Тестирование реализаций клиента протокола TLS

А.В.Никешин <alexn@ispras.ru>
Н.В.Пакулин <npak@ispras.ru>
В.З. Шнитман <vzs@ispras.ru>
Институт системного программирования РАН,
109004, Россия, г. Москва, ул. А. Солженицына, дом 25

Аннотация. При разработке протоколов безопасности особую роль играют задачи обеспечения качества, надёжности и отказоустойчивости реализаций. Разработчики стандартов для таких протоколов ответственно подходят к разработке спецификаций и стараются предусмотреть множество возможных атак для того, чтобы минимизировать риск успешных атак на реализации. Поэтому для надежности реализаций очень важно соответствовать требованиям спецификации протокола, прежде всего в части обработки ошибок во входящих сообщениях или ошибки в последовательности сообщений, так как именно такие ситуации являются основным средством для осуществления атак на реализации протоколов.

Одним из основных инструментов для проверки соответствия реализации стандарту является тестирование. Данная статья продолжает серию работ авторов по автоматизации тестирования соответствия спецификациям протоколов безопаности интернета. В статье описаны результаты работы по созданию тестового набора для тестирования соответствия реализаций клиента протокола TLS спецификациям Интернета. В качестве базы для построения тестов мы использовали технологию UniTESK и программный пакет JavaTesK, реализующий эту технологию. Для построения атакующих воздействий на целевую реализацию использовался подход мутационного тестирования, при котором вместо корректных сообщений строятся их искаженные по определённым правилам копии. Были разработаны операторы мутации для некоторых основных типов данных, которые используются в формальной модели протокола. Подход был апробирован на ряде известных открытых реализаций клиентов протокола TLS. Этот подход доказал свою эффективность, поскольку обеспечил обнаружение отклонений от спецификации и других ошибок во всех выбранных реализациях клиента протокола.

Ключевые слова: тестирование; верификация; формальные методы; формальные спецификации; тестирование с использованием моделей; TLS; SSL; UniTESK; мутационное тестирование.

DOI: 10.15514/ISPRAS-2015-27(2)-9

Для цитирования: Никешин А.В., Пакулин Н.В., Шнитман В.З. Тестирование реализаций клиента протокола TLS. Труды ИСП РАН, том 27, вып. 2, 2015 г., стр. 145-160. DOI: 10.15514/ISPRAS-2015-27(2)-9.

1. Введение

Технология UniTESK поддерживает нотацию формальных спецификаций, автоматическую динамическую генерацию тестовых воздействий и автоматический анализ результатов. UniTESK предоставляет средства для реализации формальных спецификаций протоколов в виде контрактных спецификаций и переходов расширенного конечного автомата [1]. Наш подход к применению технологии UniTESK для тестирования соответствия, в частности, вопросы построения формальной модели из текстовой спецификации и построения тестовых сценариев подробно изложены в [2,3]. Однако кроме воздействий, определяемых спецификацией, для более качественной проверки реализаций необходимо проводить тестирование на входных данных, противоречащих спецификации или не определяемых ею. Такие ситуации постоянно возникают из-за ошибок пользователей или из-за целенаправленных действий злоумышленника. Особенно это актуально для систем безопасности. Поэтому мы расширили наш тестовый набор средствами мутационного тестирования.

Наши предыдущие работы, относящиеся к протоколу TLS, были сосредоточены вокруг серверной части этого протокола [4]. В то же время, широко применяются клиентские приложения, использующие этот протокол (в том числе почтовые клиенты, web-браузеры и др.). Сегодня известны многочисленные уязвимости, найденные, например, в различных web-сервисах [5,6] (особенно использующих сценарии Java/Java scripts), демонстрирующие, что неадекватное поведение клиентских приложений может приводить к проблемам безопасности со стороны клиента. Кроме того, по объективным причинам тестированию клиентских приложений уделяется меньше внимания: многие серверные приложения являются коммерческими продуктами, в то время как, соответствующие клиентские решения часто предлагаются бесплатно. Поэтому тестирование клиентских приложений является актуальной задачей. Нами был разработан новый тестовый набор для TLS-клиентов, дополненный операторами мутации. Ниже дано краткое описание разработанного тестового набора и приведены результаты его апробации.

2. Тестовый стенд

Тестовый стенд для тестирования клиентов протокола TLS состоит из двух сетевых узлов: инструментального и целевого. На инструментальном узле исполняется основной поток управления тестовой системы, обход тестового автомата и верификация наблюдаемых реакций. На целевом узле функционирует тестируемая реализация и дополнительный агент, через

который инструментальный узел осуществляет управление реализацией. 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.

3. Инструментальные средства

Разработка тестового набора велась с использованием инструмента автоматизированного тестирования JavaTesK [7], который реализует технологию UniTESK на базе объектно-ориентированного языка с автоматической сборкой мусора и обеспечивает значительное упрощение модели протокола и тестовых сценариев.

4. Формальная спецификация

Формальная спецификация TLS состоит из следующих компонентов:

- модельного состояния, которое содержит набор структур данных, моделирующих концептуальные структуры данных из стандарта TLS;
- спецификационных стимулов, которые формализуют требования к изменению состояния реализации TLS при внешнем воздействии на систему;
- спецификационных реакций, которые формализуют требования к реакциям реализации TLS на внешние воздействия.

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

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

Параметрами спецификационных методов служат сообщения TLS в модельном представлении, т.е. объекты Java. Наборы полей соответствующих классов следуют структуре сообщений, описанных в стандарте TLS [8,9].

5. 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 и RFC 5746 [8,9].

TLS-соединение модели протокола 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 включает:

sessionID идентификатор сессии, соответствующей данному соединению;

keys ключи криптографических алгоритмов;

sequence number порядковый номер сообщений;

security parameters параметры безопасности.

Модельный тип параметров безопасности соответствует структурам, описанным в стандарте:

```
struct {
    ConnectionEnd         entity;
    PRFAlgorithm          prf_algorithm;
    BulkCipherAlgorithm   bulk_cipher_algorithm;
    CipherType            cipher_type;
    uint8                 enc_key_length;
```

```
        uint8               block_length;
        uint8               fixed_iv_length;
        uint8               record_iv_length;
        MACAlgorithm        mac_algorithm;
        uint8               mac_length;
        uint8               mac_key_length;
        CompressionMethod   compression_algorithm;
        opaque              master_secret[48];
        opaque              client_random[32];
        opaque              server_random[32];
    } SecurityParameters;
```

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,
enc_key_length,
block_length,
fixed_iv_length,
record_iv_length

алгоритм шифрования и соответствующие ему параметры;

mac_algorithm,
mac_length,
mac_key_length

алгоритм защиты целостности сообщений и соответствующие ему параметры;

compression_algorithm алгоритм сжатия;

master_secret мастер-ключ;

client_random, server_random одноразовые массивы байт клиента и сервера;

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.

6. 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.

Разработанная библиотека модельного представления сообщений TLS позволяет создавать различные варианты таких сообщений.

7. Тестирование TLS-клиента

В роли TLS-клиента реализация генерирует запросы. Дополнительный агент на целевом узле позволяет передавать команды от тестовой системы к реализации. В начале каждого теста через агента инициируется TLS-запрос от клиента. Далее стимулами являются сообщения от инструментального узла в качестве ответов клиенту.

В тестовом сценарии создается TLS-соединение, в рамках которого формируются ответы реализации клиента 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 сообщений.

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.

8. Выбор реализации для тестирования

В качестве клиентов были выбраны семь открытых реализаций:

Библиотечные реализации протокола:

- реализация TLS в виртуальной машине Java 1.7.0_05 (JSSE - Java Secure Socket Extension) [10];
- встроенная реализация клиента TLS библиотеки openssl-1.0.1j [11].

Интернет браузеры:

- интернет браузер Mozilla Firefox 34.0.5 [12];
- интернет браузер Opera 12.17 [13];
- интернет браузер SRWare Iron 42.0.2250.1 [14].

Почтовые клиенты:

- Mozilla Thunderbird 31.7.0 [15];
- TheBat 6.8.2 [16].

9. Результаты тестирования реализаций клиента протокола TLS на соответствие требованиям спецификаций RFC 5246 и RFC 5746

При выполнении разработанного нами тестового набора был выявлен ряд особенностей, нарушений требований RFC 5246/5746 и ошибок реализаций.

Реализация JSSE:

- принимает сообщения большей длины, чем допускается спецификацией;
- сообщения RecordLayer с неизвестным типом данных (спецификация определяет 4 типа данных: change_cipher_spec, alert, handshake, application_data), игнорируются клиентом (не вызывают ошибку);
- в сообщениях Alert, ServerHelloDone, HelloRequest, Finished обрабатывает первые 2, 4, 4, 12 байт соответственно (т.е. поле длины сообщения не проверяется) и только они учитываются в различных контрольных суммах, поэтому если длина сообщения больше установленной спецификацией вычисляется не правильное контрольное значение (HelloRequest не учитывается при вычислении контрольных сумм);

- принимает дубликаты сообщений ServerHello, Certificate, ServerKeyExchange, CertificateRequest, следующие друг за другом (т.е. ServerHello- ServerHello, Certificate- Certificate), если при этом не нарушается остальной порядок обмена;
- при получении сообщения Finished от сервера после сообщений ServerHello, ServerKeyExchange, ServerCertificate, CertificateRequest (без предварительного сообщения ChangeCipherSpec) создает TLS сессию, через которую позволяет передавать пользовательские данные (данная особенность поведения является серьезным нарушением требований спецификации);
- при получении после Finished (и создании TLS сессии) сообщения ServerHello создает еще одну TLS сессию, но после получения ServerHelloDone возвращает ошибку и закрывает сразу обе сессии;
- при отсутствии подходящего сертификата в большинстве случаев вместо пустого списка сертификатов отправляет сообщение об ошибке;
- при выборе подходящего сертификата не проверяет допустимые значения алгоритмов SignatureAndHashAlgorithm из сообщения CertificateRequest, но использует их в сообщении CertificateVerify. В случае несовпадения отправленного сертификата и доступных алгоритмов, отправляется сообщение об ошибке (вместо того, чтобы сразу выбрать правильный сертификат);
- если тип сертификата сервера не совпадает с алгоритмом обмена ключами, но при этом сообщение ServerKeyExchange корректно построено и соответствует сертификату, то соединение устанавливается.

Реализация Openssl-1.0.1j:

- принимает сообщения ServerHello с расширением “Signature Algorithms”, с неизвестными расширениями, которые клиент не

запрашивал, а также дубликаты этих расширений;

- после завершения обмена рукопожатия (как основного, так и возобновленного) на сообщение ServerHello (без дополнительного запроса со стороны клиента) отвечает сообщением ClientHello; на другие сообщения протокола рукопожатия (Handshake) отвечает сообщением ClientHello и следующим за ним сообщением об ошибке;
- при возобновлении сессии реализация всегда отправляет расширение "renegotiation_info" в сообщении ClientHello (RFC 5746), даже если сервер его не поддерживает;
- если сервер не использует расширение "renegotiation_info" при первом обмене Рукопожатия (указывая, что он его не поддерживает), а при последующих – использует с правильными контрольными данными, то клиент разрешает такие согласования (согласно спецификации клиент должен отправить сообщение об ошибке);
- если сервер использует пустое расширение "renegotiation_info" при первом обмене Рукопожатия (соглашаясь на безопасное переустановление параметров соединения), а при последующих – не использует совсем, то клиент разрешает такие согласования (что является критичным нарушением требований спецификации, и делает бесполезным использование данного расширения).

Реализация Mozilla Firefox:

- принимает сообщения ServerHello с расширением “Signature Algorithms”, а также дубликаты этого расширения (согласно спецификации данное расширение отправляется только клиентом, сервер не должен использовать его в ответе; также не допускается присутствие дубликатов расширений в одном сообщении);
- выбирает личный сертификат по своим правилам не используя данные из CertificateRequest;
- если тип сертификата сервера не совпадает с алгоритмом обмена

ключами, но при этом сообщение ServerKeyExchange корректно построено и соответствует сертификату, то соединение устанавливается;

- разрешает только безопасную переустановку криптографических параметров без разрыва соединения (session resume) (используется расширение TLS Renegotiation Indication Extension, RFC 5746), нарушая некоторые требования обратной совместимости и проверки наличия в сообщениях данного расширения. Такое поведение можно назвать особенностью реализации, поскольку оно направлено на усиление безопасности соединений.

Реализация Opera:

- если сервер использует пустое расширение "renegotiation_info" при первом обмене Рукопожатия (соглашаясь на безопасное переустановление параметров соединения), а при последующих – не использует совсем, то клиент разрешает такие согласования (что по сути делает бесполезным использование данного критичного расширения);
- не проверяет поле версии TLS в сообщении ServerHello (кроме значения 3.0), используя поле версии заголовка RecordLayer (данная особенность поведения является серьезным нарушением требований спецификации);
- информационное сообщение Alert (1,*) воспринимает как критическую ошибку (для критических ошибок должен использоваться шаблон Alert (2,*));
- при завершении передачи данных и закрытии соединения не отправляет сообщение Alert (1,0).

Реализация SRWare Iron 42.0.2250.1

- принимает сообщения ServerHello с расширением “Signature

Algorithms" и с неизвестными расширениями, которые клиент не запрашивал;

- после завершения обмена Рукопожатия на сообщение ServerHello отвечает сообщением ClientHello (вместо ошибки), на другие сообщения - сообщением ClientHello и следующим за ним сообщением об ошибке.

Реализация Mozilla Thunderbird 31.7.0:

- спрашивает у пользователя какой сертификат отправить серверу;
- если тип сертификата сервера не совпадает с алгоритмом обмена ключами, но при этом сообщение ServerKeyExchange корректно построено и соответствует сертификату, то соединение устанавливается;
- принимает сообщения ServerHello с расширением "Signature Algorithms".

Реализация TheBat 6.8.2:

- поддерживает только версию TLS1.1; считает TLS1.2 устаревшей версией и советует обновить ее до более новой.

10. Мутационное тестирование

Дополнительно с тестированием на соответствие спецификации с использованием формальных моделей сетевого протокола TLS, мы использовали методы мутационного тестирования, при котором на вход тестируемой реализации подаются заведомо некорректные данные. Мутационное тестирование основано на реализованной ранее формальной модели протокола, при котором в правильно построенные сетевые сообщения вносятся изменения. Данное тестирование использовалось на всех указанных выше реализациях клиентов TLS. Некоторые отклонения были найдены в реализациях JSSE (некорректная обработка длины некоторых сообщений, принятие дубликатов сообщений) и Opera (некорректная обработка поля версии в сообщении ServerHello) и указаны выше в перечне ошибок и особенностей реализаций.

11. Заключение

В данной статье представлены результаты работы по созданию тестового набора для тестирования соответствия реализаций клиента протокола TLS спецификациям Интернета. Наш подход для автоматизации тестирования использует технологию UniTESK: тестовая последовательность строится динамически как обход некоторого автомата теста, для построения тестовых воздействий и вынесения вердикта о корректности наблюдаемого поведения реализации используется модель протокола.

Кроме того, мы использовали метод расширения тестовых сценариев на основе моделей средствами мутационного тестирования. Такая комбинация тестовых методов позволяет улучшить качество тестирования реализаций и, как показала апробация предложенного подхода, обеспечивает обнаружение отклонений от спецификации и других ошибок во всех выбранных реализациях протокола.[1]

Список литературы

[1]. Bourdonov I., Kossatchev A., Kuliamin V., and Petrenko A. UniTesK Test Suite Architecture. //Proceedings of FME 2002. LNCS 2391, pp. 77-88, Springer-Verlag, 2002

[2]. Н.В.Пакулин, В.З.Шнитман, А.В. Никешин. "Разработка тестового набора для верификации реализаций протокола безопасности TLS // Труды Института системного программирования РАН, том 23, 2012 г. Стр. 387-404.

[3]. Н.В. Пакулин, В.З. Шнитман, А.В. Никешин. Автоматизация тестирования соответствия для телекоммуникационных протоколов. // Труды Института системного программирования РАН Том 26. Выпуск 1. 2014 г. Стр. 109-148.

[4]. Никешин А.В., Пакулин Н.В., Шнитман В.З. Автоматизация тестирования соответствия реализаций стандарту протокола безопасности транспортного уровня TLS // Научно-технические ведомости СПбГПУ. Информатика. Телекоммуникации. Управление. Выпуск 2(193)/2014. стр. 180-188. ISSN 2304-9766.

[5]. OWASP Top Ten Project, https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project

[6]. Дэйв Уайтлегг “Проверьте ваши приложения на уязвимости из списка OWASP Top 10 за 2013 год”, https://www.ibm.com/developerworks/ru/library/se-owasp-top10/

[7]. JavaTESK - http://www.unitesk.ru/content/category/5/25/60/

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

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

[10]. JavaTM Secure Socket Extension (JSSE), http://docs.oracle.com/javase/6/docs/technotes/guides/security/jsse/JSSERefGuide.html

[11]. OpenSSL Project, https://www.openssl.org/

[1] Проект выполняется при поддержке РФФИ, проект № 13-07-00869 «Разработка метода оценки устойчивости протоколов безопасности к атакам»

[12]. Mozilla Firefox, https://www.mozilla.org/ru/
[13]. Opera, http://www.opera.com/ru/
[14]. SRWare Iron, http://www.srware.net/ru/software_srware_iron.php
[15]. Mozilla Thunderbird, https://www.mozilla.org/ru/
[16]. TheBat, https://www.ritlabs.com/ru/products/thebat/

TLS Clients Testing

A.V. Nikeshin <alexn@ispras.ru>
N.V. Pakulin <npak@ispras.ru>
V.Z. Shnitman <vzs@ispras.ru>
Institute for System Programming of the Russian Academy of Sciences,
25, Alexander Solzhenitsyn st., Moscow, Russia, 109004

Abstract. Quality assuarance, reliability, fault tolerance are of major concern for developers of security protocols. Authors of specifications for those protocols take responsible approach to specification development and undertake significant efforts to study potential attacks and minimize the risk of effective exploits. Therefore it is vitally important for an implementation to conform to the corresponding protocol specification, especially in the context of error prcessing in inbound meesages or sequence of messages since such kinds of errors are the major facility for implementation of attacks against protocol implementations.
Testing is one of the primary tools for evaluation whether an implementation conforms to the specification. This paper continues the series of other publications of the authors dedicated to specification-based conformance testing for Internet security protocols. The paper presents a test suite for conformance testing of TLS protocol clients. The test suite is based on UniTESK technology of test construction and JavaTESK toolkit that implements the technology. The attacking inputs are constructed using mutation testing, building malformed test packets from correct originals following specific rules called "mutation operators". We developed mutation operators for a number of primary data types used in the formal model of the protocol. The approach was applied to a number of open-source well-known implementations of TLS. The approach proved to be feasible: a number of deviations from protocol specification and other errors were identified in all selected implementations of the protocol.

Keywords: testing, verification, formal methods, formal specifications, Model Based Testing, TLS, SSL, UniTESK, Fuzz Testing.

DOI: 10.15514/ISPRAS-2015-27(2)-9

For citation: Nikeshin A.V., Pakulin N.V., Shnitman V.Z. TLS Clients Testing. *Trudy ISP RAN/Proc. ISP RAS*, vol. 27, issue 2, 2015, pp. 145-160 (in Russian). DOI: 10.15514/ISPRAS-2015-27(2)-9.

References

[1]. Bourdonov I., Kossatchev A., Kuliamin V., and Petrenko A. UniTesK Test Suite Architecture. //Proceedings of FME 2002. LNCS 2391, pp. 77-88, Springer-Verlag, 2002

[2]. Nikeshin A.V., Pakulin N.V., Shnitman V.Z. Razrabotka testovogo nabora dlya verifikatsii realizatsiy protokola bezopasnosti TLS [Development of a test suite for the verification of implementations of the TLS security protocol], Trudyi Instituta sistemnogo programmirovaniya RAN [Proceedings of IPS RAS], 2012, Vol. 23, pp. 387–404 (in Russian).

[3]. Nikeshin A.V., Pakulin N.V., Shnitman V.Z. Avtomatizaciya testirovaniya sootvetstviya dla telecommunicacionnyh protocolov [Conformance testing automation for telecommunication protocols] // Trudyi Instituta sistemnogo programmirovaniya RAN [Proceedings of IPS RAS], 2014, Vol. 26 (Issue 1), pp. 109-148 (in Russian).

[4]. Nikeshin A.V., Pakulin N.V., Shnitman V.Z. Avtomatizacija testirovanija sootvetstvija realizacij standartu protocola bezopasnosti transportnogo urovnja TLS [Conformance testing automation for Transport Layer Security Protocol TLS], Naucno-tehnicheskie vedomosti SPbGPU. Informatika. Telekommunikacii. Upravlenie, Vol. 2(193)/2014, Pp. 180-188. ISSN 2304-9766 (in Russian).

[5]. OWASP Top Ten Project, https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project

[6]. Dave Whitelegg “Scan your app to find and fix OWASP Top 10 2013 vulnerabilities”, http://www.ibm.com/developerworks/security/library/se-owasp-top10/

[7]. JavaTESK - http://www.unitesk.ru/content/category/5/25/60/

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

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

[10]. JavaTM Secure Socket Extension (JSSE), http://docs.oracle.com/javase/6/docs/technotes/guides/security/jsse/JSSERefGuide.html

[11]. OpenSSL Project, https://www.openssl.org/

[12]. Mozilla Firefox, https://www.mozilla.org/ru/

[13]. Opera, http://www.opera.com/ru/

[14]. SRWare Iron, http://www.srware.net/ru/software_srware_iron.php

[15]. Mozilla Thunderbird, https://www.mozilla.org/ru/

[16]. TheBat, https://www.ritlabs.com/ru/products/thebat/