Лекция 5

TLS/SSL

- TLS и SSL используют асимметричную криптографию для аутентификации, симметричное шифрование для конфиденциальности и коды аутентичности сообщений для сохранения целостности сообщений

SSL 2.0 problems

- Message authentication uses MD5 [MD5]. Most security-aware users have already moved away from any use of MD5 [RFC6151].
- Handshake messages are not protected. This permits a man-in-the- middle to trick the client into picking a weaker cipher suite than it would normally choose.
- Message integrity and message encryption use the same key, which is a problem if the client and server negotiate a weak encryption algorithm.
- Sessions can be easily terminated. A man-in-the-middle can easily insert a TCP FIN to close the session, and the peer is unable to determine whether or not it was a legitimate end of the session.

TLS changes

- TLS clients MUST NOT send the SSL version 2.0 compatible CLIENT- HELLO message format. Clients MUST NOT send any ClientHello message that specifies a protocol version less than { 0x03, 0x00 }. As previously stated by the definitions of all previous versions of TLS, the client SHOULD specify the highest protocol version it supports.
- TLS servers MAY continue to accept ClientHello messages in the version 2 CLIENT-HELLO format as specified in RFC 5246 [TLS1.2], Appendix E.2. Note that this does not contradict the prohibition against actually negotiating the use of SSL 2.0.
- TLS servers MUST NOT reply with an SSL 2.0 SERVER-HELLO with a protocol version that is less than { 0x03, 0x00 } and instead MUST abort the connection, i.e., when the highest protocol version offered by the client is { 0x02, 0x00 }, the TLS connection will be refused.
- Note that the number of servers that support this above-mentioned "MAY accept" implementation option is declining, and the SSL 2.0 CLIENT-HELLO precludes the use of TLS protocol enhancements that require TLS extensions. TLS extensions can only be sent as part of an (Extended) ClientHello handshake message.

TLS 1.2

- goal : provide privacy and data integrity between two communicating applications
- two layers: the TLS Record Protocol and the TLS Handshake Protocol. At the lowest level, layered on top of some reliable transport protocol (e.g., TCP [TCP]), is the TLS Record Protocol. The TLS Record Protocol provides connection security.
  - - The connection is private. Symmetric cryptography is used for data encryption (e.g., AES [AES], RC4 [SCH], etc.). The keys for this symmetric encryption are generated uniquely for each connection and are based on a secret negotiated by another protocol (such as the TLS Handshake Protocol). The Record Protocol can also be used without encryption.
  - - The connection is reliable. Message transport includes a message integrity check using a keyed MAC. Secure hash functions (e.g., SHA-1, etc.) are used for MAC computations. The Record Protocol can operate without a MAC, but is generally only used in this mode while another protocol is using the Record Protocol as a transport for negotiating security parameters.

TLS Handshake Protocol

- allows the server and client to authenticate each other and to negotiate an encryption algorithm and cryptographic keys before the application protocol transmits or receives its first byte of data. The TLS Handshake Protocol provides connection security :
  - - The peer's identity can be authenticated using asymmetric, or public key, cryptography (e.g., RSA [RSA], DSA [DSS], etc.). This authentication can be made optional, but is generally required for at least one of the peers.
  - - The negotiation of a shared secret is secure: the negotiated secret is unavailable to eavesdroppers, and for any authenticated connection the secret cannot be obtained, even by an attacker who can place himself in the middle of the connection.
  - - The negotiation is reliable: no attacker can modify the negotiation communication without being detected by the parties to the communication.

Design variation

- TLS standard, however, does not specify how protocols add security with TLS; the decisions on how to initiate TLS handshaking and how to interpret the authentication certificates exchanged are left to the judgment of the designers and implementors of protocols that run on top of TLS.

Difference TLS1.1 / 1.2

- The MD5/SHA-1 combination in the pseudorandom function (PRF) has been replaced with cipher-suite-specified PRFs. All cipher suites in this document use P_SHA256.
- - The MD5/SHA-1 combination in the digitally-signed element has been replaced with a single hash. Signed elements now include a field that explicitly specifies the hash algorithm used.
- - Substantial cleanup to the client's and server's ability to specify which hash and signature algorithms they will accept. Note that this also relaxes some of the constraints on signature and hash algorithms from previous versions of TLS.
- - Addition of support for authenticated encryption with additional data modes. - TLS Extensions definition and AES Cipher Suites were merged in from external [TLSEXT] and [TLSAES].
- - Tighter checking of EncryptedPreMasterSecret version numbers.
- - Tightened up a number of requirements.
- - Verify_data length now depends on the cipher suite (default is still 12).
- - Cleaned up description of Bleichenbacher/Klima attack defenses.

Handshake steps

- - Exchange hello messages to agree on algorithms, exchange random values, and check for session resumption.
- - Exchange the necessary cryptographic parameters to allow the client and server to agree on a premaster secret.
- - Exchange certificates and cryptographic information to allow the client and server to authenticate themselves.
- - Generate a master secret from the premaster secret and exchanged random values.
- - Provide security parameters to the record layer.
- - Allow the client and server to verify that their peer has calculated the same security parameters and that the handshake occurred without tampering by an attacker.

TLS security

- Note that higher layers should not be overly reliant on whether TLS always negotiates the strongest possible connection between two peers. There are a number of ways in which a man-in-the-middle attacker can attempt to make two entities drop down to the least secure method they support.
- The protocol has been designed to minimize this risk, but there are still attacks available: for example, an attacker could block access to the port a secure service runs on, or attempt to get the peers to negotiate an unauthenticated connection.
- The fundamental rule is that higher levels must be cognizant of what their security requirements are and never transmit information over a channel less secure than what they require.
- The TLS protocol is secure in that any cipher suite offers its promised level of security: if you negotiate 3DES with a 1024-bit RSA key exchange with a host whose certificate you have verified, you can expect to be that secure.
- These goals are achieved by the handshake protocol, which can be summarized as follows:
  - The client sends a ClientHello message to which the server must respond with a ServerHello message, or else a fatal error will occur and the connection will fail.
  - The ClientHello and ServerHello are used to establish security enhancement capabilities between client and server. The ClientHello and ServerHello establish the following attributes: Protocol Version, Session ID, Cipher Suite, and Compression Method.
  - Additionally, two random values are generated and exchanged: ClientHello.random and ServerHello.random.

Key exchange

- the server Certificate, the ServerKeyExchange, the client Certificate, and the ClientKeyExchange.
- New key exchange methods can be created by specifying a format for these messages and by defining the use of the messages to allow the client and server to agree upon a shared secret.
- This secret MUST be quite long; currently defined key exchange methods exchange secrets that range from 46 bytes upwards.

Steps

- клиент подключается к серверу, поддерживающему TLS, и запрашивает защищённое соединение;
- клиент предоставляет список поддерживаемых алгоритмов шифрования и хеш-функций;
- сервер выбирает из списка, предоставленного клиентом, наиболее надёжные алгоритмы среди тех, которые поддерживаются сервером, и сообщает о своём выборе клиенту;
- сервер отправляет клиенту цифровой сертификат для собственной аутентификации. Обычно цифровой сертификат содержит имя сервера, имя удостоверяющего центра сертификации и открытый ключ сервера;
- клиент, до начала передачи данных, проверяет валидность (аутентичность) полученного серверного сертификата, относительно имеющихся у клиента корневых сертификатов удостоверяющих центров (центров сертификации). Клиент также может проверить не отозван ли серверный сертификат, связавшись с сервисом доверенного удостоверяющего центра;
- для шифрования сессии используется сеансовый ключ. Получение общего секретного сеансового ключа клиентом и сервером проводится по протоколу Диффи-Хеллмана. Существует исторический метод передачи сгенерированного клиентом секрета на сервер, при помощи шифрования асимметричной криптосистемой RSA (используется ключ из сертификата сервера). Данные метод не рекомендован, но иногда продолжает встречаться на практике.

Hello messages

- Фаза переговоров:
  - Клиент посылает сообщение **ClientHello**, указывая последнюю версию поддерживаемого TLS протокола, случайное число и список поддерживаемых шифронаборов (методов шифрования), подходящих для работы с TLS;
  - Сервер отвечает сообщением **ServerHello**, содержащим: выбранную сервером версию протокола, случайное число, сгенерированное сервером, выбранный шифронабор из списка, предоставленного клиентом;
  - Сервер посылает сообщение **Certificate**, которое содержит цифровой сертификат сервера (в зависимости от алгоритма шифрования этот этап может быть пропущен);
  - Если переданных сервером данных недостаточно для выработки общего симметричного секретного ключа в рамках выбранного шифронабора, сервер передает сообщение **ServerKeyExchange**, в котором передаются необходимые данные. Например, в ServerKeyExchange передаётся серверная часть обмена для протокола Диффи-Хеллмана;
  - Сервер отсылает сообщение **ServerHelloDone**, идентифицирующее окончание первого раунда установления соединения;
  - Клиент отвечает сообщением **ClientKeyExchange**, которое содержит клиентскую часть протокола Диффи-Хеллмана или зашифрованный открытым ключом из сертификата сервера секрет (**PreMasterSecret**);
  - Клиент и сервер, используя ключ **PreMasterSecret** и случайно сгенерированные числа, вычисляют общий секрет. Вся остальная информация о сеансовом ключе будет получена из общего секрета;
- 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;
- 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.

ServerKeyExchange

- Это сообщение, содержащее серверную часть данных, необходимых для генерации общего сеансового ключа.
- Обычно это параметры протокола Диффи-Хеллмана (DH).
- Если используется классический вариант, то в сообщении ServerKeyExchange передаётся значение модуля и вычисленный сервером открытый ключ DH.
- В варианте на эллиптических кривых (ECDH) - идентификатор самой кривой и, аналогично DH, открытый ключ. Параметры подписываются сервером (за исключением экзотических вариантов обмена, вроде анонимного DH), клиент может проверить подпись, используя открытый ключ сервера из SSL-сертификата.
- В зависимости от используемой криптосистемы, подпись может быть DSA (сейчас практически не встречается), RSA или ECDSA.
- Подпись на параметрах DH очень важна, так как позволяет защитить соединение от атаки типа "Человек посередине". В TLS 1.3 - от значения хеш-функции, заданной в используемом шифронаборе, вычисленного для объединения ServerRandom, ClientRandom и параметров обмена DH.

ClientKeyExchange

- Содержание этого сообщения зависит от того, какой шифронабор выбран.
- Есть два основных типа - RSA и несколько вариантов протокола Диффи-Хеллмана.
- В случае использования RSA, клиент генерирует 48-байтовый случайный секрет, шифрует его открытым ключом сервера (этот ключ передаётся в составе серверного SSL-сертификата или в сообщении ServerKeyExchange) и передаёт зашифрованные данные на сервер.
- Сервер может расшифровать значение, используя соответствующий секретный ключ
- если секретный серверный ключ станет известен третьей стороне, то она сможет расшифровать ранее записанный TLS-трафик. Тем не менее, обмен сеансовым ключом при помощи RSA продолжает использоваться в некоторых реализациях TLS.
- Современный метод - использование протокола Диффи-Хеллмана. В этом случае, ClientKeyExchange содержит открытый ключ DH. Этот ключ генерируется клиентом либо в соответствии с параметрами, переданными сервером в ServerKeyExchange, либо в соответствии с параметрами, указанными в серверном SSL-сертификате, если последний поддерживает DH.
- Случай с передачей параметров классического DH в составе сертификата - сейчас является экзотическим, таких сертификатов в "дикой природе" не встречается. Напомню, что серверные параметры DH (или ECDH) подписываются серверным ключом, клиент проверяет подпись, используя открытый ключ сервера.

Master Secret generation

- Основа ключей - общий секрет Master Secret - генерируется из нескольких переменных составляющих: так называемый Premaster Secret, ClientRandom и ServerRandom. Premaster Secret - согласуется в рамках обмена ключами, это либо последовательность случайных байтов, зашифрованная открытым ключом сервера, либо значение, полученное в результате обмена по протоколу Диффи-Хеллмана. Master Secret - это массив из 48 байтов, получаемый в результате применения клиентом и сервером к этим составляющим псевдослучайной функции, определённой спецификацией. Псевдослучайная функция (PRF) TLS 1.2 построена на базе хеш-функции SHA-256 (либо может быть использована более "мощная" хеш-функция, указанная в составе шифронабора), предыдущие версии используют конструкцию на базе сочетания MD5 (которая давно не считается криптографически стойкой) и SHA-1 (которую перестали считать достаточно стойкой в 2015 году). При этом, способ использования MD5 при генерации сеансового ключа не приводит к возникновению уязвимостей, связанных с недостаточным качеством данной функции. То же самое можно сказать и про SHA-1. RFC 5246 определяет Master Secret для TLS 1.2 следующим образом:
- master_secret = PRF(pre_master_secret, "master secret", ClientHello.random + ServerHello.random)[0..47];

Other key exchange methods

- 1. **PSK - pre-shared key.** Схема основана на использовании общего секретного ключа и симметричной криптосистемы, при условии, что ключ был согласован заранее;
- 2. **временный ключ RSA** - исторический метод, предполагавший создание сеансового ключа RSA. Сейчас данный метод не используется, однако его поддержка послужила основой для атаки FREAK, опубликованной в 2014 году;
- 3. **SRP** - протокол SRP (Secure Remote Password), RFC 5054. Протокол, позволяющий сгенерировать общий симметричный секретный ключ достаточной стойкости на основе известного клиенту и серверу пароля, без раскрытия этого пароля через незащищённый канал;
- 4. **Анонимный DH** - анонимный вариант протокола Диффи-Хеллмана, в котором не используется подпись на серверных параметрах. Такая схема подвержена атаке типа "человек посередине", практически не встречается;
- 5. **DH с сертификатом** - вариант DH, в котором параметры (модуль, генератор и открытый ключ сервера) определены в серверном сертификате. Этот метод является историческим, требует специального сертификата и на практике не встречается. Основное его отличие от используемых сейчас вариантов (нередко называемых также "эфемерным Диффи-Хеллманом") состоит в том, что серверный открытый ключ протокола Диффи-Хеллмана оказывается зафиксирован: именно он входит в сертификат и подписывается удостоверяющим центром. Это означает, что схема не обладает прогрессивной секретностью. Сервер всякий раз использует один и тот же секретный ключ Диффи-Хеллмана, а раскрытие этого ключа позволяет расшифровать ранее записанный трафик TLS.

- Протокол DH на "обычной" группе работает следующим образом. Для того чтобы получить общий секретный ключ, стороны сначала выбирают общие параметры Диффи-Хеллмана — это модуль, задающий группу (простое число), а также некоторый элемент этой группы, называемый генератором - соответственно: P и G. Параметры DH открыты и считаются известными третьей стороне. На следующем шаге каждая из сторон выбирает собственное секретное число a (и, соответственно, b) и вычисляет значение $A = G^a \bmod P$ (соответственно, $B = G^b \bmod P$). То есть, все операции проводятся по модулю P, что, собственно, и отображает их результаты в элементы группы. Далее стороны обмениваются по открытому каналу значениями A, B и вычисляют $A^b \bmod P$ и $B^a \bmod P$, соответственно. Полученные значения равны, так как, из свойства степеней, $A^b = (G^a)^b = B^a = (G^b)^a = G^{ab}$. Таким образом, стороны получили общий секретный ключ G^{ab}. Если вернуться к описанной выше основной идее протокола, то операция возведения в степень (по модулю) выступает в роли однонаправленной функции, значения которой передаются по открытому каналу. Свойство групповой операции (в данном случае, эквивалентное более привычному свойству степеней), позволяет сторонам, осуществляющим обмен значениями, прийти к одинаковому результату (G^{ab}). В случае TLS, сервер передаёт в сторону клиента параметры Диффи-Хеллмана и свой открытый ключ (A), удостоверяя эти значения собственной электронной подписью (либо RSA, либо ECDSA). Подпись вычисляется от ключа, открытая часть которого указана в сертификате сервера (см. выше).

Сложность

- Прослушивающая канал сторона знает значения P, G, A и B. Но для того, чтобы определить значение секретного ключа, необходимо вычислить a или b, решив относительно x уравнение вида $A = G^x \bmod P$. Стоящая за решением этого уравнения задача и называется задачей дискретного логарифмирования в конечной группе. Эта задача вычислительно сложна для групп большой разрядности (1024 бита и выше), поэтому возведение в степень оказывается однонаправленным. (Фундаментальная причина сложности данной операции сходна с общими проблемами деления в группах: грубо говоря, деление представляет собой операцию *поиска* среди элементов группы такого, который при умножении на известный делитель давал бы делимое. Ситуация эквивалентна привычному делению из курса начальной школы: чтобы разделить 15 на 3 нужно *найти* такое число, которое при умножении на 3 давало бы 15. Если известны некоторые свойства группы, позволяющие построить её арифметическую структуру, то операцию деления можно оптимизировать, используя тот или иной "быстрый" пошаговый алгоритм, отбрасывающий из перебора заведомо неподходящие элементы.)