КУРС МОЛОДОГО БОЙЦА CISCO (v 1.1)

Сергей Фёдоров, CCIE Security #22974

Вступление: данный материал получился в результате компиляции публикаций в блоге

http://Habrahabr.ru/blogs/cisconetworks, где удалось немного обкатать и «причесать» отдельные куски. Надеюсь, этот труд будет небесполезен для настройщиков оборудования cisco, особенно для начинающих, т.е. для вас, молодые бойцы! Не взыщите за язык – это не официальная «сухая» книжка, а попытка объяснить сложное «на пальцах», используя устоявшийся сленг и реальные примеры.

Материал состоит из нескольких частей:

1. Искусство владения консолью. Там описываются некоторые самые нужные и полезные навыки
2. Тонкости настройки маршрутизаторов и коммутаторов. Ими которыми пользуются опытные настройщики
3. Защищаемся маршрутизатором. Защита проходящего трафика
   - 3.1 Списки доступа
   - 3.2 Межсетевое экранирование
   - 3.3 Система предотвращения вторжений (IPS)
   - 3.4 Перехватывающая аутентификация (cut-through proxy)
   - 3.5 NBAR
   - 3.6 QoS
4. Защищаем маршрутизатор.

Искусство владения консолью.

Многие начинающие настройщики сетевого железа боятся консоли (CLI, Command Line Interface) как огня. Ещё бы: ни тебе всплывающих подсказок по наведенному мышиному курсору, ни тебе красивых картинок, а лишь непонятные буквы на черном (зеленом, белом) фоне. Боязно…

Однако консоль является мощнейшим инструментом, не овладев которым называть себя настройщиком cisco нельзя.

При помощи консоли можно:

1. Задать начальную конфигурацию.
2. Восстановить потерянные пароли (для разных железок по-разному. Но поиском на сайте cisco.com легко найти процесс по ключевым словам “password recovery <модель железки>”)
3. Настроить нестандартные топологии
4. Включить скрытые возможности
5. Проверить правильность настроек командами `show`
6. Отладить процесс командами `debug`

Помните: при помощи консоли можно всё, при помощи GUI – не всё, а только то, что запрограммировали и разрешили разработчики.

Как же овладеть искусством быстрой настройки через консоль?

Безусловно, требуется много тренироваться :) Однако есть несколько хитростей, которые облегчат работу и могут пригодиться в реальной жизни для быстрой локализации и решения проблем или сдачи экзамена CCIE :)

Хитрость 1: Запомните горячие сочетания клавиш. Самые часто используемые ctrl+a (начало строки), ctrl+e (конец строки), ctrl+z (выход из режима настройки)

Хитрость 2: На маршрутизаторах и коммутаторах пользуйтесь командой **`do`** для вызова команд `show` из режима настройки.

Пример: `(config)# do show ip route`

Это сильно экономит время, т.к. не приходится выходить из режима настройки и возвращаться обратно в него. Т.к. консольные команды древовидные, то беготня по режимам реально отнимает время и силы

Хитрость 3: Используйте выборку из конфига или вывода других команд просмотра. Для этого есть служебные команды после символа “|” (“grep”)

Примеры: `sh run | include ip route`

```
sh ip route | include 192.168.0.0
sh run | begin router ospf 1
```

Хитрость 4: Используйте блокнот или другой текстовый редактор, чтобы создавать шаблон конфига, а потом вливайте его копи-пастом в консоль. Перевод каретки циской распознаётся как ввод команды. Не забывайте после последней команды нажать кнопку ввод :)

Хитрость 5: Используйте стрелки «вверх» и «вниз» (или горячие клавиши ctrl+p, ctrl+n) для вызова ранее введенных команд. Размер буфера команд настраивается.

Хитрость 6: Не забывайте, что разные команды возможно вводить из разных режимов: из режима непривилегированного пользователя можно мало и только посмотреть, из привилегированного – посмотреть, включить `debug`, из режима настройки – настроить параметры или перейти в

подрежимы. Не забывайте, что знак «?» введенный без пробела, подскажет возможные продолжения команды, а знак «?», введенный через пробел, – возможные дальнейшие ключевые слова

Хитрость 7: Учите команды и активно их сокращайте! Помните, что если сокращение однозначно, его циска распознает.

Пример (сравните):

```
ip nat inside source list NAT interface GigabitEthernet0/0 overload
```

```
ip nat in so l NAT int G0/0 o
```

Для ускорения обучению используйте кнопочку табуляции: если начало команды уже однозначно, циска продолжит команду автоматически.

Хитрость 8: Строчки из конфига (running или startup) являются командами. Можно подглядеть в конфиг и стереть неугодную команду, поставив перед ней ключевое слово *no*

Тонкости настроек маршрутизаторов и коммутаторов.

Будем считать, что вы уже активно осваиваете хитрости настройки через консоль. Пришло время рассказать ещё несколько тонкостей. О чём имеет смысл подумать при настройке маршрутизаторов и коммутаторов cisco.

Тонкость 1. Аккуратность.

Часто возникает задача что-нибудь добавить в текущую конфигурацию. Наверняка вы знаете, что многие элементы отдельно пишутся, и отдельно применяются (на интерфейс, ко всей железке и т.д.). Будьте крайне осторожны, изменяя настройки таких технологий, как PBR (route-map), QoS (policy-map), IPSec (crypto map), NAT. Наиболее корректно будет сначала снять правила с использования, потом изменить, потом повесить снова. Связано это с тем, что все изменения вы вносите сразу же в состояние железки. Иногда то, что уже работает (например, подгружено в оперативку) конфликтует с новым конфигом. Не редки ситуации, когда железка уходит в перезагрузку после попытки изменения конфига.

Пример: пусть у нас есть route-map, примененный на интерфейс. Пусть нам надо его изменить. Наиболее "чистый" способ такой:

1. Копируем из конфига существующий route-map

2. Вставляем его в блокнот.

3. Меняем ему имя в блокноте

4. Меняем сам route-map

    * Обязательно проверьте его на логику: будет ли он делать то, что нужно

5. Копируем его из блокнота в буфер обмена

6. Вставляем в конфиг.

* Он ещё никуда не применен, т.к. с другим именем, поэтому процедура безопасна

7. Применяем новый route-map на интерфейс

К сожалению, такую красивую процедуру делать обычно лень :)

Если политика безопасности требует, чтобы в конфиге все названия были каноническими, то можно дополнительно сделать так:

3А. Копируем route-map с новым именем, но старым содержанием

3Б. Вставляем его обратно в конфиг маршрутизатора.

3В. Временно применяем route-map с новым именем вместо канонического. Далее все действия (с 4 по 7) производим с каноническим route-map

Если нет возможности снять, например, crypto map (в этом случае упадут все туннели), а добавить новый абзац надо, то необходимо минимизировать время, в течение которого будет "недоделанный конфиг" (incomplete crypto map). Это происходит, если в абзаце не хватает одной или двух строчек из следующих

```
match address
set transform-set
set peer
```

Для минимизации времени простоя вбейте необходимый кусок конфига crypto-map в блокноте и потом копи-пастом влейте его на циску.

2. Тонкость. Подстраховка.

Помните, что пока вы не набрали

```
copy running-config startup-config
```

или, что тоже самое,

```
wr (write)
```

файл начальной конфигурации ещё старый. Поэтому, если всё поломалось, достаточно перезагрузить железяку. Дабы не бегать самому или не звонить на другой конец (офиса/города/планеты), есть технология "отложенной перезагрузки"

```
reload in <промежуток времени>
```

и перезагрузки в конкретное время

```
reload at <дата и время>
```

3. Тонкость. Сохранение конфигов

Помните, что файл конфигурации - это просто текстовый файл. Можно скидывать файл настройки на tftp, ftp, http, flash и т.д. А можно просто забрать его к себе на рабочую машину. Например, включив в PuTTy (или другом терминальном клиенте) лог выводимого на экран, можно банально дать команду

```
sh run
```

и получить свой конфиг в файл лога.

*Хинт: `sh run` выводит конфиг поэкранно, разделяя полосочками. В файле эти полосочки тоже будут. Если лень эти полосочки потом убирать, можно дать команду

```
sh run | tee  http://1.1.1.1
```

Команда `tee` копирует на экран то, что пытается отправить по http на 1.1.1.1. Отправить ей не удастся, но на экран вывалится весь конфиг без задержек и разделителей.

4. Тонкость. Имена.

Не секрет, что проще написать конфиг самому, чем разобраться в чужом :) Да и свои старые конфиги иногда трудно читать. Поэтому, чтобы помочь себе и другим рекомендую

- Все названия (ACL, route-map, crypto-map, transform-set и т.д.) писать большими буквами (ну или хотя бы с большой буквы). Приведу для примера любимую строчку из конфига ASA (в ней для внешнего NAT на внешнем интерфейсе применяется ACL с названием outside)

```
nat (outside) 10 access-list outside outside
```

Если назвать ACL большими буквами, то легко выделить, где ключевые слова, а где названия.

```
nat (outside) 10 access-list OUTSIDE outside
```

- СРАЗУ УДАЛЯТЬ НЕНУЖНОЕ! Потренировались, побились лбом, поэкспериментировали - мусор удалить! Уже через неделю трудно вспомнить, нужные это конструкции или нет.

- Если ACL всё-таки используются нумерованные (исторически, например), их можно изменять, используя синтаксис расширенных ACL

```
ip access-list ex 101

no 10

25 permit tcp 1.1.1.1 any eq 25
```

Защищаемся маршрутизатором

Не претендуя на полноту изложения, попробую описать технологии, которыми можно воспользоваться для защиты периметра, а также опишу вкратце, как защитить сам маршрутизатор от нападок извне и изнутри.

Рассматривать будем IOS с **firewall feature set**. Этот набор возможностей, как правило, есть во всех IOSах (в которых есть шифрование), кроме самого базового.

Итак, пусть на границе нашей сети стоит машрутизатор cisco, который и призван обеспечивать безопасность наших внутренних ресурсов. Понятно, что он и сам подвержен нападкам, поэтому кроме защиты пользовательского трафика надо защитить и саму железяку.

Защищаем трафик. Списки доступа.

Первым делом имеет смысл порезать ненужный трафик самым простым и грубым способом - списками доступа.

Списки доступа (ACL, Access Control List) для протокола IP - на маршрутизаторах cisco бывают стандартные (проверяют только ip адрес источника и разрешают или запрещают прохождение трафика по этому параметру) и расширенные (проверяют адреса источника и назначения, протокол передачи, порты источника и назначения, а также другие поля заголовков IP, TCP, UDP и других протоколов).

Порядок строк в списке доступа очень важен, т.к. проверяются эти строки по порядку и как только будет найдено совпадение, пакет или будет пропущен (если в правиле написано permit) или будет уничтожен (если deny)

В конце любого ACL непременно стоит невидимое "запретить все", поэтому мимо ACL пакет не проскользнёт.

Списки доступа бывают нумерованные и поименованные. Рекомендую использовать поименованные со смысловыми названиями.

Примеры:

```
access-list 1 permit 192.168.1.0 0.0.0.255
access-list 101 permit ip any 192.168.1.0 0.0.0.255
ip access-list standard TEST
  permit host 1.2.3.4
ip access-list extended TEST2
  permit tcp any 10.1.1.0 0.0.0.255 eq http
  permit tcp any 10.1.1.0 0.0.0.255 eq https
  permit udp any 10.1.1.0 0.0.0.255 eq 53
  permit ip any 10.1.1.0 0.0.0.255 dscp cs5
```

Списки доступа - это шаблоны, которые можно использовать как для фильтрации трафика, так и как критерий отбора для других технологий. Например, списками доступа определяется "интересный" трафик для шифрования, для NAT, для QoS и т.д.

Сам по себе список доступа ничего не делает, пока не будет применен для какой-либо технологии. Например, для фильтрации трафика на интерфейсе на вход или на выход ACL применяется командой (помните, что на одном интерфейсе в одном направлении может быть применен только один ip ACL)

```
ip access-group <название ACL> {in|out}
```

Традиционно рекомендуется на внешний интерфейс вешать так называемый антиспуфинговый ACL, т.е. предотвращающий атаки с подделанных адресов.

Пример:

```
ip access-list ex ANTISPOOFING
  deny ip host 0.0.0.0 any
  deny ip 10.0.0.0 0.255.255.255 any
  deny ip 172.16.0.0 0.15.255.255 any
  deny ip 192.168.0.0 0.0.255.255 any
  deny ip host 255.255.255.255 any
  deny ip 224.0.0.0 15.255.255.255 any
  permit ip any any
```

Важно: с таким ACL надо быть очень осторожным в случае, если вы работаете с шифрованными туннелями IPSec. Дело в том, что ACL, висящий на вход интерфейса, проверяет сначала заголовок зашифрованного пакета, а потом - заголовок расшифрованного. Поэтому запрет трафика от частных сетей (10, 172, 192) может нарушить работу туннеля.

Межсетевое экранирование.

Итак, порезали ненужный трафик. Пришло время заняться межсетевым экранированием. Надо обеспечить внутренних пользователей Интернетом, но при этом не пропустить снаружи внутрь несанкционированные подключения. Маршруизаторы cisco умеют быть межсетевыми экранами с сохранением сессий (stateful firewall).

Если у вас задачи простые, нет выделенных зон безопасности, нет анонса сервисов наружу, то проще всего воспользоваться базовым межсетевым экраном.

Для этого надо создать правило `ip inspect`, описать те протоколы, которые вы хотите обрабатывать и запоминать их сессии, привесить это правило на интерфейс и ... всё :) Маршрутизатор будет запоминать те сессии, которые были инициированы изнутри, и пропускать снаружи только те пакеты, которые "заказаны". Если же пришедший пакет не соответствует никакой сессии, то дальше маршрутизатор проверяет ACL, висящий на интерфейсе, на предмет наличия разрешающего правила для этого пакета. Т.е. сначала обрабатывается динамический кэш сессий, и только потом – список доступа, если динамической записи не обнаружено.

Пример конфига:

```
Ro(config)# ip inspect name FW tcp

Ro(config)# ip inspect name FW udp

Ro(config)# ip inspect name FW icmp

Ro(config)# ip inspect name FW ftp

Ro(config)# ip inspect name FW sip
```

TCP - слушает TCP сессии.

UDP - слушает UDP сессии.

остальные строки включают прослушивание и обработку соответствующего протокола, т.к. его работа сложнее, чем просто пропуск ответного пакета TCP/UDP сессии. Например, протокол FTP имеет один служебный канал, по которому идёт согласование и аутентификация, а данные передаются совсем по другому каналу, причём сессия пытается инициироваться снаружи и маршрутизатор её не пропустит. А если включить инспектирование, маршрутизатор подслушает служебную сессию, узнает, на каких портах сервер и клиент договорились слать данные и эту сессию тоже поместит в список разрешенных (т.е. в кэш сессий).

```
Ro(config)# int f0/1 (внутренний интерфейс)

Ro(config-if)# ip inspect FW in
```

Правило вешается на вход внутреннего или выход внешнего интерфейса, т.е. в направлении на ВЫХОД трафика наружу.

На внешнем интерфейсе висит строгий ACL, который почти ничего не пропускает снаружи, например

```
Ro(config)# ip access-l ex STRICT

Ro(config-ex-nacl)# deny ip any any


Ro(config)# int f0/0 (внешний интерфейс)

Ro(config-if)# ip access-g STRICT in
```

Без такого списка доступа маршрутизатор, добрая душа, будет все пропускать снаружи, хотя сессии будут создаваться и сохраняться. Но пакеты, не соответствующие сессиям, будут просто пропущены интерфейсом, т.к. нечему их задержать.

В приведенном же варианте снаружи внутрь пройдут только те пакеты, которые были запрошены изнутри. Строгий список доступа блокирует все внешние пакеты.

Есть тонкость: ACL STRICT попутно запретит весь трафик на сам роутер, т.к. по умолчанию трафик роутера не попадает в инспектируемый. Чтобы инспектировать трафик маршрутизатора надо добавить

```
Ro(config)# ip inspect name FW router
```

Если же задачи сложные, например, надо создавать различные зоны безопасности (демилитаризованные зоны, DMZ) и гибко настраивать работу протоколов между этих зон, то лучше воспользоваться так называемым зональным межсетевым экраном (zone-based firewall). Описывать его здесь не буду, ибо это уже не для молодого бойца :)

Чем ещё можно защитить трафик, проходящий через маршрутизатор?

Системой предотвращения вторжений (IPS), промежуточной аутентификацией (cut-through proxy), оценкой протоколов (технология ip nbar), созданием очередей (QoS).

Система предотвращения вторжений (Intrusion Prevention System, IPS).

Вообще линейка продуктов по системе предотвращения вторжений у компании cisco довольно широкая. Туда входят отдельно стоящие сенсоры IPS серии 42XX, модуль в 6500 - IDSM2, модуль в ASA - AIP-SSM, модуль в маршрутизатор (ISR) - NME-IPS, "карточка" в ISR - AIM-IPS. Ту же идеологию циска старается привнести и в софтовые решения на базе ISR, добавляя в IOS соответствующий функционал.

Вся идеология обнаружения и предотвращения вторжений основана на понятии сигнатуры. Сигнатура по сути шаблон "неправильности" в одном пакете или потоке пакетов. "Неправильности" бывают разные, начиная от типичных методов разведки и заканчивая сетевыми червями. Эти шаблоны старательно пишутся программистами циски и доходят до пользователя в виде обновлений. Т.е. система реактивна по своей сути и основана на постоянных обновлениях, что стоит денег. Лицензии на обновления привязываются к каждой железке непосредственно. Без лицензии можно менять ОС, но нельзя накатить обновления сигнатур.

Немного истории систем обнаружения и предотвращения вторжений на базе маршрутизаторов.

Первая система IDS (Intrusion Detection System) была внедрена на маршрутизаторах с IOS 12.2.8T с firewall feature set. Тогда это были 26XX и 36XX маршрутизаторы. Система представляла собой несколько десятков (максимум 105) сигнатур. Их можно было только отключить или задать работу не для всего трафика. Эта система включалась командами

```
ip audit name IDS attack action {alarm, drop, reset}

ip audit name IDS info action {alarm, drop, reset}

int f0/0

  ip audit IDS {in|out}
```

Это была вещь в себе. Ни гибкой настройки, ни обновлений, ни понятия о том, что же там в сигнатурах.

Следующий шаг был сделан с внедрением отдельного файла signature definition. Этот специальный файл загружался на маршрутизатор, на него указывали в настройке, в нем хранились все сигнатуры и их параметры. Настраивалась эта конструкция так:

```
ip ips sdf location flash:{256MB.sdf|128MB.sdf|attack-drop.sdf}
```

файл подбирается исходя из количества оперативной памяти на маршрутизаторе. Самый большой файл - 256MB.sdf - содержал более 1500 сигнатур и требовал минимум 256 мегабайт оперативной памяти

```
ip ips name IPS

int f0/0

  ip ips IPS {in|out}
```

После привешивания на интерфейс правила IPS циска подгружала в память сигнатуры из файла и давала возможность их настроить как через консоль, так и через web-GUI (кстати, GUI, называемый Security Device Manager, SDM, весьма удобен при настройке IPS)

Для обратной совместимости в IOSах (до 12.4.T(11)) оставались и встроенные сигнатуры. При использовании внешнего файла их рекомендовалось отключать

```
no ip ips sdf builtin
```

Можно было потребовать, чтобы трафик блокировался в случае, если невозможно подгрузить файл sdf или произошёл сбой подсистемы IPS

```
ip ips fail close
```

Но формат сигнатур тут был такой же, как и в сенсорах IPS версии 4. Этот формат не позволял глубже анализировать трафик и отсекать новые хитрые атаки. На самих IPS сенсорах уже появился к тому времени новый формат 5, в котором можно настраивать накопительные параметры риска атаки (Risk Rating), создавать зоны более пристального внимания (Target Value Rating) и многое другое.

Поэтому с версии 12.4.T(11) старый формат более не поддерживается, обновления сигнатур формата 4 прекратились в августе 2008.

Чтобы перейти на новый формат и гибко защитить сеть при помощи системы IPS надо теперь грузить другой файл

```
IOS-S###-CLI.pkg
```

в котором хранятся зашифрованные актуальные сигнатуры и их параметры. Номер ### постоянно увеличивается, обновления надо постоянно подгружать. К слову, это можно сделать автоматически командой

```
ip ips auto-update
```

Далее, надо установить на маршрутизатор открытый ключ компании cisco для расшифровывания (а вернее, проверки цифровой подписи) выкачанного файла

Делаем так:

```
crypto key pubkey-chain rsa

  named-key realm-cisco.pub signature
```

```
    key-string
30820122 300D0609 2A864886 F70D0101 01050003 82010F00 3082010A 02820101
00C19E93 A8AF124A D6CC7A24 5097A975 206BE3A2 06FBA13F 6F12CB5B 4E441F16
17E630D5 C02AC252 912BE27F 37FDD9C8 11FC7AF7 DCDD81D9 43CDABC3 6007D128
B199ABCB D34ED0F9 085FADC1 359C189E F30AF10A C0EFB624 7E0764BF 3E53053E
5B2146A9 D7A5EDE3 0298AF03 DED7A5B8 9479039D 20F30663 9AC64B93 C0112A35
FE3F0C87 89BCB7BB 994AE74C FA9E481D F65875D6 85EAF974 6D9CC8E3 F0B08B85
50437722 FFBE85B9 5E4189FF CC189CB9 69C46F9C A84DFBA5 7A0AF99E AD768C36
006CF498 079F88F8 A3B3FB1F 9FB7B3CB 5539E1D1 9693CCBB 551F78D2 892356AE
2F56D826 8918EF3C 80CA4F4D 87BFCA3B BFF668E9 689782A5 CF31CB6E B4B094D3
F3020301 0001
quit
```

Эти команды можно просто вколотить в режим

```
Ro(config)#
```

копипастом. Ключ один для всех.

Желательно создать во флеше маршрутизатора отдельную папку для файлов IPS

```
Ro# mkdir flash:/IPS
```

Туда надо скопировать файл IOS-S###-CLI.pkg , а также указать, что в ней будут храниться нужные для работы файлы

```
Ro(config)# ip ips config location flash:/IPS/
```

Теперь осталось эти самые нужные файлы туда установить. Делается это хитрой командой

```
Ro# copy flash:/IPS/IOS-S###-CLI.pkg idconf
```

Эта процедура займет существенное время (несколько минут) и в результаты вы увидите во флэше

```
21           0 May 27 2009 14:22:58 +04:00 IPS
22     8662169 May 27 2009 14:24:22 +04:00 IPS/IOS-S399-CLI.pkg
23      284871 May 28 2009 22:48:00 +04:00 IPS/ccmt-2811-sigdef-default.xml
24         255 May 27 2009 16:35:56 +04:00 IPS/ccmt-2811-sigdef-delta.xml
25       34761 May 28 2009 22:43:44 +04:00 IPS/ccmt-2811-sigdef-category.xml
26         304 May 27 2009 16:35:56 +04:00 IPS/ccmt-2811-seap-delta.xml
27        8509 May 28 2009 22:43:40 +04:00 IPS/ccmt-2811-sigdef-typedef.xml
28         491 May 27 2009 17:05:00 +04:00 IPS/ccmt-2811-seap-typedef.xml
```

Эти xml файлы содержат дефолтные настройки, ваши изменения, параметры блокировки и т.д.

Практически всё. Надо только создать правило и повесить его на интерфейс, как это делалось и раньше:

```
ip ips name IPS

int f0/0

  ip ips IPS {in|out}
```

После этого произойдёт подгрузка в память сигнатур, и те из них, которые по умолчанию включены, сразу примутся за работу.

Помните, что сигнатур много, они кушают много памяти и процессора, поэтому циска настоятельно рекомендует сделать следующее.

1. Отключить категорию сигнатур all

```
Ro(config)# ip ips signature-category

Ro(config-ips-category)# category all

Ro(config-ips-category-action)# retired true
```

2. Включить для начала категорию, рассчитанную на IOS, причём в базовом исполнении

```
Ro(config)# ip ips signature-category

Ro(config-ips-category)# category ios_ips basic

Ro(config-ips-category-action)# retired false

Ro(config-ips-category-action)# enabled true
```

Конфиг актуализируется после выхода обратно в режим (config)#

3. Далее, следя за памятью и загрузкой процессора, можно добавлять и другие категории сигнатур. Настройка самих сигнатур возможна как через консоль из режима

```
ip ips signature-definition
```

так и через SDM или более новый WEB-GUI - CCE (Cisco Configuration Expert)

Параметры и механизм настройки сигнатур максимально приближен к настройке на сенсорах, поэтому если вы имеете опыт настройки AIP-SSM, сенсоров 42XX или IDMS2, можете смело браться за дело. Если такого опыт нет, лучше почитать о настройке сигнатур. Или сходить на курс IPS 6.0 :)

Перехватывающая аутентификация (cut-through proxy)

Задача этой технологии – проверять имя и пароль пользователя перед тем, как выпустить его наружу или пустить внутрь периметра. Это часть общей идеологии IBNS (Identity Based Network System), где определяющим является имя пользователя и именно по имени можно сопоставлять

настройки конкретного клиента, например, список доступа, в котором описано, что можно данному клиенту.

Для проверки пользователей можно использовать внешние базы данных, доступные по протоколам TACACS (технология cisco, TCP/49) , RADIUS (стандартная технология, UDP/1645 или UDP/1812 для аутентификации, UDP/1646 или UDP/1813 для сбора статистики) или Kerberos 5.

Для того, чтобы заставить маршрутизатор спрашивать внешние базы пользователей, необходимо включить новую технологию AAA (Authentication, Authorization, Accounting)

```
aaa new-model
```

Эта команда коварна, т.к. включает правила аутентификации по умолчанию, а эти правила гласят «Использовать локальную базу данных пользователей». Если же локальных пользователей нет, то вы попадаете в ловушку (lockout), из которой выход только один – процедура восстановления пароля (password-recovery)

Далее, надо создать правило аутентификации, указывающее на внешний сервер и описать сам этот сервер: по какому протоколу с ним связываться, на какой адрес и с каким ключом

```
radius-server host <ip> key <ключ>
```

Или

```
tacacs-server host <ip> key <ключ>

aaa authentication login <имя> group {radius|tacacs}
```

Надо настроить правило перехватывающей аутентификации, указав по какому протоколу происходит первичное обращение клиента (http, telnet, ftp) и какой трафик инициирует ответ клиенту от циски (описывается списком доступа, где строчки permit означают «выдать клиенту окошко запроса на логин/пароль»)

```
ip auth-proxy name <имя> {http|ftp|telnet} [list <ACL>]
```

По умолчанию – любой трафик по выбранному протоколу.

Теперь осталось описать, каким правилом аутентификации пользоваться при доступе по http и привесить правило аутентификации на интерфейс (правило всегда вешается на вход интерфейса)

```
Ro(config)# ip http authentication aaa [login-authentication <имя>]

Ro(config)# int f0/0 (внутренний интерфейс, откуда приходят пакеты от клиента)

Ro(Config-if)# ip auth-proxy <имя>
```

В старых IOS не было возможности явно указать правило аутентификации, используемое для аутентификации по http, т.е. всегда использовалось дефолтное правило и надо было его обязательно поменять для использования сервера TACACS или RADIUS

```
aaa authentication login default group {radius|tacacs}
```

Теперь же есть возможность явно указать не только правило для login по http, но и отдельно правило для командной авторизации и входа в режим exec через http (SDM,CCE)

Дополнительно, чтобы изначально пользователю разрешить чуть-чуть, и только после аутентификации явно разрешить ему то, что ему положено, надо на тот же интерфейс повесить довольно строгий список доступа, который бы разрешал только то, что можно всем и всегда, например, начальные пакеты для аутентификации и, например, какие то общие локальные ресурсы

```
ip access-list ext ZLOY
  permit tcp any host 1.2.3.4 eq 80
  permit tcp any host 172.16.1.100 eq 135
int f0/0
  ip access-group ZLOY in
```

Т.к. входящий список доступа на интерфейсе всегда отрабатывает первым, то работать это будет так:

1. Приходит пакет. Мы его проверяем списком доступа, разрешен ли

2. Если разрешен, то дальше проверяем, пришёл ли пакет от аутентифицированного ip адреса источника.

3. Если нет, то проверяем, по какому протоколу пришёл пакет.

4. Если по протоколу, указанному в правиле ip auth-proxy, то выдаём в броузере, ftp или telnet приложении клиента окошко на ввод логин/пароля, в противном случае просто уничтожаем пакет

Разберём пример:

Пусть есть маршрутизатор с внутренним интерфейсом f0/0 и внешним s0/0. Мы хотим, чтобы трафик, направленный во все сети, кроме сети 172.16.1.0/24, сначала должен быть аутентифицирован по протоколу http.

```
ip access-l ext AUTHACL
  deny ip any 172.16.1.0 0.0.0.255
  permit ip any any
username admin pass cisco (на всякий случай, чтобы случайно себя не заблокировать)
aaa new-model
radius-server host 1.1.1.1 key cisco
aaa authentication login AUTH group radius
ip http server (включаем на циске http сервер)
ip http authentication aaa login-authentication AUTH
ip auth-proxy name OUT http list AUTHACL
ip access-l ex ZLOY
  permit tcp any h 1.2.3.4 eq 80
```

```
int f0/0
  ip access-group ZLOY in
  ip auth-proxy OUT
```

Если вас не устраивает небезопасный протокол http, то можно использовать безопасный протокол https. Для этого надо выработать ключевую пару RSA и включить https сервер (самоподписанный сертификат циска выпишет сама)

```
Router(config)# hostname Ro
```
(лучше задать не дефолтный hostname, т.к. в ряде случаев маршрутизатор ругается на него)

```
Ro(config)# ip domain name mydomain.com
Ro(config)#crypto key generate rsa [modulus-size <размер>] [label <метка>]
Ro(config)# do wr
Ro(config)# ip http secure-server
```

После этого соединения по https тоже будут перехватываться циской на предмет проверить логин/пароль.

Надо отметить, что технология довольно громоздкая, при работе по https существенно тормозит первое подключение, если не сохранить самоподписанный сертификат циски, как доверенный, и не слишком гибкая. Однако позволяет проверять пользователей и их права.

Защищаемся маршрутизатором. NBAR.

На многих маршрутизаторах даже в базовом IOS есть довольно удобная и наглядная цискина технология: Network-Based Application Recognition (NBAR). При помощи неё маршрутизатор может распознать различные протоколы и приложения и при необходимости использовать эти знания для реализации качества обслуживания (QoS)

Каким же образом маршрутизатор может выделить из трафика различные протоколы?

1. Анализируя заголовок TCP/UDP и сортируя по известным номерам. Это самое простое и доступное во всех IOSах

2. Анализируя заголовок протокола уровня 7. Например, разбирая заголовок HTTP и вычленяя оттуда тип передаваемых данных (например, сам протокол HTTP, citrix и некоторые другие Web-enabled приложения), URL, вложения.

3. Анализируя служебный протокол сложных (многоканальных) приложений и учитывая динамические, согласованные по служебному каналу, сессии.

4. Анализируя заголовки RTP.

5. Анализируя заголовок протоколов, отличных от TCP/UDP

Третий и четвертый алгоритм требует включения инспектирования протоколов (ip inspect), рассмотренные чуть ранее. Инспектирование требует IOS с firewall feature set.

Включить технологию NBAR можно двумя способами:

1. Просто включить исследование всех известных протоколов.

```
Ro(config)# int f0/0
Ro(config-if)# ip nbar protocol-discovery
```

После включения исследования на интерфейсе статистика по известным протоколам будет собираться и её можно посмотреть командой

```
Ro# show ip nbar protocol-discovery
```

2. Воспользоваться архитектурой MQC (Modular QoS CLI)

Создать класс трафика, указав необходимый протокол

```
Ro(config)# class-map <CLASSNAME>
Ro(config-cmap)# match protocol <protocol>
```

Например:

```
Ro(config)# class-map PD
Ro(config-cmap)# match protocol citrix isa-tag 2
```

Создать политику и указать для класса действие

```
Ro(config)# policy-map <POLICY>
Ro(config-pmap)# class <CLASSNAME>
Ro(config-pmap-c)# (action)
```

Например:

```
Ro(config)# policy-map POL
Ro(config-pmap)# class PD
Ro(config-pmap-c)# {shape|police|priority|service-policy}
```

Осталось привесить политику на интерфейс, чтобы её активировать

```
Ro(config)# int f0/0
Ro(config-if)# service-policy {input|output} <POLICY>
```

Например, повесим политику на вход интерфейса f0/0

```
Ro(config)# int f0/0
Ro(config-if)# service-policy input POL
```

Указание в классе трафика ключевых слов match protocol означает включение технологии NBAR.

Посмотреть статистику попаданий в класс можно посмотреть командой

```
Ro# sh policy-map interface f0/0
```

Осталось только напомнить, что любой процесс анализа трафика производится процессором и изрядно его напрягает. По ссылке - статья с тестом производительности без применения NBAR, с применением базовых и расширенных технологий анализа.

http://www.cisco.com/en/US/technologies/tk543/tk759/technologies_white_paper0900aecd8031b712_ps6616_Products_White_Paper.html

Результаты обработки трафика технологией NBAR можно собирать по SNMP и анализировать. С точки зрения защиты периметра самое пристальное внимание надо уделять резкому возрастанию количества трафика по какому-нибудь протоколу. Это может свидетельствовать как о наличии в сети червей, так и о других проблемах, таких как ботнеты, спам-прокси и т.д.

Защищаемся маршрутизатором: QoS

QoS - тема большая. Прежде чем рассказывать про тонкости настроек и различные подходы в применении правил обработки трафика, имеет смысл напомнить, что такое вообще QoS.

Quality of Service (QoS) - технология предоставления различным классам трафика различных приоритетов в обслуживании.

Во-первых, легко понять, что любая приоритизация (здесь и далее это выдуманное слово означает «расстановка приоритетов») имеет смысл только в том случае, когда возникает очередь на обслуживание. Именно там, в очереди, можно "проскользнуть" первым, используя своё право.

Очередь же образуется там, где узко (обычно такие места называются "бутылочным горлышком", bottle-neck). Типичное "горлышко" - выход в Интернет офиса, где компьютеры подключенные к сети как минимум на скорости 100 Мбит/сек, все используют канал к провайдеру, который редко превышает 100 МБит/сек, а часто составляет мизерные 1-2-10МБит/сек. На всех.

Во-вторых, QoS не панацея: если "горлышко" уж слишком узкое, то часто переполняется физический буфер интерфейса, куда помещаются все пакеты, собирающиеся выйти через этот интерфейс. И тогда новопришедшие пакеты будут уничтожены, даже если они сверхнужные. Поэтому, если очередь на интерфейсе в среднем превышает 20% от максимального своего размера (на маршрутизаторах cisco максимальный размер очереди составляет, как правило, 128-256 пакетов), есть повод крепко задуматься над дизайном своей сети, проложить дополнительные маршруты или расширить полосу до провайдера.

Разберемся с составными элементами технологии

Маркировка. В полях заголовков различных сетевых протоколов (Ethernet, IP, ATM, MPLS и др.) присутствуют специальные поля, выделенные для маркирования трафика. Маркировать же трафик нужно для последующей более простой обработки в очередях.

Ethernet. Поле Class of Service (CoS) - 3 бита. Позволяет разделить трафик на 8 потоков с различной маркировкой

IP. Есть 2 стандарта: старый и новый. В старом было поле ToS(8 бит), из которого в свою очередь выделялись 3 бита под названием IP Precedence. Это поле копировалось в поле CoS ethernet заголовка.

Позднее был определен новый стандарт. Поле ToS было переименовано в DiffServ, в нем выделялось 6 бит для поля Differencial Service Code Point (DSCP), в котором можно передавать требуемые для данного типа трафика параметры.

Маркировать данные лучше всего ближе к источнику этих данных. По этой причине большинство IP-телефонов самостоятельно добавляют в IP-заголовок голосовых пакетов поле DSCP = EF или CS5. Многие приложения также маркируют трафик самостоятельно в надежде, что их пакеты будут обработаны приоритетно.

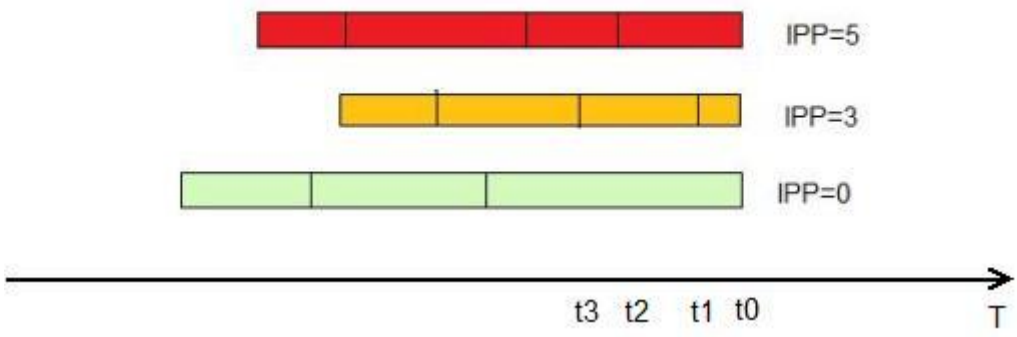
Очереди.

Даже если мы не используем никаких технологий приоритизации, это не значит, что не возникает очередей. В узком месте очередь возникнет в любом случае и будет предоставлять стандартный механизм FIFO (First In First Out). Такая очередь, очевидно, позволит не уничтожать пакеты сразу, сохраняя их до отправки в буфере, но никаких преференций, скажем, голосовому трафику не предоставит.

Если хочется предоставить некоторому выделенному классу абсолютный приоритет (т.е. пакеты из этого класса всегда будут обрабатываться первыми), то такая технология называется priority queuing. Все пакеты, находящиеся в физическом исходящем буфере интерфейса, будут разделены на 2 логические очереди, и пакеты из привилегированной очереди будут отсылаться, пока она не опустеет. Только после этого начнут передаваться пакеты из второй очереди. Эта технология простая, довольно грубая, её можно считать устаревшей, т.к. обработка неприоритетного трафика будет постоянно останавливаться. На маршрутизаторах cisco можно создать 4 очереди с разными приоритетами. В них соблюдается строгая иерархия: пакеты из менее привилегированных очередей не будут обслуживаться до тех пор, пока не опустеют все очереди с более высоким приоритетом.

Справедливая очередь (Fair Queuing). Технология, которая позволяет каждому классу трафика предоставить одинаковые права. Как правило не используется, т.к. мало даёт с точки зрения улучшения качества сервиса.

Взвешенная справедливая очередь (Weighted Fail Queuing, WFQ). Технология, которая предоставляет разным классам трафика разные права (можно сказать, что "вес" у разных очередей разный), но одновременно обслуживает все очереди. "На пальцах" это выглядит так: все пакеты делятся на логические очереди, используя в качестве критерия поле IP Precedence. Это же поле задаёт и приоритет (чем больше, тем лучше). Дальше, маршрутизатор вычисляет, пакет из какой очереди "быстрее" передать, и передаёт именно его. Считает он это по формуле:



dT=(t(i)-t(0))/(1+IPP)

IPP - значение поля IP Precedence

t(i) - Время, требуемое на реальную передачу пакета интерфейсом. Можно вычислить, как L/Speed, где L - длина пакета, а Speed - скорость передачи интерфейса

Такая очередь по умолчанию включена на всех интерфейсах маршрутизаторов cisco, кроме интерфейсов точка-точка (инкапсуляция HDLC или PPP).

WFQ имеет ряд минусов: такая очередизация использует уже отмаркированные ранее пакеты, и не позволяет самостоятельно определять классы трафика и выделяемую полосу. Мало того, как правило, уже никто не маркирует полем IP Precedence, поэтому пакеты идут немаркированные, т.е. все попадают в одну очередь.

Развитием WFQ стала **взвешенная справедливая очередь, основанная на классах (Class-Based Weighted Fair Queuing, CBWFQ)**. В этой очереди администратор сам задаёт классы трафика, следуя различным критериям, например, используя ACL как шаблон или анализируя заголовки протоколов (см.NBAR). Далее для этих классов определяется "вес", и пакеты их очередей обслуживаются соразмерно весу (больше вес - больше пакетов из этой очереди уйдёт в единицу времени)

Но такая очередь не обеспечивает строгого пропускания наиболее важных пакетов (как правило, голосовых или пакетов других интерактивных приложений). Поэтому появился гибрид Priority и Class-Based Weighted Fair Queuing - **PQ-CBWFQ**, также известный как **Low Latency Queuing (LLQ)**. В этой технологии можно задать до 4х приоритетных очередей, остальные классы обслуживать по механизму CBWFQ.

LLQ - наиболее удобный, гибкий и часто используемый механизм. Но он требует настройки классов, настройки политики и применения политики на интерфейсе. Подробнее о настройках расскажу дальше.

Таким образом процесс предоставления качества обслуживания можно поделить на 2 этапа:

Маркировка. Поближе к источникам.

Обработка пакетов. Помещение их в физическую очередь на интерфейсе, подразделение на логические очереди и предоставление этим логическим очередям различных ресурсов.

Технология QoS - достаточно ресурсоёмкая и весьма существенно грузит процессор. И тем сильнее грузит, чем глубже в заголовки приходится залезать для классификации пакетов. Для сравнения: маршрутизатору гораздо проще заглянуть в заголовок IP пакета и проанализировать там 3 бита ToS, нежели раскручивать поток практически до уровня приложения, определяя, что за протокол идёт внутри (технология NBAR)

Для упрощения дальнейшей обработки трафика и создания так называемой "области доверия" (trusted boundary), где мы верим всем заголовкам, относящимся к QoS, ловить пакеты, раскидывать их по классам надо поближе к источникам данных, даже если там никаких бутылочных горлышек нет. В качестве действия надо «перекрашивать» (маркировать) трафик по-своему или переносить значения QoS-заголовков более высокого уровня на нижние.

Например, на маршрутизаторе ловим все пакеты из гостевого WiFi домена (предполагаем, что там могут быть не управляемые нами компьютеры и софт, который может использовать нестандартные QoS-заголовки), меняем любые заголовки IP на дефолтные, сопоставляем заголовкам 3 уровня (DSCP) заголовки канального уровня (CoS), чтобы дальше и коммутаторы могли эффективно приоритизировать трафик, используя только метку канального уровня.

Настройка LLQ

Настройка очередей заключается в настройке классов, затем для этих классов надо определить параметры полосы пропускания и применить всю созданную конструкцию на интерфейс.

Создание классов:

```
class-map NAME
  match ?

  access-group         Access group
  any                  Any packets
  class-map            Class map
  cos                  IEEE 802.1Q/ISL class of service/user priority values
  destination-address  Destination address
```

```
  discard-class        Discard behavior identifier
  dscp                 Match DSCP in IP(v4) and IPv6 packets
  flow                 Flow based QoS parameters
  fr-de                Match on Frame-relay DE bit
  fr-dlci              Match on fr-dlci
  input-interface      Select an input interface to match
  ip                   IP specific values
  mpls                 Multi Protocol Label Switching specific values
  not                  Negate this match result
  packet               Layer 3 Packet length
  precedence           Match Precedence in IP(v4) and IPv6 packets
  protocol             Protocol
  qos-group            Qos-group
  source-address       Source address
  vlan                 VLANs to match
```

Пакеты в классы можно рассортировывать по различным атрибутам, например, указывая ACL, как шаблон, либо по полю DSCP, либо выделяя конкретный протокол (включается технология NBAR)

Создание политики:

```
policy-map POLICY
  class NAME1
      ?
  bandwidth         Bandwidth
  compression       Activate Compression
  drop              Drop all packets
  exit              Exit from class action configuration mode
  log               Log IPv4 and ARP packets
  netflow-sampler   NetFlow action
  no                Negate or set default values of a command
  police            Police
  priority          Strict Scheduling Priority for this Class
  queue-limit       Queue Max Threshold for Tail Drop
```

```
  random-detect    Enable Random Early Detection as drop policy
  service-policy   Configure Flow Next
  set              Set QoS values
  shape            Traffic Shaping
  <cr>
```

Для каждого класса в политике можно либо выделить приоритетно кусок полосы:

```
policy-map POLICY
  class NAME1
    priority ?

  <8-2000000>  Kilo Bits per second
  percent      % of total bandwidth
```

и тогда пакеты этого класса смогут всегда рассчитывать как минимум на этот кусок.

Либо описать, какой "вес" имеет данный класс в рамках CBWFQ

```
policy-map POLICY
  class NAME1
    bandwidth ?
  <8-2000000>  Kilo Bits per second
  percent      % of total Bandwidth
  remaining    % of the remaining bandwidth
```

В обоих случаях можно указать как абсолютное значение, так и процент от всей доступной полосы

Возникает резонный вопрос: а откуда маршрутизатор знает ВСЮ полосу? Ответ банален: из параметра **bandwidth** на интерфейсе. Даже если он не сконфигурирован явно, какое то его значение обязательно есть. Его можно посмотреть командой `sh int`.

Также обязательно надо помнить, что по умолчанию вы распоряжаетесь не всей полосой, а лишь 75%. Остальное отдано для зарезервированного класса class-default. Туда попадают пакеты, явно не попавшие в другие классы. Эту настройку можно поменять, явно указав на интерфейсе параметр

```
Max-reserved-bandwidth [percent]
```

Дефолтному классу трафика тоже можно явно задать полосу:

```
policy-map POLICY
  class class-default
    bandwidth percent 10
```

Маршрутизаторы ревностно следят, чтобы администратор не выдал случайно больше полосы, чем есть, и ругаются на такие попытки.

Создаётся впечатление, что политика будет выдавать классам полосы не больше, чем написано. Однако, такая ситуация будет лишь в том случае, если все очереди наполнены. Если же какая то пустует, то выделенную ей полосу наполненные очереди поделят пропорционально своему "весу".

Работать же вся эта конструкция будет так:

Если идут пакеты из класса с указанием priority, то маршрутизатор сосредотачивается на передаче этих пакетов. Причем т.к. таких приоритетных очередей может быть несколько, то между ними полоса делится пропорционально указанным процентам.

Как только все приоритетные пакеты закончились, наступает очередь CBWFQ. За каждый отсчёт времени из каждой очереди "зачёрпывается" доля пакетов, указанная в настройке для данного класса. Если же часть очередей пустует, то их полоса делится пропорционально "весу" класса между загруженными очередями.

Применение на интерфейсе:

```
int s0/0
  service-policy [input|output] POLICY
```

А что же делать, если надо строго рубить пакеты из класса, выходящие за дозволенную скорость? Ведь указание **bandwidth** лишь распределяет полосу между классами, когда очереди загружены.

Для решения этой задачи для класса трафика в политике есть технология

```
police [speed] [birst] conform-action [действие] exceed-action [действие]
```

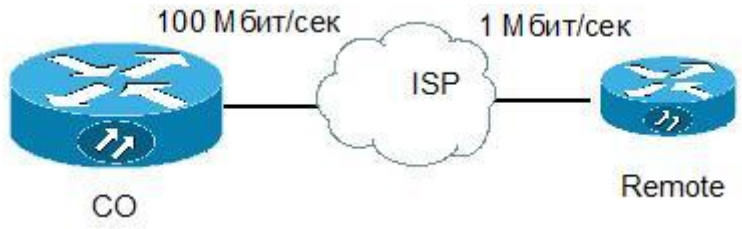
Она позволяет явно указать желаемую среднюю скорость (speed), максимальный "выброс", т.е. количество передаваемых данных за единицу времени. Чем больше "выброс", тем больше реальная скорость передачи может отклоняться от желаемой средней. Также указываются: действие для нормального трафика, не превышающего указанную скорость и действие для трафика, превысившего среднюю скорость. Действия могут быть такими

```
police 100000 8000 conform-action ?

  drop                              drop packet
```

```
exceed-action                        action when rate is within conform and
                                     conform + exceed burst
set-clp-transmit                     set atm clp and send it
set-discard-class-transmit           set discard-class and send it
set-dscp-transmit                    set dscp and send it
set-frde-transmit                    set FR DE and send it
set-mpls-exp-imposition-transmit     set exp at tag imposition and send it
set-mpls-exp-topmost-transmit        set exp on topmost label and send it
set-prec-transmit                    rewrite packet precedence and send it
set-qos-transmit                     set qos-group and send it
transmit                             transmit packet
<cr>
```

Часто возникает также и другая задача. Предположим, что надо ограничить поток, идущий в сторону соседа с медленным каналом.



Дабы точно предсказать, какие пакеты дойдут до соседа, а какие будут уничтожены в силу загруженности канала на "медленной" стороне, надо на "быстрой" стороне создать политику, которая бы заранее обрабатывала очереди и уничтожала избыточные пакеты.

И тут мы сталкиваемся с одной очень важной вещью: для решения этой задачи надо сэмулировать "медленный" канал. Для этой эмуляции не достаточно только раскидать пакеты по очередям, надо ещё сэмулировать физический буфер "медленного" интерфейса. У каждого интерфейса есть скорость передачи пакетов. Т.е. в единицу времени каждый интерфейс может передать не более, чем N пакетов. Обычно физический буфер интерфейса рассчитывают так, чтобы обеспечить "автономную" работу интерфейсу на несколько единиц времени. Поэтому физический буфер, скажем, GigabitEthernet будет в десятки раз больше какого-нибудь интерфейса Serial.

Что же плохого в том, чтобы запомнить много? Давайте рассмотрим подробно, что произойдёт в случае, если буфер на быстрой передающей стороне будет существенно больше буфера принимающей.

Пусть для простоты есть 1 очередь. На "быстрой" стороне сэмулируем малую скорость передачи. Это значит, что попадая под нашу политику пакеты начнут накапливаться в очереди. Т.к. физический буфер большой, то и логическая очередь получится внушительной. Часть приложений (работающих через TCP) поздно получат уведомление о том, что часть пакетов не получена, и долго будут держать большой размер окна, нагружая сторону-приемник. Это будет происходить в том идеальном случае, когда скорость передачи будет равна или меньше скорости приёма. Но интерфейс принимающей стороны может быть сам загружен и другими пакетами и тогда маленькая очередь на принимающей стороне не сможет вместить всех пакетов, передаваемых ей из центра. Начнутся потери, которые повлекут за собой дополнительные передачи, но в передающем буфере ведь ещё останется солидный "хвост" ранее накопленных пакетов, которые будут передаваться "вхолостую", т.к. на принимающей стороне не дождались более раннего пакета, а значит более поздние будут просто проигнорированы.

Поэтому для корректного решения задачи понижения скорости передачи к медленному соседу физический буфер тоже надо ограничить.

Делается это командой

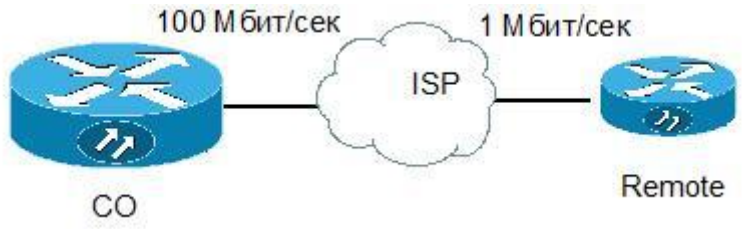
```
shape average [speed]
```

Ну а теперь самое интересное: а как быть, если мне помимо эмуляции физического буфера надо внутри него создать логические очереди? Например, выделить приоритетно голос?

Для этого создаётся так называемая вложенная политика, которая применяется внутри основной и делит на логические очереди то, что в неё попадает из родительской.

Пришло время разобрать какой-нибудь залихватский пример.

Рассмотрим ту же ситуацию:



Пусть мы собираемся создать устойчиво работающие голосовые каналы через интернет между CO и Remote. Для простоты пусть сеть Remote (172.16.1.0/24) имеет только связь с CO (10.0.0.0/8). Скорость интерфейса на Remote - 1 Мбит/сек и 25% этой скорости выделяется на голосовой трафик.

Тогда для начала нам надо выделить приоритетный класс трафика с обеих сторон и создать политику для данного класса. На CO дополнительно создадим класс, описывающий трафик между офисами

На CO:

```
class-map RTP
  match protocol rtp

policy-map RTP
  class RTP
    priority percent 25

ip access-list extended CO_REMOTE
  permit ip 10.0.0.0 0.255.255.255 172.16.1.0 0.0.0.255

class-map CO_REMOTE
  match access-list CO_REMOTE
```

На Remote поступим иначе: пусть в силу дохлости железа мы не можем использовать NBAR, тогда нам остаётся только явно описать порты для RTP

```
ip access-list extended RTP
   permit udp 172.16.1.0 0.0.0.255 range 16384 32768 10.0.0.0
0.255.255.255 range 16384 32768

class-map RTP
  match access-list RTP

policy-map QoS
  class RTP
    priority percent 25
```

Далее, на CO надо сэмулировать медленный интерфейс, применить вложенную политику для приоритизации голосовых пакетов

```
policy-map QoS
```

```
  class CO_REMOTE
    shape average 1000000
    service-policy RTP
```

и применить политику на интерфейсе

```
int g0/0
  service-policy output QoS
```

На Remote установим параметр **bandwidth** (в кбит/сек) в соответствие со скоростью интерфейса. Напомню, что именно от этого параметра будет считаться 25%. И применим политику.

```
int s0/0
  bandwidth 1000
  service-policy output QoS
```

Повествование было бы не полным, если не охватить возможности коммутаторов. Понятно, что чисто L2 коммутаторы не способны так глубоко заглядывать в пакеты и делить их на классы по тем же критериям.

На более умных L2/3 коммутаторах на маршрутизируемых интерфейсах (т.е. либо на interface vlan, либо если порт выведен на третий уровень командой `no switchport`) применяется та же конструкция, что работает и на маршрутизаторах, а если порт или весь коммутатор работает в режиме L2 (верно для моделей 2950/60) , то там для класса трафика можно использовать только указание `police`, а `priority` или `bandwidth` не доступны.

С сугубо защитной точки зрения знание основ QoS позволит оперативно предотвращать бутылочные горла, вызванные работой червей. Как известно, червь сам по себе довольно агрессивен на фазе распространения и создаёт много паразитного трафика, т.е. по сути атаку отказа в обслуживании (Denial of Service, DoS). Причем часто червь распространяется по нужным для работы портам (TCP/135,445,80 и др.) Просто закрыть на маршрутизаторе эти порты было бы опрометчиво, поэтому гуманнее поступать так:

1. Собираем статистику по сетевому трафику. Либо по NetFlow, либо NBARом, либо по SNMP.

2. Выявляем профиль нормального трафика, т.е. по статистике в среднем протокол HTTP занимает не больше 70%, ICMP - не больше 5% и т.д. Такой профиль можно либо создать вручную, либо применив накопленную NBARом статистику. Мало того, можно даже автоматически создать классы, политику и применить на интерфейсе командой `autoqos` :)

3. Далее, можно ограничить для нетипичного сетевого трафика полосу. Если вдруг и подцепим заразу по нестандартному порту, большой беды для шлюза не будет: на загруженном интерфейсе зараза займет не более выделенной части.

4. Создав конструкцию (`class-map - policy-map - service-policy`) можно оперативно реагировать на появление нетипичного всплеска трафика, создавая вручную для него класс и сильно ограничивая полосу для этого класса.

Защищаем маршрутизатор

Следуя аксиомам безопасности, будем считать, что любой узел в сети является потенциальной целью. Поэтому хорошо бы знать, какие потенциально уязвимые места есть у этих самых узлов. Рассмотрим маршрутизатор cisco. Сразу же возникнут возражения: их много, сервисы поддерживаемые – разные и вообще, трудно свалить в одну кучу CRS-1 и древний 1600. Однако, я не ставлю своей целью охватить всё, но кое какие общие вещи опишу.

Итак, первое, что надо запомнить - маршрутизатор по умолчанию не блокирует на интерфейсе никакой нормальный трафик (фреймы с неправильной контрольной суммой не в счёт). Однако часть пакетов при более глубоком рассмотрении (процессором) маршрутизатор-таки признает ненужными, например:

1. Пакеты с TTL =0 или меньше
2. Пакеты, которые неизвестно куда отправить (сеть назначения пакета не присутствует в таблице маршрутизации и нет никакого явного правила отправки пакета (PBR))
3. Пакеты, относящиеся к служебным протоколам (например, протоколам маршрутизации), которые не запущены на маршрутизаторе.

Эти уничтоженные пакеты могут сыграть злую шутку: если такого трафика будет много, то он может существенно загрузить процессор маршрутизатора.

Далее, кроме транзитного трафика, маршрутизатор обрабатывает некоторый служебный трафик (направленный на него самого). Часто по умолчанию (или незнанию) на маршрутизаторе запущены ненужные для работы протоколы. Они опасны тем, что маршрутизатор обрабатывает пакеты этого протокола. И можно устроить, например, DoS атаку, узнать удалённо сведенья, не предназначенные для распространения или исследовать топологию сети. К таким протоколам относятся

1. TFTP (маршрутизатор может выступать сервером TFTP).
2. BOOTP (может раздавать бездисковым станциям их файлы настройки)
3. DHCP (Маршрутизатор может выступать сервером и клиентом)
4. TCP Small Servers (TCP Echo, Finger и др.)
5. UDP Small Servers (UDP Echo, Discard и др.)

6. CDP (Cisco Discovery Protocol)
7. NTP (Network Time Protocol. Маршрутизатор может выступать сервером и клиентом)
8. DNS (Включен по умолчанию броадкастовый поиск ДНС серверов в сегменте)
9. PAD (Packet Assembler/Disassembler)
10. SNMP (Часто сконфигурированы дефолтные community)

Как правило, если данные протоколы не нужны, их лучше отключить.

1. `no tftp-server`
2. `no ip bootp server` (старая команда `no service bootp`)
3. `no ip dhcp pool` (`no service dhcp`)
4. `no service tcp-small-servers`
5. `no service udp-small-servers`
6. `no cdp run` (глобально), `no cdp enable` (на конкретном интерфейсе). Не стоит выключать этот протокол, если к интерфейсу подключен cisco ip phone, т.к. именно по этому протоколу происходит автоопределение подключенного устройства (данная рекомендация больше характерна для коммутаторов, но всё же)
7. `no ntp master`, `no ntp server`
8. `no ip domain-lookup`. Помните, что часто DNS на маршрутизаторе нужен, так что отключать надо не всегда
9. `no service pad`
10. `no snmp-server community {public|private}`

Однако, даже если выключить эти и другие служебные протоколы (например, http, https, ssh), пакеты этих протоколов, приходящие на интерфейс маршрутизатора, попадут в мозг и только там будут откинуты. Т.е. даже выключив все, можно попробовать нагрузить процессор маршрутизатора отбрасыванием мусора.

Хотелось бы научиться отбрасывать такие пакеты, не нагружая мозг. Также, часто возникает задача ограничить нагрузку служебным трафиком на процессор. Например, указав максимальное количество служебных пакетов (всего или по отдельным протоколам) в очереди или количество служебных пакетов в секунду.

Эти задачи решаются при помощи специального режима

```
control-plane  host
```

Чтобы воспользоваться этой технологией можно, создать специальные классы трафика

```
class-map type ?
```

```
  access-control   access-control specific class-map
  control          Configure a control policy class-map
  inspect          Configure CBAC Class Map
  logging          Class map for control-plane packet logging
  port-filter      Class map for port filter
  queue-threshold  Class map for queue threshold
  stack            class-map for protocol header stack specification
```

Создать специальную политику (Policy-map type)

```
policy-map type ?
    access-control   access-control specific policy-map
    control          Configure a control policy policy-map
    inspect          Configure CBAC Policy Map
    logging          Control-plane packet logging
    port-filter      Control-plane tcp/udp port filtering
    queue-threshold  Control-plane protocol queue limiting
```

И применить её в этом режиме:

```
control-plane host
  service-policy type ?
    logging          Control-plane packet logging
    port-filter      Control-plane tcp/udp port filtering
    queue-threshold  Control-plane protocol queue limiting
```

Ограничение же нагрузки на мозг служебными пакетами организуется похоже, только для этого достаточно описать обычный класс трафика, обычную политику, где в качестве действия указать ограничение словом police

```
police rate [units] pps
```

Разберем примеры:

1. Ограничим количество телнетовских пакетов от всех сетей, кроме хоста 10.1.1.100, до 100 пакетов в секунду

Для этого напишем список доступа

```
 ip access-list extended TELNET
  deny tcp host 10.1.1.100 any eq 23
```

```
  permit tcp any any eq 23
```

Далее, создадим класс трафика

```
class-map TELNET
  match access-group TELNET
```

Опишем политику

```
policy-map TELNET
  class TELNET
    police rate 100 pps
```

И повесим политику в control-plane

```
control-plane host
  service-policy input TELNET
```

2. Заблокируем пакеты, направленные на порты приложений, не используемые маршрутизатором

Создадим специальный класс трафика

```
class-map type port-filter PORTS
  match closed-ports
```

Опишем специальную политику

```
policy-map type port-filter PORTS
  class PORTS
    drop
```

И повесим политику в control-plane

```
control-plane host
  service-policy type port-filter input PORTS
```

Чтобы защитить management-plane, т.е. управление маршрутизатором надо помнить следующие моменты:

1. По возможности надо использовать безопасные протоколы управления: ssh, https. Для этого надо выработать ключевую пару RSA, указать правила аутентификации и включить поддержку https (ip http secure-server)
2. Даже при использовании безопасных протоколов управления, а особенно при невозможности их использования, надо ограничивать административный доступ снаружи

и изнутри при помощи списков доступа, примененных на интерфейсах, терминальных линиях (line vty) или в режиме control-plane host

3. Желательно применять сложные пароли, минимальной длиной 8 символов и содержащие цифры, буквы разного регистра и символы. А чтобы какой-нибудь юный администратор маршрутизатора не нарушил это правило создания паролей, есть команда

   ```
   security passwords min-length [длина]
   ```

4. Для защиты от «подглядывания» можно шифровать пароли в конфигурациооно м файле при помощи команды

   ```
   service password-encryption
   ```

   Однако применяемый метод шифрования нестоек и легко подвергается дешифрованию Поэтому пароли в конфиге лучше иметь захешированными. Тогда не будет возможности ни подглядеть пароль, ни расшифровать, т.к. хэш – функция односторонняя. Для этого применяйте не слово “password” , но слово “secret”

   Примеры:

   ```
   username admin secret {пароль}
   ```

   ```
   enable secret {пароль}
   ```

5. Не забывайте о порте AUX. Это практически та же консоль и имеющий доступ к железу сможет подключиться, используя AUX, к командной строке. Если пароля на AUX не будет, то подключившийся сможет попасть как минимум в непривилегированный режим.

6. У многих маршрутизаторов есть функция защиты от подбора паролей. Можно блокировать пользователя после N неправильно введенных паролей, а можно после нескольких попыток вставлять задержку.

   Пример:

   ```
   security authentication failure rate [попытки] [log]
   ```

   ```
   login block-for [сек] attempts [попытки] within [сек]
   ```

   После N неправильных паролей (по умолчанию – 10) будет вставлена 15 секундная задержка. Слово log позволяет логгировать такие события.

7. Помните, что при использовании протокола SNMP желательно использовать 3 версию протокола с аутентификацией и шифрованием. При использовании более ранних, практически ничем не защищённых версий, строго следите за тем, что принятые по умолчанию community отключены.

8. В сетях с большим количеством устройств имеет смысл выделять так называемую сеть для управления (OOB, Out-of-Band management). Это отдельный сегмент сети, не имеющий пересечений с сетью передачи данных. До недавних пор маршрутизатор мог быть помещен в OOB только посредством консольного сервера, но в новых IOS вы можете административно задать интерфейс, с которого можно настраивать маршрутизатор и

только с него. Делается это всё в том же режиме control-plane host явным указанием интерфейса и разрешенных протоколов

Пример:

```
control-plane host

  management-interface f0/0 allow ssh snmp
```

Защита протоколов маршрутизации

Защита динамических протоколов маршрутизации тоже является весьма важной темой, т.к. если злоумышленнику удастся повредить таблицу маршрутизации, нужные пакеты просто будут уничтожены или пойдут «не туда». Поэтому очень рекомендую при работе с динамическими протоколами маршрутизации использовать аутентификацию обновлений, желательно MD5 суммой (хэшем). Такую технологию поддерживают почти все протоколы: BGP, OSPF, RIPv2, EIGRP. Часть протоколов поддерживает также и аутентификацию clear text (просто ключом), но т.к. ключ передается в самом пакете обновления, назвать такой механизм защищённым язык не поворачивается.

Для настройки аутентификации по MD5 как правило надо:

1. Описать одинаковый ключ на всех маршрутизаторах, работающих по одному протоколу (или на конкретном интерфейсе, в конкретной зоне)
2. Настроить метод защиты (без защиты, clear text, MD5)
3. Включить механизм

Пример: протокол OSPF

```
Ro(config-if)# ip ospf authentication message-digest

Ro(config-if)# ip ospf message-digest-key 1234 md5 cisco
```

Где 1234 – номер ключа, а от слова «cisco» будет вычислен md5 хэш.

Послесловие:

Это первая версия компиляции. Надеюсь, что с выходом новых статеек она будет пополняться. Если вдруг у кого-нибудь возникнет творческий энтузиазм и вы решите выступить соавтором (корректором, дополнителем) подобного рода материалов, я буду рад и обязательно упомяну. Материалы, пожелания, идеи можно смело слать на

fedorov@ciscotrain.ru

или

4u@anticisco.ru

С уважением к читателям, Сергей Фёдоров, инструктор.

UPD 1. Исправлены орфографические, стилистические и понятийные ошибки. Отдельное спасибо Лене Шер.