Tor: Луковый маршрутизатор второго поколения

Roger Dingledine
The Free Haven Project
arma@freehaven.net

Nick Mathewson
The Free Haven Project
nickm@freehaven.net

Paul Syverson
Naval Research Lab
syverson@itd.nrl.navy.mil

Переведено
Алексеем Абакумкиным и Романом Инфлянскасом <infroma@gmail.com>
Москва, МГТУ им. Н. Э. Баумана, 2014 год

Аннотация

Мы представляем Tor — службу коммуникации с низкой латентностью, основанную на цепочках. Эта система луковой маршрутизации второго поколения решает проблемы оригинального проекта путём добавления совершенной прямой секретности, контроля перегрузки, серверов каталогов, проверки целостности, настраиваемых политик выхода и практичной схемы доступа к службам со скрытым местоположением через точки встречи. Tor работает в обычном интернете, не требует специальных привилегий или изменений ядра, требует небольшого количества синхронизаций между узлами и представляет собой разумный компромисс между анонимностью, эффективностью и простотой использования. Мы вкратце опишем наш опыт работы с интернациональной сетью из более чем 30 узлов. Мы завершим нашу статью списком нерешённых проблем анонимного общения.

1 Обзор

Луковая маршрутизация — это распределённая оверлейная сеть, созданная для анонимизации приложений таких как web-браузеры, SSH, клиенты мгновенных сообщений. Клиенты выбирают маршрут в сети и создают *цепочку*, в которой каждый узел (или «луковый маршрутизатор») знает только предыдущий и следующий узлы цепочки и не имеет понятия об остальных. Трафик проходит по цепочке в виде *ячеек* фиксированного размера, которые раскрываются (расшифровываются) с использованием симметричного ключа на каждом узле (подобно слоям луковицы) и передаются дальше. Проект луковой маршрутизации опубликовал несколько статей о проектировании и анализе [24, 37, 44, 45]. Несмотря на то что системы, использующие луковую маршрутизацию, развёрнуты во многих уголках планеты, единственная установка, которая действительно работала долго, была экспериментальной и работала она на одном единственном компьютере. Но даже эта простая установка системы обслуживала соединения более чем с шестьюдесятью тысяч различных IP-адресов со всего света, обрабатывая около пятидесяти тысяч ежедневно. Однако, множество просчётов в проектировании и развёртывании так и не были исправлены, а проект не обновлялся годами. В этой статье мы опишем Tor, протокол для асинхронных, слабо связанных луковых маршрутизаторов, который лучше, чем старая реализация по следующим позициям:

Совершенная прямая секретность: В первоначальном проекте одиночный вражеский узел мог записывать проходящий через него трафик, чтобы впоследствии скомпрометировать последующие узлы и заставить их расшифровать его. Вместо того чтобы многократно шифровать данные с одним и тем же ключом на определённом звене каждой цепочки, Tor использует инкрементальный способ построения цепочки, в котором узел, инициировавший обмен, согласовывает сессионные ключи после каждого успешного перехода в цепочке. Так как старые ключи удалены, последующие узлы не могут расшифровать старый трафик. Положительным побочным эффектом является то, что определение повтора передачи больше не является необходимым, а процесс составления цепочек становится более надёжным, так как инициатор знает, когда переход не удался и может попробовать осуществить её снова с использованием другого узла.

Отделение «очистки протокола» от

анонимности: Луковая маршрутизация изначально требовала отдельного «прокси уровня приложения» для каждого протокола каждого приложения. Большинство прокси так и не были написаны, то есть множество приложений никогда не поддерживались. Tor использует стандартный и близкий к SOCKS [28] прокси-интерфейс, позволяя поддерживать множество программ, работающих по TCP без изменений. В настоящий момент Tor рассчитывает на прокси уровня приложений (такие как Privoxy [35]), а не пытается повторять их функциональность.

Отсутствие перемешивание, выравнивание и шейпинг трафика (на данный момент): Изначально луковая маршрутизация группировала и перемешивала ячейки по их получению, предполагая, что они выровнены. В дальнейшем было добавлено выравнивание трафика между луковыми прокси (пользователями) и луковыми маршрутизаторами [24, 37]. Обсуждались компромиссы между защитой дополнений для выравнивания и их сложностью. Были теоретически обоснованы алгоритмы *шейпинга трафика* [45] для обеспечения серьёзного уровня безопасности без затратного выравнивания, но никакой схемы выравнивания так и не было предложено. Последние исследования [1] и опыт развёртывания [4] показали, что такой уровень использования ресурсов не является ни практичным, ни экономичным и даже полное выравнивание является уязвимым [29]. Таким образом, пока мы не найдём доказанного и удобного решения для шейпинга трафика или его перемешивания с низкой латентностью, повышающего анонимность против реального неприятеля, мы не будем использовать перемешивание, выравнивание и шейпинг трафика.

Множество TCP-потоков могут использовать одну цепочку: Изначально луковый маршрутизатор был сделан таким образом, чтобы создавать отдельную цепочку для каждого запроса уровня приложения. Это, однако, требовало множество операций с публичным ключом для каждого запроса и также представляло угрозу для анонимности в случае создания большого количества цепочек.

Топология протекающей трубы цепочки: С помощью внутриполосного оповещения внутри цепочки, узлы Tor, инициировавшие соединение, могут направлять трафик узлам, находящимся в середине цепочки. Этот новый подход позволяет трафику покинуть цепочку из середины. Это используется для того, чтобы избежать потери трафика и атак, связанных с определением конца цепочки. Это также позволяет осуществлять выравнивание трафика на большом диапазоне, если последующие исследования покажут, что это имеет смысл.

Контроль перегрузки: Старый подход к анонимности не решал проблемы узких мест при передаче трафика. К сожалению, типичные подходы к балансировке нагрузки и контролю перегрузки в оверлейных сетях используют общение между узлами и общего представления о трафике. Контроль перегрузки Tor использует сквозные подтверждения конца передачи для обеспечения анонимности, в то же время позволяя узлам на концах сети определять перегрузку или заваливание запросами и уменьшать количество пересылаемых данных до тех пор, пока перегрузка не исчезнет.

Серверы каталогов: В старом проекте луковой маршрутизации было запланировано передавать информацию о состоянии сети с помощью лавинной маршрутизации. Этот подход, вероятно, был бы сложным и ненадёжным. Tor использует упрощённое представление для пересылки этой информации. Определённые (более доверенные) узлы играют роль *серверов каталогов*: они предоставляют подписанные каталоги, описывающие известные маршруты и их текущее состояние. Пользователи периодически скачивают их через HTTP.

Изменяемые политики выхода: Tor предоставляет надежный механизм для каждого узла для объявления политик, описывающих компьютеры и порты, к которым он будет подключаться. Эти политики выхода являются критичными в распределённой сети, поддерживаемой силами добровольцев, поскольку каждый из пользователей имеет право разрешать выход различного типа трафика с его узла.

Сквозная проверка целостности: В первоначальном проекте лукового маршрутизатора не была описана проверка целостности данных. Каждый узел в цепочке может изменять содержание ячеек данных во время их прохождения — например, после запроса подключения он будет подключаться к другому web-

серверу или 'помечать' зашифрованный трафик и определять соответствующие испорченные пакеты на границах сети [13]. Tor затрудняет проведение этих атак, так как он осуществляет проверку целостности когда трафик покидает сеть.

Точки рандеву и скрытые службы: Tor предоставляет встроенный механизм для обеспечения анонимности посредством серверов со скрытым местоположением. Предыдущий проект лукового маршрутизации включал долго живущие «отвечающие луковицы», которые могли служить для создания скрытых серверов, но эти луковицы не обеспечивали прямой секретности и становились бессмысленными, если какой-то из узлов падал или менял свои ключи. В Tor клиенты согласуют *точки рандеву* для подключения к скрытым серверам; отвечающие луковицы больше не нужны.

В отличие от Freedom [8], Tor не требует патчей ядра или поддержки сетевого стека. Это мешает нам реализовать анонимизацию не TCP протоколов, но сильно помогло нам в переносимости и простоте развёртывания.

Мы реализовали всю перечисленную выше функциональность, включая точки рандеву. Наш код доступен по свободной лицензии и Tor не подпадает ни под один патент (в отличие от старых версий луковой маршрутизации). Мы развернули большую сеть альфа-версии для того, чтобы протестировать наш проект, получить больше опыта от общения с пользователями и для того, чтобы создать исследовательскую платформу для экспериментов. На момент написания этой статьи сеть состоит из 32 узлов, находящихся на двух континентах.

Мы сделаем обзор предыдущих работ в разделе 2, опишем наши цели и допущения в разделе 3 и опишем описанные выше улучшения в разделах 4, 5 и 6. Мы опишем как наше решение противостоит известным атакам в разделе 7. Мы завершим нашу статью разделом 8, в котором опишем наши дальнейшие планы.

2 Связанные работы

Современные системы обеспечения анонимности основаны на проекте **Mix-Net** [10]. Chaum предложил спрятать связь между отправителем и получателем путём оборачивания сообщений в асимметрично шифрованные слои и передачи их через путь, составленный «комбинаторами». Каждый комбинатор в свою очередь дешифрует, придерживает и перемешивает сообщения перед передачей их дальше.

Последующие проекты, основанные на ретрансляторах, разошлись в два разных направления. Системы, подобные **Babel** [25], **Mixmaster** [32] и **Mixminion** [13], попытались максимизировать анонимность ценою введения больших задержек переменной длины. Благодаря этому решению *высоко-латентные* сети выдерживают сильные атаки, но задержки настолько велики, что нормально пользоваться интерактивными приложениями, такими как web-браузеры, клиенты мгновенных сообщений и SSH не представляется возможным.

Tor относится ко второй категории: *низко-латентные* решения, которые пытаются обеспечить анонимность интерактивных сетевых приложений. Эти системы работают с большим набором двусторонних протоколов. Они также обеспечивают более удобную доставку почты, чем высоко-латентные анонимные сети, поскольку почтовый сервер предоставляет явное и ограниченное по времени подтверждение получения посылки. Такие решения обычно обрабатывают большое количество пакетов, которые должны быть доставлены быстро. Поэтому такие системы сложно защитить от атакующего, который прослушивает сообщения на обоих концах и сопоставляет задержки и объём трафика перед выходом его из анонимной сети [41]. Эти протоколы также уязвимы от активного атакующего, который создаёт трафик с определённой частотой и смотрит соответствие по времени выхода на другом конце анонимной сети. Несмотря на то, что была проделана определённая работа, чтобы избежать атаки такого рода, большая часть решений защищает в первую очередь от анализа трафика, но не от его подтверждения (см. раздел 3.1).

Наиболее простое низко-латентное решение — прокси с одним транзитным участком, такие как **Anonymizer** [3]: единственный доверенный сервер обрезает всю информацию об источнике информации перед пересылкой. Такие решения легко могут быть проанализированы [на предмет стойкости; примечание переводчика], но пользователи должны доверять анонимизирующему прокси. Увеличение кон-

центрации трафика на одном узле увеличивает анонимность (точно так же как люди, чтобы спрятаться, смешиваются с толпой), но такие решения уязвимы, если атакующий может просматривать трафик, входящий и покидающий прокси сервер.

Более сложные системы являются распределённо-доверенными. Эти анонимизирующие системы основаны на цепочках. В них пользователь устанавливает одну или более двунаправленную сквозную цепочку и туннелирует данные в виде ячеек установленного размера. Создание цепочки является вычислительно дорогим и обычно требует использования асимметричного шифрования, в то время как пересылка ячеек достаточно быстра и обычно требует только симметричного шифрования. Поскольку цепочка проходит через несколько серверов, каждый из которых знает только о своих соседях по цепочке, никакой из них не может соединить пользователя с его собеседником.

Java Anon Proxy (также известен как JAP или Web MIXes) использует фиксированные общеизвестные маршруты, известные как *каскады*. Точно так же, как и в случае с прокси с одним транзитным участком, этот подход объединяет пользователей в большие группы анонимности. Этот подход имеет тот же недостаток: атакующей стороне достаточно прослушать весть трафик с двух сторон каскада, чтобы связать трафик всей системы в единое целое. В Java Anon Proxy используется выравнивание между конечными пользователями и главным узлом каскада [7]. Однако не доказано, что такая реализация выравнивания улучшает анонимность.

Другая низко-латентная система — **PipeNet** [5, 12], которая была предложена примерно в то же время, что и луковая маршрутизация, обеспечивала лучший уровень анонимности, но позволяла единственному пользователю прекратить работу всей сети, если тот переставал пересылать пакеты. Такие системы как **ISDN mixes** [34] были спроектированы для работы в других окружениях с другими допущениями.

В P2P решениях, таких как **Tarzan** [21] и **MorphMix** [39], все участники сети одновременно генерируют трафик и пересылают его другим. Такие системы ставят своей целью скрыть, был ли заданный пользователь отправителем запроса или он всего лишь передал его от другого пользователя. В то время как Tarzan и MorphMix используют такое же многослойное шифрование, как и предыдущие системы, **Crowds** [38] просто предполагает, что атакующая сторона не знает инициатора соединения: эта система не использует шифрования открытым ключом, поэтому каждый узел может читать пользовательский трафик.

Система **Hordes** [30] основана на Crowds, но она также использует многоадресную рассылку, чтобы скрыть инициатора обмена. **Herbivore** [22] и $\mathbf{P^5}$ [42] пошли даже дальше: они используют широковещательную рассылку. Эти системы изначально задумывались для общения между пирами, однако пользователи Herbivore могут делать внешние соединения, путём использования пира в качестве прокси.

Системы, такие как **Freedom** и первоначальный проект луковой маршрутизации, создают все цепочки разом, используя «луковицу» из сообщений, зашифрованных открытым ключом, в которой каждый из слоёв предоставляет сессионные ключи и адрес следующего сервера по цепочке. Tor, описанный в данной статье, Tarzan, MorphMix, **Cebolla** [9] и Rennhard's **Anonymity Network** [40] создаёт цепочки по частям, удлиняя их на один транзитный участок за раз. В разделе 4.2 описано, как этот подход обеспечивает совершенную прямую секретность.

Решения, основанные на цепочках, должны определиться с тем, какой протокол они анонимизируют. Они могут перехватывать непосредственно IP пакеты и пересылать их целиком (обрезая, разумеется, адрес отправителя) вниз по цепочке [8, 21]. Они могут, как и Tor, обрабатывать TCP-потоки и пересылать данные из этих потоков, игнорируя разбиение данных на TCP сегменты [39, 40]. Наконец, они могут, как и Crowds, обрабатывать протоколы уровня приложения, такие как HTTP, и пересылать непосредственно запросы приложений. Выбор уровня протокола должен быть компромиссом между анонимностью и гибкостью. Например, система, понимающая HTTP, может обрезать идентифицирующую информацию из запросов, использовать кэширование для уменьшения количества запросов, покидающих сеть, группировать или кодировать за-

просы для того, чтобы сократить количество соединений. С другой стороны, анонимайзер уровня IP может совладать с любым протоколом, даже тем, что ещё не придуман. Проблема заключается в том, что такие системы требуют модификаций ядра (в некоторых операционных системах), гораздо более сложны и менее переносимы. Анонимные сети TCP уровня, такие как Tor, работают на промежуточном уровне: они независимы от приложений (разумеется когда приложение поддерживает TCP или может туннелироваться поверх него), но оно рассматривает соединения приложений как потоки данных, а не как сырые TCP пакеты, что позволяет избежать неэффективности туннелирования TCP поверх TCP.

Распределённо-доверенные анонимизирующие системы должны быть защищены от атакующих, которые добавляют так много [своих; примечание переводчика] серверов, что могут компрометировать пользовательские пути. Tor рассчитывает на небольшое множество доверенных серверов каталогов, запущенны независимыми сторонами, для определения узлов, которые могут присоединиться к сети. Tarzan и MorphMix позволяют неизвестным пользователям запускать серверы. Он решают проблему защиты от атакующего, контролирующего слишком большую часть сети, путём введения ограничений на ресурсы (такие как IP-адреса). Crowds предлагает обязать своих потенциальных пользователей предъявлять нотариально заверенные заявления в письменном виде.

Анонимное общение является необходимой частью систем, защищающих от цензурирования, таких как Eternity [2], Free Haven [17], Publius [49] и Tangler [48]. Точки рандеву Tor позволяют осуществлять соединения между взаимноанонимными сторонами; эти точки рандеву — строительные блоки для серверов со скрытым местоположением, которые необходимы для Eternity и Free Haven.

3 Цели проектирования и допущения

Цели

Так же как и другие низко-латентные анонимные решения, Tor стремиться усложнить жизнь атакующим, которые пытаются выяснить связь между собеседниками или между различными сессиями общения одного и того же пользователя. Помимо этой основной цели на эволюцию Tor повлияли несколько других соображений.

Простота развёртывания: Проектируемое решение должно развёртываться и использоваться в реальном мире. Это означает, что Tor должен быть недорогим в использовании (например, он не должен требовать большую скорость передачи, чем пользователь готов предоставить); не должен обременять пользователей ответственностью (пользователи не должны страдать от того, что атакующие использовали луковый маршрутизатор в своих нелегальных действиях); должен быть прост и дёшев в реализации (то есть он не должен требовать патчей ядра или специализированных прокси для каждого протокола). Мы также не можем требовать, чтобы неанонимные стороны (такие как сайты) использовали наше ПО (наши точки рандеву, однако, не удовлетворяют этому критерию, когда неанонимные пользователи общаются со скрытыми серверами; см. раздел 5).

Простота использования: Более сложная система имеет меньше пользователей. Анонимизирующие системы скрывают пользователей среди других. Из этого следует, что более используемая система обеспечивает лучшую анонимность. Таким образом, простота использования — это не только удобство, это требование безопасности [1, 5]. Вследствие этого Tor не должен требовать изменения знакомых приложений, не должен вызывать чрезмерных задержек, а также должен требовать как можно меньше настроек. Наконец, Tor должен быть лёгким в установке на всех популярных платформах. Мы не можем требовать от пользователя менять операционную систему для обеспечения анонимности (в данный момент Tor работает на Win32, Linux, Solaris, BSD-подобных Unix, OS X и, возможно, других системах).

Гибкость: Протокол должен быть гибким и хорошо определённым, для того чтобы Tor мог служить основой для будущих исследований. Многие открытые проблемы в низколатентных анонимных сетях, такие как генерирование мусорного трафика или защита от атак Сивилла [19], могут быть рассмотрены отдельно от проблем, решаемых Tor. Мы надеем-

ся, что будущим системам не надо будет переизобретать Tor.

Простота проекта: Проект протокола и параметры безопасности должны быть легко понимаемыми. Дополнительные возможности несут дополнительные сложности; добавление непротестированных приёмов в проект угрожает простоте развёртывания, читаемости и простоте анализа безопасности. Tor стремится быть простой и стабильной системой, которая объединяет лучшие признанные идеи для обеспечения безопасности.

Что не является целью

С целью получения простого, лёгкого в развёртывании решения, мы специально отложили из рассмотрения несколько возможных целей, поскольку они либо были решены где-то ещё, либо не были решены вовсе.

Не P2P: Tarzan и MorphMix стремятся масштабировать полностью децентрализованные окружения с тысячами недолговечными серверами, многие из которых могут контролироваться противником. Этот подход кажется привлекательным, но имеет много проблем [21, 39].

Небезопасен против сквозных атак: Tor не стремится полностью защитить от сквозных атак синхронизации или от атак пересечения. Некоторые подходы, такие как использование собственного лукового маршрутизатора у каждого пользователя, могут частично помочь.

Нет нормализации протоколов: Tor не предоставляет *нормализации протоколов* как, например, Privoxy или Anonymizer. Если отправители хотят анонимности от получателей при использовании таких сложных и изменчивых протоколов как HTTP, Tor должен быть обёрнут фильтрующим прокси таким как Privoxy, для того чтобы спрятать разницу между клиентами и убрать из протокола те возможности, которые могут раскрыть личность. Заметьте, что благодаря этому разделению Tor может также предоставлять анонимность службам, в которых пользователь аутентифицируется отвечающей стороной. Такой службой является SSH. Аналогично, Tor не туннелирует протоколы, не основанные на потоках, такие как UDP; их туннелирование должно быть предоставлено внешними службами, если это возможно.

Не является стеганографичным: Tor не пытается утаивать, кто подключён к сети.

3.1 Модель угроз

Обычно при теоретическом анализе проектов анонимных систем считается, что главной угрозой является глобальный пассивный враг. Но, как и все практические низко-латентные системы, Tor не защищает от такого сильно врага. Напротив, мы считаем, что враг может обозревать только часть сетевого трафика, но может генерировать, изменять, удалять или задерживать трафик; он может быть владельцем луковых маршрутизаторов; он может компрометировать часть других луковых маршрутизаторов.

В низко-латентных анонимизирующих системах, которые используют послойное шифрование, типичной целью атакующего является просмотр как инициатора, так и отвечающего. Путём просмотра обоих концов пассивная атакующая сторона может подтвердить, что Алиса не общается с Бобом, если синхронизация нарушена и объём трафика достаточно сильно отличается. Активный атакующий может «подписать» трафик с помощью временных задержек и обеспечить тем самым различные шаблоны трафика. Вместо того чтобы фокусироваться на этих атаках *подтверждения трафика*, мы стремимся предотвратить атаки *анализа трафика*, то есть атаки, в которых атакующий ищет шаблоны трафика, чтобы определить, какие точки сети он должен атаковать.

Наш враг может связать инициатора общения Алису с её собеседниками или может попробовать составить профиль её поведения. Он может осуществлять пассивные атаки путём наблюдения за границами сети и сопоставления трафика, поступающего в сеть и покидающего её (по связи между синхронизацией пакетов, объёма или видимых внешне опознавательных знаках пользователя). Враг может также осуществлять активные атаки путём компрометации роутеров или ключей; путём повтора передачи трафика; путём отказа в обслуживании доверенных роутеров (для того чтобы переключить пользователей на компрометированные роутеры); путём отказа в обслуживании пользователей (для того чтобы отследить, в каком месте сети передача трафи-

ка остановилась); путём внедрения шаблонов в трафик (для того чтобы в дальнейшем отследить их). Атакующий может также нарушить работу серверов каталогов, для того чтобы предоставить разным пользователям разное представление от сети. Также он может попробовать уменьшить надёжность сети с помощью атаки узлов или путём осуществления противоправных действий от имени надёжных узлов (чтобы их выключили). Все эти действия уменьшают количество пользователей сети, делая её менее анонимной и более уязвимой для атак. Мы опишем как наше решение противостоит таким атакам в разделе 7.

4 Проект Tor

Сеть Tor — это оверлейная сеть; каждый луковый маршрутизатор работает, как обычный процесс уровня пользователя без всяких специальных привилегий. Каждый луковый маршрутизатор поддерживает TLS [15] соединение с каждым другим луковым маршрутизатором сети.

Каждый пользователь запускает на своём компьютере программу, называемую луковым прокси, для получения информации из серверов каталогов, установления цепочек в сети и обработки соединений от пользовательских приложений. Эти луковые прокси принимают TCP-потоки и мультиплексируют их по цепочкам. Луковый маршрутизатор на другом конце цепочки соединяет пользователя с адресом назначения и передаёт данные дальше.

Каждый луковый маршрутизатор поддерживает долгосрочный идентификационный ключ и краткосрочный луковый ключ. Идентификационный ключ используется для подписи TLS сертификатов, *дескриптора лукового маршрутизатора* (объединение его ключей, адресов, скорость передачи, политики точки выхода и т.п.), каталогов (используется серверами каталогов). Луковый ключ используется для расшифровки запросов от пользователей на установление цепочки и для согласования временных ключей. TLS протокол также устанавливает краткосрочный ключ соединения при общении между луковыми маршрутизаторами. Краткосрочные ключи меняются периодически и независимо друг от друга, для того чтобы уменьшить ущерб от компрометации ключа.

В разделе 4.1 описаны *ячейки* фиксированного размера, которые являются единицей общения в Tor. В разделе 4.2 мы описываем, как цепочки создаются, расширяются, сокращаются и уничтожаются. В разделе 4.3 описано, как маршрутизируются TCP-потоки. Мы опишем проверку целостности в разделе 4.4, а ограничение ресурсов в разделе 4.5. Наконец, в разделе 4.6 мы опишем, как происходит контроль перегрузки и обсудим вопросы справедливости.

4.1 Ячейки

Луковые маршрутизаторы общаются друг с другом, а также с пользователями луковых прокси через TLS соединения, зашифрованными временными ключами. Использование TLS защищает данные соединения с совершенной прямой секретностью, не даёт возможности атакующему изменить данные или выдать себя за луковый маршрутизатор.

Трафик в этих соединениях передаётся ячейками фиксированного размера. Каждая ячейка имеет размер 512 байт и состоит из заголовка и полезной информации. Заголовок содержит идентификатор ячейки (circID), который определяет, на какую цепочку ссылается данная ячейка (в одном TLS соединении могут быть мультипликсированы много цепочек) и команда, которая определяет что делать с полезной информацией ячейки (идентификатор ячейки является заданным для соединения: каждая цепочка имеет различный circID на каждом соединении «луковый маршрутизатор — луковый прокси» или «луковый маршрутизатор — луковый маршрутизатор», которые она проходит). В зависимости от их команды ячейки могут быть *управляющими* или *передающими*. *Управляющие* интерпретируются узлами, которые их получают. *Передающие* несут в себе данные, которые нужно передать насквозь через сеть. Управляющие команды: *выровнять* (в данный момент используется для keepalive, может также быть использована для выравнивания ссылок); *создать* или *создано* (служит для создания новой цепочки); *уничтожить* (завершение цепочки).

Передающие цепочки имеют ещё один заголовок (передающий заголовок), который находится перед полезной информации и содер-

жит streamID (идентификатор потока: на одну цепочку могут быть мультиплексированы несколько потоков) и сквозную контрольную сумму для проверки целостности, длину передаваемой полезной информации и передаваемую команду. Всё содержимое передаваемого заголовка и передаваемой полезной информации ячейки шифруется и дешифруется вместе каждый раз, когда передающая ячейка движется по цепочке. Для этого используется 128-битный AES шифрователь, работающий в режиме счётчика для генерации шифрованного потока. Команды передачи: *передать данные*, *начать передачу данных* (для открытия потока), *закончить передачу данных* (для корректного закрытия потока), *завершить передачу данных* (для закрытия испорченного потока), *передача данных начата* (для оповещения луковых прокси, что начало передачи было успешным), *расширить цепочку* и *цепочка расширена* (для расширения цепочки за один переход и для получения подтверждения), *сократить цепочку* и *цепочка сокращена* (для того, чтобы завершить только часть цепочки, и чтобы послать подтверждение), *передать самому себе* (служит для контроля перегрузки) и *прервать передачу данных* (служит для создания пустышек на длинные дистанции). Сначала мы сделаем обзор структуры ячейки (обычной и передающей), а потом опишем все типы ячеек и их команды более подробно.

2	1	509 bytes
CircID	CMD	DATA

2	1	2	6	2	1	498
CircID	Relay	StreamID	Digest	Len	CMD	DATA

4.2 Ячейки и потоки

Луковая маршрутизация изначально использовала по отдельной цепочке на каждый TCP-поток. Поскольку создание каждой цепочки может занимать несколько десятых секунды (из-за использования шифрования с открытым ключом и задержек сети), система в целом получается медленной для приложений, открывающих много TCP-потоков (например, web-браузеров).

В Tor каждая цепочка может быть использована несколькими TCP-потоками. Для того чтобы избежать задержек, пользователи создают цепочки заранее. Для того чтобы снизить связность между его потоками, луковый прокси пользователя периодически признаёт старые неиспользуемые цепочки устаревшими и создаёт взамен новые. Луковый прокси заменяет цепочки новыми раз в минуту: таким образом даже для «тяжёлых» [генерирующих много трафика; примечание переводчика] пользователей время на генерацию ничтожно, но количество цепочек, которые могут быть связаны друг с другом через выбранный узел выхода, ограничено. В дополнение к этому, так как цепочки генерируются в фоне, луковый прокси может восстановиться после ошибки при создании цепочки незаметно для пользователя.

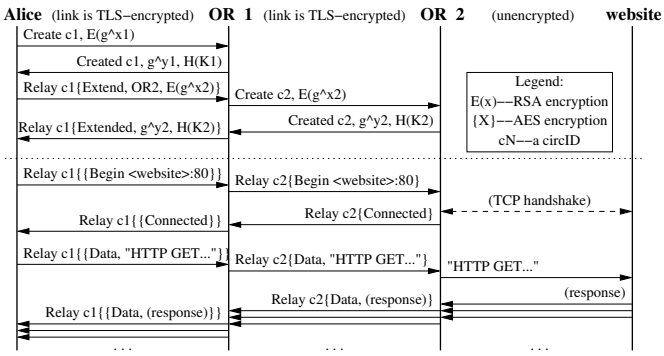

Рис. 1: Алиса создаёт цепочку с двумя переходами и начинает загружать web-страницу.

Создание цепочки

Луковый прокси пользователя создаёт цепочку последовательно, согласуя симметричный ключ с каждым луковым маршрутизатором в цепочке, по одному переходу за раз. Для начала создания новой цепочки луковый прокси (назовём его Алисой) посылает ячейку с командой *создать* на первый узел (назовём его Боб) в выбранном ею пути. Она выбирает новый circID C_{AB}, который не был использован в текущем соединении её с Бобом. Ячейка *создать* содержит в качестве полезной информации первую часть квитирования [англ. handshake; примечание переводчика] протоколом Диффи—Хеллмана (g^x), зашифрованного луковым ключом Боба. Боб отвечает ячейкой *создано*, которая содержит g^y вместе с хешем согласованного ключа $K = g^{xy}$.

Сразу после того как ячейка была создана, Алиса и Боб могут отправлять друг другу передающие ячейки, зашифрованные с помощью

согласованного ключа.[1] Для более подробного описания см. следующий раздел.

Для того чтобы расширить цепочку, Алиса шлёт ячейку *расширить цепочку* Бобу, в которой указан адрес следующего лукового маршрутизатора (назовём его Кэрол) и зашифрованное значение g^{x_2} для этого маршрутизатора. Боб копирует половину квитирования в ячейку *создать* и передаёт её дальше Кэрол для расширения цепочки. При этом Боб выбирает новый circID C_{BC}, который не был ещё использован в текущем соединении между ним и Кэрол. Алисе не нужно знать этот circID; только Бобу нужно поддерживать связь между двумя circID: C_{AB} на соединении с Алисой и C_{BC} на соединении с Кэрол. Когда Кэрол отвечает ячейкой *создано*, Боб оборачивает полезную информацию в ячейку *цепочка расширена* и посылает её назад Алисе. Теперь цепочка расширена до Кэрол и Алиса и Кэрол используют общий ключ $K_2 = g^{x_2 y_2}$.

Для расширения цепочки до третьего узла и дальше Алиса продолжает делать то же самое, каждый раз сообщая последнему узлу в цепочке расширить её на один переход.

Этот протокол квитирования уровня цепочки позволяет достичь односторонней аутентификации сущностей (Алиса знает, что она квитируется с луковым маршрутизатором, но луковому маршрутизатору всё равно, кто открывает цепочку — Алиса не использует открытого ключа и остаётся анонимной) и односторонней аутентификации ключей (Алиса и луковый маршрутизатор договариваются о ключе и Алиса знает только, что он его узнал). Это также позволяет достичь прямой секретности и свежести ключа. Более формализовано протокол может быть представлен следующим образом (где $E_{PK_{\text{Боб}}}(\cdot)$ это зашифрованное открытым ключом Боба, H — безопасная хеш-функция и $|$ — конкатенация):

$$\text{Алиса} \rightarrow \text{Боб} : E_{PK_{\text{Боб}}}(g^x)$$
$$\text{Боб} \rightarrow \text{Алиса} : g^y, H(K|<< \text{квитирование"})$$

В следующем шаге Боб подтверждает, что именно он получил g^x и выбрал y. Мы используем шифрование с открытым ключом, поскольку единственная ячейка слишком мала, чтобы содержать сразу и открытый ключ, и подпись. Предварительный анализ с помощью NRL анализатора протоколов [31] показал, что протокол является безопасным (и обладает совершенной прямой секретностью) в традиционной модели Dolev-Yao.

[1]На самом деле, согласованный ключ используется чтобы создать два симметричных ключа: каждый для своего направления.

Передающие ячейки

Как только Алиса устанавливает цепочку (то есть обменивается ключами с каждым луковым маршрутизатором в цепочке), она может слать передающие ячейки.

Перед тем как принять передающую ячейку, луковый маршрутизатор находит соответствующую цепочку, расшифровывает заголовок и полезную информацию с помощью сессионного ключа этой цепочки. Если ячейка снабжена заголовком Алисы, то луковый маршрутизатор потом проверяет корректность свёртки [англ. *digest*; примечание переводчика] расшифрованной ячейки. Процесс проверки можно сильно оптимизировать, если учесть, что первые два байта должны быть нулями. Таким образом, в большей части случаев вычислять хеш не требуется.

Если свёртка корректна, то он принимает передающую ячейку и совершает действия, описанные ниже. В противном случае, луковый маршрутизатор смотрит на circID и луковый маршрутизатор для следующего шага в цепочке, заменяет circID на подходящий и посылает расшифрованную передающую ячейку следующему луковому маршрутизатору. Если луковый маршрутизатор на конце цепочки получает передающую цепочку, которую он не может опознать, он генерирует ошибку и цепочка завершается.

Луковый прокси обрабатывает входящие передающие ячейки похожим образом: он последовательно раскрывает передающий заголовок и полезную информацию при помощи сессионных ключей, используемых совместно с каждым луковым маршрутизатором в цепочке, начиная от ближайшего, заканчивая дальним. Как только свёртка оказывается корректной, автором ячейки признаётся тот луковый маршрутизатор, зашифрованное сообщение которого было только что удалено.

Для того чтобы создать передающую ячейку для конкретного лукового маршрутизатора, Алиса находит соответствующую цепочку, рас-

шифровывает заголовок и полезную информацию с помощью сессионного ключа этой цепочки. Так как на каждом шаге свёртка шифруется в различные значения, только луковый маршрутизатор, которому предназначалось сообщение, получит осознанное значение.[2] Эта топология *протекающей трубы* позволяет потокам Алисы выходить через разные луковые маршрутизаторы с использованием единственной цепочки. Алиса может выбирать различные точки выхода благодаря политикам точек выхода, а также сохранять в тайне то, что посланные ею потоки были созданы одним отправителем.

Когда луковый маршрутизатор отвечает Алисе передающей цепочкой, он шифрует заголовок передающей ячейки и полезную информацию единственным ключом (общим с Алисой) и посылает ячейку по цепочке назад Алисе. Последующие луковые маршрутизаторы при передаче ячейки назад Алисе добавляют свои слои шифрования.

Для того чтобы уничтожить цепочку, Алиса шлёт управляющую ячейку *уничтожить*. Каждый луковый маршрутизатор, который получает ячейку *уничтожить*, закрывает все потоки этой цепочки и передаёт ячейку *уничтожить* дальше. Нормальное завершение цепочек происходит так же, как и создание: инкрементально. Алиса посылает ячейку *сократить цепочку* единственному луковому маршрутизатору из цепочки. Этот луковый маршрутизатор посылает ячейку *уничтожить* дальше и подтверждает этот факт ячейкой *цепочка сокращена*. Алиса может расширить цепочку до различных узлов без оповещения промежуточных узлов о своих намерениях. Если один из узлов прекращает свою работу, соседний узел может послать ячейку *цепочка сокращена* назад Алисе. Таким образом, мы частично защищаемся от атаки вида «сломай узел и посмотри, какие цепочки оборвались» [4].

4.3 Открытие и закрытие потоков

Когда приложение Алисы хочет создать TCP соединение с заданным адресом и портом, оно посылает запрос луковому прокси (через SOCKS). Луковый прокси выбирает последнюю открытую цепочку (или при необходимости создаёт новую) и подходящий луковый маршрутизатор в этой цепочке в качестве узла выхода (обычно им является последний узел, однако в случае конфликта политик точек выхода это не так; см. раздел 6.2.) После этого луковый прокси открывает поток путём посылки ячейки *начать передачу данных* узлу выхода, используя новый случайный streamID. Как только узел выхода соединяется с удалённым компьютером, он отвечает ячейкой *передача данных начата*. По получении луковый прокси шлёт SOCKS ответ, чтобы оповестить приложение об успехе. Луковый прокси теперь принимает данные от TCP-потока приложения, упаковывает их в ячейки *передать данные* и шлёт их по цепочке заданному луковому маршрутизатору.

Есть правда одна особенность, касающаяся SOCKS: некоторые приложения передают клиенту Tor имя узла, в то время как другие разрешают его в IP-адрес и передают уже его. Если приложение сначала разрешает имя узла с помощью DNS, то получается, что Алиса раскрывает узел назначения удалённому DNS серверу. Этого не происходит в первом случае (разрешение узла происходит уже внутри сети Tor). Этот недостаток есть в таких приложениях как Firefox и SSH.

С Firefox эту проблему легко решить: поставьте фильтрующий прокси, который называется Privoxy. Он будет выдавать имена узлов клиенту Tor, то есть компьютер Алисы никогда не будет разрешать имена узлов с помощью DNS. А вот в случае SSH всё сложно. Решения до сих пор не существует. Изменение локального сервера имен является агрессивным, нестабильным и непереносимым решением. Заставить библиотеку разрешения имён использовать TCP, а не UDP сложно. Плюс к тому, есть проблемы с переносимостью. Динамический перехват системных вызовов библиотеки разрешения имён кажется нам многообещающим решением. Мы можем также сделать утилиту типа *dig* для осуществления приватного просмотра сети Tor. На данный момент мы советуем вам всегда использовать прокси, которые уважают анонимность пользователя, такие как Privoxy.

[2] С 48-битными свёртками на каждую ячейку вероятность коллизии значительно меньше, чем вероятность отказа оборудования.

Закрытие потока Tor аналогично закрытию TCP-потока: оно требует двухступенчатого квитирования для нормальных операций или одноступенчатого для ошибок. Если поток закрывается внезапно, то сосед узел просто шлёт ячейку *завершить передачу данных*. Если поток завершается нормально, то узел шлёт ячейку *закончить передачу данных* вниз по цепочке и другая сторона отвечает своей ячейкой *закончить передачу данных*. Так как все передающие ячейки используют шифрование, только луковый маршрутизатор назначения знает, что эта ячейка является запросом закрытия потока. Это двухступенчатое квитирование позволяет Tor поддерживать TCP приложения, которые используют наполовину закрытые соединения.

4.4 Проверка целостности потоков

Поскольку в старом проекте лукового маршрутизатора использовался потоковое шифрование без проверки целостности, трафик был уязвим уступчивым атакам [англ. malleability attack; примечание переводчика]: несмотря на то что атакующий не мог расшифровать ячейки, любые изменения зашифрованного трафика порождали изменения соответствующего трафика, покидающего сеть. Это позволяло атакующему, который мог угадать зашифрованное содержимое, изменить выравнивание ячейки для её уничтожения; изменить адрес назначения в ячейке *начать передачу данных* на адрес web-сервера атакующего; или изменить команду `dir` на `rm *` (даже атакующий извне мог сделать это, поскольку шифрование канала также использовало поточный шифр).

Поскольку Tor использует TLS для шифрования своих каналов, внешние враги не могут изменить данные. Борьба с уступчивыми атаками, идущими изнутри сети, однако, более сложна.

Мы бы могли осуществлять проверку целостности передающих ячеек после каждого перехода путём включения хешей или с помощью аутентификации шифрующего режима вроде EAX [6], однако есть некоторые проблемы. Во-первых, эти способы накладывают лишние затраты на расширение сообщения при каждом переходе, что может привести к потере длины пути или лишнему расходованию байтов (когда длина сообщения меньшей максимальной происходит выравнивание). Во-вторых, эти решения могут проверить только трафик, идущий от Алисы: луковые маршрутизаторы не могут посчитать правильные хеши для промежуточных переходов, так как луковые маршрутизаторы цепочки не знают сессионных ключей других луковых маршрутизаторов. В-третьих, мы уже определились с тем, что наш проект уязвим сквозным атакам синхронизации; таким образом атаки разметкой, проведённые внутри цепочки, не дадут атакующему никакой дополнительной информации для атакующего.

Таким образом, мы будет производить проверку целостности только на границах каждого потока (не забудьте, что в нашей топологии протекающей трубы границей потока может быть любой переход цепочки). Так как Алиса согласует ключ на каждом новом переходе и каждый из собеседников инициализирует SHA-1 свёртку, производную от этого ключа, они начинают обмен с такого семени случайности, которое знают только они. Они оба инкрементально добавляют в SHA-1 свёртку содержимое всех передающих ячеек, которые они создают, и включают в каждую передающую ячейку первые четыре байта текущей свёртки. Каждая из сторон также хранит SHA-1 свёртку данных, которые она получила, чтобы удостовериться, что полученные хеши корректны.

Для того чтобы атакующий мог с уверенностью удалить или изменить ячейку, ему нужно определить текущее состояние свёртки (которое зависит от всего трафика, прошедшего между Алисой и Бобом, начиная с момента согласования их ключа). Атаки на SHA-1, в которых атакующий инкрементально добавляет значения и считает новый хеш, не проходят, так как все хеши проходят сквозное шифрование через цепочку. Вычислительная сложность получения свёртки минимальна, по сравнению с AES шифрованием, которое производится на каждом переходе по цепочке. Мы используем всего лишь четыре байта на ячейку, чтобы уменьшить вычислительную сложность. Шанс того, что атакующий сможет правильно угадать хеш допустимо малы, если учесть, что луковый прокси или маршрутизатор будет за-

вершать цепочку при получении неправильного хеша.

4.5 Ограничение скорости и справедливость

Добровольцы предпочитают использовать те службы, скорость передачи которых они могут ограничивать. Для того чтобы приспособиться к ним, серверы Tor используют подход «блока токенов» [46]. Он позволяет выдержать ограничение средней скорости входящих байтов на протяжении длительного времени, разрешая, однако, краткосрочные выбросы скорости передачи выше установленного предела.

Так как протокол Tor устроен так, что количество исходящего трафика приблизительно равно количеству входящего, обычно достаточно ограничить лишь количество входящих байт. С TCP-потоками, однако, ситуация несколько иная: передача единственного входного байта может потребовать целой ячейки в 512 байт (мы не можем просто ждать других байт, чтобы доставить их разом, поскольку локальное приложение может ждать ответа и не слать их больше). Таким образом, мы рассматриваем этот случай, как будто была прочитана вся ячейка независимо от её заполненности.

К тому же, следуя предложению Rennhard и др. [40], границы цепочки могут эвристически различать два различных типа потоков (интерактивные и массивные) путём сравнения частот, с которыми они создают ячейки. Мы можем обеспечить хорошую латентность для интерактивных потоков, предоставляя им привилегированное обслуживание, сохраняя при этом хорошую пропускная способность в целом для массивных потоков. Такая дискриминация потоков даёт врагу гипотетическую возможность проводить сквозные атаки, но на самом деле, атакующий, который может просматривать трафик на обоих концах может извлечь ту же самую информацию через атаки синхронизации.

4.6 Контроль перегрузки

Даже при использовании ограничения скорости передачи нам всё равно надо заботиться о перегрузке, которая может быть случайной или намеренной. Если достаточное количество пользователей выбирает одно и то же соединение (то есть одинаковую пару луковых маршрутизаторов) для своих цепочек, то соединение может стать насыщенным. Например, атакующий может послать большой файл через сеть Tor на свой web-сервер, а потом оказаться от чтения этой информации на другом конце цепочке. Без контроля перегрузки такие затруднения могут помешать работе всей сети. Нам не требуется изобретать TCP окна с нуля (с их последовательными номерами, возможностью пропуска часть ячеек в случае заполнения канала, с последующей их пересылкой и т.п.), так как TCP сам по себе гарантирует порядок доставки каждой из ячеек.

Ниже мы опишем решение этой проблемы.

Троттлинг уровня цепочки: Для того чтобы контролировать использование канала цепочки, каждый луковый маршрутизатор следит за двумя окнами. *Окно упаковки* отслеживает, сколько передающих ячеек с данными разрешено упаковать луковому маршрутизатору от исходящих TCP-потоков для передачи обратно луковому прокси. *Окно доставки* отслеживает, сколько передающих ячеек с данными хотят доставить TCP-потоки наружу сети. Каждое окно сначала инициализируется, скажем, 1000 ячейками данных. После того как данные пакуются или доставляются, соответствующее окно декрементируется. После того как луковый маршрутизатор получает достаточное количество ячеек данных (сейчас это значение равно 100), он посылает ячейку *передать самому себе* луковому прокси с нулевым streamID. Когда луковый маршрутизатор получает ячейку *передать самому себе* с нулевым streamID, он инкрементирует своё окно упаковки. Каждый из этих ячеек инкрементирует соответствующее окно на 100. Если окно упаковки достигает 0, луковый маршрутизатор прекращает читать данные от TCP соединений всех потоков, соответствующих цепочке, и не шлёт передающие ячейки с данными, пока не получит ячейку *передать самому себе*.

Луковый прокси работает схожим образом, за исключением того, что он должен поддерживать окно упаковки и окно доставки для каждого лукового маршрутизатора в цепочке. Если окно упаковки достигает 0, то оно перестаёт читать данные от потоков, адресованным данному луковому маршрутизатору.

Троттлинг уровня потока: Контроль перегрузки на уровне потока схож с механизмом уровня цепочки. Луковые маршрутизаторы и луковые прокси используют ячейки *передать самому себе* для того чтобы реализовать сквозной контроль отдельных TCP-потоков через цепочки. Для каждого потока в начальный момент времени выделяется окно упаковки (в данный момент с 500 ячейками), которое и увеличивается на фиксированное значение (50) при каждом получении ячейки *передать самому себе*. Вместо того чтобы всегда возвращать *передать самому себе*, как только поступит достаточное количество ячеек, контроль перегрузки уровня потока также обязан проверять, что данные были успешно сброшены в TCP-поток; он шлёт ячейку *передать самому себе*, только когда число ожидающих байтов, которые требуется сбросить, ниже определённого значения (на данный момент 10 ячеек).

Эти произвольно выбранные параметры похоже дают приемлемую пропускную способность и задержки.

5 Точки рандеву и скрытые службы

Точки рандеву являются основой для *служб со скрытым местоположением* (также известных как *анонимно отвечающих*). Службы со скрытым местоположением позволяют Бобу предоставлять TCP службы, такие как web-серверы, без раскрытия своего IP-адреса. Такой тип анонимности защищает от распределённых атак отказа в обслуживании: атакующие вынуждены атаковать всю луковую сеть целиком, так как они не знают IP-адреса Боба.

Наш проект серверов со скрытым местоположением преследует следующие цели.

Контроль доступа: У Боба должна быть возможность фильтровать входящие запросы, чтобы атакующий не мог завалить Боба просто открыв много соединений к нему.

Ошибкоустойчивость: Боб должен иметь возможность поддерживать длительную псевдоанонимную личность даже в случае с ошибок маршрутизатора. Служба Боба не должна быть привязана к конкретному луковому маршрутизатору, Боб обязан иметь возможность мигрировать с одного лукового маршрутизатора на другие.

Устойчивость к клевете: Нужно обеспечить защиту против атакующих, использующих социальную инженерию, которые стараются заклеймить маршрутизатор в ненадёжности. Они могут создавать нелегальные или сомнительные службы со скрытым местоположением и пытаться всех уверить, что эти службы были созданы этим маршрутизатором.

Прозрачность для приложений: Несмотря на то что пользователи должны использовать специальное приложение, чтобы иметь доступ к службам со скрытым местоположением, мы не должны заставлять их изменять пользовательские приложения.

Мы скрываем местоположение Боба, позволяя ему анонсировать себя на нескольких луковых маршрутизаторах сразу. Эти точки позволяют пользователям установить контакт с Бобом и называются *точками знакомства*. Он может сделать это, используя любое ошибкоустойчивое хранилище «ключ-значение» с возможностью аутентифицированного обновления, такие как распределённые хеш-таблицы (DHT) вроде CFS [11].[3] Алиса и клиент выбирают луковый маршрутизатор в качестве *точки рандеву*. Она подключается к одной из точек знакомства с Бобом, оповещает его о её точке рандеву, к которой он должен подключиться. Этот дополнительный уровень опосредованности помогает точкам знакомства Боба избежать проблем, связанных с непосредственным распространением непопулярных файлов. К примеру, Боб может публиковать материалы, которые вызывают неодобрение у сообщества, которое знает о этих точках знакомства, или службы Боба часто подвергаются атакам со стороны сетевых хулиганов. Эта опосредованность также позволяет Бобу отвечать на часть запросов, игнорируя оставшиеся.

5.1 Точки рандеву в Tor

Следующие шаги выполняются от имени Алисы и Боба их локальными луковыми прокси; интеграция с приложениями в более полном виде описана далее.

[3] Вместо того чтобы полагаться на внешнюю инфраструктуру луковая сеть может опираться на собственную службу поиска. Наша текущая реализация предоставляет простую систему поиска, размещаемую на серверах каталогов.

- Боб генерирует долгосрочный открытый ключ для идентификации его службы.
- Буб выбирает определённые точки знакомства, и создает их публикацию на службе поиска, подписанную его открытым ключом. Впоследствии он может добавить другие точки.
- Боб создаёт по цепочке для каждой точки знакомства и говорит им ждать запросов.
- Алиса узнаёт о службе Боба откуда-то извне (возможно, о службе ей сообщил Боб или она нашла её на сайте). Она узнаёт информацию о службе Боба через службу поиска. Если Алиса, посещая службу Боба, хочет остаться анонимной, ей необходимо соединиться со службой поиска через Tor.
- Алиса выбирает луковый маршрутизатор как точку рандеву для её соединения со службой Боба. Она создаёт цепочку до точки рандеву и назначает ей произвольно выбранные «куки [англ. cookie; примечание переводчика] встречи» для того чтобы опознать Боба.
- Алиса открывает анонимный поток на одну из точек знакомства и пишет через неё сообщение Бобу (зашифрованное его открытым ключом), в котором она рассказывает о себе, её точке рандеву. После этого она начинает квитирование протоколом Диффи—Хеллмана. Точка знакомства посылает это сообщение Бобу.
- Если Боб собирается общаться с Алисой, он создаёт цепочку до точки рандеву Алисы, посылает куки рандеву, вторую половину квитирования протоколом Диффи—Хеллмана и хеш их общего сессионного ключа. Алиса знает, что этот сессионный ключ знает только она с Бобом (см. объяснение в разделе 4.2).
- Точка рандеву соединяет цепочки Алисы и Боба в одну большую цепочку. Обратите внимание, что точка рандеву не может опознать ни Алису, ни Боба, ни информацию, которую они пересылают друг другу.
- Алиса посылает ячейку *начать передачу данных* по новой цепочке. Она приходит луковому прокси Боба, который соединён с его web-сервером.
- Анонимный поток был установлен, теперь Алиса и Боб могут общаться как ни в чём ни бывало.

При создании точки знакомства Боб предоставляет луковому маршрутизатору открытый ключ, идентифицирующий его службу. Боб подписывает свои сообщения, таким образом, никто не может посягать на его точку знакомства в будущем. Он использует тот же самый ключ для других точек знакомства и периодически обновляет свою запись в службе поиска.

Сообщение, которое Алиса отдаёт точке знакомства, содержит хеш открытого ключа Боба. Оно может также включать начальный токен авторизации. Точка знакомства может делать предварительную экспертизу, например, с целью блокировки повторных подключений. Её сообщение Бобу может также включать сквозной токен авторизации, чтобы Боб мог выбирать, отвечать или нет. Токены авторизации могут быть использованы для предоставления избирательного доступа: важным пользователям может быть предоставлен доступ без прерываний.

При отсутствии проблем сети служба Боба может просто предоставляться её зеркалами, в то время как Боб раздаёт токены высокоприоритетным пользователям. Если зеркала падают, то пользователи могут переключиться на другие серверы, предоставляющие службу Бобу через точки рандеву Tor.

Именно точки знакомства Боба являются объектами атак отказа в обслуживании. Он должен открыть как можно больше таких точек, чтобы снизить этот риск. Боб может предоставлять избранным пользователям текущий (или будущий) список неопубликованных точек знакомства. Это является наиболее практичным решением, когда имеется большая и стабильная группа точек знакомства. Боб может также предоставлять секретные открытые ключи для доступа к службе поиска. Все эти подходы сильно уменьшают риски, даже когда некоторые пользователи сговариваются и производят атаку отказа в обслуживании.

5.2 Интеграция с пользовательскими приложениями

Боб настраивает свой луковый прокси путём определения локального IP-адреса, порта его службы, стратегии авторизации клиентов и его

открытого ключа. Луковый прокси анонимно публикует на службе поиска подписанное ключом Боба заявление, срок действия и текущие точки знакомства службы Боба. Оно помещается в хеш таблицу под его открытым ключом. Web-сервер Боба остаётся нетронутым, ему даже не надо знать, что он находится в сети Tor.

Приложения Алисы также работают без изменений — они используют SOCKS прокси. Мы кодируем всю необходимую информацию в полное имя домена (FQDN), которое Алиса использует для установления соединения. Службы со скрытым местоположением используют специальный домен верхнего уровня, называемый `.onion`: таким образом, имена узлов принимают вид `x.y.onion` где `x` — это куки авторизации, а `y` — хеш публичного ключа. Луковый прокси Алисы проверяет адреса соединений; если точка назначения является скрытым сервером, то она декодирует ключ и начинает встречу, как описано выше.

5.3 Прошлые работы о рандеву

Точки рандеву в низко-латентных анонимизирующих системам были описаны впервые в ISDN телефонии [27, 34]. В более поздних низко-латентных проектах они использовались для скрытия местоположения мобильных телефонов и устройство отслеживания с низким потреблением [20, 36]. Рандеву для анонимизации низко-латентных интернет-соединений был предложены в ранних проектах лукового маршрутизатора [24], но первый проект был опубликован Ian Goldberg [23]. Его проект отличается от нашего в трёх вещах. Во-первых, Goldberg предлагает Алисе самостоятельно искать текущее местоположение службы через Gnutella; наш подход делает поиск служб быстрее, устойчивее к ошибкам и прозрачнее для пользователя. Во-вторых, клиент с сервером согласуют сессионные ключи с помощью протокола Диффи—Хеллмана, поэтому информация не раскрывается в открытом виде даже в точках рандеву. В-третьих, наш проект минимизирует риски от запущенной службы, что побуждает добровольцев предоставлять точки знакомства и рандеву для служб. В Tor точки знакомства не передают никаких данных клиентам; они не знают ни клиента, ни сервера; они не могут просматривать передаваемую информацию. Эта схема опосредованности также служит для аутентификации и авторизации — если Алиса не включает нужные куки в её запрос к серверу, Боб не должен даже и сообщать о своём существовании.

6 Другие проектные решения

6.1 Отказ в обслуживании

Предоставление Tor как публичной службы даёт много возможностей для совершения атак отказа в обслуживании против сети. Несмотря на то что управление потоками и ограничение скорости (которые обсуждались в разделе 4.6) не даёт пользователям возможности расходовать больше трафика, чем маршрутизаторы собираются им давать, существуют другие возможности проводить атаки. Например, пользователи могут использовать больше сетевых ресурсов, чем им полагается, или нарушать работу сети для других пользователей.

Прежде всего, есть несколько видов атак отказа в обслуживании, в которых атакующий может заставить луковый маршрутизатор производить трудоёмкие для ЦП криптографические операции. Например, атакующий может без особых вычислительных затрат сделать вид, что он начинает TLS квитирование, заставляя луковый маршрутизатор обрабатывать его половину (относительно дорогую в вычислительном плане) квитирования.

Мы пока не научились защищаться от атак такого типа, однако у нас есть на примете несколько способов. Прежде всего, луковый маршрутизатор может заставить решать клиентов головоломку [14] перед началом TLS квитирования или при приёме ячеек *создать*. Так как такие токены легко проверить и относительно сложно вычислить, этот подход существенно повышает вычислительную сложность атаки. Также луковый маршрутизатор может ограничить частоту, с которой он принимает ячейки *создать* и входящие TLS соединения. Таким образом вычислительные ресурсы будут использоваться больше на передачу ячеек, а не на симметричную криптографию, что не позволит оттянуть все ресурсы атакой отказа в обслуживании. Это ограничение, однако, позволит атакующему снизить скорость создания ячеек для других пользователей.

Неприятели могут также атаковать сетевые соединения и узлы Tor. Прерывание единственной цепочки или соединения разрушает все потоки, которые проходили через эту часть цепочки. Обслуживание пользователей также прекращается при поломке маршрутизатора или в момент его перезапуска. В данный момент Tor считает такие атаками перебоями в сети и ждёт, что пользователи и приложения ответят или восстановятся сами в нужный момент. В дальнейшем мы, возможно, будем использовать что-то вроде сквозного TCP протокола подтверждения, чтобы никакие потоки не были потеряны, если сущность и точка выхода не были повреждены. Этот подход требует большего количества буферизации на концах сети. Более того, вопросы производительности и анонимности от введения этой дополнительной сложности требуют особого рассмотрения.

6.2 Политики точки выхода злоумышленное использование

Злоумышленное использование является большой преградой для широкомасштабного развёртывания Tor. Анонимность дарит возможность хулиганам и злоумышленникам замести следы своих преступлений. Атакующие могут повредить сети Tor тем, что вовлекут серверы выхода в свои противоправные действия. Также многие приложения, такие как корпоративные почтовые или web-серверы, используют аутентификацию, основанную на IP-адресах, что может привести к недоразумению, в случае с анонимными соединениями, исходящими от выходных луковых маршрутизаторов.

Мы подчёркиваем, что Tor не открывает никакого нового способа совершать противоправные действия. Спаммеры и другие атакующие уже имеют доступ к тысячам неправильно настроенных систем по всему миру. К тому же, сеть Tor далеко не самое простое место для совершения атак.

Но так как луковые маршрутизаторы могут быть по ошибке приняты за зачинщика противоправных действий, а добровольцы, которые запустили их, могут не захотеть объяснять тонкости анонимных сетей разгневанным системным администраторам, мы должны заблокировать или ограничить противоправные действия, происходящие в сети Tor.

Для того чтобы минимизировать проблемы злоумышленного использования, *политики точек выхода* каждого лукового маршрутизатора описывают те внешние адреса и порты, к которым маршрутизатор будет подключаться. На одном конце спектра возможностей стоят узлы *открытого выхода*, которые будут подключаться ко всему. На другом конце стоят узлы-*посредники*, которые будут только передавать трафик другим узлам из сети Tor, и *частные выходы*, которые будут соединяться только с компьютерами из локальной сети. Частный выход позволяет клиенту соединяться с заданным компьютером или сетью безопаснее — внешний неприятель не может перехватывать трафик между частным выходом и конечной точкой назначения, поэтому он знает меньше о том, что делает Алиса, и о её назначении. Большая часть луковых маршрутизаторов в данный момент работает как ограниченные выходы, которые выпускают соединения для всего мира, но блокируют доступ к некоторым подверженным взлому адресам и службам, таким как SMTP. Луковый маршрутизатор также может так аутентифицировать клиентов, чтобы исключить возможность злоупотребления точками выхода без вреда для анонимности [44].

Многие администраторы блокируют все порты, кроме избранных, чтобы работали только определённые службы, такие как HTTP, SSH или AIM. Это решение явно неполно, так как злоумышленное использование этих протоколов всем хорошо известно.

Мы пока не ощущаем никаких противоправных действий в нашей сети, но мы должны рассмотреть возможность использования прокси для очистки трафика, покидающего сеть, определённых протоколов. Например, легко может быть обнаружена большая часть злоумышленного HTTP трафика (вызывающего переполнения стека, использующего скриптовые уязвимости и т.п.). Люди также могут устанавливать средства для автоматической фильтрации спама (такие как SpamAssassin) на выходах из луковой сети.

Луковые маршрутизаторы могут также изменять выходящий трафик, дописывая в заголовки или другие места информацию о том, что этот трафик прошёл через анонимную сеть. Этот подход широко используется

для анонимизирующих систем, специализирующихся только на почтовых сообщениях. Луковые маршрутизаторы также могут запускаться на серверах в доменной зоне `anonymous`, для того чтобы оповестить жертв злоумышленника об анонимной природе трафика.

Смесь из открытых и ограниченных узлов выхода является наиболее гибкой для добровольцев, запускающих серверы. Использование большого количества узлов-посредников обеспечивает большую и ошибкоустойчивую сеть. Вместе с тем, малое количество узлов выхода уменьшает количество точек, за которыми должен следить неприятель с целью анализа трафика, а также увеличивает ношу каждой из них. Это противоречие хорошо видно в каскадной модели Java Anon Proxy, в котором только один узел из каждого каскада должен справляться жалобами на злоумышленников. Из-за этого неприятелю требуется просматривать только входы и выходы каскада, чтобы осуществлять анализ трафика всех пользователей из каскада. Модель гидры (много входов, мало выходов) представляет другой компромисс: в нём требуется определённое количество выходных узлов, но это заставляет неприятеля затрачивать больше ресурсов, чтобы усмотреть за всеми клиентами; см. раздел 8.

Наконец, мы должны отметить, что злоумышленное использование выходных узлов не является мелкой проблемой. Если система выглядит неважно в глазах общественности, это уменьшает количество и разнообразие пользователя, что приводит к тому, что система становится менее анонимной. Таким образом, признание общественности, так же как и простота использования, — это требование безопасности. К сожалению, защита от противоправных действий на открытых выходных узлах является нерешённой проблемой и похоже в ближайшем будущем в этом направлении будет вестись гонка вооружений. Проблемы злоумышленного использования рассмотрены в проекте CoDeeN Принстонского университета [33].

6.3 Серверы каталогов

В проекте луковой маршрутизации первого поколения [8, 37] были внутриполосные оповещения об изменении статуса: каждый маршрутизатор отправлял подписанные заявления своим соседям, которые распространялись дальше. Но анонимные сети преследуют иные цели безопасности, нежели обычные протоколы маршрутизации. Например, задержки (случайные или намеренные), которые могут вызвать разные представления о состоянии и топологии сети представляют не только неудобства. Они дают атакующим эксплуатировать уязвимости в отличии между знаниями различных клиентов. Мы также заботимся об атаках, нацеленных на то чтобы ввести клиентов в заблуждение о списке пользователей маршрутизатора, топологии или текущем состоянии сети. Такие *атаки разбиения* клиентского знания помогают неприятелю эффективно доставлять требуемые ресурсы заданной жертве [13].

Tor использует малое количество дублированных, проверенных луковых маршрутизаторов для отслеживания изменения топологии сети и состояний узлов (включая ключи и политики точки выхода). Каждый *сервер каталогов* работает как HTTP сервер, который может быть просмотрен клиентами для получения списка маршрутизаторов и состояния сети и обновлён другими луковыми маршрутизаторами. Луковые маршрутизаторы периодически публикуют подписанные заявления об их состоянии на всех серверах. Серверы каталогов объединяют эту информацию со своими представлениями о состоянии сети и генерируют подписанное описание (*каталог*) состояния всей сети. Клиентские приложения поставляются со вшитым списком серверов каталогов и их ключами, для того чтобы произвести начальную загрузку клиентского представления сети.

Когда сервер каталогов получает подписанное заявление лукового маршрутизатора, он проверяет идентификационный ключ этого маршрутизатора. Серверы каталогов не будут публиковать неопознанные луковые маршрутизаторы. В противном случае неприятель мог бы получить контроль над сетью путём создания большого количества серверов [19]. Наоборот, новые узлы должны подтверждаться администратором сервера каталогов перед их включением в список. Механизмы для автоматизированного подтверждения являются предметом активных исследований.

Конечно, остаётся множество возможных атак. Неприятель, который контролирует сер-

вер каталогов, может отслеживать своих клиентов, предоставляя им различную информацию, например, оглашая им список только из контролируемых им узлов или информируя только определённых клиентов о выбранном узле. Даже внешний неприятель может эксплуатировать уязвимости, связанные с разницей в знании клиентов: клиенты, которые используют узел, опубликованный лишь в одном сервере каталогов, уязвимы.

Таким образом, серверы каталогов обязаны быть синхронизированными и продублированными, то есть они должны публиковать один и тот же каталог. Клиенты должны доверять только каталогам, которые подписаны подавляющим большинством серверов каталогов.

Серверы каталогов Tor были созданы по подобию серверов каталогов Mixminion [13], но в нашем случае ситуация проще. Во-первых, с целью упрощения мы делаем предположение, что все участники сети согласны о наборе серверов каталогов. Во-вторых, в то время как Mixminion должен предугадывать состояние узлов, Tor требуется лишь согласие критического большинства о текущем состоянии сети. В-третьих, мы предполагаем, что в тех случаях, когда консенсус о каталоге не может быть достигнут, мы можем прибегнуть к помощи администраторов, которые разрешат эту проблему вручную. Так как серверов каталогов мало (на данный момент их три штуки, но мы думаем, что увеличим их количество до девяти, когда сеть вырастет), мы можем себе позволить использовать широковещательную рассылку для упрощения протокола достижения консенсуса.

Для того чтобы избежать атак, в которых маршрутизатор соединён со всеми серверами каталогов, но отказывается передавать трафик другим маршрутизаторам, серверы каталогов должны создавать цепочки и использовать их для анонимной проверки надёжности маршрутизатора. [16]. К сожалению, эта защита пока не реализована.

Использование серверов каталогов проще и более надёжно, чем проведение лавинной маршрутизации. Лавинная маршрутизация дорого обходится и усложняет анализ, когда мы начинаем экспериментировать с топологиями сети, в которых связи сети представляют неполный подграф. Подписанные каталоги могут кэшироваться другими луковыми маршрутизаторами, то есть серверы каталогов не являются узким местом сети при большом количестве пользователей. Так же мы не упрощаем анализ трафика тем, что заставляем клиентов сообщать о своём существовании какому-либо центральному узлу.

7 Защиты от атак

Ниже мы кратко изложим разнообразие атак и обсудим, в какой мере наш проект справляется с ними.

Пассивные атаки

Отслеживание шаблонов пользовательского трафика. Путём отслеживания соединений пользователя нельзя раскрыть его назначения или данных, но можно выявить шаблоны трафика (как полученного, так и отправленного). Профилирование шаблонов пользовательских соединений требует последующей обработки, так как в одно и то же время на одной цепочке может быть открыто несколько потоков различных приложений.

Отслеживание контента пользователя. В то время как весь контент на пользовательском конце является зашифрованным, соединения с отвечающими сторонами могут такими не являться. В самом деле, отвечающий сайт может быть неприятельским. Так как фильтрация контента не является основной целью проекта луковой маршрутизации, Tor может непосредственно использовать Privoxy и другие фильтрующие службы для анонимизации потоков данных приложений.

Отличимость опций. Мы позволяем клиентам выбирать различные опции конфигурации. Например, клиенты, озабоченные возможностью установления связей между запросами должны менять цепочки чаще, чем те, кто обеспокоен, чтобы за ним не следили. Возможность выбора может привлекать пользователей с различными нуждами. Вместе с тем, те пользователи, которые настроили Tor под себя, выявляют себя на фоне большинства [1].

Сквозные корреляции синхронизации. Tor прячет такие корреляции по минимуму. Атакующий, ищущий шаблоны трафика инициатора и отвечающего, сможет подтвердить зависимость с большой вероятностью. На данный

момент наиболее сильной защитой от атак такого типа является скрытие соединения между луковым прокси и первым узлом Tor путём запуска лукового прокси на узле Tor или за межсетевым экраном. Этот подход требует от обозревателя разделения трафика на тот, который исходит непосредственно от лукового маршрутизатора, и тот, который лишь проходит через него. Глобальный обозреватель может сделать это, но это может быть за пределами возможностей ограниченного обозревателя.

Сквозные корреляции размеров. Простой подсчёт количества пакетов может быть эффективным для подтверждения конечных точек потока. Однако даже без выравнивания у нас есть частичная защита от этого: топология протекающей трубы означает, что количество пакетов, вошедших с одной стороны цепочки, может не совпадать с количеством пакетов, покинувших её с другой.

Отпечатки web-сайтов. Все работающие пассивные атаки, перечисленные выше, являются атаками подтверждения трафика. Защита от таких атак не входит в цели, определённые проектом. Есть также другая атака подтверждения трафика, которая потенциально может быть эффективной. Вместо того чтобы искать корреляции синхронизации и объёма на выходах, неприятель может собрать базу данных из «отпечатков», содержащую размеры файлов и шаблоны доступа для заданных web-сайтов. В дальнейшем он может проверить пользовательское соединение к заданному сайту путём простой проверки по базе данных. Эта атака оказалась эффективной против SafeWeb [26]. Вероятно, она менее эффективна против Tor, так как потоки мультипликсируются на одну и ту же цепочку и эффективность использования отпечатков будет ограничена гранулярностью ячеек (в данный момент 512 байт). Другие дополнительные защиты: увеличение размера ячеек, различные способы выравнивания, которые группируют сайты в большие кластеры, выравнивание на канальном уровне и создание пустышек на длинные дистанции. [4]

[4]Внимание, не путайте эту атаку на отпечатки с гораздо более сложными атаками на латентности, нацеленными на выявление пользователя и отвечающего сайта [5], которые требует отпечатки латентностей всех цепочек сети, вместе с отпечатками на границах сети.

Активные атаки

Компрометация ключей. Атакующий, узнавший сессионный TLS ключ, может видеть управляющие ячейки и расшифрованные передающие ячейки из каждой цепочки этого соединения. Знание сессионного ключа цепочки позволяет ему снять один слой шифрования. Атакующий, узнавший закрытый TLS ключ лукового маршрутизатора, может выдавать себя за него в течение всего срока действия этого ключа. Однако он также должен узнать луковый ключ для расшифровки ячеек *создать*. Из-за совершенной прямой секретности он не может взломать уже созданные цепочки без компрометации их сессионных ключей. Периодическая замена ключей ограничивает действие этих атак. С другой стороны, атакующий, узнавший идентификационный ключ узла, может навсегда заменить узел путём посылки новых подделанных дескрипторов на сервер каталогов.

Итеративная компрометация. Передвигающийся неприятель, который умеет компрометировать луковые маршрутизаторы (путём вторжения в систему, правового или неправового принуждения), может двигаться вниз по цепочке, компрометируя все узлы, пока он не дойдёт до её конца. Однако, в случае если неприятель не успеет завершить атаку в течение срока действия цепочки, луковые маршрутизаторы сбросят необходимую информацию до того, как атака будет завершена (благодаря совершенной прямой секретности сессионных ключей атакующий не может заставить узлы расшифровать записанный трафик после закрытия цепочек). К тому же, создание цепочек, которые пересекают зоны юрисдикции, усложняют использование правового принуждения. Это явление носит название «подсудный арбитраж». Проект Java Anon Proxy недавно почувствовал необходимость этого подхода, когда немецкий суд заставил их внедрить бэкдор на свои узлы [47].

Запуск получателя. Неприятель, который имеет запущенный web-сервер, элементарным образом выясняет шаблоны синхронизации пользователей, подключающихся к нему, и может внести произвольные задержки при ответе на их запросы. Так проще осуществлять сквозные атаки: если неприятель может добиться

того чтобы пользователи подключались к его web-серверу (например, предоставляя контент, нацеленный на данных пользователей), то он держит один из концов их соединения. Есть также опасность, что протоколы приложений и связанные с ними программы могут раскрыть информацию об инициаторе соединения. Tor полагается на Privoxy и другие программы для чистки протоколов для решения этой проблемы.

Запуск лукового прокси. Предполагается, что почти всегда конечные пользователи будут держать у себя запущенный локальный луковый прокси. Однако в некоторых случаях может возникнуть необходимость использования удалённого прокси. Обычно это происходит, когда организация хочет контролировать активность пользователей прокси. Компрометация лукового прокси компрометирует все будущие соединения, проходящие через него.

Отказ в обслуживании необозреваемых узлов. Обозреватель, который может смотреть за частью сети Tor, может увеличить свою долю в общем трафике путём атаки на необозреваемые узлы с целью их выключения, понижения их надёжности или убеждении пользователей в их ненадёжности. Лучшая защита в данном случае — ошибкоустойчивость.

Запуск вредоносного лукового маршрутизатора. Помимо простого прослушивания изолированный вредоносный узел может сам создавать цепочки или изменять шаблоны трафика, для того чтобы влиять на трафик других узлов. В любом случае, вредоносный узел должен быть непосредственным соседом обоих конечных точек, для того чтобы компрометировать анонимность цепочки. Если неприятель может контролировать несколько луковых маршрутизаторов и убедить серверы каталогов в том, что они являются доверенными и независимыми, то некоторые пользователи могут случайно выбрать один из этих маршрутизаторов как начало, а другой как конец цепочки. Если неприятель контролирует $m > 1$ из N узлов, в лучшем случае он может найти корреляцию $\left(\frac{m}{N}\right)^2$ трафика. С другой стороны, неприятель может привлечь непропорционально большое количество трафика из-за использования разрешительной политики точки выхода или уменьшением надёжности других маршрутизаторов.

Внесение синхронизации в сообщения. Эта атака является более сильным вариантом пассивной атаки синхронизации, которая обсуждалась ранее.

Атаки разметкой. Вредоносный узел может «помечать» ячейки путём их замены. Если поток представлял собой, к примеру, незашифрованный запрос к сайту, искажённый контент, пришедший в ответ в нужный момент, подтвердит связь. Однако проверка целостности ячеек защищает от таких атак.

Замена контента протоколов без аутентификации. Когда передаётся протокол без аутентификации, такой как HTTP, вредоносный узел выхода может выдать себя за сервер назначения. Клиенты должны использовать протоколы со сквозной аутентификацией.

Атаки повтора передачи. Некоторые протоколы анонимности уязвимы атакам повтора передачи. Tor неуязвим: повтор передачи половины квитирования приведёт к другому сессионному ключу, то есть остаток записанной сессии не сможет быть использован.

Клеветнические атаки. Атакующие могут использовать сеть Tor для социально осуждаемых действий, даря сети и её связистам дурную славу, для того чтобы закрыть её. Политики точки выхода уменьшают возможности злоумышленного использования, но в целом, сеть требует добровольцев, которые могут терпеть политические страсти.

Распространение вредоносного кода. Атакующий может обмануть пользователей и вынудить их запустить поддельное программное обеспечение Tor, которое, на самом деле не анонимизирует их соединения, или даже хуже, может обмануть луковые маршрутизаторы, чтобы те предоставляли пользователям меньше анонимности. Мы пытаемся бороться с этой проблемой (хотя она до конца так и не решена) путём подписывания всех выпусков Tor официальным открытым ключом и включением в каталог списка всех версий, которые считаются безопасными на данный момент. Для защиты от самой подделки официального выпуска (через уязвимости, взяточничество или путём внедрения) мы предоставляем все выпуски в виде исходного кода, поощряем аудит и часто предупреждаем пользователей, чтобы они не верили никакому программному обеспечению (даже нашему), которое распро-

страняется без исходного кода.

Атаки на каталоги

Уничтожение серверов каталогов. Если несколько серверов исчезнут, оставшиеся всё равно смогут договориться о правильном каталоге. До тех пор пока работают любые серверы каталогов, они могут производить широковещательную рассылку их представлений о сети, приходить консенсусу и генерировать общий каталог. Однако, если больше половины серверов каталогов было уничтожено, то каталог не будет иметь достаточно подписей, для того чтобы клиенты могли использовать его автоматически. В таком случае необходимо человеческое вмешательство, чтобы определить, следует ли доверять конечному каталогу.

Подделка сервера каталогов. Атакующий, захвативший один из серверов каталогов, имеет частичное влияние на конечный каталог. Так как луковые маршрутизаторы включаются или исключаются большинством голосов, только в случае ничьи подделанный сервер каталогов может испортить каталог. Вопрос в том, как часто могут происходить такие случаи на практике.

Подделка большей части серверов каталогов. Неприятель, который владеет большей частью серверов каталогов, имеет полный контроль над списком луковых маршрутизаторов. Мы должны обеспечить то, чтобы связисты серверов были независимы и защищены от атак.

Побуждение к расколу серверов каталогов. Протокол выбора каталога предполагает, что связисты серверов каталогов согласны о наборе сервер каталогов. Неприятель, который убедит связистов серверов каталогов не доверять друг другу, может разбить их на враждебные лагеря. Тем самым он заставит пользователей выбирать, кому доверяют они. Tor не рассматривает такие атаки.

Размещение вредоносного лукового маршрутизатора обманом серверов каталогов. Наша модель уязвимостей явным образом подразумевает, что связисты серверов каталогов могут отфильтровать большую часть вредоносных луковых маршрутизаторов.

Убеждение серверов каталогов в том, что работают повреждённые луковые маршрутизаторы. В текущей реализации Tor серверы каталогов считают, что луковый маршрутизатор работает корректно, если они могут установить с ним TLS соединение. Вредоносный луковый маршрутизатор может с лёгкостью обмануть этот тест, если примет входящие TLS соединения от луковых маршрутизаторов, но будет игнорировать все ячейки. Серверы каталогов должны активно тестировать луковые маршрутизаторы на то, что они создают цепочки и потоки когда нужно. Побочные эффекты схожего подхода обсуждаются в [16].

Атаки против точек рандеву

Создание большого количества запросов знакомства. Атакующий может попытаться заблокировать службу Боба заваливанием его точек знакомства запросами. Однако, так как точки знакомства могут блокировать запросы, которые не имеют токенов авторизации, Боб может ограничить объём получаемых запросов. Также он может требовать осуществить определённый объём вычислений на каждый запрос, который он получает.

Атаки на точки знакомства. Атакующий может подорвать службу со скрытым местоположением путём блокирования его точек знакомства. Но так как личность службы привязана к её открытому ключу, служба может просто опубликовать себя на другой точке знакомства. Публикация службы может быть секретной, так чтобы только высокоприоритетные клиенты знали адреса точек знакомства Боба или чтобы различные клиенты знали о разных точках знакомства. Это заставляет атакующего отключать все возможные точки знакомства.

Компрометация точки знакомства. Атакующий, который контролирует точки знакомства Боба, может завалить его запросами знакомства или, наоборот, не доставить ему корректные запросы. Боб может заметить это и закрыть цепочку. Есть однако сложность в том, чтобы отследить доставку корректных запросов. Боб должен периодически проверять точку знакомства, посылая запросы на рандеву и проверяя, что он получает их.

Компрометация точек рандеву. Точки рандеву в общем-то ничем не отличаются от любых других луковых маршрутизаторов в цепочке, так как все данные, проходящие через

них зашифрованы общим сессионным ключом Алисы и Боба.

8 Направления дальнейшего развития

Tor совмещает в себе множество инноваций в единой разворачиваемой системе. Следующие шаги развития включают:

Масштабируемость: Tor акцентирует внимание на простоте развёртывания. Простота проекта позволила нам адаптировать для клиентского знания топологию полного подграфа, полуцентрализованные каталоги и модель полной видимости сети. Эти свойства не будут масштабироваться более чем на несколько сотен серверов.

Деление луковых маршрутизаторов на классы по скорости: В этой статье предполагалось, что все луковые маршрутизаторы имеют хорошую скорость передачи и низкую латентность. Было бы неплохо применить модель MorphMix, в которой узлы публикуют свой класс скорости передачи (DSL, T1, T3). Это позволит Алисе избежать узких мест, так как она будет выбирать те узлы, скорость которых выше или совпадает с её. Это позволит пользователям с DSL соединениям приносить пользу сети Tor.

Материальная мотивация: Есть мнение, что если награждать и прославлять добровольцев, то они будут обеспечивать ещё большую анонимность [1]. Чем больше узлов, тем больше сеть. Большое количество пользователей означает, что каждый из них более анонимен. Мы должны продолжать придумывать способы поощрения за участие в Tor. В дальнейшем мы должны найти больше способов ограничения злоумышленного использования и понять, почему большую часть людей не волнуют системы сохранения приватности.

Покрытие трафика: В данный момент Tor не покрывает трафик — ясно, что это несёт расходы в производительности и в скорости передачи, но непонятно, какие преимущества появляются с точки зрения безопасности. Мы должны произвести исследования о покрытии трафика на канальном уровне и на длинные дистанции, чтобы понять, поможет ли какой-нибудь простой метод выравнивания защититься от выбранного неприятеля.

Кэширование на узлах выхода: Возможно, каждый узел выхода должен держать кэширующий web-прокси [43]. Это позволит увеличить анонимность закэшированных страниц (запросы Алисы не будут покидать сеть Tor), увеличить скорость загрузки и уменьшить стоимость за доступ к интернету. С другой стороны, это повлечёт за собой ослабление прямой секретности, так как в кэше сохраняются принятые файлы. Мы должны отыскать баланс между простотой использования и безопасностью.

Лучшее распространение каталогов: Сейчас клиенты скачивают полное описание сети каждые 15 минут. Когда сеть вырастет, а число клиентов увеличится, нам потребуется решение, в котором клиенты получают инкрементальные обновления состояния каталога. В более общем виде, мы должны найти более масштабируемые, но в то же время удобные, способы распространять свежие снимки состояния сети без внесения возможности для новых атак.

Дальнейшая оценка спецификации: Наша открытая спецификация на уровне байтов [18] требует внешней оценки. Мы надеемся, чем крупнее будет Tor, тем больше человек оценят его спецификацию.

Межоперационная совместимость: На данный момент мы совместно работаем с создателем MorphMix над унификацией спецификации и реализации общих элементов наших двух систем. Пока это выглядит достаточно простым. Межоперационная совместимость позволит проверять и проводить непосредственное сравнение на доверие и масштабируемость этих двух проектов.

Широкое развёртывание: Первоначальной целью Tor было получение опыта в развёртывании анонимной оверлейной сети, в том числе от реальных пользователей. Сейчас мы находимся на такой стадии разработки, что готовы развёртывать дальше. Когда в нашей сети появится больше пользователей мы без сомнений сможем лучше оценить наши проектные решения, включая компромиссы между ошибкоустойчивостью и латентностью, уступки, касающиеся производительности (в том числе размеры ячеек), нашу систему противодействия злоумышленному использованию и общей простоте использования.

Благодарности

Мы благодарим Peter Palfrader, Geoff Goodell, Adam Shostack, Joseph Sokol-Margolis, John Bashinski, and Zack Brown за редактирование и комментарии; Matej Pfajfar, Andrei Serjantov, Marc Rennhard за обсуждения проекта; Bram Cohen за обсуждения контроля перегрузки; Adam Back за то, что он предложил кладные цепочки; и Cathy Meadows за формальный анализ протокола *расширения*. Эта работа была поддержана ONR и DARPA.

Список литературы

[1] A. Acquisti, R. Dingledine, and P. Syverson. On the economics of anonymity. In R. N. Wright, editor, *Financial Cryptography*. Springer-Verlag, LNCS 2742, 2003.

[2] R. Anderson. The eternity service. In *Pragocrypt '96*, 1996.

[3] The Anonymizer. `<http://anonymizer.com/>`.

[4] A. Back, I. Goldberg, and A. Shostack. Freedom systems 2.1 security issues and analysis. White paper, Zero Knowledge Systems, Inc., May 2001.

[5] A. Back, U. Möller, and A. Stiglic. Traffic analysis attacks and trade-offs in anonymity providing systems. In I. S. Moskowitz, editor, *Information Hiding (IH 2001)*, pages 245–257. Springer-Verlag, LNCS 2137, 2001.

[6] M. Bellare, P. Rogaway, and D. Wagner. The EAX mode of operation: A two-pass authenticated-encryption scheme optimized for simplicity and efficiency. In *Fast Software Encryption 2004*, February 2004.

[7] O. Berthold, H. Federrath, and S. Köpsell. Web MIXes: A system for anonymous and unobservable Internet access. In H. Federrath, editor, *Designing Privacy Enhancing Technologies: Workshop on Design Issue in Anonymity and Unobservability*. Springer-Verlag, LNCS 2009, 2000.

[8] P. Boucher, A. Shostack, and I. Goldberg. Freedom systems 2.0 architecture. White paper, Zero Knowledge Systems, Inc., December 2000.

[9] Z. Brown. Cebolla: Pragmatic IP Anonymity. In *Ottawa Linux Symposium*, June 2002.

[10] D. Chaum. Untraceable electronic mail, return addresses, and digital pseudo-nyms. *Communications of the ACM*, 4(2), February 1981.

[11] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica. Wide-area cooperative storage with CFS. In *18th ACM Symposium on Operating Systems Principles (SOSP '01)*, Chateau Lake Louise, Banff, Canada, October 2001.

[12] W. Dai. Pipenet 1.1. Usenet post, August 1996. `<http://www.eskimo.com/~weidai/pipenet.txt>` First mentioned in a post to the cypherpunks list, Feb. 1995.

[13] G. Danezis, R. Dingledine, and N. Mathewson. Mixminion: Design of a type III anonymous remailer protocol. In *2003 IEEE Symposium on Security and Privacy*, pages 2–15. IEEE CS, May 2003.

[14] D. Dean and A. Stubblefield. Using Client Puzzles to Protect TLS. In *Proceedings of the 10th USENIX Security Symposium*. USENIX, Aug. 2001.

[15] T. Dierks and C. Allen. The TLS Protocol — Version 1.0. IETF RFC 2246, January 1999.

[16] R. Dingledine, M. J. Freedman, D. Hopwood, and D. Molnar. A Reputation System to Increase MIX-net Reliability. In I. S. Moskowitz, editor, *Information Hiding (IH 2001)*, pages 126–141. Springer-Verlag, LNCS 2137, 2001.

[17] R. Dingledine, M. J. Freedman, and D. Molnar. The free haven project: Distributed anonymous storage service. In H. Federrath, editor, *Designing Privacy Enhancing Technologies: Workshop on Design Issue in Anonymity and Unobservability*. Springer-Verlag, LNCS 2009, July 2000.

[18] R. Dingledine and N. Mathewson. Tor protocol specifications. `<https://www.torproject.org/svn/trunk/doc/tor-spec.txt>`.

[19] J. Douceur. The Sybil Attack. In *Proceedings of the 1st International Peer To Peer Systems Workshop (IPTPS)*, Mar. 2002.

[20] H. Federrath, A. Jerichow, and A. Pfitzmann. MIXes in mobile communication systems: Location management with privacy. In R. Anderson, editor, *Information Hiding, First International Workshop*, pages 121–135. Springer-Verlag, LNCS 1174, May 1996.

[21] M. J. Freedman and R. Morris. Tarzan: A peer-to-peer anonymizing network layer. In *9th ACM Conference on Computer and Communications Security (CCS 2002)*, Washington, DC, November 2002.

[22] S. Goel, M. Robson, M. Polte, and E. G. Sirer. Herbivore: A scalable and efficient protocol for anonymous communication. Technical Report TR2003-1890, Cornell University Computing and Information Science, February 2003.

[23] I. Goldberg. *A Pseudonymous Communications Infrastructure for the Internet*. PhD thesis, UC Berkeley, Dec 2000.

[24] D. M. Goldschlag, M. G. Reed, and P. F. Syverson. Hiding routing information. In R. Anderson, editor, *Information Hiding, First International Workshop*, pages 137–150. Springer-Verlag, LNCS 1174, May 1996.

[25] C. Gülcü and G. Tsudik. Mixing E-mail with Babel. In *Network and Distributed Security Symposium (NDSS 96)*, pages 2–16. IEEE, February 1996.

[26] A. Hintz. Fingerprinting websites using traffic analysis. In R. Dingledine and P. Syverson, editors, *Privacy Enhancing Technologies (PET 2002)*, pages 171–178. Springer-Verlag, LNCS 2482, 2002.

[27] A. Jerichow, J. Müller, A. Pfitzmann, B. Pfitzmann, and M. Waidner. Real-time mixes: A bandwidth-efficient anonymity protocol. *IEEE Journal on Selected Areas in Communications*, 16(4):495–509, May 1998.

[28] D. Koblas and M. R. Koblas. SOCKS. In *UNIX Security III Symposium (1992 USENIX Security Symposium)*, pages 77–83. USENIX, 1992.

[29] B. N. Levine, M. K. Reiter, C. Wang, and M. Wright. Timing analysis in low-latency mix-based systems. In A. Juels, editor, *Financial Cryptography*. Springer-Verlag, LNCS (forthcoming), 2004.

[30] B. N. Levine and C. Shields. Hordes: A multicast-based protocol for anonymity. *Journal of Computer Security*, 10(3):213–240, 2002.

[31] C. Meadows. The NRL protocol analyzer: An overview. *Journal of Logic Programming*, 26(2):113–131, 1996.

[32] U. Möller, L. Cottrell, P. Palfrader, and L. Sassaman. Mixmaster Protocol — Version 2. Draft, July 2003. <http://www.abditum.com/mixmaster-spec.txt>.

[33] V. S. Pai, L. Wang, K. Park, R. Pang, and L. Peterson. The Dark Side of the Web: An Open Proxy's View. <http://codeen.cs.princeton.edu/>.

[34] A. Pfitzmann, B. Pfitzmann, and M. Waidner. ISDN-mixes: Untraceable communication with very small bandwidth overhead. In *GI/ITG Conference on Communication in Distributed Systems*, pages 451–463, February 1991.

[35] Privoxy. <http://www.privoxy.org/>.

[36] M. G. Reed, P. F. Syverson, and D. M. Goldschlag. Protocols using anonymous connections: Mobile applications. In B. Christianson, B. Crispo, M. Lomas, and M. Roe, editors, *Security Protocols: 5th International Workshop*, pages 13–23. Springer-Verlag, LNCS 1361, April 1997.

[37] M. G. Reed, P. F. Syverson, and D. M. Goldschlag. Anonymous connections and onion routing. *IEEE Journal on Selected Areas in Communications*, 16(4):482–494, May 1998.

[38] M. K. Reiter and A. D. Rubin. Crowds: Anonymity for web transactions. *ACM TISSEC*, 1(1):66–92, June 1998.

[39] M. Rennhard and B. Plattner. Practical anonymity for the masses with morphmix. In A. Juels, editor, *Financial Cryptography*. Springer-Verlag, LNCS (forthcoming), 2004.

[40] M. Rennhard, S. Rafaeli, L. Mathy, B. Plattner, and D. Hutchison. Analysis of an Anonymity Network for Web Browsing. In *IEEE 7th Intl. Workshop on Enterprise Security (WET ICE 2002)*, Pittsburgh, USA, June 2002.

[41] A. Serjantov and P. Sewell. Passive attack analysis for connection-based anonymity systems. In *Computer Security – ESORICS 2003*. Springer-Verlag, LNCS 2808, October 2003.

[42] R. Sherwood, B. Bhattacharjee, and A. Srinivasan. p^5: A protocol for scalable anonymous communication. In *IEEE Symposium on Security and Privacy*, pages 58–70. IEEE CS, 2002.

[43] A. Shubina and S. Smith. Using caching for browsing anonymity. *ACM SIGEcom Exchanges*, 4(2), Sept 2003. <http://www.acm.org/sigs/sigecom/exchanges/volume_4_(03)/4.2-Shubina.pdf>.

[44] P. Syverson, M. Reed, and D. Goldschlag. Onion Routing access configurations. In *DARPA Information Survivability Conference and Exposition (DISCEX 2000)*, volume 1, pages 34–40. IEEE CS Press, 2000.

[45] P. Syverson, G. Tsudik, M. Reed, and C. Landwehr. Towards an Analysis of Onion Routing Security. In H. Federrath, editor, *Designing Privacy Enhancing Technologies: Workshop on Design Issue in Anonymity and Unobservability*, pages 96–114. Springer-Verlag, LNCS 2009, July 2000.

[46] A. Tannenbaum. Computer networks, 1996.

[47] The AN.ON Project. German police proceeds against anonymity service. Press release, September 2003. <http://www.datenschutzzentrum.de/material/themen/presse/anon-bka_e.htm>.

[48] M. Waldman and D. Mazières. Tangler: A censorship-resistant publishing system based on document entanglements. In 8^{th} *ACM Conference on Computer and Communications Security (CCS-8)*, pages 86–135. ACM Press, 2001.

[49] M. Waldman, A. Rubin, and L. Cranor. Publius: A robust, tamper-evident, censorship-resistant and source-anonymous web publishing system. In *Proc. 9th USENIX Security Symposium*, pages 59–72, August 2000.