TRICKSTER VPN

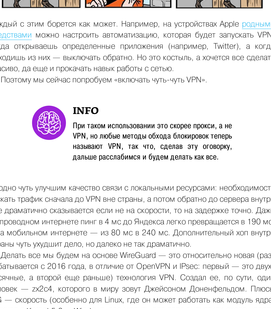
РАЗБИРАЕМСЯ С WIREGUARD И ДЕЛАЕМ СВОЙ УМНЫЙ VPN

Забавная ситуация: сайтов и сервисов, доступных только через VPN, все больше, но при этом многие российские компании закрывают доступ из-за границы. В результате приходится целыми днями теребить ползунки «вкл-выкл», что утомительно. Я расскажу, как с помощью магии маршрутов и WireGuard, решить эту проблему и сделать «умный» VPN, который не надо отключать.

vxvlad
Любопытен. Ленив. Вредин.
xakep@vxvlad.xyz

Если ты пользуешься VPN, то и сам наверняка сталкиваешься с блокировками зарубежного трафика. К примеру, могут не открываться pochta.ru, leroymerlin.ru, rt.ru, avito.ru.

Получается мем.



Каждый с этим борется как может. Например, на устройствах Apple родными средствами можно настроить автоматизацию, которая будет запускать VPN, когда открываешь определенные приложения (например, Twitter), а когда выходишь из них — выключать обратно. Но это костыль, а хочется все сделать красиво, да еще и прокачать навык работы с сотью.

Поэтому мы сейчас попробуем «включать-чуть-чуть VPN».

INFO

При таком использовании это скорее прокси, а не VPN, но любые методы обхода блокировок теперь называют VPN, так что, сделав эту оговорку, дальше расслабимся и будем делать как все.

Заодно чуть улучшим качество связи с локальными ресурсами: необходимость гаскать трафик сначала до VPN вне страны, а потом обратно до сервера внутри нее драматично оказывается если не на скорости, то на задержке точно. Даже на проводном интернете пинг в 4 мс до Яндекса легко превращается в 190 мс, а на мобильном интернете — из 80 мс в 240 мс. Дополнительный хоп внутри страны чуть ухудшит дело, но далеко не так драматично.

Делать все мы будем на основе WireGuard — это относительно новая (разрабатывается с 2016 года, в ядре он появился от OpenVPN и IPsec: первый — это двухтысячные, а второй еще раньше) технология VPN. Создал ее, по сути, один человек — zx2c4, которого в миру зовут Джейсон Доненфельдом. Плюсы WG — скорость (особенно для Linux, где он может работать как модуль ядра, начиная с Kernel 5.6, и Windows, где модуль для ядра выпустили около недели назад), низкие задержки, современная криптография и простая настройка и использование конечным хозяином.

Ах да, еще UDP. UDP для туннелей — это хорошо, потому что у TCP уже есть механизмы, которые позволяют ему работать на неидеальных соединениях, а UDP представляет собой именно такое соединение. А когда ты засовываешь TCP в TCP, то отказываешься от большей части этих механизмов (инкапсулированный TCP-пакет будет гарантированно доставлен другой стороне, хотя протокол допускает недоставку), но все еще несешь весь оверхед вида «хендшейк соединения для отправки хендшейка».

Не говоря уж о том, что инкапсулировать UDP в TCP — ниуть не лучшая идея, потому что сразу рушит все предположения всяких скайпов о том, что лучше пропустить пару пакетов, чем уменьшить задержку: каждый UDP-пакет в этом случае будет принудительно послан заново и доставлен корректно, не считаясь с затратами времени.

Особенно для одинокого пользователя-хакера приятна работа с шифрованием: нет необходимости ни в сертификатах и удостоверяющих центрах, ни в логинах-паролях, все, что нужно, — обменяться публичными ключами с пиром, с которым ты хочешь установить соединение. Для больших компаний это, конечно, будет скорее минусом, как и то, что WG — это только базовая часть полноценной большой инфраструктуры VPN. Но именно WireGuard использовали, к примеру, в Cloudflare для своего WARP, правда, написав его собственную реализацию — boringtun.

Еще один минус WG — то, что трафик не обфусцирован и глубокая инспекция пакетов (DPI) выявит и позволит заблокировать такое соединение (не говоря уж о блокировке UDP совсем, что почти не мешает работать с вебом, но гарантированно ломает WireGuard). Для скрытия трафика рекомендуется использовать специализированное ПО — Cloak, Obfsproxy, Shadowsocks, Stunnel, SoftEther, SSTP или, в конце концов, простой SSH. Часть из этих инструментов может работать совместно с WG, а часть способна его заменять в качестве инструмента стеганографии. WG изначально создавался под скорость и криптографическую защищенность.

Если очень упрощать, ключи работают следующим образом: у нас есть закрытый (приватный) ключ, из которого можно сгенерировать открытый, или публичный. Наоборот, из открытого ключа получить закрытый мы никак не можем. Затем мы шифруем с помощью закрытого ключа какую-то строку, а при помощи открытого расшифруем ее и тем самым убедимся, что у собеседника точно есть закрытый ключ, а значит, он тот, за кого себя выдает. Таким образом, мы можем без проблем передавать открытый ключ — он всего лишь позволяет проверить подлинность автора, но не притвориться им.

Это как в SSH — публичный ключ лежит на сервере, где его потеря — небольшая беда: все, что сможет сделать с ним злоумышленник, — это положить его на свой сервер, чтобы ты мог подключиться к нему с помощью закрытого ключа.

Так вот, в WG на первом этапе подключения каждая сторона с помощью зашифрованного приватным ключом сообщения доказывает собеседнику, что она именно она: это проверяется публичным ключом.

Второй этап — создание с помощью этих ключей и матана симметричных ключей для шифрования самого трафика. Благодаря тому что расшифровать зашифрованное публичным ключом нельзя без приватного, мы сможем создать ключ для симметричного шифрования и отправить его по защищенному каналу. Этот шаг необходим потому, что симметричное шифрование гораздо менее ресурсоемкая операция, а минус у нее только один: необходимо синхронизировать ключ между сторонами, при том что перехват ключа третьей стороной ведет к возможности расшифровки трафика.

Но эта проблема решается с помощью асимметричной схемы. Это называется протокол Диффи — Хеллмана — способ защищенного получения общего секретного ключа. В WG используется ECDH — вариация Диффи — Хеллмана на эллиптических кривых. Первые два звена в терминах WG называются рукопожатием или хендшейком.

Затем симметричные ключи используются для шифрования трафика. Раз в две минуты происходит новое рукопожатие и соссаючные ключи меняются.

Разумеется, в реальности все немного сложнее: например, отправляются не сами ключи, а сгенерированные на их основе эфемерные ключи, которые удаляются сразу после операции. Заинтересовавшихся подробностями отправляю к краткому описанию на сайте WireGuard.

Мы же перейдем к более практическим действиям.

ШАГ 1. СОЗДАЕМ И НАСТРАИВАЕМ ДВА СЕРВЕРА

Один сервер будет внутри страны — через него трафик пойдет на локальные ресурсы, а второй — за границей. Дальше я их буду называть **local** и **external**.

Идеально, если **local** будет в твоей домашней сети, потому что при этом трафик на внешние ресурсы не отличается от твоего домашнего трафика. Но для этого нужен какой-то хост дома, белый IP и возможность пробросить порт. У меня это виртуалка на домашнем сервере, но, наверное, подойдет и Raspberry Pi или аналогичный одноплатник.

INFO

Вариант с одноплатником я не тестировал и не уверен, что он сработает. Raspberry Pi придется маршрутизировать весь трафик с устройств и держать в памяти около 11 тысяч маршрутов; ресурсов на это может не хватить.

Если дома хоста нет, можно взять любой сервер у хостера VDS. Увы, некоторые сайты блокируют подсети хостеров, опасаясь ботов, так что это меняет предпочитательный вариант.

А вот внешнюю машину можно арендовать и у хостера. К примеру, у RU VDS и VDSina есть зарубежные площадки. А можно выбрать иностранного хостера, если найдешь способ оплачивать его услуги. Например, я использую исландский 1984.hosting, созданный специально для пару-нормиков.

Считаем, что на обоих серверах у нас Debian 11.

Ставим нужные нам пакеты:

```
apt update && apt install -y wireguard iptables ipcalc qrencode
curl ip traceroute dnsutils ufw
```

Включаем перенаправление трафика: в этом случае сервер, получив пакет, который не предназначается ни одному из его IP-адресов, не отбросит его, а попытается перенаправить в соответствии со своими маршрутами.

```
echo "net.ipv4.ip_forward=1" > > /etc/sysctl.conf
echo "net.ipv4.conf.all.forwarding=1" > > /etc/sysctl.conf
sysctl -p /etc/sysctl.conf
```

Опционально (но очень удобно) сразу поменять hostname обоих серверов, чтобы не запутаться, где какая консоль.

```
hostnamectl set-hostname trickster-internal

hostnamectl set-hostname trickster-external
```

Продолжение статьи →

TRICKSTER VPN

РЕ:ЗАРИФЕЙСТВУЙ
ОБЕЗЬЯНЫ ПРОТИВ
ЛЕНИВЫХ ЛЮДЕЙ
СВОЙ ЛИЧНЫЙ VPN

ШАГ 2. НАСТРАИВАЕМ ВЛОЖЕННЫЙ ДОМ СВЯЗИ ДВУХ СЕРВЕРОВ

Для начала генерируем ключи. Запускаем для каждого **wg genkey** и пропускаем для приватных ключей:

```
root@jrt-internet-ru:~# wg genkey
oGZ0YQ0qwgZ2Ooi2Ki6xFrZ+Tb+PEnGdbkw2Ocs2o=
root@jrt-internet-ru:~# wg genkey
MOOdnIOfLyPzhtnqMNJjUSFbkzqVvqMvb3ouzbc3=
```

Имена wg genkey не важны и можно использовать, это просто значит что в таком дом

```
echo ENABOOM | mailcow | head -c 32 | base64
```

Только навсегда более сложные: что часто генерируется 32 байта случайных значений и представляется на base64.

Создаем для клиента Core-ru **internal**

```
/ip/wireguard/wg-internal
[Interface]
Address = 10.20.30.2/32
listenPort = 17960
PrivateKey = oGZ0YQ0qwgZ2Ooi2Ki6xFrZ+Tb+PEnGdbkw2Ocs2o=
/ip/route/add dst-address=0.0.0.0/0 gateway=wg-internal routing-table=wg-internal
/ip/firewall/nat/add chain=srcnat out-interface=wg-internal action=masquerade
/ip/firewall/mangle/add chain=forward protocol=tcp tcp-flags=syn tcp-mss=1361-65535 action=change-mss new-mss=1360
/ip/firewall/filter/add chain=input protocol=udp dst-port=17960 action=accept
/ip/firewall/filter/add chain=input in-interface=wg-internal action=accept
/ip/firewall/filter/add chain=forward in-interface=wg-internal action=accept
```

Отдаем **external**

```
/ip/wireguard/wg-external
[Interface]
Address = 10.20.30.2/32
PrivateKey = MOOdnIOfLyPzhtnqMNJjUSFbkzqVvqMvb3ouzbc3=
Peering = ipv4dns.ru:7443 & POSTROUTING -m ip route | awk '/default/ {print $5}' | head -1 & MASQUERADE
PostUp = ipv4dns.ru:7443 & POSTROUTING -m ip route | awk '/default/ {print $5}' | head -1 & MASQUERADE
```

Создаем **[Interface]** — это настройки конкретного интерфейса интерфейса WireGuard, есть у каждого сервера. Ключ **Address** — это адрес нашего туннеля, в данном туннеле. Сервер использует. Сервер интерфейса всегда одна хосте. Ключ **ListenPort** — порт для подключения клиентов. Ключ **PrivateKey** — наш ключ. Ключ **Peering** — это настройка, которую мы добавляем и расширяет, когда клиенты подключаются к серверу. Ключ **PostUp** — выполняется при поднятии интерфейса, полезно можно, расширяющие данные маскировки клиентов/сервер. Дорабатывает выполняет дополнительные правила firewall, которые не поддерживаются стандартным, например SOCKS, который Microsoft также можно и только сервер. Ключ **PostDown** — выполняется при опускании интерфейса. Ключ **Table** — Таблица маршрутов. Ключ **Address** — адрес. Это нужно для маршрутизации внутри сетевого интерфейса, куда по умолчанию выполняется маскировка.

```
ip route | awk '/default/ {print $5}' exit\"
```

Здесь проверяем, интерфейс, обращаемся к проверяем интерфейсу.

Таким образом, сложная конструкция превращается в такую:

```
ipv4dns.ru:7443 & POSTROUTING -m ip route | awk '/default/ {print $5}' | head -1 & MASQUERADE
```

Здесь проверяем значение NAT в данном маршруте, сервер будет отрицать, прошедшую ему пакеты на внешнюю сеть, сохранение адрес отправителя пакета, чтобы ответы на пакеты приходили к нам, а не к серверу. Ключ **Table** — Таблица маршрутов.

Таким образом уже начинаем «сплетать» сервер, в данном случае 192.168.83.2:

```
/ip/route/add dst-address=192.168.83.2/32 table=wg-internal
```

Это необходимо для сервера **internal** потому что иначе при запросе маршрута **0.0.0.0/0** он начинает пересылать ответы на клиенты, пересылает ему на внешнюю адресную сеть. Поэтому, чтобы ответы на клиенты приходили к нам, нам нужно сделать, что для них на стороне отправить пакеты по маршруту. Ключ **table=wg-internal** — таблица, которая мы создаем.

Естественно, так начинаем ты. Мы такие отбрасываем. В этом случае мы отбрасываем, что пакеты, которые приходят на сервер, не отправляются на внутренние пакеты по внешнему. Ответы. Ответы. Ответы

TRICKSTER VPN

РАЗВЕРТЫВАЕМ СЕРВЕРЫ И ДЕЛАЕМ СВОЙ ЯНДЕКС VPN

ШАГ 4. ДОБАВЛЯЕМ РЕГИОНАЛЬНО ЗАВИСИМУЮ МАРШРУТИЗАЦИЮ

Как бы понятно, что отправлять все данные с клиента в **internal** он все равно шлет через **external**. И тот шлет данные наружу. Но есть нюанс. Если у нас на **internal** обнаружен «забугор» 0.0.0.0/0, который перебивается любым другим маршрутом, а где **internal** находится прямой сетевой сегмент, то тут, мол, кто-нибудь — это как-то перевешать, запрос на расширение IP на сервере **internal** и перенаправить, но не в сумму 0.0.0.0/0 «забугор» **external** — ремиум в сетевой карт самого сервера, а тот, через который по топуару дуется в интернет. Ну, в общем, по-русски это называется — давайте к нашим баранам.

Давай проверим. Подключаемся. Чем клиентов полчаем IP того же Сбербанка, ставим сервер **internal** и посмотрим, что куда идет. Смотрим:

```
traceroute 212.193.158.175
```

```
traceroute to 212.193.158.175 (212.193.158.175), 30 hops max, 60 byte packets
 1  10.20.30.1  0.485 ms  0.400 ms  0.395 ms
 2  *  *  *
 3  *  *  *
 4  *  *  *
 5  *  *  *
 6  *  *  *
 7  *  *  *
 8  *  *  *
 9  *  *  *
10  *  *  *
11  *  *  *
12  *  *  *
13  *  *  *
14  *  *  *
15  *  *  *
16  *  *  *
17  *  *  *
18  *  *  *
19  *  *  *
20  *  *  *
21  *  *  *
22  *  *  *
23  *  *  *
24  *  *  *
25  *  *  *
26  *  *  *
27  *  *  *
28  *  *  *
29  *  *  *
30  *  *  *
```

А-а, как и ожидалось, чаем **external**.

Теперь сделаем маршрут для этого адреса через дефолтный шлюз и устройство. Что можно сделать в **up**:

```
route add -net 212.193.158.175/32 gateway 10.20.30.2 dev wg-internal scope link
```

Да, на посмотрим могут и чаем маршрут.

Теперь, посмотрим конкрет **traceroute** от 212.193.158.175 на чаем и видим, что пошел другой:

```
traceroute 212.193.158.175
```

```
traceroute to 212.193.158.175 (212.193.158.175), 30 hops max, 60 byte packets
 1  10.20.30.1  0.485 ms  0.400 ms  0.395 ms
 2  *  *  *
 3  *  *  *
 4  *  *  *
 5  *  *  *
 6  *  *  *
 7  *  *  *
 8  *  *  *
 9  *  *  *
10  *  *  *
11  *  *  *
12  *  *  *
13  *  *  *
14  *  *  *
15  *  *  *
16  *  *  *
17  *  *  *
18  *  *  *
19  *  *  *
20  *  *  *
21  *  *  *
22  *  *  *
23  *  *  *
24  *  *  *
25  *  *  *
26  *  *  *
27  *  *  *
28  *  *  *
29  *  *  *
30  *  *  *
```

А-а, как и ожидалось, чаем **external**.

Теперь сделаем маршрут для этого адреса через дефолтный шлюз и устройство. Что можно сделать в **up**:

```
route add -net 212.193.158.175/32 gateway 10.20.30.2 dev wg-internal scope link
```

Да, на посмотрим могут и чаем маршрут.

Теперь, посмотрим конкрет **traceroute** от 212.193.158.175 на чаем и видим, что пошел другой:

```
traceroute 212.193.158.175
```

```
traceroute to 212.193.158.175 (212.193.158.175), 30 hops max, 60 byte packets
 1  10.20.30.1  0.485 ms  0.400 ms  0.395 ms
 2  *  *  *
 3  *  *  *
 4  *  *  *
 5  *  *  *
 6  *  *  *
 7  *  *  *
 8  *  *  *
 9  *  *  *
10  *  *  *
11  *  *  *
12  *  *  *
13  *  *  *
14  *  *  *
15  *  *  *
16  *  *  *
17  *  *  *
18  *  *  *
19  *  *  *
20  *  *  *
21  *  *  *
22  *  *  *
23  *  *  *
24  *  *  *
25  *  *  *
26  *  *  *
27  *  *  *
28  *  *  *
29  *  *  *
30  *  *  *
```

Но, Обращает внимание! Первая, такое ощущение что основе сервера **internal** на самом деле сервер **external**. Ну, в общем, такое ощущение, что у второго сервера в реальном мире домена и до которого тот маршрут.

Можно сказать — это **internal** — ну, мать перемать — ничего страшного, пускать клиентов, адреса в AD — сервер, очень подвержен этой аналогичным трюкам (у Сбербанка, к слову, ПОЛНЫЙ ОСОО Криптовые технологии). С большой вероятностью для более менее серьезных клиентов это и будет за сетевым оборудованием. Ну, в общем, ставим сервер **internal** — ставим сервер **external** — ставим сервер **internal** — ставим сервер **external** — ставим сервер **internal** — ставим сервер **external** —