

Министерство науки и высшего образования Российской Федерации
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ (НИ ТГУ)
Институт прикладной математики и компьютерных наук
Кафедра компьютерной безопасности

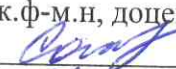
ДОПУСТИТЬ К ЗАЩИТЕ В ГЭК
Руководитель ООП
канд. техн. наук, доцент
 В.Н. Тренькаев
« 16 » января 2023 г.

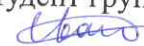
ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА СПЕЦИАЛИСТА
(ДИПЛОМНАЯ РАБОТА)

ТЕСТИРОВАНИЕ РЕАЛИЗАЦИЙ ПРОТОКОЛА TLS С ИСПОЛЬЗОВАНИЕМ
КОМБИНАТОРНОГО И ФАЗЗИНГ ТЕСТИРОВАНИЯ

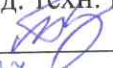
по специальности 10.05.01 Компьютерная безопасность,
специализация (профиль) «Анализ безопасности компьютерных систем»

Сапрыкин Валентин Олегович

Научный руководитель ВКР
к.ф.-м.н, доцент, доцент КБ
 С.И. Самохина
« 16 » января 2023 г.

Автор работы
студент группы № 931724
 В.О. Сапрыкин
« 16 » января 2023 г.

Министерство науки и высшего образования Российской Федерации.
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ (НИ ТГУ)
Институт прикладной математики и компьютерных наук

УТВЕРЖДАЮ
Руководитель ООП
канд. техн. наук, доцент
 В.Н. Тренькаев
« 27 » июня 2022 г.

ЗАДАНИЕ

по выполнению выпускной квалификационной работы специалиста обучающегося

Сапрыкину Валентину Олеговичу

Фамилия Имя Отчество обучающегося

по направлению подготовки Код Наименование направления подготовки,
направленность (профиль) «Наименование образовательной программы»

1 Тема выпускной квалификационной работы

Тестирование реализаций протокола TLS с использованием комбинаторного и фаззинг
тестирования

2 Срок сдачи обучающимся выполненной выпускной квалификационной работы:

а) в учебный офис / деканат – 16.01.2022 б) в ГЭК – 27.01.2022

3 Исходные данные к работе:

Объект исследования – протокол TLS

Предмет исследования – тестирование реализаций TLS на соответствие абсолютным
требованиям спецификаций

Цель исследования – разработка программного средства тестирования для проверки
соответствия реализаций TLS требованиям спецификации
протокола с использованием комбинаторного и фаззинг
тестирования на основе покрытия кода

Задачи:

- Реализация механизма создания тестового шаблона, который можно параметризовать
для автоматической генерации множества тестовых случаев с использованием
t-покрывающего комбинаторного тестирования.
- Реализация механизма систематического ограничения модели входных параметров
тестовых шаблонов «разумными» значениями.
- Создание множества тестовых шаблонов, каждый из которых проверяет требование
на основе RFC.

Методы исследования:

Методы программирования, теоретический и экспериментальный на базе ЭВМ

Организация или отрасль, по тематике которой выполняется работа, –

Кафедра компьютерной безопасности Томского государственного университета

4 Краткое содержание работы

Разработка программного средства тестирования соответствия реализаций протокола TLS требованиям спецификаций с использованием методов комбинаторного и фаззинг тестирования. Проведение сравнений по скорости и покрытию разных решений для тестирования реализаций протокола TLS.

Руководитель выпускной квалификационной работы

к.ф.-м.н, доцент, доцент КБ

должность, место работы



подпись

С.И. Самохина

И.О. Фамилия

Задание принял к исполнению

студент группы № 931724

должность, место работы



подпись

В.О. Сапрыкин

И.О. Фамилия

АННОТАЦИЯ

Дипломная работа содержит 35 страниц, 6 рисунков, 5 таблиц, 2 алгоритма, 1 листинг и 17 литературных материалов.

ПРОТОКОЛ TLS, АБСОЛЮТНЫЕ ТРЕБОВАНИЯ СПЕЦИФИКАЦИЙ, ФАЗЗИНГ НА ОСНОВЕ ПОКРЫТИЯ КОДА, КОМБИНАТОРНОЕ ТЕСТИРОВАНИЕ, Т-ПОКРЫВАЮЩЕЕ ТЕСТИРОВАНИЕ, ТЕСТИРОВАНИЕ СЕРОГО ЯЩИКА.

Объекты исследования: протокол TLS, фаззинг на основе покрытия кода, комбинаторное тестирование.

Цель работы: разработка программного средства тестирования для проверки соответствия реализаций TLS требованиям спецификации протокола с использованием комбинаторного и фаззинг тестирования на основе покрытия кода.

Результат: разработано программное средство тестирования реализаций протокола TLS с использованием комбинаторного и фаззинг тестирования на основе покрытия кода. Реализован механизм создания тестовых шаблонов для автоматической генерации множества тестовых случаев. Реализован механизм систематического ограничения модели входных параметров тестовых шаблонов «разумными» значениями. Создано множество тестовых шаблонов, каждый из которых проверяет требование на основе RFC.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ.....	6
1 Протокол Transport Layer Security (TLS).....	9
1.1 TLS 1.2.....	9
1.2 TLS 1.3.....	10
1.3 Протокол Alert.....	11
1.4 Тестирование реализаций протокола.....	11
2 Модель входных параметров.....	14
2.1 Выбор параметров.....	15
3 Генерация тестовых случаев.....	17
3.1 t-покрывающее тестирование.....	17
3.2 Фаззинг-тестирование серого ящика.....	20
3.3 Комбинирование t-покрывающего тестирования и фаззинг-тестирования.....	22
4 Тестовый шаблон.....	25
5 Абсолютные требования RFC.....	28
6 Этапы тестирования.....	30
7 Скорость, покрытие, сравнение с другими решениями.....	32
ЗАКЛЮЧЕНИЕ.....	33
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ И ЛИТЕРАТУРЫ.....	34

ВВЕДЕНИЕ

Протокол Transport Layer Security (TLS) является одним из наиболее широко используемых протоколов безопасности. TLS поддерживает зашифрованную передачу данных с проверкой целостности, а также взаимную аутентификацию сторон, участвующих в общении. Протокол TLS был выбран в качестве объекта исследования для данной работы.

В настоящее время рекомендуемыми стандартами TLS являются версии 1.2 и 1.3 [1, 2]. Помимо основных стандартов, существует множество сопутствующих RFC, определяющих расширения TLS и дополнительные криптографические алгоритмы для протокола. В спецификации TLS 1.3 особое внимание уделяется утверждениям с использованием терминологии для абсолютных требований из RFC 2119 [3]. Такие требования выражаются ключевыми словами MUST, SHALL или REQUIRED. Спецификация TLS 1.3 содержит почти в два раза больше требований, чем спецификация предшествующей версии протокола. Требования предписывают конкретное поведение протокола, например, определение потоков протокола, обработка заголовков уровня записей или точных сообщений протокола Alert. Эти требования выходят за рамки функциональных аспектов. Несоответствие абсолютным требованиям может привести к проблемам совместимости и к критическим ошибкам безопасности.

Большое количество версий, расширений и криптографических алгоритмов увеличивает сложность TLS и приводит к различным ошибкам в реализациях протокола. Требования обратной совместимости подразумевают, что библиотеки TLS должны поддерживать несколько версий протокола, начиная с TLS 1.0, а также устаревшие криптографические алгоритмы, такие как 3DES. Сложность протокола и требование обратной совместимости привели к нескольким критическим атакам. Например, в 2018 году было обнаружено, что атака Блейхенбахера [4] (на тот момент ей было 20 лет) все еще широко распространена среди популярных реализаций TLS. Более того, многие атаки могли быть реализованы только при определенных комбинациях параметров, которые затрагивают разные части кода. Также было показано, что иногда уязвимости CBC padding oracle проявляются только для конкретных значений параметров протокола [5]. Например, замена алгоритма обмена ключами в наборе криптоалгоритмов уже может изменить порядок выполнения кода в достаточной степени, чтобы выявить или скрыть уязвимость. Таким образом, несмотря на огромное количество исследований и тщательную

спецификацию протокола, реализации TLS по-прежнему уязвимы для атак с использованием сложных комбинаций параметров.

Тестирование большинства реализаций TLS обычно ограничивается функциональным тестированием, которое заключается в проверке корректности завершения рукопожатия при использовании стандартных наборов криптоалгоритмов. Однако задача становится сложнее, если необходимо тестировать абсолютные требования безопасности, поскольку они должны выполняться для всех комбинаций значений входных параметров протокола. Большое количество возможных значений приводит к тому, что для тщательного покрытия необходимо экспоненциальное количество тестов. В этом случае автоматизация может быть достигнута путем определения абстрактных тестовых шаблонов, из которых можно автоматически генерировать множество тестовых случаев. Таким образом, шаблон может определить правильное поведение протокола для множества комбинаций входных параметров. Разработка таких тестовых шаблонов может быть далеко не тривиальной задачей, поскольку каждый шаблон требует дополнительного анализа для определения набора параметров, которые могут быть параметризованы. Даже при наличии таких тестовых шаблонов тестирование всех комбинаций параметров все еще невозможно, поскольку количество тестовых случаев растет экспоненциально. Одним из способов уменьшить количество возможных комбинаций значений параметров является метод комбинаторного тестирования, называемый *t*-покрывающим тестированием. В основе метода лежит утверждение о том, что большинство ошибок и сбоев программного обеспечения вызвано одним или двумя параметрами. Причем количество ошибок уменьшается после трех параметров.

Стандартный подход при использовании комбинаторного тестирования предполагает, что возможно применение всех различных комбинаций значений параметров. Однако в таком протоколе, как TLS, не все потенциальные значения параметров уместны для конкретного теста. Например, тест, который анализирует вычисления на эллиптических кривых, должен ограничивать свой выбор криптоалгоритмами, которые поддерживают эллиптические кривые. Причем взаимодействие значений параметров может быстро стать сложным в зависимости от контекста теста. Например, рассмотрим тест для проверки подписей, сгенерированных сервером. В этом случае существует взаимодействие между выбранным набором шифров, алгоритмом подписи и сертификатом сервера, что в конечном итоге влияет на достоверность подписи. Выбранная комбинация этих трех параметров может быть недействительной независимо от значения подписи. Если эти зависимости

проигнорировать, результат теста для конкретной реализации будет соответствовать не проверке цифровой подписи, а неверному выбору параметров, что будет являться ложным срабатыванием.

Цель работы – разработка программного средства тестирования для проверки соответствия реализаций TLS абсолютным требованиям спецификаций протокола.

Для достижения данной цели были поставлены следующие задачи:

1. Реализация механизма создания тестового шаблона, который можно параметризовать для автоматической генерации множества тестовых случаев с использованием t-покрывающего комбинаторного тестирования. Использование t-покрывающего тестирования позволит свести к минимуму количество тестовых случаев, при этом покрывая все t-комбинации параметров.

2. Реализация механизма систематического ограничения модели входных параметров тестовых шаблонов «разумными» значениями. Механизм необходим для сведения к минимуму количества ложных срабатываний.

3. Создание множества тестовых шаблонов, каждый из которых проверяет требование на основе RFC.

1 **Протокол Transport Layer Security (TLS)**

Протокол TLS позволяет двум взаимодействующим узлам устанавливать безопасный канал обмена данными. С этой целью узлы выполняют рукопожатие TLS для согласования криптографических параметров, определенных в так называемых наборах шифров (например, TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256) и получают сеансовые ключи TLS. Затем согласованные параметры и ключи используются при общении для обеспечения конфиденциальности, целостности и подлинности сообщений, которыми обмениваются собеседники. Существует две широко используемые и рекомендуемые версии TLS: 1.2 и 1.3.

1.1 TLS 1.2

Рукопожатие TLS 1.2 с использованием алгоритма обмена ключами Диффи-Хеллмана и аутентификацией сервера требует двух циклов. Схема рукопожатия для данной версии протокола представлена на рисунке 1. Клиент начинает рукопожатие с сообщения ClientHello, которое содержит версию TLS, список предлагаемых наборов шифров TLS, именованные группы для алгоритма обмена ключами и расширения. В ответ сервер посылает четыре сообщения. В сообщении ServerHello сервер выбирает версию TLS и криптографические параметры из предложенных клиентом. Затем сервер отправляет сообщение Certificate, содержащее цепочку сертификатов X.509 с открытым ключом сервера. Сервер использует свой закрытый ключ для аутентификации сгенерированного эфемерного открытого ключа Диффи-Хеллмана и отправляет его в сообщении ServerKeyExchange. Наконец, сервер отправляет сообщение ServerHelloDone, указывающее на завершение цепочки сообщений. Клиент продолжает рукопожатие сообщением ClientKeyExchange, которое содержит публичные данные, относящиеся к процессу обмена ключами Диффи-Хеллмана. После этого обе стороны могут вычислить предварительный секрет, который используется для получения всех криптографических ключей. Затем клиент отправляет сообщение ChangeCipherSpec, чтобы уведомить сервер о переходе в зашифрованное состояние, и отправляет сообщение Finished. Сервер завершает рукопожатие, отправляя свои собственные сообщения ChangeCipherSpec и Finished.

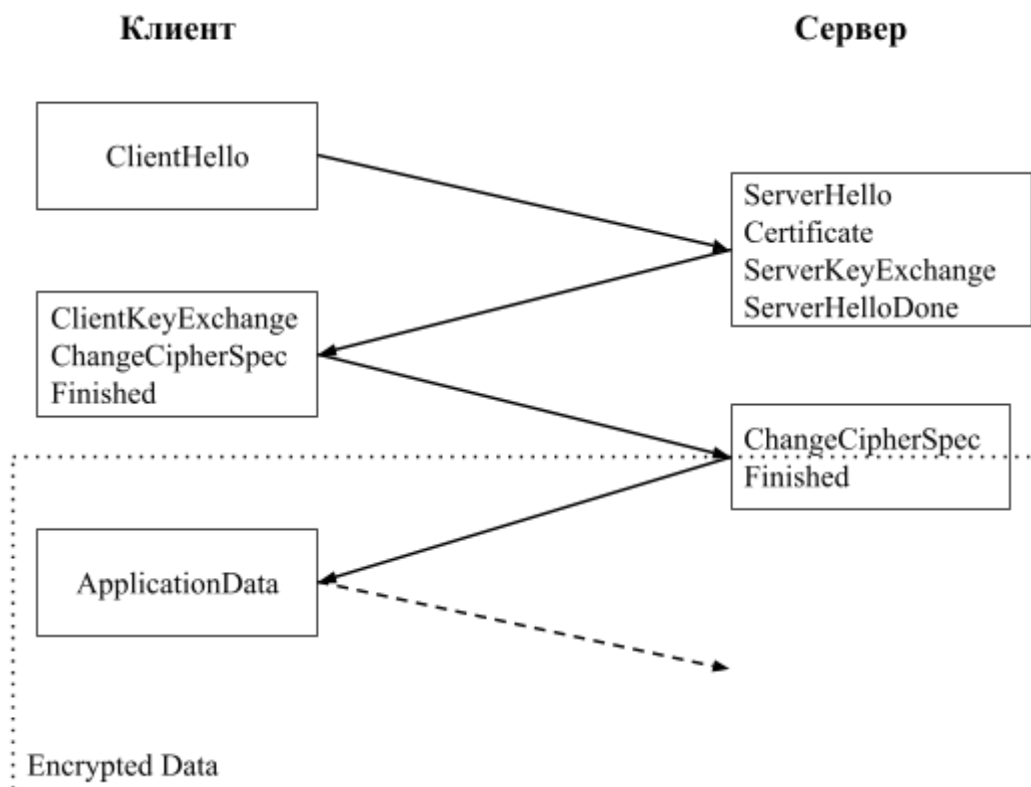


Рисунок 1 – Схема рукопожатия TLS 1.2

1.2 TLS 1.3

В отличие от TLS 1.2, для установления криптографических ключей требуется меньшее количество шагов. Это достигается за счет изменения порядка сообщений и добавления новых расширений. Схема рукопожатия для данной версии протокола представлена на рисунке 2. TLS 1.3 согласовывает общий секрет сообщениями **ClientHello** и **ServerHello**, которые содержат параметры протокола вместе с публичными данными для алгоритма обмена ключами Диффи-Хеллмана. Затем обе стороны вычисляют дополнительные секреты и начинают шифровать последующие сообщения. Сервер отправляет сообщение **EncryptedExtensions** (сообщение содержит дополнительные параметры в зашифрованном виде), **Certificate** и **CertificateVerify**. Сообщение **CertificateVerify** содержит подпись ранее отправленных сообщений, которую можно проверить с помощью открытого ключа из сертификата сервера. Последнее сообщение, отправляемое сервером при рукопожатии, – это сообщение **Finished**, аналогичное сообщению **Finished** в TLS 1.2. Клиент также отправляет сообщение **Finished** для завершения рукопожатия. Таким образом, узлы при первой возможности переходят к

зашифрованному общению, и практически все сообщения в Handshake получаются зашифрованными.

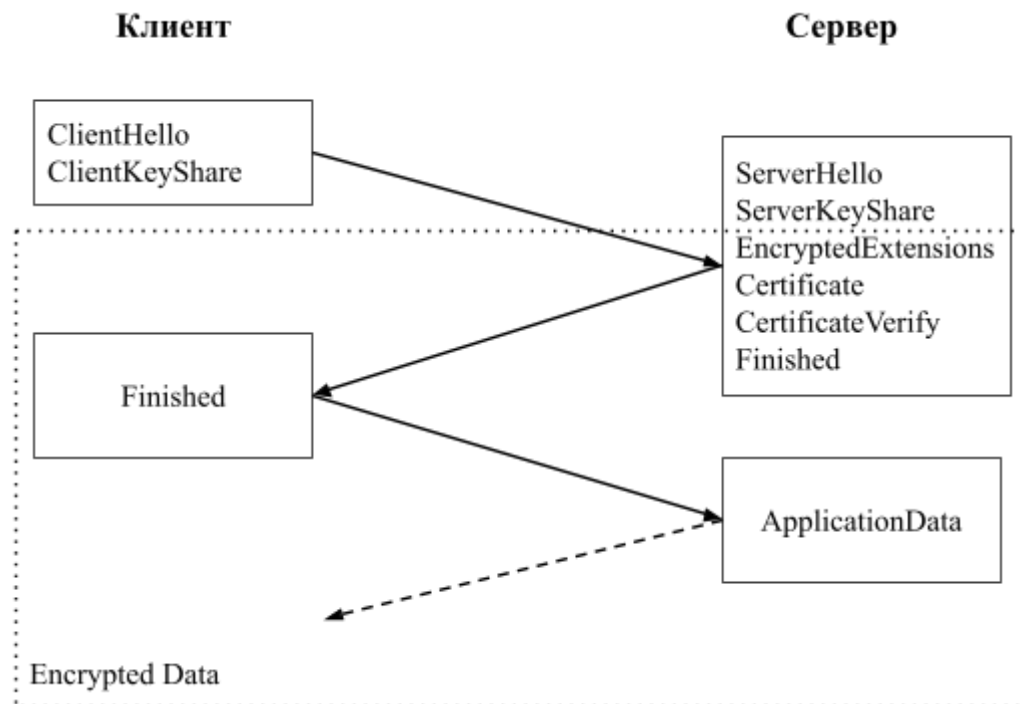


Рисунок 2 – Схема рукопожатия TLS 1.3

В TLS 1.3 были удалены устаревшие и небезопасные криптографические примитивы, которые присутствовали в предыдущих версиях:

- 1) Алгоритмы шифрования DES, 3DES и блочный шифр AES в режиме CBC;
- 2) Алгоритмы хеширования MD5 и SHA-1.

1.3 Протокол Alert

В протоколе TLS есть механизм, оповещающий узлы о возникших ошибках, – протокол Alert. Спецификации TLS определяют более 30 различных предупреждений и способы их использования в конкретных случаях. Например, оповещение Close Notify уведомляет узел о закрытии соединения. Bad Record MAC информирует узел о том, что полученное сообщение содержит недопустимый код аутентификации. Правильная обработка сообщений Alert имеет решающее значение для безопасности реализаций TLS, поскольку они могут предоставить информацию потенциальному злоумышленнику, которая может быть использована для атак на протокол.

1.4 Тестирование реализаций протокола

В данной работе речь пойдет непосредственно о тестировании абсолютных требований спецификаций в реализациях протокола TLS. Требования предписывают конкретное поведение протокола и выходят за рамки сугубо функциональных требований. Выражаются ключевыми словами `MUST`, `SHALL` или `REQUIRED`, используя терминологию из RFC 2119.

Существует множество инструментов для тестирования реализаций протокола. Рассмотрим наиболее популярные инструменты:

1. `tlsfuzzer` [6] – утилита на языке Python. Основой для тестирования является набор тестовых случаев (генерирующих последовательности сообщений протокола), которые проверяют правильную обработку ошибок в реализации протокола. Среди этих тестовых примеров есть явные требования RFC, однако основной целью является функциональное тестирование реализаций. Методы фаззинга используются при изменении входных сообщений. В отличие от обычных фаззеров, он проверяет не только то, что тестируемая система не дала сбой на программном уровне, но и правильность возвращаемых сообщений об ошибках.

2. `TLS-Attacker` [7] – фреймворк на основе Java для анализа библиотек TLS. Инструмент способен генерировать и отправлять сообщения протокола в произвольном порядке. Помимо тестирования на уязвимость к известным криптографическим атакам, он позволяет оценить поведение TLS-сервера с помощью метода фаззинга, основанного на динамических изменениях полей сообщений. *TLS-Attacker* обнаруживает проблемы реализации с помощью инструментации программы и анализатора сообщений TLS, который проверяет правильность выходных сообщений протокола.

Данные инструменты в исходном виде обладают наборами тестов, которые способны обеспечить высокое покрытие кода при тестировании реализаций. Также они позволяют создавать свои тестовые случаи с конкретными параметрами и тестировать известные атаки на протокол. Стоит отметить, что тестирование абсолютных требований спецификаций не является приоритетной целью данных инструментов, и они больше подходят для тестирования функциональных требований реализаций протокола TLS. При использовании данных утилит тестировщик способен конструировать и наблюдать за конкретными потоками протокола, однако не способен генерировать множества тестовых случаев с разными комбинациями значений параметров (например необходимо

тестировать абсолютные требования, которые должны выполняться для всех комбинаций значений входных параметров) и при этом сохранять правильность тестового оракула. Например, некоторые тесты могут быть несовместимы с конкретной реализацией протокола (отсутствие поддержки конкретных версий протокола, криптоалгоритмов и т.д.) или параметры протокола, указанные в тесте, несовместимы друг с другом. Такие тесты будут вызывать ложные срабатывания, что приводит к неправильной оценке реализации.

Помимо представленных утилит, существует несколько инструментов [8, 9], способных генерировать множество тестовых случаев с использованием шаблонов и методов комбинаторного тестирования (для сокращения общего количества комбинаций). Данные решения определяют правильное поведение протокола за счет сравнения результатов работы тестируемой реализации и эталонной реализации. Таким образом, каждое отклонение от эталонной реализации рассматривается как недостаток. Однако такой подход приводит к очень высокому уровню ложных срабатываний, поскольку в TLS часто бывает несколько действительных ответов на определенный ввод.

2 Модель входных параметров

При создании модели входных параметров для комбинаторного тестирования необходимо выбрать конкретный набор значений для каждого параметра в тестовом шаблоне. Для параметров, которые имеют только ограниченное количество значений, можно рассматривать все значения, в то время как для параметров с потенциально большим количеством различных значений необходимо выбирать отдельные интересные значения. Например, параметр может отвечать за добавление определенного расширения и, соответственно, принимать только два значения. Здесь оба возможных значения «истина» и «ложь» могут рассматриваться как значения параметра. Однако могут быть параметры, количество значений которых вызывает комбинаторный взрыв. Так что необходимо выбрать ограниченное число параметров, вызывающих ошибку с наибольшей вероятностью.

Для конкретного параметра все возможные значения должны иметь одинаковую семантику. Например, они либо все действительны, либо недействительны в контексте тестируемого требования. Однако некоторые комбинации значений параметров могут изменить семантику другого значения параметра. Например, в TLS подпись RSA можно использовать только в том случае, если в данном сеансе используется сертификат RSA. Поэтому необходимо тщательно моделировать ограничения значений параметров и исключать комбинации параметров, в которых взаимодействие конкретных значений изменяет их семантику.

Реализации обычно содержат не все возможные функции протокола, а только их подмножество. В контексте тестового шаблона некоторое значение параметра может быть действительным, однако в рамках конкретной реализации этот параметр может быть недопустим, поскольку соответствующий функционал не поддерживается. Таким образом, модель входных параметров необходимо ограничивать с учетом особенностей тестируемой реализации. В данной работе для определения поддерживаемых функций тестируемой реализации используется утилита SSLyze. Полученная информация используется для ограничения набора значений параметров в модели входных параметров.

В данной работе реализованы две модели входных параметров: EmptyModel (без параметров), DefaultModel (содержит все базовые параметры). Модели определены внутри классов и содержат список параметров. Для каждого параметра (кроме логических параметров) реализован свой собственный класс. В классе содержатся все возможные

значения параметра, а также методы, позволяющие фильтровать множество его значений. Например, значения параметра CipherSuite можно фильтровать по режиму шифрования, алгоритму хэширования, поддержке в TLS 1.3 и многим другим признакам. Фильтрация значений параметра производится внутри функции шаблона. Модели входных параметров можно дополнять в контексте тестового шаблона. Например, параметр AppDataLength (длина сообщения типа ApplicationData) не входит в базовые параметры, однако необходим для тестирования требований, связанных с проверкой сервером длины сообщения. После определения модели входных параметров, добавления дополнительных параметров и ограничения значений параметров, к модели применяются функция find_semantic_faults, которая позволяют определить список комбинаций значений параметров, которые являются невалидными для протокола TLS. Модель входных параметров и список невалидных комбинаций подается на вход алгоритму комбинаторного тестирования для определения набора тестовых случаев.

2.1 Выбор параметров

Параметрами могут быть простые свойства, такие как набор шифров или добавление необязательного расширения. Но также параметром может выступать определенная позиция битов, которые делают недействительным код аутентификации сообщения. К базовым логическим параметрам, определяющим добавление расширений, относятся: TcpFragmentation, SessionTicket, ExtendedMasterSecret, EncryptThenMac, ALPN, GreaseCipherSuites, GreaseNamedGroups, GreaseSigAlgs, Heartbeat, RenegotiationIndication, PskKeyExchModes, SendLegacyCCS. В таблице 1 перечислены базовые параметры, принимающие более двух возможных значений. В таблице 2 представлены дополнительные параметры.

Таблица 1 – Базовые параметры, содержащие более двух возможных значений

Параметр	Кол-во возможных значений
CipherSuite	3 - 44
NamedGroup	4 - 18

Таблица 2 – Дополнительные параметры. Параметры с приставкой Change необходимы для генерации тестовых случаев с невалидными значениями (ошибками в различных байтах строки) параметров протокола

Параметр	Кол-во возможных значений
AlertDescription	34
ChangeByte	1 - 48
ChangeBitInByte	8
SelectGreaseCipherSuite	16
SelectGreaseExtension	16
SelectGreaseNamedGroup	16
SelectGreaseProtocolVersion	16
SelectGreaseSignatureAlgo	16
ProtocolMessageType	6

3 Генерация тестовых случаев

3.1 t-покрывающее тестирование

Комбинаторное тестирование является широко известным методом выбора комбинаций из набора параметров. Тесты разрабатываются таким образом, чтобы один и тот же тест мог быть запущен с разными значениями этих параметров. Эти тесты называются параметризованными тестами или тестовыми шаблонами.

Одним из способов уменьшить количество возможных комбинаций параметров является разновидность комбинаторного тестирования, называемая t-покрывающим тестированием. При тестировании используются все возможные комбинации значений параметров длины t. Если n – количество тестовых параметров и $t < n$, то количество тестов можно уменьшить, включив два или более t-наборов в один и тот же тестовый вектор. Если $t \ll n$, это приводит к значительному сокращению количества тестов.

Некоторые ошибки при тестировании могут возникать только из-за взаимодействия различных входных параметров, поэтому их можно обнаружить только путем выполнения тестов, охватывающих все комбинации параметров. На практике это часто неосуществимо, так как сложное программное обеспечение может иметь множество параметров с разными значениями, что приводит к экспоненциальному росту генерируемых тестовых случаев. Однако исследования в области тестирования программного обеспечения показали, что большинство наблюдаемых ошибок являются не результатом взаимодействия всех возможных параметров, а скорее их небольшого подмножества [10]. Это привело к концепции t-покрывающего тестирования. Тестирование охватывает все комбинации значений параметров для каждого подмножества из t параметров. Все возможные тестовые параметры и их значения заранее определены в так называемой модели входных параметров. Создание идеального набора тестов, т.е. минимального набора тестов, необходимого для охвата всех взаимодействий t-переменных является сложной задачей. Тем не менее современные алгоритмы могут сгенерировать такой набор тестов за короткое время. Чтобы избежать создания бессмысленных тестовых входных данных и ускорить создание набора тестов, многие алгоритмы комбинаторного тестирования предоставляют возможность вручную определять ограничения для конкретных комбинаций значений параметров. Если тестовые входы выбраны тщательно, t-покрывающее тестирование даже с небольшим значением t

может обеспечить высокую скорость обнаружения ошибок при сохранении сравнительно небольшого набора тестов [11].

В данной работе для генерации тестовых случаев из тестового шаблона используется алгоритм IPOG (In-Parameter-Order-General) [12], так как он подходит для эффективной выработки комбинаций с $t > 2$. В данной работе используется реализация алгоритма IPOG из библиотеки CTLog, написанная на языке Python. Алгоритм IPOG может быть описан следующим образом: для системы с t или более параметрами строится тестовый набор из комбинаций значений длины t первых t параметров. Тестовый набор расширяется для первых $t + 1$ параметров, а затем продолжает расширяться, пока не добавит все параметры. Расширение существующего набора дополнительным параметром выполняется в три шага:

1. Вычисляется набор p комбинаций длины t из значений нового параметра и значений $t - 1$ параметров среди предыдущих параметров.
2. Горизонтальное расширение — дополняет каждый существующий тест, добавляя одно значение нового параметра, причем значение выбирается таким образом, чтобы тест содержал наибольшее количество комбинаций из p . Данные комбинации удаляются из p .
3. Вертикальное расширение — если набор p не пустой, тесты изменяются для добавления комбинаций из p (если возможно), или добавляются новые тесты, чтобы покрыть оставшиеся комбинации.

Далее представлен полный алгоритм IPOG для генерации тестовых случаев, охватывающих все комбинации t -переменных.

Алгоритм 1. Алгоритм IPOG.

Входные данные: сложность t и набор параметров ps .

- 1) Подготовка:
 - a) Инициализация пустого тестового набора ts ;
 - b) Обозначим параметры в ps в произвольном порядке как $P_1, P_2, \dots, P_n$;
 - c) Добавить в тестовый набор ts тест для каждой комбинации значений первых t параметров.
- 2) Для каждого P_i из множества ps , не считая первые t параметров,

выполняется:

- a) пусть π – множество комбинаций значений длины t , включающих параметр P_i и $t - 1$ параметр среди первых $i - 1$ параметров ps ;
- b) горизонтальное расширение для параметра P_i : для каждого теста $\tau = (v_1, v_2, \dots, v_{i-1})$ в наборе тестов ts выполняется:
 - i) выбрать значение v_i параметра P_i и заменить τ на $\tau^* = (v_1, v_2, \dots, v_{i-1}, v_i)$ так, чтобы τ^* покрывало наибольшее количество комбинаций значений в π ;
 - ii) удалить из π комбинации значений, охватываемых τ^* .
- c) вертикальное расширение для параметра P_i : для каждой комбинации σ в множестве π выполняется:
 - i) если существует тест, который уже покрывает σ , то удалить σ из π , иначе изменить существующий тест, если это возможно, или добавить новый тест, чтобы покрыть σ и удалить его из π .

На выходе получаем тестовый набор ts , покрывающий все комбинации значений параметров длины t .

На рисунке 3 приведен пример работы Алгоритма 1. Пример рассматривает систему из четырех параметров – P1, P2, P3 и P4, где P1, P2, P3 имеют два значения - 0 и 1, а P4 имеет три значения - 0, 1 и 2. В пункте А перечислены все возможные комбинации значений первых трех параметров. Пункт В демонстрирует результат горизонтального расширения для параметра P4. Например, четвертая строка расширилась значением 0, тем самым добавив в тест три новые комбинации из π : (P1.0, P2.1, P4.0), (P1.0, P3.1, P4.0), (P2.1, P3.1, P4.0). Пункт С демонстрирует вертикальное расширение для параметра P4. Например, после горизонтального расширения комбинация (P1.1, P2.0, P4.0) осталась непокрытой. Невозможно найти существующий тест, который можно было бы изменить, чтобы охватить эту комбинацию. Таким образом, создается новый тест (P1.1, P2.0, P3.-, P4.0) (тест номер 9), чтобы покрыть эту комбинацию. Также стоит заметить, что (P2.0, P3.1, P4.0) – это еще одна комбинация, которая не была покрыта после горизонтального расширения. Эту комбинацию можно покрыть, изменив значение P3 с «-» на 1 в 9-м тесте.

P1 P2 P3	P1 P2 P3 P4	P1 P2 P3 P4
0 0 0	0 0 0 0	0 0 0 0
0 0 1	0 0 1 1	0 0 1 1
0 1 0	0 1 0 2	0 1 0 2
0 1 1	0 1 1 0	0 1 1 0
1 0 0	1 0 0 1	1 0 0 1
1 0 1	1 0 1 2	1 0 1 2
1 1 0	1 1 0 0	1 1 0 0
1 1 1	1 1 1 1	1 1 1 1
		1 0 1 0
		0 1 0 1
		0 0 1 2
		1 1 0 2
		- 0 0 2
		- 1 1 2

Рисунок 3 – Пример работы Алгоритма 1. Символ «-» обозначает, что на данном месте может быть любое значение параметра

3.2 Фаззинг-тестирование серого ящика

Существует три основные категории фаззинга в зависимости от степени использования внутренней структуры программы:

- 1) Фаззинг чёрного ящика – требует только выполнения программы, информация об исполняемом файле или программе неизвестны;
- 2) Фаззинг белого ящика – для тестирования требуется анализ программы и инструментация исполняемого файла;
- 3) Фаззинг серого ящика – легковесная инструментация исполняемого файла для сбора информации о отклике программы.

В качестве отклика программы используется множество метрик. Одной из таких метрик является покрытие кода. Метрики покрытия кода отслеживают суммарное количество выполненных строк кода, базовых блоков, количество сделанных условных переходов. Задача фаззера — генерировать данные, которые приводят к увеличению покрытия кода. Одним из наиболее популярных фаззеров серого ящика на основе покрытия кода является American Fuzzy Lop (AFL). AFL был создан в 2013 году и стал

главным двигателем прогресса в современном фаззинге. Схема работы фаззера AFL представлена на рисунке 4.



Рисунок 4 – Схема работы фаззера серого ящика на основе покрытия кода (AFL)

Далее представлен алгоритм работы фаззера AFL в общем случае.

Алгоритм 2. Алгоритм работы AFL – фаззера серого ящика на основе покрытия кода.

Входные данные: корпус S с входными данными программы.

- 1) Подготовка:
 - a) $T_x = \emptyset$;
 - b) $T = S$.
- 2) Если $T = \emptyset$, то добавляем в T пустую строку;
- 3) Повторяем до полной остановки фаззера:
 - a) $t = ChooseNext(T)$ – выбираем следующее входное значения t из корпуса T ;
 - b) $p = AssignEnergy(t)$ – вычисление значения p - кол-во раз, когда на основе t будут сгенерированы новые входные значения;
 - c) Выполняем p раз:
 - i) $t^* = MutateInput(t)$ – мутация значения t ;
 - ii) Если t^* вызвало ошибку, приведшую к завершению программы, то t^* добавляется в корпус T_x ;
 - iii) Если t^* увеличило покрытие кода, то оно добавляется в корпус T .

На выходе получаем корпус T_X — входные данные, приводившие к завершению программы с ошибкой.

В AFL значение p вычисляется на основе информации о времени выполнения программы и покрытии кода при использовании входного значения t . Методы, применяемые в AFL при мутации входных значений: инверсия определенных битов, замена байтов на граничные значения (0 и 255), удаление и замена блоков байтов.

3.3 Комбинирование t-покрывающего тестирования и фаззинг-тестирования

В данной работе для генерации тестовых случаев, помимо методов комбинаторного тестирования, используется фаззинг-тестирование на основе покрытия кода. Фаззинг используется в комбинации с t-покрывающим тестированием для шаблонов, в которых необходима генерация тестовых случаев с заведомо невалидными параметрами протокола. В качестве фаззера используется модернизированная реализация AFLNet, которая, в свою очередь, является расширением фаззера AFL.

Например, необходимо проверить, что сервер возвращает правильное сообщение Alert при ошибке в коде аутентификации, который содержится в сообщении клиента. Рассмотрим данный случай в рамках версии протокола 1.3. Сообщение клиента с ошибкой будет послано сразу после успешного рукопожатия узлов. Структура данного сообщения представлена на рисунке 5.

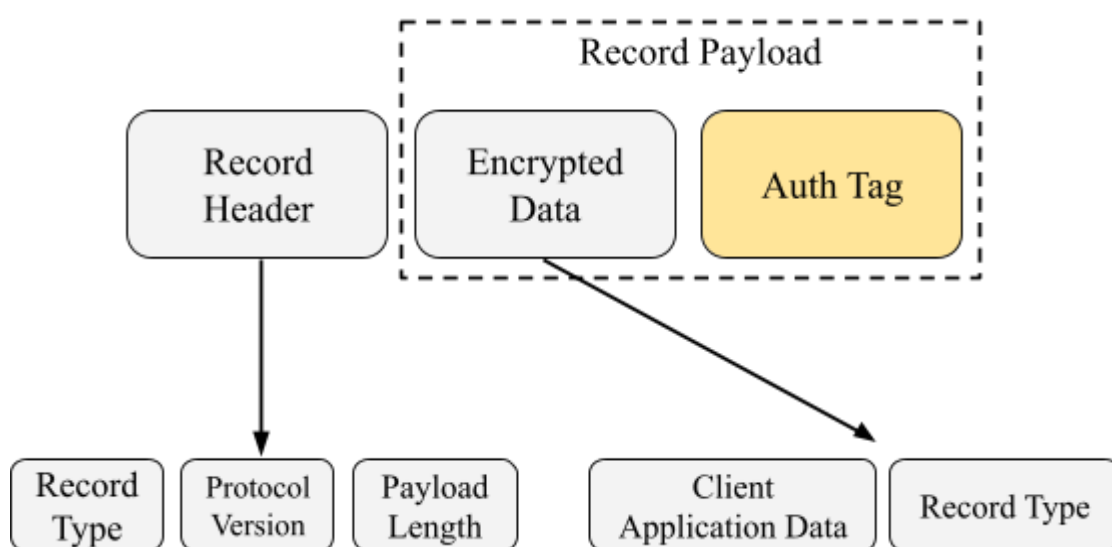


Рисунок 5 – Структура пакета Application Data

Процесс генерации и выполнения тестовых случаев для подобного шаблона состоит из следующих пунктов:

1. Генерация тестовых случаев с использованием t-покрывающего тестирования. Для генерации потоков протокола с невалидным кодом аутентификации используется параметр ChangeByte. Значение данного параметра определяет идентификатор байта кода аутентификации, в котором будет допущена ошибка. Таким образом, для каждой комбинации параметров (из набора определенным алгоритмом IPOG) будет определен поток протокола, содержащий ошибку в определенном байте кода аутентификации.

2. Выполнение тестовых случаев.

3. Валидация с использованием тестового оракула (определён шаблоном), который оценивает, было ли выполнение тестового случая успешным или нет. Результаты каждого тестового случая сохраняются для формирования отчета по тестированию.

4. Тестовые случаи подаются на вход фаззера в качестве корпуса – набора репрезентативных входов, который использоваться для дальнейших мутаций.

5. Фаззер мутирует тестовые случаи (последовательности сообщений протокола) из корпуса, используя различные методы мутации для достижения максимального покрытия кода. Особенностью данного шага является то, что изменяются не все сообщения в последовательности (тестовом случае), а только сообщение, тестируемое в контексте шаблона. Причем мутации подвергается только код аутентификации.

6. Измененные последовательности отправляются серверу. Происходит валидация ответа сервера с использованием тестового оракула шаблона. В отчет по тестированию сохраняются только те тестовые случаи, которые не прошли валидацию.

7. Если фаззеру удастся сгенерировать последовательность сообщений, которая увеличила общее покрытие кода (относительно последовательностей из корпуса), то такая последовательность добавляется в исходный корпус. Изменение покрытия кода свидетельствует о том, что поведение реализации изменилось (при использовании определенной мутации) и дальнейшие мутации данной последовательности могут привести к нарушению требования (если еще не привели), заданного шаблоном. Последовательности сообщений, которые дают новое покрытие кода, являются приоритетными для фаззера.

Таким образом, фаззинг-тестирование дополняет набор тестовых случаев (сгенерированных t-покрывающим тестированием) с помощью различных методов

мутации выбранного параметра. Стоит отметить, что информацию о покрытии кода можно получить, только инструментируя исполняемый файл программы. Если такой возможности нет, то фаззинг в шаблонах игнорируется. Количество раундов мутаций по-умолчанию равно 10000, однако может быть задано в шаблоне тестировщиком.

4 Тестовый шаблон

Тестовый шаблон определяет сообщения, которые должны быть отправлены реализации и значения параметров (модель входных параметров) в зависимости от тестируемого требования. Для валидации наблюдаемого поведения сервера к шаблону прикрепляется функция проверки, которая реализует логику его тестового оракула. Поскольку шаблоны независимы, их можно запускать параллельно. Механизм создания шаблонов и моделей входных параметров реализован на языке Python.

Пример тестового шаблона для конкретного требования: шаблон проверяет, отправляет ли сервер правильное сообщение Alert, если сообщение клиента содержит неверный паддинг. Данный тест изменяет первое зашифрованное сообщение клиента (после рукопожатия) так, чтобы оно содержало неверный паддинг. Шаблон выполнится только в том случае, если реализация поддерживает блочные шифры в режиме CBC. В противном случае шаблон будет проигнорирован. Ошибка в паддинге вводится в несколько позиций (для каждого байта паддинга). Позиции для каждого тестового случая определяются комбинаторным тестированием. Шаблон для данного примера представлен в листинге 1. Рассмотрим его построчно:

Строка 1: Объявление тестового шаблона. Входные параметры: модель входных параметров для реализации (создается после получения поддерживаемых реализацией функций) и объект класса Runner, задающий последовательность сообщений протокола.

Строка 2-4: Текст требования из RFC 5246.

Строка 5-7: Создание модели входных параметров по-умолчанию и объединение её с моделью входных параметров реализации, ограничение значений параметра CipherSuite наборами, которые содержат блочные шифры в режиме CBC.

Строка 8: В качестве потока протокола выбирается Handshake (рукопожатие).

Строка 9: После рукопожатия добавляется новое действие - отправка сообщения типа ApplicationData.

Строка 10-14: Функция шаблона возвращает следующие значения: описание требования, модель входных параметров, объект потока протокола, информация о тестируемом параметре. Последнее значение состоит из пятёрки объектов: индекс изменяемого сообщения, часть сообщения, которую необходимо заменить, класс изменяемого параметра, значения которого будут подставляться в данное сообщение (если

такой класс есть, иначе подставляются случайные значения), необходимость фаззинга (bool), кол-во раундов мутаций.

Строка 16: Объявление тестового оракула для данного шаблона.

Строка 17: Извлечение последнего ответа сервера из списка ответов сервера - именно в нём должно содержаться необходимое нам сообщение протокола Alert.

Строка 18: Создание объекта класса TestResult, который будет содержать всю информацию о выполнении теста.

Строка 19-20, 25-26: Проверка того, что последнее сообщение сервера содержит сообщение типа Alert.

Строка 21-24: Проверка того, что сообщение содержит нужное предупреждение Alert: BAD_RECORD_MAC.

Строка 27: Тестовый оракул возвращает объект класса TestResult. Данный объект может содержать несколько результатов теста. В данном случае результат проверки типа сообщения из заголовка записи и проверки типа сообщения Alert (если первое условие выполнилось). Это обусловлено тем, что поведение реализации может не совпадать с требованиями RFC, однако отличаться лишь незначительно (например неправильный код ошибки). Такое поведение оценивается как спорное.

```
1 def cbc_padding_test(server_params_model, runner: Runner):
2     description = "Each uint8 in the padding data vector MUST be filled with" \
3         " the padding length value. The receiver MUST check this padding" \
4         " and MUST use the bad_record_mac alert to indicate padding error"
5     current_model = DefaultModel()
6     current_model = DefaultModel.intersect(server_params_model)
7     current_model.cipher_suite.is_cbc()
8     runner.gen_work_flow(Flows.Handshake)
9     runner.add_action(action.SendAction, ApplicationDataMsg)
10    return (
11        description,
12        current_model,
13        runner,
14        (-1, PADDING, ChangeByte, True, 5000))
15
16 def cbc_padding_test_oracle(runner: Runner):
```

```
17     alert = runner.recieved_msgs[-1]
18     result = TestResult(alert)
19     if alert.type == ALERT:
20         result.add(result_descriptions.ALERT_RECIEVED, True)
21         if alert.desc == types.BAD_RECORD_MAC:
22             result.add(result_descriptions.ALERT_DESCRIPTION, True)
23         else:
24             result.add(result_descriptions.ALERT_DESCRIPTION, False)
25     else:
26         result.add(result_descriptions.ALERT_RECIEVED, False)
27     return result
```

Листинг 1 – Пример тестового шаблона

5 Абсолютные требования RFC

В качестве основного источника для тестовых шаблонов использовались семь спецификаций (RFC) для TLS 1.2, 1.3 и их расширений. Полный список с кратким изложением областей применения отдельных RFC представлен в таблице 3.

Таблица 3 – Рассмотренные RFC и их краткое описание

RFC	Описание
5246	Стандарт TLS 1.2
8446	Стандарт TLS 1.3
6066 [13]	Новые расширения для TLS
8701 [14]	Задаёт набор констант GREASE, предназначенных для обеспечения взаимодействия между узлами с разными функциями протокола
8422 [15]	Определяет обработку наборов криптоалгоритмов с использованием эллиптических кривых для TLS 1.2 и более ранних версий
7919 [16]	Документ вводит группы ffdhe (Finite Field Diffie-Hellman Ephemeral) в регистр Supported Groups (поддерживаемые группы) параметров TLS. Введение данных групп позволяет узлам установить общие параметры алгоритма Диффи-Хеллмана для конечного поля, что позволяет устранить недостатки традиционного обмена ключами с использованием алгоритма Диффи-Хеллмана на основе конечного поля
7507 [17]	Определяет механизмы защиты от понижения версии

В представленных RFC особое внимание уделялось абсолютным требованиям (согласно RFC 2119), а именно требованиям с использованием терминов MUST, SHALL, REQUIRED или абсолютным запретам с использованием терминов MUST NOT или SHALL NOT. Были исключены утверждения, которые касаются исключительно будущих расширений протокола и не направлены на реализацию протокола. Также некоторые требования были объявлены устаревшими в новых RFC, поэтому они также были

исключены. Были исключены требования, которые невозможно проверить в рамках данного подхода. В основном это относится к требованиям, которые нельзя проверить без доступа к внутренним компонентам реализации. Например, требования, связанные со временем. Из общего списка требований были исключены все требования, связанные с сертификатами и проверкой сертификатов с модифицированными структурами и цепочками X.509.

После обработки всех RFC набор тестов состоит из 121 уникальных тестовых шаблонов.

6 Этапы тестирования

Схема тестирования реализаций протокола TLS представлена на рисунке 6. Рассмотрим все этапы тестирования:

- 1) Подготовка реализации и средства тестирования:
 - a) Добавление инструментации в исполняемый файл программы для сбора информации о покрытии кода. Пункт обязателен при необходимости расширения тестовых случаев методом фаззинг-тестирования;
 - b) Добавление новых тестовых шаблонов (если это необходимо). По-умолчанию, средство тестирования содержит 121 шаблон, которые покрывают требования из 7 разных спецификаций.
- 2) Определение поддерживаемых функций тестируемой реализации;
- 3) Формирование списка шаблонов, которые поддерживаются данной реализацией. Реализация может не поддерживать все функции протокола. Отбрасываются шаблоны, проверяющие недостающий функционал реализации;
- 4) Обработка шаблонов (подпункты ниже реализуются для всех шаблонов):
 - a) Ограничение модели входных параметров на основе доступных функций реализации и требований шаблона;
 - b) Генерация комбинаций параметров алгоритмом IPOG. Из результирующего набора исключаются комбинации, которые недопустимы для протокола;
 - c) Генерация и выполнение тестовых случаев на основе комбинаций параметров из предыдущего пункта;
 - d) Фаззинг-тестирование реализации с использованием множества тестовых случаев в качестве корпуса. Пункт опционален и неприменим без инструментации исполняемого файла программы;
 - e) Валидация результатов выполнения тестовых случаев с использованием тестового оракула из шаблона.
- 5) Формирование отчета, состоящего из результатов тестирования по каждому шаблону.

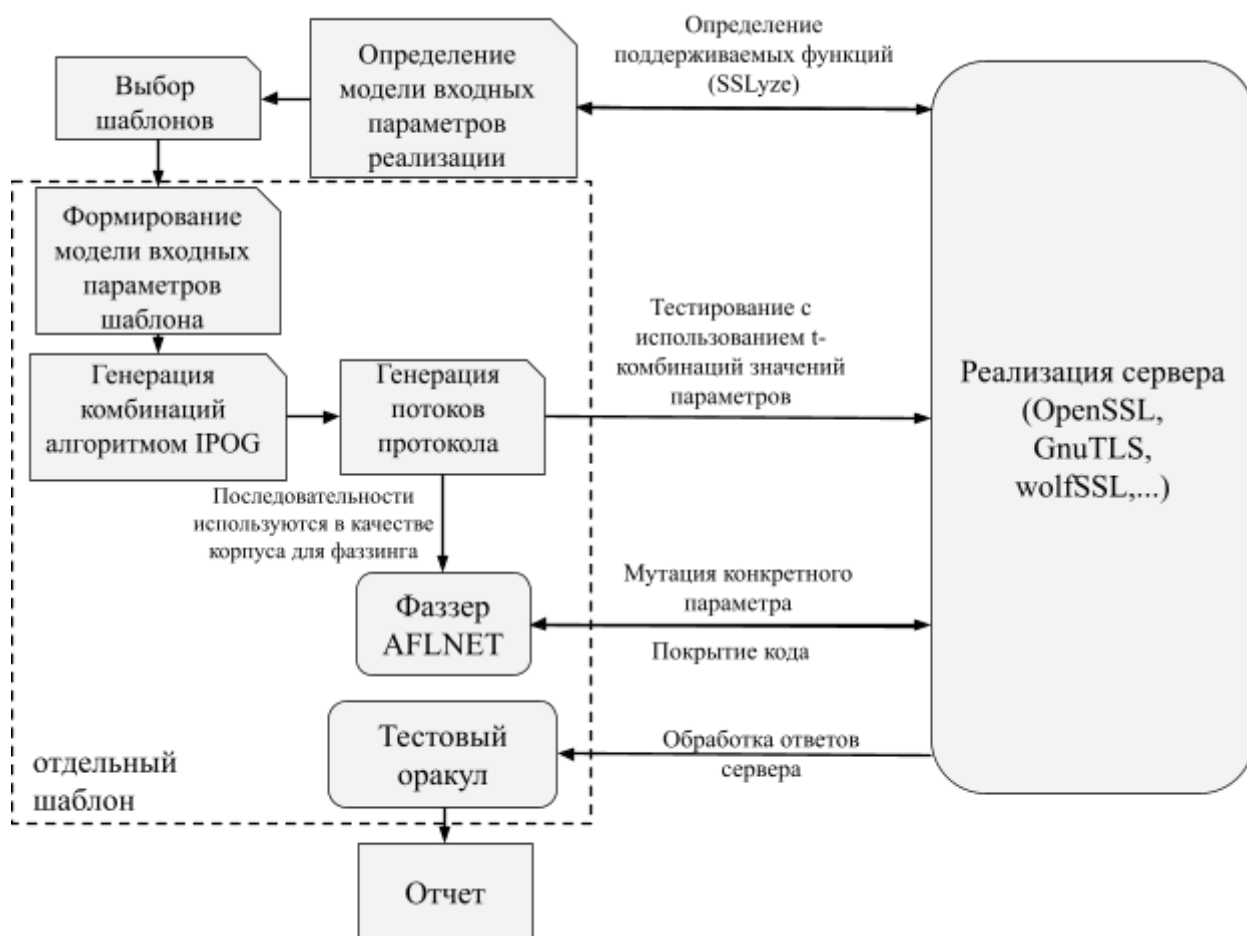


Рисунок 6 – Схема работы инструмента тестирования

7 Скорость, покрытие, сравнение с другими решениями

В таблице 4 представлено время выполнения полного тестирования для трех популярных реализаций протокола TLS: OpenSSL, GnuTLS и wolfSSL.

Таблица 4 – Время выполнения полного тестирования для популярных реализаций протокола TLS с разной силой t

	$t = 3$		$t = 2$		$t = 1$	
Реализация	Время выполнения, ч.		Время выполнения, ч.		Время выполнения, ч.	
	без фаззинга	с фаззингом	без фаззинга	с фаззингом	без фаззинга	с фаззингом
OpenSSL	19.3	27.6	4.2	12.7	0.7	9.3
GnuTLS	18.9	30.1	4	15.4	0.6	12.1
wolfSSL	31.3	39.1	6.9	14.6	1.7	9.3

Было проведено сравнение данной реализации с другими аналогичными решениями в таблице 5. Сравнение проводилось по данным о покрытии при тестирования реализации OpenSSL. Во время тестирования для сбора информации о покрытии кода использовалась утилита gsov. По результатам сравнения текущее решение незначительно уступило tlsfuzzer, хоть и ограничивается тестированием абсолютных требований спецификации.

Таблица 5 – Покрытие строк при использовании разных решений для тестирования реализации OpenSSL. Стоит учитывать, что OpenSSL, помимо реализации TLS, содержит множество различных криптографических элементов, которые недоступны при тестировании протокола

Решение	текущее решение	tlsfuzzer	TLS-Attacker
покрытие строк, %	16.7	16.8	15.3

ЗАКЛЮЧЕНИЕ

В рамках данной работы разработано средство тестирования реализаций протокола TLS с использованием методов комбинаторного и фаззинг тестирования на основе покрытия кода.

Реализован механизм создания тестовых шаблонов для автоматической генерации множества тестовых случаев. Реализован механизм систематического ограничения модели входных параметров тестовых шаблонов «разумными» значениями. Создано множество тестовых шаблонов, каждый из которых проверяет требование на основе RFC. Проведены сравнения по скорости и покрытию разных решений для тестирования реализаций протокола TLS.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ И ЛИТЕРАТУРЫ

1. T. Dierks, E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2, RFC 5246.
2. E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3, RFC 8446.
3. S. Bradner. Key words for use in RFCs to Indicate Requirement Levels, RFC 2119.
4. Новая атака представляет угрозу даже для TLS 1.3. — Электрон. дан. — URL: <https://xakep.ru/2019/02/11/new-bleichenbacher-attacks/> (дата обращения 18.01.2022).
5. Robert Merget, Juraj Somorovsky, Nimrod Aviram, Craig Young, Janis Fliegenschmidt, Jörg Schwenk, Yuval Shavitt. Scalable Scanning and Automatic Classification of TLS Padding Oracle Vulnerabilities // USENIX Security Symposium 2019.
6. H. Kario. TLSFuzzer: TLS Test Suite and Fuzzer. — URL <http://github.com/tomato42/tlsfuzzer> (дата обращения 18.01.2022).
7. J. Somorovsky. Systematic Fuzzing and Testing of TLS Libraries // 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS '16).
8. Bernhard Garn, Dimitris E. Simos, Feng Duan, Yu Lei, Josip Bozic, Franz Wotawa. Weighted Combinatorial Sequence Testing for the TLS Protocol // 2019 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW).
9. Dimitris E. Simos, Josip Bozic, Feng Duan, Bernhard Garn, Kristoffer Kleine, Yu Lei, Franz Wotawa. Testing TLS Using Combinatorial Methods and Execution Framework // Testing Software and Systems - 29th IFIP WG 6.1 International Conference, ICTSS 2017.
10. Zachary B. Ratliff, D. Richard Kuhn, Raghu N. Kacker, Yu Lei, Kishor S. Trivedi. The Relationship between Software Bug Type and Number of Factors Involved in Failures // 2016 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW).
11. D. Richard Kuhn, Raghu N. Kacker, Yu Lei. A Model for T-Way Fault Profile Evolution during Testing // 2017 IEEE International Conference on Software Testing, Verification and Validation Workshops, ICST Workshops 2017.
12. Yu Lei, Raghu Kacker, D. Richard Kuhn, Vadim Okun, James Lawrence. IPOG: A General Strategy for T-Way Software Testing // 14th Annual IEEE International Conference and Workshop on Engineering of Computer Based Systems (ECBS 2007).

13. D. Eastlake 3rd. Transport Layer Security (TLS) Extensions: Extension Definitions, RFC 6066.
14. D. Benjamin. Applying Generate Random Extensions And Sustain Extensibility (GREASE) to TLS Extensibility, RFC 8701.
15. Y. Nir, S. Josefsson, M. Pegourie-Gonnard. Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS) Versions 1.2 and Earlier, RFC 8422.
16. D. Gillmor. Negotiated Finite Field Diffie-Hellman Ephemeral Parameters for Transport Layer Security (TLS), RFC 7919.
17. B. Moeller, A. Langley. TLS Fallback Signaling Cipher Suite Value (SCSV) for Preventing Protocol Downgrade Attacks, RFC 7507.



АНТИПЛАГИАТ
ОБНАРУЖЕНИЕ ЗАИМСТВОВАНИЙ

Отчет о проверке на заимствования №1



Автор: Сапрыкин Валентин

Проверяющий: Сапрыкин Валентин

Отчет предоставлен сервисом «Антиплагиат» - <http://users.antiplagiat.ru>

ИНФОРМАЦИЯ О ДОКУМЕНТЕ

№ документа: 22
Начало загрузки: 20.01.2023 15:04:25
Длительность загрузки: 00:00:01
Имя исходного файла: ВКР 23 САПРЫКИН.pdf
Название документа: ВКР 23 САПРЫКИН
Размер текста: 48 кБ
Символов в тексте: 49293
Слов в тексте: 5728
Число предложений: 352

ИНФОРМАЦИЯ ОБ ОТЧЕТЕ

Начало проверки: 20.01.2023 12:04:27
Длительность проверки: 00:00:04
Комментарии: не указано
Модуль поиска: Интернет Free



СОВПАДЕНИЯ

1,64%

САМОЦИТИРОВАНИЯ

0%

ЦИТИРОВАНИЯ

0%

ОРИГИНАЛЬНОСТЬ

98,36%

Совпадения — доля всех найденных текстовых пересечений, за исключением тех, которые система отнесла к цитированиям, по отношению к общему объему документа.
Самоцитирования — доля фрагментов текста проверяемого документа, совпадающий или почти совпадающий с фрагментом текста источника, автором или соавтором которого является автор проверяемого документа, по отношению к общему объему документа.
Цитирования — доля текстовых пересечений, которые не являются авторскими, но система посчитала их использование корректным, по отношению к общему объему документа. Сюда относятся оформленные по ГОСТу цитаты; общеупотребительные выражения; фрагменты текста, найденные в источниках из коллекций нормативно-правовой документации.
Текстовое пересечение — фрагмент текста проверяемого документа, совпадающий или почти совпадающий с фрагментом текста источника.
Источник — документ, проиндексированный в системе и содержащийся в модуле поиска, по которому проводится проверка.
Оригинальность — доля фрагментов текста проверяемого документа, не обнаруженных ни в одном источнике, по которым шла проверка, по отношению к общему объему документа.
Совпадения, самоцитирования, цитирования и оригинальность являются отдельными показателями и в сумме дают 100%, что соответствует всему тексту проверяемого документа.
Обращаем Ваше внимание, что система находит текстовые пересечения проверяемого документа с проиндексированными в системе текстовыми источниками. При этом система является вспомогательным инструментом, определение корректности и правомерности заимствований или цитирований, а также авторства текстовых фрагментов проверяемого документа остается в компетенции проверяющего.

№	Доля в отчете	Источник	Актуален на	Модуль поиска
[01]	0,99%	https://datatracker.ietf.org/meeting/96/agenda/tls-drafts.pdf https://datatracker.ietf.org	23 Авг 2017	Интернет Free
[02]	0,15%	https://datatracker.ietf.org/doc/pdf/draft-ietf-tls-tls13-28 https://datatracker.ietf.org	08 Дек 2022	Интернет Free
[03]	0%	draft-ietf-tls-tls13-25 - The Transport Layer Security (TLS) Protocol Version 1.3 https://datatracker.ietf.org	10 Июл 2020	Интернет Free

Еще источников: 5

Еще совпадений: 0,51%

С результатами ознакомлена