

Практическая криптография в Java. Защита коммуникаций

Николай Жишкевич

1. Введение. Защищенные сети, применение и реализация в Java

В предыдущей статье серии я уделил большое внимание созданию и использованию средств SSL. Давайте вспомним, какие задачи мы решали с использованием данной технологии, а также подготовимся к следующему шагу нашего рассказа.

Основная область применения SSL — это обеспечение безопасной коммерции в интернет при пересылке информации с одного сайта на другой, от клиента к серверу. В ходе подобного взаимодействия с помощью протокола HTTPS создается нечто похожее на специальную выделенную линию связи, информация по которой передается с гарантией того, что она не будет искажена или подделана злоумышленником, обеспечивается это за счет использования хеш-функций и сигнатур на основе математического аппарата двухключевой криптографии. Гарантируется также, что информация не будет прочитана злоумышленником благодаря шифрованию.

Разумеется, что такие возможности очень полезны и не только для сетей интернет, но также и для сетей интранет. Каждая организация по мере своего роста начинает сталкиваться с проблемой организации собственной сети. До тех пор пока размер организации не превосходит нескольких десятков компьютерных рабочих мест, а ее географическое размещение ограничивается пределами одной комнаты, этажа, или одного здания, то проблем как таковых нет. Если в организации идет перемещение секретной информации, то большей своей частью данный трафик идет через линии связи находящиеся в монопольном ведении организации и грамотного системного

администратора. В таком случае большинство методов взлома основанных на подключении к линии передачи и съема показаний с последующей его интерпретацией будет невозможно. Сами представьте ситуацию: входит в ваш офис незнакомый человек, достает из чемоданчика набор шпионских приспособлений, подключает их к проводу, предварительно выковыряв его из короба, и уходит, напоследок предупредив секретаршу или офис-менеджера, чтобы они не трогали оборудование до того, как завтра он повторно придет и заберет результаты секретных переговоров. Если же ваш провод связи соединяет несколько зданий, то к нему подключиться гораздо легче. Если вы используете беспроводные сети то, я думаю, вы уже сами догадались что произойдет.

По мере роста компьютерной сети, будут возрастать затраты на обеспечение ее безопасности, и это аксиома не требующая доказательств. Что же, давайте усложним ситуацию и добавим директора нашей конторы "Рога и копыта ltd.", который с ноутбуком ездит по городу (области, стране, миру) и хочет оперативно получать сведения о результатах деятельности менеджера Васи Тапкина который как настоящий менеджер появляется на работе раз в месяц за зарплатой, а все время сидит дома за своим домашним компьютером и в перерывах между игрой в lines&tetris играет на рынке nasdaq, и которому для этого крайне необходима возможность прямого, в режиме реального времени, общения с Петей Чебурашкиным, отличным специалистом по международному рынку высоких технологий, который приходит на работу в 6-00, а домой вообще не возвращается.

Итак, подведя итог, у нас есть заданное количество лиц, которые должны иметь возможность в режиме реального времени взаимодействовать между собой, возможность совместной работы над проектами. Причем местоположение данных лиц способно изменяться, т.е. система связи должна быть мобильна. Если мой пример с тремя сотрудниками гипотетической организации «Рога и копыта ltd.» показался вам надуманным, то я могу сказать что в нашей организации из-за огромного увеличения спроса на рынке на рога и копыта, возникает необходимость открытия ряда филиалов в соседних городах или даже странах. В каждом филиале есть выделенный компьютер, на котором собирается свежая информация о результатах продаж за очередной период. Далее эта информация должна поступать в головной офис, где и принимается решение о том продавать или покупать. Я не буду говорить о том, что данная информация должна поступать не просто оперативно, а сверх оперативно, не буду говорить о том,

что злые конкуренты только делают, что спят и видят как украсть, подменить информацию (в общем, занимаются не правильной конкурентной борьбой).

Способы решения таких ситуаций в общем случае уже тщательно продуманы и реализованы многократно. Я не могу играть роль открывателя истин, ибо они уже были открыты, я просто попытаюсь их повторить, но уже для Java. В данной статье я хочу пройтись по двум основным направлениям обеспечения безопасности для электронной коммерции. Я рассмотрю вопросы защиты трафика данных передаваемого по сети с помощью средств туннелирования или создания VPN (частные виртуальные сети), а также как частный, но все таки относительно самостоятельный случай я рассмотрю и вопрос защиты электронной почты. Решая эти две задачи я буду максимально использовать возможности Java и Java Crypto API.

Однако, давайте прежде всего снова вернемся к вопросу с которого и начали. Как защитить передаваемую информацию? В общем случае существует два способа решения данной проблемы:

- Создание специальных выделенных линий связи защищенных от доступа извне. Как вариант можно предположить не покупку, а аренду, однако в любом случае согласитесь, что такое решение помимо того, что требует очень больших капиталовложений, и если честно довольно слабоват в защите, не говоря уже о том, что идет снижение мобильности пользователей.
- Использование общедоступных телекоммуникационных каналов, говоря проще через интернет с помощью модема, тарелки, выделенной линии в особых случаях с помощью мобильного. Разумеется, в данном случае мы будем передавать информацию по открытой линии связи, и перехватить трафик теперь не будет составлять ни малейшей сложности. Следовательно, возникает необходимость передаваемый трафик шифровать и проводить качественную процедуру проверки входа в сеть.

Далее привожу цитату с сайта www.osp.ru:

Однако реальность такова, что в настоящее время установка криптосистем и создание VPN обходится во много раз дешевле, причем виртуальные частные сети оказываются зачастую надежнее собственных частных и ведомственных сетей с точки зрения безопасности: на дешифрацию трафика хорошо защищенного канала могут уйти миллионы лет работы всего нынешнего парка компьютеров. По

некоторым данным, Пентагон даже в случае всемирной ядерной войны намерен до 75-80% всей конфиденциальной информации передавать по Internet. Думается, это не случайно.

Таким образом данную статью я решил посвятить методам защиты передаваемой информации и в частности VPN, а также вопросу применения для защиты трафика сети средств Java. Как я уже говорил, и наверное повторю еще не раз, VPN — это средство для надежной передачи информации через ненадежную сеть. Если внимательно посмотреть на рынок услуг VPN то можно заметить, что создаются решения не только для связывания разрозненных офисов или пользователей через общедоступные и априори ненадежные глобальные сети, а через любую сеть которая не надежна в плане сохранения конфиденциальности передаваемой информации. Итак всякие умные организации которые зарабатывают на VPN много запрещенных условных единиц, я надеюсь, что вы не сомневаетесь в прибыльности данного сегмента рынка. Короче говоря, сети VPN грубо делят на четыре категории, краткое перечисление и комментарий к которым я привожу ниже.

1. **Intranet VPN** — мы создаем связь между множеством офисов, филиалов некоторой организации, которые расположены по всему миру.
2. **Remote Access VPN** — нечто похожее на предыдущий вариант, однако в данном случае связь идет не между локальными сетями офисов, а связь сети офиса и мобильного пользователя который ездит по миру с ноутбуком и сотовым телефоном. В данном случае существуют технические отличия связанные с тем, что как правило мобильные пользователи пользуются услугами коммутируемого доступа и опять таки как правило получают динамические ip-адреса, что усложняет их идентификацию.
3. **Client/Server VPN** — в данном случае виртуальный канал организуется между двумя узлами сети одной (внимание) одной физической сети. Как пример я приведу с ситуацию с Васей Тапкиным хранящим секретные данные на сервере huz.net.comp в локальной сети, или даже в той же комнате, но в этой сети у нас есть некоторое количество лиц (на которых, конечно, никто не указывает), но организация им не очень верит и боится как бы они эту информацию не перехватывали со всевозможными неправильными целями.
4. **Extranet VPN** — Суть в том, что к нашей сети имеют доступ некоторое количество сторонних лиц, использующих, например, гостевой статус, и которые не должны

получить доступ к информации им не предназначенной.

2. VPN — Virtual Private Network — это виртуальная частная сеть

Разумеется, что использование средств криптографии при обмене информацией не снимает требования внимательного подхода к установке брандмауэра сети, он же сетевой экран, основная задача которого защитить локальную сеть от несанкционированного внешнего трафика и при этом желательно сделать так, чтобы пользователи не испытывали неудобств связанных с запретом определенных сервисов интернет. Можно поступить очень грубо и просто, закрыть доступ по всем портам кроме, например почтовых, и пусть сотрудники не по чатам и сайтам для взрослых ходят, попутно скачивая мегабайты новых вирусов и троянов, а работают на благо нашей организации. Но, к сожалению, грань между нужным для работы и остальным достаточно прозрачна.

Я хочу поставить и реализовать (ну, или хотя бы сделать основные наброски) приложения позволяющего выполнять туннелирование запросов сети с их защитой. Конечно, подобные средства уже достаточно давно были встроены во многие сетевые операционные системы, например в Windows 2000/XP есть поддержка и туннелирования и брандмауэра соединения с Интернет, которая доступна даже домашним пользователям (хотя Microsoft прозрачно намекает на проблемы совместного использования этих вещей) а если поискать, то можно найти решения и сторонних поставщиков.

Итак, мы определили, что передаваемую информацию следует защищать. Остается только решить еще один концептуальный вопрос. Нам следует определиться на каком именно уровне модели OSI будут реализованы криптографические расширения.

Задача, которую я сформулировал в начале данной статьи, к сожалению, не имеет решения средствами Java. Увы, но задача туннелирования сообщений, да причем так, чтобы прикладные приложения не догадывались ни о чем, должна быть реализована на более низком уровне, чем предоставляет Java. Чтобы подтвердить этот факт, я попытаюсь кратко изложить основные сведения по туннелированию и его прикладной реализации для Windows NT/2000. На картинке ниже я привел пример, как выглядит схема функционирования сети VPN. Между двумя локальными сетями связь

осуществляется через потенциально опасный интернет. А мы для защиты информации выполняем ее упаковку в специальные пакеты, которые зашифрованы, имеют средства обеспечения целостности и уникальности.

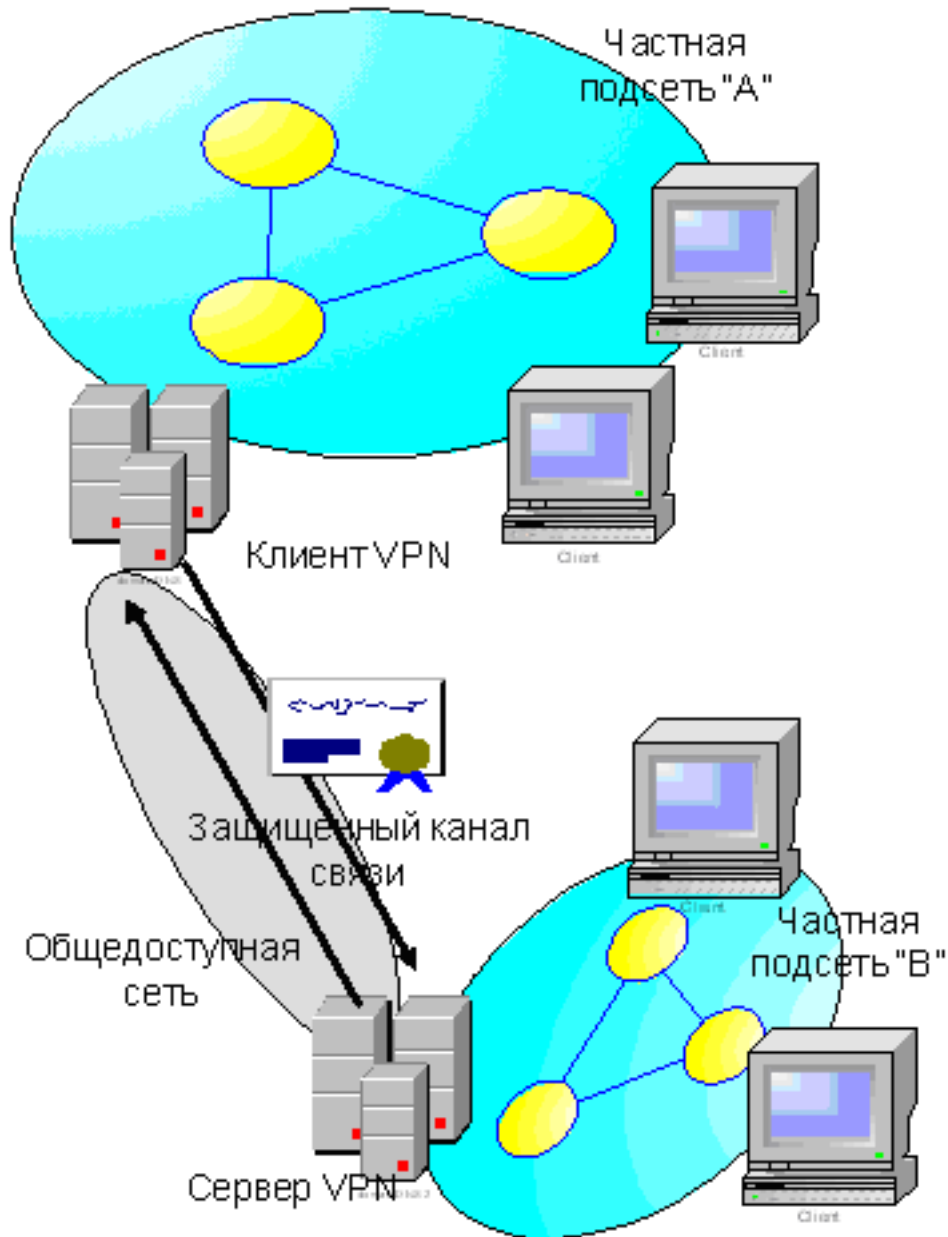
Если читатель знаком с понятием стека протоколов TCP/IP и знает что информация при передаче по сети разбивается на небольшие части которые и называются пакеты, в каждом из которых хранятся сведения о том, на какую именно машину сети, т.е. ее IP-адрес, должно прийти данное сообщение. В случае использования средств туннелирования эти передаваемые пакеты помещаются внутрь еще одних пакетов, играющих роль несгораемых сейфов, в которых информация может безопасно перемещаться по потенциально опасной внешней сети. Я только что сказал «потенциально опасной внешней сети», на самом деле критерий того, между какими машинами в сети необходимо выполнять туннелирование достаточно условен, и вполне возможна ситуация, что вы будете вынуждены в локальной сети создать подсеть, например, «Менеджеры», также подсеть «Директорат», а между ними обмен информацией вести через туннель VPN. Дело в том, что большинство вторжений в сеть организации выполняются не извне, а изнутри, в любой организации вопрос лояльности и честности сотрудников никогда не может быть решен на все 100%.

3. Схема работы средств туннелирования

Я попытаюсь кратко напомнить о уровнях модели OSI и результатах, к которым приведет попытка внедрения именно на эти уровни криптографических алгоритмов.

OSI — это сетевая модель, которая была принята как стандарт построения распределенных сетей и систем использующих неоднородные компоненты. Целью было провести унификацию средств обмена информацией от различных поставщиков решений. Структурно модель OSI можно представить в виде пирамиды из семи уровней или этажей, и как в реальной жизни дом строится на базе фундамента, потом ставят стены и в конце дом покрывают крышей и никак иначе, такой порядок определен здравым практическим смыслом, так и модель OSI состоит из набора слоев, или этажей здания, каждый из этих уровней решает свою задачу, в общем случае не располагая никакой технической информацией о том, какую и как именно решает свою задачу другой уровень. Кстати говоря, данный принцип применим практически в любой сфере практики. Есть закон построения больших и сложных систем: *"Никакая*

часть сложной системы не должна догадываться о том, как устроена другая часть данной сложной системы, взаимодействие должно идти на уровне интерфейсов". Если вам знакомо понятие черного ящика, то это тот самый случай. Уровень под номером "X" догадывается о существовании только уровня "X-1" и "X+1". Снова возвращаясь к выбору уровня модели OSI, для создания нами средства туннелирования рассмотрим уже созданные и зарекомендовавшие себя решения.



В настоящий момент существуют следующие протоколы построения VPN: "PPTP", "L2TP", "IPsec". Я никоим образом не буду углубляться в технические детали их функционирования более чем это будет необходимо, самое главное это понять на каком уровне модели OSI реализованы эти протоколы.

Протокол IPsec работает на сетевом уровне модели OSI. Сетевой уровень это уровень протокола IP из стека TCP/IP, на данном уровне решаются вопросы маршрутизации сообщения. Итак, входящий пакет IP преобразуется в новый формат, в ходе этого данные шифруются и к ним добавляются заголовки для обеспечения их целостности и невозможности модификации случайной или специальной атаки злоумышленника. Затем новый сформированный пакет передается на второй уровень модели OSI (канальный) и далее пакет отправляется по сети как обычный, отличия возникают уже после его приема вторым участником процесса коммуникации. Раз информация будет защищена с помощью средств шифрования то расшифровать его сможет только тот, кто знает специальный сеансовый ключ, который был сформирован на этапе установления соединения "стадия рукопожатия".

Второй протокол, PPTP, обладает также возможностями по шифрованию данных, однако защищенной информация передается только через канал, в то время как внутри самой локальной сети данные передаются в открытом виде. Трудно сказать насколько это большой недостаток, то что это недостаток, увы, очевидно. Однако дело в том, что использовать криптографические расширения для защиты всего трафика несколько глупо, все таки не все сведения которыми вы обмениваетесь являются секретными. Одно дело, если вы пересылаете квартальный финансовый отчет директору и другое дело если вы решили посмотреть по сети новое кино "Терминатор 5", матрицу с которым принес ваш сосед Вася Тапкин.

Следует пожалуй уделить пару слов и тому, как решается вопрос о выборе объектов защиты. В общем случае перед нами три пути: защищать все, не защищать ничего, защищать то, что сотрудник посчитает нужным. Второй путь явно не наш. Первый не совсем хорош именно из-за потери скорости работы, ведь вы не думаете, что защита информации является бесплатной с точки зрения системных ресурсов. Третий путь с первого взгляда кажется неплохим, однако как известно что наиболее слабым звеном любой системы является именно человек. Можете быть уверены, что ваши сотрудники будут забывать защитить информацию, перестраховываться, писать пароли на бумажках и расклеивать их на мониторах, расскажут все коды доступа первому встречному крякеру хоть немножко знакомому со средствами социальной инженерии.

Ну да ладно, не в том дело. А дело в том, что данный протокол (PPTP) работает на канальном уровне модели OSI. И соответственно, дает возможность выполнять

туннелирование для протоколов TCP/IP, IPx/spx. Однако дело в том, что раз мы находимся немножко ниже, чем третий уровень, на котором решается вопрос маршрутизации и адресации в сети, то мы должны будем самостоятельно реализовать целый набор функций, которые в общем случае уже реализованы шагом выше. И действительно для данного протокола создается не один канал, а два, из которых один служит для обмена информацией, а второй выполняет роль служебного. По этому каналу отправляются пакеты TCP с помощью которых стороны договариваются о средствах шифрования и аутентификации, а также периодически выполняется проверка доступности линии связи, т.е. сбой в среде передачи не произошел ли случаем.

Следующий протокол для рассмотрения, L2TP, также работает на канальном уровне и также использует дополнительный канал для управления потоком данных. Отличие от предыдущего варианта в том, что используются для управляющих взаимодействий не пакеты TCP, а ненадежные, но зато очень быстрые пакеты UDP.

Итак я рассказал о том какие существуют подходы к защите передаваемой информации, описал понятие VPN, кратко рассмотрел существующие решения и их основу - местонахождение в стеке TCP/IP. Теперь нам пора делать вывод. Если мы хотим создать механизм туннелирования, который позволял бы клиентским приложениям выполнять безопасный обмен информацией, то необходимо работать на уровне сетевом или канальном, увы в Java ниже чем транспортный протокол уйти довольно тяжело, кроме того при практической реализации возникает необходимость работать с сырыми пакетами данных, полезным была бы возможность подменять в пакете информацию о отправителе пакета. Такие фокусы возможны на C с использованием WINAPI, да и то я боюсь, что могут возникнуть ряд сложностей, как бы не пришлось писать код на уровне драйверов устройств.

И, кроме того, я, пожалуй, с целью окончательно убедить вас в том, что задача построения VPN действительно сложна и требует работы на низком уровне операционной системы, приведу выдержки требований предъявляемых к качественной системе VPN. Взято на сайте www.infotecs.ru:

Программное обеспечение агента VPN должно быть реализовано на низком уровне операционной системы, взаимодействовать непосредственно с сетевым драйвером и обеспечивать перехват любого трафика, независимо от используемого протокола. К

сожалению далеко не все реализации агентов VPN работают на таком низком уровне, а обеспечивают перехват только некоторых IP-протоколов (обычно TCP, UDP) и оставляют компьютер уязвимым по многим другим протоколам. Агент VPN должен не только обеспечивать преобразование (шифрование) трафика в зависимости от адресата, но и выполнять функции мощного персонального сетевого экрана и обеспечивающего возможность безопасной работы с открытыми ресурсами локальной сети. Иначе функции шифрования могут оказаться уязвимыми для атак на компьютер. Функции шифрования и фильтрации для снижения уязвимости должны выполняться единым программным модулем. Функции фильтрации трафика должны обеспечиваться как для зашифрованных, так и открытых пакетов. Такой персональный сетевой экран, устанавливаемый на компьютер не очень продвинутого пользователя, должен обеспечивать надежную защиту без каких либо дополнительных настроек со стороны пользователя. Важная функция — способность автоматически поддерживать протоколы DHCP, то есть динамическое выделение IP-адресов. Иначе для пользователей будет сплошной головной болью следить за изменением адресов других пользователей, особенно соединяющихся с Интернет по DIAL UP. Это означает, что должны присутствовать надежные криптографические защищенные процедуры авторегистрации IP-адресов, специальные программы — серверов IP-адресов, взаимодействующих между собой и выполняющих функции оповещения о новых IP-адресах участников VPN, их реальном наличии в сети. К сожалению, многие реализации VPN не поддерживают протоколов DHCP, а если и поддерживают, то только средства VPN, установленные на входе локальной сети. То есть для двух компьютеров, адреса у которых могут меняться, обмениваться данными между собой по защищенному VPN — соединению остается проблемой. Способность работать через различные типы Firewall-Proxy, осуществляющих преобразование адресов. Это важный момент, так как практически во всех локальных сетях с внутренней адресной структурой, так или иначе подключенных к Интернет, уже установлены различные типы Firewall-Proxy. Многие стандартные и не стандартные протоколы VPN (SKIP, IP-sec и другие), не могут работать через уже установленные во многих организациях с внутренней адресной структурой различные типы Firewall или простые Прокси. Поэтому агенты VPN должны преобразовывать получаемые ими IP-пакеты в некий единый протокол, например UDP, для которого возможно выполнение функций NAT (Network address translation). Или на компьютер с Прокси локальной сети должен устанавливаться

некоторый модуль VPN, функционирующий ниже ПО Прокси и самостоятельно выполняющий функции адресного Прокси-сервера для защищенных соединений. Быть полностью прозрачными для любых типов IP-протоколов, формируемых как прикладными программами, так и операционной системой.

Ну и так далее, за более подробным списком требований, которых, кстати, около 20, отсылаю к оригиналу на момент написания данного материала www.iitrust.ru/articles/IP-vpn.htm.

Подводя некоторый итог тому, что было сказано раньше. Мы не можем реализовать механизмы VPN на Java подобные тем, что присутствуют в IPSec, данный протокол реализован на уровне IP (сетевой уровень, или уровень № 3). За счет этого он обеспечивает высокий уровень защиты, внимание, прозрачной для большинства приложений, служб и протоколов верхнего уровня. Помните, что я говорил уровни модели OSI взаимодействуют только на уровне интерфейсов, не вникая в технические детали того, как выполняется та или иная функция. Следовательно, использование IPSec не требует внесения изменений в существующие приложения.

Когда мы на сетевом уровне поставили слой IPSec, то автоматически защитили все протоколы которые находятся выше, это TCP, UDP и соответственно все протоколы которые находятся на более высоких уровнях чем 4, это HTTP, FTP, SMTP, POP3 и другие.

Как все прекрасно получается в теории, но к сожалению на практике возникает некоторое количество сложностей связанных с необходимостью работы на более низких уровнях модели OSI чем предоставляет Java.

4. Защищенные сети и SSL

Есть и другой подход. Я много говорил в предыдущих статьях о SSL. Мы создали веб-сервер который общался с клиентом по протоколу HTTPS, а теперь ответьте сами себе разве существует принципиальная разница что защищать: то ли трафик HTTP, то ли любую другую информацию передаваемую по сети. Однако следует четко понимать, что раз мы находимся выше чем третий уровень, то прикладные приложения должны догадываться о нашем существовании и сотрудничать с кодом VPN.

Ладно, не будем пытаться пробить головой бетонную стену, мы пойдем другим путем. Поставим перед собой задачу создать механизм туннелирования о котором клиент должен, пожалуй, знать, знать только что такой механизм есть, и чтобы пользоваться его услугами надо выполнить следующий ряд волшебных манипуляций, но клиент не должен в общем случае ничего знать о том каким способом выполняется преобразование передаваемых данных и как выполняется проверка того, что соединение может быть установлено.

В качестве основы для разрабатываемой системы будет лежать как вы уже наверное догадались тот самый https сервер который нам удалось создать в предыдущей статье серии. Веб-сервер предоставлял клиенту свой сертификат и защищал трафик. Совсем не сложно было бы добавить требование со стороны сервера в том, чтобы клиент также представился для начала взаимодействия. Технически это осуществляется с помощью вызова метода:

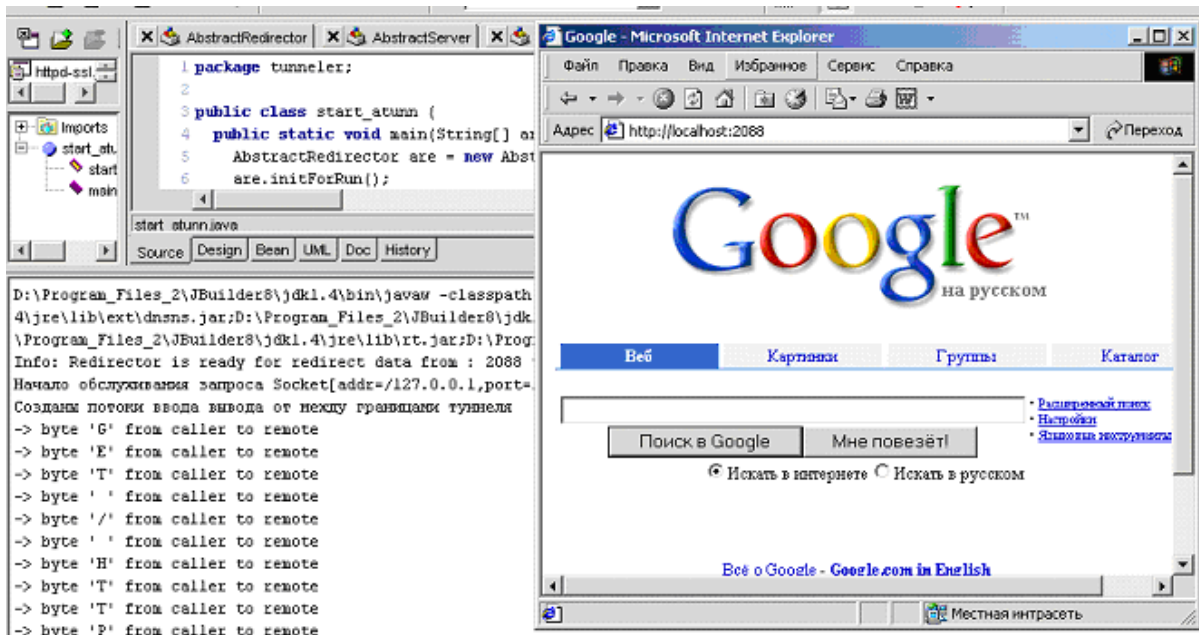
```
SSLServerSocket serverSocket = (SSLServerSocket)
    ssfa.createServerSocket(localPort); //
ssfa --- javax.net.ssl.SSLSocketFactory
serverSocket.setNeedClientAuth(true);
```

Однако не забывайте, что я поставил требование к разрабатываемой системе. Клиенты не должны иметь представления о таких технических деталях как сертификаты, SSL, в идеале они должны верить что взаимодействуют как по открытому каналу. Поэтому необходимо будет создать слой который будет играть роль посредника между ничего не знающим клиентом и тем слоем который выполняет непосредственно защиту трафика. Однако давайте мы сначала сосредоточимся на решении подзадачи - создании приложения для перенаправления сетевых вызовов.

5. Пример кода. Туннель для сетевых вызовов

Сначала я разработаю приложение, единственное назначение которого будет выполнять туннелирование запросов от одной машины сети к другой. В общем случае клиент обращается на определенный порт сервера, а его запрос максимально прозрачно перенаправляют на другой адрес IP, и на другой номер порта. Разумеется, что в ряде ситуаций подобный ход не удастся, особенно в том случае, если адресат проверяет информацию о том, кто отправил данное сообщение, или идет попытка

выполнить встречное соединение, при этом адресат ничего не знает о туннелировании, но, к сожалению, все эти задачи средствами только Java реализовать нельзя. Если вы найдете решение, то я буду только рад.



Обратите внимание на картинку, я на ней привел пример не только среды JBuilder, но и окно браузера в котором открыт поисковик Google. Но посмотрите на адресную строку, смею вас уверить, что я работаю не в Google, и на самом деле запрос, который я послал локальной машине на порт 2088, был перенаправлен в интернет на сайт Google. Кстати, если вы внимательно посмотреть на скриншот, то можно увидеть лог пересылаемых данных по туннелю. По крайней мере, я думаю, что начало любого запроса http “GET / HTTP 1.1” увидеть можно.

Вот исходный код примера, обратите внимание на то, что для хранения информации я использую файлы xml и анализ данного файла выполняю с помощью того же вспомогательного класса XTools, который я применял и для создания веб-сервера.

Схема работы приложения очень проста: я создаю серверный сокет ServerSocket и жду входящих соединений на определенный порт, номер которого указывается в файле конфигурации. Затем после поступления запроса создается соответствующий клиентский сокет для чтения запросов от иницилирующей связ стороны и сразу же

Практическая криптография в Java. Защита коммуникаций

создается сокетное соединение с тем хостом и портом на который входящий запрос должен быть перенаправлен. Дальнейшие действия просты и очевидны: необходимо каждый байт приходящий от стороны иницирующей соединение, (давайте для определенности будем называть ее сторона "А") перенаправлять к стороне "В" (информация о ней находится в файле конфигурации)) и соответственно каждый байт который посылает нам сторона "В" должна быть направлена стороне "А". Разумеется что данные действия должны выполняться асинхронно и независимо друг от друга, поэтому лучшего решения чем создать два класса потока, класса производных от Thread и перекрыть в нем метод run так чтобы в цикле читать байт из входного потока и писать в выходной поток. И цикл должен будет продолжаться до тех пор пока сокетное соединение не будет закрыто.

Следующим шагом будет добавление защиты пересылаемой информации, для этого на обеих сторонах соединения необходимо будет пользоваться для пересылки информации не обычными сокетами, а сокетами SSL и, внимание, нам необходимо будет затребовать аутентификацию клиента делающего соединение к серверу. Однако это будет после а пока я привожу файл конфигурации необходимый для работы просто "Redirector"-а.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSPY v5 rel. 4 U (
  http://www.xmlspy.com) by Nikolai (Comp) -->
<Redirector>
  <Routes>
    <Route>
      <Localport>2088</Localport>
      <Remoteport>2089</Remoteport>
      <Remotehost>localhost</Remotehost>
      <Direction>PlainToSSL</Direction>
    </Route>
    <Route>
      <Localport>2089</Localport>
      <Remoteport>1088</Remoteport>
      <Remotehost>localhost</Remotehost>
      <Direction>SSLToPlain</Direction>
    </Route>
  </Routes>
</Redirector>
```

Для анализа данного документа XML я использовал вспомогательный класс XTools который был приведен в предыдущей статье серии. И наконец я привожу и исходный код приложения.

```
package com.javable.www.columns.security;

import java.io.*;
import java.net.*;
import java.util.*;

public class AbstractRedirector {
    class Route {
        protected int localPort;
        protected int remotePort;
        protected String remoteHost;
        public Route(String remH, int loP, int reP) {
            remoteHost = remH;
            remotePort = reP;
            localPort = loP;
        }

        public String toString() {
            return "Route record: from localhost :" +
                localPort + " to " + remoteHost +
                ":" + remotePort;
        }
    }

    Vector routes = new Vector();
    protected String fConfigName;

    org.w3c.dom.Document xmlDocConfig;
    /**
     * Конструктор абстрактного перенаправителя
     * запросов по сети основные параметры
     * необходимые для работы должны находиться
     * в файле конфигурации
     * среди таковых являются:
     * 1)
     * @param fName
```

```
*/

public AbstractRedirector(String fName) {
    fConfigName = fName;
}

public boolean initForRun() {
    try {
        org.apache.xerces.parsers.DOMParser dp =
            new org.apache.xerces.parsers.
                DOMParser();
        dp.parse(new org.xml.sax.InputSource(new InputStreamReader(new
            FileInputStream(fConfigName),
                "utf-8"))));
        org.w3c.dom.Document doc = dp.getDocument();
        xmlDomConfig = doc;

        for (int i = 0; ; i++) {

            Integer lopo = XTools.getXPathInt("Redirector/Routes/Route["+i+
                "]/Localport",
                doc.getDocumentElement());
            Integer repo = XTools.getXPathInt("Redirector/Routes/Route[" + i +
                "]/Remoteport",
                doc.getDocumentElement());
            String remoteHost = XTools.getXPathText("Redirector/Routes/Route[" + i +
                "]/Remotehost",
                doc.getDocumentElement());

            if (lopo == null && repo == null && remoteHost == null)
                break;
            // если не удалось прочитать информацию о
            // следующем маршруте перенаправления, то
            // прекращаем чтение файла конфигурации,

            // значит больше конфигураций нет

            if (lopo == null || repo == null || remoteHost == null) {
                System.out.println(
                    "Критическая ошибка файла конфигурации,
```

```
        отсутствует номер порта для внутренней или
        внешней стороны редиректора или не указан
        удаленный сервер к которому будут туннелироваться запросы");
    return false;
}

Route ro = new Route(remoteHost, lopo.intValue(), repo.intValue());
System.out.println(
    "Info: Redirector is ready for server route : " + ro);
routes.add(ro);
}

}
catch (Exception e) {
    e.printStackTrace();
    return false;
}

return true;
}

public void synchroRun() {
    for (int i = 0; i < routes.size(); i++) {
        Route r = (Route) routes.get(i);
        serveRoute(r);
    }
}

protected void serveRoute(final Route r) {
    new Thread() {
        public void run() {
            ServerSocket ssock = getServerSocket(r);
            while (true) {
                try {
                    final Socket cliso = ssock.accept();
                    System.out.println("Начало обслуживания запроса " + cliso);
                    final Socket soToRemote = getClientSocket(r);
                    final InputStream fromCaller = cliso.getInputStream();
                    final OutputStream toCaller = cliso.getOutputStream();
                }
            }
        }
    }
}
```

```
final InputStream fromRemote = soToRemote.getInputStream();
final OutputStream toRemote = soToRemote.getOutputStream();
System.out.println(
    "Созданы потоки ввода вывода от между границами туннеля");
new Thread() {
    public void run() {
        int b;
        try {
            do {
                b = fromCaller.read();
                System.out.println("-> byte '" + (char) b +
                    "' from caller to remote");
                toRemote.write(b);
            }
            while (b != -1);
            cliso.close();
            soToRemote.close();
        }
        catch (IOException ex) {
            ex.printStackTrace();
            System.out.println("Туннелирование данных от
                запрашивающего к удаленному серверу
                завершено исключением " +
                ex+"; r="+ r);
        }
    }
}

.start();
// и по той же схеме запускаем туннелирование
// для ответов удаленного вызываемого приложения к том
// который данный процесс туннелирования начал
new Thread() {
    public void run() {
        int b;
        try {
            do {
                b = fromRemote.read();
                System.out.println("<- byte '" + (char) b +
                    "' from remote to caller");
            }
            while (b != -1);
            cliso.close();
            soFromRemote.close();
        }
        catch (IOException ex) {
            ex.printStackTrace();
            System.out.println("Туннелирование данных от
                удаленного сервера к запрашивающему
                завершено исключением " +
                ex+"; r="+ r);
        }
    }
}

.start();
```

```
        toCaller.write(b);
    }
    while (b != -1);
    cliso.close();
    soToRemote.close();
}
catch (IOException ex) {
    System.out.println("Туннелирование данных от
                        отвечающего сервера к запрашивающему
                        соединению завершено исключением " +
                        ex);
}
}
}

.start();

System.out.println("Потоки для Route = " + r +
                  " запущены, ожидаем следующего соединения");

}
catch (IOException ex) {
    ex.printStackTrace();
    System.out.println("Ошибка невозможно обслужить запрос");
}
}
}

.start();

}

/**
 * Схема поведения и использования данной
 * функции очень похожа на то
 * что я делал для веб-сервера весь код
 * связанный с созданием самих сокетов
 * следует вынести в отдельные методы,
```

```
* в общем случае мало (или вообще нет)
* зависящих от остальных методов данного класса
* @return
* @throws IOException
*/
protected ServerSocket getServerSocket(Route r) {
    try {
        return new ServerSocket(r.localPort);
    }
    catch (IOException ex) {
        ex.printStackTrace();
        return null;
    }
}

protected Socket getClientSocket(Route r) {
    try {
        return new Socket(r.remoteHost, r.remotePort);
    }
    catch (IOException ex) {
        ex.printStackTrace();
        return null;
    }
}
}
```

Пока созданные мною классы умеют делать только простое перенаправление и ретрансляцию запросов по протоколу TCP/IP с одного порта указанного компьютера на котором разумеется должна быть запущена программа туннелирования на другой порт другого компьютера. Кстати очень полезное приложение в том случае, когда необходимо подслушать протокол общения между собой двух сетевых приложений.

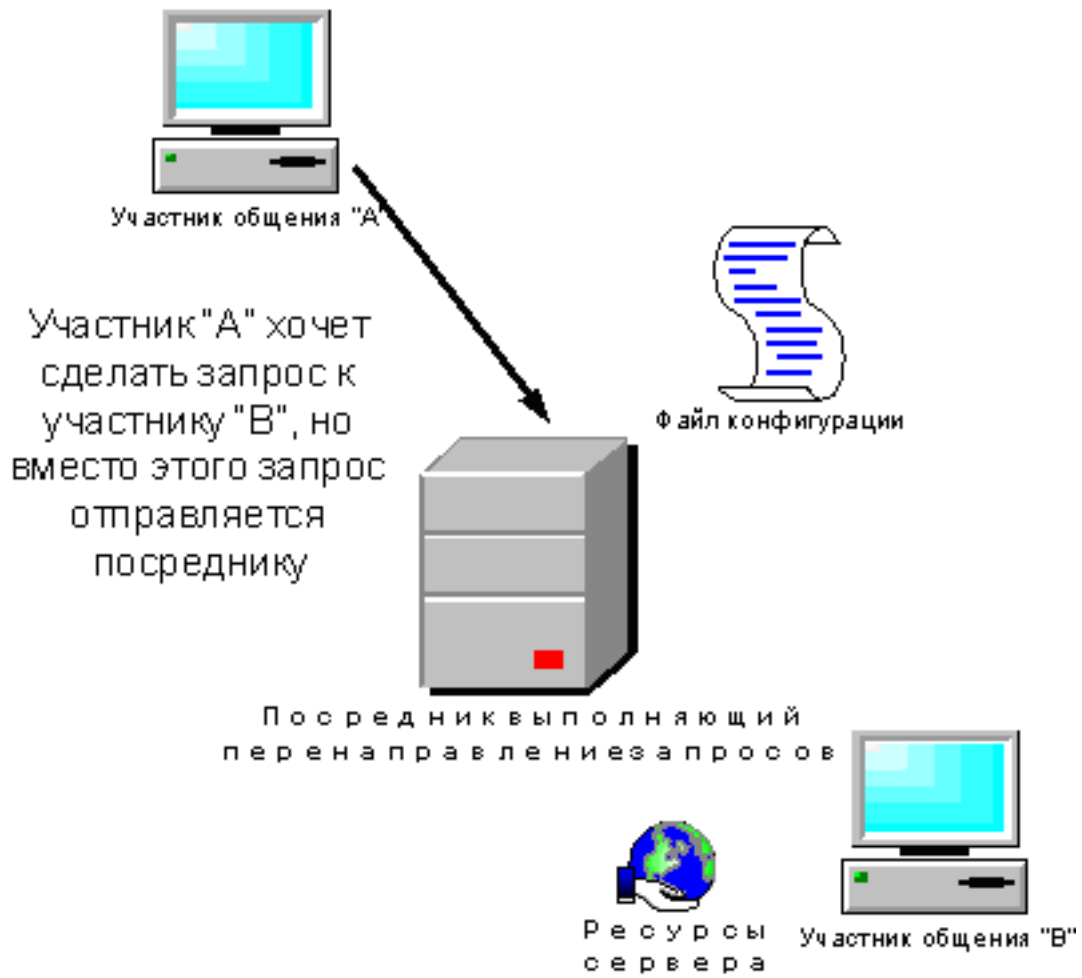
Давайте кратко рассмотрим основные моменты работы программы:

- Информация о перенаправлениях хранится в файле конфигурации, каждый элемент `Route` задает порт локальной машины, который будет слушаться на предмет входящих запросов, а также соответственно адрес и порт удаленной машины, куда запрос должен быть переправлен.
- Для хранения подобных учетных записей перенаправления был создан

вспомогательный класс `Route` содержащий три поля, служащих для хранения информации, которую я назвал шагом раньше.

- При вызове конструктора класса `AbstractRedirector` в качестве параметра передается имя файла настроек. Из данного файла читаются все элементы `Route` и помещаются в специальный массив. После создания объекта класса для начала непосредственно обслуживания должна быть вызвана функция `synchroRun`. Данная функция создает набор экземпляров безымянного класса производного от `Thread` в данном классе перекрыта функция `run`, которая в цикле читает побайтно информацию из сокета входящего запроса и отправляет его в поток вывода для сокета исходящего соединения. Данный цикл прекращается, когда сокетное соединение с любой из двух сторон будет прервано.

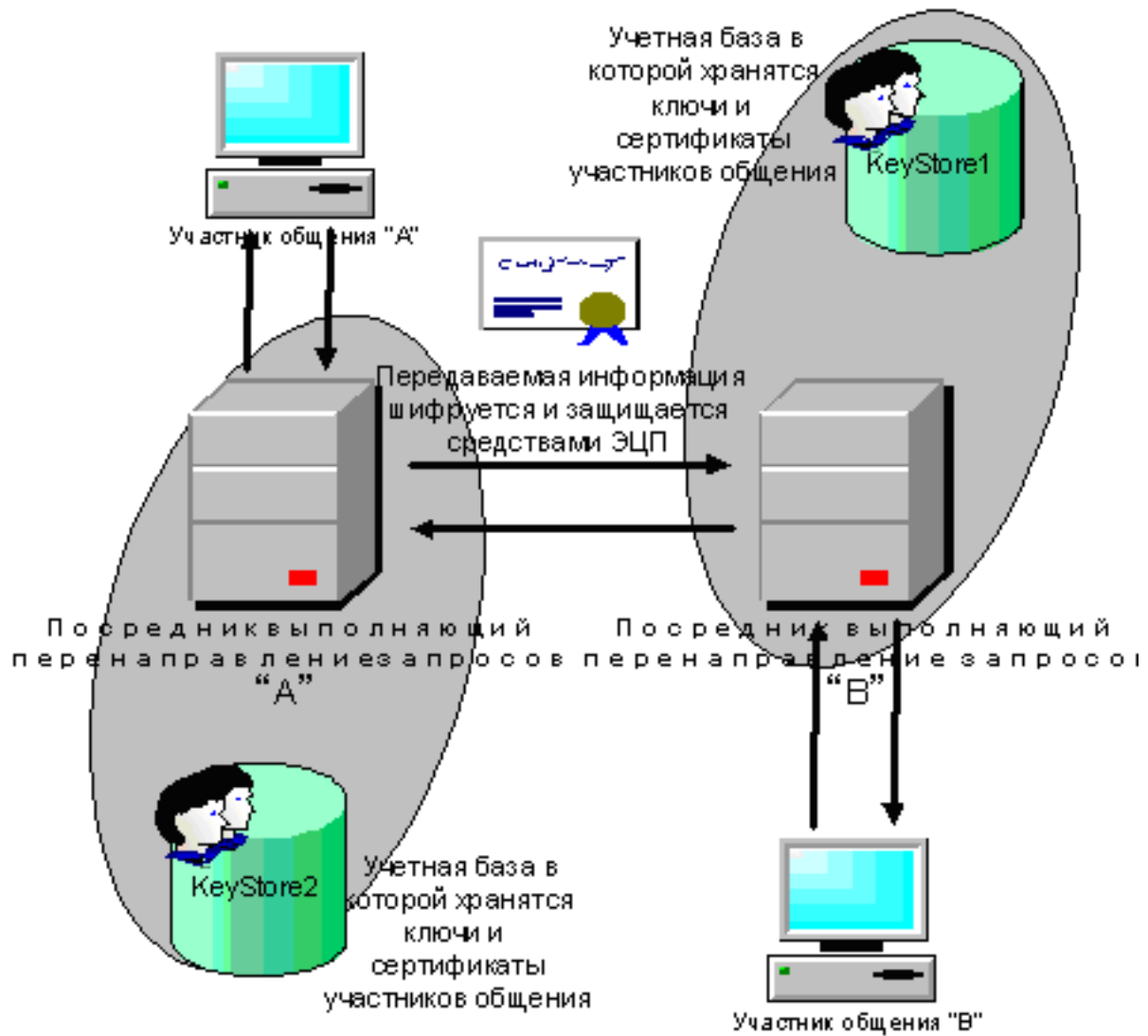
Как демонстрацию принципов работы данного приложения привожу следующую СЛС.





6. Код примера. Защищаем сетевой туннель.

А теперь давайте добавим возможность выполнения туннелирования через протокол SSL. Общая схема реализованного приложения практически идентична общей схеме создания VPN которую я приводил ранее. Небольшие отличия я привожу в виде следующей СЛС



Разумеется что для работы следующего кода нам потребуется внести изменения в файл конфигурации. Необходимо будет добавить новый элемент "SSL-section".

```
<SSL-section>
<Keystore-path>E:\mdocs\kolya\cbuilder_projects\
    httpd-ssl\store</Keystore-path>
<Keystore-password>bigsecret</Keystore-password>
<Alias-password>bigsecret</Alias-password>
</SSL-section>
```

И, наконец, я привожу исходный код примера. Обратите внимание на то что в классе `SSLRedirector` порожденном от `AbstractRedirector` я перекрыл метод `initForRun` добавив в него чтение "SSL" секции файла конфигурации, теперь каждому элементу `Route` необходимо будет добавить как характеристику признак того следует ли выполнять защиту описываемого туннеля. И следовательно мне пришлось расширить класс `Route`, создав класс `SSLRoute` и добавив в него булевскую переменную, выполняющую роль признака надо или нет выполнять защиту туннеля. Разумеется, что изменения также затронули и ключевые методы `getServerSocket` и `getClientSocket`. Теперь эти методы способны в зависимости от настроек переданного им в качестве параметра объекта `SSLRoute` создать или просто `ServerSocket` и соответственно `Socket` или же если поле `plainToSecret` установлено в `true`, то будет создан `SSLSocket`-ы.

```
package com.javable.www.columns.security;

import java.io.*;
import java.net.*;
import java.security.*;
import Javax.net.ssl.*;

public class SSLRedirector
    extends AbstractRedirector {
    protected KeyStore ks;
    protected String fStoreName;
    protected String keystorePassword;
    protected String aliasPassword;
    protected boolean isClientSecure;
    class SSLRoute
        extends Route {
            // именно в данной переменной флажке будет
            // храниться признак направления туннелирования данных
            boolean plainToSecret;
            public SSLRoute(boolean ptos, String h, int lo, int re) {
                super(h, lo, re);
                plainToSecret = ptos;
            }

            public SSLRoute(Route pa, boolean ptos) {
```

```
        super(pa.remoteHost, pa.localPort, pa.remotePort);
        plainToSecret = ptos;
    }

    public String toString() {
        return super.toString() + " ; mode = " +
            (plainToSecret ? "Plain to Secret" : "Secret to Plain");
    }
}

public SSLRedirector(String fName) {
    super(fName);
}

public boolean initForRun() {

    if (!super.initForRun()) {
        return false;
    }
    try {
        ks = KeyStore.getInstance(KeyStore.getDefaultType());
        fStoreName = XTools.getXPathText("Redirector/SSL-section/Keystore-path",
            xmlDomConfig.getDocumentElement());
        keystorePassword = XTools.getXPathText(
            "Redirector/SSL-section/Keystore-password",
            xmlDomConfig.getDocumentElement());
        aliasPassword = XTools.getXPathText("Redirector/SSL-section/Alias-password",
            xmlDomConfig.getDocumentElement());

        // как вы заметили последовательность действий
        // практически идентична тем, что мы выполняли когда
        // создавали защищенный веб-сервер,
        // и там и тут необходимо прочитать файл конфигурации из него следует
        // извлечь
        isClientSecure = true; // что это за защищенный канал когда
            // не нужно выполнять аутентификацию обеих сторон

        if (keystorePassword == null) {
            System.out.println("Пароль необходимый для доступа
                к хранилищу ключей не был найден в
                файле конфигурации,
```

```
        необходимо ввести его с клавиатуры");
    keystorePassword = new DataInputStream(System.in).readLine();
}

if (aliasPassword == null) {
    System.out.println("Пароль необходимый для доступа
        к ключу шифрования не был найден в
        файле конфигурации,
        необходимо ввести его с клавиатуры");
    aliasPassword = new DataInputStream(System.in).readLine();
}

ks.load(new FileInputStream(fStoreName),
        keystorePassword.toCharArray());

// отлично после того как мы загрузили
// все необходимые параметры для работы с SSL
// нам необходимо будет выполнить также дополнение
// к записям о перенаправлении запросов
// дело в том, что есть два вида перенаправления -
// 1) данные пришедшие в открытом виде направляются
// во внешний мир будучи зашифрованными
// 2) данные пришедшие извне через канал VPN должны
// быть расшифрованы (стать открытым текстом) и должны быть
// перенаправлены к машине внутренней сети,
// внутри данной внутренней сети защита данных не используется

System.setProperty("javax.net.ssl.keyStore", fStoreName);
System.setProperty("javax.net.ssl.keyStorePassword", keystorePassword);

// здесь я хочу вывести на экран
// список тех CA криптографических услуг
// которым я доверяю

//getAcceptedIssuers
TrustManagerFactory tmfa = TrustManagerFactory.getInstance("SunX509");
tmfa.init(ks);
TrustManager [] mana = tmfa.getTrustManagers();
for (int i=0; i < mana.length; i++){
```

```
System.out.println("----- START TrustManager # "
    +i+" Info -----");
X509TrustManager xtm = (X509TrustManager)mana[i];
java.security.cert.X509Certificate cert []=
    xtm.getAcceptedIssuers();
for (int j=0; j < cert.length; j++){
    System.out.println("----- START Certificates # "
        +j+" Info -----");
    System.out.println(cert[i].getSubjectDN().getName());
    System.out.println("----- FINISH Certificates # "
        +j+" Info -----");
}
System.out.println("----- FINISH TrustManager # "
    +i+" Info -----");
}

for (int i = 0; i < routes.size(); i++) {
    String mode = XTools.getXPathText(
        "Redirector/Routes/Route[" + i +
        "]/Direction",
        super.xmlDomConfig.getDocumentElement());
    routes.set(i,
        new SSLRoute( (Route) routes.get(i),
            mode.equalsIgnoreCase("PlainToSSL")));
}

}

catch (Exception ex) {
    ex.printStackTrace();
    return false;
}

return true;
// раз мы дошли до этой строчки и ни разу не споткнулись,
// ни разу не было выброшено исключение,
// следовательно нам удалось провести инициализацию
// приложения и мы готовы к тому
// чтобы выполнять перенаправление трафика

}
```

```
// внимание, тут мы перепишем код функций так
// чтобы они возвращали не простые сокеты а сокеты SSL
protected ServerSocket getServerSocket(Route r) {
    try {
        if ( ! ( (SSLRoute) r).plainToSecret) { // важно именно с отрицанием
            KeyManagerFactory kmfa = KeyManagerFactory.getInstance("SunX509");
            // Создаем фабрику менеджеров ключей
            // для генерации на основе их ключей
            kmfa.init(ks, aliasPassword.toCharArray());
            SSLContext sslco = SSLContext.getInstance("SSLv3");
            // Привязать фабрику ключей к
            // созданному контексту соединений по SSL3

            TrustManagerFactory tmfa =
                TrustManagerFactory.getInstance("SunX509");
            tmfa.init(ks);

            sslco.init(kmfa.getKeyManagers(), tmfa.getTrustManagers() , null);
            Javax.net.ssl.SSLServerSocketFactory ssfa = sslco.
                getServerSocketFactory();
            // а теперь имя полностью готовую для работы
            // фабрику ssl сокетов создадим и сам сокет
            SSLServerSocket serverSocket =
                (SSLServerSocket) ssfa.createServerSocket(r.localPort);
            serverSocket.setNeedClientAuth(isClientSecure);
            System.out.println("Создан серверный сокет с защитой : "
                + serverSocket);

            return serverSocket;
        }
        System.out.println("Создан серверный сокет , без защиты : at "
            + r.localPort);
        return super.getServerSocket(r);
    }
    catch (Exception ex) {
        ex.printStackTrace();
        return null;
    }
}
```

```
protected Socket getClientSocket(Route r) {
    try {
        if ( ( (SSLRoute) r).plainToSecret) {

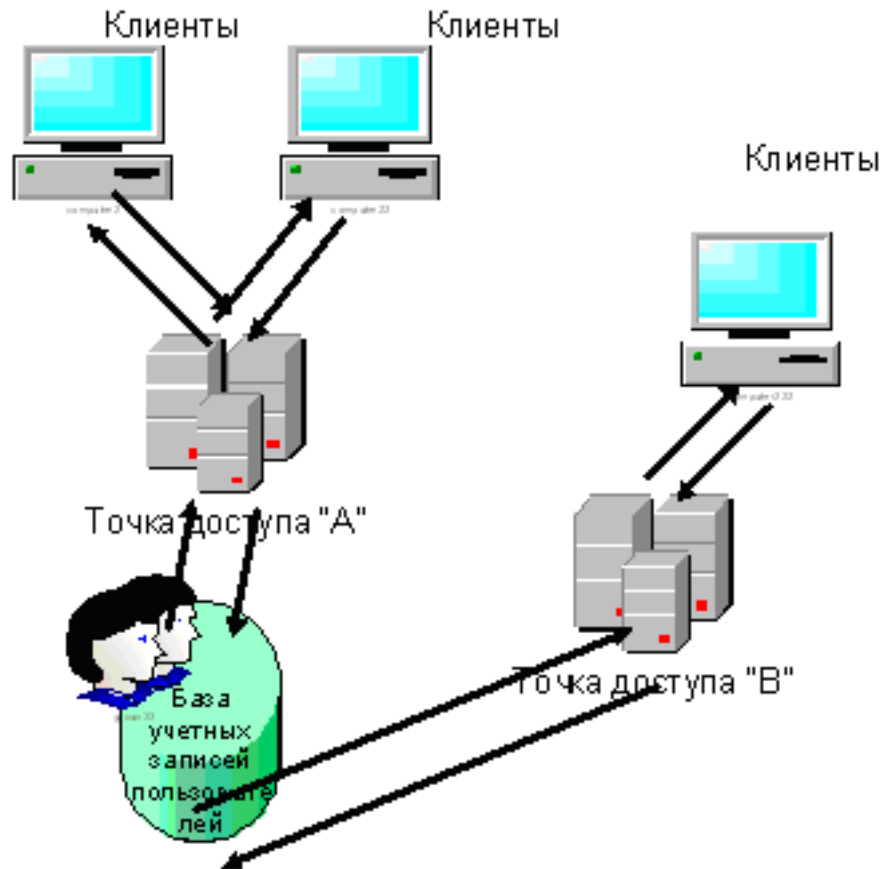
            /*******
             *
             * KeyManagerFactory kmfa =
             *     KeyManagerFactory.getInstance("SunX509");
             * // Создаем фабрику менеджеров ключей
             * // для генерации на основе их ключей
             * kmfa.init(ks, aliasPassword.toCharArray());
             *
             * /*******

            TrustManagerFactory tmfa =
                TrustManagerFactory.getInstance("SunX509");
            tmfa.init(ks);
            SSLContext sslco = SSLContext.getInstance("SSLv3");
            // Привязать фабрику ключей к
            // созданному контексту соединений по SSL3
            sslco.init(kmfa.getKeyManagers(),
                tmfa.getTrustManagers(), null);
            Javax.net.ssl.SSLSocketFactory ssfa =
                sslco.getSocketFactory();
            // а теперь имя полностью готовую для
            // работы фабрику ssl сокетов создадим и сам сокет
            SSLSocket clientSocket =
                (SSLSocket) ssfa.createSocket(r.remoteHost, r.remotePort);
            clientSocket.setNeedClientAuth(true);
            // разумеется нам следует потребовать
            // того чтобы клиент представился
            System.out.println("Создан клиентский сокет ,
                с защитой : " +clientSocket);
            return clientSocket;
        }
        System.out.println("Создан клиентский сокет ,
            без защиты : to " +r.remotePort);
        return super.getClientSocket(r);
    }
    catch (Exception ex) {
```

```
        ex.printStackTrace();
        return null;
    }
}
```

7. Направления развития примера

Давайте еще раз посмотрим на исходный код приложения и подумаем, в чем его недостаток. Недостаток надо сказать, более концептуальный, чем недостаток реализации. Как известно, врагом номер один любой организации является неразбериха, путаница, которая чаще всего порождается именно тем, что информация дублируется. Именно такая ошибка существует и в приведенном решении, на обоих концах туннеля который мы создали, находится наша программа туннелирования которой для работы необходим файл хранилища ключей и сертификатов, разумеется что информация в этих двух файлах должна быть идентична. Каждый раз при добавлении нового ключа, или при аннулировании сертификата необходимо изменения вносить синхронно, смею вас заверить, что на этом этапе по закону Мерфи будет допущена ошибка. Наилучшим решением будет выполнить централизацию всего набора хранимой информации, например, на специально выделенной машине, центральном сервере, с которым каждый участник туннеля будет общаться. Таким образом, схема работы туннеля в случае множественных точек доступа, а я надеюсь, что вы не забыли с чего мы начинали данный рассказ: у нас было множество сотрудников некоторой гипотетической организации "Рога и копыта" которые хотели получать доступ к корпоративным ресурсам через общедоступные сети. Итак, схема работы выглядит примерно так.



Если честно сказать то, когда я нарисовал такую схему, вспомнил, что где-то когда-то я нечто подобное видел, и действительно трудно поверить в то, что похожего решения не было ранее. Покопавшись в книжках, я нашел похожее изображение, под которым значилось "Схема сети VPN с использованием протокола RADIUS". Вот уж действительно не знаешь, где найдешь, а где потеряешь. На этом этапе я решил остановиться, технически реализация данного протокола или подобного ему несложна, правда я чувствую душою, что объем кода будет очень велик (я помню, что когда то я писал собственный почтовый клиент и поддержка такого простого протокола как POP3&SMTP, качественная поддержка, заняла у меня массу времени и сил), меня более интересовал вопрос "неужели до сих пор, никто не пытался решать вопрос о связи между Java и тем же RADIUS". Вообще то для истинных любителей делать все своими руками я могу сослаться на RFC RADIUS, вот его адрес <ftp://ftp.isi.edu/in-notes/rfc2865.txt>. Сам я все таки не решился и просто поискал в сети

продукты решающие подобные задачи. Первой в мой сачок после посещения сайта sun и просмотра списка его партнеров попала компания "appgate". Основные цели которые она ставит, это дать возможность авторизованным пользователям используя систему appgate получить доступ к различным приложениям (услугам), подчеркните у себя слово многим, путем выполнения только одной процедуры входа в сеть (систему). Причем трафик, которым будет обмениваться пользователь и приложения, будет защищен с помощью криптографических методов.

Разумеется, что подобные решения будут особо приветствоваться именно на рынке мобильных услуг, согласитесь, что было бы очень неплохо дать возможность с помощью мобильного устройства пользоваться услугами электронной коммерции, делать покупки, управлять своим банковским счетом и т.д., ну или в крайнем (или, кто его знает в самом крайнем случае, вы используя решения appgate сможете подключившись к компьютерной сети вашего дома делать заказ на покупку товаров, используя ваш счет в банке, доступ к которому также централизованно управляется appgate) разумеется, что подобные решения должны быть реализованы в первую очередь на Java, чтобы не говорили поклонники очень большой компании. Как подтверждение своих слов могу привести известный факт, о том что в palms5 среди прочих разных и удивительных вещей появилась поддержка VPN.

Не забывайте о том, что для корпоративного заказчика поддержка технологий VPN очень важна. К примеру, та же nokia решила создать специальный отдел (подразделение), которое будет заниматься продажами интегрированных решений в области сотовой связи корпоративным клиентам и ставку делает именно на Java, symbian, VPN.

Составными компонентами решения appgate являются:

- AppGate Appliance — сервер, который защищает локальную сеть, контролируя и санкционируя всякий доступ извне к ее услугам.
- AppGate Client — именно то ПО с которым будет обращаться конечный пользователь.
- AppGate Personal Firewall (компонент не обязательный, но настоятельно рекомендуемый)
- Secure local print — модуль для выполнения дистанционной печати документов.

Разработчики "appgate" говорят о том, что они представляют возможность

централизованного доступа к различным сервисам организации, даже доступным под различными платформами. Для выполнения авторизации и аутентификации пользователя используются принятые как стандарты PKI/Smart Cards (Entrust, Smarttrust, Telia EID), Token Cards (SecurID, RADIUS), и просто пароли.

Достаточно подробную информацию о решениях AppGate можно найти на сайте Sun по следующей ссылке solutions.sun.com/catalog.static/en_US/9/43251.html

8. Заключение

Подведем итоги. Несмотря на то, что сегодняшний материал более носил характер теоретический, мы посвятили большую часть времени на то, чтобы рассмотреть существующие решения в области защиты сетевых коммуникаций и в частности VPN. Мы много говорили о том, каким требованиям должна удовлетворять система VPN, строили аналогии. Нам также удалось создать практическое приложение для выполнения задач туннелирования сетевого трафика, а в последствии развить данный пример так, чтобы дать возможность создать защищенный канал между выделенными машинами сети. Кратко, но тем не менее были рассмотрены решения от компании appgate.

9. Ресурсы:

1. http://solutions.sun.com/catalog.static/en_US/9/43251.html — ссылка на appgate с сайта Sun.
2. <ftp://ftp.isi.edu/in-notes/rfc2865.txt> — RFC по RADIUS.
3. В следующей статье серии нам потребуются знания по почтовым протоколам, MIME, S/MIME и о том как это все реализовано в Java. Следовательно вам необходима документация по JavaMail. очень хороший учебник я нашел на www.jguru.com
4. Хорошая книжка по сетевым технологиями никогда не помешает. Автор использовал при написании статьи материалы из "Сетевые технологии Windows 2000 для профессионалов", Алексей Вишнявский.
5. MSDN. Без комментариев.