snort می باشد که باید از قبل (با دستور mkdir) ایجاد شده باشد. با هر بار اجرای نرم افزار، فایلی با عنوان برچسب زمانی مربوطه داخل آن تشکیل شده و بسته ها داخل آن ذخیره می شود. (در صورت تنظیم صحیح فایل پیکربندی، Snort بر مبنای آدرس IP یکی از میزبان های ذکر شده در دیتاگرام (اصولاً میزبان موجود در شبکه تحت حفاظت یا منبع صدور بسته یا میزبانِ دارای شماره پورت بیشتر)، زیر شاخه هایی تشکیل داده و ذخیره سازی نتایج داخل آن انجام می شود.) گزینه ها و دستورات مربوط به این مد کاری به شرح زیر است :

- ❖ l- برای ثبت (ذخیره) log بسته ها روی دیسک؛ با دستور: #snort -vdel /var/log/snort
- ❖ h- برای ذخیره و ثبت وقایع یک شبکه یا میزبان خاص روی یک دایرکتوری جداگانه؛ با نمونه دستور:

#snort -vdel /var/log/snort -h 192.168.1.0/24

- ❖ b- برای ذخیره کل وقایع شبکه (تمام بسته ها) روی یک تک فایل با فرمت استاندارد tcpdump برای شبکه های بسیار پر سرعت یا به جهت ذخیره سازی مختصر بسته ها برای تحلیل های بعدی (همراه با حذف سوییچ های vde- و h-) ؛ با دستور: #snort -l /var/log/snort -b
- ❖ N- برای متوقف کردن کار ثبت رخداد توسط نرم افزار

از طرف دیگر برای خواندن بسته های ذخیره شده، علاوه بر روشهای ذخیره سازی و بازیابی اطلاعات در پایگاه های داده (مثل MySQL)، می توان از نرم افزارهای Sniffer مثل tcpdump یا Ethereal که فرمت tcpdump را پشتیبانی کنند، برای بازخوانی بسته های ذخیره شده به صورت باینری استفاده کرد. همچنین خود نرم افزار نیز امکان بازخوانی فایلهای log را با سوییچ r- فراهم می کند:

❖ r- برای انجام بازخوانی و تجزیه ی بسته هایtcpdump (باینری/غیر باینری) که در پیمایش های قبلی ذخیره شده اند و نمایش آنها روی صفحه نمایش (با هر یک از سوییچ های مد کاری packet sniffer)؛

با نمونه دستور : # snort -r /var/log/snort snort.log.1147025043

خروجی در این حالت به صورت زیر است:

```
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+

05/07-15:36:47.479540 172.16.2.225:28556 -> 208.64.230.62:2082
TCP TTL:64 TOS:0x0 ID:11321 IpLen:20 DgmLen:591 DF
***AP*** Seq: 0x632C73F2  Ack: 0x80D9AA98  Win: 0x16D0  TcpLen: 20
47 45 54 20 2F 6C 6F 67 6F 75 74 2F 20 48 54 54  GET /logout/ HTT
50 2F 31 2E 31 0D 0A 48 6F 73 74 3A 20 61 72 61  P/1.1..Host: ara
73 68 61 62 74 61 68 69 2E 63 6F 6D 3A 32 30 38  shabtahi.com:208
32 0D 0A 55 73 65 72 2D 41 67 65 6E 74 3A 20 4D  2..User-Agent: M
6F 7A 69 6C 6C 61 2F 35 2E 30 20 28 58 31 31 3B  ozilla/5.0 (X11;
20 55 3B 20 4C 69 6E 75 78 20 69 36 38 36 3B 20   U; Linux i686;
65 6E 2D 55 53 3B 20 72 76 3A 31 2E 37 2E 31 30  en-US; rv:1.7.10
29 20 47 65 63 6B 6F 2F 32 30 30 35 30 37 31 35  ) Gecko/20050715
20 46 69 72 65 66 6F 78 2F 31 2E 30 2E 36 20 53   Firefox/1.0.6 S
55 53 45 2F 31 2E 30 2E 36 2D 31 36 0D 0A 41 63  USE/1.0.6-16..Ac
63 65 70 74 3A 20 74 65 78 74 2F 78 6D 6C 2C 61  cept: text/xml,a
70 70 6C 69 63 61 74 69 6F 6E 2F 78 6D 6C 2C 61  pplication/xml,a
70 70 6C 69 63 61 74 69 6F 6E 2F 78 68 74 6D 6C  pplication/xhtml
2B 78 6D 6C 2C 74 65 78 74 2F 68 74 6D 6C 3B 71  +xml,text/html;q
3D 30 2E 39 2C 74 65 78 74 2F 70 6C 61 69 6E 3B  =0.9,text/plain;
71 3D 30 2E 38 2C 69 6D 61 67 65 2F 70 6E 67 2C  q=0.8,image/png,
2A 2F 2A 3B 71 3D 30 2E 35 0D 0A 41 63 63 65 70  */*;q=0.5..Accep
74 2D 4C 61 6E 67 75 61 67 65 3A 20 65 6E 2D 75  t-Language: en-u
73 2C 65 6E 3B 71 3D 30 2E 35 0D 0A 41 63 63 65  s,en;q=0.5..Acce
70 74 2D 45 6E 63 6F 64 69 6E 67 3A 20 67 7A 69  pt-Encoding: gzi
70 2C 64 65 66 6C 61 74 65 0D 0A 41 63 63 65 70  p,deflate..Accep
74 2D 43 68 61 72 73 65 74 3A 20 49 53 4F 2D 38  t-Charset: ISO-8
38 35 39 2D 31 2C 75 74 66 2D 38 3B 71 3D 30 2E  859-1,utf-8;q=0.
37 2C 2A 3B 71 3D 30 2E 37 0D 0A 4B 65 65 70 2D  7,*;q=0.7..Keep-
41 6C 69 76 65 3A 20 33 30 30 0D 0A 43 6F 6E 6E  Alive: 300..Conn
65 63 74 69 6F 6E 3A 20 6B 65 65 70 2D 61 6C 69  ection: keep-ali
76 65 0D 0A 52 65 66 65 72 65 72 3A 20 68 74 74  ve..Referer: htt
70 3A 2F 2F 61 72 61 73 68 61 62 74 61 68 69 2E  p://arashabtahi.
63 6F 6D 3A 32 30 38 32 2F 66 72 6F 6E 74 65 6E  com:2082/fronten
64 2F 78 2F 69 6E 64 65 78 2E 68 74 6D 6C 0D 0A  d/x/index.html..
43 6F 6F 6B 69 65 3A 20 63 70 72 65 6C 6F 67 69  Cookie: cprelogi
6E 3D 6E 6F 0D 0A 41 75 74 68 6F 72 69 7A 61 74  n=no..Authorizat
69 6F 6E 3A 20 42 61 73 69 63 20 59 58 4A 68 63  ion: Basic YXJhc
32 67 36 64 47 56 71 59 58 4A 68 64 46 38 33 4E  2g6dGVqYXJhdF83N
7A 67 77 0D 0A 0D 0A                             zgw....
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+

05/07-15:36:51.682537 172.16.2.182:53 -> 172.16.2.225:1182
UDP TTL:128 TOS:0x0 ID:7222 IpLen:20 DgmLen:118
Len: 90
0D 1F 81 80 00 01 00 02 00 00 00 00 0B 63 68 61  .............cha
74 65 6E 61 62 6C 65 64 04 6D 61 69 6C 06 67 6F  tenabled.mail.go
```



```
6F 67 6C 65 03 63 6F 6D 00 00 01 00 01 C0 0C 00  ogle.com........
05 00 01 00 00 04 B2 00 11 01 62 0A 67 6F 6F 67  ..........b.goog
6C 65 6D 61 69 6C 01 6C C0 1D C0 39 00 01 00 01  lemail.l...9....
00 00 01 1D 00 04 42 66 0B BD                    ......Bf..
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
05/07-15:37:12.806233 ARP who-has 172.16.2.124 tell 172.16.2.168

05/07-15:37:20.625112 ARP who-has 172.16.2.1 tell 172.16.2.144

05/07-15:37:28.928102 172.16.2.225 -> 224.0.0.22
IGMP TTL:1 TOS:0xC0 ID:0 IpLen:24 DgmLen:40 DF
IP Options (1) => Opt 148: 00 00
22 00 FA 02 00 00 00 01 03 00 00 00 E0 00 00 FB  "...............

+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
```

❖ انجام عمل تجزیه (parse) توسط فیلتر بسته برکلی یا BPF (Berkeley Packet Filter) که در داخل Snort برای عناصر خاص مورد نظر (مثلاً تنها نمایش بسته های UDP یا TCP یا ICMP) تعبیه شده است؛ با نمونه دستورهای:

#snort -vder /var/log/snort/snort-06291600.log udp

#snort -vder /var/log/snort/snort-06291600.log tcp

#snort -vder /var/log/snort/snort-06291600.log icmp

۳-۵) مد کاری Full Network Intrusion Detection

در این حالت که پیچیده ترین و قابل انعطاف ترین مد کاری Snort است، نرم افزار به عنوان یک NIDS ، ترافیک شبکه را با توجه به ruleset یا مجموعه قوانینِ تعریف شده، بررسی کرده و بر اساس نتیجه ی این بررسی، عملکرد خاصی از خود نشان می دهد؛ یا بسته (سالم) را بدون توجه عبور می دهد(pass) و یا نسبت به بسته مشکوک (ناقض قوانین) اعلام هشدار(alert) کرده و آن بسته و هشدار مربوطه را به صورت فایل ASCII ثبت (log) می‌کند. در این حالت فایل اصلی تنظیمات Snort در آدرس /etc/snort/snort.conf و مجموعه قوانین تعریف شده در /etc/snort/rules به صورت فایل متنی ساده قرار دارد. همچنین هشدارهای اعلام شده در آدرس /var/log/snort/alert ذخیره می شود.

برای اجرا کردن Snort در این حالت از سوییچ c- استفاده می شود. نمونه دستور زیر که در آن گزینه‌های v- و e- (برای سادگی و جلوگیری از پر شدن سریع فایل‌های log) حذف شده اند، این مطلب را روشن می کند:

Ebrahim MolavianJazi

#snort -d /var/log/snort -h 192.168.1.0/24 -c /etc/snort/snort.conf

۵-۳-۱) گزینه های دیگر برای خروجی در مد کاری IDS

چنانچه Snort در مد NIDS کار کند، در حالت عادی، ذخیره کردن بسته ها و اعلام هشدار به صورت هشدار کامل یا full alert و در فرمت ASCII دکد شده، انجام می شود. منظور از هشدار کامل آنست که پیام هشدار به همراه کل سرآیند بسته پرینت می شود. اما گزینه های دیگری (مانند هشدار سریع، هشدار روی نمایشگر، هشدار cmg، توقف هشدار) نیز برای اعلام هشدار وجود دارد که می توان با سوییچ A- طبق جدول زیر از آنها استفاده کرد:

Option	Description
-A fast	Fast alert mode. Writes the alert in a simple format with a timestamp, alert message, source and destination IPs/ports.
-A full	Full alert mode. This is the default alert mode and will be used automatically if you do not specify a mode.
-A unsock	Sends alerts to a UNIX socket that another program can listen on.
-A none	Turns off alerting.
-A console	Sends "fast-style" alerts to the console (screen).
-A cmg	Generates "cmg style" alerts.

مثلا با دستور زیر بسته ها در شاخه پیش فرض /var/log/snort ذخیره می شوند ولی هشدارها به فایل هشدار سریع (fast alert) ارسال می گردند:

```
./snort -c snort.conf -A fast -h 192.168.1.0/24
```

هم چنین می توان پیام های هشدار را با سوییچ s- به syslog فرستاد. مثلاً در قانون زیر ثبت عادی بسته ها به فرم ASCII ـ دکد شده در فایل پیش فرض انجام می شود ولی پیام های هشدار به syslog ارسال می شوند:

```
./snort -c snort.conf -A fast -h 192.168.1.0/24
```

۵-۳-۲) خروجی استاندارد برای هشدار در مد کاری IDS

پیش از آنکه فرمت و اجزای خروجی یک هشدار را شرح دهیم، ابتدا یک مورد از این هشدارها را به طور کامل (شامل اصل هشدار، پیام هشداری و بسته مشکوک، و قانون نقض شده) مشاهده می کنیم:

اصل هشدار

Jan 25 15:20:53 homebrew snort: [1:10000008:1] ALERT!!! NACHI Infection!!
[Classification: A Network Trojan was detected] [Priority: 1]: {ICMP}
67.169.207.107 -> 67.168.231.153

پیام هشدار و بسته مشکوک

```
[**] ALERT!!! NACHI Infection[**] !!
25/01-112372, 14:18:42 67.168.223.131->67.168.231.153
ICMP TTL:123 TOS:0x0 ID:18910 IpLen:20 DgmLen:92
Type:8  Code:0  ID:512   Seq:42401  ECHO
AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA................
AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA................
AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA................
AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA................
+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=
```

قانون نقض شده

```
#This rule is for tracking Nachi infections
alert icmp $HOME_NET any -> any any (msg: "ALERT!!! NACHI Infection!!";
content: "laaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa
aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa
aaaal"; dsize:64; itype: 8; icode: 0; classtype:trojan-activity ; sid:10000008; rev:1;)
```

حال به بررسی فرمت و اجزای خروجی یک هشدار می پردازیم. زمانی که یک هشدار (alert) توسط مولدهای Snort تولید می شود، سطر اول آن به شکل زیر است:

```
[**] [116:56:1] (snort_decoder): T/TCP Detected [**]
```

به اولین عدد (در اینجا ۱۱۶) شماره شناسه مولد (Generator ID) گفته می شود که نشان می دهد کدام یک از اجزای نرم افزار این هشدار را تولید کرده است. در مثال فوق جزء مربوطه بخش decode (کدگشایی) بوده است. لیست زیر که از فایل /etc/snort/rules/generators گرفته شده است ، عناوین تمامی مولدهای نسخه ی 2.6 را نشان می دهد: (موارد دارای علامت * در لیست مولدهای نسخه فعلی وجود ندارند.)

rules subsystem	1 # Snort Rules Engine	frag2	113 # 2nd fragment preprocessor
tag subsystem	2 # Tagging Subsystem	fnord	114 # NOP detector *
portscan	100 # Portscan1	asn1	115 # ASN.1 Validator *
minfrag	101 # Minfrag *	decode	116 # Snort Internal Decoder
http_decode	102 # HTTP decode 1/2	scan2	117 # portscan2
defrag	103 # First defragmenter *	conversation	118 # conversation
spade	104 # SPADE *	reserved	119 # TBA
bo	105 # Back Orifice	reserved	120 # TBA
rpc_decode	106 # RPC Preprocessor	snmp	121 # Andrew Baker's newer SNMP decoder
stream2	107 # 2nd stream preprocessor *	sfportscan	122 # Dan Roelkers portscan
stream3	108 # 3rd stream preprocessor (AVL nightmare) *	frag3	123 # Marty Roesch's ip frag reassembler
telnet_neg	109 # telnet option decoder	smtp	124 # SMTP decoder/normalizer
stream4	111 # Stream4 preprocessor	ftp	125 # FTP decoder
arpspoof	112 # Arp Spoof detector	telnet	126 # telnet decoder/normalizer

به دومین عدد (در اینجا ۵۶) شماره شناسه ی قانون نرم افزار (Snort ID) یا شماره شناسه ی نشانه (Signature ID) گفته می شود که چنانچه GID=1 باشد، SID مربوطه داخل بخش گزینه های قانون در کنار کلیدواژه ی (sid) ذکر شده است و چنانچه GID عددی غیراز ۱ باشد، SID مربوط به مولدهای پیش پردازشگر (pre-processor) است که لیست SID های مربوط به آنها در فایل etc/snort/rules/gen-msg.map

آمده است و ما آن لیست را در ضمیمه (ض- ۱) می آوریم.

سومین عدد (در اینجا ۱) شماره شناسه بازبینی (revision ID) است که مربوط به کلیه واژه ی (rev) در بخش گزینه های قوانین Snort است و برای هر بازبینی جدیدِ مجموعه قوانین، باید این عدد را یک واحد افزایش داد.

۵- ۴) مد کاری In line

در این مد کاری جدید که از نسخه 2.3 به بعد در Snort تعبیه شده است، بسته ها به جای libpcap از طریق iptables جمع آوری می شوند و سپس Snort به عنوان یک سیستم جلوگیری از مزاحمت یا IPS (Intrusion Prevention System) بر اساس قوانین مخصوص این مدکاری، iptables را به انداختن (drop) یا عبور دادن (pass) بسته ها وادار می کند؛ در واقع با این مد کاری می توان درهر لحظه علیه بسته های مشکوک اقدام کرد و گزینه هایی نظیر reject و drop و sdrop در همین راستا طراحی شده اند تا یک سیستمِ جلوگیری از مزاحمت (نفوذ) پیاده سازی شود. اولویت گزینه به طور پیش فرض به این صورت است:

```
->activation->dynamic->drop->sdrop->reject->alert->pass->log
```

برای راه اندازی این مد کاری لازم است که iptables دریافت (دانلود) و کامپایل شود طوری که شامل make install-devel باشد. با این کارکتابخانه ای libipq نصب می شود که امکان تعامل Snortinline با iptables را فراهم می کند.به علاوه لازم است که بسته LibNet نیز دریافت و نصب شود. برای نصب این مد کاری باید دستورات زیر را اجرا کرد:

```
./configure --enable-inline
make
make install
```

سپس باید از بار (load) شدنِ ماژول ip-queue مطمئن شویم. آنگاه بااستفاده از QUEUE target ترافیک شبکه را به سمت Snort Inline می فرستیم. مثلاً با قانون زیر تمام ترافیک TCP از firewall به سمت پورت ۸۰ را به QUEUE target و در نهایت به سمت Snort Inline می فرستیم:

```
iptables -A OUTPUT -p tcp --dport 80 -j QUEUE
```

در نهایت با دستور زیر Snort Inline را اجرا می کنیم:

```
snort_inline -QDc ../etc/drop.conf -l /var/log/snort
```

عملکرد سوییچ های c- و l- شبیه حالت قبلی است. گزینه D- نیز به معنای عملکرد نرم افزار در حالت Daemon یا "خوابِ پس زمینه ای" است. سوییچ مهم در اینجا Q- است که بسته ها را از iptables می گیرد.

یک امکان جالب در این مد، جایگزینی بسته ها با بسته های مشابه و برگشت دادن آنها به مبدأ تولید بسته است که روش "جایگزینی محتوا" اثر آقای Jad Haile امکان این کار را (با شرط تساوی طول رشته محتوایی جایگزین با طول رشته ی محتوایی اصلی) فراهم می کند. قوانین نمونه زیر این موضوع را نشان می دهد:

```
alert tcp any any <> any 80 (msg: "tcp replace"; content:"GET"; replace:"BET";)
alert udp any any <> any 53 (msg: "udp replace"; \
    content: "yahoo"; replace: "xxxxx";)
```

۵- ۵) سایر سوییچ های اجرایی Snort

ترکیب نوشتاری (syntax) سوییچ های Snort به طور کامل در پایین مشاهده می شود:

snort [-options] <filter options>

```
Options:
        -A          Set alert mode: fast, full, console, or none (alert file alerts
only)               "unsock" enables UNIX socket logging (experimental).
        -b          Log packets in tcpdump format (much faster!)
        -c <rules>  Use Rules File <rules>
        -C          Print out payloads with character data only (no hex)
        -d          Dump the Application Layer
        -D          Run Snort in background (daemon) mode
        -e          Display the second layer header info
        -f          Turn off fflush() calls after binary log writes
        -F <bpf>    Read BPF filters from file <bpf>
        -g <gname>  Run snort gid as <gname> group (or gid) after initialization
        -G <0xid>   Log Identifier (to uniquely id events for multiple snorts)
        -h <hn>     Home network = <hn>
        -i <if>     Listen on interface <if>
        -I          Add Interface name to alert output
        -k <mode>   Checksum mode (all,noip,notcp,noudp,noicmp,none)
        -K <mode>   Logging mode (pcap[default],ascii,none)
        -l <ld>     Log to directory <ld>
        -L <file>   Log to this tcpdump file
        -m <umask>  Set umask = <umask>
        -n <cnt>    Exit after receiving <cnt> packets
        -N          Turn off logging (alerts still work)
        -o          Change the rule testing order to Pass|Alert|Log
        -O          Obfuscate the logged IP addresses
        -p          Disable promiscuous mode sniffing
        -P <snap>   Set explicit snaplen of packet (default: 1514)
        -q          Quiet. Don't show banner and status report
        -r <tf>     Read and process tcpdump file <tf>
```

```
        -R <id>    Include 'id' in snort_intf<id>.pid file name
        -s         Log alert messages to syslog
        -S <n=v>   Set rules file variable n equal to value v
        -t <dir>   Chroots process to <dir> after initialization
        -T         Test and report on the current Snort configuration
        -u <uname> Run snort uid as <uname> user (or uid) after initialization
        -U         Use UTC for timestamps
        -v         Be verbose
        -V         Show version number
        -w         Dump 802.11 management and control frames
        -X         Dump the raw packet data starting at the link layer
        -y         Include year in timestamp in the alert and log files
        -Z         Set the performonitor preprocessor file path and name
        -z         Set assurance mode, match on established sesions (for TCP)
        -?         Show this information
<Filter Options> are standard BPF options, as seen in TCPDump
```

برخی از سوییچ های مهم و کاربردی، پیشتر در زمان توضیح مدهای کاری ذکر شدند. حال به چند مورد دیگر از سوییچ های مفید Snort اشاره می کنیم:

- C- برای نمایش قسمت payload از بسته ها تنها به صورت کاراکتری (بدون اعداد HEX معادل)
- G <0x*id*>- برای تعیین یکتای رخدادها برای چند نرم افزار در حال کار همزمان با هم
- i <*interface*>- برای sniff کردن بسته ها روی interface ذکر شده
- I- برای ذکر نام interface مربوطه در داخل پیام هشدار
- K (pcapl asciil none)- برای تعیین مد ثبت رخداد (به طور پیش فرض pcap)
- L <*binary_log_file*>- برای تنظیم نام فایل خروجی برای ثبت کننده رخدادها (logger)
- O- برای محو کردن (obfuscating) آدرس IP در بسته های ثبت شده و هشدارها
- T- برای تست کردن و ارائه گزارش از وضعیت پیکربندی فعلی Snort
- V- برای نمایش شماره نسخه نرم افزار
- ?- برای نمایش اطلاعات راهنمای مربوط به سوییچ های نرم افزار

همان طور که قبلاً هم گفته شد، filter options یا expression مشخص می کند که کدام دسته از بسته ها برای انجام عملکرد مورد اشاره باید مورد بررسی قرار گیرند؛ یعنی تنها بسته هایی که آن عبارت برایشان صادق باشد، مورد بررسی قرار می گیرند. چنانچه عبارتی در این قسمت ذکر نشود، تمامی انواع بسته ها مورد بررسی قرار

می گیرند. به عنوان مثال در دستور snort -vde udp تنها بسته های UDP مورد بررسی قرار می گیرند و به سایر بسته ها توجهی نمی شوند. ما در اینجا از توضیح فرمت و مشروح عبارات به کار رفته خودداری می کنیم.

٦) قوانین نرم افزار Snort و چگونگی نوشتن آنها

همانطور که پیشتر ذکر شد، Snort در دو مُد کاری Snifter و Logger براساس مشخصات ذکر شده در دستور (مثل آدرس شبکهٔ مورد نظر، نوع بسته های مورد نظر، نوع اطلاعات مورد نظر از هر بسته)، خروجی متناسبی روی صفحهٔ کامپیوتر یا یک فایل خروجی درج می کند. اما مد کاری اصلی مورد نظر برای این نرم افزار، مد کاری کشف مزاحمت یا IDS (و به علاوه در نسخه های جدیدتر، مد کاری جلوگیری از مزاحمت یا IPS) است که Snort در آن بر مبنای سیاست های کلی تشخیص تهاجم (مزاحمت)، عملکردهای مشکوک و مزاحمت های ایجاد شده داخل شبکه را کشف می کند (و احیاناً از آنها جلوگیری می نماید). این قوانین که به آنها Snort Rules یا Snort Signatures گفته می شود، در فایلهای با فرمت ASCII نگهداری می‌شوند.

قوانین می تواند به صورت از پیش آماده از اینترنت مخصوصاً سایت رسمی نرم افزار یعنی www.snort.org دریافت شود (لیست اسامی این قوانین در ضمیمه (ض-۲) آمده است) و همچنین می تواند باتوجه به شرایط خاص شبکه توسط کاربر نرم افزار نوشته شود. نوشتن قوانین (rules) برای این نرم افزار، آسان و مبتنی بر یک زبان توصیفی ساده، انعطاف پذیر و قدرتمند است. آنچه در پی می آید، یک راهنمای جامع نوشتن قوانین بر مبنای Snort User Manual (نسخهٔ 2.6.0) است که در سال ۲۰۰۶ تهیه شده و در قسمت GET DOCS از سایت رسمی این نرم افزار، قابل دریافت است.

۶-۱) کلیات

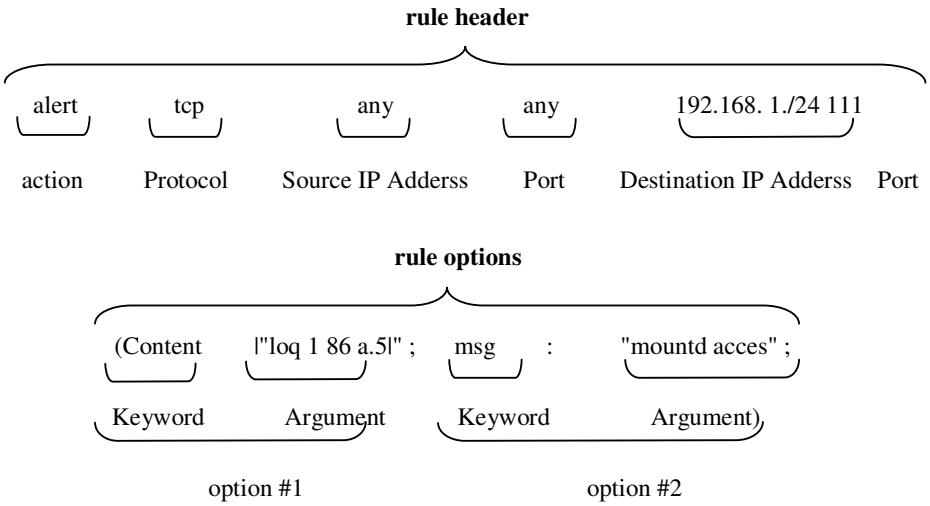
هر قانون Snort ، شامل دو بخش منطقی اصلی به نام «سرآیند قانون» (rule header) و «گزینه های قانون» (rule option) می باشد که به صورت روبرو نوشته می شود: rule header (rule options;)

بخش اول (rule header) شامل اطلاعاتی است که تعیین می کند، آن قانون وقوع موارد متناقض و مشکوک مربوط به کدام شبکه، کدام محل و چه نوع بسته ای را باید مورد بررسی قرار داده و چه عملی را باید در مقابل

این حرکت مشکوک و غیرقانونی انجام دهد. بخش دوم (rule options) که قلب موتور تشخیص مزاحمت (Snort's Intrusion Detection engine) را تشکیل می دهد، شامل گزینه های متنوع (و کلیدواژه های متناسب با آن) برای کشف مزاحمت در شبکه و اعلام هشدار است.

در زمان نگارش قانون، باید دقت کرد که بخش گزینه های قانون کلاً داخل یک زوج پرانتز قرار گیرد. در هر قانون، می توان بسته به سیاست مورد نظر از تعداد مورد نیازی از کلید واژه های هر دسته از گزینه ها استفاده کرد منتها هر گزینه باید با علامت (;) از گزینهٔ بعدی جدا شود. بدیهی است که در این صورت باید تمام گزینه ها در مورد یک بسته صادق باشد تا آن قانون عمل نماید. (دسته های مختلف گزینه ها در قسمت مربوطه مفصلاً تشریح خواهد شد.) به علاوه، باید بین کلید واژه ها و آرگومان (argument) مربوطه از علامت (:) استفاده کرد. اکثر قوانین در یک سطر نوشته می شوند ولی در نسخه های 1.8 به بعد، این امکان فراهم شده که یک قانون، در صورت طولانی شدن، در چند سطر نوشته شود؛ در این صورت باید در پایان هر سطر قانون علامت (\) قرار داده شود. همچنین، باید دقت کرد که وجود بخش گزینه های قانون الزامی نیست و یک قانون می تواند تنها شامل بخش سرآیند (header) باشد؛ علّت وجود بخش گزینه های قانون، آن است که تعریف دقیق تری از بسته های مشکوک ارائه شود. در پایین، یک نمونه از قوانین Snort و تجزیه آن به بخش های مختلف را مشاهده می کنیم:

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



نکتهٔ پایانی در این قسمت، آن است که برخی قوانین کنترلی دیگر نیز ممکن است در مجموعه قوانین نوشته شود تا عملکرد قوانین اصلی را تنظیم کند یا احیاناً در مواردی تغییر دهد. در نسخهٔ 2.6.0، این قوانین از سه نوع قوانین event tresholding برای جلوگیری از ثبت غیرضروری حجم زیادی از اطلاعات در مورد بسته ها- event suppression برای جلوگیری از عمل کردن و ثبت رخداد توسط یک قانون در موارد خاص- Multi-event logging (event queue) برای منظم کردن چند قانون مختلف مربوط به یک بسته ی واحد هستند.

۶-۲) اجزای سرآیند قانون (rule header)

سرآیند هر قانون به فرم زیر نوشته می شود:

Action Protocol Source_IP_Address Port Direction Destination_IP_Address Port

۱- عملکرد قانون (rule action): این قسمت، تعیین می کند که در صورت تطبیق یک بسته با معیار ارائه شده در قانون، نرم افزار Snort چه عملی را انجام خواهد داد. گزینه های مختلف امکان پذیر در قسمت action به شرح زیر هستند: (سه گزینهٔ آخر مربوط به نسخه های از 2.3 به بعد است که در آنها مد Inline نیز وجود دارد.)

- ❖ alert: تولید یک پیام هشدار و سپس ثبت بسته
- ❖ log: ثبت بسته
- ❖ pass: عبور دادن بسته (بدون هیچ عملی)
- ❖ activate: تولید پیام هشدار و سپس فعال کردن یک قانون از نوع "dynamic"
- ❖ dynamic: این قانون معلّق (بیکار) است تا آنکه با یک قانون از نوع "activate" فعال شود. در این صورت، به عنوان یک قانون log عمل می کند.
- ❖ drop: انداختن (حذف) بسته توسط iptables و سپس ثبت بسته

❖ reject : انـداختن بـسته توسـط iptables ، ثبـت بـسته و سـپس پایـان دادن بـه ارتبـاط (ارسـال سـیگنال TCP reset برای ارتباط TCP یا ارسال پیام ICMP Port unreachable برای ارتباط UDP)

❖ sdrop : انداختن بسته توسط iptables بدون ثبت بسته

علاوه بر عملکردهای فوق، کاربر می تواند " نوع قوانین" خود (rule type) را نیز تعریـف کـرده و یـک یـا چنـد plugin خروجی را به آن نسبت دهد و از آن پس، از عنوان ایـن rule typeهـا بـه عنـوان عملکـرد (action) در قوانین Snort استفاده کند. دو نمونه را در زیر مشاهده می کنیم:

```
ruletype suspicious                          (نام عملکرد تعریف شده)
{
    type log                                 (عملکرد از نوع ثبت)
    output log_tcpdump: suspicious.log       (نوع و محلّ خروجی)
}
```

rule type ذکر شده در مثال فوق، عملکرد به شکل " ثبت بسته به صورت tcpdump " دارد.

```
ruletype redalert                            (نام عملکرد تعریف شده)
{
    type alert                               (عملکرد از نوع هشدار)
    output alert_syslog: LOG_AUTH LOG_ALERT  (نوع خروجی)
    output database: log, mysql, user=snort dbname=snort host=localhost
}
```

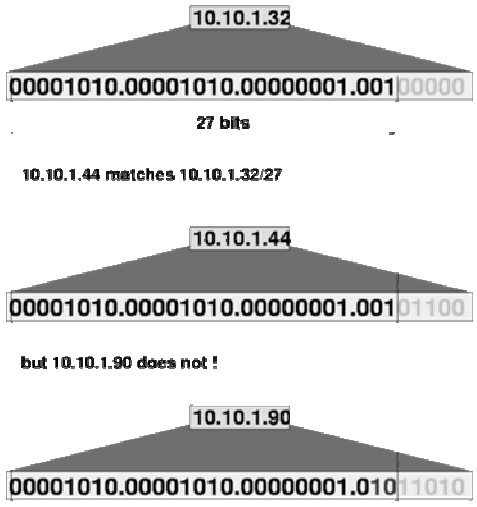
rule type ذکر شده در مثال فوق، عملکردی به صورت " ثبت در Syslog و پایگاه داده MySQL " دارد.

۲- پروتکل(Protocol) : چهار پروتکلی که در نـسخهٔ فعلـی (2.6) بـرای شناسـایی رفتارهـای مـشکوک، مـورد بررسی و تحلیل قرار می گیرند عبارتنـد از: UDP ، TCP ، IP ، ICMP . در آینـده، پروتکـل هـای دیگـری

مثل ARP ، IGRP ، GRE ، OSPF ، RIP و IPX نیز به مجموعهٔ پروتکل های مورد استفاده در Snort افزوده خواهد شد.

۳- آدرسهای IP مبدا و مقصد (IP Address) : از آنجا که نرم افزار امکان تبدیل نام میزبان به آدرس عددی IP را ندارد، لازم است که آدرس IP مبدأ و مقصد بسته های مورد نظر به صورت عددی با فرمت CIDR یا Classless Inter Domain Routing مشخص شوند.

در این روش یک IP به فرم A.B.C.D/n نمایش داده می‌شود که در آن پس از نوشتن معادل باینری قسمت اول (A.B.C.D)، عدد n که طول پیشوند (prefix length) می‌باشد، مشخص می کند که چند بیت از این ۳۲ بیت (از چپ به راست) به عنوان پیشوند فرض می‌شود و چند بیت از انتهای سمت راست آن آزاد (قابل تغییر) است. لذا مثلاً آدرس 192.168.30.2/32 مشخص می کند که کل ۳۲ بیت مشخص شده پیشوند است و لذا بیت آزادی وجود ندارد. این به معنای یک آدرس IP خاص (192.168.30.2) است. اما آدرس 192.168.30.0/24 به معنای ۲۴ بیت پیشوند و ۸ بیت آزاد است و لذا کل فیلد D (در فرمت A.B.C.D) آزاد است و در نتیجه یک کلاس از آدرس‌های (192.168.30.0 تا 192.168.30.255) را نشان می‌دهد (که به آن کلاس C گفته می شود). همچنین آدرس 192.168.30.0/21 به معنای پیشوند بودن ۲۱ بیت و آزاد بودن ۱۱ بیت است. لذا این عبارت ۸ کلاس C یعنی آدرس‌های (192.168.30.0 تا192.168.30.255) را مشخص می کند.



مفهوم آدرس دهی CIDR

برای آدرس دهی مبدا و مقصد، علاوه بر آدرس دهی CIDR مستقیم دو امکان اضافی نیز وجود دارد: یکی آنکه با عملگر (!) به نرم افزار اعلام شود که تمام آدرس های IP موجود به جز آدرس ذکر شده مورد بررسی قرار گیرند؛ دیگر آنکه می توان با یک زوج براکت ([]) و بدون قراردادن فاصلهٔ بین آدرس ها، تنها با قراردادن علامت (,) یک لیست از آدرس های IP تهیه کرد.

در مثال زیر، نسبت به هر ارتباط TCP شروع شده از هر پورت ماشین های خارج از شبکهٔ کلاس C مربوط به 192.168.1.0 به مقصد پورت portmapper متعلق به یک ماشین داخل شبکهٔ کلاس C مربوط به همان آدرس که حاوی مقدار باینری "00 01 86 a5" باشد، هشدار داده و پیام هشدار "external mounted access" را اعلام می کند.

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

می توان در مثال فوق، شبکهٔ مورد بررسی را به کلاس C مربوط به 192.168.1.0 و کلاس C مربوط به 10.1.1.0 توسیع داد. در این صورت داریم:

```
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";)
```

به علاوه می توان برای اشاره به کلِ آدرس های IP موجود از واژه "any" استفاده کرد.

۴- شماره پورت مبدا و مقصد (port) : برای اشاره به شماره پورت مبدأ و مقصد، می توان از واژه any (برای اشاره به تمام پورت ها) استفاده کرد و یا می توان با ذکر یک شماره، پورتی خاص را مورد نظر قرار داد. به علاوه، می توان یک رنج از پورت ها را ذکر کرد. مثلاً:

- ❖ همه پورت های با شماره بین ۱ تا ۱۰۲۴ — 1 : 1024
- ❖ همهٔ پورت های کمتر یا مساوی ۶۰۰۰ — : 6000
- ❖ همهٔ پورت های با شماره مساوی یا بیشتر از ۵۰۰ — 500 :

مثال زیر تمام ارتباطات TCP از هر مبدأیی با پورت شماره ۱۰۲۴ یا کمتر به مقصد شبکهٔ کلاس C مربوط به

192.168.1.0 با شماره پورت مقصد بزرگتر یا مساوی ۵۰۰ را ثبت می کند:

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

می توان عملگر نقیض(!) را بر سر شماره پورت خاص یا رنج پورت ها اعمال کرد و همه ی پورت ها به جز پورت (های) اشاره شده را مورد نظر قرار داد. در مثال زیر تمام ارتباطات TCP (از هر آدرس و پورت مبدأ دلخواه) به مقصد شبکهٔ کلاس C مربوط به 192.168.1.0 با هر شماره پورتی به جز شماره پورت های ۶۰۰۰ تا ۶۰۱۰ را ثبت می کند:

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

در اینجا یک لیست از شماره برخی از پورت های مهم (که در مثالهای این متن نیز به کار رفته) ارائه می شود:

20/TCP,UDP	پورت داده FTP	80/TCP	HTTP
21/TCP,UDP	پورت کنترلی FTP	110/TCP	POP3
22/TCP,UDP	SSH	111	Portmapper
23/TCP,UDP	Telnet	143/TCP,UDP	IMAP4
25/TCP,UDP	SMTP	161/TCP,UDP	SNMP
53/TCP,UDP	DNS	530/TCP	RPC

۵- Direction: این بخش جهت یا توالی ترافیکِ مورد بررسی را مشخص می کند. در حالت (<-) بسته های ارسالی از مبدأ (که در سمت چپ علامت جهت ذکر می شوند) به مقصد (که در سمت راست علامت جهت ذکر می شوند) با قانون تطبیق می شوند. در حالت (<>) یا دوطرفه، بسته های ارسالی هر دو طرف به طرف مقابل مورد بررسی قرار می گیرد. علامت (->) به منظور یکسان کردن نحوه نگارش قوانین و جلوگیری از ابهام در درک قوانین، مورد استفاده قرار نمی گیرد. در مثال زیر بررسی دو طرفهٔ (bi-directional) ترافیک را مشاهده می کنیم:

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

در این مثال، هر ارتباط (ارسالی یا دریافتی) بین پورت ۲۳ (telnet) از کلاس C شبکهٔ 192.165.1.0 و خارج

این شبکه ثبت خواهد شد.

۶-۲-۱) اجزای سرآیند قانون در قوانین activate و dynamic

آنچه در بالا در مورد سرآیند قانون ذکر شد، مربوط به قوانین با عملکردی غیر از activate و dynamic بود. اما در این دو حالت، در سرآیند قانون باید الزاماً موارد دیگری نیز ذکر شود:

- در بخش سرآیند قانون activate علاوه بر موارد فوق، یک گزینهٔ دیگر نیز با نام "activates" وجود دارد که پس از آن، شماره قانون dynamic مربوطه ذکر می شود. در صورت بروز رخداد مورد نظرِ این قانون، شبیه قانون alert پیغام هشدار داده می شود و سپس قانون dynamic ذکر شده فعال می گردد.
- در بخش سرآیند قانون dynamic ، علاوه بر موارد مذکور دو گزینهٔ دیگر وجود دارد: یکی "activated_by" که مشخص می کند توسط کدام قانون فعال می شود و دیگری count که مشخص می کند که پس از فعال شدن باید چه تعداد بسته ای را (شبیه قانون log) ثبت کند.

مثال زیر این دو قانون را کنار هم نشان می دهد:

```
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;)
```

دو قانون فوق در کنار هم باعث می شود که چنانچه سرریز بافر IMAP رخ داد و هشدار مربوطه توسط قانون اول داده شد، قانون دوم فعال شود و ۵۰ بستهٔ بعدی به مقصد پورت ۱۴۳ (IMAP4) را که از خارج شبکهٔ HOME_NET به داخل این شبکه ارسال می شود، ثبت کند. اهمیت مطلب در آن است که چنانچه بسته های ابتدایی در راستای حمله ای به قصد ایجاد سرریز در پورت مذکور ارسال شده باشد، بسته های ارسالی بعدی به این پورت به احتمال زیاد دارای داده های غیرعادی است که قرار است از طریق این پورت ارسال شود؛ لذا با ثبت این داده ها، امکان تحلیل بعدی وجود خواهد داشت.

تذکر: در نسخه های جدید باتوجه به وجود گزینه های flowbit و tag و امکان تلفیق آنها، به تدریج این دو قانون (activate , dynamic) از دور خارج می شوند.

۳-۶) گزینه های قانون (rule options)

گزینه های قانون که قلب موتور تشخیص مزاحمت Snort را تشکیل می دهند، شامل ۵ دستهٔ کلّی هستند که هرکدام به کمک مجموعه ای از کلید واژه ها عمل می کنند و برای کشف مزاحمت در شبکه و اعلام هشدار استفاده می شوند:

۱. گزینه های فوق داده ای یا Meta-data که تأثیری بر عملیات کشف مزاحمت ندارند و تنها اطلاعاتی در مورد قانون فراهم می کنند. (شامل کلید واژه های msg ، reference، sid، rev، type class، priority)

۲. گزینه های Payload که داده های مربوط به مزاحمت را در بخش Payload از بسته جستجو می کنند و می توانند به هم مرتبط (inter-related) باشند. (شامل کلید واژه های content، nocase، rawbytes ، depth ، offset ، distance، within ، uricontent ، isdataat ، pcre ، byte_ test، byte_ jump، ftpbounce)

۳. گزینه های Non-Payload که داده های مربوط به مزاحمت را در بخش های غیر payload از بسته جستجو می کند. (شامل کلید واژه های fragoffset، ttl، tos، id، ipopts، fragbits، dsize، flags، flow، flowbits، seq، ack، window، itype، icode، icmp_id، icmp_seq، rpc، ip_proto، sameip)

۴. گزینه های پس از تشخیص یا Post-Detection که پس از عمل کردن یک قانون فعال می شوند. (شامل کلید واژه های logto، session، resp، react، tag)

۵. گزینه های آستانه‌ی ثبت رخداد یا Event Thresholding که تعداد دفعات وقوع یک رخداد خاص در یک فاصله زمانی معین را محدود می‌کند. (شامل کلید واژه های limit، threshold، both)

۶-۳-۱) گزینه های Meta-Data

۱- msg: این گزینه با فرمت ("متن پیام" :msg) باعث می شود که موتور هشداردهنده و ثبت کننده، در زمان بروز یک رخداد، در کنار عملکردهای دیگر، پیام مورد نظر را چاپ کند. چنانچه متن شامل یک کاراکتر گسسته (مثل ;) باشد، برای جلوگیری از ایجاد اشکال در مترجم Snort، باید علامت (\) را کنار آن قرار داد.

۲- reference: این گزینه با فرمت (reference : < id system > , <:d>) باعث می شود که مجموعه قوانین Snort، ارجاعات به سیستم های شناسایی مزاحمتِ خارج از Snort را نیز شامل شود. درحال حاضر، ارجاع مستقیم به bugtraq، cve، nessus، arachnids، mcafee امکان پذیر است و همچنین می توان یک URL خاص را نیز ذکر نمود. اسامی فوق در آرگومان (id system) نوشته می شوند و شماره قانون مربوطه از آن سیستم، تحت عنوان (id) ذکر می شود. جدول سیستم های پشتیبانی شده به شرح زیر است:

System	URL Prefix
bugtraq	http://www.securityfocus.com/bid/
cve	http://cve.mitre.org/cgi-bin/cvename.cgi?name=
nessus	http://cgi.nessus.org/plugins/dump.php3?id=
arachnids	(currently down) http://www.whitehats.com/info/IDS
mcafee	http://vil.nai.com/vil/dispVirus.asp?virus_k=
url	http://

می توان در یک قانون، چند بار از این گزینه استفاده کرد. یک مثال از این گزینه، به شرح زیر است:

```
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;)
```

۳- sid: این گزینه با فرمت (شناسهٔ قانون sid:Snort) برای شناسایی یکتای قوانین Snort توسط plug-inهای خروجی استفاده می شود. در این صورت نرم افزار در زمان alert Post-Processing یا هشدار با پردازش بعدی، این امکان را دارد که براساس نگاشتی که در فایل sid-msg.map بین پیامهای هشدار و شناسهٔ قوانین Snort وجود دارد، هر شناسهٔ قانونی (rule ID) را به آسانی به یک پیام هشدار بنگارد. (دقت کنید که این گزینه باید همیشه به همراه گزینهٔ "rev" به کار رود.)

شماره شناسهٔ قوانین Snort به شرح زیراست:

رزرو شده برای استفاده در آینده: sid < ۱۰۰

مورد استفاده برای قوانین فعلی که به همراه نسخه های نرم افزاری منتشر می شود: Sid ۱۰۰ -۱/۰۰۰/۰۰۰

برای استفاده در قوانین محلّی (نوشته شده توسط کاربر): Sid > ۱/۰۰۰/۰۰۰

۴- rev: این گزینه با فرمت (شماره صحیح نسخهٔ :rev) برای شناسایی یکتای شماره بازنگری(نسخه) قوانین Snort (همواره به دنبال کلید واژه sid) به کار می رود. در این صورت، امکان تصحیح و جایگزینی قوانین قبلی با اطلاعات به روز شده، وجود خواهد داشت. مثال زیر یک قانون Snort با شماره شناسه ۱/۰۰۰/۹۸۳ و شماره بازنگری(نسخه) شماره ۱ را نشان می دهد:

```
alert tcp any any -> any 80 (content:"BOB"; sid:1000983; rev:1;)
```

۵- classtype: این گزینه با فرمت (نام کلاس :classtype) این امکان را فراهم می کند که هشدارها را براساس اولویت آنها طبقه بندی کنیم. در واقع با تعریف یک کلاس، یک مقدار اولویت پیش فرض به آن کلاس اختصاص می یابد. در نسخه های فعلی نرم افزار، مقدار ۱ نشانگر سطح اولویت بالا و مقدار ۳ نشانگر سطح اولویت پایین است. فرمت کلاسهای هشدار به صورت زیر است:

config classification: نام کلاس, توصیف کلاس, اولویت پیش فرض

کلاسهای هشدار همگی داخل فایلی به نام /etc/classification.config ذخیره می شوند. طبقه بندی استاندارد این کلاسها در جدول صفحه بعد آمده است.

۶- priority: این گزینه با فرمت (عدد صحیح اولویت :priority) یک severity level یا سطح اهمیت به هر قانون اختصاص می دهد. در واقع ما با استفاده از گزینهٔ priority این امکان را داریم که مقدار پیش فرض اولویت قانون را تغییر دهیم یا به قانونی که بدون نوع کلاس است و سطح اولویتِ از پیش تعریف شده ای ندارد، سطح اولویت اختصاص بدهیم.

Snort Default Classifications

Classtype	Description	Priority
attempted-admin	Attempted Administrator Privilege Gain	high
attempted-user	Attempted User Privilege Gain	high
shellcode-detect	Executable code was detected	high
successful-admin	Successful Administrator Privilege Gain	high
successful-user	Successful User Privilege Gain	high
trojan-activity	A Network Trojan was detected	high
unsuccessful-user	Unsuccessful User Privilege Gain	high
web-application-attack	Web Application Attack	high
attempted-dos	Attempted Denial of Service	medium
attempted-recon	Attempted Information Leak	medium
bad-unknown	Potentially Bad Traffic	medium
denial-of-service	Detection of a Denial of Service Attack	medium
misc-attack	Misc Attack	medium
non-standard-protocol	Detection of a non-standard protocol or event	medium
rpc-portmap-decode	Decode of an RPC Query	medium
successful-dos	Denial of Service	medium
successful-recon-largescale	Large Scale Information Leak	medium
successful-recon-limited	Information Leak	medium
suspicious-filename-detect	A suspicious filename was detected	medium
suspicious-login	An attempted login using a suspicious user-name was detected	medium
system-call-detect	A system call was detected	medium
unusual-client-port-connection	A client was using an unusual port	medium
web-application-activity	access to a potentially vulnerable web appli-cation	medium
icmp-event	Generic ICMP event	low
misc-activity	Misc activity	low
network-scan	Detection of a Network Scan	low
not-suspicious	Not Suspicious Traffic	low
protocol-command-decode	Generic Protocol Command Decode	low
string-detect	A suspicious string was detected	low
unknown	Unknown Traffic	low

مثال های مربوط به گزینه های classtype و priority را در پایین مشاهده می کنیم:

```
alert TCP any any -> any 80 (msg: "WEB-MISC phf attempt"; flags:A+; \
     content: "/cgi-bin/phf"; priority:10;)
alert tcp any any -> any 80 (msg:"EXPLOIT ntpdx overflow"; \
    dsize: >128; classtype:attempted-admin; priority:10 );

alert tcp any any -> any 25 (msg:"SMTP expn root"; flags:A+; \
   content:"expn root"; nocase; classtype:attempted-recon;)
```

۶-۳-۲) گزینه های شناسایی در Payload

۱- content : این گزینه با فرمت (" رشتهٔ محتوایی " [!] :content) یکی از مهمترین و پرکاربردترین ابزارهای نرم افزار Snort است. از طریق این گزینه می توان با کمک الگوریتم رشته یاب Boyer-Moore ، یک

رشتهٔ محتوایی خاص را داخل قسمت Payload از بسته جستجو کرد. این جستجو حساس به حروف کوچک و بزرگ می باشد. رشتهٔ مورد جستجو، می تواند ترکیبی از حروف (متن) و اعداد باینری (با نمایش HEX) باشد که برای جستجوی قسمت های باینری لازم است از علامت لولهٔ (|) در دو سوی عدد باینری استفاده شود. همانگونه که در فرمت این گزینه مشاهده می شود، امکان استفاده از علامت نقیض(!) نیز وجود دارد که در این صورت بسته هایی مورد نظر خواهند بود که شامل یک رشته (عبارت) خاص نباشند. ضمناً دقت کنید که در داخل رشته محتوایی از کاراکترهای (\ ;) نباید استفاده کرد.

روشن است که در صورت استفاده چند باره از این گزینه در یک قانون، می توان بادقت بالایی بسته های مشکوک را شناسایی کرده و از تشخیص اشتباه بسته های سالم به جای بسته های مشکوک (خطای مثبت غلط یا false positive) جلوگیری کرد.

در مثال زیر، قانونی را مشاهده می کنیم که نسبت به ارتباطات TCP به مقصد پورت ۱۳۹ (NetBIOS) که حاوی رشتهٔ |5c 00|P|00|I|00|P|00|E|00| است، هشدار می دهد: (در کدگذاری ASCII عبارت ØEØPØIØPØ معادل عبارت PIPE است، با علامت Ø به معنای Null)

```
alert tcp any any -> any 139 (content:"|5c 00|P|00|I|00|P|00|E|00 5c|";)
```

مثال زیر نیز قانونی را نشان می دهد که در مورد ارتباطات TCP به مقصد پورت ۸۰ (http) که در بخش payload حاوی رشتهٔ "GET" نباشند، هشدار می دهد:

```
alert tcp any any -> any 80 (content:!"GET";)
```

شش گزینهٔ بعدی (within – distance – offset – depth – rawbytes – nocase) کلید واژه هایی هستند که نحوه عملکرد گزینهٔ content- ای را که پیش از آنها در قانون ذکر شده است، تغییر می دهند.

Snort از نسخه ی 2.0 به بعد، از گزینه content (و uricontent) تحت حمایت یک موتور تطبیق الگوی setwise استفاده می کند که در آن هر چه طول رشته محتوایی مورد جستجو بزرگتر باشد، تطبیق دقیقتر و سریعتر

است. لذا توصیه می شود که در متن هر قانون حداقل یک بار از گزینه ی content استفاده شود. این کار سرعت اجرای نرم افزار Snort را به شدت افزایش می دهد.

۲ ـ nocase : این گزینه ی بدون آرگومان با فرمت (nocase) حساسیت الگوریتم جستجوی content به حروف بزرگ و کوچک را برمی‌دارد و لذا جستجوی عبارات متنی تنها بر اساس اصل حروف (و نه کوچکی و بزرگی آنها) انجام می‌شود. مثال زیر قانونی را نشان می‌دهد که نسبت به ارتباطات TCP به مقصد پورت ۲۱ (پورت کنترلی FTP) که در قسمت payload خود حاوی رشته‌ی "user root" (با هر املایی مثلاً user root یا USER root یا user ROOT و ...) باشند، هشدار داده و پیام "FTP Root" را چاپ می کند:

```
alert tcp any any -> any 21 (msg:"FTP ROOT"; content:"USER root"; nocase;)
```

۳ ـ rawbytes : این گزینه ی بدون آرگومان با فرمت (rawbytes) باعث می‌شود که عملیات جستجوی گزینه content مستقیماً روی بایت‌های خام قسمت payload بسته انجام بگیرد و تغییرات و کد گشایی‌هایی که توسط پیش پردازشگرهای نرم افزار روی بایت‌های خام انجام گرفته است، در جستجو مورد توجه قرار نگیرد. مثال زیر کاربرد این گزینه را نشان می دهد:

```
alert tcp any any -> any 21 (msg: "Telnet NOP"; content: "|FF F1|"; rawbytes;)
```

۴ ـ depth : این گزینه با فرمت (مقدار عددی :depth) مشخص می کند که نرم افزار برای یافتن رشته محتوایی مورد نظر در گزینه content، باید تا چند بایت از قسمت payload از بسته را پیش برود.

۵ ـ offset : این گزینه با فرمت (مقدار عددی :offset) مشخص می کند که نرم افزار عملیات جستجوی رشته محتوایی مورد نظر در گزینه content را باید از کجا (چند بایت پس از شروع قسمت payload بسته) آغاز کند. مثال زیر قانونی را نشان می‌دهد که نسبت به ارتباطات TCP به مقصد پورت ۸۰ (http) که در ۲۰ بایت شروع شونده از بایت پنجم در قسمت payload حاوی عبارت (cgi-bin/phf) باشند، هشدار می‌دهد:

```
alert tcp any any -> any 80 (content: "cgi-bin/phf"; offset:4; depth:20;)
```

۶ ـ distance : این گزینه با فرمت (تعداد بایت :distance) شبیه گزینه‌ی offset است که در آن عملیات جستجو برای گزینه content دوم، پس از N بایت از پایان جستجوی گزینه content اولی آغاز می‌شود، و نه N بایت پس از شروع payload (N = تعداد بایت ذکر شده در گزینه distance). در واقع حداقل فاصله بین دو عبارت مورد جستجو باید N بایت باشد. در مثال زیر ارتباط مشکوک، یک ارتباط TCP است که در قسمت payload آن، بین دو رشته‌ی (ABC) و (DEF) حداقل یک بایت فاصله وجود داشته باشد:

```
alert tcp any any -> any any (content:"ABC"; content: "DEF"; distance:1;)
```

۷ ـ within : این گزینه با فرمت (تعداد بایت :within) برعکس گزینه قبلی، باعث می‌شود که حداکثر فاصله‌ی بین دو رشته مورد جستجو در گزینه های content برابر N بایت باشد (N = تعداد بایت ذکر شده در گزینه within). مثلاً قانون زیر نسبت به ارتباطات TCP که در آنها دو رشته (ABC) و (DEF) حداکثر به فاصله ۱۰ بایت از هم باشند، هشدار می‌دهد:

```
alert tcp any any -> any any (content:"ABC"; content: "EFG"; within:10;)
```

۸ ـ uricontent : این گزینه با فرمت (رشته محتوایی [!] :uricontent) برای جستجو در فیلد URI مناسب است. (منظور از URI شناساگر یکنواخت مأخذ یا Uniform Resource Identifier است که رشته ای فشرده از کاراکترهاست و برای نامگذاری یک مأخذ (URN) یا موقعیت یابی یک مأخذ (URL) به کار می‌رود.) نکته مهم آنست که این جستجو در URIِ نرمال شده انجام می‌گیرد؛یعنی شکلی از URI که در آن پیمایش‌های شاخه‌ای (directoy traversal) و همچنین کاراکترهای خاص آدرس دهی مثل 2f٪ حذف شده است. لذا متنی که در گزینه‌ی uricontent نوشته می‌شود، نباید شامل چنین عباراتی باشد. دو مثال از نرمال سازی URI را در زیر می‌بینیم:

```
/scripts/..%c0%af../winnt/system32/cmd.exe?/c+ver
```

⟶ /winnt/system32/cmd.exe?/c+ver

```
\begin{verbatim} /cgi-bin/aaaaaaaaaaaaaaaaaaaaaaaaa/..%252fp%68f?
```

⟶ /cgi-bin/phf?

واضح است که اگر بخواهیم جستجو در کل URI (بدون نرمال سازی) انجام شود، باید از گزینه content استفاده کنیم. مجدداً تاکید می کنیم که با توجه به استفاده کردن Snort از گزینه uricontent (و content) تحت حمایت یک موتور تطبیق الگوی setwise و انجام دقیقتر و سریعتر تطبیق با استفاده از این گزینه، توصیه می شود که در متن قانون حداقل یک بار از گزینه ی uricontent (یا content) استفاده شود. این کار سرعت اجرای Snort را به شدت افزایش می دهد.

۹ ـ isdataat : این گزینه با فرمت ([, relative] عدد صحیح :isdataat) بررسی می‌کند که آیا در فاصله‌ی N بایت از شروع قسمت payload (یا در فاصله‌ی N بایت از پایان آخرین تطبیق رشته‌ای، اگر عبارت (relative) هم به عنوان آرگومان گزینه ذکر شود) هنوز داده‌ای وجود دارد یا نه (N= عدد صحیح ذکر شده در گزینه). در مثال زیر زمانی هشدار داده می‌شود که یک ارتباط TCP به مقصد پورت ۱۱۱ (portmapper) شامل عبارت PASS باشد و همچنین ۵۰ بایت پس از پایان این عبارت هنوز داده‌ای وجود داشته باشد و همچنین هیچ کاراکتر خط جدید یا newline (معادل ASCII از کاراکتر 0a) در فاصله‌ی ۵۰ بایت از پایان تطبیق قبلی (PASS) وجود نداشته باشد.

```
alert tcp any any -> any 111 (content:"PASS"; isdataat:50,relative; \
   content:!"|0a|"; distance:0;)
```

۱۰ ـ pcre : این گزینه با فرمت "pcre:[!]"(/<regex>/|m<delim><regex><delim>)[ismxAEGRUB] امکان نوشتن قوانین به کمک PCRE (Perl Compatible Regular Expressian) را فراهم می‌کند. حال مفهوم PCRE و همچنین آرگومان های به کار رفته در این گزینه را تشریح می کنیم.

زبان خلاصه سازی و گزارش عملی(Practical Extraction and Report Language) یا Perl یک زبان برنامه نویسیِ رویه‌ای دینامیک است که با کمک گرفتن از ویژگی‌های زبانهای دیگر مثل C و shell و AWK و lisp و sed بیشتر به پردازش متن می‌پردازد. یکی از موارد مهم در پردازش متن، مفهوم "عبارت معمول" یا regular expression (به اختصار regex) است که منظور از آن رشته‌ای است که بر اساس قوانین ترکیبی

خاص، یک مجموعه از رشته‌ها را توصیف می‌کند. بر این اساس، PCRE یک regex برگرفته از خروجی perl است و به علت قوت و انعطاف زیادی که نسبت به regexهای استاندارد دارند، بسیار مورد استفاده قرار می‌گیرند.

منظور از آرگومان delim، کاراکترهای حائل (delimiter) مثل (;) یا (,) است که برای جداسازی موارد مختلف استفاده می‌شوند. همچنین این گزینه اصلاح کننده‌ها (modifier) مختلفی دارد که توضیح آنها در جدول زیر بیان شده است:

i	case insensitive
s	include newlines in the dot metacharacter
m	By default, the string is treated as one big line of characters. ^ and $ match at the beginning and ending of the string. When m is set, ^ and $ match immediately following or immediately before any newline in the buffer, as well as the very start and very end of the buffer.
x	whitespace data characters in the pattern are ignored except when escaped or inside a character class
A	the pattern must match only at the start of the buffer (same as ^)
E	Set $ to match only at the end of the subject string. Without E, $ also matches immediately before the final character if it is a newline (but not before any other newlines).
G	Inverts the "greediness" of the quantifiers so that they are not greedy by default, but become greedy if followed by "?".

سه گزینه آخر (RUB) اصلاح کننده‌های مخصوص Snort است که به شرح زیر می‌باشند:

R : شروع کردن تطبیق جدید از اتمام تطبیق قبلی (شبیه به گزینه‌ی distance:0;)

U : تطبیق با بافر URI کد گشایی شده (شبیه uricontent)

B : تطبیق بدون استفاده از تغییرات و کد گشایی عبارت (شبیه rawbytes)

دقت کنید که گزینه‌های R و B نباید با هم به کار روند. (این جمله از متن manual رسمی Snort نقل شده است ولی با توجه به اشکالات تایپی دیگری نیز که در این متن وجود دارد، احتمالاً جمله‌ی صحیح این است: " دقت کنید که گزینه‌های U و B نباید با هم بکار روند.")

یک مثال از کاربرد این گزینه را در قانون زیر با هدف جستجوی رشته‌ی (BLAH) در قسمت payload مشاهده می‌کنیم:

```
alert ip any any -> any any (pcre:"/BLAH/i";)
```

در مورد این گزینه باید دقت کرد که اولاً در نسخه‌های فعلی Snort (نسخه 2.6) کاربرد URIهای چندگانه برای پردازش با گزینه PCRE با اشکال مواجه است و تنها URI اولی پردازش می‌شود (برای اقدام به پردازش چند URI در یک قانون باید از گزینه‌های دیگری مثل content و uricontent استفاده کرد) و ثانیاً برای حفظ سرعت بالای نرم افزار توصیه می شود که تا حد امکان از کاربرد گزینه هایی مثل pcre پرهیز شود و به جای آن از گزینه های content و uricontent استفاده شود.

۱۱ ـ byte_test: این گزینه با فرمتِ

byte_test: بایت‌های مورد نظر برای تبدیل , [!]عملگر , مقدار , آفست [, relative] [, endian نوع]

[, نوع عدد , string];

این امکان را فراهم می‌کند که یک سری بایت‌های مورد نظر، بر اساس عملگرمشخص شده، با یک مقدار باینری خاص (یا معادل باینری یک رشته بایت) مقایسه شود. حال به توضیح فرمت این قانون می‌پردازیم:

منظور از *(بایت‌های مورد نظر برای تبدیل)* تعداد بایت‌هایی از بخش payload بسته است که باید برای عمل مقایسه، در نظر گرفته شود. برای این کار بر اساس مقدار گزینه آفست (offset) یک تعداد بایت از ابتدای بسته رها می‌شود و سپس بایت‌های مربوطه برداشته می‌شوند؛ چنانچه آرگومان *(relative,)* نیز استفاده شود، مقدار آفست نسبت به پایان الگوی تطبیق یافته قبلی در نظر گرفته می‌شود. در آرگومان *(مقدار)* مقدار عددی که بایت‌های مورد نظر با آنها مقایسه می‌شوند، تعیین می گردد. آرگومان *(string)* مشخص می‌کند که بایت‌های مورد نظر برای مقایسه از نوع رشته می باشند؛ در اینصورت آرگومان *(نوع عدد)* تعیین می‌کند که رشته تبدیل شده به فرم hex (مبنای ۱۶) یا dec (دهدهی) یا oct (مبنای هشت) نمایش داده شود. آرگومان *(عملگر)* نوع مقایسه را نشان می‌دهد که در اینصورت یکی از اعمال کوچکتری (<) یا بزرگتری (>) یا تساوی (=) یا AND منطقیِ بیت به بیت (&) یا OR منطقیِ بیت به بیت (^) انتخاب می‌شود و می‌توان در جلوی هر یک آنها علامت نقیض (!) را هم اعمال کرد ؛ مثلاً (> !). قرار دادن علامت نقیض (!) به تنهایی، به معنای عدم تساوی (≠) در نظر گرفته می‌شود. همچنین endian نوع ترتیب رتبه‌ای ارقام عدد را معلوم می‌کند؛ گزینه (big) که پیش فرض است، به معنای آنست که رقم

سمت چپ بیشترین ارزش عددی و رقم سمت راست کمترین ارزش عددی را دارد. گزینه (little) عکس این حالت را بیان می کند.

در اینجا به چند مثال از کاربرد این گزینه اشاره می کنیم:

```
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;)
```

قانون فوق پس از انجام دو عمل تطبیق الگو به کمک گزینه content، در بخش مقایسه با گزینه (byte_test)، ابتدا ۲۰ بایت نسبت به پایان تطبیق الگوی قبلی (ناشی از گزینه content دوم) را رها کرده و سپس ۴ بایت را برمی‌دارد و با عدد ۱۰۰۰ مقایسه از نوع بزرگتری (>) را انجام می‌دهد (تست می کند که آیا data ؟ ۱۰۰۰<)

```
alert udp any any -> any 1235 \
     (byte_test: 3, =, 123, 0, string, dec; \
     msg: "got 123!";)
```

قانون فوق ۳ بایت اول از قسمت payload از بسته‌ی udp را که به مقصد پورت ۱۲۳۵ ارسال شود، به صورت رشته دریافت کرده و به صورت دسیمال تبدیل می کند و با عدد ۱۲۳ تست تساوی می کند و در صورت صحت تساوی، پیام (got 123) را اعلام می نماید.

```
alert udp any any -> any 1238 \
     (byte_test: 8, =, 0xdeadbeef, 0, string, hex; \
     msg: "got DEADBEEF!";)
```

قانون فوق ۸ بایت اول از قسمت payload از بسته udp را که به مقصد پورت ۱۲۳۸ ارسال می شود، به صورت رشته دریافت کرده و به صورت HEX تبدیل می کند و با عدد " deadbeef " در مبنای ۱۶ تست تساوی می کند و در صورت صحت تساوی، پیام "got DEADOEE!" را اعلام می نماید.

با توجه به سرعت کم گزینه هایی مثل byte_test توصیه می شود که تا حد امکان از کاربرد این گزینه ها پرهیز شود و به جای آن حداقل یکبار از گزینه ی content استفاده شود.

۱۲ ـ byte_jump : این گزینه با فرمتِ

*byte_jump: بایت‌های برای تبدیل مورد نظر , آفست[, relative][, multiplier مقدار ضرب شونده] *

[, big][, little][, string][, hex][, dec][, oct] [, align][, from_beginning]);

بر اساس مقدار آفست، یک تعداد از بایتها را از ابتدای قسمت payload بسته (یا از انتهای تطبیق الگوی قبلی، در صورت وجود گزینه relative) رها می‌کند و سپس به تعداد بایت مشخص شده در آرگومانِ *(بایت‌های مورد نظر برای تبدیل)* بایت از بسته می‌خواند و به معادل عددی آن تبدیل می‌کند و به همان تعداد بایت جلو می‌رود و اشاره گر را برای کشف مزاحمت بعدی در آنجا قرار می‌دهد. به این اشاره گر detect offset end pointer یا به اختصار (doe-ptr) گفته می‌شود. در صورتی که داده به صورت رشته (string) در بسته ذخیره شده باشد، آرگومان (string) را نیز قرار می‌دهیم و در اینصورت داده ی رشته‌ای تبدیل شده ممکن است به صورت مبنای ۱۶ (hex) یا مبنای ۱۰ (dec) یا مبنای ۸ (oct) نمایش داده شود. همچنین این امکان وجود دارد که یک ضریبی از آن تعداد بایت (و نه دقیقاً خود همان تعداد بایت) محاسبه شده و اشاره گر به آن مقدار جلو برود. این کار با ذکر یک مقدار صحیح در کنار آرگومان (multiplier) امکانپذیر است. نیز، شبیه گزینه byte_test، برای ارزشگذاری اعداد ممکن است endian از نوع (big) یا (little) باشد. (پیش فرض بر روی حالت عادی یعنی big می‌باشد.) به علاوه با گزینه‌ی (align) می‌توان تعداد بایت‌های تبدیل شده را به مقدار ۳۲ بیتی بالایی گرد کرد و با گزینه‌ی (from beginning) می‌توان رها کردن تعداد بایت مورد نظر را از ابتدای قسمت payload بسته (به جای محل فعلی اشاره گر در بسته) انجام داد.

برای درک کاربرد این گزینه به قانون نمونه زیر دقت می کنیم. در این قانون پس از دو مورد عملیات تطبیق رشته‌ای با گزینه‌ی content که روی بسته‌های ارسالی udp به مقصد پورتهای ۳۲۷۷۰ تا ۳۴۰۰۰ انجام می‌شود، ۱۲ بیت از محل تطبیق الگوی قبلی رها می‌شود، ۴ بایت خوانده شده و به مقدار عددی تبدیل می‌شود و عدد حاصل به مقدار ۳۲ بیتی بعدی گرد شده و اشاره گر به تعداد بایت حاصل جلو می‌رود؛ آنگاه بر اساس محل اشاره گر

جدید ۲۰ بایت رها می‌شود، ۴ بایت قرائت می‌گردد و مقدار خوانده شده با عدد ۹۰۰ تست بزرگتری می‌شود (که آیا data > 900 ؟) در صورت وجود تمام شرایط فوق پیام هشدار ظاهر خواهد شد:

```
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";)
```

۱۳ ـ ftpbounce : این گزینه ی بدون آرگومان با فرمت (ftpbounce) برای آشکار سازی حمله FTP bounce به کار می‌رود. در حمله FTP bounce مهاجم که امکان دسترسی مستقیم به پورت‌های خاص را ندارد، با استفاده از فرمان (PORT) درخواست اتصال غیر مستقیم به برخی پورت‌ها را می‌نماید. (به واسطه‌ی یک ماشین "قربانی" میانی به عنوان middle-man .) برای نمونه نرم افزار پویشگر پورت nmap از این حمله برای پیمایش سایر serverها استفاده می‌کند. در مثال زیر کاربرد این گزینه را برای پورت شماره ۲۱ (پورت کنترلی FTP) مشاهده می کنیم:

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 21 (msg:"FTP PORT bounce attempt"; \
flow:to_server,established; content:"PORT"; nocase; ftpbounce; pcre:"/^PORT/smi";\
classtype:misc-attack; sid:3441; rev:1;)
```

۶-۳-۳) گزینه های شناسایی در قسمت‌های غیر payload

۱ ـ fragoffset : این گزینه با فرمت (عدد [<یا>] : fragoffset) اجازه می‌دهد که فیلد آفست در تکه سازی پروتکل IP (IP fragment offset) را با یک مقدار دسیمال (دهدهی) مقایسه کنیم. مثلاً همانگونه که در مثال زیر نیز دیده می‌شود، با مقایسه‌ی این فیلد با عدد صفر (0) می‌توان اولین تکه‌ی یک ارتباط IP را کشف کرد:

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

۲ ـ ttl : این گزینه با فرمت (عدد [= یا > یا <] [- عدد] : ttl) امکان بررسی کردن مقدار time-to-live یا زمان بقای بسته‌ی IP را فراهم می‌کند که با آن می‌توان اقدامات مسیر یابی (traceroute attempts) را کشف

کرد. مثلاً عبارت (ttl: < 3) بسته‌های با زمان بقای کمتر از ۳ واحد و (ttl: 3-5) بسته‌های با زمان بقای بین ۳ تا ۵ واحد را کشف می‌کند.

۳ـ tos : این گزینه با فرمت (عدد [!]:tos) برای بررسی کردن فیلد نوع سرویس یا Type of service (TOS) به کار می‌رود. مواردی مانند میزان تأخیر و قابلیت اعتماد بسته در این فیلد مقدار دهی می‌شوند. مثلاً عبارت (tos: ! 4;) مقادیری از TOS را که ۴ نباشند، جستجو می‌کند.

۴ـ id : این گزینه با فرمت (عدد :id) برای بررسی مقدار فیلد IP ID یا کد شناسه‌ی بسته‌ی IP استفاده می‌شود. این گزینه به این علت مفید است که برخی ابزارهای مزاحمت مثل پویشگرها (Scanners) این فیلد را به خاطر برخی مقاصد با یک مقدار خاص مقدار دهی می‌کنند. (مثلاً ۳۱۳۳۷ مورد تمایل برخی هکرهاست.) لذا عبارت (id: 31337) بسته‌های IP با کد شناسه‌ی برابر با ۳۱۳۳۷ را جستجو خواهد کرد.

۵ـ ipopts : این گزینه با فرمت (any یا satid یا ssrr یا lssr یا sec یا ts یا nop یا eol یا rr : ipopts) برای چک کردن فیلد option در بسته‌ی IP بکار می‌رود. در فرمت فوق الذکر حروف اختصاری به این معناست :

rr برای Record route ، eol برای پایان لیست یا end of list ، nop برای بیکاری یا No operation ، ts برای برچسب زمانی یا time stamp ، sec برای امنیت IP یا IP security ، lssr برای مسیر دهی سست منبع یا Loose souree Routing ، ssrr برای مسیر دهی محکم منبع یا strict source Routing ، satid برای شناساگر دنباله یا stream identifier ، any برای هر گزینه‌ی ست شده‌ی IP. مثلاً گزینه‌ی (ipopts: lsrr;) به دنبال گزینه‌ی loose source routing در بسته‌ی IP می‌گردد. (البته در هر قانون تنها می‌توان یکی از این موارد را به عنوان آرگومان گزینه ipopts ذکر کرد.)

با توجه به آنکه بسته‌های IP با بیت‌های ست شده برای گزینه‌هایIP، مخصوصاً موارد lssr و ssrr و rr دچار مشکلات امنیتی هستند و موجب مسدود شدن مسیر یاب (router) می‌شوند، استفاده از آنها رایج نیست و لذا سِت بودن این نوع بیت‌ها می‌تواند احتمال وجود یک حمله را نشان دهد.

۶ ـ fragbits : این گزینه با فرمت (fragbits:[+ یا * یا !][MDR]) برای چک کردن این مطلب که بیت‌های مربوط به تکه سازی بسته‌ی IP و بیت‌های رزرو بسته‌ی IP در بخش سرآیند IP سِت شده است یا نه، به کار می‌رود.

از بیت های مربوط به سرآیند بسته‌ی IP که در فرمت این گزینه ذکر شده است، M به معنای بیت تکه‌های بیشتر (More fragments)، D به معنای بیت عدم تکه سازی (Don’ t fragment) و R به معنای بیت رزرو (Reserved bit) است. همچنین علامت (+) به معنای بررسیِ سِت بودن تمام بیت‌های ذکر شده، (*) به معنای بررسیِ سِت بودن یکی از بیت‌های ذکر شده و (!) به معنای بررسیِ سِت نبودن تمام بیت‌های ذکر شده می‌باشد. مثلاً عبارت (frag bits:MD+;) به دنبال بسته‌های مشکوکی می گردد که در آنها هر دو بیت تکه سازی بیشتر (More fragments) و عدم تکه سازی (Don’t fragment) با هم سِت شده‌اند.

۷ ـ dsize : این گزینه با فرمت (disze:[< >] عدد [< > عدد]) امکان بررسی اندازه یا سایز بخش payload بسته را فراهم می‌کند. واضح است که یک بسته با اندازه‌ی payload بزرگ و غیر عادی احتمال یک حمله سرزیر بافر (buffer overflow) را نشان می‌دهد. مثلاً گزینه‌ی (dsize:300< >400;) به دنبال بسته‌های با اندازه‌ی payload بین ۳۰۰ و ۴۰۰ می‌گردد. (دقت داشته باشید که این گزینه در مورد دنباله‌ی بسته‌های بازسازی شده (stream rebuilt packets) کارا نیست و از کار می‌افتد.)

۸ ـ flags : این گزینه با فرمت (flags:[! یا * یا +]< FSRPAU120 >[, < FSRPAU120 >]) کاری شبیه به گزینه‌ی fragbits در IP را برای پرچمهای TCP انجام می‌دهد. در واقع این گزینه بررسی می‌کند که کدام پرچمها در ارتباط TCP فعال است. پرچمهای مورد استفاده عبارتند از: F برای FIN (پرچم اعلام پایان ارتباط) ـ S برای SYN (پرچم همزمان سازی) ـ R برای RST (پرچم قطع ارتباط) ـ P برای PSH (پرچم اعلام ارسال بسته) ـ A برای ACK (پرچم تأیید دریافت) ـ U برای URG (پرچم اشاره گر اضطراری) ـ 1 برای بیت رزرو شماره۱ ـ 2 برای بیت رزرو شماره۲ ـ 0 به معنای ست نشدن هیچ پرچم TCP. به علاوه این اصلاح کننده ها (modifier) نیز نوع تطبیق را اصلاح می‌کنند: (+) به معنای سِت شدن تمام بیت‌های ذکر شده، (*) به معنای سِت

شدن یکی از بیت‌های ذکر شده و (!) به معنای سِت نشدن بیت‌های ذکر شده. دقت کنید که آرگومان اختیاری دوم به معنای option mask یا در نظر نگرفتن بیت‌های ذکر شده در آرگومان دوم است.

به عنوان یک نمونه از کاربرد این گزینه، قانون زیر نسبت به ارتباطات TCP که پرچمهای SYN و FIN آنها، سِت شده است (بدون در نظر گرفتن بیت‌های رزرو شماره ۱ و شماره ۲) اعلام هشدار می‌کند:

```
alert tcp any any -> any any (flags:SF,12;)
```

۹ـ flow: این گزینه با فرمتِ

flow: [established یا stateless][, to_client یا to_server یا from_client یا from_server]

[, no_stream یا only_stream];*

در ارتباط با بازسازی دنباله TCP، تعیین جهت ترافیک مربوطه، تعیین وضعیت اتصال، و وضعیت بسته‌ها استفاده می‌شود. در ادامه فرمت گزینه را تشریح می کنیم:

آرگومان (stablished) نشانگر اتصال کامل ارتباط TCP است و گزینه‌ی (stateless) نشان می‌دهد که وصل شدن یا وصل نشدن ارتباط TCP اهمیتی ندارد. (کلیدواژه اخیر برای یافتن بسته‌هایی مناسب است که با هدف خراب کردن ماشین طراحی شده‌اند.) برای تعیین جهت ترافیک، آرگومان to_client نشانگر ارتباطی به مقصد client ، to_server نشانگر ارتباطی به مقصد server ، from_client نشانگر ارتباطی از مبدأ client ، و from_server نشانگر ارتباطی از مبدأ server می باشد. (با کمک این آرگومان می‌توان میان بسته‌های مربوط به مشتریهای $HOME_NET که از صفحات وب بازدید می کنند، با سرورهایی که $HOME_NET را اجرا می‌کنند، تفاوت قائل شد.) آرگومان no_stream به معنای عدم اعلام هشدار نسبت به دنباله‌ی بسته‌های بازسازی شده و only_stream به معنای اعلام هشدار تنها برای دنباله‌ی بسته‌های بازسازی شده است.

در پایین دو مثال مربوط به کاربرد این گزینه مشاهده می کنیم.

در قانون زیر، تنها ترافیک از clientهای خارج شبکه به داخل آن به مقصد پورت ۲۱ (پورت کنترلی FTP) مشکوک است.

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

در قانون زیر ترافیک از هر نوع (وصل شده یا نشده) از پورت صفر خارج شبکه (پورت رزرو شده‌ی غیر مجاز) به پورت صفر داخل شبکه مشکوک است.

```
alert tcp !$HOME_NET 0 -> $HOME_NET 0 (msg: "Port 0 TCP traffic"; \
    flow:stateless;)
```

۱۰ـ flow bits: این گزینه با فرمتِ

flow bits:[set یا unset یا toggle یا isset یا isnotset یا noalert][, نام حالت])

امکان تعقیب state یا حالتهای خاص از ارتباطات مختلف مخصوصاً ارتباط TCP را (که اکثراً توسط خودِکاربر نامگذاری شده اند) فراهم می کند.

آرگومان‌های گزینه به این شرح‌اند: (set) حالت ذکر شده را، سِت کرده و (unset)، آن را از سِت خارج می کند. (toggle) سِت بودن/نبودن حالت را هر چه بوده، عکس می‌کند. همچنین (isset) بررسی می‌کند که حالت مذکور سِت باشد و (isnotset) بررسی می‌کند که حالت مذکور سِت نباشد. (noalert) سبب می‌شود که قانون بدون توجه به کشف شدن/نشدن سایر گزینه‌ها دیگر هشداری تولید نکند. در پایین دو مثال از کاربرد گزینه فوق را می‌بینیم:

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

alert tcp any any -> any 143 (msg:"IMAP LIST"; content:"LIST";
  flowbits:isset,logged_in;)
```

۱۱ـ seq: این گزینه با فرمت (عدد :seq) شماره مسلسل ارتباط TCP (TCP Sequence Number) را بررسی می کند. مثلاً (seq:0;) به دنبال ارتباطات TCP با شماره مسلسل صفر می گردد.

۱۲ـ ack: این گزینه با فرمت (عدد: ack) شماره‌ی تأیید ارتباط TCP (TCP acknowledge number) را بررسی می کند. مثلاً عبارت (ack:0;) به دنبال ارتباطات TCP با شماره تأیید صفر می گردد.

۱۳ ـ window : این گزینه با فرمت (عدد [!] : window) سایز یا اندازه‌ی پنجره‌ی TCP را بررسی می‌کند. مثلاً عبارت (window:55808;) به دنبال ارتباط TCP با اندازه‌ی پنجره‌ی ۵۵۸۰۸ می‌گردد.

۱۴ـ itype : این گزینه با فرمت([عدد< یا >]عدد [< یا >] :itypc) مقدارمتغیر ICMP type را بررسی می‌کند. مثلاً عبارت (itype: >30;) به دنبال ارتباطات ICMP با نوع بزرگتر از ۳۰ می‌گردد.

۱۵ ـ icode : این گزینه با فرمت ([عدد< یا >]عدد [< یا >] :icode) مقدار متغیر ICMP code را بررسی می‌کند. مثلاً عبارت (icode: >30;) به دنبال ارتباطات ICMP با کد بزرگتر از ۳۰ می‌گردد.

۱۶ ـ icmp_id : این گزینه با فرمت (عدد :icmp_id)مقدار متغیر ICMP ID را بررسی می‌کند. مثلاً عبارت (icmp_id:0;) به دنبال ارتباطات ICMP با شناسه‌ی صفر می‌گردد. اهمیت این گزینه در آشکار کردن برنامه‌های با کانال پنهان است که از فیلدهای ICMP استاتیک برای ارتباط استفاده می‌کنند. یک مورد از این برنامه‌ها "stacheldraht" با حمله‌ی DDoS است.

۱۷ ـ icmp_seq : این گزینه با فرمت (عدد :icmp_seq) برای بررسی ICMP Sequence number یا شماره مسلسل ICMP به کار می‌رود. مثلاً عبارت (icmp_seq:0;) به دنبال ارتباطات ICMP با شماره مسلسل صفر می‌گردد. علت اهمیت این گزینه نیز شبیه مورد قبل در آشکار سازی برنامه‌های با کانال پنهان و با فیلدهای ICMP استاتیک (مثل Stacheldraht) است.

۱۸ ـ rpc : این گزینه با فرمت ([* یا شماره رویه] , [* یا شماره نسخه] , شماره کاربرد :rcp)، در یک تقاضای SUNRPC CALL مقادیر شماره کاربرد (application) و شماره نسخه (version) و شماره رویه (procedure) را بررسی می‌کندکه در آن برای آنکه شماره نسخه و شماره رویه شامل تمام موارد ممکن باشد، می‌توان از علامت (*) استفاده کرد. (منظور از RPC فراخوانی رویه از دور یا Remote Procedure call است که به معنای اجرای یک سابروتین روی یک کامپیوترِ دور از دسترس توسط برنامه‌ی یک کامپیوترِ در دسترس است بدون آنکه جزئیات این تعامل (interaction) به طور صریح کد شود. یکی از نمونه‌های بسیار رایج آنها

Open Network Comuting RPC یا ONC RPC محصول مشترک Sun Microsystems است که به آن SUN RPC نیز گفته می‌شود.)

در مثال زیر بررسی درخواست GETPORT برای پورت Portmapper به فرم RPC مشاهده می‌شود:

```
alert tcp any any -> any 111 (rpc: 100000,*,3;);
```

دقت کنید که جستجوهای مذکور در این گزینه نسبت به گزینه ی بسیار سریع content کندتر است و زمان اجرای کامل قانون را افزایش می‌دهند. لذا توصیه می شود که تا حد امکان از کاربرد گزینه rpc پرهیز شود و به جای آن از گزینه ی content استفاده شود.

۱۹ ـ ip_proto : این گزینه با فرمت (نام یا شماره [> < !] :ip_proto) امکان تشخیص نوع پروتکل را با بررسی سرآیند بسته IP فراهم می کند. مثلاً قانون زیر به دنبال بسته‌های تحت پروتکل IGMP می گردد:

```
alert ip any any -> any any (ip_proto:igmp;)
```

۲۰ ـ sameip : این گزینه‌ی بدون آرگومان با فرمت (sameip) امکان تشخیص یکسانی آدرسهای IP مبدأ و مقصد را فراهم می کند. مثلاً قانون زیر ترافیک بسته‌های IP با آدرسهای مبدأ و مقصد یکسان را دنبال می کند:

```
alert ip any any -> any any (sampeip;)
```

۶-۳-۴) گزینه های پس از تشخیص

۱ ـ log to : این گزینه با فرمت (" نام فایل" :log to) باعث می‌شود که تمام بسته‌هایی که طبق این قانون (که گزینه‌ی log to جزئی از آن است) مشکوک در نظر گرفته شده‌اند و نسبت به آنها اعلام هشدار شده است، در یک فایل ثبتی خاص با نام ذکر شده در متن گزینه، ثبت (log) شوند. این گزینه که به طور همزمان با ثبت باینری بسته‌ها فعال نمی‌شود، برای ترکیب داده‌های بدست آمده از فعالیت NMAP و پویش HTTP GCI بسیار مناسب است.

۲ ـ session : این گزینه با فرمت (session:[printable یا all]) برای استخراج داده‌ها از session یا جلسات TCP و فهمیدن محتوای بسته‌های ارسالی در ارتباطات Telnet، rlogin ، FTP بسیار مفید است. منظور

از printable در این گزینه کاراکترهایی هستند که تایپ شده و مشاهده می‌شوند و منظور از all تمامی کاراکترها (قابل مشاهده و غیر قابل مشاهده) می‌باشد. به عنوان یک مثال از کاربرد این گزینه، قانون زیر، رشته‌ها یا کاراکترهای قابل چاپ تمام ارتباطات TCP با پورت ۲۳ یا Telnet (ارسالی یا دریافتی) را ثبت می‌کند:

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

دقت کنید که این گزینه به جز در مورد فایلهای ثبتی از نوع باینری (pcap) زمان اجرای قانون را افزایش داده و سرعت نرم افزار را کم می‌کند.

۳ ـ resp : این گزینه با فرمت ([مکانیزم پاسخ ,] مکانیزم پاسخ :resp) این امکان را فراهم می‌کند که چنانچه بسته‌ای یک قانون را فعال کرد، به آن ارتباط کلاً پایان داده شود. این عملگر که در زمان نصب با قرار دادن عبارت (--enable-flexresp)جلوی عبارت configure فعال می‌شود، امکان ختم کردن چند گانه ی یک ارتباط در داخل یک گزینه (با چند بار استفاده از کلید واژه‌های مربوط به resp در متن قانون) را نیز دارد. انواع پایان دادنِ ارتباط در جدول زیر دیده می‌شود:

Option	Description
rst_snd	Send TCP-RST packets to the sending socket
rst_rcv	Send TCP-RST packets to the receiving socket
rst_all	Send TCP_RST packets in both directions
icmp_net	Send a ICMP_NET_UNREACH to the sender
icmp_host	Send a ICMP_HOST_UNREACH to the sender
icmp_port	Send a ICMP_PORT_UNREACH to the sender
icmp_all	Send all above ICMP packets to the sender

مثلاً قانون زیر تمام ارتباطات TCP به مقصد پورت ۱۵۲۴ را پس از اعلام هشدار، با ارسال بسته‌های RST به هر دوی ماشین‌های مبدأ و مقصد reset می‌کند :

```
alert tcp any any -> any 1524 (flags:S; resp:rst_all;)
```

۴ ـ react : این گزینه با فرمت ([اصلاح کننده‌ی اضافی برای واکنش,] اصلاح کننده اصلی برای واکنش :react) این امکان را برای کاربر نرم افزار فراهم می‌کند که در صورت تطبیق شدن یک بسته با قانون (فعال شدن قانون)، نسبت به آن بسته یا ارتباط واکنش نشان دهد. basic modifier یا اصلاح کننده‌ی اصلی برای واکنش که با کلید واژه‌های (block) مورد اشاره قرار می‌گیرد، ارتباطات تجاوزکارانه را به طور فعال قطع می‌کند (مثلاً از مرور کردن

صفحات سایت‌های مستهجن و یا سایت‌های جالبی مثل NewYork Times یا slashdot جلوگیری می‌کند) و react additional modifier یا اصلاح کننده‌ی اضافی برای واکنش، با کلید واژه‌ای (msg) یک اخطاریه قابل مشاهده برای مرورگر ارسال می‌کند. روشن است که می‌توان هر دو اصلاح کننده (اصلی و اضافی) را به کار برد؛ منتها این دقت لازم است که گزینه‌ی react باید آخرین گزینه در لیست گزینه‌های یک قانون باشد.

به عنوان یک مثال از کاربرد این گزینه، قانون زیر نسبت به هرگونه ارتباط TCP (ارسالی یا دریافتی) به صورت اینترنتی (پورت ۸۰ یا HTTP) با شبکه داخلی کلاس C از آدرس (192.168.1.0) که حاوی صفحه‌ی (bad.htm) باشد، اعلام هشدار کرده و ضمن بستن ارتباط، پیام اخطاریِ (! Not for children) را به مرورگرِ صفحه نشان می‌دهد:

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

این گزینه در هنگام نصب Flexible Response با نوشتن(--enable-flexresp) فعال می‌شود.

۵ ـ tag : این گزینه با فرمت ([direction] , واحد , تعداد , نوع :tag) این امکان را فراهم می‌کند که قانون علاوه بر بسته‌ای که آن را فعال می‌کند، بسته‌های بیشتری را ثبت کند. در اینصورت زمانی که یک قانون فعال شد، ترافیک بعدیِ مربوط به مبدأ یا مقصد برچسب گذاری می شود و برای post-attack traffic یا تحلیل ترافیک پس از حمله به plugin خروجی ارسال می‌شود. روشن است که plugin خروجی باید امکان کار کردن با این ترافیک را داشته باشد. (البته خروجی پایگاه داده‌ای یا database در حال حاضر این امکان را به خوبی فراهم نمی‌کند.)

در فرمت این گزینه آرگومان (*نوع*) یکی از کلید واژه‌های session (برای ثبت کل بسته‌های جلسه ای که قانون در آن نقض شده است) یا host (برای ثبت بسته‌های میزبانی که tag را فعال کرده است) را اختیار می کند و البته اصلاح کننده‌ی [direction] به همراه کلید واژه‌ی host قابل استفاده است. آرگومان (*تعداد*)، تعداد واحد را مشخص می‌کند وآرگومان (*واحد*) با یکی از دو کلید واژه‌ی (packets) یا (seconds) تعیین می‌کند که واحد آرگومان (*تعداد*) چیست. مثلاً در قانون زیر ۱۰ ثانیه اول از کلیه ارتباطات Telnet ثبت می‌شود:

```
alert tcp any any -> any 23 (flags:s,12; tag:session,10,seconds;)
```

۶-۳-۵) گزینه‌ی آستانه‌ی ثبت رخداد

گزینه‌ی Event Thresholding یا آستانه‌ی ثبت رخداد، تعداد دفعاتی را که یک رخداد خاص در یک فاصله زمانی معین می‌تواند ثبت شود، محدود می‌کند؛ به این ترتیب هم تعداد false alarm ها یا اخطارهای غلط کم می‌شود و هم نوشتن نسخه‌های جدیدتراز قوانین شلوغ (noisy rules) ممکن می گردد.

این گزینه با کلید واژه‌ی (threshold) به ۳ صورت اعمال می‌شود:

- ❖ (limit) که اولین m رخداد از کل رخدادهای مشابه در طول بازه زمانی مورد اشاره را ثبت می‌کند و سپس سایر رخدادهای مشابه را تا پایان بازه زمانی رها می‌کند.
- ❖ (threshold) که پس از m بار رخ دادن یک رخداد یکسان در طول بازی زمانی مورد اشاره، یک بار هشدار می‌دهد و مجدداً بازه زمانی را از اول در نظر می‌گیرد. روشن است که اگر در طول بازه زمانی مقرر، رخداد مورد نظر کمتر از m بار رخ دهد، آن رخداد اصلاً ثبت نمی‌شود.
- ❖ (both) که هر دوی عملیات بالا را همزمان اعمال می‌کند؛ یعنی چنانچه در طول بازه‌ی زمانی اشاره شده، یک رخداد خاص m بار رخ دهد، یکبار هشدار داده می‌شود و سپس بدون اینکه بازه‌ی زمانی جدیدی در نظر گرفته شود، وقوع آن رخداد تا پایان همان بازه‌ی زمانی بدون توجه رها می‌شود. بدیهی است که چنانچه آن رخداد در کل بازه‌ی زمانی m بار رخ ندهد، اصلاً چیزی ثبت نمی‌شود.

گزینه‌های آستانه‌ی ثبت رخداد می‌توانند به صورت یک قانون جداگانه (standalone) ذکر شوند و می‌توانند به صورت یک گزینه در داخل یک قانون ذکر شود. این دو فرمت در پایین دیده می‌شوند:

فرمت جداگانه

threshold gen_id مولد شماره شناسه , sig_id قانون شماره شناسه , type \

<limit یا threshold یا both > , track < by_src یا by_dst > , count تعداد , seconds تعداد ;

فرمت گزینه ای (داخل قانون)

threshold: type < timit یا threshold یا both> , track \

< by_src یا by_dst > , count تعداد , seconds تعداد ;

به لحاظ عملکردی، تفاوتی وجود ندارد که آستانه‌ی ثبت رخداد به صورت جداگانه یا داخلی برای یک قانون اعمال شود. اما به لحاظ منطقی و مفهومی، برخی از قوانین بدون وجود یک آستانه در داخل خود بی‌معنا می شوند ؛ مثلاً قانونی که تعداد loginهای بیشتر از ۵ بار در یک زمان کوتاه را مبنای هشدار قرار می‌هد، منطقاً باید از یک گزینه‌ی limit داخل خود استفاده کند.

دقت کنید که چنانچه آستانه به صورت جداگانه اعمال شود لازم است که این آستانه حاوی شماره قانون (SID) و شماره مولد هشدار (generator ID) باشد و البته به یک زوج (GID , SID) تنها می‌توان یک آستانه اعمال کرد؛ در غیر اینصورت نرم افزار با یک پیغام خطا به کار خود پایان می‌دهد. همچنین لازم به ذکر است که تعقیب ترافیک تنها می‌تواند بر مبنای آدرس IP مبدأ یا مقصد انجام بگیرد (و نه چیز دیگری مثل شماره پورت).

در سه مثال زیر، مواردی از آستانه رخداد جداگانه آمده است که در اولی ثبت رخدادهای یکسان از SID=1851 در هر ۶۰ ثانیه به یک بار محدود شده است؛ در دومی با هر سه بار رخداد یکسان مربوط به SID=1852 در ۶۰ ثانیه، یک پیام ثبت می‌شود و ۶۰ ثانیه بعدی آغاز می شود؛ در سومی با ۳۰ بار رخداد یکسان از SID=1853 در ۶۰ ثانیه، یک بار پیام ثبت می‌شود و هشدارهای مشابه تا پایان آن ۶۰ ثانیه بی‌توجه رها می شود:

```
threshold gen_id 1, sig_id 1852,   \
    type threshold, track by_src, \
    count 3, seconds 60

threshold gen_id 1, sig_id 1851, \
    type limit, track by_src,   \
    count 1, seconds 60

threshold gen_id 1, sig_id 1853, \
    type both, track by_src, \
    count 30, seconds 60
```

در پایین مثالهایی از آستانه‌ی ثبت رخداد به صورت کلید واژه‌ی داخل قانون دیده می‌شود. توضیحات شبیه حالت قبل است:

```
alert tcp $external_net any -> $http_servers $http_ports \
    (msg:"web-misc robots.txt access"; flow:to_server, established; \
    uricontent:"/robots.txt"; nocase; reference:nessus,10302; \
    classtype:web-application-activity; threshold: type limit, track \
    by_src, count 1 , seconds 60 ; sid:1000852; rev:1;)
```

```
alert tcp $external_net any -> $http_servers $http_ports \
    (msg:"web-misc robots.txt access"; flow:to_server, established; \
    uricontent:"/robots.txt"; nocase; reference:nessus,10302; \
    classtype:web-application-activity; threshold: type threshold. \
alert tcp $external_net any -> $http_servers $http_ports \
    (msg:"web-misc robots.txt access"; flow:to_server, established; \
    uricontent:"/robots.txt"; nocase; reference:nessus,10302; \
    classtype:web-application-activity; threshold: type both , track \
    by_dst, count 10 , seconds 60 ; sid:1000852; rev:1;)
```

می‌توان آستانه‌ی ثبت رخدادها را به صورت عمومی (global) بر تمام قوانین و حتی تمام مولدهای هشدار اعمال کرد. در اینصورت قانون شبیه به حالت آستانه‌ی ثبت رخداد جداگانه نوشته می‌شود که در آن برای نشان دادن عمومیت آستانه برای تمام قوانین، عبارت sig_id 0 درج می‌شود. می‌توان با قرار دادن gen_id 0 آستانه‌ی ذکر شده را به طور همزمان بر تمام مولدهای هشدار نیز اعمال کرد. البته دقت کنید که همواره آستانه‌ی خاصتر (آستانه‌ی جداگانه‌ی مخصوص یک قانون یا آستانه‌ی کلید واژه‌ای داخل قانون) برای تعیین آستانه ی ثبت رخداد یک قانون غالب است؛ یعنی اگر یک قانون خودش دارای آستانه باشد، آن آستانه بر آستانه‌ی عمومی (global) اولویت دارد.

در پایین مثالهایی از آستانه‌ی ثبت رخداد به صورت عمومی (global) دیده می‌شود. در اولی به ازای هر آدرس IP که در هر ۶۰ ثانیه یک قانون دلخواه را به تعداد دفعات دلخواهی فعال کند، تنها یکبار پیام هشدار ثبت می‌شود. (مقدار متغیر gen_id 1 فرض شده است.) ؛ در دومی به ازای هر آدرس IP که در هر ۶۰ ثانیه یک قانون دلخواه مربوط به یک مولد هشدار دلخواه را به تعداد دفعات دلخواهی فعال کند، تنها یکبار پیام هشدار ثبت می‌شود.

```
threshold gen_id 1, sig_id 0, type limit, track by_src, count 1, seconds 60

threshold gen_id 0, sig_id 0, type limit, track by_src, count 1, seconds 60
```

تذکر : در مبحث آستانه‌ی ثبت رخداد، دقت شود که تنظیم آستانه برای یک رخداد، تنها مربوط به ثبت رخدادهای واقع شده است و الا events یا رخدادهای مشکوک همچنان به طور عادی در Snort تولید می‌شوند.

۶-۴) جلوگیری از رخداد

قانون suppress با دو آرگومان اجباری و دو آرگومان اختیاری یک قانون جداگانه (standalone) با فرمت زیر است :

suppress gen_id شماره شناسه مولد , sig_id شماره شناسه قانون \

; [آدرس IP با / بدون بیت‌های IP از نوع mask شده , < by_dst یا by_src < track ,] ,

در واقع قانونِ Event Suppression یا جلوگیری از رخداد این امکان را فراهم می‌کند که یک قانون با SID خاص و GID خاص بدون آنکه از مجموعه قوانین حذف شود ، برای همه‌ی آدرسهای IP از عملکرد ساقط شود و یا اینکه برای یک آدرس IP خاص یا یک دسته از آدرسهای IP (بر اساس مبدأ / مقصد بودنشان در بسته) غیر فعال گردد. دقت کنید که می‌توان چندین قانون جلوگیری از رخداد (event suppression) را (حتی همراه با یک قانون threshold) به یک SID و GID خاص اعمال کرد.

قوانین زیر مثالهایی از قانون suppress است. در مورد اول رخداد به طور کامل، در دومی رخداد برای یک IP خاص و در سومی رخداد برای یک کلاس C از آدرسهای IP جلوگیری شده است:

```
suppress gen_id 1, sig_id 1852:
suppress gen_id 1, sig_id 1852, track by_src, ip 10.1.1.54
suppress gen_id 1, sig_id 1852, track by_dst, ip 10.1.1.0/24
```

۶-۵) ثبت چند رخداد یا صف بندی رخدادها

با قانونِ Multi-event logging یا Event Queue امکان ثبت چند رخداد مختلف (با اولویت یا طولهای مختلف) برای هر یک از بسته‌ها فراهم می شود. فرمت قانون به صورت

config event_queue:[max_event اندازه] [log اندازه][order_events نوع]

است و در آن منظور از (max_events) حداکثر طول صف رخدادهاست (به طور پیش فرض ۸) و منظور از (log) حداکثر تعداد رخدادهای قابل ثبت برای هر بسته است (به طور پیش فرض ۳) و منظور از

(order_events) نحوه‌ی مرتب کردن رخدادها در صف است که به یکی از دو صورت priority (یعنی اختصاص اولویت رخدادها با اولویت ۱ برای بالاترین اولویت) و content_length (یعنی بر اساس طول محتوای بسته پیش از کدگشایی و پیش پردازش) انجام می‌شود. (به طور پیش فرض content_length).

در زیر مثالهایی از تعریف صف رخدادها را می‌بینیم. در اولی طول صف به ۱۰ و نحوه مرتب کردن به (priority) تغییر کرده است؛ در دومی تنها نوع مرتب کردن به priority تغییر کرده و در سومی حداکثر تعداد رخداد قابل چاپ به ۲ رخداد تغییر کرده است:

```
config event_queue: max_queue 10 log 3 order_events content_length
config event_queue: order_events priority
config event_queue: log 2
```

۶-۶) چند مثال

در این مرحله، این امکان را یافته ایم که یک قانون نوشته شده را معنا و تفسیر کنیم یا اینکه بر اساس سیاستهای کلی کشف مزاحمت قوانین جدیدی بنویسیم. در ادامه، به منظور مرور و تمرین موارد ذکر شده پیرامون قوانین، چند قانون Snort را معنا و بررسی می کنیم:

مثال ۱

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

این قانون نسبت به همه بسته‌های تحت پروتکل TCP (با هر آدرس IP مبدأ و هر پورتی) که مقصد آنها پورت شماره ۱۴۳ (پورت IMAP4 برای بازیابی email) از یک کلاس C آدرسهای IP از 192.168.1.0 تا 192.168.1.255 باشد و دارای محتوای (90C8 C0FF FFFF/bin/sh) باشند، اعلام هشدار کرده و پیغام IMAP bufferoverflow! را چاپ می کند.

مثال ۲

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

این قانون نسبت به همه ارتباطهایی (با هر آدرسIP مبدأ و هر پورتی) که با شناسه کاربری root به پورت ۲۱ (پورت کنترل FTP) از کلاس C از آدرس 192.168.1.0 تا 192.168.1.255 متصل شوند، اعلام هشدار کرده و پیغام FTP root user access attempt را چاپ می کند.

مثال ۳

```
alert tcp any any -> 192.168.1.0/24 any (flags:SF; msg:"Possible SYN FIN Scan";)
```

این قانون نسبت به همه ارتباطات تحت پروتکل TCP (با هر آدرس IP مبدأ و هر پورتی) که به هر پورت باز کلاس C از آدرس 192.168.1.0 تا 192.168.1.255 متصل شوند و پرچم SYN و FIN آنها روشن باشد، اعلام هشدار کرده و پیغام هشدار Possible SYN FIN Scan را چاپ می کند.

۷) امنیت Snort

مهمترین چالش های امنیتی Snort (که در مورد سایر نرم افزارهای IDS نیز مطرح می باشد) به این شرح است:

- حمله گریز (evasion) با بسته های محو شده؛ مهاجم به جای ارسال یکجای بسته ی تهاجمی، بسته هایی را که هر کدام بخشی از داده های بسته اصلی را دارا هستند، از مسیرهای مختلف و حتی از طریق ارتباطات بی سیم (wireless) ارسال می کند تا از کمند IDS بگریزد.
- حمله سرریز بافر؛ مثلاً با یک میلیارد بسته قلابی از نوع SQL slammer .
- حمله با ارسال بسته های قلابی؛کور کردن نرم افزار با ارسال نویز تصادفی(random snort noise) مبتنی بر خودِ قوانین Snort . دو مقاله پیرامون فریب Snort با استفاده از بسته های قلابی در آدرسهای الکترونیکی زیر ارائه شده است:

http://packetstormsecurity.nl/distributed/stick.htm

http://www.stolenshoes.net/sniph/index.html

❖ ضعف عملکرد در صورت مواجهه با حجم زیادی از بسته ها

به هر حال، در مواجهه با این مسائل امنیتی تنها راه موجود اینست که IDS امکان تشخیص موارد عادی و مشکوک را داشته باشد و این جز با به هنگام سازی قوانین IDS مبتنی بر تجارب انسانی و یا یافته های ماشینهای هوشمند ممکن نیست.

بحث کوتاه بالا در مورد آسیب پذیری و امنیت Snort تنها گوشه ای از منابع و مقالات متعدد پیرامون این موضوع است. به عنوان نمونه، یک مقاله مروری کامل در این مورد در آدرس زیر قابل دسترسی است:

http://www.linuxdevcenter.com/pub/a/linux/2003/06/02/snort.html

۸) سایر نکات در مورد Snort

❖ با هر بار اجرای Snort یک آمار کامل در مورد انواع بسته های تحلیل شده و همچنین تعداد بسته های پویش نشده ارائه می شود. به علاوه با به روز رسانی kernel ها و libpcap ، امکان ارائه آمار دقیقی از انواع بسته های تحلیل نشده نیز فراهم می شود.

❖ علل عدم ثبت رخدادها در پایگاه داده به شرح زیر است: ۱- عدم تنظیم پایگاه داده در فایل تنظیمات (snort.conf) ۲- قدیمی بودن نسخه پایگاه داده ۳- وجود سوییچی مثل A- یا s- در خط دستور، که بر تنظیمات فایل snort.conf ۴- وجود اشکال در تنظیمات داخلی پایگاه داده

❖ برای ثبت پویش پورت (port scan) لازم است که در فایل تنظیمات، تنظیم پایگاه داده ی خروجی از log به alert تغییر داده شود.

❖ در نوشتن قوانین دقت کنید که گزینه‌های زیر گسسته هستند و عموماً باید به عنوان اولین گزینه‌ها در متن قانون ذکر شوند: disize,flags,flow,fragbits,icmp_id,icmp_seq,icode,id, ipopts,

ip_proto,itype,seq,session ,tos,ttl,ack, window, resp, sameip

❖ چنانچه نرم افزار Snort با نرم افزارهای دیگری مثل SnortSnarf و Sguil و OSSIM و BASE همراه شود، امکان نمایش گرافیکی داده های مزاحم نیز فراهم می شود.

❖ مجموعه ای از مراجع و linkهای مناسب در مورد سیستم IDS و مخصوصاً Snort در پایین آمده است:

• SANS (http://www.sans.org)
• Usenix (http://www.usenix.org/event/)
• Networld/Interop (http://www.key3media.com/interop/)
• CanSecWest (http://www.cansecwest.com)
• Snort 2.0 Intrusion Detection: Brian Caswell, Jay Beale, Foster/Faircloth, Syngress, Feb. 2003

Title	Author(s)	Publisher	ISBN
Snort: The Complete Guide to Intrusion Detection	Jeff Nathan, Dragos Ruiu, & Jed Haile	Wiley & Sons	0471455970
Intrusion Detection with Snort: Advanced IDS Techniques	Rafeeq Rehman	Prentice Hall	I0131407333
Snort Intrusion Detection	Ryan Russell	Syngress Media	1931836744
Snort Intrusion Detection	Jack Koziol	New Riders	157870281X
Network Intrusion Detection: An Analyst's Handbook	Stephen Northcutt	New Riders	0735708681
Intrusion Signatures and Analysis	Stephen Northcutt	New Riders	0735710635
TCP/IP Illustrated, Volume 1 The Protocols	W. Richard Stevens	Addison-Wesley	0201633469
Intrusion Detection	Rebecca G. Bace	MacMillan Technical Publishing	1578701856

ضمائم

ض-۱) لیست SID های مربوط به مولدهای پیش پردازشگرهای Snort

```
# $Id$
# GENERATORS -> msg map
# Format: generatorid || □lerted || MSG

1 || 1 || snort general alert
2 || 1 || tag: Tagged Packet
3 || 1 || snort dynamic alert
100 || 1 || spp_portscan: Portscan Detected
100 || 2 || spp_portscan: Portscan Status
100 || 3 || spp_portscan: Portscan Ended
101 || 1 || spp_minfrag: minfrag alert
102 || 1 || http_decode: Unicode Attack
102 || 2 || http_decode: CGI NULL Byte Attack
102 || 3 || http_decode: large method attempted
102 || 4 || http_decode: missing uri
102 || 5 || http_decode: double encoding detected
102 || 6 || http_decode: illegal hex values detected
102 || 7 || http_decode: overlong character detected
103 || 1 || spp_defrag: Fragmentation Overflow Detected
103 || 2 || spp_defrag: Stale Fragments Discarded
104 || 1 || spp_anomsensor: SPADE Anomaly Threshold Exceeded
104 || 2 || spp_anomsensor: SPADE Anomaly Threshold Adjusted
105 || 1 || spp_bo: Back Orifice Traffic Detected
105 || 2 || spp_bo: Back Orifice Client Traffic Detected
105 || 3 || spp_bo: Back Orifice Server Traffic Detected
105 || 4 || spp_bo: Back Orifice Snort Buffer Attack
106 || 1 || spp_rpc_decode: Fragmented RPC Records
106 || 2 || spp_rpc_decode: Multiple Records in one packet
106 || 3 || spp_rpc_decode: Large RPC Record Fragment
106 || 4 || spp_rpc_decode: Incomplete RPC segment
106 || 5 || spp_rpc_decode: Zero-length RPC Fragment
110 || 1 || spp_unidecode: CGI NULL Attack
110 || 2 || spp_unidecode: Directory Traversal
110 || 3 || spp_unidecode: Unknown Mapping
110 || 4 || spp_unidecode: Invalid Mapping
111 || 1 || spp_stream4: Stealth Activity Detected
111 || 2 || spp_stream4: Evasive Reset Packet
111 || 3 || spp_stream4: Retransmission
111 || 4 || spp_stream4: Window Violation
111 || 5 || spp_stream4: Data on SYN Packet
111 || 6 || spp_stream4: Full XMAS Stealth Scan
111 || 7 || spp_stream4: SAPU Stealth Scan
111 || 8 || spp_stream4: FIN Stealth Scan
111 || 9 || spp_stream4: NULL Stealth Scan
111 || 10 || spp_stream4: NMAP XMAS Stealth Scan
111 || 11 || spp_stream4: VECNA Stealth Scan
111 || 12 || spp_stream4: NMAP Fingerprint Stateful Detection
111 || 13 || spp_stream4: SYN FIN Stealth Scan
111 || 14 || spp_stream4: TCP forward overlap detected
111 || 15 || spp_stream4: TTL Evasion attempt
111 || 16 || spp_stream4: Evasive retransmitited data attempt
111 || 17 || spp_stream4: Evasive retransmitited data with the data split
attempt
```

```
111 || 18 || spp_stream4: Multiple acked
111 || 19 || spp_stream4: Shifting to Emegency Session Mode
111 || 20 || spp_stream4: Shifting to Suspend Mode
111 || 21 || spp_stream4: TCP Timestamp option has value of zero
111 || 22 || spp_stream4: Too many overlapping TCP packets
111 || 23 || spp_stream4: Packet in established TCP stream missing ACK
111 || 24 || spp_stream4: Evasive FIN Packet
112 || 1 || spp_arpspoof: Directed ARP Request
112 || 2 || spp_arpspoof: Etherframe ARP Mismatch SRC
112 || 3 || spp_arpspoof: Etherframe ARP Mismatch DST
112 || 4 || spp_arpspoof: ARP Cache Overwrite Attack
113 || 1 || spp_frag2: Oversized Frag
113 || 2 || spp_frag2: Teardrop/Fragmentation Overlap Attack
113 || 3 || spp_frag2: TTL evasion detected
113 || 4 || spp_frag2: overlap detected
113 || 5 || spp_frag2: Duplicate first fragments
113 || 6 || spp_frag2: memcap exceeded
113 || 7 || spp_frag2: Out of order fragments
113 || 8 || spp_frag2: IP Options on Fragmented Packet
113 || 9 || spp_frag2: Shifting to Emegency Session Mode
113 || 10 || spp_frag2: Shifting to Suspend Mode
114 || 1 || spp_fnord: Possible Mutated GENERIC NOP Sled detected
114 || 2 || spp_fnord: Possible Mutated IA32 NOP Sled detected
114 || 3 || spp_fnord: Possible Mutated HPPA NOP Sled detected
114 || 4 || spp_fnord: Possible Mutated SPARC NOP Sled detected
115 || 1 || spp_asn1: Indefinite ASN.1 length encoding
115 || 2 || spp_asn1: Invalid ASN.1 length encoding
115 || 3 || spp_asn1: ASN.1 oversized item, possible overflow
115 || 4 || spp_asn1: ASN.1 spec violation, possible overflow
115 || 5 || spp_asn1: ASN.1 Attack: Datum length > packet length
116 || 1 || snort_decoder: Not Ipv4 datagram!
116 || 2 || snort_decoder: WARNING: Not Ipv4 datagram!
116 || 3 || snort_decoder: WARNING: hlen < IP_HEADER_LEN!
116 || 4 || snort_decoder: Bad Ipv4 Options
116 || 5 || snort_decoder: Truncated Ipv4 Options
116 || 45 || snort_decoder: TCP packet len is smaller than 20 bytes!
116 || 46 || snort_decoder: TCP Data Offset is less than 5!
116 || 47 || snort_decoder: TCP Data Offset is longer than payload!
116 || 54 || snort_decoder: Tcp Options found with bad lengths
116 || 55 || snort_decoder: Truncated Tcp Options
116 || 56 || snort_decoder: T/TCP Detected
116 || 57 || snort_decoder: Obsolete TCP options
116 || 58 || snort_decoder: Experimental TCP options
116 || 95 || snort_decoder: Truncated UDP Header!
116 || 96 || snort_decoder: Invalid UDP header, length field < 8
116 || 97 || snort_decoder: Short UDP packet, length field > payload length
116 || 105 || snort_decoder: ICMP Header Truncated!
116 || 106 || snort_decoder: ICMP Timestamp Header Truncated!
116 || 107 || snort_decoder: ICMP Address Header Truncated!
116 || 108 || snort_decoder: Unknown Datagram decoding problem!
116 || 109 || snort_decoder: Truncated ARP Packet!
116 || 110 || snort_decoder: Truncated EAP Header!
116 || 111 || snort_decoder: EAP Key Truncated!
116 || 112 || snort_decoder: EAP Header Truncated!
116 || 120 || snort_decoder: WARNING: Bad PPPOE frame detected!
116 || 130 || snort_decoder: WARNING: Bad VLAN Frame!
116 || 131 || snort_decoder: WARNING: Bad LLC header!
116 || 132 || snort_decoder: WARNING: Bad Extra LLC Info!
116 || 133 || snort_decoder: WARNING: Bad 802.11 LLC header!
116 || 134 || snort_decoder: WARNING: Bad 802.11 Extra LLC Info!
116 || 140 || snort_decoder: WARNING: Bad Token Ring Header!
```

```
116 || 141 || snort_decoder: WARNING: Bad Token Ring ETHLLC Header!
116 || 142 || snort_decoder: WARNING: Bad Token Ring MRLEN Header!
116 || 143 || snort_decoder: WARNING: Bad Token Ring MR Header!
116 || 150 || snort_decoder: Bad Traffic Loopback IP!
116 || 151 || snort_decoder: Bad Traffic Same Src/Dst IP!
117 || 1 || spp_portscan2: Portscan detected!
118 || 1 || spp_conversation: Bad IP protocol!
119 || 1 || http_inspect: ASCII ENCODING
119 || 2 || http_inspect: DOUBLE DECODING ATTACK
119 || 3 || http_inspect: U ENCODING
119 || 4 || http_inspect: BARE BYTE UNICODE ENCODING
119 || 5 || http_inspect: BASE36 ENCODING
119 || 6 || http_inspect: UTF-8 ENCODING
119 || 7 || http_inspect: IIS UNICODE CODEPOINT ENCODING
119 || 8 || http_inspect: MULTI_SLASH ENCODING
119 || 9 || http_inspect: IIS BACKSLASH EVASION
119 || 10 || http_inspect: SELF DIRECTORY TRAVERSAL
119 || 11 || http_inspect: DIRECTORY TRAVERSAL
119 || 12 || http_inspect: APACHE WHITESPACE (TAB)
119 || 13 || http_inspect: NON-RFC HTTP DELIMITER
119 || 14 || http_inspect: NON-RFC DEFINED CHAR
119 || 15 || http_inspect: OVERSIZE REQUEST-URI DIRECTORY
119 || 16 || http_inspect: OVERSIZE CHUNK ENCODING
119 || 17 || http_inspect: UNAUTHORIZED PROXY USE DETECTED
119 || 18 || http_inspect: WEBROOT DIRECTORY TRAVERSAL
120 || 1 || http_inspect: ANOMALOUS HTTP SERVER ON UNDEFINED HTTP PORT
121 || 1 || flow-portscan: Fixed Scale Scanner Limit Exceeded
121 || 2 || flow-portscan: Sliding Scale Scanner Limit Exceeded
121 || 3 || flow-portscan: Fixed Scale Talker Limit Exceeded
121 || 4 || flow-portscan: Sliding Scale Talker Limit Exceeded
122 || 1 || portscan: TCP Portscan
122 || 2 || portscan: TCP Decoy Portscan
122 || 3 || portscan: TCP Portsweep
122 || 4 || portscan: TCP Distributed Portscan
122 || 5 || portscan: TCP Filtered Portscan
122 || 6 || portscan: TCP Filtered Decoy Portscan
122 || 7 || portscan: TCP Filtered Portsweep
122 || 8 || portscan: TCP Filtered Distributed Portscan
122 || 9 || portscan: IP Protocol Scan
122 || 10 || portscan: IP Decoy Protocol Scan
122 || 11 || portscan: IP Protocol Sweep
122 || 12 || portscan: IP Distributed Protocol Scan
122 || 13 || portscan: IP Filtered Protocol Scan
122 || 14 || portscan: IP Filtered Decoy Protocol Scan
122 || 15 || portscan: IP Filtered Protocol Sweep
122 || 16 || portscan: IP Filtered Distributed Protocol Scan
122 || 17 || portscan: UDP Portscan
122 || 18 || portscan: UDP Decoy Portscan
122 || 19 || portscan: UDP Portsweep
122 || 20 || portscan: UDP Distributed Portscan
122 || 21 || portscan: UDP Filtered Portscan
122 || 22 || portscan: UDP Filtered Decoy Portscan
122 || 23 || portscan: UDP Filtered Portsweep
122 || 24 || portscan: UDP Filtered Distributed Portscan
122 || 25 || portscan: ICMP Sweep
122 || 26 || portscan: ICMP Filtered Sweep
122 || 27 || portscan: Open Port
123 || 1 || frag3: IP Options on fragmented packet
123 || 2 || frag3: Teardrop attack
123 || 3 || frag3: Short fragment, possible DoS attempt
123 || 4 || frag3: Fragment packet ends after defragmented packet
```

```
123 || 5 || frag3: Zero-byte fragment
123 || 6 || frag3: Bad fragment size, packet size is negative
123 || 7 || frag3: Bad fragment size, packet size is greater than 65536
123 || 8 || frag3: Fragmentation overlap
124 || 1 || smtp: Attempted command buffer overflow
124 || 2 || smtp: Attempted data header buffer overflow
124 || 3 || smtp: Attempted response buffer overflow
124 || 4 || smtp: Attempted specific command buffer overflow
124 || 5 || smtp: Unknown command
124 || 6 || smtp: Illegal command
125 || 1 || ftp_pp: Telnet command on FTP command channel
125 || 2 || ftp_pp: Invalid FTP command
125 || 3 || ftp_pp: FTP parameter length overflow
125 || 4 || ftp_pp: FTP malformed parameter
125 || 5 || ftp_pp: Possible string format attempt in FTP command/parameter
125 || 6 || ftp_pp: FTP response length overflow
125 || 7 || ftp_pp: FTP command channel encrypted
125 || 8 || ftp_pp: FTP bounce attack
126 || 1 || telnet_pp: Telnet consecutive AYT overflow
126 || 2 || telnet_pp: Telnet data encrypted
131 || 1 || dns: Obsolete DNS Rdata Type
131 || 2 || dns: Experimental DNS Rdata Type
131 || 3 || dns: Client Rdata TXT Overflow
```

ض-۲) لیست اسامی قوانین Snort

attack-responses.rules	local.rules	shellcode.rules
backdoor.rules	misc.rules	smtp.rules
bad-traffic.rules	multimedia.rules	snmp.rules
chat.rules	mysql.rules	sql.rules
classification.config	netbios.rules	telnet.rules
classification.config-sample	nntp.rules	tftp.rules
reference.config-sample	ddos.rules	oracle.rules
virus.rules	info.rules	scan.rules
deleted.rules	other-ids.rules	web-attacks.rules
dns.rules	p2p.rules	web-cgi.rules
dos.rules	policy.rules	web-client.rules

experimental.rules	pop2.rules	web-coldfusion.rules
exploit.rules	pop3.rules	web-frontpage.rules
finger.rules	porn.rules	web-iis.rules
ftp.rules	reference.config	web-misc.rules
icmp-info.rules	web-php.rules	imap.rules
icmp.rules	rpc.rules	x11.rules
rservices.rules		

مراجع

1. http://snort.org/docs/ Snort ™Users Manual 2.6.0, The Snort Project, May 23, 2006
2. http://snort.org/docs/ The Snort FAQ ,The Snort Core Team
3. http://snort.org/docs/ Snort 2.2.0 راهنمای نصب و تنظیم
4. http://www.fata.ir/bulletin/archives/000007.php