Министерство образования и науки Российской Федерации

Московский Государственный Технический Университет им. Н.Э. Баумана

Факультет «Информатика и системы управления»
Кафедра «Компьютерные системы и сети»

Сурков Л.В.

Методические указания

к лабораторной работе по дисциплине

Корпоративные сети

Раздел

Удаленное управление и мониторинг сети

Туннелирование соединений с использованием протокола SSL

2011

Теоретическая часть

SSL

SSL (Secure Sockets Layer) — криптографический протокол, который обеспечивает установление безопасного соединения между клиентом и сервером. SSL изначально разработан компанией Netscape Communications. Впоследствии на основании протокола SSL 3.0 был разработан и принят стандарт RFC, получивший имя TLS.

Протокол обеспечивает конфиденциальность обмена данными между клиентом и сервером, использующими TCP/IP, причём для шифрования используется асимметричный алгоритм с открытым ключом. При шифровании с открытым ключом используется два ключа, причем любой из них может использоваться для шифрования сообщения. Тем самым, если используется один ключ для шифрования, то соответственно для расшифровки нужно использовать другой ключ. В такой ситуации можно получать защищённые сообщения, публикуя открытый ключ, и храня в тайне секретный ключ.

Протокол SSL состоит из двух подпротоколов: протокол SSL записи и рукопожатия. Протокол SSL записи определяет формат, используемый для передачи данных. Протокол SSL включает рукопожатие с использованием протокола SSL записи для обмена сериями сообщений между сервером и клиентом, во время установления первого соединения. Для работы SSL требуется, чтобы на сервере имелся SSL-сертификат.

SSL предоставляет канал, имеющий 3 основных свойства:

- Аутентификация. Сервер всегда аутентифицируется, в то время как клиент аутентифицируется в зависимости от алгоритма.
- Целостность. Обмен сообщениями включает в себя проверку целостности.
- Частность канала. Шифрование используется после установления соединения и используется для всех последующих сообщений.

Значительное использование протокола SSL привело к формированию протокола HTTPS (Hypertext Transfer Protocol Secure), поддерживающего шифрование. Данные, которые передаются по протоколу HTTPS, «упаковываются» в криптографический протокол SSL или TLS, тем самым обеспечивая защиту этих данных. Такой способ защиты широко используется в мире Веб для приложений, в которых важна безопасность соединения, например в платёжных системах. HTTPS поддерживается всеми браузерами. В отличие от HTTP, для HTTPS по умолчанию используется TCP-порт 443.

Изначально виртуальные частные сети (VPN) на основе SSL разрабатывались как дополнительная и альтернативная технология удалённого доступа на основе IPsec VPN. Однако, такие факторы как достаточная надёжность и дешевизна сделали эту технологию

привлекательной для организации VPN. Также SSL получил широкое применение в электронной почте.

Основные цели протокола

Криптографическая безопасность: SSL устанавливает безопасное соединение между двумя сторонами.

- Совместимость: Программисты, независимо друг от друга могут создавать приложения использующие SSL, которые впоследствии будут способны успешно обмениваться криптографическими параметрами без всякого знания кода чужих программ.
- Расширяемость: SSL стремится обеспечить рабочее пространство, в котором новые открытые ключи и трудоемкие методы шифрования могут быть включены по мере необходимости.
- Относительная эффективность: работа протокола на основе SSL требует больших скоростей от CPU, в частности для работы с открытыми ключами. По этой причине SSL протокол был включен в необязательную сессию схемы кеширования для уменьшения числа соединений, которые необходимо устанавливать с нуля. Кроме того, большое внимание уделяется тому, чтобы уменьшить сетевую активность.

SSL поддерживает 3 типа аутентификации:

- аутентификация обеих сторон (клиент — сервер),
- аутентификация сервера с неаутентифицированным клиентом,
- полная анонимность.

Всякий раз, когда сервер аутентифицируется, канал безопасен против попытки перехвата данных между веб-сервером и браузером, но полностью анонимная сессия по своей сути уязвима к такой атаке. Анонимный сервер не может аутентифицировать клиента. Если сервер аутентифицирован, то его сообщение сертификации должно обеспечить верную сертификационную цепочку, ведущую к приемлемому центру сертификации. Проще говоря, аутентифицированный клиент должен предоставить допустимый сертификат серверу. Каждая сторона отвечает за проверку того, что сертификат другой стороны еще не истек и не был отменен.

Главная цель процесса обмена ключами — это создание секрета клиента (pre_master_secret), известного только клиенту и серверу. Секрет (pre_master_secret) используется для создания общего секрета (master_secret). Общий секрет необходим для того чтобы создать сообщение для проверки сертификата, ключей шифрования, секрета MAC (message authentication code) и сообщения «finished». При посылке верного

сообщения «finished», тем самым стороны докажут что они знают верный секрет (pre_master_secret).

Анонимный обмен ключами.

Полностью анонимная сессия может быть установлена при использовании алгоритма RSA или Диффи-Хеллмана для создания ключей обмена. В случае использования RSA клиент шифрует секрет (pre_master_secret) с помощью открытого ключа несертифицированного сервера. Открытый ключ клиент узнает из сообщения обмена ключами от сервера. Результат посылается в сообщении обмена ключами от клиента. Поскольку перехватчик не знает закрытого ключа сервера, то ему будет невозможно расшифровать секрет (pre_master_secret). При использовании алгоритма Диффи-Хеллмана открытые параметры сервера содержатся в сообщении обмена ключами от сервера, и клиенту посылают в сообщении обмена ключами. Перехватчик, который не знает приватных значений, не сможет найти секрет (pre_master_secret).

Обмен ключами при использовании RSA и аутентификация.

В этом случае обмен ключами и аутентификация сервера может быть скомбинирована. Открытый ключ также может содержаться в сертификате сервера или может быть использован временный ключ RSA, который посылается в сообщении обмена ключами от сервера. Когда используется временный ключ RSA, сообщения обмена подписываются server's RSA или сертификат DSS. Сигнатура включает текущее значение сообщения Client_Hello.random, таким образом, старые сигнатуры и старые временные ключи не могут повторяться. Сервер может использовать временный ключ RSA только однажды для создания сессии. После проверки сертификата сервера клиент шифрует секрет (pre_master_secret) при помощи открытого ключа сервера. После успешного декодирования секрета (pre_master_secret) создается сообщение «finished», тем самым сервер демонстрирует, что он знает частный ключ соответствующий сертификату сервера.

Когда RSA используется для обмена ключами, для аутентификации клиента используется сообщение проверки сертификата клиента. Клиент подписывается значением, вычисленным из master_secret и всех предшествующих сообщений протокола рукопожатия. Эти сообщения рукопожатия включают сертификат сервера, который ставит в соответствие сигнатуре сервера, сообщение Server_Hello.random, которому ставит в соответствие сигнатуру текущему сообщению рукопожатия.

Обмен ключами при использовании Diffie-Hellman и аутентификация

В этом случае сервер может также поддерживать содержащий конкретные параметры алгоритм Диффи-Хеллмана или может использовать сообщения обмена ключами от сервера для посылки набора временных параметров подписанных сертификатами DSS или RSA. Временные параметры хэшируются с сообщением hello.random перед подписанием, для того чтобы злоумышленник не смог совершить повтор старых параметров. В любом случае клиент может проверить сертификат или сигнатуру, для уверенности, что параметры принадлежат серверу.

Если клиент имеет сертификат, содержащий параметры алгоритма Diffie-Hellman, то сертификат также содержит информацию требуемую для того чтобы завершить обмен ключами. Заметим, что в этом случае клиент и сервер должны будут сгенерировать те же Diffie-Hellman результаты (pre_master_secret), каждый раз, когда они устанавливают соединение. Для того чтобы предотвратить остановку секрета (pre_master_secret) в памяти компьютера на время дольше, чем необходимо, секрет должен быть переведен в общий секрет (master_secret) настолько быстро, на сколько это возможно. Параметры клиента должны быть совместимы с теми, которые поддерживает сервер для того, чтобы работал обмен ключами.

Протокол записи (Record Layer)

Протокол записи — это уровневый протокол. На каждом уровне сообщения включают поля для длины, описания и проверки. Протокол записи принимает сообщения, которые нужно передать, фрагментирует данные в управляемые блоки, разумно сжимает данные, применяя MAC (message authentication code), шифрует и передаёт результат. Полученные данные он расшифровывает, проверяет, распаковывает, собирает и доставляет к более верхним уровням клиента.

Существует четыре протокола записи: протокол рукопожатия (handshake protocol), протокол тревоги (alert protocol), протокол изменения шифра (the change cipher spec protocol), протокол приложения (application data protocol). Если SSL реализация получает тип записи, который ей неизвестен, то эта запись просто игнорируется. Любой протокол созданный для использования совместно с SSL должен быть хорошо продуман, так как будет иметь дело с атаками на него. Заметим, что из-за типа и длины записи, протокол не защищен шифрованием. Внимание следует уделить тому, чтобы минимизировать трафик.

Протокол рукопожатия (handshake)

SSL клиент и сервер договариваются об установлении связи с помощью процедуры рукопожатия. Во время рукопожатия клиент и сервер договариваются о различных параметрах, которые будут использованы, чтобы обеспечить безопасность соединения.

1) Рукопожатие начинается тогда, когда клиент подключается к SSL серверу. Запрос безопасного соединения представляет собой список поддерживаемых шифров и хэш-функций.
2) Из этого списка сервер выбирает самый сильный шифр и хэш-функцию, которую он также поддерживает, и уведомляет клиентов о принятом решении.
3) Сервер отсылает это решение в виде цифрового сертификата. Сертификат, обычно, содержит имя сервера, доверенный Центр Сертификации, и открытый ключ шифрования сервера. Клиент может связаться с сервером, который выдал сертификат (доверенного центра сертификации, выше) и убедиться, что сертификат является подлинным прежде чем продолжить.
4) Для того, чтобы сгенерировать ключи сеанса, используется безопасное соединение. Клиент шифрует случайное число с помощью открытого ключа (ОК) сервера и отправляет результат на сервер. Только сервер в состоянии расшифровать его (с его закрытым ключом (ЗК)), и только этот факт делает ключи скрытыми от третьей стороны, так как только сервер и клиент имели доступ к этим данным. Клиент знает открытый ключ и случайное число, а сервер знает закрытый ключ и (после расшифровки сообщения клиента) случайное число. Третья сторона, возможно, знает только открытый ключ, если закрытый ключ не был взломан.
5) Из случайного числа обе стороны создают ключевые данные для шифрования и расшифровывания.

На этом рукопожатие завершается, и начинается защищенное соединение, которое зашифровывается и расшифровывается с помощью ключевых данных. Если любое из перечисленных выше действий не удается, то рукопожатие SSL не удалось, и соединение не создается.

Протокол изменения параметров шифрования

(The Change Cipher Spec Protocol)

Он существует для сигнализации перехода в режим шифрования. Протокол содержит единственное сообщение, которое зашифровано и сжато при текущем установленном соединении. Сообщение состоит только из одного бита со значением 1.

struct { enum { change_cipher_spec(1), (255) } type; } ChangeCipherSpec;

Сообщение изменения шифра посылается и клиентом и сервером для извещения принимающей стороны, что последующие записи будут защищены в соответствии с новым договоренным CipherSpec и ключами. Принятие этого сообщения заставляет получателя отдать приказ уровню записи незамедлительно копировать состояние отложенного чтения в состояние текущего чтения. Сразу после послания этого сообщения, тот, кто послал должен отдать приказ уровню записи перевести режим отложенной записи в режим текущей записи. Сообщение изменения шифра посылается во время рукопожатия, после того как параметры защиты были переданы, но перед тем как будет послано сообщение ‘finished’.

Протокол тревоги (Alert Protocol)

Один из типов проверки, поддерживаемых в протоколе SSL записи, — это протокол тревоги. Сообщение тревоги передаёт трудности, возникшие в сообщении, и описание тревоги. Сообщение тревоги с критическим уровнем незамедлительно прерывает соединение. В этом случае другие соединения, соответствующие сессии, могут быть продолжены, но идентификатор сессии должен быть признан недействительным. Как и другие сообщения, сообщение тревоги зашифровано и сжато, как только указано текущее состояние соединения.

Протокол приложения (Application Data Protocol)

Сообщение приложения данных работает на уровне записи. Он фрагментируется, сжимается и шифруется на основе текущего состояния соединения. Сообщения считаются прозрачными для уровня записи.

Ошибки в протоколе SSL

В протоколе SSL обработка ошибок очень проста. Когда ошибка обнаружена, тот, кто её обнаружил, посылает об этом сообщение своему партнёру. Неустранимые ошибки требуют от сервера и клиента разрыва соединения. Протокол SSL определяет следующие ошибки:

Unsupported_Certificate_Type_Error: такая ошибка возникает, когда клиент/сервер получает тип сертификата, который не поддерживается. Ошибка устранима (только для аутентификации клиента).

No_Cipher_Error: ошибка возникает, когда сервер не может найти размер ключа или шифр, который поддерживается также и клиентом. Ошибка неустранима.

Bad_Certificate_Error: такая ошибка возникает, когда сертификат считается принимающей стороной плохим. Это означает, что или некорректна подпись сертификата, или его значение некорректно. Ошибка устранима (только для аутентификации клиента).

No_Certificate_Error: если послано сообщение Request_Certificate, то эта ошибка может быть прислана по причине того, что клиент не имеет сертификата. Ошибка устранима.

Перечислим некоторые виды атак, которые могут быть предприняты против протокола SSL. Однако, SSL устойчив к этим атакам:

Раскрытие шифров

Как известно, SSL зависит от различных криптографических параметров. Шифрование с открытым ключом RSA необходимо для пересылки ключей и аутентификации сервера/клиента. Однако в качестве шифра используются различные криптографические алгоритмы. Таким образом, если осуществить успешную атаку на эти алгоритмы, то SSL не может уже считаться безопасным. Атака на определенные коммуникационные сессии производится записью сессии, и потом, в течение долгого времени подбирается ключ сессии или ключ RSA. SSL же делает такую атаку невыгодной, так как тратится большое количество времени и денег.

Злоумышленник посередине

Также известна как MitM (Man-in-the-Middle) атака. Предполагает участие трех сторон: сервера, клиента и злоумышленника, находящегося между ними. В данной ситуации злоумышленник может перехватывать все сообщения, которые следуют в обоих направлениях, и подменять их. Злоумышленник представляется сервером для клиента и клиентом для сервера. В случае обмена ключами по алгоритму Диффи-Хелмана данная атака является эффективной, так как целостность принимаемой информации и ее источник проверить невозможно. Однако такая атака невозможна при использовании протокола SSL, так как для проверки подлинности источника (обычно сервера) используются сертификаты, заверенные центром сертификации.

Атака будет успешной, если:

- Сервер не имеет подписанного сертификата.
- Клиент не проверяет сертификат сервера.
- Пользователь игнорирует сообщение об отсутствии подписи сертификата центром сертификации или о несовпадении сертификата с кэшированным.

Атака отклика

Злоумышленник записывает коммуникационную сессию между сервером и клиентом. Позднее, он пытается установить соединение с сервером, воспроизводя записанные сообщения клиента. Но SSL отбивает эту атаку при помощи особого уникального идентификатора соединения (ИС). Конечно, теоретически третья сторона не в силах предсказать ИС, потому что он основан на наборе случайных событий. Однако, злоумышленник с большими ресурсами может записать большое количество сессий и попытаться подобрать «верную» сессию, основываясь на коде nonce, который послал сервер в сообщение Server_Hello. Но коды nonce SSL имеют, по меньшей мере, длину 128 бит, а значит, злоумышленнику необходимо записать 2127 кодов nonce, чтобы получить вероятность угадывания 50 %. Но 2127 достаточно большое число, чтобы сделать эти атаки бессмысленными.

Атака против протокола рукопожатия

Злоумышленник может попытаться повлиять на обмен рукопожатиями для того, чтобы стороны выбрали разные алгоритмы шифрования, а не те, что они выбирают обычно. Из-за того, что многие реализации поддерживают 40-битное экспортированное шифрование, а некоторые даже 0-шифрование или MAC-алгоритм, эти атаки представляют большой интерес.

Для такой атаки злоумышленнику необходимо быстро подменить одно или более сообщений рукопожатия. Если это происходит, то клиент и сервер вычислят различные значения хэшей сообщения рукопожатия. В результате чего стороны не примут друг от друга сообщения Finished. Без знания секрета злоумышленник не сможет исправить сообщение Finished, поэтому атака может быть обнаружена.

Алгоритмы, использующиеся в SSL:

- Для обмена ключами и проверки их подлинности применяются: RSA, Diffie-Hellman, ECDH, SRP, PSK.
- Для аутентификации: RSA, DSA, ECDSA.
- Для симметричного шифрования: RC2, RC4, IDEA, DES, Triple DES или AES, Camellia.
- Для хеш-функций: SHA, MD5, MD4 и MD2.

TLS

TLS (Transport Layer Security) — криптографический протокол, обеспечивающий защищённую передачу данных между узлами в сети Интернет.

TLS-протокол основан на протоколе Netscape SSL версии 3.0 и состоит из двух частей — TLS Record Protocol и TLS Handshake Protocol. Различия между SSL 3.0 и TLS 1.0 незначительны, но, всё же они есть.

TLS предоставляет возможности аутентификации и безопасной передачи данных через Интернет с использованием криптографических средств. Часто происходит лишь аутентификация сервера, в то время как клиент остается неаутентифицированным. Для взаимной аутентификации каждая из сторон должна поддерживать инфраструктуру открытого ключа (PKI), которая позволяет защитить клиент-серверные приложения от перехвата сообщений, редактирования существующих сообщений и создания поддельных.

SSL включает в себя три основных фазы:

- Диалог между сторонами, целью которого является выбор алгоритма шифрования
- Обмен ключами на основе криптосистем с открытым ключом или аутентификация на основе сертификатов.
- Передача данных, шифруемых при помощи симметричных алгоритмов шифрования

Клиент и сервер, работающие по TLS, устанавливают соединение, используя процедуру handshake («рукопожатие»). В течение этого handshake клиент и сервер принимают соглашение относительно параметров, используемых для установления защищенного соединения.

Последовательность действий при установлении TLS-соединения:

- клиент подключается к серверу, поддерживающему TLS, и запрашивает защищенное соединение;
- клиент предоставляет список поддерживаемых алгоритмов шифрования и хеш-функций;
- сервер выбирает из списка, предоставленного клиентом, наиболее устойчивые алгоритмы, которые также поддерживаются сервером, и сообщает о своем выборе клиенту;
- сервер отправляет клиенту цифровой сертификат для собственной аутентификации. Обычно цифровой сертификат содержит имя сервера, имя доверенного центра сертификации и открытый ключ сервера;

- клиент может связаться с сервером доверенного центра сертификации и подтвердить аутентичность переданного сертификата до начала передачи данных;
- для того чтобы сгенерировать сеансовый ключ для защищенного соединения, клиент шифрует случайно сгенерированную цифровую последовательность открытым ключом сервера и посылает результат на сервер. Учитывая специфику алгоритма асимметричного шифрования, используемого для установления соединения, только сервер может расшифровать полученную последовательность, используя свой закрытый ключ;

Согласно протоколу TLS приложения обмениваются записями, инкапсулирующими (хранящими внутри себя) информацию, которая должна быть передана. Каждая из записей может быть сжата, дополнена, зашифрована или идентифицирована MAC (код аутентификации сообщения) в зависимости от текущего состояния соединения (состояния протокола). Каждая запись в TLS содержит следующие поля: content type (определяет тип содержимого записи), поле, указывающее длину пакета, и поле, указывающее версию протокола TLS.

Когда соединение только устанавливается, взаимодействие идет по протоколу TLS handshake, content type которого 22.

Простой пример установления соединения:

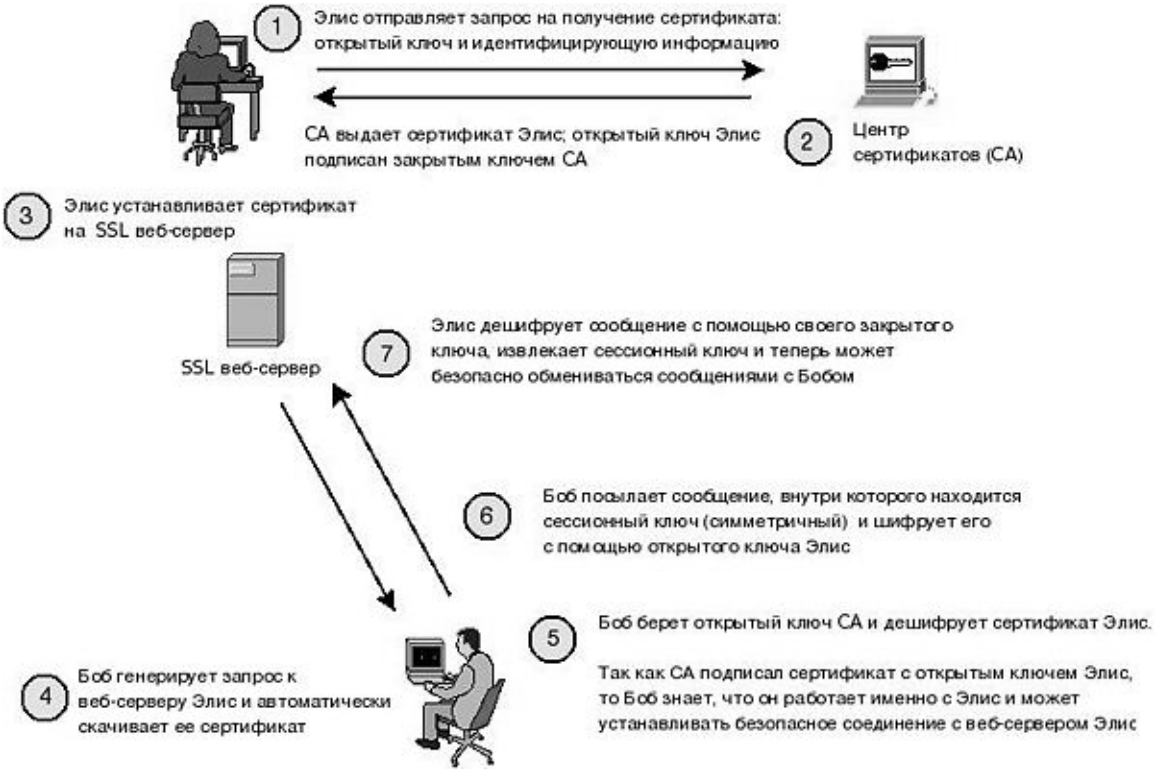
- Клиент посылает сообщение ClientHello, указывая наиболее последнюю версию поддерживаемого TLS протокола, случайное число и список поддерживаемых методов шифрования и сжатия, подходящих для работы с TLS.
- Сервер отвечает сообщением ServerHello, содержащим: выбранную сервером версию протокола, случайное число, посланное клиентом, подходящий алгоритм шифрования и сжатия из списка предоставленного клиентом.
- Сервер посылает сообщение Certificate, которое содержит цифровой сертификат сервера (в зависимости от алгоритма шифрования этот этап может быть пропущен)
- Сервер может запросить сертификат у клиента, в таком случае соединение будет взаимно аутентифицировано.
- Сервер отсылает сообщение ServerHelloDone, идентифицирующее окончание handshake.
- Клиент отвечает сообщением ClientKeyExchange, которое содержит открытый ключ PreMasterSecret или ничего (опять же зависит от алгоритма шифрования).

- Клиент и сервер, используя ключ PreMasterSecret и случайно сгенерированные числа, вычисляют общий секретный ключ. Вся остальная информация о ключе будет получена из общего секретного ключа (и сгенерированных клиентом и сервером случайных значений).
- Клиент посылает сообщение ChangeCipherSpec, которое указывает на то, что вся последующая информация будет зашифрована установленным в процессе handshake алгоритмом, используя общий секретный ключ. Это сообщения уровня записей и поэтому имеет тип 20, а не 22.
- Клиент посылает сообщение Finished, которое содержит хеш и MAC, сгенерированные на основе предыдущих сообщений handshake.
- Сервер пытается расшифровать Finished-сообщение клиента и проверить хеш и MAC. Если процесс расшифровки или проверки не удается, handshake считается неудавшимся, и соединение должно быть оборвано.
- Сервер посылает ChangeCipherSpec и зашифрованное сообщение Finished, и в свою очередь клиент тоже выполняет расшифровку и проверку.

С этого момента handshake считается завершённым, протокол установленным. Все последующее содержимое пакетов идет с типом 23, а все данные будут зашифрованы.

Также можно привести ещё один пример работы протокола с более подробным объяснением:

На рисунке описаны основные сообщения, которыми обмениваются пользователь (Боб) и веб-сервер (Элис). Пользователь, использует браузер для доступа к серверу, который поддерживает SSL. Такая ситуация может возникать, например, при заказе товара по Интернет.



Прежде чем веб-сервер начнет взаимодействовать с клиентом, на веб-сервере должен быть установлен сертификат подписанный центром сертификатов (CA). Браузер клиента должен знать этот центр сертификатов (доверять ему). Это значит, что сертификат этого центра сертификатов уже установлен в браузере и срок действия сертификата не истек. Если сертификат не установлен, то при доступе клиента к веб-серверу, у него высветится сообщение предлагающее принять или отклонить сертификат веб-сервера, так как он подписан неизвестным центром сертификатов.

Для получения сертификата Элис необходимо:

1. Подготовить запрос на получение сертификата для отправки его CA:
   - Сгенерировать пару открытый/закрытый ключ (с помощью утилиты генерации ключей, которая обычно есть в веб-сервере).
   - Сгенерировать запрос на получение сертификата, который включает в себя идентифицирующую информацию о компании Элис (Distinguished Name) и открытый ключ.
   - Отправить запрос на получение сертификата выбранному CA (например, запрос может быть отправлен почтой).
2. В ответ на запрос CA отправляет Элис цифровой сертификат, в котором содержится открытый ключ Элис, подписанный CA.
3. После получения ответа от CA, Элис помещает цифровой сертификат в базу сертификатов веб-сервера. После того как Элис установила

цифровой сертификат подписанный CA, она может начинать работу с клиентами. Когда Боб обращается к веб-серверу Элис, выполняются такие действия:

4. Боб обращается с веб-серверу Элис. При этом броузер Боба автоматически скачает цифровой сертификат с веб-сервера Элис.
5. Браузер Боба определит доверяет ли он CA, который подписал цифровой сертификат Элис:

- Сертификат CA дожен быть уже установлен в браузере Боба.
- Если сертификат установлен и срок его действия не истек, то дальнейший процесс будет прозрачным для Боба.
- Браузер дешифрует цифровой сертификат Элис с помощью открытого ключа CA, который получен из цифрового сертификата CA.
- Теперь у браузера Боба есть открытый ключ Элис.

6. Боб отправляет Элис сообщение в котором содержится сессионный ключ и шифрует его с помощью открытого ключа Элис:

- Сессионный ключ используется для шифрования трафика между Бобом и Элис.
- Этот ключ симметричный и будет использоваться для шифрования и дешифрования данных передаваемых между Бобом и Элис.
- Так как сообщение можно расшифровать только закрытым ключем Элис, то сессионный ключ безопасно доставляется на веб-сервер Элис.

7. Как только веб-сервер получает сообщение в котором хранится сессионный ключ, он его дешифрует с помощью закрытого ключа Элис.

- Все дальнейшие сообщения шифруются с помощью сессионного ключа (может передаваться информация о адресе, телефоне, кредитной карточке Боба).
- В зависимости от длительности соединения с веб-сервером, может использоваться несколько сессионных ключей, которые будут меняться после истечения их срока жизни.

В данной текущей версии протокола доступны следующие алгоритмы:

- Для обмена ключами и проверки их подлинности применяются комбинации алгоритмов: RSA (асимметричный шифр), Diffie-Hellman (безопасный обмен ключами), DSA (алгоритм цифровой подписи) и алгоритмы технологии Fortezza.
- Для симметричного шифрования: RC2, RC4, IDEA, DES, Triple DES или AES;

- Для хеш-функций: MD5 или SHA.

Алгоритмы могут дополняться в зависимости от версии протокола.

Отличия SSL и TLS

Отличия между протоколами TLS 1.0 и SSL 3.0 небольшие, но протоколы не взаимозаменяемы. Для взамодействия двух сторон необходимо чтобы обе стороны использовали или SSL или TLS.

Улучшения в протоколе TLS, по сравнению с SSL:

- Вместо алгоритма Message Authentication Code (MAC), который использовался в SSL, в TLS используется Hash Message Authentication Code (HMAC).
- Добавлены новые alert сообщения.
- Небольшие изменения существуют в полях некоторых сообщений.

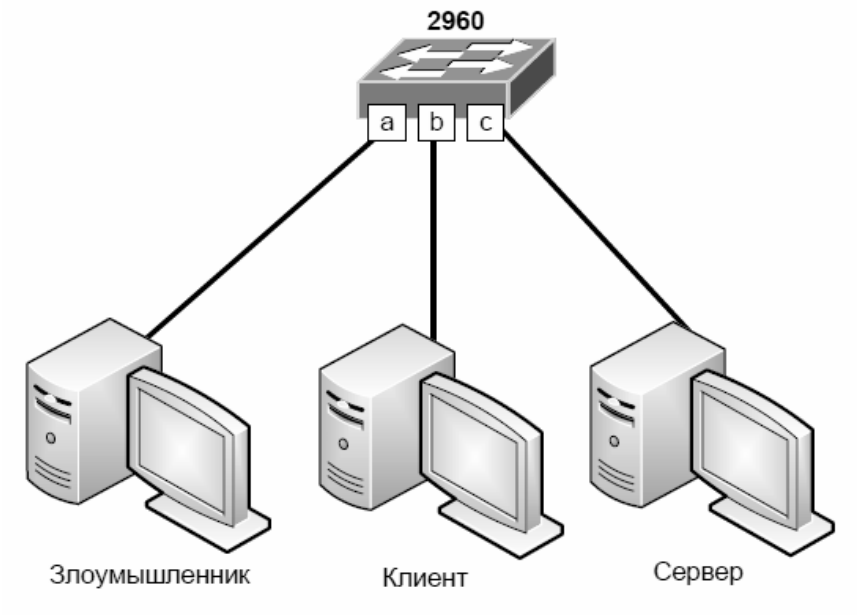
Примеры использования TLS/SSL:

- Безопасные SSL-соединения с сайтами электронной коммерции
- Доступ к сайтам, которые требуют аутентификации клиента
- Удаленный доступ
- Доступ к базам данных
- Почта

Практическая часть

Цель работы:

Изучение принципов безопасного обмена информации с использованием протокола SSL и ОС Linux.



Рисунок

Порядок выполнения работы:

1. Соберите топологию сети, представленную на рисунке

2. Настройте «зеркалирование» (Port Mirroring) на коммутаторе следующим образом: порт «a» – приемник, порт «b» – источник. Таким образом, злоумышленник сможет «прослушивать» весь трафик клиента.

3. Запустите на сервер POP3-сервер с авторизацией через ***/etc/passwd***.

4. Получите почту пользователя *root* с машины клиента. Параллельно с этим процессом запустите утилиту ***tcpdump*** на компьютере злоумышленника. Обнаружьте пароль в перехваченных пакетах на почтовый ящик.

5. На сервере запустите утилиту ***stunnel*** на 995 порту для запуска почтового сервера.

6. Включите использование SSL/TLS на клиенте. Выполните шаг 5.

7. Сравните перехваченные утилитой ***tcpdump*** пакеты.

Контрольные вопросы

1. Что подразумевается под термином «SSL», «TLS»?
2. Где можно применить SSL и TLS протоколы?
3. Какие протоколы записи SSL можно выделить и в чём их отличие?
4. Какие атаки можно проводить на SSL и в чём заключается основная идея атак?

Замечания

Пример зеркалирования трафика на коммутаторах Cisco

Настройка источника трафика (в режиме конфигурации):

 # monitor session 1 source interface fastethernet 0/10 both

Настройка получателя трафика (порт, к которому подключен анализатор трафика)(в режиме конфигурации):

 # monitor session 1 destination interface fastethernet 0/1