УДК 004.05

ОПЫТ ВЕРИФИКАЦИИ РЕАЛИЗАЦИЙ КЛИЕНТА 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 клиента криптографического протокола TLS версии 1.3. TLS сегодня является одним из наиболее востребованных криптографических протоколов, предназначенных для создания защищенных каналов передачи данных. Протокол обеспечивает необходимую для своих задач функциональность: конфиденциальность передаваемых данных, целостность данных, аутентификацию сторон. В новой версии протокола TLS 1.3 была существенно переработана архитектура, устранен ряд недостатков предыдущих версий, выявленных как при разработке реализаций, так и в процессе их эксплуатации.

В работе использован новый тестовый набор для верификации реализаций клиента протокола TLS 1.3 на соответствие спецификациям интернет, разработанный на основе спецификации RFC 8446 с использованием технологии UniTESK и методов мутационного тестирования. 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 спецификации. Мутационные методы тестирования используются для обнаружения нестандартного поведения тестируемой системы (завершение из-за фатальной ошибки, «подвисание», ошибки доступа к памяти) с помощью передачи некорректных данных, такие ситуации часто остаются за рамками

требований спецификаций. В сообщения, сформированные на основе разработанной модели протокола, вносятся какие-либо изменения. Модель протокола дает возможность 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 соответствие с заданной программой. Представленный подход доказал свою эффективность в нескольких наших проектах при тестировании сетевых протоколов, обеспечив обнаружение различных отклонений от спецификации и других ошибок. Текущая работа является частью проекта верификации протокола TLS 1.3 и охватывает реализации клиентской части протокола.

***Ключевые слова**: безопасность, TLS, TLSv1.3, протоколы, тестирование, оценка устойчивости, интернет, стандарты, формальные методы спецификации.*

ВВЕДЕНИЕ

Новая версия протокола TLS 1.3, представленная в августе 2018 года, постепенно вытесняет устаревшие версии [1, 2]. Многие разработчики программного обеспечения (ПО) уже включили поддержку TLS 1.3 в свои реализации. Поэтому исследования в области верификации и безопасности реализаций новой версии протокола остаются актуальной задачей, решающей несколько важных задач: проверка функциональной совместимости различных реализаций, проверка соответствия реализаций требованиям спецификации и устойчивость реализаций к нестандартным воздействиям.

В новой версии протокола оптимизирована структура обмена сообщениями, последние сгруппированы в несколько блоков: обмен ключами, параметры сервера, фаза аутентификации. Сделан акцент на шифрование как можно большего количества данных. Часть механизмов протокола перенесена в расширения, а набор сервисов безопасности расширен.

Предыдущие работы были посвящены верификации серверных реализаций протокола TLSv1.3 [3,4]. Текущая работа является ее продолжением и охватывает реализации клиентской части протокола.

В настоящей работе использованы наработанные нами методики тестирования сетевых протоколов: автоматизированное тестирование на соответствие формальным спецификациям и методы мутации данных, доказавшие свою эффективность в наших предыдущих проектах при тестировании сетевых протоколов.

Структура статьи: в первом разделе представлен краткий обзор новой версии протокола, во втором разделе обозначены применяемые методы верификации, в третьем разделе описан тестовый стенд, в четвертом разделе представлены результаты тестирования.

1. КРАТКИЙ ОБЗОР НОВОЙ ВЕРСИИ ПРОТОКОЛА

Напомним общую архитектуру протокола.

TLS предназначен для создания защищенных каналов передачи данных, обеспечивая их конфиденциальность и целостность, а также аутентификацию сторон. Механизм обработки сообщений состоит из двух уровней:

– Протокол рукопожатия (Handshake protocol) 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 (Record protocol) 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) и не зависит от протоколов прикладного уровня.

Далее представлено 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}
[Application Data]  <-------->  [Application Data]
```

* 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.

Как видно из схемы обмена, сообщения сгруппированы в три блока:

Первый блок предназначен для обмена ключами и состоит из двух сообщений: ClientHello и ServerHello. Клиент, начиная диалог, отправляет ClientHello, в которое включает наборы поддерживаемых алгоритмов и соответствующий ключевой материал. Сервер выбирает из предложенного множества один криптографический набор, который соответствует его политикам безопасности, добавляет свой ключевой материал и отправляет эти данные клиенту. Здесь применяются встроенные механизмы защиты от понижения согласованного номера версии протокола TLS, а сам процесс согласования версий перемещен в расширения, что

должно улучшить совместимость с существующими серверами и промежуточными устройствами, которые часто неправильно реализуют согласование версий.

Теперь сервер обладает полным набором данных, чтобы вычислить все необходимые сеансовые ключи. Потому все последующие сообщения (в отличие от предыдущих версий TLS) отправляются в зашифрованном виде. Кроме этого, спецификация разрешает серверу сразу начать отправку прикладных данных, не дожидаясь получения ответа с аутентификацией клиента. Такая схема обмена позволяет быстрее начать передачу пользовательских данных за счет уменьшения количества служебных сообщений (в предыдущей версии TLS присутствовало дополнительное обязательное сообщение ChangeCipherSpec, и требовался дополнительный раунд обмена). Однако у нее есть и обратная сторона. Во-первых, в этой точке обмена любые данные посылаются не аутентифицированному клиенту. Во-вторых, появляется возможность проведения эффективных DOS-атак (Denial of Service, отказ в обслуживании), поскольку затраты на серию сообщений ClientHello небольшие, а сервер в ответ на каждое такое сообщение вынужден вычислять полный набор ключей и хэш-суммы сообщений CertificateVerify и Finished (при этом никакой аутентификации от клиента не требуется).

Второй блок сообщений передает параметры сервера. Сообщение EncryptedExtensions содержит расширения, ответные для соответствующих расширений ClientHello, которые не требуются для создания криптографических параметров (например, поддержка протокола прикладного уровня). Такие расширения ранее отправлялись в открытом виде. Также сервер может запросить сертификат клиента.

Третий блок сообщений – фаза аутентификации. Сообщения Certificate, CertificateVerify, Finished выполняют аутентификацию сервера (и, при необходимости, клиента), подтверждают созданные сеансовые ключи и обеспечивают целостность всего обмена рукопожатия.

Для случаев заранее распределенных ключей (PSK, pre-shared key) TLS 1.3 позволяет использовать сокращенный режим рукопожатия, в нем используется меньше сообщений для обмена и трудоемких вычислений с ключами, что может быть полезно в условиях отсутствия инфраструктуры для работы с сертификатами.

```
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 можно использовать 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 прямую секретность, forward secrecy), так и сложность дополнительных криптографических вычислений.

Из других особенностей новой версии протокола TLS можно отметить механизм возобновления сессии. В предыдущей версии для этого существовал специальный режим возобновления сеанса. Теперь же используются общий ключ, согласованный в предыдущем обмене рукопожатия, и указанный выше обмен PSK.

На основе режима рукопожатия PSK реализован новый для протокола TLS способ отправки так называемых «ранних данных» (early data, 0-RTT). 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]
```

Данные 0-RTT добавляются к рукопожатию 1-RTT в первом рейсе в сообщение ClientHello. Однако свойства безопасности для таких данных слабее, чем для других данных TLS, например, нет защиты от повторного воспроизведения между разными соединениями (в спецификации рассмотрены такие виды атак), хотя внутри одного соединения данные 0-RTT дублироваться не могут, т. к. данные 0-RTT и 1-RTT защищены разными ключами.

TLSv1.3 как расширяемый протокол предоставляет дополнительные, необязательные сервисы безопасности, согласуемые, как правило, через механизм расширений. Среди них можно отметить несколько наиболее важных.

Это механизм обновления сеансовых ключей, повышающий безопасность передачи данных; никаких дополнительных данных передавать не требуется. Сообщение KeyUpdate просто сообщает партнеру, что отправитель создал новые сеансовые ключи (используется предыдущий ключевой материал), и последующие сообщения зашифрованы новыми ключами.

Расширение ServerName позволяет клиенту указать имя сервера, с которым он связывается [5]. Это может пригодиться, если по одному сетевому адресу размещено несколько «виртуальных» серверов, для каждого из которых используется свой сертификат. Для сервера получение этого расширения носит рекомендательный характер, при этом должны учитываться и другие настройки политик безопасности.

Расширение MaxFragmentLength позволяет клиенту использовать сообщения меньшего размера, чем предусмотрено спецификацией (2^{14} байт) [5]. Однако у него есть ряд существенных недостатков, связанных с фиксированным набором неизменяемых значений. Расширение RecordSizeLimit решает эти проблемы, позволяя задать произвольное максимальное значение сообщений (не превышающее значение, определенное версией протокола, согласованной партнерами) для каждого направления передачи данных. В спецификации более подробно изложены преимущества и недостатки этих двух расширений [6].

Расширение PostHandshakeAuth указывает серверу, 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 CertificateVerify не используется.

Сервер может отправить несколько сообщений CertificateRequest, 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 ClientHello).

2. ВЕРИФИКАЦИЯ ПРОТОКОЛА

В процессе тестирования сетевых протоколов решается несколько важных задач: проверяются функциональная совместимость различных реализаций, 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 использована 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 применена технология UniTESK, предоставляющая средства автоматизации тестирования на основе использования конечных автоматов [7]. 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 сообщения, сформированные на основе разработанной модели протокола, вносятся какие-либо изменения. Модель протокола позволяет изменять данные на любом этапе обмена, что дает возможность тестовому сценарию проходить через все значимые состояния протокола и в каждом таком состоянии проводить тестирование реализации в соответствии с

заданной программой.

3. ТЕСТОВЫЙ СТЕНД

Для тестирования реализаций клиента протокола TLS были использованы 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.

Работа с реализациями клиентов имеет свои особенности по сравнению с серверами. Сервер представляет собой непрерывно работающий процесс, прекращение работы этого процесса, как правило, является сбоем его работы. Клиентское приложение с точки зрения установления соединения работает короткое время и завершает свою работу либо установив соединение, либо ошибкой. Для многократного повторения этих действий требуется дополнительный агент.

В качестве реализаций клиента TLSv1.3 были выбраны:

– реализация TLS в виртуальной машине Java, JDK-14 (Java Development Kit) [8],

– реализация TLS библиотеки openssl-3.0.5 [9],

– интернет-браузер Mozilla Firefox, Portable Edition 97.0.1 [10],

– интернет-браузер Chromium, Portable Edition 103.0.5060.114 (Official Build, ungoogled-chromium) [11],

– интернет-браузер Atom 25.0.0.24 [12].

Первые две реализации являются частью широко используемых библиотек с открытым исходным кодом.

4. РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ

На данный момент в рамках технологии UniTESK (с использованием инструмента JavaTesK [13]) получены следующие результаты:

– расширена модель основной функциональности протокола TLS версии 1.3 для работы с реализациями клиента,

– разработан набор тестов для тестирования реализаций клиентов протокола, покрывающий часть требований спецификации.

Найдено несколько отклонений реализаций от спецификации.

JDK-14:

- В сообщениях ClientHello/ServerHello значения поля session_ID должны совпадать (для TLSv1.3), клиент должен это проверять. Реализация игнорирует данное поле.
- В сообщении ServerHello расширение signature_algorithms_cert с некорректными значениями игнорируется. Данное расширение задает допустимые алгоритмы подписи для сертификатов.
- Неизвестные расширения (а также дубликаты таких расширений) в сообщениях ServerHello/ServerHelloRetry игнорируются. Сообщение ServerHello может содержать только расширения, которые присутствовали в ClientHello (исключением является расширение Cookie). При этом расширения должны присутствовать в единственном экземпляре.
- Если сообщение ClientHello содержит недостаточно параметров, сервер может отправить сообщение ServerHelloRetry, предлагающее клиенту прислать исправленное сообщение ClientHello. Повторные сообщения ServerHelloRetry не допускаются. Реализация отвечает обычным образом на повторные сообщения ServerHelloRetry от сервера.
- Сообщение ServerHelloRetry должно содержать несколько обязательных расширений. Реализация клиента принимает сообщение ServerHelloRetry с единственным расширением Supported_versions (расширение необходимо, чтобы использовать версию TLS 1.3).
- Сообщение ServerHelloRetry в расширении KeyShare содержит алгоритм, который сервер желает использовать и который поддерживается клиентом (присутствует в ClientHello), но ключ для которого клиент не прислал. ServerHelloRetry не должно предлагать алгоритмы, ключи для которых уже присутствуют в ClientHello. Реализация клиента принимает такие некорректные сообщения.
- В ответ на ServerHelloRetry клиент присылает исправленное сообще-

ние ClientHello. Сервер отвечает сообщением ServerHello. При этом клиент должен проверить, что выбранный для сессии криптографический набор (поле CipherSuite) в ServerHelloRetry и ServerHello один и тот же. Реализация клиента не проверяет это требование.

Openssl-3.0.5:

- В сообщении ServerHello расширение signature_algorithms_cert с некорректными значениями игнорируется. Данное расширение задает допустимые алгоритмы подписи для сертификатов.
- В сообщении ServerHello игнорируется значение поля ProtocolVersion (версия протокола), если оно больше 3.3 (например, 4.2). Значение 3.3 соответствует последней на данный момент версии TLSv1.3.
- Неизвестные расширения (а также их дубликаты) в сообщениях ServerHello/ServerHelloRetry игнорируются. Сообщение ServerHello может содержать только расширения, которые присутствовали в ClientHello (исключением является расширение Cookie). При этом расширения должны присутствовать в единственном экземпляре.

Firefox 97.0.1:

Отклонений от спецификации не обнаружено.

Chromium 103.0.5060.114:

- Реализация не отвечает на сообщение KeyUpdate (с флагом 1), требующее обновить ключевой материал текущей сессии. Данное сообщение с указанным флагом требует от партнера подтвердить ответным сообщением новый ключевой материал.
- В сообщении CertificateRequest игнорируется наличие дополнительных и недопустимых расширений (кроме необходимого SignatureAndHash_Algorithm).

Atom 25.0.0.24:

- Реализация не отвечает на сообщение KeyUpdate (с флагом 1), требующее обновить ключевой материал текущей сессии. Данное сообщение с указанным флагом требует от партнера подтвердить ответным сообщением новый ключевой материал.
- В сообщении CertificateRequest игнорируется наличие дополнительных и недопустимых расширений (кроме необходимого SignatureAndHash_Algorithm).

ЗАКЛЮЧЕНИЕ

В работе представлен опыт верификации реализаций клиента криптографического протокола TLS версии 1.3.

TLS сегодня является одним из наиболее востребованных криптографических протоколов, предназначенных для создания защищенных каналов передачи данных. Протокол обеспечивает необходимую для своих задач функциональность: конфиденциальность передаваемых данных, целостность данных, аутентификацию сторон. В новой версии протокола TLS 1.3 была существенно переработана архитектура, устранен ряд недостатков предыдущих версий, выявленных как при разработке реализаций, так и в процессе их эксплуатации.

Нами был использован новый тестовый набор для верификации реализаций клиента протокола TLS 1.3 на соответствие спецификациям интернета, разработанный на основе спецификации RFC 8446 с использованием технологии UniTESK и методов мутационного тестирования. 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 спецификации. Мутационные методы тестирования использованы для обнаружения нестандартного поведения тестируемой системы (завершение из-за фатальной ошибки, «подвисание», ошибки доступа к памяти) с помощью передачи некорректных данных, такие ситуации часто остаются за рамками требований спецификаций. В сообщения, сформированные на основе разработанной модели протокола, вносятся какие-либо изменения. Модель протокола позволяет вносить изменения в поток данных на любом этапе сетевого обмена, что дает возможность тестовому сценарию проходить через все значимые состояния протокола и в каждом таком состоянии проводить тестирование реализации в соответствии с заданной программой.

Представленный подход доказал свою эффективность в наших предыдущих

проектах при тестировании сетевых протоколов, обеспечив обнаружение различных отклонений от спецификации и других ошибок [14–16]. Текущая работа является продолжением нашего проекта верификации протокола TLS 1.3 и охватывает реализации клиентской части протокола.

На данный момент обнаружено несколько отклонений реализаций JDK-14, openssl-3.0.5 и браузеров Chromium и Atom от спецификации.

Благодарности

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

СПИСОК ЛИТЕРАТУРЫ

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. *Никешин А.В., Шнитман В.З.* Верификация функций безопасности протокола TLS версии 1.3 // Научный сервис в сети Интернет: труды XXII Всероссийской научной конференции (21–25 сентября 2020 г., онлайн). М.: ИПМ им. М.В. Келдыша, 2020. С. 515–526. https://doi.org/10.20948/abrau-2020-22

4. *Никешин А.В., Шнитман В.З.* Верификация функций безопасности расширений протокола TLS 1.3 // Научный сервис в сети Интернет: труды XXIII Всероссийской научной конференции (20–23 сентября 2021 г., онлайн). М.: ИПМ им. М.В. Келдыша, 2021. С. 251–264. https://doi.org/10.20948/abrau-2021-14

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

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

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

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

9. *OpenSSL Project.* URL: https://www.openssl.org/

10. *Mozilla Firefox, Portable Edition 97.0.1.*
URL: https://portableapps.com/apps/internet/firefox_portable/

11. *Ungoogled Chromium Portable 103.0.5060.114.*
URL: https://portapps.io/app/ungoogled-chromium-portable/

12. *Atom 25.0.0.24.* URL: https://browser.ru/

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

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

15. *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.

16. *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. https://doi.org/10.15514/ISPRAS-2018-30(6)-5

EXPERIENCE OF TLS 1.3 CLIENTS 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 client implementations of the TLS cryptographic protocol version 1.3. TLS is a widely used cryptographic protocol today, designed to create secure data transmission channels. The protocol provides the necessary functionality for its tasks: confidentiality of transmitted data, data integrity, and authentication of the parties. In the new version 1.3 of the TLS architecture was significantly redesigned, eliminating a number of shortcomings of previous versions that were identified both during the development of implementations and during their operation. We used a new test suite for verifying client 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. 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. 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. The current work is part of the TLS 1.3 protocol verification project and covers TLS client implementations.

Keywords*: security, TLS, TLSv1.3, 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. *Nikeshin A.V., Shnitman V.Z.* Verification of security properties of the TLS protocol version 1.3 // Nauchnyi servis v seti Internet: trudy XXII Vserossiiskoi nauchnoi konferentsii (21–25 sentiabria 2020 g., online). M.: IPM im. M.V. Keldysha, 2020. P. 515–526. https://doi.org/10.20948/abrau-2020-22

4. *Nikeshin A.V., Shnitman V.Z.* Verification of security properties of the TLS 1.3

extensions // Nauchnyi servis v seti Internet: trudy XXIII Vserossiiskoi nauchnoi konferentsii (20–23 sentiabria 2021 g., online). M.: IPM im. M.V. Keldysha, 2021. P. 251–264. https://doi.org/10.20948/abrau-2021-14

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

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

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

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

9. *OpenSSL Project.* URL: https://www.openssl.org/

10. *Mozilla Firefox, Portable Edition 97.0.1.*
URL: https://portableapps.com/apps/internet/firefox_portable/

11. *Ungoogled Chromium Portable 103.0.5060.114.*
URL: https://portapps.io/app/ungoogled-chromium-portable/

12. *Atom 25.0.0.24.* URL: https://browser.ru/

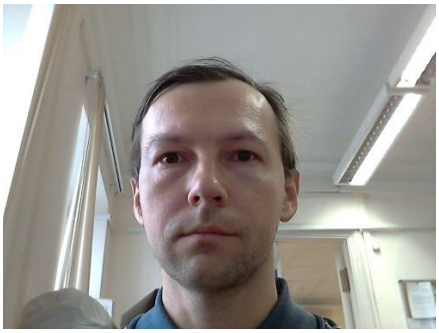
13. *JavaTESK.* URL: http://www.unitesk.ru/content/category/5/25/60/

14. *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.

15. *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.

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

СВЕДЕНИЯ ОБ АВТОРАХ



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

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 30 января 2023 года