Занятие 4. Хеширование, кодирование, шифрование. Протоколы SSL/TLS. Протокол HTTPS

Хеширование, кодирование и шифрование используются во многих протоколах, средах передачи данных, иногда совершенно прозрачно для пользователя. Эти понятия кажутся очень похожими, в том числе потому что все они служат для преобразования исходных данных в другое представление, однако каждое из них используется с определенной целью и имеет свои особенности.

ХЕШИРОВАНИЕ

Это преобразование по определенному алгоритму входного массива данных произвольной длины в выходную битовую строку фиксированной длины. Такие преобразования также называются хеш-функциями, функциями свертки.

Некоторыми из применений хеширования являются:

- Построение ассоциативных массивов
- Поиск дубликатов данных
- Создание уникальных идентификаторов данных
- Вычисление контрольных сумм данных и сигналов для вычисления ошибок передачи в них
- Хранение паролей пользователей
- и др

В общем случае нет однозначного соответствия между хешем (результатом хеширования) и исходными данными. Но поскольку значения хешей менее разнообразны в силу ограничений на длину, чем значения входных данных, то случаются ситуации, при которых хеширование преобразует более чем один массив входных данных в одинаковые хеши, это называется коллизией. Вероятность возникновения коллизий используется для оценки качества хеш-функций (функция создания хеша из входных данных). Хорошая хеш-функция должна быстро вычисляться и минимизировать количество коллизий, то есть максимально уменьшить вероятность нахождения способов генерации входных данных, хешируемых одинаковым значением.

Одними из самых быстрых, но и самых простых видов хеш-функций являются контрольные суммы. Чаще всего они используются в транспортных протоколах для гарантии целостности полученных данных, такие функции имеют достаточно много коллизий, однако обладают очень высокой скоростью вычисления.

MD5 (Message Digest Algorithm) - один из первых стандартов алгоритма, который применялся в целях проверки целостности файлов. Также с его помощью хранили пароли в базах данных web-приложений. Функциональность относительно проста: алгоритм выводит для каждого ввода данных фиксированную 128-битную строку. Его особенность - это простота операций и короткая выходная длина, в результате чего MD5 является относительно легким для взлома.

Secure Hashing Algorithm (SHA1) - это алгоритм, созданный Агентством национальной безопасности (NSA). Он создает 160-битные хеши фиксированной длины. На деле SHA1 лишь улучшил MD5 и увеличил длину вывода, а также увеличил число однонаправленных операций и их сложность. Со временем появилась альтернатива — SHA2, а потом и SHA3 (также известен под названием *Keccak*), в отличие от MD5, SHA1, SHA2 этот алгоритм обладает другой внутренней структурой, а так же имеет множество настраиваемых параметров, например, возможные размеры хешей: 224, 256, 384 и 512 бит.

КОДИРОВАНИЕ

Это преобразование входных данных в определенную последовательность бит или символов для удобства дальнейшей работы. Например, *BASE64* - кодирование бинарных данных при помощи только 64 символов ASCII, *UUE (uuencode)* - кодирование бинарных данных в текстовую форму для передачи через текстовые протоколы - HTTP, SMTP, FTP, кодирование данных для передачи по физическому каналу или записи на цифровой носитель. Даже простое преобразование сообщений в световые или любые другие сигналы - это кодирование (например, азбука Морзе).

ШИФРОВАНИЕ

Это преобразование входные данных, делающее их нечитаемыми для посторонних лиц, при этом доверенные лица могут провести дешифрование и прочитать исходные данные. Существует множество алгоритмов шифрования, однако секретность данных основана не на тайном алгоритме, а на том, что ключ шифрования известен только доверенным лицам. С помощью шифрования одновременно обеспечивается конфиденциальность информации (информация скрыта от третьих лиц), целостность (предотвращение изменений в процессе передачи или хранения информации), идентифицируемость (шифрование позволяет определить и аутентифицировать источник информации).

В зависимости от использования ключа или ключей для работы с исходной и зашифрованной информацией существует два метода шифрования:

- Симметричное шифрование (используется один и тот же ключ и для зашифровывания, и для расшифровывания)
- Асимметричное шифрование (используется два разных ключа: один для зашифровывания - открытый ключ, другой для расшифровывания - закрытый ключ)

В *симметричном методе* для шифрования и расшифровывания используется один и тот же ключ. Сами алгоритм шифрования и ключ выбираются заранее и известны обоим сторонам обмена информацией. В таком методе передача и сохранения ключа шифрования является важной задачей для установления и поддержания защищенного канала связи.

Схема реализации симметричного метода шифрования:

- Генерация ключа и передача ключа второй стороне
- Шифрование и передача сообщения
- Получение и расшифровка сообщения

Недостатками симметричного шифрования являются проблема передачи ключа второй стороне и невозможность установить/удостоверить авторство информации.

В *асимметричном методе* (также называется методом с открытым ключом) используются два ключа - открытый и закрытый, связанные определенным математическим алгоритмом друг с другом. Открытый ключ передается по незащищенному каналу, он используется для шифрования и проверки/удостоверения авторства информации, закрытый ключ используется для расшифровки информации.

Схема реализации асимметричного метода шифрования:

- Генерация ключевой пары, выбор алгоритма, передача открытого ключа по незащищенному каналу
- Шифрование информации при помощи открытого ключа и передача зашифрованного сообщения
- Расшифровка сообщения при помощи закрытого ключа

Если необходимо наладить канал связи в обе стороны, то первую операцию необходимо проделать на обеих сторонах, таким образом, каждый будет знать свои закрытый, открытый ключи и открытый ключ собеседника. Закрытый ключ каждой стороны не передается по незащищенному каналу, тем самым оставаясь в секретности.

Для подробного изучения алгоритмов шифрования и дешифрования и генерации ключей изучите дополнительно материал: https://2hourscrypto.info/

ПРОТОКОЛЫ SSL/TLS

Для того чтобы обеспечить защищенное взаимодействие клиента и сервера и предотвратить возможное прослушивание трафика злоумышленниками, были разработаны протоколы, работающие между транспортным и прикладным уровнями - SSL и его «последователь» TLS.

ПРОТОКОЛ SSL

SSL изначально разработан компанией Netscape Communications для добавления протокола HTTPS в свой веб-браузер Netscape Navigator. Впоследствии на основании протокола SSL 3.0 был разработан и принят стандарт, получивший имя TLS. Протокол SSL обеспечивает защищённый обмен данными за счёт аутентификации (как сервера, так и возможно клиента) и шифрования. SSL использует асимметричную криптографию для аутентификации ключей обмена, симметричный шифр для сохранения конфиденциальности, коды аутентификации сообщений для целостности сообщений. Первая версия протокола SSL 1.0 свет так и не увидела, версия SSL 2.0 была выпущена в 1995 году и имела очень много недостатков по безопасности и привела к разработке и выпуску в 1996 году версии SSL 3.0.

Принцип работы протокола SSL имеет многослойную структуру, благодаря которой может быть достигнут определенный уровень безопасности обмена информацией.

Первый слой протокола SSL - слой протокола подтверждения подключения (Handshake) содержит в себе несколько подпротоколов:

- Протокол подтверждения подключения (*Handshake Protocol*), он используется для согласования данных подключения и сессии между обеими сторонами, передачу ключей, включая параметры алгоритма шифрования и алгоритма хеширования, аутентификацию сторон при помощи сертификатов, алгоритм сжатия данных и т.д.
- Протокол изменения параметров шифра (*Cipher Spec Protocol*), он используется для уведомления одной из сторон о необходимости изменения набора ключей шифрования.
- Предупредительный протокол (*Alert Protocol*), он используется для уведомления какой-либо из сторон о произошедшей ошибке, например, закрытии соединения или невозможности расшифровать полученное сообщение и т.д.

Для проверки принадлежности открытых ключей владельцам, а так же для аутентификации сторон используются цифровые сертификаты, такие сертификаты могут быть выданы удостоверяющим центром, которому по умолчанию доверяют и клиент и сервер, сертификат может быть самоподписанным, то есть выданным сервером самому себе, или пустым. Удостоверяющий центр (CA, Certification Authority) - сторона или организация, чья честность неоспорима, а открытый ключ широко известен. Задача центра сертификации - подтверждать подлинность ключей шифрования с помощью сертификатов электронной подписи. Самоподписанные и пустые сертификаты могут использоваться только в тестовом окружении, даже в случае использования цифровых сертификатов только в ограниченной локальной сети предпочтительно использование локального удостоверяющего центра и соответствующей настройки клиентов.

Второй слой протокола SSL - слой протокола записи, он отвечает непосредственно за шифрование и расшифровывание информации. На вход протокола слоя записи поступает

информация в открытом или зашифрованном виде, протокол разбивает ее на блоки (как правило по 16 байт) и шифрует или расшифровывает их. В зависимости от используемого алгоритма блоки могут шифроваться по-отдельности, а могут XOR-иться с предыдущим результатом, чтобы усложнить анализ зашифрованного сообщения.

Существует ряд атак, которые могут быть предприняты против протокола SSL. Сейчас SSL 2.0 по умолчанию отключена во многих современных браузерах. В октябре 2014 года была выявлена уязвимость протокола SSL 3.0, позволяющая злоумышленнику осуществить атаку man-in-the-middle, а так же SSL 3.0 содержит еще ряд других проблем.

ПРОТОКОЛ TLS

TLS основан на спецификации протокола SSL 3.0, он широко используется в приложениях, работающих по сети, особенно в сети Интернет. Протокол TLS так же служит для обеспечения защищенной передачи информации.

Так как большинство протоколов может быть использовано как с TLS так и без него, то для установки защищенного соединения необходимо явно указать серверу об этом. Это может быть достигнуто обращением клиента по специальному номеру порта, по которому соединение всегда устанавливается с помощью TLS (например, 443 порт для HTTPS), либо по произвольному номеру порта и специальной команды серверу на переключение соединения на TLS (например, команда STARTTLS для клиентов и серверов электронной почты).

Кратко, алгоритм установки защищенного соединения между клиентом и сервером по протоколу TLS выглядит следующим образом:

- Клиент подключается к серверу и запрашивает защищенное подключение
- Клиент предоставляет серверу список поддерживаемых алгоритмов шифрования и хеширования
- Сервер выбирает наиболее защищенные алгоритмы из списка клиента и, если они поддерживаются сервером, сообщает клиенту о выбранных алгоритмах для установки соединения
- Сервер отправляет клиенту свой цифровой сертификат для собственной аутентификации, обычно он содержит в себе информацию о сервере, удостоверяющем центре и открытый ключ сервера
- Клиент до начала передачи информации проверяет валидность полученного серверного сертификата
- Осуществляется передача сеансового ключа, который используется для симметричного шифрования информации в текущей сессии. Передача сеансового ключа может быть осуществлена при помощи асимметричного шифрования RSA, используя открытый ключ сервера (этот метод не рекомендован), либо при помощи протокола Диффи-Хеллмана.

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

TLS имеет множество мер безопасности: защита от понижения версии протокола или перехода к менее надежному алгоритму шифрования, подтверждение подлинности переданных ранее сообщений (например, при установке параметров сессии), обеспечение целостности сообщений в процессе аутентификации и т.п. Однако в протоколе TLS 1.0 были обнаружены и использованы уязвимости, позволяющие как дешифровать передаваемые сообщения, так и организовывать атаку типа man-in-the-middle и добавлять злоумышленнику в сообщения клиента собственные запросы. Также существуют варианты атак, основанные непосредственно на программной реализации протокола, а не на его алгоритме.

В 2011 году стала запрещена реализация обратной совместимости со старыми версиями SSL по всех протоколах TLS. Новейшим обновлением (на сегодняшний день) стал TLS 1.3, описаный в 2018 году. Все протоколы SSL признаны устаревшими, хотя еще поддерживаются некоторыми сайтами в сети Интернет. TLS 1.0 и TLS 1.1 уже считаются устаревшими, хотя TLS 1.1 поддерживается многими сайтами и системами, TLS 1.2 сейчас считается протоколом по умолчанию для многих ресурсов, некоторые уже поддерживают работу только по протоколу TLS 1.3.

ПРОТОКОЛ HTTPS

HTTPS (HyperText Transfer Protocol Secure) - это расширение протокола HTTP для поддержки шифрования, данные в HTTPS передаются поверх протокола TLS (SSL признан устаревшим, так что в большинстве случаев не поддерживается), соединения по протоколу HTTPS по умолчанию принимаются на 443/TCP порту.

Для работы вэб-сервера по протоколу HTTPS необходим процесс подготовки и установки на стороне сервера его сертификата открытого ключа, а так же закрытого ключа сервера. Сертификат открытого ключа удостоверяет и аутентифицирует сервер при установке защищенного соединения, сам открытый ключ и сертификат открытого ключа посылаются клиенту, а для расшифровки сообщений от клиента используется закрытый ключ. Так же можно проводить аутентификацию клиентов на вэб-сервере, но для этого понадобится генерация сертификатов для каждого пользователя и установка их в браузеры.

Поскольку HTTPS работает поверх SSL/TLS (в модели OSI протоколы SSL/TLS принято относить к 6 уровню), то установка защищенного соединения происходит по алгоритмам выбранного протокола шифрования и прозрачно для самого протокола HTTP, однако требует гораздо больших ресурсов как от сервера, так и от клиента.

Несмотря на использование в протоколе HTTPS шифрования всех передаваемых между клиентом и сервером данных с применением максимально защищенных из поддерживаемых клиентом и сервером алгоритмов существует несколько видов уязвимости протокола:

- Совместное использование HTTP и HTTPS на сайтах может приводить к подмене злоумышленником части загружаемого контента (например, передать клиенту javascript, содержащий вредоносный код) или получить часть данных о клиенте или сессии клиента. Для защиты от подобных уязвимостей был разработан механизм HSTS (HTTP Strict Transport Security), принудительно активирующий использование соединения HTTPS вместо HTTP.

- Уязвимости к атакам с использованием анализа трафика, которые выводят свойства защищенных данных путем измерения размера трафика и времени передачи сообщений в нем. Это возможно, поскольку шифрование оказывает минимальное влияние на размер и время прохождения трафика

- Уязвимости к атакам посредника (man-in-the-middle) особенно распространены при атаке на пользователей, часто имеющий дело с самоподписанными сертификатами и пренебрегают проверкой сертификата, когда браузер отправляет предупреждение. В этот момент злоумышленник может встроиться между клиентом и сервером, установить соединение с сервером, а клиенту отправить свой сертификат и ключ, таким образом будет установлено два соединения: между клиентом и злоумышленником, между злоумышленником и сервером. Злоумышленник будет расшифровывать все сообщения клиента и пересылать их по защищенному каналу серверу и обратно.