Администрирование Linux. Часть7.

Пособие для дистанционных курсов.
http://linuxnavigator.ru.

21.07.2009

ОГЛАВЛЕНИЕ

НАСТРОЙКА OPENVPN.

OpenVPN замечательный софт, позволяющий создавать VPN соединения.

Возможные варианты:

- *Точка — точка*. Кода шифруется вся информация, передаваемая между двумя компьютерами.
- *Точка — сеть*. То же самое, что и в предыдущем случае, но ещё клиент получает доступ во внутреннюю сеть, стоящую за сервером OpenVPN.
- *Сеть — сеть*. Когда мы связываем между собой две сети, через шифрованный тоннель.

В своей работе OpenVPN использует UDP пакеты, зашифрованные при помощи ssl. UDP даёт скорость обмена, а ssl позволяет использовать стандартные механизмы шифрации.

Ещё одним из неоспоримых преимуществ OpenVPN является то, что он может работать через любое количество NAT преобразований. Т.е. клиент может находиться где угодно. Главное, что бы он мог получить доступ к серверу.

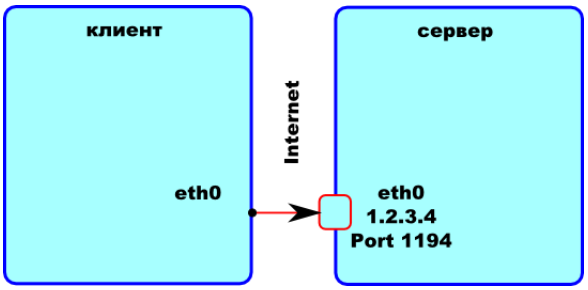
Кроме реализации Open VPN для UNIX систем, существует реализация и для Windows. В том числе и для Vista! В которой она без проблем работает. Причем реализация для Windows может выполнять как роль клиента, так и роль сервера. В последнем случае её можно запустить в виде сервиса.

КАК ЭТО РАБОТАЕТ.

Как всё это работает? Как всегда, очень просто. Предположим, что мы хотим сделать один сервер OpenVPN, к которому могут подключаться несколько клиентов. Конечно есть возможность для каждого клиента запускать свой экземпляр сервера, но этот вариант будет *кушать* много памяти.

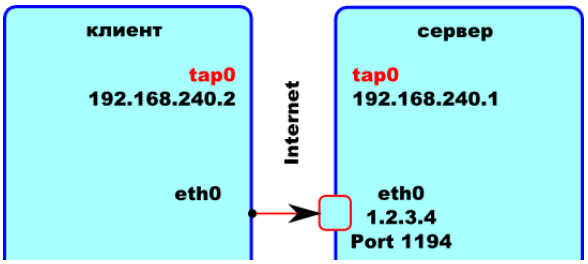
Для работы Open VPN используются псевдо сетевые интерфейсы *tun (tunnel software network interface)* или *tap (ethernet tunnel software network interface)*. В нашем случае мы будем использовать *tap* интерфейс, который эмулирует Ethernet интерфейс. *Tun* используют когда хотят соединить только две машины, поскольку он работает как point-to-point интерфейс. Драйвера интерфейсов стандартно присутствуют в ядре Linux и мы просто будем ими пользоваться.

При настройке сервера необходимо указать порт, на котором он будет слушать запросы. А на клиенте указать IP адрес машины, где находится сервера и порт, к которому необходимо подключаться. Предположим, что на сервере мы выбрали порт 1194.

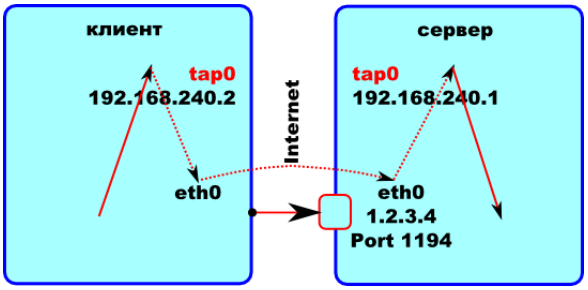


Клиент подключается к серверу на порт 1194, и *только это соединение* шифруется по ssl. Если с клиента послать пакет на сервер на любой другой порт, это соединение шифроваться не будет. Что же передаётся в зашифрованном соединении? В этом соединении передаются пакеты, которые были направлены на специальный сетевой интерфейс *tap0*.

При конфигурации сервера можно указывать IP адреса, которые получают *tun* или *tap* интерфейсы на клиенте и на сервере. Предположим, что на сервере мы присваиваем интерфейсу *tap0* IP адрес *192.168.240.1*. Так же на сервере мы можем указать диапазон IP адресов, которые будут присваиваться клиентским машинам. На сервере запоминается, какой IP какому клиенту был присвоен и при следующем подключении клиент получает тот же IP адрес. Предположим, что за клиентом был закреплён IP *192.168.240.2*.



Дальше программа на обеих машинах все пакеты, направляемые на интерфейс *tap0*, шифрует и передаёт на другую машину. Т.е. если вы хотите передать зашифрованные данные с клиента на сервер, отправляйте их на IP *192.168.240.1*. А если с сервера на клиент, тогда на IP *192.168.240.2*. Вам будет казаться, что машина с IP 192.168.240.x находится в одном сегменте сети с вашей машиной.



Вот как виден этот интерфейс в системе:

```
# ifconfig tap0
tap0      Link encap:Ethernet  HWaddr 62:29:85:6B:D7:10
          inet addr:192.168.240.1  Bcast:192.168.240.255
          Mask:255.255.255.0
```

А так эта сеть видна в таблице маршрутизации.

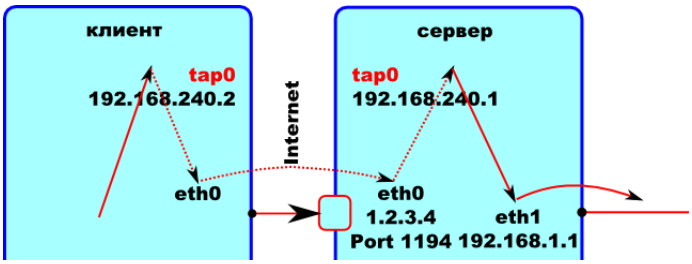
```
# route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
192.168.240.0   0.0.0.0         255.255.255.0   U     0      0        0 tap0
```

Я немного *порезал* вывод программ. Ну и в случае *route* у меня съехали строки. Но всё равно видно, что с точки зрения таблицы маршрутизации — это обыкновенная сеть, подключённая к интерфейсу *tap0*.

Ну как? Всё понятно? Если понятно, продолжайте читать дальше. Если не понятно, перечитайте и попытайтесь понять, как это работает. Потому что читать дальше без понимания как всё работает бессмысленно.

ДОСТУП КО ВНУТРЕННИМ СЕТЯМ.

Напомню, что на данным момент установлено зашифрованное соединение клиента к серверу, и мы видим машину клиент как машину в том же сегменте сети, что и сервер. Теперь нам необходимо дать доступ клиенту ко внутренней сети, находящейся *за* сервером. Предположим, что это будет сеть 192.168.1/24, подключённая к интерфейсу *eth1* на сервере.



Подумайте. Пакеты ходят, в таблице маршрутизации локальная сеть 192.168.240/24 прописана. Ну что? Мысль промелькнула? Догадались?

Именно так, очень простое решение и очень стандартное. Нам надо на клиенте прописать маршрут к сети *192.168.1/24* через интерфейс *tap0*! И всё! Тут работает стандартный роутинг! Ничего выдумывать не надо.

Ну а если вы хотите получать доступ ко внутренней сети, подключенной к другому интерфейсу клиентской машины. Вы… Ну вы поняли ☺ — добавляем на сервере маршрут к сети клиента через интерфейс *tap0*. Я же говорил — стандартный роутинг.

Единственно, что добавлять и, в случае разрыва соединения, удалять маршруты будет сама программа openvpn. У неё есть специальные параметры. Но об этом мы поговорим позже. Сейчас займемся настройкой простого режима: *машина-машина*.

НАСТРОЙКА СЕРВЕРА.

Я надеюсь, что вы внимательно прочитали предыдущий раздел и поняли как работает openvpn. Теперь перейдём к практике. В первую очередь настроим сервер. OpenVPN не входит в стандартную поставку дистрибутива CentOS. Готовые пакеты можно найти в репозитарии dag, который мы уже подключили. Поэтому ставиться он стандартным способом.

```
# yum install openvpn
```

Сначала создадим файл, необходимый для работы сервера. Для чего он нужен, я так и не понял. Но без него сервер работать не будет.

```
# cd /etc/openvpn
# openssl dhparam -out dh1024.pem 1024
```

В директории */etc/openvpn* создадим конфигурационный файл *server.conf*. Имя файла не имеет значение, главное, что бы он имел расширение *.conf*. Стартовый скрипт запускает столько экземпляров программы openvpn, сколько файлов с расширением *.conf* будет обнаружено в этой директории.

Содержимое файла показано ниже:

```
dev tap0
proto udp
mode server
comp-lzo
log-append /var/log/openvpn.log
daemon
ifconfig-pool 192.168.240.2 192.168.240.12
ifconfig 192.168.240.1 255.255.255.0
tls-server
dh /etc/openvpn/dh1024.pem
ca /etc/pki/CA/CA.crt
cert /etc/pki/tls/certs/st1.kryukov.biz.pem
key /etc/pki/tls/private/st1.kryukov.biz.key
```

```
port 1194
user nobody
group nobody
persist-tun
persist-key
verb 0
```

Разбираемся с параметрами.

- *dev* — определяет тип сетевого интерфейса. Для того, что бы к нашему серверу могли подключаться несколько клиентов, мы будем использовать устройство *tap0*. Устройство *tun* обычно применяют тогда, когда для подключения каждого клиента запускается отдельный экземпляр сервера.
- *proto* — определяет транспортный протокол, который будет использоваться для передачи данных. Самый лучший вариант — это использование UDP.
- *mode* — определяет режим работы. В случае сервера, используем параметр *server*.
- *comp-lzo* — использовать компрессию при передаче данных.
- *log-append* — определяет имя файла журнальной регистрации. При каждом запуске сервера информация будет добавляться в уже существующий файл.
- *daemon* — работать в режиме демона.
- *ifconfig-pool* — определяет диапазон IP адресов, которые будут выдаваться клиентам. После первого подключения IP адрес закрепляется за клиентом. При повторном подключении клиента ему выдаётся тот же IP.
- *ifconfig* — определяет IP адрес сервера OpenVPN. О том, как используются указанные IP и как конфигурируется сеть, будет рассказано ниже.
- *tls-server* — заставляет программу использовать ssl для работы с клиентами.
- *dh* — определяет путь к файлу, который мы генерировали при помощи программы *openssl dhparam*. Для чего он нужен я не знаю, но его наличие обязательно.
- *ca* — определяет путь к файлу с публичным сертификатом корневого центра сертификации.
- *cert* — определяет путь к файлу сертификата нашего сервера.
- *key* — определяет путь к файлу ключа нашего сервера.
- *port* — порт, на котором будет слушать запросы сервер.
- *user* — пользователь, с правами которого будет работать программа.
- *group* — группа, с правами которой будет работать программа.
- *persist-tun* — при получении сигнала USR1 (перечитать конфигурацию и перезапустить сервер), программа обычно закрывает устройство *tun*, а потом открывает его по новой. Этот параметр заставляет программу не выключать устройство при получении этого сигнала.
- *persist-key* — не перечитывать файлы ключей, при получении сигнала USR1. Рекомендуется использовать тогда, когда программа запускается не с правами пользователя root.
- *ping* — если в течении определенного промежутка времени (в нашем случае 20 секунд), в соединении не было ни одного пакета, посылается пакет ping. Это необходимо делать, что бы firewall-ы с отслеживанием соединений не делали задержек при посылке нового пакета, после длительного простоя. Такая задержка может быть весьма значительной. Следует учитывать, что это не пакет *echo-request* протокола icmp. Это внутренний пакет OpenVPN и он посылается только в одну сторону. Если написать этот параметр только на сервере, тогда только сервер будет посылать такие пакеты.
- *verb* — определяет уровень отладки. Насколько подробные сообщения помещать в файл журнальной регистрации. Значение 0 отменяет посылку сообщений.

Делает так, что бы сервер автоматически запускался при старте компьютера:

```
# chkconfig openvpn on
```

И запускаем сервер.

```
# service openvpn start
```

Если сервер не запустился, смотрите логи в файле */var/log/openvpn.log*. И пытаетесь разобраться, что вы сделали не так.

Теперь заключительный штрих — настройка firewall. Нам необходимо в цепочке *INPUT* разрешить принимать *udp* пакеты, приходящие на порт *1194*. Поэтому отрываем файл *~/bin/rc.fw* и добавляем в функцию init следующие строки:

```
### Open VPN input connections
$IPT -A INPUT -p udp --dport 1194 -j ACCEPT
```

Строки добавляем до завершающих строк функции init:

```
# SAVE rules in file ==
$IPTS -c > /etc/sysconfig/iptables
echo "Done"
```

Перезапускаем firewall.

```
# rc.fw init
```

Доступ к серверу OpenVPN мы обеспечили. А вот разрешение хождения пакетов внутри VPN — это немного другая задача. Мы будем делать такие разрешения по мере потребности открытия доступа к службам внутри VPN соединения.

КЛИЕНТ WINDOWS.

Как я уже писал выше, у OpenVPN есть реализация и для Windows. Сейчас мы настроим клиент. Настройку сервера вы посмотрите сами.

Для начала, необходимо сгенерировать ключ и сертификат для клиентской машины. Мы будем делать это на Linux машине, где находится сервер OpenVPN.

```
# cd /etc/pki/tls
# openssl req -days 3650 -nodes -new -keyout \
             private/win.st1.kryukov.biz.key -out \
             certs/win.st1.kryukov.biz.csr -extensions client
# openssl ca -days 3650 -out certs/win.st1.kryukov.biz.pem \
             -in certs/win.st1.kryukov.biz.csr -extensions client
```

В качестве имени файла ключа (параметр *-keyout*) и файла запроса (параметр *-out*) рекомендую использовать имя компьютера, для которого они генерируются. В дальнейшем вам будет легко отличать какие файлы для каких машин предназначены.

Клиент для Windows берем тут: http://openvpn.net/index.php/downloads.html. Если вы используете Vista, берите клиент 2.1 самой последней версии. Несмотря на то, что на момент написания этого раздел он был в состоянии rc (release candidate), клиент работает великолепно.

Скачайте файлы сертификата (*win.st1.kryukov.biz.pem*), ключа клиентской машины (*win.st1.kryukov.biz.key*) и публичного ключа сервера сертификации (*/etc/pki/CA/CA.crt*). Сделать это можно например, при помощи программы Winscp. Поместите их в директорию *C:\Program Files\OpenVPN\config*.

После установки клиента, перейдите в директорию *C:\Program Files\OpenVPN\config* и создайте текстовый конфигурационный файл *client.ovpn* следующего содержания:

```
dev tap
proto udp
remote st1.kryukov.biz 1194
client
nobind
tls-client
comp-lzo
```

```
ns-cert-type server
ca "C:\\Program Files\\OpenVPN\\config\\CA.crt"
cert "C:\\Program Files\\OpenVPN\\config\\win.st1.kryukov.biz.pem"
```

Строка выше пишется в виде одной строки!

```
key "C:\\Program Files\\OpenVPN\\config\\win.st1.kryukov.biz.key"
```

Строка выше пишется в виде одной строки!

```
ping 15
ping-restart 45
ping-timer-rem
persist-key
persist-tun
verb 1
```

В составе пакета находится программа OpenVPN GUI. Она позволяет управлять соединениями.

Обратите внимание на то, что в конфигурационном файле Windows клиента, параметр *verb* равен 1. Программа OpenVPN GUI по отладочной информации, выдаваемой клиентом понимает, было ли подключение к серверу. И сможет выдавать корректную информацию об установленном соединении.

Так же существует возможность запускать OpenVPN в Windows в виде сервиса, при старте компьютера. После установки программы посмотрите список доступных сервисов.

КЛИЕНТ LINUX.

Для клиента Linux тоже потребуется создать файлы ключа и сертификата.

```
# cd /etc/pki/tls
# openssl req -days 3650 -nodes -new -keyout \
             private/lin.st1.kryukov.biz.key -out \
             certs/lin.st1.kryukov.biz.csr -extensions client
# openssl ca -days 3650 -out certs/lin.st1.kryukov.biz.pem \
             -in certs/win.st1.kryukov.biz.csr -extensions client
```

Как и в случае клиента Linux, все файлы необходимо скопировать на клиентскую машину.

На клиенте создадим две директории:

```
# mkdir /etc/openvpn/{certs,keys}
```

Соответственно в директорию */etc/openvpn/certs* поместим файлы: *CA.crt* и *lin.st1.kryukov.biz.pem*, а в */etc/openvpn/keys* — *lin.st1.kryukov.biz.key*.

Закроем доступ к файлу ключа.

```
# chown nobody:nobody /etc/openvpn/{certs,keys}
# chmod 700 /etc/openvpn/keys/lin.st1.kryukov.biz.key
# chmod 700 /etc/openvpn/keys
```

В директории /etc/openvpn создадим конфигурационный файл client.conf, следующего содержания:

```
remote st1.kryukov.biz 1194
dev tap
proto udp
ca /etc/openvpn/certs/CA.crt
cert /etc/openvpn/certs/lin.st1.kryukov.biz.pem
key /etc/openvpn/keys/lin.st1.kryukov.biz.key
client
tls-client
comp-lzo
user nobody
group nobody
ping 30
ping-restart 120
ping-timer-rem
persist-key
persist-tun
verb 0
```

Естественно в вашем случае IP адрес, порт и пути к сертификатам надо поменять.

Если клиент работает в дистрибутиве с системой инициализации SystenV, например RedHAT или CentOS, то его запуск происходит как и на сервере.

Если хотите, что бы клиент автоматически запускался при старте компьютера:

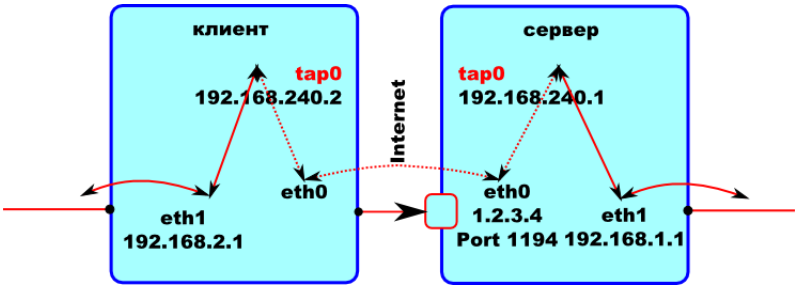
```
# chkconfig openvpn on
```

Не забудьте его после этого запустить.

```
# service openvpn start
```

КОНФИГУРАЦИЯ СЕТЬ-VPN-СЕТЬ.

Все конфигурации, которые были рассмотрены выше, показывали как соединить один компьютер с другим, через VPN. Или доступ с клиентского компьютера во внутреннюю сеть предприятия. Достаточно часто необходимо связать между собой несколько внутренних сетей предприятия, расположенных в разных филиалах.



Для того, что бы машина клиент могла передавать пакты в сеть *192.168.1.0/24*, на ней требуется явно в таблице маршрутизации добавить маршрут к этой сети через интерфейс *tap0*. Маршрут сам добавляться не будет, об этом придётся позаботиться нам. Конечно, можно явно вызывать программу *route* при каждом соединении, но это не правильный вариант.

Для решения этой задачи надо использовать параметр *push*, который позволяет передавать различные конфигурационные параметры openvpn на клиент openvpn. В нашем случае, с машины сервера (192.168.240.1) необходимо передать команду на машину клиент, что бы она добавила у себя маршрут к сети 192.168.1.0/24.

```
push "route 192.168.1.0 255.255.255.0 192.168.240.1"
```

Но это еще не все. Сеть требует взаимности и нам потребуется, что бы пакеты из сети 192.168.1.0/24 могли передаваться обратно в сеть 192.168.2.0/24. Но, к сожалению, команда *push* нам не поможет. Она работает только на сервере. Т.е. только с сервер может передавать команды, которые будут выполняться на клиенте.

Поэтому в конфигурационном файле сервера придется писать команду route для каждой сети, подключенной к клиенту.

```
route 192.168.2.0 255.255.255.0 192.168.240.2
```

Так же, что бы все работало нормально, вам придется позаботиться, что бы VPN клиенту при каждом соединении выдавался один и тот же IP адрес. Это можно сделать несколькими способами. Если учитывать конфигурацию сервера, приведенную выше. Мы добавим в конфигурационный файл сервера всего один параметр: *ifconfig-pool-persist*

При помощи этого параметра мы задаем имя файла в котором будет хранится соответствие VPN клиента и IP адреса, который ему присваивается. Этот файл сервер будет заполнять сам.

```
ifconfig-pool-persist /etc/openvpn/iplist
```

Имя файла может быть любым.

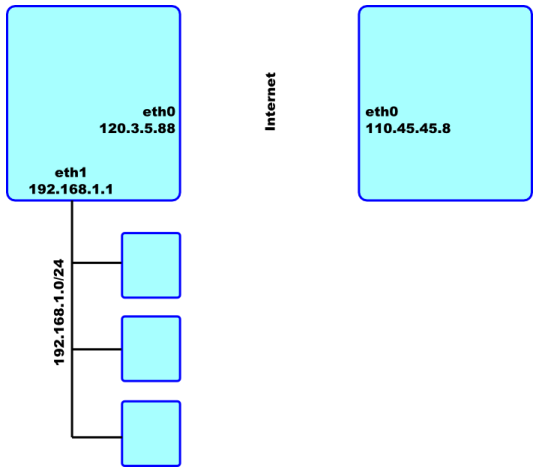
В итоге, необходимые маршруты будут автоматически добавляться на клиенте. На сервере маршрут будет присутствовать всегда. Если соединение будет разорвано, и затем восстановлено, маршруты восстановятся автоматически.

ROUTING И VPN.

После выпуска первой редакции этого учебника, у слушателей возникло много вопросов по применению VPN в реальных задачах. Эта глава была добавлена специально для того, что бы показать, как можно использовать VPN.

Рассмотрим задачу, возникшую у одного из слушателей.

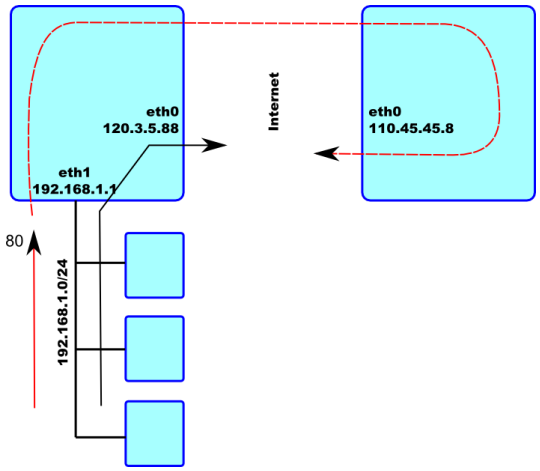
ИСХОДНЫЕ ДАННЫЕ.



Существует внутренняя сеть 192.168.1.0/24. Она выходит в Интернет через роутер с внутренним интерфейсом 192.168.1.1 и внешним интерфейсом 120.3.5.8[1].

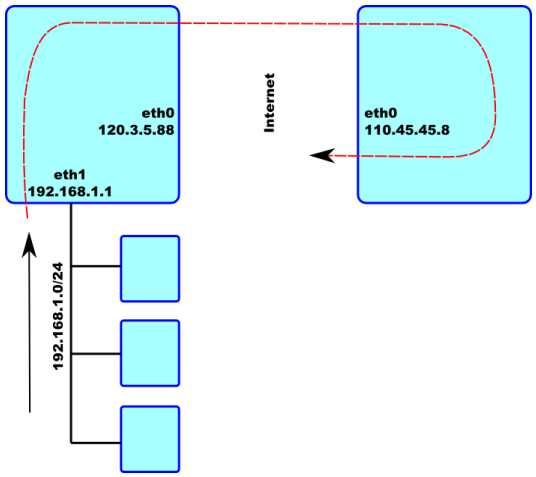
Где то «в замке у шефа» в буржуинском сегменте Интернет стоит сервер с IP 110.45.45.8, который тоже принадлежит нашей компании.

ЗАДАЧА 1.



Необходимо весь трафик, предназначенный на WEB сервера (80 порт) отправлять только через сервер 110.45.45.8. Трафик между нашей сетью и этим сервером должен быть зашифрован. Весь остальной трафик из нашей сети должен идти обычным образом. Т.е.

ЗАДАЧА 2.



Необходимо весь исходящий трафик нашей сети, предназначенный для Интернет, оправлять через роутер 110.45.45.8. Трафик между нашей сетью и этим сервером должен быть зашифрован.

Что тут сказать, параноидальные задачи, но выполнимые. Итак, будем играть в Штирлица и шифроваться.

НАСТРОЙКА VPN.

Материал, описанный в этом разделе необходим для решения обеих задач.

[1] Этот и все другие IP адреса взяты совершенно случайно и никакого отношения к реальным серверам не имеют.

Для шифрования трафика меду двумя серверами мы будем использовать Open VPN. Сервер, выводящий нашу компанию в Интернет, будет работать как VPN клиент. Сервер в Интернет всегда включен, поэтому он будет работать в качестве VPN сервера.

Open VPN настраивается так же как и в предыдущей главе, поэтому не буду повторять материалы этой главы. Просто покажу готовые конфигурационные файлы клиента и сервера.

Конфигурационный файл VPN сервера.

```
dev tap0
proto udp
mode server
comp-lzo
log-append /var/log/openvpn.log
daemon
ifconfig-pool 192.168.240.2 192.168.240.12
ifconfig 192.168.240.1 255.255.255.0
tls-server
dh /etc/openvpn/dh1024.pem
ca /etc/pki/CA/CA.crt
cert /etc/pki/tls/certs/server.pem
key /etc/pki/tls/private/server.key
port 5000
user nobody
group nobody
persist-tun
persist-key
```

verb 0

Необходимо подставить реальные файлы ключей и сертификатов.

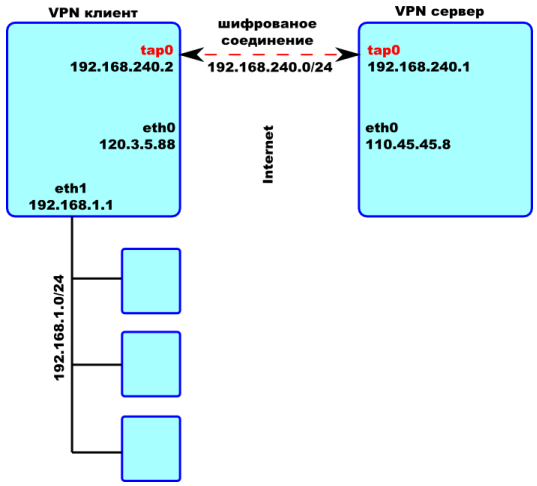
Что изменилось в файле конфигурации сервера, по сравнению с предыдущей главой? Во-первых, используется 5000 порт, а не 1194. За время, прошедшее с момента написания первой главы, создатели программы стали рекомендовать использовать порт 5000. На самом деле номер порта может быть любой, но мы будем следовать рекомендациям отцов-основателей.

Конфигурационный файл Linux клиента.

```
remote 110.45.45.8 5000
dev tap
proto udp
ca /etc/openvpn/certs/CA.crt
cert /etc/openvpn/certs/client.pem
key /etc/openvpn/keys/client.key
client
tls-client
comp-lzo
user nobody
group nobody
ping 30
```

```
ping-restart 120
ping-timer-rem
persist-key
persist-tun
verb 0
```

В этом файле мы тоже заменили порт, куда будет подключаться клиент на 5000.



Посмотрим, что получится после настройки такого решения.

Таким образом, мы обеспечили шифрование при передаче данных между первым и вторым серверами. Но этого мало! Надо сделать так, что бы VPN сервер мог доставлять пакеты в сеть 192.168.1.0/24. Поэтому в конфигурации **VPN клиента** добавим команду push, которая внесет изменения в таблицу маршрутизации VPN сервера.

```
push "route 192.168.1.0 255.255.255.0 192.168.240.2"
```

В результате конфигурационный файл клиента будет выглядеть так:

```
remote 110.45.45.8 5000
dev tap
proto udp
ca /etc/openvpn/certs/CA.crt
cert /etc/openvpn/certs/client.pem
key /etc/openvpn/keys/client.key
client
tls-client
comp-lzo
user nobody
group nobody
push "route 192.168.1.0 255.255.255.0 192.168.240.2"
ping 30
ping-restart 120
ping-timer-rem
persist-key
persist-tun
verb 0
```

Теперь внимательно приглядимся к нашей схеме. У нас получилось два сегмента внутренних сетей:

- 192.168.1.0/24
- 192.168.240.0/24

И *два*! *Целых два выхода в Интернет*! Через роутер 192.168.1.1 и роутер 192.168.240.1.

Господа, как только у вас появляется больше чем один выход в Интернет, необходимо использовать дополнительные возможности, предоставляемые ядром Linux в области маршрутизации. Эти возможности очень хорошо описаны в документе *Linux Advanced Routing & Traffic Control HOWTO*:

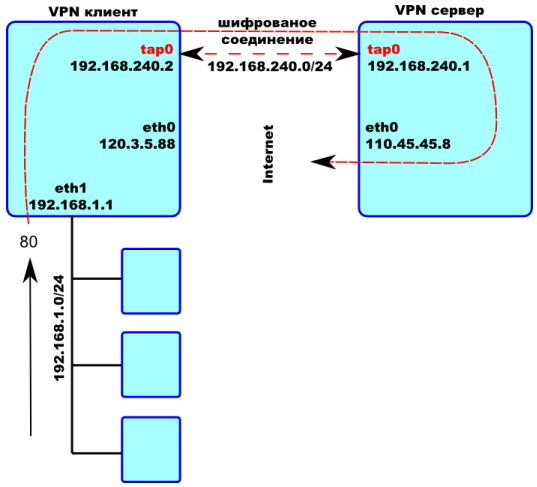
- Английский оригинал: http://tldp.org/HOWTO/Adv-Routing-HOWTO
- Русский перевод: http://opennet.ru/docs/RUS/LARTC

Настоятельно рекомендую в дальнейшее подробно изучить данный HOWTO!

А теперь займемся решением задач.

РЕШЕНИЕ ЗАДАЧИ 1.

Напомню, что требуется сделать.



Нам необходимо весь трафик на WEB сервера (порт 80) отправлять в Интернет через удаленный сервер, через интерфейс с IP адресом 110.45.45.8.

Весь остальной трафик из нашей сети должен выходит в Интернет через локальный сервер, через интерфейс с IP адресом 120.3.5.88.

На удаленном сервере нам надо сделать только NAT преобразование. Все пакеты, пришедшие из сети 192.168.1.0/24 или 192.168.240.0/24[2] пропускать через SNAT.

Поэтому в firewall удаленного сервера надо добавить следующие правила:

```
iptables -t nat -A POSTROUTING -o eth0 -s 192.168.1.0/24 \
             -j SNAT --to-source 110.45.45.8
iptables -t nat -A POSTROUTING -o eth0 -s 192.168.240.0/24 \
             -j SNAT --to-source 110.45.45.8
```

Не забудьте разрешить прохождение этих пакетов через firewall.

```
iptables -A FORWARD -p tcp --dport 80 -i tap0 -j ACCEPT
```

Не забудьте разрешить перенос пакетов с одного сетевого интерфейса на другой.

[2] А вдруг у вас на клиентском сервере работает прокси сервер Squid?

```
echo 1 > /proc/sys/net/ipv4/ip_forward
```

В файле /etc/sysctl.conf отредактируйте строку.

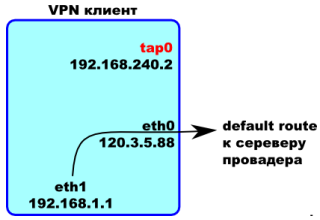
```
net.ipv4.ip_forward = 1
```

С удаленным сервером все. Займемся локальным сервером. Наша задача отправлять пакеты предназначенные на 80 порт в Интернет через машину 192.168.240.1.

Добавить маршрут в таблицу маршрутизации? Не поможет, в таблице маршрутизации нельзя указывать порты.

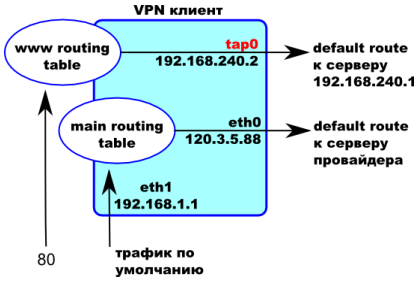
Сделать DNAT? Тоже не поможет, на нельзя менять IP назначения.

SNAT? Тоже ничего не даст.



Во всех случаях, перечисленных выше, пакет с локального сервера будет уходить по маршруту по умолчанию через интерфейс eth0.

Пришло время включить дополнительные возможности ядра Linux. А вы знаете, что в Linux можно сделать 256 таблиц маршрутизации? И 512 можно ☺!



Для решения нашей проблемы мы добавим еще одну таблицу маршрутизации.

В этой таблице сделаем маршрут по умолчанию на машину 192.168.240.1 через интерфейс tap0.

Все пакеты, приходящие на наш сервер на 80 порт из внутренней сети, а также все пакеты, которые генерируют программы, работающие на нашем сервере на 80 порт, *будем передавать в новую таблицу маршрутизации*. А там уровень IP все сделает сам.

Создадим новую таблицу маршрутизации с именем www.

```
echo 430 www >> /etc/iproute2/rt_tables
```

Таким образом, мы создали таблицу номер 430 с именем *www*. Номер и имя можно использовать любые. Главное, что бы они уже не использовались в вашей системе. Вы можете сначала посмотреть содержимое файла *rt_tables* при помощи программы *cat*.

```
cat /etc/iproute2/rt_tables
```

Добавим в таблицу маршрут по умолчанию.

```
ip route add default via 192.168.240.1 dev tap0 table www
```

Таблица создана. Как заставить необходимые нам пакеты попадать именно в неё? Тут следует сделать два действия:

1. При помощи *iptables* и действия *MARK* пометить пакет. Присвоить ему номер от 1 до 64.

2. При помощи программы *ip*, все помеченные нужным нам номером пакеты перенаправить в таблицу *www*.

```
iptables -t mangle -A PREROUTING -i eth1 -p tcp --dport 80 \
            -j MARK --set-mark 1
iptables -t mangle -A OUTPUT -p tcp --dport 80 \
            -d ! 192.168.1.0/24 -j MARK --set-mark 1
```

Обратите внимание на то, что метки мы ставим в таблице *mangle* в цепочках *PREROUTING*[3] и *OUTPUT*[4], до того, как пакеты попадут в таблицу маршрутизации.

```
ip rule add fwmark 1 table www
```

Можете посмотреть какие пакеты, каким таблицам будут передаваться.

```
ip rule show
```

Теперь займемся firewall.

Разрешаем хождение пакетов на 80 порт, через интерфейс *tap0*.

```
iptables -A FORWARD -p tcp --dport 80 -o tap0 -j ACCEPT
```

Остальными правилами разрешаете то, что должно выходить через интерфейс *eth0*. Я не буду писать эти правила. Посмотрите, как это делалось в разделе, посвященном firewall. Единственное добавление — теперь надо явно указывать output интерфейс при помощи параметра *–o*.

Последнее замечание — NAT преобразования. Нам необходимо делать NAT только на интерфейсе *eth0*. На интерфейсе *tap0* ничего делать не надо. Там пакеты идут без преобразования.

РЕШЕНИЕ ЗАДАЧИ 2.

А тут я поступлю немного хитрее. Я хочу, что бы вы сами решили эту задачу. Если все заработает, значит, вы все поняли правильно. Если не заработает, присылайте мне ваше решение. У нас будет тема для обсуждения.

Единственная подсказка — трафик на 22 порт удаленного сервера не пускайте через VPN. Если вдруг не получится, вы хоть сможете подключиться к удаленному серверу. И вторая причина — в этом соединении уже итак все зашифровано ☺.

[3] Для пакетов, пришедших из внутренней сети.
[4] Для пакетов, которые генерирует программное обеспечение, работающее на нашей машине.