Лекция 19

SSL и TLS

SSL

SSL (secure sockets layer — уровень защищённых сокетов) — криптографический протокол сеансового уровня для обеспечения безопасной связи. Использует асимметричную криптографию для аутентификации ключей обмена, симметричное шифрование для сохранения конфиденциальности и коды аутентификации сообщений для целостности сообщений.
В настоящее время протокол считается небезопасным и должен быть заменен на протокол TLS, который является его дальнейшим развитием.

Протокол SSL предоставляет "безопасный канал", который имеет три основных свойства:

- *Канал является частным.* Шифрование используется для всех сообщений после простого диалога, который служит для определения секретного ключа.
- *Канал аутентифицирован.* Серверная сторона диалога всегда аутентифицируется, а клиентская делает это опционно.
- *Канал надежен.* Транспортировка сообщений включает в себя проверку целостности.

SSL независим от прикладного протокола. Протоколы приложений (HTTP, FTP, TELNET и т.д.) могут работать поверх протокола SSL совершенно прозрачно — SSL может согласовывать алгоритм шифрования и ключ сессии, а также аутентифицировать сервер до того, как приложение примет или передаст первый байт сообщения.

Протокол SSL был изначально разработан компанией Netscape Communications. Версия 1.0 никогда не была обнародована. Версии 2.0 была выпущена в феврале 1995 года, но содержала много недостатков по безопасности, которые привели к разработке SSL версии 3.0. SSL версии 3.0, выпущенный в 1996 году, послужил основой для создания протокола TLS 1.0, стандарт протокола Internet Engineering Task Force (IETF), который впервые был определен в RFC 2246 в январе 1999 года. TLS 1.1 был представлен в RFC 4346 в апреле 2006 года. В августе 2008 года в RFC 5246 был анонсирован TLS 1.2.

SSL использует среду с несколькими слоями, что обеспечивает безопасность обмена информацией. Конфиденциальность общения присутствует за счет того, что безопасное соединение открыто только целевым пользователям.

Протокол SSL размещается между двумя протоколами: протоколом, который использует программа-клиент (HTTP, FTP, TELNET) и транспортным протоколом TCP/IP. Работу протокола можно разделить на два уровня:

1. Слой протокола подтверждения подключения (Handshake Protocol Layer)
2. Слой протокола записи

Первый слой, в свою очередь, состоит из трех подпротоколов:

1. Протокол подтверждения подключения (Handshake Protocol)
2. Протокол изменения параметров шифра (Cipher Spec Protocol)
3. Предупредительный протокол (Alert Protocol)

Протокол подтверждения подключения используется для согласования данных сессии между клиентом и сервером. К данным сессии относятся:

- ID сессии
- Сертификаты обеих сторон
- Параметры алгоритма шифрования
- Алгоритм сжатия информации
- Информация для создания симметр. ключей; открытый ключ

Протокол изменения параметров шифра используется для изменения данных ключа (keyingmaterial) - информации, которая используется для создания ключей шифрования. Протокол состоит всего из одного сообщения, в котором сервер говорит, что отправитель хочет изменить набор ключей.

Предупредительный протокол содержит сообщение, которое показывает сторонам изменение статуса или сообщает о возможной ошибке. Обычно предупреждение отсылается тогда, когда подключение закрыто и получено неправильное сообщение, сообщение невозможно расшифровать или пользователь отменяет операцию.

Протокол подтверждения подключения производит цепочку обмена данными, что в свою очередь начинает аутентификацию сторон и согласовывает шифрование, хэширование и сжатие. Следующий этап - *аутентификация участников*, которая осуществляется также протоколом подтверждения подключения.

Для проверки соединения SSL использует сертификаты. Сертификаты расположены на безопасном сервере и используются для шифрования данных и идентификации Web-сайта.

Способы получения SSL-сертификата:

- Использовать сертификат, выданный центром сертификации
- Использовать самоподписанный сертификат
- Использовать "пустой" сертификат

Центр сертификации или *удостоверяющий центр* (Certification authority, CA) — сторона (отдел, организация), чья честность неоспорима, а открытый ключ широко известен. Задача центра сертификации — подтверждать подлинность ключей шифрования с помощью сертификатов электронной подписи, имеющих следующую структуру:

- серийный номер сертификата;
- объектный идентификатор алгоритма электронной подписи;
- имя удостоверяющего центра;
- срок действия сертификата;
- имя владельца сертификата (имя пользователя, которому принадлежит сертификат);
- открытые ключи владельца сертификата (ключей может быть несколько);
- объектные идентификаторы алгоритмов, ассоциированных с открытыми ключами владельца сертификата;
- электронная подпись, сгенерированная с использованием секретного ключа удостоверяющего центра (подписывается результат хэширования всей информации, хранящейся в сертификате).

Самоподписанный сертификат — сертификат, созданный самим пользователем - в этом случае издатель сертификата совпадает с владельцем сертификата.
“Пустой” сертификат — сертификат, содержащий фиктивную информацию, используемую в качестве временной для настройки SSL и проверки его функциональности в данной среде.

Механизмы образования ключа для текущей сессии в SSL/TLS

1. *RSA*. При утере приватного ключа криптоаналитик получает возможность расшифровать все записанные прошлые сообщения и будущие сообщения.
2. *Fixed Diffie-Hellman.* Использует постоянный публичный ключ, который прописан в сертификате сервера. При каждом новом соединении, клиент предоставляет свою часть ключа. После обмена ключами, образуется новый симметричный ключ для обмена информацией для текущей сессии. При раскрытии приватного ключа сервера, криптоаналитик сможет расшифровать ранее записанные сообщения, а также все

будущие сообщения.

3. *Anonymous Diffie-Hellman.* Не предоставляет гарантий секретности, ибо данные передаются незашифрованными.
4. *Ephemeral Diffie-Hellman.* Гарантирует безопасность прошлых и будущих сообщений. При каждом новом соединении сервером и клиентом создается одноразовый ключ. Даже если криптоаналитику достанется текущий частный ключ, он сможет расшифровать только текущую сессию, но не предыдущие или будущие сессии.

Протокол диалога SSL

Протокол диалога SSL содержит 2 основные фазы.

Первая фаза используется *для установления конфиденциального канала* коммуникаций.
Эта фаза инициализирует соединение, когда оба партнера обмениваются сообщениями “hello”. Клиент посылает сообщение CLIENT-HELLO. Сервер получает это сообщение, обрабатывает его и посылает в ответ сообщение SERVER-HELLO.
В этот момент и сервер и клиент имеют достаточно информации, чтобы знать, нужен ли новый master key. Если ключ не нужен, сервер и клиент переходят в фазу 2.
Когда возникает необходимость создания нового master key, сообщение сервера SERVER-HELLO уже содержит достаточно данных для того, чтобы клиент мог сгенерировать master key. В эти данные входят подписанный сертификат сервера, список базовых шифров и идентификатор соединения (случайное число, сгенерированное сервером, которое используется на протяжении всей сессии). После генерации клиентом master key он посылает серверу сообщение CLIENT-MASTER-KEY или же сообщение об ошибке, когда клиент и сервер не могут согласовать базовый шифр.
После определения master key сервер посылает клиенту сообщение SERVER-VERIFY, что аутентифицирует сервер.

Фаза 2 называется фазой аутентификации. Т. к. сервер уже аутентифицирован на первой фазе, то на второй фазе осуществляется аутентификация клиента. Сервер отправляет запрос клиенту и, если у клиента есть необходимая информация, он присылает позитивный отклик, если же нет, то сообщение об ошибке. Когда один партнер выполнил аутентификацию другого партнера, он посылает сообщение finished. В случае клиента сообщение CLIENT-FINISHED содержит зашифрованную форму идентификатора CONNECTION-ID, которую должен верифицировать сервер. Если верификация была неудачной, сервер посылает сообщение ERROR.
Когда один из партнеров послал сообщение finished он должен принимать сообщения до тех пор, пока не получит сообщение finished от другого партнера, и только когда оба партнера послали и получили сообщения finished протокол диалога SSL закончит свою работу. С этого момента начинает работу прикладной протокол.

TLS

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

Так как большинство протоколов связи могут быть использованы как с, так и без TLS (или SSL), при установке соединения необходимо явно указать серверу, хочет ли клиент устанавливать TLS. Это может быть достигнуто либо с помощью использования унифицированного номера порта, по которому соединение всегда устанавливается с использованием TLS (например, порт 443 для HTTPS), либо с использованием произвольного порта и специальной команды серверу со стороны клиента на переключение соединения на TLS с использованием специальных механизмов протокола (например, STARTTLS для протоколов электронной почты). Как только клиент и сервер договорились об использовании TLS, им необходимо установить защищённое соединение. Это делается с помощью процедуры подтверждения связи. Во время этого процесса клиент и сервер принимают соглашение относительно различных параметров, необходимых для установки безопасного соединения.

Основные шаги процедуры создания защищённого сеанса связи:

1. клиент подключается к серверу, поддерживающему TLS, и запрашивает защищённое соединение;
2. клиент предоставляет список поддерживаемых алгоритмов шифрования и хеш-функций;
3. сервер выбирает из списка, предоставленного клиентом, наиболее надёжные алгоритмы среди тех, которые поддерживаются сервером, и сообщает о своём выборе клиенту;
4. сервер отправляет клиенту цифровой сертификат для собственной аутентификации. Обычно цифровой сертификат содержит имя сервера, имя удостоверяющего центра сертификации и открытый ключ сервера;
5. клиент может связаться с сервером доверенного центра сертификации и подтвердить аутентичность переданного сертификата до начала передачи данных;
6. для генерации сеансового ключа для защищённого 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.

На этом заканчивается процедура подтверждения связи. Между клиентом и сервером установлено безопасное соединение, данные, передаваемые по нему, шифруются и расшифровываются с использованием ключа шифрования до тех пор, пока соединение не будет завершено.

При возникновении ошибки на любом из вышеуказанных шагов подтверждение связи завершится с ошибкой и соединение не будет установлено.

Меры безопасности в TLS:

- Защита от понижения версии протокола к предыдущей (менее защищённой) версии или менее надёжному алгоритму шифрования;
- Нумерация последовательных записей приложения и использование порядкового номера в коде аутентификации сообщения (MAC);
- Использование ключа в идентификаторе сообщения (только владелец ключа может проверить код аутентификации сообщения). Хеш-код идентификации сообщений (HMAC), используемый в большинстве шифров из набора шифров TLS, был определён в RFC 2104;
- Сообщение, которым заканчивается подтверждение связи («Finished»), содержит в себе хэш всех сообщений, которыми обменялись стороны в процессе подтверждения связи;
- Псевдослучайная функция разбивает входные данные на две части и обрабатывает каждую разной хэш-функцией (MD5 и SHA-1), а затем вычисляет XOR от двух полученных свёрток, чтобы создать код аутентификации сообщения. Это обеспечивает безопасность даже в случае уязвимости одной из хэш-функций.

Процедура подтверждения связи в TLS в деталях

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 идёт по протоколу TLS handshake, content type которого 22.

Простое подтверждение связи в TLS

Далее показан простой пример установления соединения, при котором сервер (но не клиент) проходит аутентификацию по его сертификату.

1. **Фаза переговоров:**

- Клиент посылает сообщение ClientHello, указывая 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 отсылает сообщение ServerHelloDone, идентифицирующее окончание подтверждения связи;
- Клиент отвечает сообщением ClientKeyExchange, которое содержит открытый ключ PreMasterSecret(Этот PreMasterSecret шифруется с помощью открытого ключа сертификата сервера) 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);

2. **Клиент посылает сообщение ChangeCipherSpec**, которое указывает на то, что вся последующая информация будет зашифрована установленным в процессе подтверждения связи 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 процедуры подтверждения связи;
- Сервер пытается расшифровать Finished-сообщение клиента и проверить хеш и MAC. Если процесс расшифровки или проверки не удаётся, подтверждение связи считается неудавшимся, и соединение должно быть оборвано;
3. **Сервер посылает ChangeCipherSpec и зашифрованное сообщение Finished**, и в свою очередь клиент тоже выполняет расшифровку и проверку.

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

Возобновление TLS-соединения

Алгоритмы асимметричного шифрования, использующиеся при генерации сеансового ключа, обычно являются дорогими с точки зрения вычислительных мощностей. Для того чтобы избежать их повторения при возобновлении соединения, TLS создаёт специальный ярлык при подтверждении связи, использующийся для возобновления соединения. При этом при обычном подтверждении связи клиент добавляет в сообщение ClientHello идентификатор предыдущей сессии session id. Клиент связывает идентификатор session id с IP-адресом сервера и TCP-портом так, чтобы при соединении к серверу можно было использовать все параметры предыдущего соединения. Сервер сопоставляет идентификатор предыдущей сессии session id с параметрами соединения, такими как использованный алгоритм шифрования и master secret. Обе стороны должны иметь одинаковый master secret, иначе соединение не будет установлено. Это предотвращает использование session id криптоаналитиком для получения несанкционированного доступа. Случайные цифровые последовательности в сообщениях ClientHello и ServerHello позволяют гарантировать, что сгенерированный сеансовый ключ будет отличаться от сеансового ключа при предыдущем соединении. В RFC такой тип подтверждения связи называется сокращённым.

1. Фаза переговоров:
- Клиент посылает сообщение ClientHello, указывая последнюю версию поддерживаемого TLS протокола, случайное число и список поддерживаемых методов шифрования и сжатия, подходящих для работы с TLS; Также в сообщение добавляется идентификатор предыдущего соединения session id.
- Сервер отвечает сообщением ServerHello, содержащим: выбранную сервером версию протокола, случайное число, посланное клиентом, подходящий алгоритм шифрования и сжатия из списка предоставленного клиентом. Если сервер узнал идентификатор сессии session id, то он добавляет в сообщение ServerHello тот же самый идентификатор session id. Это является сигналом для клиента о том, что можно использовать возобновление предыдущей сессии. Если сервер не узнал идентификатор сессии session id, то он добавляет в сообщение ServerHellодругое значение вместо session id. Для клиента это означает, что использовать возобновлённое соединение нельзя. Таким образом, сервер и клиент должны иметь одинаковый master secret и случайные числа для генерации сеансового ключа;
2. Сервер посылает сообщение ChangeCipherSpec, которое указывает на то, что вся последующая информация будет зашифрована установленным в процессе подтверждения связи алгоритмом, используя общий секретный ключ. Это сообщения уровня записей и поэтому имеет тип 20, а не 22;
- Сервер посылает зашифрованное сообщение Finished, которое содержит хеш и MAC, сгенерированные на основе предыдущих сообщений процедуры подтверждения связи;
- Клиент пытается расшифровать Finished сообщение сервера и проверить хеш и MAC. Если процесс расшифровки или проверки не удаётся, подтверждение связи считается

неудавшимся, и соединение должно быть оборвано;

3. Клиент посылает сообщение ChangeCipherSpec, которое указывает на то, что вся последующая информация будет зашифрована установленным в процессе подтверждения связи алгоритмом, используя общий секретный ключ.
- Клиент посылает своё зашифрованное сообщение Finished;
- Сервер схожим образом пытается расшифровать Finished сообщение клиента и проверить хеш и MAC;

С этого момента подтверждение связи считается завершённым, протокол установленным. Всё последующее содержимое пакетов идёт с типом 23, а все данные будут зашифрованы.

Кроме преимуществ с точки зрения производительности, алгоритм возобновления соединения может быть использован для реализации единого входа, поскольку гарантируется, что исходной сессия, как и любая возобновлённая сессия, инициирована тем же самым клиентом (RFC 5077). Это имеет особенно важное значение для реализации FTPS протокола, который в противном случае был бы уязвим к атаке типа человек посередине, при которой злоумышленник мог бы перехватить содержание данных при установлении повторного соединения.

Мандаты сессий

RFC 5077 расширяет TLS через использование мандатов сессий (session tickets), вместо идентификаторов соединений (session id). Он определяет способ возобновления сеанса TLS, не требуя session id придыдущей сессии, состояние которой хранится на TLS сервере.

При использовании сессионных мандатов, TLS сервер хранит сеансовое состояние в мандате сеанса и посылает его для хранения на TLS клиенте. Клиент возобновляет TLS сессию отправив мандат сеанса на сервер, а сервер возобновляет TLS сессию в соответствии с параметрами конкретной сессии, сохранёнными в принятом мандате. Сессионный мандат шифруется, в зашифрованном виде проходит аутентификацию на сервере, и сервер проверяет его обоснованность прежде чем использовать содержимое мандата.

Одна из слабостей этого метода — для шифрования и аутентификации передаваемых сессионных мандатов всегда используется только метод AES128-CBC-SHA256, независимо от того, какие параметры TLS выбраны и используются для самого TLS соединения. Это означает, что информация о TLS сессии (сохраняемая в сессионном мандате) не так хорошо защищена, как в рамках самой TLS сессии.

Алгоритмы, использующиеся в TLS

- *Для обмена ключами и проверки их подлинности* применяются комбинации алгоритмов: RSA (асимметричный шифр), Diffie-Hellman (безопасный обмен ключами), DSA (алгоритм цифровой подписи), ECDSA;
- *Для симметричного шифрования:* RC4, IDEA, Triple DES, SEED, Camellia или AES;
- *Для хеш-функций:* MD5, SHA, SHA-256/384.