МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
АНГАРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
ИНСТИТУТ ВЫЧИСЛИТЕЛЬНОЙ МАТЕМАТИКИ И МАТЕМАТИЧЕСКОЙ ГЕОФИЗИКИ СО РАН

НОВЫЕ ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ В ИССЛЕДОВАНИИ СЛОЖНЫХ СТРУКТУР

МАТЕРИАЛЫ ЧЕТЫРНАДЦАТОЙ МЕЖДУНАРОДНОЙ КОНФЕРЕНЦИИ 19–24 сентября 2022 г.

Томск
Издательский Дом Томского государственного университета
2022

2. OWASP Top Ten [Электронный ресурс] // OWASP. – Annapolis: OWASP Foundation, Inc., 2022. – URL: https://owasp.org/www-project-top-ten/ (дата обращения: 02.07.2022).
3. *Kolomeets A., Shirokova E., Gromov M., Yevtushenko N.* Checking Robustness of Web Services based on Solving Automata Equations // 2021 IEEE 22nd International Conference of Young Professionals in Electron Devices and Materials (EDM) – New-York : IEEE, 2021. – P. 499–502. – DOI: 10.1109/EDM52169.2021.9507669.
4. *Yu F., Alkhalaf M., Bultan T.* Generating vulnerability signatures for string manipulating programs using automata-based forward and backward symbolic analyses [Электронный ресурс] // UC Santa Barbara Computer Science. Tech Reports. – Santa Barbara : University of California, 2009. – Report ID 2009-11. – 12 p. – URL: https://www.cs.ucsb.edu/sites/default/files/documents/2009-11.pdf.
5. *Kari L.* On language equations with invertible operations // Theoret. Comput. Sci. – 1994. – Vol. 132. – P. 129–150.

ЭКСПЕРИМЕНТ ПО ВЫЯВЛЕНИЮ ИЗБЫТОЧНОГО СЕТЕВОГО ВЗАИМОДЕЙСТВИЯ DART-РЕАЛИЗАЦИИ SMTP-КЛИЕНТА С СЕРВЕРОМ

А.П. Сотников, Н.В. Шабалдина, М.Л. Громов, С.А. Прокопенко, А.С. Твардовский

Томский государственный университет, Томск, Россия
sotnikhtc@gmail.com

REDUNDANT INTERACTION EXPERIMENTAL IDENTIFICATION OF THE DART-IMPLEMENTATION OF THE SMTP CLIENT WITH THE SERVER

A.P. Sotnikov, N.V. Shabaldina, M.L. Gromov, S.A. Prokopenko, A.S. Tvardovskii

Tomsk State University, Tomsk, Russia

При совместной работе клиентской и серверной реализаций протоколов возникают ситуации, когда клиент осуществляет избыточные запросы к серверу, в то время как и без этих запросов ответ клиента конечному пользователю предопределен. Такая предопределенность может быть обусловлена как требованиями RFC, так и, в некоторых случаях, накопленной к этому этапу взаимодействия информацией, например, о настройках сервера или о реализованных на сервере опциях.

В данной работе описан проведенный эксперимент по выявлению избыточного сетевого взаимодействия Dart-реализации SMTP-клиента [1] со специально написанной программой-сервером, частично имитирующей работу SMTP-сервера согласно спецификации [2]. Под лишним сетевым взаимодействием здесь понимается такое взаимодействие, которого можно избежать полностью, так как клиент к настоящему моменту уже накопил определенную информацию и может самостоятельно анализировать данные перед отправкой запроса на сервер.

В ходе проведения эксперимента выявлены два случая избыточного сетевого взаимодействия.

Первый случай такого взаимодействия обнаружен при попытке клиента аутентифицироваться на сервере с неподдерживаемым типом аутентификации. Согласно RFC-спецификации SMTP-протокола [2], перед отправкой команды AUTH на сервер, клиент отправляет команду EHLO, в ответ на которую от сервера приходит информация о том, какие типы аутентификации сервер поддерживает. Таким образом, на этапе отправки команды AUTH клиент имеет всю необходимую информацию для того, чтобы определить, является ли запрос корректным. Для проверки наличия сетевого взаимодействия в этом случае был написан специальный юнит-тест, в котором после получения ответа на команду EHLO клиент отправляет запрос на аутентификацию с типом, который не поддерживается сервером. Для обнаружения сетевого взаимодействия в исходный код клиента было добавлено фиксирование всех исходящих к серверу сетевых запросов. Эксперимент показал, что, имея информацию о поддерживаемых на сервере типах аутентификации, клиент все равно осуществляет сетевое взаимодействие с заведомо неподдерживаемым типом аутентификации.

Второй случай избыточного взаимодействия обнаружен при проверке сценария отправки письма с пустым списком получателей. В этом случае на стороне клиента сработала проверка на пустоту списка получателей. Сетевого взаимодействия между клиентом и сервером не было, клиент выдал исключение, предупреждающее о пустом списке получателей. Также в рамках данного сценария был проверен случай, в котором список получателей задавался непустым, однако адреса получателей оставались пустыми строками нулевой длины. Здесь стоит уточнить, что такая возможность связана с деталями реализации – получатель представлен классом, имеющим несколько полей: имя, фамилия и почтовый адрес; то есть создается объект получателя, у которого все строки остаются пустыми, этот получатель добавляется в список получателей и ему отправляется письмо. Такое письмо проходит проверку на стороне клиента и со стороны клиента отправляется запрос на сервер.

Экспериментальные результаты показали, что рассматриваемая клиентская реализация не накапливает информации о предыдущем сетевом взаимодействии, что, соответственно, не позволяет избежать избыточного сетевого взаимодействия клиентской и серверной частей. В дальнейшем планируется формализовать выявление лишнего сетевого взаимодействия путем анализа формальных моделей.

Литература

1. *J. Klensin.* Simple Mail Transfer Protocol [Электронный ресурс] // RFC Editor. – URL: https://www.rfc-editor.org/rfc/rfc5321.html (дата обращения: 06.07.2022).
2. enough_mail 2.1.1. IMAP, POP3 and SMTP clients for Dart and Flutter email developers [Электронный ресурс] // pub.dev. The official package repository for Dart and Flutter apps. – URL: https://pub.dev/packages/enough_mail (дата обращения: 06.07.2022).

ПРОВЕРКА СВОЙСТВА РЕАКТИВНОСТИ В DART-РЕАЛИЗАЦИИ SMTP-КЛИЕНТА

А.П. Сотников, М.Л. Громов, С.А. Прокопенко, Н.В. Шабалдина

Томский государственный университет, Томск, Россия
sotnikhtc@gmail.com

CHECKING INPUT-OUTPUT BEHAVIOUR OF THE SMTP-CLIENT DART IMPLEMENTATION

A.P. Sotnikov, M.L. Gromov, S.A. Prokopenko, N.V. Shabaldina

Tomsk State University, Tomsk, Russia

При работе клиент-серверных приложений в некоторых случаях важным является сохранение свойства реактивности, которое подразумевает, что следующий запрос от клиента к серверу отправляется после того, как клиент получил ответ на предыдущий запрос. Например, в SMTP протоколе [1] необходимо дождаться ответа на команду (запрос) EHLO перед тем, как продолжать взаимодействие с сервером. Это связано с тем, что сервер в ответе на данную команду указывает поддерживаемые им механизмы аутентификации и прочую информацию, которая нужна для организации дальнейшего взаимодействия. Кроме того, сохранение взаимодействующими реализациями свойства реактивности позволяет описать их поведение (в том числе, совместное) при помощи модели с конечным числом переходов, например, конечного автомата [2], что может быть полезно для анализа системы и синтеза тестов.

В данной работе мы проверяем сохранение свойства реактивности в Dart-реализации клиентской части SMTP. В языке Dart есть механизм асинхронного выполнения функций, необходимый, например, для работы графического интерфейса без блокирования элементов интерфейса. Асинхронные функции присутствуют практически в любой реализации клиентской части, и при некорректном их использовании могут привести к нарушению свойства реактивности при взаимодействии с сервером. Поэтому актуальным является наличие в клиентской реализации проверки, что свойство реактивности не нарушается. Убедиться в том, что такая проверка осуществляется, можно с помощью юнит-теста, в котором команды вызываются без ключевого слова await или использования метода .then. Отсутствие await и .then как раз позволяет смоделировать ситуацию нарушения свойства реактивности за счет того, что следующая команда юнит-теста отправляется без ожидания ответа на предыдущую команду.

В качестве исследуемой программы для эксперимента по проверке свойства реактивности была выбрана Dart-реализация клиентской части SMTP из работы [3]. Написана программа-сервер, отвечающая на запросы клиента с фиксированной задержкой, что имитирует работу реальной сети. План эксперимента следующий. Используем экземпляр класса клиента SMTP. С помощью вызовов методов этого класса добьёмся того, чтобы на сервер была отправлена последовательность команд, в которой очередная команда должна по-хорошему подаваться в зависимости от ответа сервера на предыдущую команду. Методы будем вызывать без использования ключевого слова `await` или использования метода `.then`. Если на очередной вызов метода до прихода ответа на предыдущие команды клиент ответит отказом (выбросит исключение или вернёт ошибку), то рассматриваемая реализация клиентской части следит за сохранением реактивности, иначе нет.

В эксперименте была выбрана следующая последовательность. Клиент сначала посылает на сервер команду EHLO. Напомним, что ответ на эту команду содержит, в том числе, поддерживаемые сервером механизмы аутентификации. Далее клиент пытается аутентифицироваться на сервере, указывая неподдерживаемый сервером механизм аутентификации, после чего посылает запрос на отправку письма. Подчеркнем, что в юнит-тесте все команды клиента намеренно отправлялись одна за другой (без анализа ответов сервера). Сервер ответил ошибкой на попытку аутентификации с неподдерживаемым механизмом, но к