Санкт-Петербург, ул. Седова, 80, телефон (812) 449-0770, факс (812) 449-0771, E-mail: info@bilim.com

Правила Snort

Основы

Snort использует простой язык описания правил с широкими возможностями и достаточной гибкостью. Ниже приводится множество примеров, которые помогут разобраться с правилами Snort.
Большинство правил Snort записывается в одну строку. В прежних версиях программы (до 1.8) запись правила в одну строку была обязательной, но в современных вариантах программы допускается запись одного правила в несколько строк, если все строки, за исключением последней, завершаются символом \.
Каждое правило Snort делятся на две части – заголовков и опции. Заголовок правила включает действие, протокол, IP-адреса и маски для отправителя и получателя, а также номера портов отправителя и получателя. Опции правила включают сообщение и информацию о том, какие части пакета следует проверить, чтобы определить следует ли применять по отношению к пакеты заданное действие.
Ниже показан пример простого правила Snort

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

Опции правила указываются в круглых скобках, слова, за которыми следует двоеточие, являются ключевыми.
При соблюдении всех указанных в правиле условий по отношению к пакету выполняется заданное правилом действие (генерация сигнала в приведенном примере).

Заголовок правила

Заголовок правила имеет вид

```
<действие> <протокол> <отправитель> <порт> <направление> <получатель> <порт>
```

Действие

Заголовок правила содержит ключевое слово, которое определяет действие, выполняемое при совпадении всех заданных правилом условий. Действие всегда указывается первым и может быть одним из ключевых слов

1. **alert** – генерировать сигнал с использованием выбранного метода и записать информацию о пакеты в журнальный файл;
2. **log** – записать информацию о пакеты в журнальный файл;
3. **pass** – пропустить (игнорировать) пакет;
4. **activate** – генерировать сигнал и активизировать другое динамическое правило;
5. **dynamic** – правило не выполняет никаких действий до его активации с помощью действия activate в другом правиле, а при активации действует как log.

Можно также создавать свои типы правил (ruletype) и связывать с ними подключаемые модули (plugin), используя в дальнейшем такие правила как действия Snort. В приведенном примере создается тип правила для записи пакетов

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

Во втором примере создается тип правила, который будет записывать информацию в syslog и базу данных MySQL

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

Протокол

Следующее поле заголовка указывает протокол. Snort в настоящее время различает 4 протокола - tcp, udp, icmp и ip. В будущем планируется включить также протоколы ARP, IGRP, GRE, OSPF, RIP, IPX и т. п.

Адреса IP

После протокола в правиле указываются адреса и номера портов для данного правила. Ключевому слову **any** будут соответствовать все адреса IP (0.0.0.0/0). Snort не поддерживает механизма определения адресов по именам хостов, поэтому в правилах должны указываться адреса IP или блоки CIDR [RFC1518]. Блок CIDR показывает префикс сети и размер маски, которая будет применяться правилом к адресам во всех пакетах для проверки соответствия указанному префиксу. Блок CIDR /24 указывает сеть класса C, /16 – класса B, а /32 указывает адрес отдельного хоста. Например, запись 192.168.1.0/24 будет соответствовать адресам в диапазоне 192.168.1.0 - 192.168.1.255. Нотация CIDR обеспечивает краткий и удобный способ представления непрерывных блоков сетевых адресов.

Приведенному ниже правилу будут соответствовать пакеты, отправленные с любого (any) адреса в сеть класса C 192.168.1.0

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

Применительно к адресам и блокам может использоваться оператор отрицания !. При использовании этого оператора правилу будут соответствовать пакеты, которые не попадают в указанный диапазон адресов. Например, правилу

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

будут соответствовать пакеты, отправленные в сеть класса C 192.168.1.0 из всех остальных сетей (не 192.168.1.0/24).

Адреса можно задавать также в виде списка, заключенного в квадратные скобки и разделенного запятыми. Правилу

```
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:"mountd access";)
```

будут соответствовать пакеты, направленные в сети класса C 192.168.1.0 и 10.1.1.0 из всех прочих сетей.

Номера портов

Номера портов можно задавать в виде конкретного значения, диапазона, списка или ключевого слова any (любой порт). Для портов также поддерживается оператор отрицания. Для задания диапазона указываются верхний и нижний пределы, разделенные двоеточием (:). Граничные значения включаются в диапазон. Правилу

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

будут соответствовать все пакеты UDP, адресованные в порты с 1 по 1024 хостов сети класса C 192.168.1.0. Если одна из границ диапазона не задана вместо нее используется минимальный (0) или максимальный (65535) номер порта. Правилу

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

будут соответствовать пакеты TCP, адресованные в порты с номерами не более 6000 сети класса C 192.168.1.0. Более сложному правилу

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

будут соответствовать пакеты TCP, направленные из портов с номерами не более 1024 в порты с номерами от 500 и выше всех хостов сети класса C 192.168.1.0. Оператор отрицания позволяет задать соответствие всем номерам портов за исключением указанных в правиле. Например, правилу

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

будут соответствовать пакеты TCP направленные с любого адреса в сеть класса C 192.168.1.0 и адресованные в любые порты, за исключением портов X Window (6000 – 6010). Отметим, что оператор отрицания недопустимо использовать применительно к ключевому слову any.

Оператор направления

Оператор направления -> показывает направление передачи трафика для данного правила. Адреса и порт слева от этого оператора относятся к отправителю, а справа – к получателю пакетов. Можно также создавать "двунаправленные" правила с помощью оператора <>. В этом случае каждая из пар "адрес-порт" будет трактоваться программой Snort как отправитель и как получатель. Такие правила удобны для анализа пакетов в сеансовых соединениях (например, POP3). Пример двунаправленного правила показан ниже.

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

В соответствии с этим правилом будут записываться все пакеты, адресованные в порт telnet каждого хоста сети класса C 192.168.1.0 с любого хоста за пределами этой сети, а также все пакеты, исходящие из порта telnet хостов сети 192.168.1.0 и адресованные в другие сети.

Отметим, что использование оператора <- недопустимо в правилах Snort.

Правила Activate/Dynamic

Пары правил activate/dynamic существенно расширяют возможности Snort. Вы можете с помощью одного правила (activate) активизировать при наступлении определенных условий другое правило, действие которого будет выполнено для заданного числа пакетов. Это очень полезно в тех случаях, когда вы хотите сохранить некоторое количество пакетов при возникновении того или иного события. Правила активизации похожи на правила alert, но включают добавочное поле activates. Динамические правила похожи на правила log, но включают два добавочных поля – activated_by и count. Все добавочные поля правил activate/dynamic являются обязательными.

Правила activate подобны правилам alert, но кроме генерации сигнала говорят программе Snort о необходимости добавления (активизации) другого правила при выполнении заданных условий. Правила dynamic подобны правилам log, но включаются только при выполнении правила activate с заданным идентификатором. Ниже приведен пример пары правил с активизацией по условию.

```
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;)
```

Эта пара правил говорит Snort о необходимости генерации сигнала при обнаружении переполнения буфера IMAP и записи следующих 50 пакетов, адресованных в порт 143 из сетей за пределами $HOME_NET на хосты $HOME_NET. Если переполнение буфера произойдет, такая пара правил позволит записать 50 (или иное число) следующих пакетов, адресованных в тот же порт для их последующего анализа.

Опции правил

Опции правил являются важнейшей частью работы Snort. Для разделения опций в правилах Snort используется точка с запятой (;). Ключевые слова опций отличаются от аргументов двоеточием (:).

Существует 4 основных категории опций правил.

meta-data
: информация о правиле, не оказывающая влияния на детектирование пакетов и выполняемые по отношению к ним операции;

payload
: опция просмотра поля данных пакета (packet payload)

non-payload
: опция просмотра служебных полей пакета;

post-detection
: опция, указывающая, что нужно сделать после выполнения заданных для правила условий.

Опции Meta-Data

msg

Опция msg говорит машине записи в журнальный файл и генерации сигналов о необходимости включения текстового сообщения в запись журнального файла или дампа пакета. Эта опция представляет собой просто текстовую строку с использованием \ в качестве escape-символа для задания символов, имеющих специальное значение в правилах Snort (например, символ ;).

Формат

```
msg: "<текст сообщения>";
```

reference

Ключевое слово reference позволяет включать в правила ссылки на внешние системы идентификации атак. Подключаемый модуль в настоящее время поддерживает несколько таких систем, а также уникальные URL (см таблицу 1). Этот модуль может использоваться подключаемыми модулями вывода (output plugin) для предоставления дополнительной информации в генерируемых сигналах.

Не забывайте также систему индексации описаний сигналов на основе sid, на сайте http://www.snort.org/snort-db/ (см. ниже).

Таблица 1

Система	*Ссылка*
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	http://www.whitehats.com/info/IDS
mcafee	http://vil.nai.com/vil/dispVirus.asp?virus_k=
url	http://

Format

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

Ниже приведен пример правил с использованием ссылок

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

sid

Ключевое слово sid предназначено для идентификации правил Snort. Значения идентификаторов используются подключаемыми модулям вывода. Опцию следует использовать вместе с ключевым словом rev (см. ниже). В правилах глобального значения, распространяемых с программой Snort, используются значения sid в диапазоне от 100 до 1000000. Значения меньше 100 зарезервированы, а значения превышающие 1 000 000, предназначены для локального использования (идентификации ваших собственных правил).

Файл sid-msg.map содержит список сигналов для различных значений sid, используемых правилами Snort. Эта информация может быть полезна при последующей обработке сигналов, поскольку позволяет получить текст сообщения по его идентификатору.

Формат

```
sid: <идентификатор правила>;
```

Ниже приведен пример правила с идентификатором 1000983.

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

rev

Ключевое слово sid служит для идентификации правил Snort, а используемое вместе с ним ключевое слово rev позволяет указать номер ревизии (версии) правила с данным идентификатором. Эту опцию следует использовать совместно с ключевым словом sid (см. выше).

Формат

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

Ниже приведен пример правила с идентификатором 1000983 ревизии 1.

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

classtype

Ключевое слово classtype указывает категорию сигнала в соответствии с классом атаки по частоте использования и важности. Пользователь может самостоятельно задать уровень приоритета для каждого типа правил.

Формат

```
classtype: <имя класса>;
```

Классификация правил определена в файле **classification.config** с использованием простого синтаксиса:

```
config classification:  <имя класса>,<описание класса>,<приоритет по умолчанию>
```

Стандартная классификация правил Snort показана в таблице 2.

Таблица 2: Классификация правил Snort

Тип (Classtype)	*Описание*	*Приоритет*
attempted-admin	Попытка получения привилегий администратора	Высокий
attempted-user	Попытка получения привилегий пользователя	Высокий
shellcode-detect	Обнаружен исполняемый код	Высокий
successful-admin	Получены права администратора	Высокий
successful-user	Получены права пользователя	Высокий
trojan-activity	Обнаружена сетевая троянская программа	Высокий
unsuccessful-user	Неудачная попытка получения привилегий пользователя	Высокий
web-application-attack	Атака на Web-приложение	Высокий
attempted-dos	Предпринята атака на службы (DoS)	Средний
attempted-recon	Попытка несанкционированной передачи информации (утечка)	Средний
bad-unknown	Непонятный трафик, который может оказаться опасным	Средний
denial-of-service	Обнаружена атака на службы (DoS)	Средний
misc-attack	Прочие атаки	Средний
non-standard-protocol	Зафиксировано использование нестандартного протокола	Средний
rpc-portmap-decode	Обнаружен запрос RPC[1]	Средний
successful-dos	Успешная атака на службы (DoS)	Средний
successful-recon-largescale	Крупномасштабная утечка информации	Средний
successful-recon-limited	Утечка информации	Средний
suspicious-filename-detect	Обнаружено подозрительное имя файла	Средний
suspicious-login	Попытка входа в систему с использованием подозрительного имени	Средний
system-call-detect	Обнаружен вызов системной функции	Средний
unusual-client-port-connection	Клиент использует необычный порт	Средний
web-application-activity	Доступ к потенциально опасному web-приложению	Средний
icmp-event	Обычный пакет ICMP	Низкий
misc-activity	Прочие действия	Низкий
network-scan	Обнаружено сканирование сети	Низкий
not-suspicious	Трафик не является подозрительным	Низкий
protocol-command-decode	Обнаружена обычная команда протокола	Низкий
string-detect	Обнаружена подозрительная строка	Низкий
unknown	Непонятный трафик	Низкий

Ниже показан пример использования классификации.

```
alert tcp any any -> any 80 (msg: "EXPLOIT ntpdx overflow"; \
      dsize: >128; classtype:attempted-admin; proirity:10;)
```

[1]Remote Procedire Call – удаленный вызов процедуры.

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

Предупреждение
classtype использует классификации, определенные конфигурационной опцией classification. Классификации, используемые для правил из дистрибутива Snort, определены в файле etc/classification.config

priority

Тег priority использует для присваивания правилам уровня приоритета. Опция classtype присваивает правилу принятый по умолчанию уровень приоритета, который можно изменить с помощью priority. Пример совместного использования опций classtype и priority был приведен в описании опции classtype, а пример независимого задания приоритета показан ниже.

```
alert tcp any any -> any 80 (msg: "WEB-MISC phf attempt"; flags:A+; \
      content: "/cgi-bin/phf"; priority:10;)
```

Формат

```
priority: <целое число>;
```

Опции проверки содержимого пакетов (payload)

content

Ключевое слово content обеспечивает поддержку одной из важнейших функций Snort – проверку содержимого пакетов. Эта опция позволяет пользователю создавать правила для поиска в пакетах определенной информации и выполнения тех или иных действий при ее обнаружении. Для проверки содержимого пакетов используется функция поиска по шаблону Boyer-Moore. Если заданная последовательность данных обнаружена в поле содержимого пакета, проверка считается успешной и выполняется остальная часть правила. Следует помнить, что при поиске учитывается регистр символов.

Аргумент опции может содержать как текст, так и двоичные данные (обычно они указываются между парой символов | и задаются последовательностью шестнадцатеричных представлений байтов). Ниже показан пример задания строки поиска, содержащей текст и бинарные данные.

```
alert tcp any any -> any 139 (content:"|5c00|P00|I|00|E|00 5c|";)
```

Отметим, что в одном правиле может присутствовать более одной опции content, что позволяет снижать уровень ложных срабатываний за счет более точного задания искомых последовательностей.

Если перед опций помещен знак отрицания (`!`), правилу будут соответствовать пакеты, не содержащие указанных данных. Такая возможность полезна для генерации сигналов в случае обнаружения пакетов, не содержащих заданной последовательности.

Формат

```
content: [!] "<строка поиска>";
```

Пример задания смешанной (текст и бинарные данные) был приведен выше, а здесь дается пример поиска текстовой строки.

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

Изменение параметров поиска

Ключевое слово content поддерживает множество опций-модификаторов, которые изменяют поведение системы поиска. Список модификаторов приведен ниже:

68. **depth** (размер области поиска)
69. **offset** (смещение начало поиска от начала поля данных)
70. **distance** (количество пропускаемых байтов после первого найденного соответствия)
71. **within** (размер области поиска после первого найденного соответствия)
72. **nocase** (без учета регистра символов)
73. **rawbytes** (поиск в необработанных данных)

nocase

Ключевое слово **nocase** позволяет задать программе Snort, что следует осуществлять поиск, заданный предыдущей опцией **content** без учета регистра символов.

Формат

```
nocase;
```

Пример поиска без учета регистра символов показан ниже.

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

rawbytes

Ключевое слово rawbytes позволяет искать в пакете необработанные (raw) данные, игнорирую декодирование, выполняемое препроцессорами. Ключевое слово изменяет поиск данных, указанных предыдущей опцией content.

Формат

```
rawbytes;
```

Приведенный пример показывает правило поиска необработанных данных вместо результатов декодирования препроцессором telnet.

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

depth

Ключевое слово **depth** показывает размер блока данных из пакета, в котором осуществляется поиск, заданных предыдущей опцией **content**. Например, при задании глубины 5 программ Snort будет просматривать в поисках заданной последовательности только первые 5 байтов поля данных в пакете.

Ключевое слово **depth** меняет режим поиска для опции **keyword**, указанной перед **depth**.

Формат

```
depth: <целое число>;
```

Пример использования опции **depth** показан ниже.

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

offset

Ключевое слово **offset** позволяет задать смещение в поле данных пакет, с которого начинается поиск последовательности, заданной предыдущей опцией **content**. Например, offset 5 будет говорить программе Snort о необходимости начинать поиск, пропустив первые 5 байтов поля данных.

Ключевое слово **offset** меняет режим поиска для опции **keyword**, указанной перед **offset**.

Пример использования опций content, offset и depth был показан выше. В этом примере программа будет искать текст / **cgi-bin**/**phf** в 20 байтах пакета HTTP после пропуска первых 5 байтов поля данных.

Формат

```
offset: <целое число>;
```

distance

Ключевое слово distance показывает программе Snort сколько байтов нужно пропустить после найденной предыдущей опцией content строки поиска для начало поиска последовательности, заданной другой опцией **content**. Например, правило

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

позволяет находить в поле данных пакета строки вида ABC?DEF (знак вопроса означает любой символ)

Формат

```
distance: <число байтов>;
```

within

Ключевое слово **within** размер области поиска для опции **content** от конца подстроки, найденной предыдущей опцией **content**. Например, правило

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

ограничивает поиск подстроки DEF 10 байтами после найденной в поле данных подстроки ABC.

Формат

```
within: <число байтов>;
```

uricontent

Опция uricontent служит для поиска в нормализованных полях запросов URI. Это означает, что при создании правил, включающих нормализуемые данные (например, %2f или обход каталогов - directory traversal), эти правила не следует использовать.

Например, URI

```
/scripts/..%c0%af../winnt/system32/cmd.exe?/c+ver\end{verbatim}
```

будет нормализоваться в

```
\begin{verbatim}/winnt/system32/cmd.exe?/c+ver
```

Нормализованная форма URI

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

будет иметь вид

```
\begin{verbatim}/cgi-bin/phf?
```

При создании правил **uricontent** укажите содержимое, которое вы хотите найти в контексте нормализованного URI. Например, если Snort нормализует обход каталогов (directory traversal), не включайте directory traversal.

Вы можете создавать правила поиска в ненормализованном содержимом пакетов с помощью опции content (см. выше).

Эта опция использует тот же набор модификаторов, который применяется для описанной выше опции content.

Эта опция работает совместно с препроцессором HTTP Inspect.

Формат

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

isdataat

Эта опция позволяет проверить наличие данных в заданном участке пакета (возможно относительно завершения подстроки, найденной с помощью опции **content**).

Формат

```
isdataat:<int>[,relative];
```

Например, правило

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

будет обеспечивать поиск в поле данных пакета подстроки PASS и после ее обнаружения будет проверять наличие в пакете по крайней мере еще 50 байтов, после чего проверит отсутствие символа новой строки в 50 байтах после PASS.

pcre

Ключевое слово **pcre** позволяет создавать правила, содержащие регулярные выражения, совместимые с языком perl. Более детальную информацию об использовании таких регулярных выражений вы найдете на сайте http://www.pcre.org.

Формат

```
pcre:[!]"(/<regex>/|m<delim><regex><delim>)[ismxAEGRUB]";
```

Модификаторы в конце правила устанавливают флаги для регулярного выражения.

Таблица 3: Модификаторы, совместимые с Perl

i	Регистр символов не принимается во внимание
s	Включать символы новой строки в dot metacharacter
m	По умолчанию строка трактуется как одна большая последовательность символов. С помощью специальных символов ^ и $ можно задать проверку соответствия для начала или конца строки. При наличии модификатора m символы ^ и $ задают поиск соответствия в начале или в конце каждой новой строки (относительно символа перевода строки в буфере), а также в начале и в конце буфера.
x	Символы пробелов в шаблоне поиска игнорируются за исключением случаев использования перед таким символом escape-символа или включения пробела в символьный класс (character class)

Таблица 4: Модификаторы, совместимые с PCRE

A	Наличие заданной подстроки проверяется только в начале буфера (аналогично ^)
E	Задает для $ поиск соответствия только в самом конце строки. Без модификатора E символ $ будет задавать также поиск перед символом новой строки (если таковой имеется) в конце буфера.
G	Инвертирует трактовку параметров количества повторов (quantifier) так, что если по умолчанию они не являются “жадными” (greedy - число повторов может быть любым, вплоть до максимального) установка знака вопроса (?) вслед за параметром, меняет “состояние жадности”.

Таблица 5: Модификаторы Snort

R	Задает поиск соответствия относительно конца предыдущего найденного соответствия (аналогично опции distance:0;)
U	Задает поиск в декодированном буфере URI (аналогично uricontent)
B	Отключает использование декодированного буфера (аналогично rawbytes)

Модификаторы R и B не следует использовать совместно.

Приведенный ниже пример задает нечувствительный к регистру символов поиск в поле данных подстроки BLAH.

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

byte_test

Эта опция позволяет сравнить байт с заданным значением. Опция может использоваться применительно к двоичным значениям или их символьному представлению.

Формат

```
byte_test: <bytes to convert>, [!]<operator>, <value>, <offset>  \
    [,relative] [,<endian>] [,<number type>, string];
```

Таблица 6: Параметры опции byte_test

Параметр	Описание
`bytes_to_convert`	Число байтов, которые могут извлекаться из пакета
`operator`	Операция, выполняемая для сравнения байта с заданным значением: ● < - меньше ● > - больше ● = - равно ● ! - не совпадает ● & - побитовая операция И (AND) ● | - побитовая операция ИЛИ (OR)
`value`	Значение, с которым выполняется сравнение
`offset`	Смещение в поле данных пакета, с которого начинается операция сравнения
`relative`	Задает отсчет смещения от конца предыдущего найденного соответствия
`endian`	Задает порядок следования: ● `big` - big endian (старший разряд слева, используется по умолчанию) ● `little` - little endian (старший разряд справа)
`string`	Указывает, что данные в пакете представлены в символьном формате
`number type`	Задает тип считываемых значений: ● `hex` – шестнадцатеричное число ● `dec` - десятичное число ● `oct` - восьмеричное число

Любой из операторов можно использовать со знаком инверсии (!). При использовании знака ! без оператора в качестве последнего принимается оператор равенства (=).

Ниже показано несколько примеров использования опции byte_test.

```
alter 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;)

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

alert udp any any -> any 1234 \
      byte_test: 4, =, 1234, 0, string, dec; \
      msg: "got 1234!";)

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

alert udp any any -> any 1236 \
      byte_test: 2, =, 12, 0, string, dec; \
      msg: "got 12!";)

alert udp any any -> any 1237 \
      byte_test: 10, =, 1234567890, 0, string, dec; \
      msg: "got 1234567890!";)

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

byte_jump

Опция **byte_jump** позволяет создавать простые правила считывания данных из пакетов с пропускам некоторого количества байтов, задаваемого значением поля в пакете. С помощью этой опции считывается размер части пакета, которую следует пропустить и считываются данные, расположенные после этой части.

Опция **byte_jump** сначала определяет размер пропускаемой области данных, преобразуя считанную из пакета информацию в целое число и потом пропускает соответствующее число байтов, устанавливая указатель, который будет использоваться для следующего считывания информации из пакета. Этот указатель называют **detect offset end pointer** или **doe_ptr**.

Формат

```
byte_jump: <bytes_to_convert>, <offset> \
        [,relative] [,multiplier <multiplier value>] [,big] [,little][,string]\
        [,hex] [,dec] [,oct] [,align] [,from_beginning];
```

Таблица 7: Параметры опции byte_jump

Параметр	Описание
bytes_to_convert	Число байтов, считываемых из пакета.
offset	Смещение в поле данных пакета, с которого начинается обработка.
relative	Задает использование смещения относительно конца предыдущего найденного соответствия.
multiplier <value>	Умножает количество вычисленных байтов на значение параметра **<value>** и пропускает полученное количество байтов.
big	Обрабатывает данные как big endian (старший разряд сначала – используется по умолчанию).
little	Обрабатывает данные как little endian (сначала младший разряд).
string	Данные в пакете представлены в виде текстовой строки.
hex	Преобразовать строку данных в шестнадцатеричное значение.
dec	Преобразовать строку данных в десятичное значение.
oct	Преобразовать строку данных в восьмеричное значение.
align	Округляет число конвертируемых байтов по следующей 32-битовой границе.
from_beginning	Задает отсчет пропускаемых байтов от начала поля данных пакета, а не от текущей позиции в пакете.

Ниже приведен пример использования опции **byte_jump**.

```
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";)
```

regex

Взамен **regex** следует использовать опцию pcre (см. 7).

content-list

Опция **content-list** устарела и ее не следует использовать.

Опции детектирования для заголовков пакетов

fragoffset

Опция **fragoffset** позволяет сравнивать смещение фрагмента дейтаграммы IP с заданным десятичным значением. Для отсечения всех первых фрагментов можно использовать ключевое слово **fragbits** и просмотр опции More fragments при установке **fragoffset: 0** (см. пример ниже).

Формат

```
fragoffset:[<|>]<целое число>
```

Ниже приведен пример правила с использованием опции **fragoffset**

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

ttl

Ключевое слово **ttl** используется для проверки времени жизни дейтаграмм IP. Эта опция может быть полезна при детектировании попыток трассировки с помощью traceroute.

Формат

```
ttl:[[<целое число>-]><=]<целое число>;
```

Приведенный ниже пример проверяет, что значение поля TTL в заголовках пакетов меньше 3.

```
ttl:<3;
```

Второй пример позволяет детектировать пакеты со значением TTL от 3 до 5.

```
ttl:3-5;
```

tos

Эта опция позволяет проверять в пакетах поле IP TOS (тип обслуживания).

Формат

```
tos:[!]<целое число>;
```

В приведенном ниже примере проверяется отличие значения поля TOS от 4

```
tos:!4;
```

id

Ключевое слово **id** используется для проверки наличия в поле IP ID заданного значения. Некоторые программы (эксплойты, сканеры, старые программы) устанавливают в этом поле определенное значение (например, число 31337 весьма популярно среди хакеров).

Формат

```
id:<целое число>;
```

В приведенном примере проверяется наличие в поле IP ID значения 31337.

```
id:31337;
```

ipopts

Ключевое слово **ipopts** позволяет проверять наличие в заголовке IP указанных опций. Поддерживается проверка следующих опций IP:

rr
- Record route (запись маршрута)

eol
- End of list (завершение списка опций)

nop
- No op (нет опции)

ts
- Time Stamp (временная метка)

sec
- IP security option (опция безопасности)

lsrr
- Loose source routing (нежестко заданный отправителем маршрут)

ssrr
- Strict source routing (жестко заданный отправителем маршрут)

satid
- Stream identifier (идентификатор потока)[2]

any
- any IP options are set (любые опции)

Чаще всего проверяются опции ssrr и lsrr, которые не используются в распространенных приложениях Internet.

Формат

```
ipopts:<rr|eol|nop|ts|sec|lsrr|ssrr|satid|any>;
```

В приведенном примере проверяется наличие в заголовке IP опции Loose Source Routing.

```
ipopts:lsrr;
```

Отметим, что в правиле недопустимо наличие нескольких ключевых слов **ipopts**.

fragbits

Ключевое слово **fragbits** используется для проверки наличия в заголовке IP битов фрагментации и резервного бита. Опция поддерживает следующие параметры:

M
- More Fragments (проверять бит MF)

D
- Don't Fragment (проверять бит запрета фрагментации)

R
- Reserved Bit (проверять резервный бит)

[2]Эта опция устарела и не должна использоваться.

Для изменения характера проверки могут использоваться перечисленные ниже модификаторы:

+
- соответствует, если установлены указанные биты

-
- соответствует, если установлен любой из указанных битов

!
- обращение (соответствует, если не установлен ни один из указанных битов)

Формат

```
fragbits:[+-*]<[MDR]>
```

В приведенном ниже примере проверяется установка флагов More Fragments и Do not Fragment.

```
fragbits:MD+;
```

dsize

Ключевое слово **dsize** используется для проверки размера поля данных пакета. Данная опция позволяет детектировать пакеты аномальных размеров, которые достаточно часто применяются для переполнения буферов.

Формат

```
dsize: [<>]<целое число>[<><целое число>];
```

Приведенный ниже фрагмент правила позволяет детектировать пакеты размером от 300 до 400 байтов.

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

Отметим, что условие **dsize** не будет выполняться для пакетов перестроения потока (stream rebuilt packet), независимо от их размера.

flags

Ключевое слово **flags** используется для проверки наличия заданных флагов TCP. Список проверяемых флагов приведен ниже:

F
- FIN (младший бит поля флагов TCP)

S
- SYN

R
- RST

P
- PSH

A
- ACK

U
- URG

1
- резервный бит 1 (старший бит байта TCP Flags)

2
- резервный бит 2

0
- отсутствие флагов TCP

Перечисленные ниже модификаторы позволяют менять поведение опции:

+
- соответствует, если установлены указанные биты

*
- соответствует, если установлен любой из указанных битов

!
- обращение (соответствует, если не установлен ни один из указанных битов)

Для создания правил обработки пакетов инициирования сессий (например, пакеты ECN, где установлен флаг SYN и резервные биты 1 и 2), можно задавать маски опций. Маска отделяется от проверяемых флагов запятой. Например, для детектирования SYN-пакетов независимо от значений резервных битов можно задать маску S,12.

Формат

```
flags:[!|*|+]<FSRPAU120>[,<FSRPAU120>];
```

Для детектирования пакетов с флагами SYN и FIN независимо от значений резервных битов 1 и 2 может использоваться правило

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

flow

Опция **flow** используется вместе со сборкой потоков TCP и позволяет применять правило лишь к некоторым направлениям потока трафика. Вы можете в результате создавать правила, которые будут относиться только к клиентам или только к серверам, что дает возможность легко дифференцировать пакеты, относящиеся к клиентам из $HOME_NET, просматривающим web-страницы, от пакетов, относящихся к серверам, расположенным в $HOME_NET.

Ключевое слово **established** будет заменять опцию **flags: A+**, часто используемую применительно к уже организованным соединениям TCP.

Формат

```
flow: [(established|stateless)]
      [,(to_client|to_server|from_client|from_server)]
      [,(no_stream|only_stream)]
```

Таблица 8: Параметры опции flow

Параметр	Описание
to_client	Переключается на серверные отклики от A к B
to_server	Переключается на клиентские запросы от A к B
from_client	Переключается на клиентские запросы от A к B
from_server	Переключается на серверные отклики от A к B
established	Переключается только на организованные соединения TCP
stateless	Переключается независимо от состояния обработчика потока (stream processor) и может быть полезно для детектирования пакетов, направленных на аварийное завершение работы системы
no_stream	Не переключается пакетами перестроения потока (полезно для опций **dsize** и **stream4**)
only_stream	Переключается только пакетами перестроения потока

Ниже приведен пример использования опции **flow** для обнаружения потенциально опасного трафика.

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

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

flowbits

Опция **flowbits** используется совместно со средствами отслеживания соединений препроцессора Flow. Это позволяет создавать правила для сеансов транспортного уровня. Опция **flowbits** наиболее полезна для сеансов TCP.

Для опции flowbits поддерживается 7 ключевых слов. Большинство параметров требует указания определенного пользователем имени специфического состояния, которое будет проверяться. При создании таких имен следует ограничиваться буквами латиницы, цифрами, а также символами точки, дефиса и подчеркивания.

Формат

```
flowbits: [set|unset|toggle|isset,reset,noalert][,<STATE_NAME>];
```

Таблица 9: Параметры опции flowbits

Параметр	Описание
set	Устанавливает (определяет) указанное состояние для текущего потока данных.
unset	Отменяет указанное состояние для текущего потока данных.
toggle	Устанавливает указанное состояние, если оно еще не установлено, и отменяет установленное ранее.
isset	Проверяет, установлено ли указанное состояние.
isnotset	Проверяет, что указанное состояние не установлено.
noalert	Отключает для правила генерацию сигнала независимо от остальных опций детектирования.

```
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;)
```

seq

Опция **seq** служит для проверки значения порядковых номеров TCP.

Формат

```
seq:<целое число>;
```

Приведенный ниже пример проверяет равенство порядкового номера TCP нулю.

```
seq:0;
```

ack

Ключевое слово **ack** используется для проверки номеров подтверждений TCP.

Формат

```
ack: <целое число>;
```

Приведенный пример проверяет равенство нулю порядкового номера подтверждения TCP.

```
ack:0;
```

3.6.13 window

Ключевое слово **window** используется для проверки размера окна TCP.

Формат

```
window:[!]<целое число>;
```

В приведенном примере проверяется совпадение размера окна TCP со значением 55808.

```
window:55808;
```

itype

Ключевое слово **itype** используется для проверки типа сообщения ICMP.

Формат

```
itype:[<|>]<целое число>[<><целое число>];
```

Приведенный пример позволяет детектировать сообщения ICMP с идентификатором типа, превышающим 30.

```
itype:>30;
```

icode

Ключевое слово **itype** используется для проверки значения кода ICMP.

Формат

```
icode: [<|>]<целое число>[<><целое число>];
```

В приведенном примере детектируются сообщения ICMP с кодом более 30.

```
code:>30;
```

icmp_id

Ключевое слово **icmp_id** служит для проверки значений идентификаторов ICMP. Такая проверка может оказаться полезной для обнаружения некоторых программ организации скрытых каналов, которые используют для передачи информации статические поля ICMP. Подключаемый модуль был создан, в частности, для детектирования DdoS-агентов stacheldraht.

Формат

```
icmp_id:<целое число>;
```

В приведенном примере проверяется наличие нулевого значения в поле ICMP ID.

```
icmp_id:0;
```

icmp_seq

Ключевое слово **icmp_seq** используется для проверки порядковых номеров ICMP. Такая проверка может оказаться полезной для обнаружения некоторых программ организации скрытых каналов, которые используют для передачи информации статические поля ICMP. Подключаемый модуль был создан, в частности, для детектирования DdoS-агентов stacheldraht.

Формат

```
icmp_seq: <целое число>;
```

В приведенном примере детектируются сообщения ICMP с порядковым номером 0.

```
icmp_seq:0;
```

rpc

Ключевое слово **rpc** используется для проверки приложений RPC, номеров версий и процедур в запросах SUNRPC CALL.

Для номера версии и процедуры допускается использование шаблона **0**, которому соответствуют любые значения номеров.

Формат

```
rpc: <номер приложения>, [<номер версии>|*], [<номер процедуры>|*];
```

Ниже показан пример детектирования запросов an RPC portmap GETPORT.

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

В силу особенностей машины поиска соответствий детектирование по ключевому слову **rpc** работает несколько медленней, чем поиск значений RPC с использованием ключевого слова content (см. стр. 5).

ip_proto

Ключевое слово **ip_proto** позволяет проверять идентификатор протокола в заголовке IP. Список протоколов вы можете найти в файле /etc/protocols.

Формат

```
ip_proto:[!><] <имя или номер протокола>;
```

Приведенный пример позволяет детектировать трафик IGMP.

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

sameip

Ключевое слово **sameip** позволяет детектировать пакеты с совпадающими адресами IP для получателя и отправителя.

Формат

```
sameip;
```

Приведенный пример обеспечивает генерацию сигнала при совпадении IP-адресов получателя и отправителя.

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

Опции пост-детектирования

logto

Опция **logto** говорит программе Snort о необходимости записи всех пакетов, которые соответствуют правилу в специальный файл. Следует отметить, что эта опция не будет работать, когда программа Snort находится в режиме ведения бинарного журнала (binary logging mode).

Формат

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

session

Ключевое слово **session** позволяет получать пользовательскую информацию из сеансов TCP. Во многих случаях данные, вводимые пользователем в сеансах telnet, rlogin, ftp и т. п., могут представлять значительный интерес.

Опция может использоваться с двумя аргументами - printable (выводить только печатаемые символы) и all (выводить все). Во втором случае непечатаемые символы выводятся в виде шестнадцатеричных кодов.

Формат

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

Приведенный пример обеспечит вывод в журнальный файл всех печатаемых символов из пакетов telnet.

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

Использование опции **session** может существенно замедлять работу Snort, поэтому пользоваться данной опцией следует с осторожностью. Эта опция очень удобна для обработки сохраненных файлов (формат pcap).

resp

Ключевое слово **resp** используется для попытки закрыть сессию при генерации сигнала. В программе Snort такая реакция на события называется flexible response (гибкий отклик).

Механизмы завершения сеансов для гибких откликов перечислены в таблице 10.

Перечисленные в таблице опции можно комбинировать для передачи множества откликов одному хосту.

Таблица 10: Параметры опции resp

Параметр	Описание
rst_snd	Отправлять пакеты TCP-RST передающему сокету
rst_rcv	Отправлять пакеты TCP-RST принимающему сокету
rst_all	Отправлять пакеты TCP_RST в обоих направлениях
icmp_net	Передавать пакеты ICMP_NET_UNREACH отправителю
icmp_host	Передавать пакеты ICMP_HOST_UNREACH отправителю
icmp_port	Передавать пакеты ICMP_PORT_UNREACH отправителю
icmp_all	Передавать все перечисленные выше пакеты пакеты ICMP отправителю

Формат

```
resp: <resp_mechanism>[,<resp_mechanism>[,<resp_mechanism>]];
```

Поддержка опции **resp** не включена по умолчанию и для ее использования нужно задать флаг **-enable-flexresp** при конфигурировании Snort перед компиляцией.

Пользоваться этой опцией следует с осторожностью, поскольку можно достаточно легко создать бесконечный цикли типа приведенного ниже:

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

Кроме того, с помощью этой опции можно создать препятствия для легитимного трафика.

Приведенное ниже правило будет пытаться сбрасывать попытки соединений TCP с портом 1524.

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

react

Опция **react** является частью механизмов гибкого реагирования (Flex Resp) для трафика, соответствующего правилу Snort. Основным назначением этой опции является блокирование нежелательных сайтов (например, napster или порно-сайтов). Код Flex Resp позволяет Snort активно закрывать соединения и/или передавать в пользовательскую программу соответствующее сообщение. Для опции поддерживается два основных модификатора:

- **block** – закрыть соединение и передать пользователю видимое уведомление;
- **warn** – передать пользователю видимое предупреждение (будет доступно вскоре).

Кроме основных модификаторов опция может использоваться с дополнительными параметрами:

- **msg** – включить заданный текст в передаваемое пользователю сообщение;
- **proxy**: **<номер порта>** - использовать порт proxy для передачи пользователю видимого предупреждения (будет доступно вскоре).

Дополнительные аргументы разделяются запятыми. Ключевое слово **react** должно использоваться последним в списке опций правила.

Формат

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

Ниже приведен пример блокирования нежелательных сайтов с генерацией для пользователя соответствующего сообщения.

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

Поддержка опции **react** не включена по умолчанию и для ее использования нужно задать флаг **-enable-flexresp** при конфигурировании Snort перед компиляцией.

При использовании опции **react** следует соблюдать осторожность, чтобы не возникло петель.

tag

Ключевое слово **tag** позволяет записывать в журнальные файлы не только пакет, который вызвал срабатывание правила. После срабатывания правила весь последующий трафик для данной пары "отправитель-получатель" будет помечаться, а отмеченный трафик можно протоколировать для последующего анализа. Кроме того, для помеченного трафика генерируется дополнительный сигнал, который позволяет подключаемому модулю вывода выполнить соответствующие действия. Отметим, что модуль вывода в базы данных пока не поддерживает дополнительных сигналов для помеченного трафика.

Формат

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

type

- **session** – записывать пакеты сессии, для которой сработало правило;
- **host** – записывать пакеты с хоста, который вызвал срабатывание правила (с учетом направления);

count

- подсчитывать с использованием единиц, заданных параметром **<metric>**.

metric

- **packets** – пометить **<count>** пакетов для хоста/сессии;
- **seconds** – помечать пакеты для хоста/сессии в течение **<count>** секунд.

Отметим, что сами пакеты, для которых сработало правило, помечаться не будут. Например, может казаться, что приведенное ниже правило будет помечать в течение 600 все пакеты, включающие адрес 10.1.1.1.

```
alert tcp any any <> any 10.1.1.1 (tag:host,600,seconds,src;)
```

Однако такие пакеты не будут помечаться совсем, поскольку для каждого из них правило будет срабатывать. В данном случае будет полезна опция **flowbits**.

```
alert tcp any any <> any 10.1.1.1 (flowbits:isnotset,tagged; \
    flowbits:set,tagged; tag:host,600,seconds,src;)
```

приведенный ниже пример обеспечит запись пакетов в течение первых 10 секунд любого сеанса telnet.

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

Создание правил

При создании правил Snort следует пользоваться приведенными ниже рекомендациями.

Проверка содержимого

Вторая версия машины детектирования Snort на первой фазе поиска выполняет проверку соответствия заданным шаблонам (pattern). Чем длиннее искомая последовательность, тем точнее будет результат поиска. Правила без опции **content** (или **uricontent**) замедляют работу системы в целом.

Хотя некоторые опции (например, **pcre** и **byte_test**) проверяют соответствие для полей данных пакета, эти опции не используют машину setwise pattern matching (“горизонтальный” поиск). Поэтому следует по возможности использовать вместо таких опций правило **content**.

Бороться с причиной, а не проявлением

Старайтесь писать правила так, чтобы они противодействовали использованию уязвимостей, а не конкретным эксплойтам.

Например, разумно искать уязвимые команды со слишком длинными аргументами, нежели shell-коды.

Правила для предотвращения использования уязвимостей менее уязвимы с точки зрения их обхода путем простого изменения кода эксплойта.

Учитывайте в правилах случаи нетипичного использования протоколов

Многие службы (например, FTP) обычно передают команды с использованием символов верхнего регистра. В случае FTP для передачи имени пользователя клиент будет посылать строку:

```
user username_here
```

Простое правило для детектирования попыток подключения к серверу FTP пользователя root может иметь вид:

```
alert tcp any any -> any any 21 (content:"user root";)
```

Такая задача может показаться тривиальной, однако хорошо написанное правило будет учитывать не только очевидное написание, но и множество (или все) возможные варианты. Например, для упомянутого случая строка может иметь вид:

```
user root
user     root
user<tab>root
```

Для того, чтобы предусмотреть в правиле все варианты строки, которые могут быть восприняты сервером, правило придется усложнить. Хороший пример реализации такого правила показан ниже:

```
alert tcp any any -> any 21 (flow:to_server,established; content:"root";
   pcre:"/user\s+root/i";)
```

Для этого правила следует отметить несколько важных моментов:

- правило включает опцию **flow**, проверяющую, что трафик относится к существующему соединению;
- правило включает опцию **content**, которая используется для поиска слова **root**, являющегося самым длинным в данном случае идентификатором атаки; эта опция использует машину поиска setwise pattern match и ускоряет работу Snort;
- правило включает опцию **pcre**, которая ищет слово **user**, сопровождаемое по крайней мере одним пробельным символом (включая символ табуляции), без учета регистра символов в слове **root**.

Оптимизация правил

Система поиска по содержимому (content matching) в некоторых случаях использует рекурсию, поэтому неаккуратно написанные правила могут приводить к тому, что программа Snort будет тратить ненужное время на дублирование проверок.

Рекурсия работает следующим образом – если заданный шаблон найден, но любое из условий проверки, расположенных после него, не выполняется, начинается новый поиск по шаблону. Повторение происходит до тех пор, пока соответствие не будет найдено снова или не закончится список опций.

На первый взгляд такой подход может показаться неразумным, однако он обусловлен необходимостью. Рассмотрим для примера правило:

```
alert ip any any -> any any (content:"a"; content:"b"; within:1;)
```

Программа будет искать в пакетах символ “a”, непосредственно за которым следует символ “b”. Без использования рекурсии строка “aab” даст отрицательный результат, хотя она и содержит символ “a” за которым непосредственно следует b. Отказ обусловлен тем, что за первым символом “a” не следует сразу же “b”.

Рекурсия важна для детектирования, но реализации механизма рекурсии недостаточно эффективна. Ниже приведен пример неоптимизированного правила:

```
content:"|13|"; dsize:1;
```

На первый взгляд может показаться, что это правило просто будет искать в пакете байт 0x13. Однако, по причине использования рекурсии для пакета, содержащего 1024 байта 0x13, будет выполняться 1023 попытки поиска этого байта и 1023 проверки dsize. Почему? Код 0x13 будет быть найден в первом байте, проверка **dsize** даст отрицательный результат и по причине использования рекурсии поиск кода 0x13 будет продолжен с того места, где было обнаружено предыдущее вхождение этого кода, с проверкой значения **dsize**, пока код 0x13 не будет найден снова.

Смена порядка опций, при которой опция с дискретной проверкой (в данном случае dsize) будет перемещена вперед, приведет к значительному ускорению работы правила Snort. Оптимизированное правило будет иметь вид:

```
dsize:1; content:"|13|";
```

Для пакета, содержащего 1024 байта 0x13 отрицательный результат будет получен незамедлительно, поскольку сначала проверяется дискретная опция dsize.

Перечисленные ниже опции являются дискретными и их следует размещать в начале правил:

- dsize
- flags
- flow
- fragbits
- icmp_id
- icmp_seq
- icode
- id
- ipopts
- ip_proto
- itype
- seq
- session
- tos
- ttl
- ack
- window
- resp
- sameip

Проверка числовых значений

Опции **byte_test** и **byte_jump** предназначены для создания правил для протоколов, использующих данные указанного в пакете размера (прежде всего, для протокола RPC).

Для того, чтобы понять смысл использования **byte_test** и **byte_jump** рассмотрим попытку использования эксплойта для сервиса sadmind. Ниже показан дамп пакета для этого эксплойта:

```
89 09 9c e2 00 00 00 00 00 00 00 02 00 01 87 88  ................
00 00 00 0a 00 00 00 01 00 00 00 01 00 00 00 20  ................
40 28 3a 10 00 00 00 0a 4d 45 54 41 53 50 4c 4f  @(:.....metasplo
49 54 00 00 00 00 00 00 00 00 00 00 00 00 00 00  it..............
00 00 00 00 00 00 00 00 40 28 3a 14 00 07 45 df  ........@(:...e.
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 06 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 04 00 00 00 00 00 00 00 04  ................
7f 00 00 01 00 01 87 88 00 00 00 0a 00 00 00 04  ................
7f 00 00 01 00 01 87 88 00 00 00 0a 00 00 00 11  ................
00 00 00 1e 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 3b 4d 45 54 41 53 50 4c 4f  .......;metasplo
49 54 00 00 00 00 00 00 00 00 00 00 00 00 00 00  it..............
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 06 73 79 73 74 65 6d 00 00  ........system..
00 00 00 15 2e 2e 2f 2e 2e 2f 2e 2e 2f 2e 2e 2f  ....../../../../
2e 2e 2f 62 69 6e 2f 73 68 00 00 00 00 00 04 1e  ../bin/sh.......
<snip>
```

Рассмотрим более подробно поля пакета и способ обнаружения эксплойта. Отметим сначала некоторую специфику RPC:

- целые числа записываются в формате uint32s (4 байта); например, число 26 будет иметь вид 0x0000001a.
- строка представляется значением uint32, задающим размер, собственно строкой и нулевыми байтами (от 0 до 3), служащими для выравнивания по 4-байтовой границе; строка “bob” будет иметь вид 0x00000003626f6200.

```
89 09 9c e2      - уникальный идентификатор запроса (случайное число uint32)
00 00 00 00      - тип rpc (call = 0, response = 1)
00 00 00 02      - версия rpc (2)
00 01 87 88      - программа rpc (0x00018788 = 100232 = sadmind)
00 00 00 0a      - версия программы rpc (0x0000000a = 10)
00 00 00 01      - процедура rpc (0x00000001 = 1)
00 00 00 01      - разновидность “мандата” (1 = auth\_unix)
00 00 00 20      - размер данных auth\_unix (0x20 = 32)
## следующие 32 байта содержат данные auth\_unix
40 28 3a 10 - unix timestamp (0x40283a10 = 1076378128 = feb 10 01:55:28 2004 gmt)
00 00 00 0a – длина имени клиентской машины (0x0a = 10)
4d 45 54 41 53 50 4c 4f 49 54 00 00  - строка “metasploit”
```

```
00 00 00 00 - uid запрашивающего пользователя (0)
00 00 00 00 - gid запрашивающего пользователя (0)
00 00 00 00 – идентификаторы дополнительных групп (0)

00 00 00 00     - разновидность верификатора (0 = auth\_null – нет верификатора)
00 00 00 00     - размер верификатора (0, нет верификатора)
```

Остальная часть пакета содержит запрос, передаваемый процедуре 1 программы sadmind.

Известно, что уязвимость sadmind заключается в доверии к значениям uid, приходящим от клиента. Программа sadmind выполняет все запросы, полученные от клиента с uid = 0 (root).

Мы уже имеем данные, достаточные для создания правила. Сначала убедимся, что пакет содержит вызов RPC.

```
content:"|00 00 00 00|"; offset:4; depth:4;
```

После этого проверим, что пакет адресован программе sadmind.

```
content:"|00 01 87 88|"; offset:12; depth:4;
```

Далее нужно удостовериться, что пакет является вызовом уязвимой процедуры 1.

```
content:"|00 00 00 01|"; offset:16; depth:4;
```

Также нужно проверить, что пакет содержит “мандат” auth_unix.

```
content:"|00 00 00 01|"; offset:20; depth:4;
```

Нам не нужно имя хоста, но нужно проверить числовое значение, расположенное вслед за этим именем. В этом случае будет полезна опция byte_test. Начиная с поля размера имени хоста мы имеем последовательность байтов:

```
00 00 00 0a 4d 45 54 41 53 50 4c 4f 49 54 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00
```

Нам нужно прочитать первые 4 байта, преобразовать их в число и пропустить соответствующее этому числу количество байтов, а также байты заполнения. Сделав это, мы получим последовательность байтов:

```
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00
```

которая начинается с интересующего нас значения uid.

Словами это можно выразить так – мы хотим прочесть 4 байта, расположенных со смещением 36 от начала пакета, преобразовать эти байты в целое число и пропустить соответствующее количество байтов плюс байты заполнения. Соответствующее правило Snort будет иметь вид:

```
byte_jump:4,36,align;
```

после этого мы проверяем равенство нулю значения uid

```
content:"|00 00 00 00|"; within:4;
```

теперь соберем вместе созданные компоненты правила.

```
content:"|00 00 00 00|"; offset:4; depth:4;
content:"g00 01 87 88|"; offset:12; depth:4;
content:"|00 00 00 01|"; offset:16; depth:4;
content:"|00 00 00 01|"; offset:20; depth:4;
byte_jump:4,36,align;
content:"|00 00 00 00|"; within:4;
```

Третья и четвертая строки обеспечивают проверку содержимого последовательных байтов и их можно объединить:

```
content:"|00 00 00 00|"; offset:4; depth:4;
content:"|00 01 87 88|"; offset:12; depth:4;
content:"|00 00 00 01 00 00 00 01|"; offset:16; depth:8;
byte_jump:4,36,align;
content:"|00 00 00 00|"; within:4;
```

если служба sadmind уязвима к переполнению буфера, вместо определения длины имени хоста и пропуска соответствующего числа байтов, следует проверить не слишком ли велик размер имени хоста. Для этого мы считаем 4 байта, начиная с байта 36, преобразуем их в целое число и проверим, что это значение не превышает разумное (скажем, 200). Правило Snort будет иметь вид:

```
byte_test:4,>,200,36;
```

И результирующее правило получает форму:

```
content:"|00 00 00 00|"; offset:4; depth:4;
content:"|00 01 87 88|"; offset:12; depth:4;
content:"|00 00 00 01 00 00 00 01|"; offset:16; depth:8;
byte_test:4,>,200,36;
```

Библиография

RFC1518: Y. Rekhter, T. Li., An Architecture for IP Address Allocation with CIDR, 1993