Разработка тестового набора для верификации реализаций протокола безопасности TLS

А.В. Никешин, Н.В. Пакулин, В.З. Шнитман

Аннотация. Статья посвящена разработке тестового набора для проверки соответствий реализаций узлов Интернет спецификациям протокола безопасности TLS [1-11]. Для построения тестового набора использовалась технология автоматического тестирования UniTESK [12] и программный пакет JavaTESK [13], реализующий эту технологию.

Работа выполнялась в Институте системного программирования РАН в рамках проекта «Верификация реализаций расширяемых протоколов Интернета» при поддержке гранта РФФИ № 10-07-00145. В ходе ее выполнения были выделены требования к реализациям TLS, разработаны формальные спецификации и прототип тестового набора для верификации реализаций TLS. В статье кратко описаны метод формализации требований TLS, тестовый набор, а также результаты тестирования существующих реализаций сервера TLS. Эти результаты показывают, что предложенный в данной работе метод верификации позволяет эффективно автоматизировать тестирование таких сложных протоколов, как протоколы безопасности.

Ключевые слова: тестирование, верификация, формальные методы, формальные спецификации, MBT, тестирование с использованием моделей, модели программ, TLS, SSL, JavaTESK, UniTESK.

1. Введение

Протокол TLS обеспечивает защиту передаваемых данных между приложениями, взаимодействующими по схеме клиент-сервер. Он располагается поверх протокола транспортного уровня, инкапсулируя протоколы прикладного уровня (такие как HTTP, FTP, SMTP, NNTP, XMPP, LDAP). Первоначально TLS использовался с надежными транспортными протоколами, такими как TCP. Позже появилась реализация для протоколов, ориентированных на передачу дейтаграмм (таких как UDP, DCCP), выделенная в независимый стандарт DTLS (Datagram Transport Layer Security).

В настоящее время TLS широко используется в сочетании с самыми разными протоколами:

– для защиты WWW-трафика (протокол HTTPS),

– с почтовыми протоколами SMTP, IMAP, POP3,
– при организации виртуальных защищенных сетей (OpenVPN),
– с протоколом запуска сессий (SIP: Session Initiation Protocol) и основанными на нем приложениями (например VoIP),
– с протоколом LDAP в качестве средства шифрования сеансов LDAP и защиты от спуфинга.

Задачи, решаемые протоколом, в порядке приоритетности:

1. Криптографическая безопасность: TLS используется для установления безопасного соединения между двумя участниками.
2. Совместимость: взаимодействие приложений по протоколу TLS, не зависит от внутренних особенностей реализаций.
3. Расширяемость: возможность расширения функциональности, в том числе добавления новых алгоритмов шифрования и обеспечения целостности данных.
4. Относительная эффективность: криптографические операции, особенно операции с открытым ключом, требуют значительных затрат вычислительных ресурсов. В TLS имеется механизм кэширования сессий, который позволяет уменьшить количество соединений, и, как следствие, усилить безопасность за счет уменьшения сетевой активности.

Задачу тестирования соответствия можно условно разделить на две подзадачи: построение тестовых воздействий и оценка правильности наблюдаемых результатов. К первой задаче примыкает проблема оценки полноты покрытия – чем шире будет спектр тестовых воздействий, тем шире получится охват функций протокола при тестировании. Вторая задача заключается в вынесении вердикта о соответствии тестируемой системы спецификации протокола.

Разработанный метод верификации [14] основан на автоматизированном тестировании соответствия формальным спецификациям. Требования, представленные в тексте стандарта, изложены на английском языке и представляют собой неформальный текст, описывающий желаемое поведение системы на естественном языке. Для того, чтобы автоматизировано извлечь тесты для протокола, необходимо перевести его спецификацию в вид, пригодный для решений этой задачи. В разработанном подходе в этой роли выступают формальные спецификации, в которых требования задаются как логические выражения, записанные посредством математического формализма.

В основе подхода лежит представление протоколов как асинхронных автоматов, причем автомат задаётся неявно посредством контрактных спецификаций. А именно, формальное описание поведения протокола задаётся как контрактная спецификация: набор сообщений протокола рассматривается как некоторый формальный интерфейс между реализацией

протокола и её окружением, поведение протокола описывается посредством пред- и постусловий. Такое задание протоколов позволяет описывать поведение сложных недетерминированных протоколов, таких как протоколы защиты передачи данных в сетях IP. Эти протоколы отличаются сложными структурами данных сообщений и состояния, недетерминированным поведением и неполными спецификациями – в спецификациях умышленно оставлены пробелы для облегчения реализации протоколов. Контрактные спецификации позволяют представлять требования к сложным протоколам в форме, пригодной для автоматизированного тестирования реализаций таких протоколов. Для протоколов, формальная спецификация которых задана в виде контрактной спецификации, разработан метод автоматизированного построения тестовых последовательностей с полностью автоматическим вынесением вердиктов о соответствии наблюдаемого поведения реализации её спецификации. В совокупности разработанные методы позволяют автоматизировать тестирование соответствия реализаций сложных сетевых протоколов их спецификациям.

В рамках предложенного метода тест представляет собой конечный автомат. С каждым переходом автомата сопоставлено определённое тестовое воздействие. При выполнении перехода это воздействие подаётся на тестируемую реализацию, регистрируются реакции реализации и автоматически выносится вердикт о соответствии наблюдаемого поведения спецификации. Обход автомата теста совершается автоматически во время тестирования, алгоритм обхода не зависит от протокола, тестируемой реализации или конкретного теста. Последовательность переходов определяет тестовую последовательность. В силу недетерминизма протоколов и различий в поддержке необязательных функций в реализациях конкретные тестовые последовательности, получаемые при прогоне тестов на разных реализациях, могут не совпадать друг с другом.

Предложенный подход к верификации функций безопасности включает два метода: метод формализации стандартов протоколов и метод формального задания тестовых наборов [15]. Метод формализации стандартов протоколов включает анализ спецификации протокола и извлечение требований, определение формального интерфейса протокола, формализацию функциональных требований к реализации протокола, задание критериев покрытия и разработку функции реконструкции состояния. Метод формализации тестовых последовательностей состоит из определения целей тестирования, разработки проекта автомата теста для конкретной цели тестирования, задания переходов автомата теста, задания функции определения состояния автомата теста по модельному состоянию контрактной спецификации протокола, проектирования настроечной информации автомата теста и разработки формата для представления опций, а также включает прогон тестового сценария и анализ результатов тестирования. Для описания автоматов тестов в данном методе используется специальный вид задания автоматов тестов, который называется тестовым сценарием.

2. Обзор протокола TLS

Протокол TLS разработан на основе спецификации протокола SSL 3.0 (Secure Socket Layer), опубликованной корпорацией Netscape. Ранние версии протокола описаны в RFC 2246 (версия 1.0) [1] и RFC 4346 (версия 1.1) [4]. Последняя модификация TLS версии 1.2 определена в RFC 5246 [10]. Как указано в спецификации, не смотря на то, что различия между протоколами не значительны, TLS и SSL 3.0 несовместимы. Хотя в TLS предусмотрен механизм, позволяющий реализациям TLS общаться с реализациями SSL 3.0.

Протокол TLS состоит из двух уровней: протокола Записей TLS (TLS Record Protocol) и протокола Рукопожатия (TLS Handshake Protocol). На нижнем уровне находится протокол Записей TLS (TLS Record Protocol), работающий поверх некоторого надежного транспортного протокола (например, TCP). Этот протокол обеспечивает конфиденциальность и надежность соединений:

1. Конфиденциальность соединения. Для защиты данных используются алгоритмы симметричного шифрования, ключи для которых уникальны для каждой сессии и создаются на основе секрета, согласованного другими протоколами (например, протоколом Рукопожатия). Протокол Записей TLS также может использоваться без шифрования.
2. Надежность соединения. Целостность сообщений обеспечивается путем вычисления и включения в сообщения кода аутентификации (MAC). Для этого используются криптографические алгоритмы на основе хэш-функций (такие как SHA-1, MD5 и др.). Данный протокол может использоваться без вычисления MAC.

Протокол Записей TLS используется для инкапсуляции протоколов более высокого уровня. Одним из таких протоколов является протокол Рукопожатия, который позволяет серверу и клиенту аутентифицировать друг друга и согласовать параметры безопасности сессии (такие как криптографические алгоритмы и ключи), перед началом использования протокола прикладного уровня. Для аутентификации используются криптографические алгоритмы с открытым ключом (такие как RSA, DSS и др.). Протокол Рукопожатия также может использоваться без аутентификации. Однако обычно она необходима, по крайней мере, для одного из участников.

Одно из преимуществ TLS состоит в том, что он независим от протоколов прикладного уровня. Для протоколов более высокого уровня использование TLS является прозрачным, однако спецификация TLS не определяет схему их взаимодействия. Решение о том, как инициировать TLS-диалог и как интерпретировать сертификаты аутентификации, оставляется на усмотрение разработчиков протоколов, которые работают поверх TLS.

2.1. Протокол Записей TLS

Получив данные для передачи, протокол Записей TLS фрагментирует их на блоки нужной длины, если необходимо сжимает данные, вычисляет MAC, шифрует и передает результат транспортному протоколу. Полученные данные дешифруются, проверяется их целостность, выполняется декомпрессия и дефрагментация, и результат передается протоколу верхнего уровня.

По протоколу TLS происходит обмен блоками данных, называемыми Записи TLS, инкапсулирующими данные протокола верхнего уровня. Каждая Запись TLS содержит поле “Content type”, которое определяет тип протокола верхнего уровня.

Спецификация TLS определяет четыре протокола, работающие поверх TLS: протокол Рукопожатия (Handshake protocol, content type 22), протокол Оповещения (Alert protocol, content type 21), протокол Изменения состояния (Change cipher spec protocol, content type 20) и протокол прикладных данных(content type 23). Другие протоколы могут поддерживаться TLS, однако они должны быть зарегистрированы в соответствующем реестре IANA (Internet Assigned Numbers Authority) ([10], раздел 12).

2.2. Состояние соединения

Состояние соединения TLS определяет набор параметров для работы протокола Записей TLS. К этим параметрам относятся криптографические алгоритмы сжатия, шифрования и вычисления MAC, а также соответствующие ключи. Существуют четыре состояния соединения: текущие состояния чтения и записи и ожидаемые состояния чтения и записи. Для любой обработки данных используются установки текущего состояния. Параметры ожидаемых состояний устанавливаются через протокол Рукопожатия. Протокол Изменения состояния (Change Cipher Spec) заменяет текущее состояние ожидаемым, а ожидаемое состояние инициализируется пустым состоянием. Начальное текущее состояние всегда определяется без использования шифрования, сжатия и вычисления MAC.

2.3. Протокол Рукопожатия

Протокол рукопожатия используется для согласования параметров безопасности соединения.

Полная схема обмена выглядит следующим образом:

```
  Client                                   Server

  ClientHello            -------->
                                           ServerHello
                                           Certificate*

ServerKeyExchange*
```



```
CertificateRequest*
                         <--------         ServerHelloDone
  Certificate*
  ClientKeyExchange
  CertificateVerify*
  [ChangeCipherSpec]
  Finished               -------->

[ChangeCipherSpec]
                         <--------         Finished
  Application Data       <------->         Application Data
```

* необязательные или зависящие от ситуации сообщения, которые посылаются не всегда.

- Клиент посылает сообщение ClientHello, содержащее максимальный номер версии протокола TLS, который он поддерживает, случайное число ClientHello.random, предлагаемые криптографические алгоритмы (Cipher Suite) и метод сжатия (Compression Method).
- Сервер отвечает сообщением ServerHello, содержащим выбранные их предложенных клиентом версию TLS, идентификатор сессии (Session ID), криптографические алгоритмы (Cipher Suite) и метод сжатия (Compression Method),а также свое случайное число ServerHello.random. Выбранная версия TLS должна быть максимальной из поддерживаемых. Сервер также может ответить критической ошибкой и разорвать соединение.

Для обмена ключами используются четыре сообщения: сертификат сервера (server Certificate), ServerKeyExchange, сертификат клиента (client Certificate) и ClientKeyExchange.

- После Hello сервер посылает свой сертификат для аутентификации (если он есть). Дополнительно может быть послано сообщение ServerKeyExchange (если сервер не имеет сертификата или его сертификат служит только для цифровой подписи). Если сервер аутентифицирован, он может запросить сертификат клиента (CertificateRequest).
- Сервер посылает сообщение ServerHelloDone, указывающее, что фаза приветствия завершена. После чего ждет ответа клиента.
- Если был соответствующий запрос сервера, клиент посылает свой сертификат. Затем посылается сообщение ClientKeyExchange, содержимое которого зависит от выбранного в сообщениях ClientHello и ServerHello алгоритма с открытым ключом. Если сертификат клиента используется для цифровой подписи, то посылается подписанное сообщение CertificateVerify для явной проверки подлинности сертификата.
- Клиент посылает сообщение об изменении состояния ChangeCipherSpec, и заменяет текущее состояние ожидаемым.

- В завершение клиент посылает сообщение Finished, защищенное только что согласованными алгоритмами и содержащее код аутентификации (MAC), вычисленный над всеми предыдущими сообщениями обмена.
- Сервер расшифровывает полученное сообщение Finished, проверяет правильность MAC. В случае ошибки соединение разрывается.
- Наконец сервер отвечает своим сообщением изменения состояния ChangeCipherSpec, заменяет текущее состояние ожидаемым и посылает заключительное сообщение Finished, применяя согласованные алгоритмы и ключи.
- Клиент также дешифрует и проверяет последнее сообщение

На этом обмен протокола Рукопожатие считается завершенным, и клиент и сервер могут начать обмен данными прикладного уровня. Сообщения Finished являются первыми сообщениями, защищенными только что согласованными алгоритмами и ключами.

Если клиент и сервер хотят возобновить предыдущую сессию или дублировать существующую (вместо согласования новых параметров безопасности), используется сокращенная схема обмена:

```
  Client                                Server

  ClientHello           -------->
                                        ServerHello

[ChangeCipherSpec]
                        <--------       Finished
  [ChangeCipherSpec]
  Finished              -------->
  Application Data      <------->       Application Data
```

Клиент посылает ClientHello, используя идентификатор сессии (Session ID), которую требуется возобновить. Сервер ищет соответствующий идентификатор сессии в своем кэше сессий. Если идентификатор найден, и политика сервера разрешает возобновить сессию, то сервер посылает ServerHello с этим идентификатором сессии. После этого сразу происходит обмен сообщениями изменения состояния ChangeCipherSpec и завершающими сообщениями Finished. Обмен Рукопожатия считается завершенным, и можно отправлять данные прикладного уровня.

Если соответствующий идентификатор Session ID не найден, сервер создает новый ID, и выполняется полная схема Рукопожатия.

2.4. Протокол Изменения состояния

Протокол состоит из единственного сообщения, которое зашифровано и сжато, в соответствии с параметрами текущего состояния соединения. Сообщение об изменении состояния может посылаться как клиентом, так и

сервером для уведомления партнера о том, что следующие сообщения будут защищены только что согласованными алгоритмами и ключами. При этом получатель данного сообщения должен немедленно заменить текущее состояние чтения ожидаемым состоянием чтения, а отправитель - текущее состояние записи ожидаемым состоянием записи.

2.5. Протокол Оповещения

Сообщение содержит код оповещения (предупреждающее или критическое) и его описание. Критическое сообщение приводит к немедленному закрытию соединения. При этом другие соединения данной сессии могут быть продолжены, но новые соединения в рамках данной сессии создавать запрещено. Сообщения оповещения зашифрованы и сжаты, как определено в текущем состоянии соединения.

2.6. Механизмы расширения протокола TLS

Протокол TLS допускает расширения для добавления новой функциональности. Для согласования расширений используются сообщения ClientHello и ServerHello протокола Рукопожатия. Каждое расширение содержит тип расширения (поле “extension type”) и данные, формат которых зависит от конкретного расширения. Список расширений управляется IANA и доступен по адресу <http://www.iana.org/assignments/tls-extensiontype-values>. Механизм расширений обеспечивает обратную совместимость, т.е. реализации, поддерживающие расширения могут взаимодействовать с теми, которые их не поддерживают. Если конкретного типа расширения не было в запросе ClientHello, он не должен присутствовать в ответном сообщении ServerHello.

Первые расширения (как и сами механизмы расширений) описаны в спецификации RFC 4366. Там же определены два новых сообщения протокола Рукопожатия: CertificateURL и CertificateStatus. Дальнейшие расширения были определены в RFC 4680, 4681, 5077 и RFC 5246. Ниже перечислены некоторые конкретные расширения TLS:

- Указание имени сервера. Тип расширения: server_name. Значение: 0 (RFC 4366).
- Согласование максимального размера фрагментов. Тип расширения: max_fragment_length. Значение: 1 (RFC 4366).
- Универсальные локаторы ресурсов сертификатов клиента. Тип расширения: client_certificate_url. Значение: 2 (RFC 4366).
- Указание доверенного центра сертификации. Тип расширения: trusted_ca_keys. Значение: 3 (RFC 4366).
- Усеченный HMAC. Тип расширения: truncated_hmac. Значение: 4 (RFC 4366).
- Запрос статуса сертификата. Тип расширения: status_request. Значение: 5 (RFC 4366).

– Отображение пользователей. Тип расширения: user_mapping. Значение: 6 (RFC 4681).
– Алгоритмы цифровой подписи. Тип расширения: signature_algorithms. Значение: 13 (RFC 5246).
– Возобновление сессии без сохранения состояния на сервере. Тип расширения: SessionTicket TLS. Значение: 35 (RFC 5077).
– Указание повторного согласования. Тип расширения: renegotiation_info. Значение: 65281 (RFC 5746).

3. Формальная спецификация TLS

Подробное описание модели тестирования приведено в [16,17]. В соответствии с этой моделью тестирования формальная спецификация TLS состоит из нескольких компонентов:

- модельного состояния, которое содержит набор структур данных, моделирующих концептуальные структуры данных из стандартов TLS;
- формального интерфейса TLS, включающего спецификационные стимулы, формализующие требования к изменению состояния реализации TLS при внешнем воздействии на систему и спецификационные реакции, которые формализуют требования к реакциям реализации TLS на внешние воздействия;
- критериев покрытия, идентифицирующих различные ветви функциональности TLS.

3.1. Модельное состояние

Модельное состояние представлено множеством TLS-соединений и множеством TLS-сессий на узле, моделирующем состояние целевой реализации.

TLS-соединение содержит параметры безопасности конкретного соединения, такие как криптографические алгоритмы, ключи. TLS-сессия содержит данные, необходимые для повторного использования согласованных ранее параметров безопасности, такие как сертификаты, криптографические алгоритмы, мастер-ключ (master secret).

Полное описание этих структур приведено в RFC 5246.

Рассмотрим, как эти концептуальные структуры данных представлены в формальной спецификации.

Модельный тип TLS-соединения включает следующие блоки данных:

selector	селекторы трафика (адреса и порты TCP соединения), являющиеся идентификатором соединения

current read state	текущее состояние чтения
current write state	текущее состояние записи
pending read state	ожидаемое состояние чтения
pending write state	ожидаемое состояние записи

Для каждого соединения TLS определяются четыре состояния: текущие состояния чтения и записи и ожидаемые (pending) состояния чтения и записи. Все сообщения обрабатываются, используя параметры текущего состояния. Параметры ожидаемых состояний устанавливаются с помощью обмена рукопожатия (TLS Handshake Protocol).

Модельный тип TLS состояния содержит следующие структуры данных:

sessionID	идентификатор сессии, соответствующей данному соединению
keys	ключи криптографических алгоритмов
sequence number	порядковый номер сообщений
security parameters	параметры безопасности

Модельный тип параметров безопасности содержит следующие поля:

connectionEnd	данный флаг определяет, является узел сервером или клиентом для данного соединения
prf_algorithm	алгоритм, используемый для создания криптографических ключей из мастер-ключа
bulk_cipher_algorithm	алгоритм шифрования и необходимые параметры
enc_key_length	
block_length	
fixed_iv_length	
record_iv_length	
mac_algorithm	алгоритм защиты целостности сообщений и необходимые параметры
mac_length	
mac_key_length	
compression_algorithm	алгоритм сжатия

master_secret	мастер-ключ
client_random	одноразовые массивы байт клиента и сервера
server_random	

Модельный тип TLS-сессии включает:

id	идентификатор сессии
peer_certificate	сертификат партнера, если такой используется
compression_method	метод сжатия
cipher_spec	криптографические алгоритмы
master_secret	мастер-ключ
isResumable	данный флаг определяет, может ли сессия использоваться для инициализации новых соединений

3.2. Модель TLS-сообщений

Протокол TLS для передачи данных использует структуры, называемые TLS-записями (TLS Records), инкапсулирующие весь TLS-трафик. Спецификация определяет четыре типа передаваемых данных:

- Обмен Рукопожатия (TLS Handshake Protocol), который используется для согласования новых параметров безопасности;
- Протокол изменения состояния (Change Cipher Spec Protocol), единственное сообщение ChangeCipherSpec заменяет параметры текущего состояния параметрами соответствующего ожидаемого состояния;
- Протокол Оповещения (Alert Protocol), предназначенный для передачи информационных сообщений и сообщений об ошибках;
- Данные протокола верхнего уровня, использующего TLS в качестве транспорта.

Каждое TLS-сообщение может содержать данные только одного из перечисленных выше типов, однако структур данных этого типа может быть несколько (например, TLS-сообщение может содержать несколько сообщений протокола Рукопожатия). Тип данных указывается в заголовке TLS-записи.

Для модельного представления TLS-сообщений разработана библиотека соответствующих спецификационных типов, позволяющая моделировать различные варианты сообщений.

3.3. Спецификационные функции

В спецификации каждое входящее TLS-сообщение рассматривается как последовательность стимулов. Каждый стимул в этой последовательности соответствует обработке отдельного блока данных в TLS-сообщении (напомним, что TLS-сообщение может содержать несколько структур данных конкретного типа).

Предусловие каждого стимула проверяет допустимость отправки сообщения, правильность значений полей блоков данных и прогнозирует ожидаемую реакцию целевой системы.

Каждое исходящее TLS-сообщение рассматривается как последовательность реакций. Каждая реакция в этой последовательности соответствует отдельному блоку данных в TLS-сообщении.

Постусловия реакций проверяют допустимость получения конкретного сообщения, и правильность значений полей блоков данных.

4. Тестирование реализаций TLS

4.1. Каталог функциональных требований к реализации протокола TLS

Спецификация протокола TLS написана на естественном языке. Для тестирования реализации на соответствие необходимо выделить из стандарта отдельные требования и затем их формализовать. В результате анализа текста стандарта был составлен полный список требований (около 300 требований). Эти требования разбиты на несколько групп:

1. Требования, относящиеся к протоколу записей TLS (TLS Record Protocol);
2. Требования, относящиеся к протоколам рукопожатия (TLS Handshaking Protocols);
3. Требования, относящиеся к правилам формирования ключевого материала.

4.2. Устройство тестового стенда и тестовый набор

В состав тестового стенда входят инструментальный узел и целевой узел. На инструментальном узле исполняется основной поток управления тестовой системы. На целевом узле функционирует тестируемая реализация. Инструментальный и целевой узлы могут располагаться в разных сегментах сети.

Стимулами в разработанном тестовом наборе являются сообщения от инструментального узла, а реакциями - сообщения со стороны тестируемого узла. Основная часть требований спецификации TLS проверяется в постусловиях реакций.

4.3. Тестирование TLS-сервера

В роли TLS-сервера реализация не генерирует запросы, а лишь поддерживает информационный обмен, инициированный другим узлом. Стимулами являются сообщения от инструментального узла.

В тестовом сценарии создается TLS-соединение, в рамках которого формируются запросы в модельном представлении, передаваемые затем функции отправки сообщений. В предусловии спецификационных функций стимулов проверяется правильность структуры тестового сообщения и его своевременность, и на основании этого делается вывод о том, должен ли на него быть ответ, сообщение об ошибке или реализация должна его проигнорировать. Из модельного представления тестового сообщения строится реализационное, которое и отправляется в сеть.

Сборщик реакций в течение заданного времени собирает ответные сообщения целевой системы. Из реализационных TLS сообщений строятся их модельные представления. Последовательность блоков данных в полученных сообщениях рассматривается как последовательность реакций целевой системы.

В постусловии реакций данные проверяются на соответствие требованиям спецификации. Проверка разделена на несколько стадий. Сначала проверяется допустимость такого сообщения от реализации и его своевременность, затем - структура самого сообщения (присутствующие поля и их значения должны соответствовать текущему обмену).

После проверки всех требований, результат передается тестовому сценарию, где в зависимости от плана сценария, принимается решение о продолжении или завершении информационного обмена. В случае выявления нарушения требований принимается решение о критичности ошибки и возможности отправки следующих запросов.

В случае успешного завершения обмена рукопожатия, создается новая TLS-сессия (за исключением сокращенного варианта обмена, в котором используется уже существующая сессия), параметры которой могут в дальнейшем использоваться для инициализации других TLS соединений.

4.4. Выбор реализации для тестирования

Для тестирования на соответствие стандарту были выбраны следующие реализации:

1. Почтовый сервер Postfix.2.9.3 с открытой реализацией протокола TLS openssl.1.0.1c под управлением операционной системы Red Hat Enterprise Linux 5.5;
2. Реализация TLS в виртуальной машине Java 1.7.0_05 (Java Secure Socket Extension);
3. Тестовый сервер TLS интернет ресурса https://www.mikestoolbox.net.

4.5. Результаты тестирования реализаций TLS в роли сервера

При выполнении тестового набора был выявлен ряд особенностей, нарушений требований RFC 5246 и шибок реализаций. Ниже дано описание этих особенностей.

1. Почтовый сервер Postfix.2.9.3:

- при получении сообщения оповещения (Alert) CLOSE_NOTIFY сервер разрывает соединение без отправки ответного уведомления;
- если вслед за сообщением ChangeCipherSpec клиент отправляет любое оповещение (Alert), сервер отвечает ошибкой UNEXPECTED_MESSAGE (неожиданное сообщение);
- если в сообщении ClientHello присутствуют дубликаты неизвестных серверу расширений, сервер их игнорирует и продолжает обмен данными;
- в сообщении ClientHello при наличии расширений присутствуют несколько полей длины: длина сообщения, общая длина расширений, длина каждого расширения. Сервер не проверяет поле общей длины расширений;
- если в TLS сообщении после блока ClientHello имеются какие-то неправильные данные (например неизвестный заголовок), сервер сначала отвечает на ClientHello последовательностью сообщений ServerHello, ServerCertificate, ServerHelloDone и только затем отправляет сообщение об ошибке и разрывает соединение;
- согласно спецификации размер каждого TLS-сообщения не должен превышать (2^14 + 2048) байт. Реализация принимает и отвечает на такие сообщения;
- если, после завершения обмена рукопожатия, не разрывая соединения, отправить сообщение ClientHello, сервер ответит на него не критичным (т.е. без разрыва соединения) сообщением Alert NO_RENEGOTIATION. После этого на любые сообщения протокола Рукопожатия и сообщения Alert (в том числе критичные, которые должны приводить к закрытию соединения) сервер отвечает оповещением NO_RENEGOTIATION, при этом продолжая принимать, обрабатывать и отвечать на сообщения с прикладными данными (APPLICATION data). Если клиент отправляет в прикладном сообщении команду “quit” (команда завершения сеанса связи с почтовым сервером протокола SMTP), сервер корректно отвечает на нее и отправляет завершающее сообщение Alert CLOSE_NOTIFY. Однако на ответное сообщение Alert CLOSE_NOTIFY (как предусмотрено спецификацией) сервер также отвечает сообщением Alert NO_RENEGOTIATION.

2. Реализация TLS в виртуальной машине Java 1.7.0_05:

- реализация игнорирует дубликаты расширений в сообщении ClientHello;
- реализация не проверяет поле длины во входящих сообщениях протокола Рукопожатия (по крайней мере в сообщениях ClientHello, ClientKeyExchange, Finished);
- в сообщении ClientHello при наличии расширений присутствуют несколько полей длины: длина сообщения, общая длина расширений, длина каждого расширения. Сервер не проверяет поле общей длины расширений.

3. Тестовый сервер TLS интернет ресурса https://www.mikestoolbox.net:

- если в обмене рукопожатия используется алгоритм RSA, клиент создает и отправляет серверу 48-байтный предварительный ключ (premaster secret), два первых байта которого содержат версию протокола TLS из предшествующего сообщения ClientHello. Сервер должен проверить правильность номера версии. Данная реализация не выполняет эту проверку. Данное требование предназначено для противодействия атакам rollback и является критичным.
- после установления соединения, в ответ на сообщения протокола Рукопожатия отличные от ClientHello, сервер возвращает сообщение об ошибке, при этом версия протокола в TLS сообщении установлена 3.0.
- если в сообщении ClientHello значение поле CipherSuite (криптографический набор) равно TLS_NULL_WITH_NULL_NULL, или в сообщении присутствуют дубликаты расширений, сервер, как и требуется, возвращает ошибку, однако TLS-запись содержит два одинаковых сообщения об ошибке.

Тестирование реализаций на соответствие требованиям спецификации RFC 5746 (TLS Renegotiation Indication Extension)

Данный документ рекомендует отказаться от стандартной схемы переустановки TLS соединений (RFC 5246) и использовать только предложенную безопасную схему.

1. Почтовый сервер Postfix.2.9.3:

- поддерживает только безопасную переустановку TLS соединений.

2. Реализация TLS в виртуальной машине Java 1.7.0_05:

- с настройками по умолчанию поддерживает только безопасную переустановку TLS соединений, дополнительные настройки позволяют использовать стандартный вариант переустановки.

3. Тестовый сервер TLS интернет ресурса https://www.mikestoolbox.net:

- поддерживает оба варианта переустановки TLS соединений.

Следует отметить, что, несмотря на некоторые особенности и нарушения, реализации в целом соответствуют спецификации. Однако вторая реализация нарушает критичное требование проверки версии протокола TLS в сообщении ClientKeyExchange с использованием алгоритма RSA.

5. Заключение

В ходе выполнения работы были выделены требования к реализациям TLS, разработаны формальные спецификации обработки входящих и исходящих сообщений протокола TLS, медиаторы для доставки тестовых воздействий на целевую реализацию, а также тестовые сценарии для проверки обработки входящих и исходящих сообщений подсистемой TLS. В статье описаны также результаты тестирования существующих реализаций.

Данная работа показала, что разработанный метод верификации, основанный на контрактных спецификациях, позволяет эффективно автоматизировать тестирование таких сложных протоколов, как протоколы безопасности. При этом тестовые наборы обладают формально определенным и прослеживаемым покрытием требований, что в значительной степени улучшает качество тестирования.

Список литературы

[1] IETF RFC 2246. Dierks, T. and C. Allen, "The TLS Protocol Version 1.0", January 1999.
[2] IETF RFC 3268. Chown, P., "Advanced Encryption Standard (AES) Ciphersuites for Transport Layer Security (TLS)", June 2002.
[3] IETF RFC 3546. Blake-Wilson, S., Nystrom, M., Hopwood, D., Mikkelsen, J., and T. Wright, "Transport Layer Security (TLS) Extensions", June 2003.
[4] IETF RFC 4346. Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.1", April 2006.
[5] IETF RFC 4366. Blake-Wilson, S., Nystrom, M., Hopwood, D., Mikkelsen, J., and T. Wright, "Transport Layer Security (TLS) Extensions", April 2006.
[6] IETF RFC 4507. Salowey, J., Zhou, H., Eronen, P., and H. Tschofenig, "Transport Layer Security (TLS) Session Resumption without Server-Side State", May 2006.
[7] IETF RFC 4680. Santesson, S., "TLS Handshake Message for Supplemental Data", October 2006.
[8] IETF RFC 4681. Santesson, S., Medvinsky, A., and J. Ball, "TLS User Mapping Extension", October 2006.
[9] IETF RFC 5077. Salowey, J., Zhou, H., Eronen, P., and H. Tschofenig, "Transport Layer Security (TLS) Session Resumption without Server-Side State", January 2008.
[10] IETF RFC 5246. Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", August 2008.

[11] IETF RFC 5746. E. Rescorla, M. Ray, S. Dispensa, N. Oskov. Transport Layer Security (TLS) Renegotiation Indication Extension. February 2010.
[12] Bourdonov I., Kossatchev A., Kuliamin V., Petrenko A. UniTesK Test Suite Architecture // Proceedings of FME, LNCS 2391. Springer-Verlag, 2002. P. 77-88.
[13] Bourdonov I.B., Demakov A.V., Jarov A.A., Kossatchev A.S., Kuliamin V.V., Petrenko A.K. and Zelenov S.V. Java Specification Extension for Automated Test Development // Proceedings of PSI'2001. Novosibirsk, Russia July 2-6 2001, LNCS 2244:301-307. Springer-Verlag, 2001.
[14] Н.В. Пакулин. Формализация стандартов и тестовых наборов протоколов Интернета. Автореферат диссертации на соискание учёной степени кандидата физико-математических наук. Москва, 2006.
[15] Н.В. Пакулин, А.В. Хорошилов "Разработка формальных моделей и тестирование соответствия для систем с асинхронными интерфейсами и телекоммуникационных протоколов", Журнал "Программирование" № 5, 2007 г., ISSN 0132-3474, с. 1-29.
[16] А.В. Никешин, Н.В. Пакулин, В.З. Шнитман "Разработка тестового набора для верификации реализаций протокола безопасности IPsec v2", Труды Института системного программирования РАН, т. 18, 2010, стр. 151-182.
[17] А.В. Никешин, Н.В. Пакулин, В.З. Шнитман "Верификация функций безопасности протокола IPsec v2", Журнал "Программирование" № 1, 2011, стр. 36-56.

Creation of a test suite for verification of the TLS security protocol

Nikeshin A.V., Pakulin N.V., Shnitman V.Z.
alexn@ispras.ru, npak@ispras.ru, vzs@ispras.ru
ISP RAS, Moscow, Russia

Abstract. Despite the fact that TLS and its predecessor SSL are in use for more than 15 years, there are no accepted public conformance test suite for those protocols. Implementers do have their own test suites, but the primary objective of those tests are internals of the corresponding implementation rather than conformance against the standard. Furthermore, such tests are too much implementation specific to be used to be used for implementation other than they were developed for. In general it is impossible to test a TLS/SSL implementation with tests from another implementation. The paper presents an approach to conformance testing of TLS/SSL server implementations. The approach is based on Model-Based Testing methodology. A test suite was developed and propped against a number of open TLS/SSL server implementations. The developed approach to conformance testing is based on automated testing against formal specifications. Requirements in the text specification are statements in a natural language, describing desired behavior of TLS implementations. Following the approach used we elicited more than 300 functional requirements to server implementations and formalized in the specification extension of Java language. The language used is part of UniTESK family of MBT technologies. Using the extension, the protocol model is presented as a contract specification with pre- and postcondition defining constraints on the allowed behavior. The specification is backed with a technique for on-the-fly test construction provided by UniTESK tools. We tried the test suite against three open implementations of TLS. Testing revealed discrepancies with the standard in all three implementations, and one implementation suffers from violation of a critical requirement. The work proved that the proposed approach to verification, based on protocol modeling with contract specifications, provides efficient automation of protocols as complex as security protocols. Furthermore, thanks to formal models, the tests have well-defined and trackable coverage criteria that greatly enhances quality of testing.

Keywords: testing, verification, formal methods, formal specification, model-based testing, MBT, software models, TLS, SSL, JavaTESK, UniTESK.

References

[1] IETF RFC 2246. Dierks, T. and C. Allen, "The TLS Protocol Ve rsion 1.0", January 1999.

[2] IETF RFC 3268. Chown, P., "Advanced Encryption Standard (AES) Ciphersuites for Transport Layer Security (TLS)", June 2002.

[3] IETF RFC 3546. Blake-Wilson, S., Nystrom, M., Hopwood, D., Mikkelsen, J., and T. Wright, "Transport Layer Security (TLS) Extensions", June 2003.

[4] IETF RFC 4346. Dierks, T. and E. Rescorla , "The Transport Layer Security (TLS) Protocol Version 1.1", April 2006.

[5] IETF RFC 4366. Blake-Wilson, S., Nystrom, M., Hopwood, D., Mikkelsen, J., and T. Wright, "Transport Layer Security (TLS) Extensions", April 2006.

[6] IETF RFC 4507. Salowey, J., Zhou, H., Eronen, P., and H. Tschofenig, "Transport Layer Security (TLS) Session Resumption without Server-Side State", May 2006.

[7] IETF RFC 4680. Santesson, S. , "TLS Handshake Message fo r Supplemental Data", October 2006.

[8] IETF RFC 4681. Santesson, S., Medvinsky , A., and J. Ball, "TLS User Mapping Extension", October 2006.

[9] IETF RFC 5077. Salowey, J., Zhou, H., Eronen, P., and H. Tschofenig, "Transport Layer Security (TLS) Session Resumption w ithout Server-Side State", January 2008.

[10] IETF RFC 5246. Dierks, T. and E. Rescorla , "The Transport Layer Security (TLS) Protocol Version 1.2", August 2008. 403

[11] IETF RFC 5746. E. Rescorla, M. Ray, S. Dispensa, N. Oskov. Transport Layer Security (TLS) Renegotiation Indication Extension. February 2010.

[12] Bourdonov I., Kossatchev A., Kuliamin V. , Petrenko A. UniTesK Test Suite Architecture // Proceedings of FME, LN CS 2391. Springer-Verlag, 2002. P. 77-88.

[13] Bourdonov I.B., Demakov A.V., Ja rov A.A., Kossatc hev A.S., Kuliami n V.V., Petrenko A.K. and Zelenov S.V. Java Specification Extension for Automated Test Development // Proceedings of PSI'2001. Novosibirsk, Russia July 2-6 2001, LNCS 2244:301-307. Springer-Verlag, 2001.

[14] N.V. Pakulin. Formalizacija standartov i testovyh naborov protokolov Interneta [Formalization of standards and test suites for Internet protocols]. Avtoreferat dissertacii na soiskanie uchjonoj stepeni kandidata fiziko-matematicheskih nauk [PhD thesis]. Moskva, 2006

[15] N.Pakulin, A. Khoroshilov. Development of formal models and conformance testing for systems with asynchronous interfaces and telecommunications protocols. Programming and Computer Software 33 (6), 316-335

[16] A.V. Nikeshin, N.V. Pakulin, V.Z. Shnitman Razrabotka testovogo nabora dlja verifikacii realizacij protokola bezopasnosti IPsec v2 [Test suite development for verification of IPsec v2 protocol implementations], Trudy ISP RAN [The Proceedings of ISP RAS], t. 18, 2010, str. 151-182

[17] A. Nikeshin, N. Pakulin, V. Shnitman. Verification of security functions of IPsec v2 protocol. Programming and Computer Software. 37 (1), 26-40