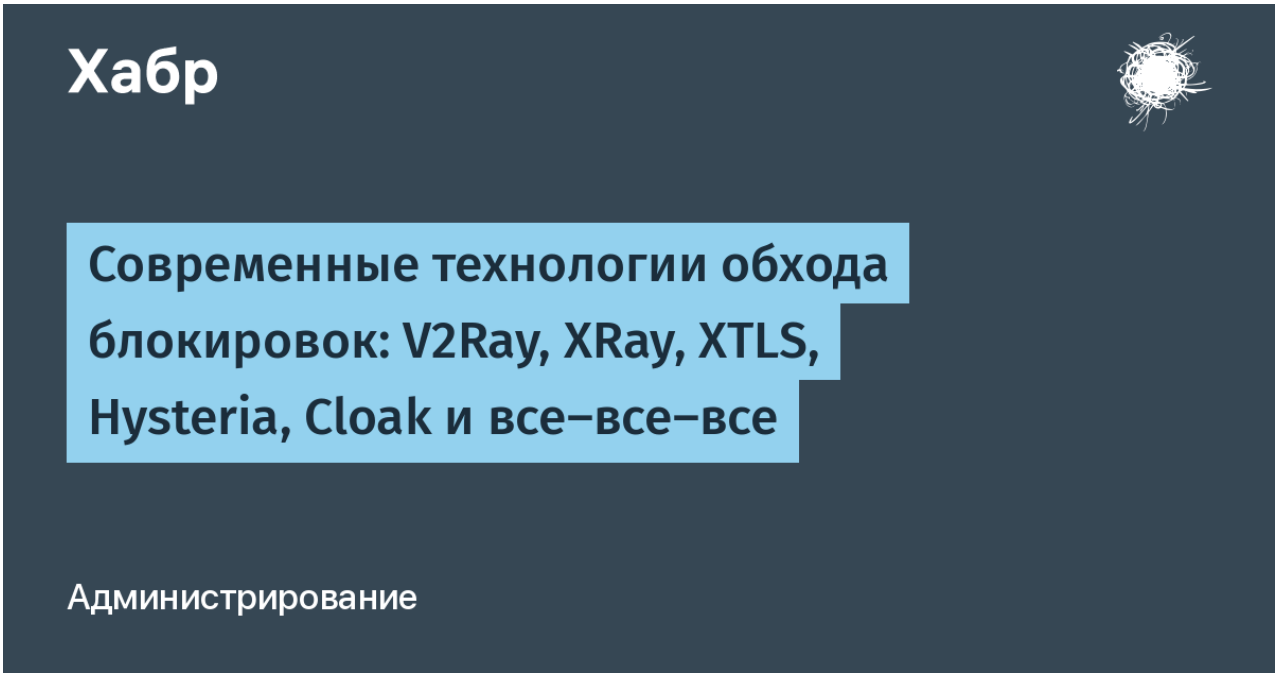
Современные технологии обхода блокировок: V2Ray, XRay, XTLS, Hysteria, Cloak и все-все-все

habr.com/ru/articles/727868

Deleted user April 9, 2023



Deleted-user 10 апр 2023 в 04:15
20 мин

442K

Три месяца назад здесь на Хабре я опубликовал статью “Интернет-цензура и обход блокировок: не время расслабляться”, в которой простыми примерами показывалось, что практически все популярные у нашего населения для обхода блокировок VPN- и прокси-протоколы, такие как Wireguard, L2TP/IPSec, и даже SoftEther VPN, SSTP и туннель-через-SSH, могут быть довольно легко детектированы цензорами и заблокированы при должном желании. На фоне слухов о том, что Роскомнадзор активно обменивается опытом блокировок с коллегами из Китая и блокировках популярных VPN-сервисов, у многих людей стали возникать вопросы, что же делать и какие технологии использовать для получения надежного нефильтрованного доступа в глобальный интернет.

Мировым лидером в области интернет-цензуры является Китай, поэтому имеет смысл обратить наш взор на технологии, которые разработали энтузиасты из Китая и других стран для борьбы с GFW (“великим китайским файрволом”). Правда, для неподготовленного пользователя это может оказаться нетривиальной задачей: существует огромное количество программ и протоколов с похожими названиями и

с разными не всегда совместимыми между собой версиями, огромное количество опций, плагинов, серверов и клиентов для них, хоть какая-то нормальная документация существует нередко только на китайском языке, на английском - куцая и устаревшая, а на русском ее нет вообще.

Поэтому сейчас мы попробуем разобраться, что же это все такое, как это использовать, и при этом не сойти с ума.

В этой статье я проведу обзор самых передовых протоколов и технологий, которые:

- позволяют делать передаваемый трафик не похожим вообще ни на один существующий стандартный протокол, делая его полностью неразличимым для цензоров;
- либо наоборот, позволяют максимально достоверно маскироваться под безобидный HTTPS-трафик, включая защиту сервера от детектирования методом active probing с помощью фейкового веб-сайта, маскировку TLS fingerprint клиента под обычный браузер, и защиту от выявления туннеля нейросетями (детектирования TLS-inside-TLS);
- позволяют работать в условиях жесткого шейпинга канала и потерь пакетов;
- позволяют создавать цепочки из серверов и настраивать маршруты (например, фильтровать трафик до российских адресов).

Итак, поехали.

Shadowsocks, ShadowsocksR, Shadowsocks-AEAD, Shadowsocks-2022

Начнем, по традиции, с “дедушки”, прародителя многих других современных средств обхода блокировок - протокола Shadowsocks.

Логотип ShadowSocks

Идея Shadowsocks проста: авторы взяли классический SOCKS-протокол, который передает все данные в открытом виде и поэтому очень легко детектируется на DPI, прикрутили к нему шифрование разными алгоритмами, выкинули ненужный функционал (например, нет нужды в авторизации по логину и паролю, проверка свой/чужой определяется ключом шифрования), и добавили несколько других штук

для усложнения детектирования. И это сработало - долгое время Shadowsocks был излюбленным инструментом тысяч людей, позволяющим пробиваться через великий китайский файервол.

Оригинальный Shadowsocks был разработан программистом с ником “clowwindy”. В 2015 году clowwindy написал в своем Github, что к нему нагрянула китайская полиция и сделала предложение, от которого не было возможности отказаться, и в результате чего он был вынужден прекратить работу над проектом и удалить все исходники из репозитория.

После этого другие энтузиасты создали форк под названием ShadowsocksR и продолжили дело. Через некоторое время разработка ShadowsocksR заглохла, но развитие протокола продолжилось в разных других репозиториях под оригинальным названием. В изначальном протоколе ShadowSocks исследователи обнаружили ряд уязвимостей, позволявших его индентификацию и блокировку (например, с помощью replay-атак), поэтому в 2017 году появился Shadowsocks-AEAD с измененным алгоритмом аутентификации, а в прошлом году была выпущена новая версия протокола под названием Shadowsocks-2022, в которой авторы продолжили работу по улучшению устойчивости протокола к блокировкам. Все эти версии между собой не совместимы.

Со стороны цензоров подключение через Shadowsocks, если вы не используете какие-либо дополнительные расширения для маскировки под TLS (**Shadow-TLS**) или Websockets, выглядит как непонятное нечто - просто не похожий ни на что поток данных. Старые версии Shadowsocks уже давно не считаются надежными и устойчивыми к выявлению, однако современные версии протокола до недавних пор вполне себе могли использоваться как средство обхода блокировок в случае если цензоры спокойно относятся к “неопределенным” протоколам. В конце 2022 года группа исследователей под названием GFW-report опубликовала отчет о том, что цензоры научились выявлять подобные “неопределенные” протоколы по… отношению количества 0 и 1 битов в потоке данных. Ими была выпущена специально пропатченная версия shadowsocks, однако во-первых пропатченные клиент и сервер не совместимы с “обычными версиями”, а во-вторых патч подходит только для старых версий протокола, но не для Shadowsocks-2022 (авторы сказали, что работают над этим). Из сторонних клиентов поддержка этого хака под названием ReducedIvHeadEntropy есть только в SagerNet и V2Ray и отсутствует практически во всех GUI-клиентах.

Оригинальный Shadowsocks был написан на C с использованием библиотеки libev. Данная версия более не развивается, основная актуальная на сегодняшний день реализация написана на Rust. Между тем, протоколы Shadowsocks разных версий поддерживаются в том числе и в других клиентах и серверах (таких как V2Ray, XRay, SagerNet, Sing-box, и т.д.), о которых речь пойдет позже, поэтому Shadowsocks вполне можно рассматривать как запасной вариант, активировав его на одном сервере с другими протоколами.

V2Ray, V2Fly, XRay (VMess, VLESS, XTLS)

Все началось с проекта под названием **V2Ray**, автором которого была Victoria Raymond (отсюда, видимо, и появилось название). Достоверно неизвестно, существовал ли в реальности человек с такими именем, или это чья-то виртуальная личность, но в итоге случилось следущее: в один момент Victoria Raymond перестала выходить на связь что на Github, что в Twitter, что где-либо еще (ничего не напоминает, правда?).

Логотип Project V

В результате остальные контрибьюторы проекта, не имея административного доступа к Git-репозиториям и веб-сайту, были вынуждены форкнуть его под названием **V2Fly** для того чтобы продолжить разработку. Грубо говоря, если вы видите github-юзера или веб-сайт с названием V2Ray - весьма вероятно, что там содержится старый код и устаревшая информация, а вот с названием V2Fly - это уже нечто гораздо более актуальное. Между тем, многие люди (и даже сами разработчики!) по-прежнему продолжают называть V2Fly как V2Ray, бинарники и пакеты по-прежнему называются v2ray-core, что добавляет немного путаницы.

XRay - это форк V2Fly, когда некоторые разработчики из-за ряда разногласий с остальным сообществом (где-то я встречал упоминания что разногласия были по технической части, где-то же написано что из-за лицензий) ушли из проекта V2Fly и продолжили развивать код параллельно под названием XRay, придумав ему слоган “Penetrates everything”, что очень недалеко от правды. Формат конфигурационных файлов остался прежним, но при этом новая реализация считается более эффективной в плане производительности, а самое главное - разработчики добавили туда несколько очень крутых фич, направленных в том числе на снижение детектируемости подключений на DPI (например, с помощью выявления TLS-in-TLS), таких как XTLS, речь о которых пойдет ниже.

V2Ray/XRay - это не протокол, а, можно сказать, фреймворк - разные протоколы с разными транспортами и расширениями ~~под одной крышей~~ в одном приложении. Идея простая: что клиент, что сервер - это один бинарник. В конфигурации задаются inbounds (обработчики входящих подключений) и outbound (обработчики исходящих подключений).

На клиенте inbound обычно будет работать как HTTP- или SOCKS-прокси сервер, принимая подключения от браузеров и других программ, а outbound будет настроен как клиент какого-нибудь прокси-протокола для подключения к удаленному серверу.

На сервере все наоборот, inbound - это сервер какого-нибудь протокола (их может быть несколько одновременно с разными вариантами), а outbound - это, например “freedom” (выход в чистый интернет), “blackhole” (блокировка исходящих подключений, если вам, например, нужно ограничить доступ в зависимости от каких-то правил), или следущий прокси в цепочке, и т.д.

Для каждого из используемых протоколов можно задать также тип транспорта, например, просто TCP, либо TLS, либо Websockets, либо еще что, и таким образом создавать самые разнообразные комбинации и варианты.

Для связи inbounds и outbouds можно задавать всевозможные правила маршрутизации. Например, уже на клиенте можно автоматически отправлять все запросы к доменам “.ru” и российским IP-адресам согласно базе GeoIP на outbound “freedom” (прямой доступ к интернет без прокси), а все остальное проксировать на удаленный сервер. Или наоборот, по умолчанию отправять все на freedom, а проксировать только адреса и домены из списка (в том числе с масками и регулярными выражениями). Можно использовать разные прокси и протоколы в зависимости от типа подключения (TCP или UDP), в зависимости от порта назначения (например, перехватывать DNS-запросы на 53-ий порт, и т.д.). Можно строить цепочки из серверов - приняли подключение на одном прокси-сервере, передали его дальше на следущий, и т.д. Короче говоря, штука получилась очень гибкая и фунциональная.

Непосредственно классических протоколов в V2Ray и XRay всего два с половиной: VMess, VLESS и VLite (это та самая половина).

VMess - самый первый и самый старый. Поддерживат определение свой/чужой по ID пользователя и опционально шифрование данных.

В качестве ID-пользователя выступает UUID и (в оригинальной реализации VMess) специальное число под названием alterId. Если эти данные совпадают на клиенте и на сервере - подключение устанавливается, если нет - извините :) В конфигурации сервера может быть определено сразу много пользователей. Не буду детально углублятся в то, что такое alterId, скажу просто - это значение могло быть в принципе любым (обычно от 1 до 64), главное что оно должно было совпадать на клиенте и сервере, и изначально было нужно для механизма повышения надежности протокола. Со временем выяснилось, что механизм аутентификации оригинального VMess уязвим к ряду атак, в итоге разработчики выпустили новый вариант протокола с переделанным алгоритмом проверки пользователя, который активировался при выставлении значения alterId в **0**. То есть в наше время alterId по сути дела не используется, благо практически все серверы и клиенты умеют в новый вариант протокола.

В настоящее время VMess считается устаревшим, а при работе через просто TCP - небезопасным, однако вариант VMess-over-Websockets-over-TLS по-прежнему вполне себе жизнеспособен и может использоваться при отсутствии поддерживаемых в каком-либо клиенте альтернатив.

VLESS (как отметили в комментах, именно так, большими буквами) - это более новый протокол. В отличие от VMess он не предусматривает механизма шифрования (подразумевается, что шифрование должно производиться нижележащим транспортным протоколом, например TLS), а только проверку “свой/чужой” и паддинг данных (изменение размеров пакетов для затруднения детектирования паттернов траффика). В протоколе исправлен ряд уязвимостей старого VMess, и он активно развивается - например, автор планирует добавить поддержку компрессии алгоритмом Zstd - не сколько для производительности, сколько для затруднения анализа “снаружи”. При этом, при установлении соединения (хендшейке) клиент и сервер обмениваются версией протокола и списком поддерживаемых фич, то есть при дальнейшем развитии должна сохраняться обратная совместимость. В общем и целом, на сегодняшний день это самый свежий и прогрессивный протокол.

Обратите внимание: то, что VLESS не предусматривает шифрования на уровне протокола, не значит, что данные передаются в нешифрованном виде. VLESS всегда работает поверх TLS, трафик шифруется именно механизмами TLS, а не самого VLESS. Никакой проблемы с безопасностью тут нет, все секьюрно :)

VLite есть только в V2Ray (в XRay его нет), поддерживает только передачу UDP-пакетов, и максимально оптимизирован именно для этого, что может быть полезно, например, для онлайн игр, но параллельно придется настроить еще VMess/VLESS для TCP - поэтому я считаю его только “половиной” :)

Кроме VMess и VLESS сервера и клиенты V2Ray и XRay также поддерживают протокол Shadowsocks (в том числе версий AEAD и 2022) о котором я говорил выше, а также Trojan, о котором речь пойдет в следущей главе.

С протоколами закончили, перейдем к транспортам. VLESS, VMess и другие могут работать, скажем так, разным образом. Самый простой вариант - обычный TCP-транспорт. VMess+TCP в данном случае очень похож на Shadowsocks, а VLESS+TCP не имеет смысла (из-за отсутствия шифрования). Более интересный вариант - TLS-транспорт, когда устанавливается обычное TLS-подключение (как и в случае с любыми HTTPS-сайтами), а уже внутри этого зашифрованного соединения работает протокол. V2Ray и XRay умеют также работать поверх mKCP (о нем будет в следущих главах), QUIC (aka HTTP/3, правда в России его массово блокируют и смысла в нем мало), gRPC, и самое интересное - через Websockets.

Вариант с Websockets очень ценен тем, что:

1. Позволяет легко поставить V2Ray/XRay не перед, а за Nginx/Caddy/любымдругимвебсервером;

2. Позволяет пролезать через строгие корпоративные фаерволы;
3. Добавляет дополнительный уровень защиты (не зная URI невозможно достучаться до прокси-сервера);
4. И самое интересное - позволяет работать через CDN (upd.: gRPC тоже позволяет).

На последнем пункте остановимся чуть подробнее. Некоторые CDN, в том числе и имеющие бесплатные тарифы, такие как Cloudflare и GCore, разрешают проксирование веб-сокетов даже на бесплатных тарифах. Таким образом, это может быть хорошим подспорьем - если по какой-то причине IP-адрес вашего сервера попал в бан, вы все равно можете подключиться к нему через CDN, а полный бан всей CDN гораздо менее вероятен, чем какого-то одного VPS. А еще Cloudflare (возможно и GCore тоже, не уточнял) умеет проксировать IPv4 запросы на IPv6 адрес, то есть свой прокси-сервер вы можете поднять даже на копеечном (можно найти варианты за 60 центов в месяц!) IPv6-only или NAT VPS без IPv4 адреса, и наплодить таких серверов чуть ли не десяток в разных локациях :)

Недостатком транспорта через веб-сокеты является более долгий хендшейк (установление каждого соединения) чем напрямую через TLS. Но и здесь есть решение.

Сервера XRay и Sing-Box (возможно и V2Ray тоже, не проверял) позволяют задавать также механизм fallaback'ов для разных протоколов. Например, при подключении пользователя первым делом сервер пытается обработать входящее подключение как VLESS-over-TCP. Если хендшейк оказался успешным, пользователь опознан - работаем, если нет - передаем следущему обработчику. Следущий обработчик, может, например, попытаться воспринять это новое подключение как VMess-over-Websocket. Если сработало - отлично, если нет - то передаем подключение следущему inbound'у. А тот, в свою очередь, не разбираясь, перенаправляет подключение на локальный веб-сервер с котиками. Таким образом у нас есть возможность одновременно принимать подключения и через VLESS-TCP, и через VLESS-Websockets или VMess-Websockets на одном порту, а если не сработал ни один из вариантов, прикидываться безобидным веб-сайтом.

Еще одна фича V2Ray и XRay - мультиплексирование соединений (**mux** или **mux.cool**). В этом случае на каждое новое подключение к какому-либо сайту не будет устанавливаться новое подключение к прокси, а будут переиспользованы существующие. Что в теории может ускорить хендшейк и привлекать меньше внимания со стороны цензоров (меньше параллельных подключений к одному хосту), с другой стороны снижает скорость передачи данных из-за оверхеда на дополнительные заголовки пакетов.

XUDP и **Packet** - расширения VLESS для более эффективной передчи UDP-пакетов и реализации Full Cone NAT. Packet - версия подревнее, XUDP по-новее. Без их использования многие NAT-тесты будут жаловаться на кривой NAT ("endpoint

address not changed), а с XUDP вы получаете нормальный честный Full Cone. Это может быть полезно для онлайн-игр, месседжеров и разного софта с передачей аудио и видео. XUDP и Packet нельзя использовать одновременно с MUX из прошлого параграфа из-за особенностей реализации (авторы старались впихать все в рамки существующего протокола и сохранить обратную совместимость, поэтому были вынуждены переиспользовать некоторые механизмы).

А теперь про самое интересные фичи.

uTLS предназначена для обмана механизма детектирования на основе TLS fingerprint, о котором я рассказывал в прошлой статье. Почитать про TLS fingerprint можно на посвященном ему сайте. В Китае и Иране цензоры активно используют этот механизм для детектирования прокси-клиентов - если мы обращаемся к какому-нибудь прокси, замаскированному под HTTPS-сайт, но при этом TLS fingerprint клиента отличается от популярных браузеров (особенно если клиент написан на Go, у которого очень специфичный фингерпринт), то соединение блокируется. uTLS - это специально пропатченный вариант стандартной TLS-библиотеки Go, позволяющий маскироваться под другие приложения. Некоторые клиенты дают выбор из нескольких вариантов (например chrome, firefox, safari), некоторые позволяют выбирать желаемый fingerprint вплоть до версии конкретного браузера.
В нынешних реалиях uTLS является очень крутой и почти что жизненно необходимой штукой (РКН пока что по фингепринтам не блочит, но как показывает опыт других стран, может начать в любой момент), поэтому рекомендуется его использовать во всех случаях, если он поддерживается клиентом (а если не поддерживается - лучше выбрать клиент, который поддерживает).

И наконец, **XTLS**, фирменная фишка XRay.

Сегодня почти что все веб-сайты работают не через голый HTTP, а через HTTPS (TLS). Используя прокси с TLS мы, по сути дела, еще раз шифруем уже зашифрованные данные. Во-первых это неэффективно, а во-вторых, что гораздо хуже - китайские цензоры научились определять TLS-inside-TLS (возможно с помощью нейросетей). Авторы XRay посмотрели на это, и решили: зачем шифровать то, что уже зашифровано? И придумали XTLS.

Суть проста: прокси-сервер подслушивает передаваемый трафик, и если видит, что если между клиентом (например, браузером) и удаленным хостом (например веб-сервером) устанавливается TLS-соединение, то дожидается окончания хендшейка, и после чего перестает шифровать трафик, начиная передавать пакеты данных “как есть”. В итоге существенно снижается нагрузка на прокси-сервер и клиент, и что важнее - со стороны трафик выглядит гораздо менее подозрительно (у нас подключение по TLS, поэтому до сервера бегают простые TLS-пакеты без аномалий, никакого двойного шифрования).

XTLS имеет несколько разных версий, которые отличаются алгоритмами работы, xtls-rprx-origin и xtls-rprx-direct - самые первые из них, в xtls-rprx-splice задействован механизм ядра Linux splice для более эффективного копирования данных между сокетами. Все они уже не актуальны, в настоящее время рекомендуется использовать последнюю версию **XTLS-Vision** (xtls-rprx-vision), подробное описание работы которой можно прочитать здесь. По ряду сообщений, на сегодняшний день связка VLESS+XTLS-Vision является единственной, которую пока еще не умеет эффективно блокировать китайский GFW (при условии соблюдения рядя важных моментов, например, запрета доступа к китайским сайтам через прокси). Единственный минус - **xtls-rprx-vision** пока что поддерживается не всеми клиентами, и XTLS по понятным причинам не работает через CDN.

Хозяйке на заметку: в отличие от предыдущих версий, при настройке клиентов для xtls-rprx-vision нужно выбирать тип транспорта не “XTLS”, а просто “TLS”. Нелогично, видимо связано с особенностями реализации, но такова жизнь.

И наконец, заглянем в завтрашний день: **XTLS-Reality**. Это самое новое изобретение от авторов XRay. Он уже поддерживается в master-ветке xray и даже в некоторых клиентах, но про него все еще мало что известно. В отличие от всех остальных вариантов, определение “свой/чужой” здесь происходит еще на этапе TLS-хендшейка в момент чтения ClientHello. Если клиент опознан как “свой”, сервер работает как прокси, а если нет - вжух! - и TLS подключение передается на какой-нибудь другой абсолютно реальный хост с TLS (например, google.com или gosuslugi.ru), и таким образом клиент (или цензор, желающий методом active probing проверить, а что же прячется на том конце) получит настоящий TLS-сертификат от google.com или gosuslugi.ru и настоящие данные с этого сервера. Полное соответствие. Определение "свой-чужой" происходит по значения некоторых полей пакетов TLS-хендшейшка, которые формально должны быть рандомные, а по факту генерируются специальным образом, но не зная исходного "секрета", который использовался при их генерации, невозможно определить, действительно ли это рандом или нет - соответственно для прокси этот механизм позволяет достоверно определить подлинность клиента, но вместе с тем не вызывать подозрения у цензоров и быть устойчивым к replay-атакам. Вероятно за этим будущее :)

Upd:
«Bleeding-edge обход блокировок: настраиваем сервер и клиент XRay с XTLS-Reality быстро и просто»

Trojan-GFW и Trojan-Go

Наряду с Shadowsocks и V2Ray протокол Trojan является одним из первых и популярных способов обхода блокировок в Китае, и по принципу работы в принципе соответствует своему названию :) Для стороннего наблюдателя работа через него выглядит как подключение к обычному веб-серверу, но на самом деле это веб-сервер с ~~подвохом~~ секретом (аки троянский конь).

Trojan работает поверх TLS (точно так же как HTTPS). После установления TLS-сессии сервер ожидает хендшейк в специальном формате, одним из полей которого является хеш секретного ключа. Если сообщение и ключ корректны - дальше сервер работает как прокси, если нет - запрос передается на стоящий рядом веб-сервер, и таким образом имитируется работа безобидного сайта через HTTPS.

Trojan-GFW - оригинальная версия, написанная на C++. Trojan-Go - продолжение проекта, теперь уже на языке Go . Trojan поддерживается многими мультипротокольными клиентамии серверами типа Sing-box и V2Ray/XRay - в этом случае вместе с Trojan также можно использовать упомянутые выше фичи uTLS и XTLS, что повышает надежность протокола и уменьшает вероятность его детектирования.

Если вы внимательно читали до этого, то Вам сразу же станет понятно, что Trojan в принципе аналогичен VLESS+TLS с настроенным fallback на веб-сайт. Каких-либо явных преимуществ перед VLESS+TLS у Trojan лично я не вижу, можно относиться к нему как к еще одной альтернативе.

Naiveproxy

Идея Naiveproxy, опять же, простая до невозможности. Если наша цель - замаскировать трафик от прокси-клиента так, чтобы он был вообще ничем неотличим от трафика от обычного браузера - почему бы не использовать для этого сам браузер?

Именно так и рассудил автор Naiveproxy и сделал следущее: взял исходники браузера Chromium, оторвал оттуда код сетевого стека, и использовал его в своем прокси-клиенте, причем в качестве прокси-протокола используется самый обычный метод CONNECT + HTTP/2.

В итоге одним выстрелом убивается сразу несколько зайцев:

- TLS finerprint и вообще поведение такого подключения полностью до мельчайших деталей соответствует настоящему браузеру Chromium - более того, автор периодически синхронизируется с кодовой базой Chromium, чтобы иметь самые новые версии его сетевого стека, и таким образом максимально соответствовать свежим версиям браузера;
- Определение паттернов трафика, характерных для определенных веб-сайтов, затрудняется благодаря HTTP/2 мультиплексированию;
- Определение свой/чужой, чтобы не демаскировать прокси при active probing осуществляется посредством стандартных HTTP-заголовков (“Proxy-Authorization”). Если там содержатся правильные данные - ларчик открывается, если нет, либо же заголовки отсутствуют - сервер делает вид, что не понимает, что от него хотят и выдает фейковый сайт.

В теории, на “той стороне” в качестве прокси-сервера может выступать вообще любой сервер, поддерживающий метод CONNECT (например, tinyproxy), а авторизацию “свой-чужой” можно сделать с помощью reverse-proxy такого как HAProxy. Однако гораздо лучше использовать реализации, знающие про особенности naiveproxy - в таком случае в пакеты данных также добавляется padding (грубо говоря, мусорные данные, не несущие смысловой нагрузки) для усложнения анализа паттернов трафика. Это может быть, например, сам naiveproxy на сервере, или же патченный плагин для известного веб-сервера Caddy.

Cloak

Посоветовали тут в комментариях. Штука интересная. Cloak - это не прокси-протокол, а только транспорт, то есть он делает подключение "точка-точка" (между вашим устройством и сервером), а внутри него уже можете гонять тот же Shadowsocks, или OpenVPN, или что угодно.

Работает поверх TLS 1.3, "свой/чужой" определяется по содержимому полей в ClientHello специальным образом (видимо очень схожим с XTLS-Reality), если "чужой" - то подключение передается на фейковый веб-сайт. Также используется TLS fingerprint от Chrome либо Firefox. Механизмы обфускации и подключения подробно описаны вот здесь: https://github.com/cbeuw/Cloak/wiki/Steganography-and-encryption. Есть также клиент под Android, и ещё транспорт Cloak поддерживается в некоторых мультпротокольные клиентах (например, Shadowrocket.

Если вы не хотите разбираться со всеми этими V2Ray, XRay, и подобным, у вас уже все настроено, и вы просто хотите обезопасить ваш существующий сервер (например, OpenVPN) от блокировки, то Cloak может быть отличным выбором

KCP (kcptun), mKCP

В сравнении со всем описанным выше, KCP - это протокол совершенно другого рода.

Авторы KCP переосмыслили алгоритмы передачи данных и разработали протокол, который работая поверх UDP обеспечивает надежную передачу, так же как TCP, но при этом в сравнении с TCP средняя задержка (пинг) при его использовании ниже на 30–40 %, а максимальная задержка меньше в три раза (правда, за счет потери полосы пропускания на 10–20%).

И все это как нельзя кстати оказалось для обхода блокировок, потому что в Китае и в некоторых арабских странах “неизвестные” протоколы нередко не блокировались полностью, а то ли намеренно, то ли из-за кривости механизмов фильтрации резались путем замедления и потерь пакетов. Также KCP может быть полезным при работе через отвратительные соединения (например, олдовый 3G в условиях плохого покрытия сети).

Для эффективной работы KCP требует указания в конфигурации измеренной реальной пропускной способности канала на прием и передачу.

Теперь разберемся с версиями и реализациями.

KCP - это оригинальный протокол. Kcptun - реализация туннеля на основе KCP.

mKCP - это вариант протокола KCP от V2Ray - по сути дела тот же KCP, но с небольшими изменениями (KCP и mKCP между собой не совместимы, имейте в виду). В V2Ray/XRay mKCP не является самостоятельным прокси-протоколом, а только лишь транспортом - то есть поверх него все так же нужно использовать VMess или VLESS. V2Ray/XRay имеют также опцию “congestion” для автоматической перенастройки параметров канала в случае высоких потерь пакетов, как и в оргигинальном kcptun можно задать секретный ключ (тут он называется “seed”) для усложнения детектирования, а еще можно маскировать внешний вид UDP-пакетов под SRTP (используемый, например, в Apple FaceTime), uTP (Bittorrent), WeChat, и DTLS (используемый в WebRTC, например, многими месседжерами).

Hysteria

Hysteria во многом очень похож на KCP, а ещё на всем известный QUIC, и авторы то ли вдохновлялись их механизмами, то ли напрямую в какой-то мере переиспользовали их. Кто авторы - тоже неизвестно, они сохраняют анонимость, документация на официальном сайте только на английском и китайском языке, но в примерах конфигурации среди секретных ключей встречается строка “Мать-Россия”, прямо так, кириллицей, что наталкивает на размышления.

Логотип Hysteria

Hysteria - это прокси-инструмент, как и Kcptun предназаченный для работы через нестабильные сети с потерями пакетов, ну и обхода блокировок, само собой.

В отличие от KCP, Hysteria передает данные не просто поверх UDP, а с использованием протокола QUIC (HTTP/3). Поэтому для работы на сервере должен иметься TLS-сертификат, сервер Hysteria умеет автоматически запрашивать

сертификаты методом ACME (например, от Let's Encrypt). Поскольку QUIC часто полностью блокируется в ряде стран (в том числе и в России), есть также возможность установить ключ для обфускации данных, в результате чего UDP-пакеты становятся ни на что не похожи.

Также как и для KCP, на стороне клиента Hysteria необходимо задать доступную ширину канала (например, в мегабитах), а еще есть интересный режим port hopping - разработчики подметили, что в Китае при детектировании "неправильных протоколов" бан накладывается не на весь IP-адрес целиком, а на связку IP+порт, поэтому сервер может слушать сразу на большом количестве портов, а "клиент" может прыгать на разные рандомные порты при неудачных попытках соединения.

И еще одна интересная возможность Hysteria - FakeTCP. В этом режиме клиент и сервер будут обмениваться пакетами, которые выглядят как TCP-пакеты (согласно их заголовку), но в обход системного TCP-стека и его механизмов. В итоге для всех промежуточных роутеров и цензоров обмен данными выглядит как TCP-подключение, хотя на самом деле им не является. Это может помогать в случае использования корпоративных фаерволов или цензоров, полностью режущих UDP. FakeTCP поддерживается только в Linux.

Meiru, TUIC, Brook, Pingtunnel

Эти протоколы вы мало где встретите, разве что только в самых упоротых клиентах. Я не встречал на просторах интернета упоминания их массового использования, поэтому просто пройдусь очень кратко, для общего развития, так сказать:

Meiru - аналог Shadowsocks / VMess+TCP, просто зашифрованный поток данных с паддингом поверх TCP или UDP.

TUIC - прокси-протокол поверх QUIC нацеленный на минимальный оверхед (0-RTT)

Brook - официально называется даже не прокси, а "cross-platform network tool designed for developers", видимо, чтобы не привлекать внимание цензоров, хотя в футере сайта есть гордое заявление "Undetectable Protocol". Информации об идеях в основе протокола и его преимуществах практически нет даже на официальном сайте и Github'е, судя по обрывочным данным, может работать в режиме "random" (как и Shadowsocks, непонятный поток данных), HTTP/HTTPS, в том числе поверх Websockets, и т.д. Возможно разработчики действительно придумали какие-то оригинальные идеи, затрудняющие детектирование, но никому об этом открыто не рассказывают, чтобы не привлекать внимания, в надежде что лезть и изучать исходники у цензоров не хватит терпения и квалификации, либо рассчитывают на эффект "неуловимого Джо".

PingTunnel - как следует из названия, позволяет проксировать TCP и UDP с помощью обычных ICMP-пингов. Звучит многообещающе.

Что использовать?

Зависит от того, насколько вы себя уверенно чувствуете в системном администрировании и готовы во всем этом разбираться методом проб и ошибок.

Если не уверены, либо не готовы и хочется максимально простое и универсальное решение, то я могу посоветовать настроить XRay в варианте VLESS-over-Websockets с fallback'ом на какой-нибудь безобидный веб-сайт. Со стороны клиентов обязательно выбрать опцию uTLS, и желательно добавить настройку чтобы ресурсы в зоне .ru и с российскими IP открывались без прокси.

Такая связка поддерживается практически всеми клиентами, еще долгое время будет устойчива к детектированию (учитывая отсталость и тормознутость нашего родного РКН), при наличии зарегистрированного домена можно работать через CDN, а устанавливается и настраивается все это элементарно парой команд в консоли (об этом будет в одной из следущих статей).

Если вы готовы к подвигам и экспериментам, то можно задуматься о настройке XRay с VLESS+XTLS-Vision, добавить fallback на VLESS+Websockets для старых клиентов и CDN разных видов, и на том же сервере поднять еще mKCP/Hysteria, Shadowsocks-2022 и классический SSH-туннель. И на будущее присмотреться к XTLS-Reality. В случае чего, хоть один из вариантов, но сработает.

А еще в одной из следущих статей я расскажу про клиенты. Потому что с клиентами дело обстоит примерно так же, как с серверами и протоколами: их много, они разные, некоторые из них являются форками друг друга, но с существенными отличиями, некоторые поддерживают одно, некоторые другое, а некоторые, казалось бы, поддерживают только это и это, но при правильном подходе их можно заставить поддерживать то, что они, казалось бы, не поддерживают :) Короче говоря, будет интересно, не переключайтесь.

Продолжения:

Программы-клиенты для недетектируемого обхода блокировок сайтов: V2Ray/XRay, Clash, Sing-Box, и все-все-все

Обход блокировок: настройка прокси-сервера XRay для Shadowsocks-2022 и VLESS с XTLS, Websockets и фейковым веб-сайтом

Bleeding-edge обход блокировок: настраиваем сервер и клиент XRay с XTLS-Reality быстро и просто

Если вы хотите сказать спасибо автору — сделайте пожертвование в один из благотворительных фондов: "Подари жизнь", "Дом с маяком", "Антон тут рядом".