УДК 004.05

ОПЫТ ВЕРИФИКАЦИИ РЕАЛИЗАЦИЙ ПРОТОКОЛА TLS 1.3

А. В. Никешин[1] **[0000-0001-5781-9736]**, **В. З. Шнитман**[2] **[0000-0002-1509-0972]**

[1, 2]Институт системного программирования им. В.П. Иванникова РАН,
ул. А. Солженицына, 25, г. Москва,109004

[1]alexn@ispras.ru, [2]vzs@ispras.ru

Аннотация

Представлен опыт верификации реализаций сервера криптографического протокола TLS версии 1.3. TLS – широко распространенный криптографический протокол, предназначенный для создания защищенных каналов передачи данных и обеспечивающий необходимую для этого функциональность: конфиденциальность передаваемых данных, целостность данных, аутентификацию сторон. Новая версия протокола TLS 1.3 была представлена в августе 2018 года и имеет ряд существенных отличий по сравнению с предыдущей версией 1.2. Ряд разработчиков протокола TLS уже включил поддержку последней версии в свои реализации. Данные обстоятельства делают актуальным проведение исследований в области верификации и безопасности реализаций новой версии протокола TLS. В работе использован новый тестовый набор для верификации реализаций протокола TLS 1.3 на соответствие спецификациям интернета, разработанный на основе спецификации RFC 8446 с использованием технологии UniTESK и методов мутационного тестирования. Текущая работа является частью проекта верификации протокола TLS 1.3 и охватывает часть дополнительной функциональности и необязательных расширений протокола.

Для тестирования реализаций на соответствие формальным спецификациям применена технология UniTESK, предоставляющая средства автоматизации тестирования на основе использования конечных автоматов. Состояния тестируемой системы задают состояния автомата, а тестовые воздействия – переходы этого автомата. При выполнении перехода заданное воздействие передается на тестируемую реализацию, после чего регистрируются реакции реализации и автоматически выносится вердикт о соответствии наблюдаемого поведения специ-

фикации. Мутационные методы тестирования используются для обнаружения нестандартного поведения тестируемой системы с помощью передачи некорректных данных. В поток обмена протокола, создаваемый в соответствии со спецификацией, вносятся некоторые изменения: либо изменяются значения полей сообщений, сформированных на основе разработанной модели протокола, либо изменяется порядок сообщений в потоке обмена. Модель протокола позволяет вносить изменения в поток данных на любом этапе сетевого обмена, что позволяет тестовому сценарию проходить через все значимые состояния протокола и в каждом таком состоянии проводить тестирование реализации в соответствии с заданной программой. На данный момент было обнаружено несколько отклонений реализаций от спецификации.

Представленный подход доказал свою эффективность в нескольких наших проектах при тестировании сетевых протоколов, обеспечив обнаружение различных отклонений от спецификации и других ошибок.

Ключевые слова: *безопасность, TSL, TSLv1.3, протоколы, тестирование, оценка устойчивости, Интернет, стандарты, формальные методы спецификации.*

ВВЕДЕНИЕ

TLS является сегодня одним из наиболее востребованных криптографических протоколов, предназначенных для создания защищенных каналов передачи данных и обеспечивающий необходимую для этого функциональность: конфиденциальность передаваемых данных, целостность данных, аутентификацию сторон. В процессе использования протокола неизбежно выявляются его недостатки, появляются новые угрозы безопасности, что приводит к дальнейшему развитию протокола. В августе 2018 года была представлена спецификация новой версии протокола TLS 1.3, в которой, по сравнению с предыдущей версией 1.2, была существенно переработана архитектура [1, 2].

Все больше разработчиков протокола TLS включает поддержку последней версии в свои реализации. Данные обстоятельства делают актуальным проведение исследований в области верификации и безопасности реализаций новой вер-

сии протокола TLS. В процессе тестирования сетевых протоколов решается несколько важных задач: проверяется функциональная совместимость различных реализаций, проверяется соответствие реализации требованиям спецификации и устойчивость реализации к нестандартным воздействиям.

В данной работе использованы наработанные нами методики по тестированию сетевых протоколов: автоматизированное тестирование на соответствие формальным спецификациям и методы мутации данных, доказавшие свою эффективность в наших предыдущих проектах при тестировании сетевых протоколов, обеспечив обнаружение различных отклонений от спецификации и других ошибок.

1. ОСНОВНАЯ ФУНКЦИОНАЛЬНОСТЬ ПРОТОКОЛА TLS ВЕРСИИ 1.3

Архитектура TLS делится на два уровня обработки сообщений:

– Протокол рукопожатия (Handshakeprotocol) является основной частью архитектуры TLS. Он выполняет аутентификацию взаимодействующих сторон, согласовывает версию протокола, криптографические режимы и параметры безопасности, устанавливает общий ключевой материал. Встроенные механизмы защиты противодействуют вмешательству третьей стороны и понижению уровня безопасности согласованного криптографического набора.

– Протокол записей (Recordprotocol) используется для инкапсуляции протоколов более высокого уровня, в том числе протокола рукопожатия. Он использует параметры, установленные протоколом рукопожатия для защиты данных, передаваемых между партнерами. Весь трафик делится на последовательность записей, каждая из которых независимо защищается с помощью согласованных криптографических ключей. Протокол записей, работает поверх надежного транспортного протокола (как правило, TCP).

Неудачное рукопожатие или другая ошибка протокола вызывают завершение соединения, которому может предшествовать сообщение уведомления.

TLS не зависит от протоколов прикладного уровня, для которых он является прозрачным, спецификация TLS не определяет схему их взаимодействия. Решение о том, как инициировать TLS-диалог и интерпретировать результаты его работы, оставляется на усмотрение разработчиков протоколов верхнего уровня.

Ниже представлено основное полное рукопожатие TLSv1.3:

C→S: ClientHello (Key Exchange phase)
+ key_share*
+ signature_algorithms*
+ psk_key_exchange_modes*
+ pre_shared_key*

C←S: ServerHello
+ key_share*
+ pre_shared_key*
{EncryptedExtensions} (Server Parameters phase)
{CertificateRequest*}
{Certificate*} (Authentication phase)
{CertificateVerify*}
{Finished}
[Application Data*]
C→S:
{Certificate*}
{CertificateVerify*}
{Finished}

[ApplicationData] <-------> [ApplicationData]

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

{} Обозначает сообщения, защищенные с помощью сеансовых ключей для обмена рукопожатия.

[] Обозначает сообщения, защищенные с помощью сеансовых ключей для прикладных данных.

Назначение сообщений и расширений подробно описано в спецификации.

Сообщения протокола рукопожатия сгруппированы в три блока:

– Обменключами (Key Exchange): Сообщения Client Hello, Server Hello. Устанавливает ключевой материал и выбирает криптографические параметры.

– Параметры сервера (Server Parameters): Сообщения EncryptedExtensions, CertificateRequest. Устанавливает дополнительные параметры рукопожатия (например, запрос аутентификации клиента, поддержка протокола прикладного уровня и др.).

– Аутентификация (Authentication): Сообщения Certificate, Certificate Verify, Finished. Аутентифицирует сервер (и, при необходимости, клиента), подтверждает созданные сеансовые ключи и обеспечивает целостность всего обмена рукопожатия.

2. ОСНОВНЫЕ ОСОБЕННОСТИ НОВОЙ ВЕРСИИ ПРОТОКОЛА TLS

Все сообщения, отправляемые после фазы обмена ключами (т. е. после сообщения Server Hello), зашифрованы соответствующими ключами. Это одна из особенностей данной версии протокола TLS – шифровать как можно больше передаваемых данных.

Сообщение Encrypted Extensions содержит расширения, ответные для соответствующих расширений Client Hello, которые не требуются для создания криптографических параметров. Такие расширения ранее отправлялись в открытом виде.

После формирования сообщения Server Hello сервер получает возможность вычислить все необходимые сеансовые ключи. Это позволяет начать отправку прикладных данных до приема сообщений аутентификации клиента. Однако необходимо понимать, что в этой точке любые данные посылаются не аутентифицированному клиенту.

Набор сообщений протокола рукопожатия был переработан, чтобы стать более согласованным и ликвидировать излишние сообщения, такие как Change Cipher Spec (за исключением случаев обеспечения обратной совместимости с промежуточными сетевыми устройствами (middlebox)).

Механизм согласования версий перемещен в расширения, что должно улучшить совместимость с существующими серверами, которые часто неправильно реализуют согласование версий. Также изменен механизм защиты от понижения номера версии.

Изменена концепция набора шифров, чтобы разделить механизмы аутентификации и обмена ключами от алгоритмов защиты данных. Список поддерживаемых алгоритмов переработан и избавлен от всех устаревших и ненадежных алгоритмов. Оставшиеся алгоритмы шифрования являются алгоритмами аутентифицированного шифрования со связанными данными (Authenticated Encryption with Associated Data, AEAD).

Функции вычисления ключей были переделаны. Новая конструкция упрощает специалистам анализ криптографических механизмов благодаря улучшенным свойствам разделения ключей.

Переработан механизм возобновления сеанса. Теперь для этого используется общий ключ, согласованный в предыдущем обмене рукопожатия.

Добавлен новый механизм передачи данных 0-RTT (zeroround-triptime), позволяющий несколько сократить количество обменов данными между партнерами.

Также стоит отметить, что в более ранних спецификациях TLS определенные части было трудно понять, что приводило к возникновению проблем с функциональной совместимостью реализаций и безопасностью. В данной спецификации более четко проработаны вопросы поведения реализаций в различных ситуациях.

3. РАСШИРЕННАЯ ФУНКЦИОНАЛЬНОСТЬ ПРОТОКОЛАTLS1.3

Кроме базового полного рукопожатия, TLS как расширяемый протокол предоставляет дополнительные, необязательные сервисы безопасности, согласуемые, как правило, через механизм расширений. Ниже рассмотрена часть таких сервисов, указанных в спецификации TLS.

Режим PSK

TLS1.3 позволяет использовать сокращенный режим рукопожатия с использованием заранее распределенных ключей **(**pre-sharedkey, PSK) [2]. Данный режим использует меньше сообщений для обмена и меньше трудоемких вычислений с ключами и может быть полезен в небольших системах, в которых проще сконфигурировать PSK на каждом узле, чем разворачивать инфраструктуру для

работы с сертификатами. Также стороны могут иметь свой механизм установления общего секретного ключа.

C→S: ClientHello
+ key_share*
+ psk_key_exchange_modes
+ pre_shared_key
C←S: ServerHello
+ pre_shared_key
+ key_share*
{EncryptedExtensions}
{Finished}
[Application Data*]
C→S:
{Finished}

[Application Data] <-------> [Application Data]

Режим PSK может использоваться в двух вариантах, определяемых расширением psk_key_exchange_modes: PSK и PSK-(EC)DHE. Последний дополняет процедуру формирования сеансовых ключей из PSK алгоритмом Диффи–Хеллмана (используется расширение key_share), что увеличивает как надежность итоговых ключей (обеспечивая прогрессирующую секретность, forwardsecrecy), так и сложность дополнительных криптографических вычислений.

Возобновление сеанса

Рассматриваемая версия протокола, в отличие от предыдущей версии, не выделяет отдельно режим возобновления сеанса. Вместо этого предлагается механизм согласования общего секрета PSK. Для этого используются так называемые удостоверения (tickets) и сообщение New Session Ticket, которое отправляется сервером клиенту в любое время после завершения сеанса рукопожатия и может использоваться в следующих соединениях. Сообщений New Session Ticket может быть несколько, каждое содержит одно удостоверение (ticket), которое с

одной стороны используется для формирования нового ключа PSK, с другой – является уникальным идентификатором этого ключа. Также при создании PSK используется соответствующий секрет из предыдущего сеанса рукопожатия, связывая ключевые материалы разных сеансов и обеспечивая дополнительную безопасность соединений.

Данные 0-RTT

На основе режима рукопожатия PSK реализован новый для протокола TLS способ отправки так называемых «ранних данных» (earlydata). При наличии общего ключа PSK, TLS 1.3 позволяет клиенту отправить некоторые данные во время первого рейса. Клиент использует PSK одновременно для аутентификации сервера и шифрования ранних данных.

```
C→S:    ClientHello
         + early_data
         + key_share*
         + psk_key_exchange_modes
         + pre_shared_key
        (Application Data*)
C←S:    ServerHello
         + pre_shared_key
         + key_share*
        {EncryptedExtensions}
         + early_data*
        {Finished}
        [Application Data*]
C→S:
(EndOfEarlyData)
        {Finished}

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

Таким образом, данные 0-RTT просто добавляются к рукопожатию1-RTT в первом рейсе в сообщение Client Hello.

Как отмечено в спецификации, свойства безопасности для данных 0-RTT слабее, чем свойства безопасности для других данных TLS:

– Эти данные не обладают свойством прогрессирующей секретности, поскольку зашифрованы ключами, полученными с использованием только предлагаемого ключа PSK.

– Нет защиты от повторного воспроизведения между разными соединениями, поскольку данные 0-RTT не зависят от нового Server Hello (использующего случайное значение, Random). При этом внутри одного соединения данные 0-RTT дублироваться не могут, поскольку данные 0-RTT и 1-RTT защищаются разными ключами.

Обновление сеансовых ключей

Для повышения безопасности передачи данных рекомендуется периодически обновлять ключевой материал. Во время обмена рукопожатия создается базовый ключ соединения (mastersecret), из которого получаются первоначальные сеансовые ключи. Сообщение Key Update сообщает партнеру, что отправитель создал новые сеансовые ключи, и последующие сообщения зашифрованы новыми ключами.

Расширение Cookie

Если в сообщении Client Hello недостаточно данных для продолжения рукопожатия, сервер может отправить сообщение Hello Retry Request, требующее прислать исправленное Client Hello с указанным ключевым материалом. При этом сервер может включить в Hello Retry Request необязательное расширение cookie [2]. Клиент должен скопировать это расширение в своем повторном Client Hello.

Как указано в спецификации, cookie преследует две основные цели:

- Заставляет клиента продемонстрировать достижимость на сетевом адресе (обеспечивая защиту от DoS-атак). Это прежде всего полезно для транспорта, не ориентированного на соединения.
- Позволяет серверу не сохранять состояние сеанса. Поскольку первое сообщение Client Hello (точнее, его хеш) участвует в криптографических вычислениях, сервер должен где-то его хранить. Вместо этого

сервер может выгрузить эти данные клиенту, включив их в расширение cookie, а клиент вернет его в ответном сообщении.

Расширение Signaturealgorithmscert

Чтобы указать, какие алгоритмы подписи могут использоваться в цифровой подписи, TLS 1.3 предоставляет два расширения [2]. Расширение “signature_algorithms_cert” применяется к подписям в сертификатах, а расширение “signature_algorithms” – к подписям в сообщениях Certificate Verify. В предыдущей версии TLS 1.2 используется только расширение “signature_algorithms”. Расширение “signature_algorithms_cert” было добавлено в новой версии протокола, чтобы позволить реализациям, поддерживающим разные наборы алгоритмов для сертификатов и для самого TLS, явно просигнализировать о своих возможностях. Ключи сертификатов должны соответствовать алгоритмам подписи, с которыми они используются. Если расширение “signature_algorithms_cert” отсутствует, то предполагается, что реализации используют одну и ту же политику в обоих случаях, и расширение “signature_algorithms” применяется также к подписям в сертификатах (что соответствует поведению TLS 1.2).

Расширение Servername

Могут возникнуть ситуации, когда по одному сетевому адресу размещено несколько «виртуальных» серверов, для каждого из которых используется свой сертификат. Данное расширение позволяет клиенту указать имя сервера, с которым он связывается. Для сервера получение этого расширения носит рекомендательный характер. Сервер может использовать информацию, содержащуюся в расширении, для выбора нужного сертификата, учитывая также и другие настройки политики безопасности. В этом случае серверу следует включить это расширение в соответствующее ответное сообщение. Хотя данное расширение представлено в отдельной более ранней спецификации, его поддержка и использование являются обязательными для реализаций TLS 1.3 [3].

РасширениеMax fragment length / Recordsizelimit

Если клиент хочет использовать сообщения меньшего размера, чем предусмотрено спецификацией (2^14 байт), то он может воспользоваться расширением “max_fragment_length" [3]. Однако у него есть ряд существенных недостатков:

Данное расширение предлагает к применению всего несколько фиксированных значений, при этом максимальное значение (2^12 байт) сильно меньше предлагаемого спецификацией TLS (2^14 байт).

Добавление новых значений не предусмотрено.

Сервер не может запросить более низкое значение, чем то, которое предложил клиент. Это серьезная проблема, если сервер более ограничен.

Расширение плохо подходит для случаев, когда возможности клиента и сервера неодинаковы, т. е. если, например, конечная точка хочет отправлять записи большего размера, чем те, которые она получает.

Чтобы обойти эти ограничения, было предложено другое расширение для TLS – “recordsizelimit” [4]. Оно позволяет задать произвольное максимальное значение сообщений (не превышающее значение, определенное согласованной партнерами версией протокола) для каждого направления передачи данных.

Расширение Posthandshakeauth

Расширение “post_handshake_auth” добавлено в TLS 1.3 и используется для указания того, что клиент хочет выполнить аутентификацию после рукопожатия [2]. В этом случае сервер может запросить аутентификацию клиента в любой момент времени после завершения рукопожатия путем посылки сообщения CertificateRequest. Клиент должен ответить соответствующими сообщениями аутентификации.

C←S: [CertificateRequest]

C→S:

[Certificate]

[CertificateVerify*]

[Finished]

Если у клиента нет соответствующих сертификатов, то отправляется пустое сообщение Certificate, а сообщение Certificate Verify не используется.

Сервер может отправить несколько сообщений Certificate Request, как в разное время, так и последовательно (например, если требуется доступ к нескольким сервисам). При этом ответы могут приходить в произвольной последовательности (для разделения запросов используются соответствующие уникальные идентификаторы).

Такая функциональность протокола может также использоваться и для режима рукопожатия PSK, во время которого сертификаты не используются, но зато после завершения рукопожатия можно запросить сертификат клиента (если ранее клиент включил расширение “post_handshake_auth” в сообщение Client Hello).

4. ВЕРИФИКАЦИЯ ПРОТОКОЛА

В процессе тестирования сетевых протоколов решается несколько важных задач: проверяется функциональная совместимость различных реализаций, проверяются соответствие реализации требованиям спецификации и устойчивость реализации к нестандартным воздействиям.

В наших проектах используются наработанные нами методики по тестированию сетевых протоколов: автоматизированное тестирование на соответствие формальным спецификациям и методы мутации данных.

В текущих экспериментах используется модель протокола TLS версии 1.3, разработанная нами на основе спецификаций RFC и описывающая сложную схему функционирования протокола.

Для тестирования реализаций на соответствие формальным спецификациям используется технология UniTESK, предоставляющая средства автоматизации тестирования на основе использования конечных автоматов [5]. Состояния тестируемой системы задают состояния автомата, а тестовые воздействия – переходы этого автомата. При выполнении перехода заданное воздействие передается на тестируемую реализацию, после чего регистрируются реакции реализации и автоматически выносится вердикт о соответствии наблюдаемого поведения спецификации. В UniTESK алгоритм обхода конечного автомата реализован как внутренний компонент и не зависит от протокола и тестируемой системы.

Мутационные методы тестирования используются для обнаружения нестандартного поведения тестируемой системы (завершение из-за фатальной

ошибки, «подвисание», ошибки доступа к памяти). Как правило, подобные ситуации не рассматриваются в спецификациях. В поток обмена протокола, создаваемый в соответствии со спецификацией, вносятся некоторые изменения: либо в сообщениях, сформированных на основе разработанной модели протокола, изменяются значения полей, либо изменяется порядок сообщений в потоке обмена. Модель протокола позволяет изменять данные на любом этапе обмена, что позволяет тестовому сценарию проходить через все значимые состояния протокола и в каждом таком состоянии проводить тестирование реализации в соответствии с заданной программой.

5. ТЕСТОВЫЙ СТЕНД

Для тестирования реализаций сервера протокола TLS используются два сетевых узла. На одном узле функционирует модельная реализация под управлением UniTESK, выполняются основной поток управления тестовыми сценариями, обход тестового автомата и верификация наблюдаемых реакций. На другом узле функционирует тестируемая реализация. Тестовые сообщения протокола, сформированные модельной реализацией, передаются тестируемой системе, после чего регистрируются реакции тестируемого узла.

В качестве реализаций сервера TLSv1.3 используются:

– реализация TLS в виртуальной машине Java, JDK-14 (Java Development Kit) [6],

– реализация TLS библиотеки openssl-1.1.1f [7],

– реализацияTLSв Windows 11 Pro 21H2 22000-100 (IIS).

Эти реализации являются частью широко используемых библиотек с открытым исходным кодом, имеют развитую функциональность и обеспечивают хорошую журнализацию событий.

6. РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ

ПротоколTLSv1.3 позволяет добавлять новую функциональность за счет использования механизма расширений. В предыдущей части проекта акцент был сделан на использовании базовой функциональности протокола с минимальным количеством обязательных расширений. В данной работе используются как базо-

вые, так и дополнительные (описанные выше) возможности протокола. На данный момент в рамках технологии UniTESK (с использованием инструмента JavaTesK [8]) получены следующие результаты:

– разработана модель основной функциональности протокола TLS версии 1.3,

– разработаны спецификации и медиаторы для протокола TLS 1.3,

– разработан набор тестов, покрывающий часть требований спецификации.

Найдены несколько отклонений реализаций от спецификации, касающиеся обработки расширенияcookie. Напомним, что в ответ на сообщение Client Hello, в котором отсутствует необходимый ключевой материал для выбранных сервером алгоритмов, сервер может отправить сообщение Hello Retry Request, требующее предоставить необходимые данные. После этого клиент должен прислать исправленное Client Hello, которое полностью повторяет первое, но с требуемым ключевым материалом. При этом, если сервер включил в Hello Retry Request необязательное расширение cookie, клиент должен скопировать это расширение в свое второе Client Hello.

JDK-14:

- сообщение Change Cipher Plain принимается после Client Finished (такие сообщения разрешается принимать только до Client Finished),
- сообщение от клиента с пользовательскими данными (тип Application) между сообщениями Server Finished и Client Finished принимается и игнорируется, хотя должны быть сообщения об ошибке (сообщения с типом Application клиент может отправлять только после Client Finished, при этом для шифрования Client Finished и Application используются разные ключи),
- сервер допускает несколько циклов сообщений Client Hello – Server Hello Retry (сообщение Server Hello Retry используется, если Client Hello предлагает приемлемые алгоритмы, но не хватает каких-либо данных; спецификация TLS разрешает только одно сообщение Server Hello Retry для каждого соединения),

- после обмена Рукопожатия (Handshake) сервер игнорирует тип входящих сообщений Протокола записей (Recordprotocol). Данный тип не несет никакой нагрузки, поскольку все сообщения зашифрованы, но в целях маскировки должен устанавливаться в значение для пользовательских данных (Application),
- странное поведение сервера: лишние данные после сообщения Client Hello остаются в буфере входящих сообщений сервера и в последующем рассматриваются как начальные байты следующего сообщения. При этом Client Hello не зашифровано, а остальные входящие сообщения должны быть зашифрованы. Данное поведение можно рассматривать как потенциальную уязвимость,
- реализация использует расширение cookie в сообщении HelloRetryRequest. При этом реализация успешно принимает ответное ClientHello с другим набором криптографических алгоритмов (поле TLS Cipher Suite), если клиент не включил в него расширениеcookie (если cookie присутствует, то реализация требует точного повторения всех полей, как и положено по спецификации),
- сообщение Hello Retry Request содержит алгоритм, для которого требуется прислать ключевой материал. Если клиент включил во второе ClientHello расширение cookie, то он может отправить ключевой материал для другого алгоритма (отличного от того, что запросил сервер). Если сервер поддерживает этот новый алгоритм, то сообщение принимается.
- Таким образом, в ответ на Hello Retry Request клиент может поменять в новом Client Hello предлагаемые алгоритмы. Однако такое поведение сервера может быть следствием того, что сервер не сохраняет первоначальное состояние сеанса (спецификация допускает работу сервера без сохранения состояния).

openssl-1.1.1f:

- не отвечает на сообщение Key Update с флагом 1, хотя для входящих от клиента сообщений создает новые ключи (сообщение Key Update уведомляет получателя, что отправитель изменил ключи для своих исходя-

щих сообщений, и следующие его сообщения защищаются новыми ключами; флаг 1 указывает, что получатель должен сделать тоже самое: отправить ответное сообщение Key Update и создать новые ключи для своих исходящих сообщений),

- реализация не использует расширение cookie в сообщении Hello Retry Request (что не запрещается), но при этом игнорирует присутствие и содержание данного расширения в ответном сообщении Client Hello,
- реализация игнорирует присутствие и содержание данного расширения в начальном сообщении Client Hello (расширение cookie используется клиентом только в повторном Client Hello в ответ на получение HelloRetryRequest).

InternetInformationServices,Windows 11:

- Сервер отвечает на сообщение Client Hello с единственным расширением Supported Versions <3.4> (спецификация требует присутствия в ClientHello еще нескольких обязательных расширений помимо Supported Versions, таким образом, серверу следовало бы ответить ошибкой и разорвать соединение).
- Сервер принимает Client Hello с дубликатами расширений (спецификация допускает присутствие только по одному расширению каждого типа).
- Сервер принимает от клиента сообщения Application и Record Layer Undefined Type Message до получения Client Finished (т. е. до завершения рукопожатия, эти сообщения уже зашифрованы сеансовыми ключами). Сервер игнорирует эти сообщения (хотя они нарушают допустимый порядок сообщений рукопожатия) и продолжает сеанс вместо разрыва соединения. Record Layer Undefined Type Message – тестовый тип сообщения, в котором в заголовке Record Layer поле Type выставлено в произвольное не зарегистрированное значение; Application – сообщение с пользовательскими данными.

ЗАКЛЮЧЕНИЕ

В работе представлен опыт верификации реализаций сервера криптографического протокола TLS версии 1.3. Протокол TLS является одним из наиболее широко используемых протоколов безопасности, обеспечивающих шифрование и защиту целостности данных при взаимной аутентификации сторон. Его используют в качестве транспорта многие другие прикладные протоколы и приложения (почтовые клиенты, интернет-браузеры, приложения голосовой и видеосвязи и др.). Появление версии 1.3 протокола TLS сделалоактуальным проведение исследований в области верификации и безопасности реализаций этой новой версии протокола.

В данном проекте мы развиваем оригинальный подход к верификации реализаций и разработке тестового набора, сочетающий в себе два направления тестирования: тестирование реализаций протокола на соответствие требованиям стандарта и методы мутационного тестирования, предназначенные для обнаружения неадекватного поведения тестируемой системы (завершение из-за фатальной ошибки, «подвисание», ошибки доступа к памяти) в ситуациях, не определенных спецификацией.

Разработка модели протокола и тестового набора проводилась с использованием инструмента JavaTESK и языка программирования Java. С помощью этого нового тестового набора было проведено тестирование трех реализаций сервера протокола TLSv1.3: в виртуальной машине Java, JDK-14 (Java Development Kit), в библиотеке openssl-1.1.1f и реализация TLS в Windows 11 Pro 21H2 22000-100 (IIS). Результатом этих экспериментов стало обнаружение целого ряда отклонений этих реализаций от требований спецификации.

Таким образом, используемый нами подход к тестированию, совмещающий методы тестирования на соответствие требованиям спецификации и методы мутации данных, а также проведенные эксперименты по тестированию реализации протокола TLSv1.3 показали, что предложенный в данном проекте метод верификации позволяет эффективно автоматизировать тестирование; тестовые наборы обладают более полным покрытием требований и выявляют большее число ошибок, чем тестовые наборы, разработанные вручную или не использующие средств мутационного тестирования. Заметим также, что используемый нами

подход доказал свою эффективность и в наших предыдущих проектах при тестировании сетевых протоколов, обеспечив обнаружение различных отклонений от спецификации и других ошибок [9–11].

Исследование выполняется при поддержке РФФИ, проект № 20-07-00493 «Верификация функций безопасности и оценка устойчивости к атакам реализаций протокола TLS версии 1.3».

СПИСОКЛИТЕРАТУРЫ

1. *Dierks T., Rescorla E.* The Transport Layer Security (TLS) Protocol Version 1.2. August 2008. IETF RFC 5246. URL: https://tools.ietf.org/html/rfc5246

2. *Rescorla E.* The Transport Layer Security (TLS) Protocol Version 1.3. August 2018. IETF RFC 8446. URL: https://tools.ietf.org/html/rfc8446

3. *Eastlake D.* Transport Layer Security (TLS) Extensions: Extension Definitions. January 2011. IETF RFC 6066. URL: https://tools.ietf.org/html/rfc6066

4. *Thomson M.* Record Size Limit Extension for TLS. August 2018. IETF RFC 8449. URL: https://tools.ietf.org/html/rfc8449

5. *Bourdonov I., Kossatchev A., Kuliamin V., and Petrenko A.* UniTesK Test Suite Architecture // Proceedings of FME 2002. LNCS 2391. P. 77–88, Springer-Verlag, 2002.

6. Java Development Kit 14.0.1 GA. URL: https://jdk.java.net/14/

7. OpenSSL Project. URL: https://www.openssl.org/

8. JavaTESK. URL: http://www.unitesk.ru/content/category/5/25/60/

9. *Никешин А.В., Пакулин Н.В., Шнитман В.З.* Разработка тестового набора для верификации реализаций протокола безопасности TLS // Труды ИСП РАН. 2012. Т. 23. С. 387–404.

10. *Никешин А.В., Пакулин Н.В., Шнитман В.З.* Тестирование реализаций клиента протокола TLS // Труды ИСП РАН. 2015.Т. 27, вып. 2. С. 145–160.

11. *Никешин А.В., Шнитман В.З.* Тестирование соответствия реализаций протокола EAP и его методов спецификациям Интернета // Труды ИСП РАН. 2018. Т. 30, вып. 6. С. 89–104. URL: https://doi.org/10.15514/ISPRAS-2018-30(6)-5

EXPERIENCE OF IMPLEMENTATION OF THE PROTOCOL TLS 1.3 VERIFICATION

A. V. Nikeshin[1] **[0000-0001-5781-9736], V. Z. Shnitman**[2] **[0000-0002-1509-0972]**

Ivannikov Institute for System Programming of the Russian Academy of Sciences, Alexander Solzhenitsyn st., 25, Moscow, 109004

[1]alexn@ispras.ru, [2]vzs@ispras.ru

Abstract

This paper presents the experience of verifying server implementations of the TLS cryptographic protocol version 1.3. TLS is a widely used cryptographic protocol designed to create secure data transmission channels and provides the necessary functionality for this: confidentiality of the transmitted data, data integrity, and authentication of the parties. The new version 1.3 of the TLS protocol was introduced in August 2018 and has a number of significant differences compared to the previous version 1.2. A number of TLS developers have already included support for the latest version in their implementations. These circumstances make it relevant to do research in the field of verification and security of the new TLS protocol implementations. We used a new test suite for verifying implementations of the TLS 1.3 for compliance with Internet specifications, developed on the basis of the RFC8446, using UniTESK technology and mutation testing methods. The current work is part of the TLS 1.3 protocol verification project and covers some of the additional functionality and optional protocol extensions. To test implementations for compliance with formal specifications, UniTESK technology is used, which provides testing automation tools based on the use of finite state machines. The states of the system under test define the states of the state machine, and the test effects are the transitions of this machine. When performing a transition, the specified impact is passed to the implementation under test, after which the implementation's reactions are recorded and a verdict is automatically made on the compliance of the observed behavior with the specification. Mutational testing methods are used to detect non-standard behavior of the system under test by transmitting incorrect data. Some changes are made to the protocol exchange flow created in accordance with the specification: either the values of the message fields formed on the basis of the developed protocol model are changed, or the order of messages in the

exchange flow is changed. The protocol model allows one to make changes to the data flow at any stage of the network exchange, which allows the test scenario to pass through all the significant states of the protocol and in each such state to test the implementation in accordance with the specified program. So far, several implementations have been found to deviate from the specification. The presented approach has proven effective in several of our projects when testing network protocols, providing detection of various deviations from the specification and other errors.

Keywords: *security, TSL, TSLv1.3, protocols, testing, verification, evaluate robustness, Internet, standards, formal specifications.*

REFERENCES

1. *Dierks T., Rescorla E.* The Transport Layer Security (TLS) Protocol Version 1.2. August 2008. IETF RFC 5246. URL: https://tools.ietf.org/html/rfc5246

2. *Rescorla E.* The Transport Layer Security (TLS) Protocol Version 1.3. August 2018. IETF RFC 8446. URL: https://tools.ietf.org/html/rfc8446

3. *Eastlake D.* Transport Layer Security (TLS) Extensions: Extension Definitions. January 2011. IETF RFC 6066. URL: https://tools.ietf.org/html/rfc6066

4. *Thomson M.* Record Size Limit Extension for TLS. August 2018. IETF RFC 8449. URL: https://tools.ietf.org/html/rfc8449

5. *Bourdonov I., Kossatchev A., Kuliamin V., and Petrenko A.* UniTesK Test Suite Architecture // Proceedings of FME 2002. LNCS 2391. P. 77–88, Springer-Verlag, 2002.

6. Java Development Kit 14.0.1 GA. URL: https://jdk.java.net/14/

7. OpenSSL Project. URL: https://www.openssl.org/

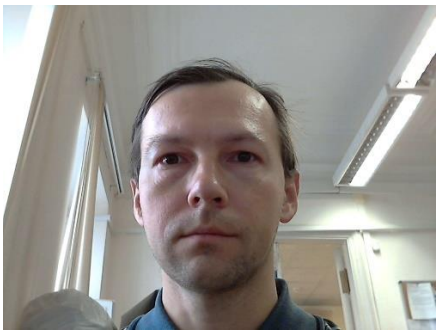
8. JavaTESK. URL: http://www.unitesk.ru/content/category/5/25/60/

9. *Nikeshin A.V., Pakulin N.V., Shnitman V.Z.* Razrabotka testovogo nabora dlya verifikatsii realizatsiy protokola bezopasnosti TLS // Trudy ISP RAN /Proc. ISP RAS. 2012. Vol. 23. P. 387–404.

10. *Nikeshin A.V., Pakulin N.V., Shnitman V.Z.* TLS Clients Testing // Trudy ISP RAN /Proc. ISP RAS. 2015.Vol. 27, issue 2. P. 145–160.

11. *Nikeshin A.V., Shnitman V.Z.* Conformance testing of Extensible Authentication Protocol implementations // Trudy ISP RAN/Proc. ISPRAS. 2018. Vol. 30, issue 6. P. 89–104 (in Russian). URL: https://doi.org/10.15514/ISPRAS-2018-30(6)-5

СВЕДЕНИЯОБАВТОРАХ



НИКЕШИН Алексей Вячеславович – научный сотрудник Института системного программирования им. В.П. ИванниковаРАН.

Aleksey Vyacheslavovich NIKESHIN– researcher, Ivannikov Institute for System Programming of the RAS.

Email: alexn@ispras.ru

ORCID: 0000-0001-5781-9736

ШНИТМАН Виктор Зиновьевич – д. т. н., с. н. с., заведующий отделом Института системного программирования им. В.П. Иванникова РАН.

Victor Zinovievich SHNITMAN – Doctor of Technical Sciences, Senior Research Officer, Head of Department**,** Ivannikov Institute for System Programming of the RAS.

Email: vzs@ispras.ru

ORCID: 0000-0002-1509-0972

Материал поступил в редакцию 22 октября 2021 года