*electronics*

*Article*

# API Message-Driven Regression Testing Framework

**Emine Dumlu Demircioğlu * and Oya Kalipsiz**

Department of Computer Engineering, Yildiz Technical University, 34320 Istanbul, Turkey
* Correspondence: emine.dumlu@borsaistanbul.com

**Abstract:** With the increase in the number of APIs and interconnected applications, API testing has become a critical part of the software testing process. Particularly considering the business-critical systems using API messages, the importance of repetitive API tests increases. Successfully performing repetitive manual API testing for a large number of test scenarios in large business enterprise applications becomes even more difficult due to the fact that human errors may prevent performing thousands of human-written tests with high precision every time. Furthermore, the existing API test automation tools used in the market cannot be integrated into all business domains due to their dependence on applications. These tools generally support web APIs over the HTTP protocol. Hence, this study is motivated by the fact that there is a lack of API message-driven regression testing frameworks in a particular area in which API messages are used in client-server communication. This study has been prepared to close the gap in a specific domain which uses business domain APIs, rather than HTTP, in client-server communication. We propose a novel approach based on the use of network packets for regression testing. We developed a proof-of-concept test automation tool implementing our approach and evaluated it in a financial domain. Unlike prior studies, our approach can provide the use of real data packets in software testing. The use of network packets increases the generalization of the framework. Overall, our study reports remarkable reuse capacity and makes a significant impact on a real-world business-critical system by reducing effort and increasing the automation level of API regression testing.

**Keywords:** regression testing; API testing; case study; client-server applications

check for updates

**Citation:** Demircioğlu, E.D.; Kalipsiz, O. API Message-Driven Regression Testing Framework. *Electronics* **2022**, *11*, 2671. https://doi.org/10.3390/electronics11172671

Academic Editors: Sanghyuk Lee, Soo Kyun Kim, Asad Abbas, Seokhun Kim and Scott Uk-Jin Lee

Received: 18 July 2022
Accepted: 19 August 2022
Published: 26 August 2022

## 1. Introduction

API, short for application programming interface, is a software agent which enables two applications to communicate with one another. Each time an instant message is sent using an application such as WhatsApp, an API is used [1]. In this case, the application connects to the server and starts to send an API message as a request. Then, the server accepts the incoming request from the clients and conveys the responses over the network using API messages. This is the usual work of an API. Since APIs are constantly evolving and the number of interconnected applications is increasing, API testing has become a critical part of software testing. API testing validates whether the developed API meets development expectations in terms of functionality, reliability, security, and performance. API testing of client-server applications, such as web services and TCP servers, in which the communication is performed via API messages, are specifically significant as well. This type of testing has challenges, as inputs/outputs are sequences of request/response API messages to a remote server [2]. In addition to API testing, there is also an increasing need for effective regression testing used to check whether the software still works correctly after changes in the software systems, and which is also used to detect a regression in an API message, since any minor change or new function in the software can produce many unexpected results [3]. Regression testing is the most challenging testing activity in large-scale software system development. While software testing constitutes 50–60% of the total cost of a project, regression testing may account for 80% of the total testing cost [4–7].

Moreover, especially in large-scale business software systems, the manual realization of this endeavor is very difficult, laborious, and time-consuming due to the enormous number of possible test scenarios. Consequently, a vast investment in human resources would be necessary to effectively perform this kind of test [8–10]. Moreover, without any testing tool, performing regression testing repeatedly for order-dependent tests (OD tests), which are tests that are passed when run in one sequence, but are failed when run in another sequence, is also difficult 1. For OD tests, test results depend on the test execution order. For example, there is a database connection in test 1, and test 2 uses this connection instead of re-connecting to the database. In this case, t2 should be run after running t1; otherwise t2 will fail. These types of tests are referred to as OD tests. If OD tests are not executed in a specified order, it may cause the test to fail. This failure causes developers to believe that their changes are breaking existing functionality, causing wasted time for the developers. Moreover, the productivity of developers is also compromised, as they search for errors in their changes that do not exist 1. Thus, the order of test cases for OD tests should be guaranteed by the regression test automation.

Today, web services play a major role in the development of enterprise applications [11]. This kind of service provides data via web API over a network by using HTTP as an application layer protocol. Due to the increasing number of web services, there are many studies about their testing process, but the existing API test automation tools and studies in the literature are generally regarding web APIs over HTTP protocols; specifically, REST APIs are the most widely employed APIs [12–18]. Moreover, the existing API testing tools used in the market cannot be integrated into all business domains due to the dependence on applications and limitations. The need of test script code development, extended test data preparation, the manual creation of test cases, and difficulty in reuse are a few of the difficulties of existing API testing tools [19]. Hence, there is a lack of API regression test automation tools in the software testing industry for an area in which domain-specific API messages are used for communication, other that for the HTTP protocol. To this end, we propose a novel API message-driven regression testing framework for client-server architecture systems to ensure end-to-end automation based on a real industrial need for the business domain-specific APIs that reflect the knowledge of the domain. This paper proposes a novel approach using network data packets for regression testing. Basically, our proposed framework automatically extracts test cases from the captured network packets during the client-server communication over the network by using reverse engineering mechanism in regression testing. To validate the effectiveness of the proposed framework, we administered it in a real-world financial system.

In our previous work [20], we presented an approach on extracting the test cases from network packets by using the reverse engineering method. In this work, we extend our previous work by adding execution, validation, and defect reporting phases in order to replay network packets into the test environment. Moreover, we applied the developed framework to the financial domain to evaluate its effectiveness and accuracy.

The main contributions of this study are:

- This study is aimed to fill the gap in the regression testing framework in a specific domain—client-server architecture applications such as financial applications and trading systems that use business domain-specific API messages in their communication.
- The use of network packets increases the generalization of the framework. The easiest method to receive different request messages coming from various clients is to use network packets. This ensures more application independence. For this reason, this study uses network packets as an input. No previous research has attempted to implement and use network packets in regression testing. The network log file is standard, and is not specific to our study. It includes API request-response message pairs.
- By capturing the real data packets in production, the real client behavior can be simulated into the test environment. Moreover, a bug that occurs in the produc-

tion environment, but not in the test environment, can be quickly simulated in the test environment by replaying the network packets obtained from the production environment.
- We applied the proposed framework to a real-world financial application domain in order to validate its effectiveness and obtain better test automation outcomes. We worked on two different business domain-specific API protocols used by the financial domain. According to the extracted metrics, which are proved to be accurate via the test engineers, this framework has shown remarkable reuse capacity, filling the gap in the software testing industry for business-critical systems that have business domain-specific APIs.

The rest of the study is composed as follows: Section 2 provides the relevant background information and related work. The proposed approach is described in Section 3. Section 4 explains the case study and shows the test results. Finally, Section 5 offers the conclusions.

## 2. Background and Related Work

To better understand the rest of the study, we describe the targeting system structure and provide a brief overview of the concept of API testing. We finish the section by reviewing the related work in the scope of API testing tools in the literature.

### 2.1. Case Description and Targeting System Structure

As discussed in Section 1, this work was initiated by an industrial need and in this section, the targeting system structure is described.

At a high level, the targeting system is indicated in Figure 1. This is a client-server system. The communication between client and server is performed over the network via business domain-specific API messages. As a transport layer protocol, TCP or UDP can be used. The API messages are carried via TCP or UDP data packets. TCP is short for transmission control protocol, which is a slower but more reliable connection-oriented protocol, whereas UDP, which is short for user datagram protocol, is a faster connectionless protocol. Each TCP/UDP data packet consists of a header and a payload section, of which the header section contains destination/source port numbers. Moreover, TCP protocol guarantees that data arrives, but UDP does not. TCP protocol tracks data packets to guarantee packet delivery. Therefore, the TCP header includes acknowledgement and sequence numbers in order to track how much data has been transferred [21]. The payload section carries application layer data, which is an API message, for the application. Each client connects to the server by using TCP or UDP as a transport layer protocol. After successful connection, the client sends a request API message, carried via the payload section of the data packet; the server receives this request, processes it, prepares a response API message, and finally sends it to the client. Hence, the communication between the client and server application is accomplished via API messages.

Our approach basically accesses the data at the packet level. Therefore, in order to obtain data packets in the communication, a packet capture analyzer tool is used. There are many different types of tools on the market, including Wireshark [22], Tcpdump [23], Arkime [24], etc. These are a few open-source packet capture and network traffic analyzer tools referenced in the literature. In our case study, to collect data packets over the network, we used the Wireshark tool, which is also a free network traffic packet analyzer tool.

Over the network, the packets are captured and stored in a pcap file using the Wireshark tool. Any version of this tool can be used to create a pcap file. Pcap is short for packet capture, and it consists of network data. Figure 2 displays the content of a pcap file (TestSuite.pcap) with a screenshot of the Wireshark tool. Each line in the file represents the communication data packets between client and server applications.

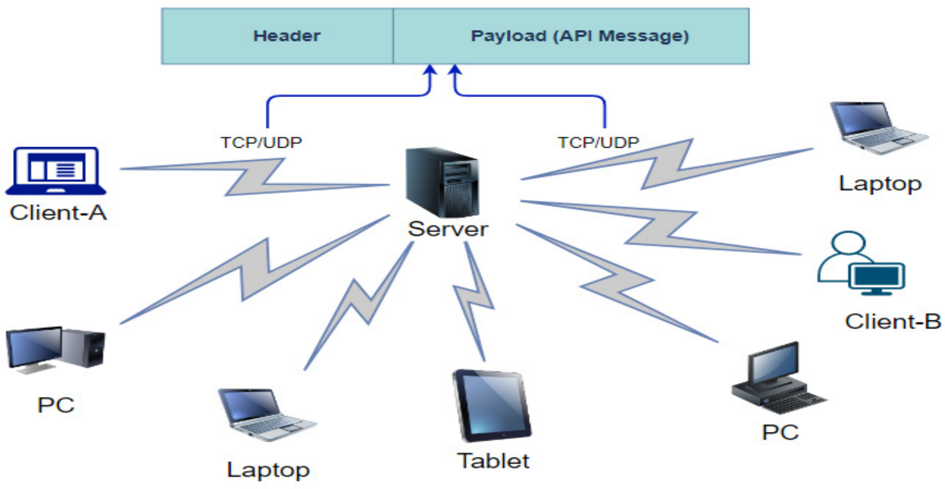
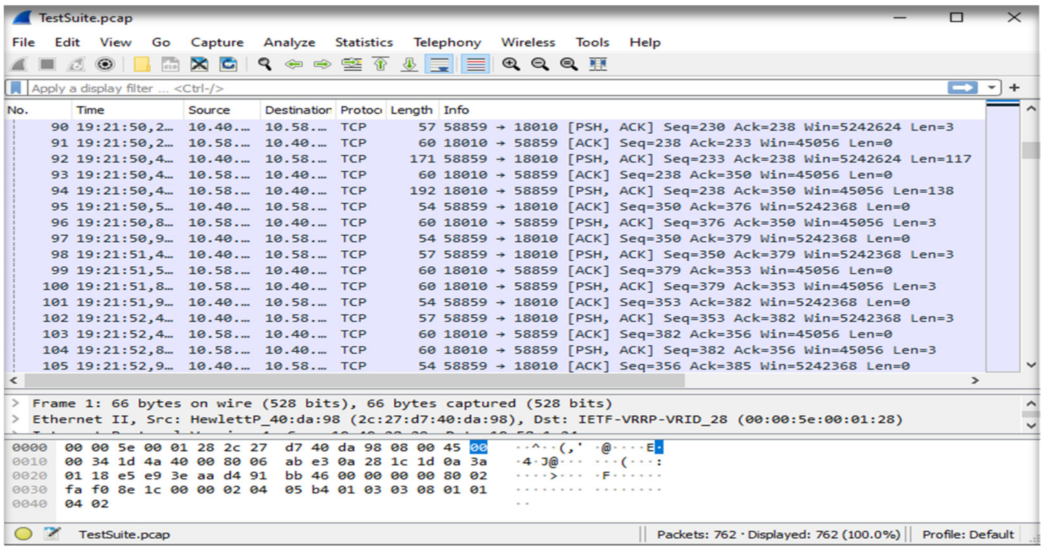


**Figure 1.** Targeting System Architecture.


TestSuite.pcap
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>

```
No.   Time          Source    Destination Protocol Length Info
   90 19:21:50,2…  10.40.…   10.58.…   TCP     57 58859 → 18010 [PSH, ACK] Seq=230 Ack=238 Win=5242624 Len=3
   91 19:21:50,2…  10.58.…   10.40.…   TCP     60 18010 → 58859 [ACK] Seq=238 Ack=233 Win=45056 Len=0
   92 19:21:50,4…  10.40.…   10.58.…   TCP    171 58859 → 18010 [PSH, ACK] Seq=233 Ack=238 Win=5242624 Len=117
   93 19:21:50,4…  10.58.…   10.40.…   TCP     60 18010 → 58859 [ACK] Seq=238 Ack=350 Win=45056 Len=0
   94 19:21:50,4…  10.58.…   10.40.…   TCP    192 18010 → 58859 [PSH, ACK] Seq=238 Ack=350 Win=45056 Len=138
   95 19:21:50,5…  10.40.…   10.58.…   TCP     54 58859 → 18010 [ACK] Seq=350 Ack=376 Win=5242368 Len=0
   96 19:21:50,8…  10.58.…   10.40.…   TCP     60 18010 → 58859 [PSH, ACK] Seq=376 Ack=350 Win=45056 Len=3
   97 19:21:50,9…  10.40.…   10.58.…   TCP     54 58859 → 18010 [ACK] Seq=350 Ack=379 Win=5242368 Len=0
   98 19:21:51,4…  10.40.…   10.58.…   TCP     57 58859 → 18010 [PSH, ACK] Seq=350 Ack=379 Win=5242368 Len=3
   99 19:21:51,5…  10.58.…   10.40.…   TCP     60 18010 → 58859 [ACK] Seq=379 Ack=353 Win=45056 Len=0
  100 19:21:51,8…  10.58.…   10.40.…   TCP     60 18010 → 58859 [PSH, ACK] Seq=379 Ack=353 Win=45056 Len=3
  101 19:21:51,9…  10.40.…   10.58.…   TCP     54 58859 → 18010 [ACK] Seq=353 Ack=382 Win=5242368 Len=0
  102 19:21:52,4…  10.40.…   10.58.…   TCP     57 58859 → 18010 [PSH, ACK] Seq=353 Ack=382 Win=5242368 Len=3
  103 19:21:52,4…  10.58.…   10.40.…   TCP     60 18010 → 58859 [ACK] Seq=382 Ack=356 Win=45056 Len=0
  104 19:21:52,8…  10.58.…   10.40.…   TCP     60 18010 → 58859 [PSH, ACK] Seq=382 Ack=356 Win=45056 Len=3
  105 19:21:52,9…  10.40.…   10.58.…   TCP     54 58859 → 18010 [ACK] Seq=356 Ack=385 Win=5242368 Len=0

> Frame 1: 66 bytes on wire (528 bits), 66 bytes captured (528 bits)
> Ethernet II, Src: HewlettP_40:da:98 (2c:27:d7:40:da:98), Dst: IETF-VRRP-VRID_28 (00:00:5e:00:01:28)

0000  00 00 5e 00 01 28 2c 27  d7 40 da 98 08 00 45 00   ··^··(,' ·@····E·
0010  00 34 1d 4a 40 00 80 06  ab e3 0a 28 1c 1d 0a 3a   ·4·J@··· ···(···:
0020  01 18 e5 e9 3e aa d4 91  bb 46 00 00 00 00 80 02   ····>··· ·F······
0030  fa f0 8e 1c 00 00 02 04  05 b4 01 03 03 08 01 01   ········ ········
0040  04 02                                              ··

TestSuite.pcap    Packets: 762 · Displayed: 762 (100.0%)    Profile: Default
```

**Figure 2.** Screenshot of Wireshark displaying TCP data packets of a TestSuite.pcap file.

## 2.2. *API Testing*

APIs define protocols, order the client calls, and incorporate their own specifications. APIs enable communication between two software applications by establishing precise rules. More specifically, APIs are responsible for determining the types of requests and data formats. As companies strive to build scalable and flexible software products, they are increasingly turning to API-driven development. Most major enterprise applications, such as government apps, banking apps, payment apps, e-mail apps, or airlines apps use APIs and their functionalities to provide services [25]. There are many open APIs provided by the software industry in order to access their services. According to the report on the API testing market performed by Markets, the number of APIs is expanding exponentially, and it is expected that the size of the API testing market will increase from USD 384.3 million in 2016 to USD 1099.1 million by 2022 [26].

With the rise in the number of APIs, cloud applications, and interconnected platforms, API testing has utmost importance as a type of software testing. API testing is performed to detect bugs and ensure the quality of the application business logic. Therefore, it is performed at the business logic layer of the software application. The content of the business logic layer is very critical to the successful operation of applications. Any bug in the API message can cause large-scale software failures. So, API testing should be conducted to ensure the quality and have bug-free APIs.

### *2.3. Related Work*

Generally, there is a reported lack of research on test automation and especially on automated API testing. In industrial practice, the experience reports of real-word applications related to test automation are rare [27]. Moreover, the API testing tools in presented in the literature are generally for web-APIs, such as SOAP web services or REST APIs, by using XML or JSON data message formats that can be accessed using HTTP as an application layer protocol [28–30]. Initially, SOAP was used as a messaging protocol. It sends documents in XML format over HTTP, but today REST APIs are the most widely used APIs [31]. To transfer REST API responses over HTTP, REST-based web applications use XML or JSON data formats [32]. These specific data formats are not suitable for the other business domain-specific APIs in terms of usability. Furthermore, we have conducted some tests to determine whether existing testing tools are the really fit to the desired system. To this end, the Apache JMeter testing tool, which is released by the Apache Software Foundation an American corporation, is selected, which is a widely used testing tool in the market 32. Test scenarios were created manually on the 5.4.1 version of tool, but there were some limitations and difficulties during the creation, execution, and validation of the test cases. Consequently, the Apache JMeter testing tool could not be adapted into a financial domain using the API protocol.

Moreover, there are more generic testing tools, such as GoReplay [33], Polly.js [34], and node-replay [35] on GitHub. These tools capture and replay network traffic over a network and are used for load testing, but they do not validate the functionality of an application and do not perform regression testing. Hence, there is a gap in the literature in terms of API testing tools supporting business domain-specific API protocol except for the HTTP protocol. We are not aware of any existing test automation tool in the industry which is suitable for domain-specific APIs, such as financial application domain APIs. This type of testing is rather limited.

We examined existing software test automation tools in the literature and listed some of the various types of software regression test automation tools [36]:

- Functional testing tools: Selenium, Ranorex, QTP (Quick Test Professional, JMeter, etc.)
- Load testing tools: Jmetter-Blazemeter, Webload, LoadRunner, etc.
- Regression testing tools: Selenium, JMeter, Postman, etc.

In the literature, there are different types of test automation tools that can be used according to the purpose of the testing, such as the functional correctness of the application, load testing, stress testing, etc. For example, to be able to check the functional correctness of a web application, Selenium, jMeter, Postman, etc. can be used. These are mostly for web, mobile, and desktop applications. Extended test data preparation, high overhead, manual creation of test cases, and difficulty in reuse are a few of drawbacks of existing API testing tools [19]. Moreover, the tester needs to develop test script codes to adapt the existing test automation tool to the desired system. This process is both time-consuming and costly [37]. Furthermore, the existing API automation tools, such as Postman, Soap-UI, and others, which allow you to test REST and SOAP APIs do not automatically generate tests. To be able to automate the testing process, test cases must be manually created. Moreover, they only exist for REST-API and SOAP-API that support web APIs over the HTTP protocol, and do not support other business domain-specific APIs. To be able to test a TCP server which provides APIs by using third-party testing tools, the tester should develop test script code to adapt the existing test automation to the desired system. Based on common experience, we know that test script code requires a maintenance process which is costly and time-consuming. Our intention is to eliminate the test script code development and maintenance process.

Table 1 compares the capabilities of the proposed framework with existing tools. The first column lists the supported features of the tools/studies. A ✓sign shows the existence of the capability, and an X sign represents the lack thereof. According to our comparison, no existing tools or studies that closely fit the requirements of API testing in a business-critical system context could be found. With this objective in mind, this paper proposes a framework for client-server applications, relying on the capturing of packets at the network level to ensure end-to-end testing. Our framework basically generates test cases extracted from network packets between client-server communications over a network by using the reverse engineering technique. This paper proposes an approach using network packages for regression testing.

**Table 1.** Comparison with other tools/studies that perform the provided features.

| *Features* | *Our Tool* | *[8–11]* | *Postman* | *JMeter* |
|---|---|---|---|---|
| Supports API Testing | ✓ | ✓ | ✓ | ✓ |
| Supports Web-API Testing | ✓ | ✓ | ✓ | ✓ |
| Supports Business Domain Specific APIs | ✓ | X | X | with TCP sampler, but limited |
| Simulates Real Client Behaviors with Prod. Data | ✓ | X | X | X |
| Test Case Generation | ✓ | ✓ | X | X |

## 3. The Approach: API Message-Driven Regression Testing Framework and Its Implementation

The basic idea is to capture messages on the network, extract test cases, and replay them to perform regression testing of the API messages. Figure 3 depicts the overall approach, including four steps. The input file of the proposed framework is a network log file. In the first step, the file is read by the framework, and the reverse engineering approach is applied in order to extract API messages which are stored in the TCP/UDP data packet's payload section. Next, matching between the request and response API messages is performed by using source/destination IP/port information stored in the TCP/IP header in order to create a test case. Each test case in the test suite contains both messages sent by the client and the responding messages sent by the server to the client. Therefore, in the first three steps, the test suite, which includes a set of test cases, is extracted from the network packets. Next, previously extracted test suites consisting of test cases are implemented in the test environment, and validations are conducted. Finally, test results which include pass/fail counts, unexpected API messages, missing API messages, and detailed validation errors are produced and tabulated. The implementation details of each step are explained below.

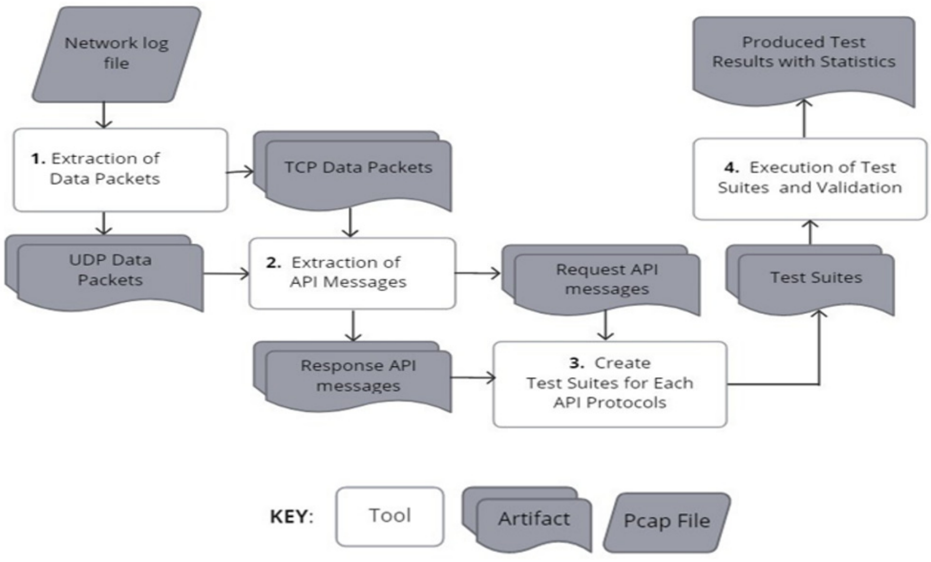

Network log file
1. Extraction of Data Packets
TCP Data Packets
UDP Data Packets
2. Extraction of API Messages
Request API messages
Response API messages
3. Create Test Suites for Each API Protocols
Test Suites
4. Execution of Test Suites and Validation
Produced Test Results with Statistics
KEY:
Tool
Artifact
Pcap File

**Figure 3.** The overall process.

The proposed framework is implemented in Java programming language. Algorithm 1 shows the pseudocode of the proposed methodology. Basically, the framework takes the pcap file as an input and produces a test suite by calling a generateTestSuites(file) routine, which is a set of test cases. Next, each test case is executed through the targeting test system by calling a replay(τ) routine. After replaying a test case, the actual API response message is obtained. At the end, validation is performed on the expected and actual message, and a test result is saved. As a result, our study is conducted as follows:

(1) Extraction of test suites from the network packets.
(2) Execution of the test suite.
(3) Validation and defect reporting.

**Algorithm 1** replay()

```
Input file: .PCAP
1: testSuites ← generateTestSuites (file)
2  for each test case τ in testSuites do
3:   actualAPIMessage ← replay (τ)
4:   testResult ← validate (t.actualAPIMessage, t.expectedAPIMessage)
5:   addToTestResultList(τ);
6: end for
```

We explain each step of the approach, with implementation details, below.

### *3.1. Extraction of Test Suites from Network Packets*

The first step of the proposed approach is to extract test suites from the network packets. A pcap file is used as an input file for the framework. This file is analyzed, and then the API messages transferred between client-server applications are extracted by using the reverse engineering approach. The extracted API messages are mapped as a request and response pair. At the end, the framework generates a test suite, which is a collection of test cases. Each test case in a test suite includes both client-sent request messages and corresponding server-sent response messages.

Algorithm 2 illustrates our algorithm of how to create a test suite from a pcap file. On a higher level, the algorithm describes the "generateTestSuite()" routine. It loops until the end of the pcap file. The proposed framework basically reads the pcap file and automatically generates test suites, including test cases, to use them in regression testing.

**Algorithm 2** generateTestSuites (file)

```
Input file: file
1: testSuites ← empty
2: do
3:     packet ← nextPacket()
4:     packetDetails ← extractPacketDetails(packet)
5:     apiMessages ← buildAPIMessageInPacket (packet)
6:     putIntoTestSuite (apiMessages)
7: while not end of file
8: return testSuites;
```

For each iteration, a network data packet is obtained by the calling nextPacket() method. When a data packet is received, the framework specifies its transport layer protocol as TCP or UDP. A data packet consists of a header and a payload section. In line 3, by making a call extractPacketDetails (packet), detailed packet information, such as destination IP/port, source IP/port, and arrival time information, is extracted from the header section of the packet. In addition, if the packet is recognized as a TCP data packet at the beginning, the framework determines whether it is a retransmission packet or not. To be able to detect it, the framework uses sequence numbers from the data packet's header sections. The sequence number of the next package is calculated by summing the sequence number of the current data packet and the length of the payload section. For example, if a data packet sequence number is 1 and the length of the payload is 60, the next packet sequence number should be 61. If the sequence number of the next packet in the pcap file is less than the calculated sequence number, this packet is marked as a retransmitted packet by considering the retransmitted packet length. If the packet is a type of retransmitted packet, the framework ignores it. The retransmitted packet is not re-added to the test suite because it is considered that it was previously added to test suite. There is no retransmission mechanism due to the connectionless UDP, so there is no need for this kind of calculation in UDP. In line 4, the API messages are constructed from the payload section of the data packet. The payload section of a data packet carries API messages. To be able to detect which API protocol is used, the heuristic search technique is applied. This technique is a search technique that tries to detect a good, if not perfect, solution from the available options [38–40]. The application-layer protocol pattern is searched in the payload section. The API message pattern provided by targeting systems are investigated from within the packet's payload. For example, in a financial domain context, to transfer financial data electronically from the clients to the server, the FIX API protocol can be used. To apply the FIX API protocol to the framework, the following pattern can be defined in the framework as a search pattern:

"Pattern.compile('8=FIXT*?\\x0110=.{3}\\x01')"

As each FIX API message starts with "8=FIX" and ends with "10=XXX" (XXX shows the value of checksum calculation), this kind of pattern can be searched in the network packet payload section. If this pattern can find a suitable FIX API message, it creates a message; otherwise, the framework continues to try another API message pattern to be able to extract API message. Consequently, the framework extracts all information, such as source/destination, IP/Port, and API messages related to a data packet. As a final step, the API messages extracted according to the pre-defined messaging format are added to the test suite in line 5 using a mapping request and a response pair. This process is repeated until the end of the pcap file.

Various protocols can contain packets that are related to a specific environment and therefore, cannot be reused. For instance, the HTTP response has the most recently modified time of the target page, authorization field, content length, session information, or response date, which are volatile. Such values should be ignored in validation, because they can take different values in each run. To this end, instead of replaying the network packet without the extraction of the API message, the proposed framework analyzes and extracts application layer packets and ignores the volatile values during validation.

Such approaches can be adapted for the different business domains. To do this, the buildAPIMessageInPacket(packet) method should be implemented, in accordance with the targeting system's API message structure. By adding a new application protocol identifier into the framework, the heuristic search technique can be enhanced, and the proposed framework can easily be extended into the different API protocols.

### *3.2. Execution of Test Suite*

The goal of this step is to execute previously created test cases on the system under test (SUT). Each test case consists of destination IP, destination port, and API messages (request/response pair) extracted from the pcap file. After detecting transport layer protocol, the framework first tries to connect to the server by using destination IP and port information. After connecting to the test server, it starts to send API messages. Tables 2 and 3 provide examples of the test cases extracted from the pcap file in a financial domain system. The request API message in Table 2 is a new order message which belongs to the FIX API protocol. Similarly, the API request message in Table 3 is an update message which belongs to the FIX API protocol. These messages were extracted from the payload section of a data packet, as they meet the pre-defined FIX API message pattern: ('8=FIXT*?\\x0110=.{3}\\x01')". The whole list of FIX API protocol message structures is available in the specification document [41]. For the first example, the framework connects to the server by using destination IP and port, which is 10.40.X.X:17090. After successful connection, it sends the API request message to the test server and waits for the response message. The framework replays all test cases extracted from the pcap file.

**Table 2.** An example of a test case for FIX API protocol extracted from the pcap file—new operation.

| **Test Case** | **1** |
|---|---|
| Dest IP | 10.40.x.x |
| Dest Port | 17090 |
| API Request Message | [8 = FIXT.1.1 \| 35 = D \| 34 = 7 \| 49 = BIA \| 50 = F91 \| 52 = 14:16:54.058 \| 56 = BI \| 1 = XYZ \| 11 = 22102 \| 38 = 20 \| 40 = 2 \| 44 = 21.000 \| 54 = 1 \| 55 = ACS.E \| 59 = 0 \| 60 = 14:16:54.057 \| 70 = AAA \| 528 = A \| 10 = 048 \| ] |
| Expected API Message | [8 = FIXT.1.1 \| 35 = 8 \| 34 = 7 \| 49 = BI \| 52 = 14:16:54.335 \| 56 = BIA \| 57 = FX91 \| 1 = XYZ \| 6 = 0 \| 11 = 22102 \| 14 = 0 \| 17 = 1 \| 22 = M \| 37 = **6054100010C68** \| 38 = 20.0000000 \| 39 = 0 \| 40 = 2 \| 44 = 21.0000000 \| 54 = 1 \| 55 = ACS.E \| 59 = 0 \| 60 = 14:16:54.297 \| 70 = AAA \| 119 = 220.0000000 \| 150 = 0 \| 151 = 20.0000000 \| 528 = A \| 10 = 188] |

**Table 3.** An example of a test case for FIX API protocol extracted from the pcap file—update operation.

| **Test Case** | **2** |
|---|---|
| Dest IP | 10.40.x.x |
| Dest Port | 17090 |
| Request API Message | [8 = FIXT.1.1 \| 35 = G \| 34 = 9 \| 49 = BIA \| 50 = F91 \| 52 = 14:25:04.944 \| 56 = BI \| 11 = 22103 \| 37 = **6054100010C68** \| 38 = 20 \| 40 = 2 \| 41 = NONE \| 44 = 21.00000 \| 54 = 1 \| 55 = ACS.E \| 60 = 14:25:04.944 \| 70 = AAA \| 10 = 031 \| ] |
| Expected API Message | [8 = FIXT.1.1 \| 35 = 8 \| 34 = 9 \| 49 = BI \| 52 = 14:25:05.177 \| 56 = BIA \| 57 = FX91 \| 1 = ACC \| 6 = 0 \| 11 = 22103 \| 14 = 0 \| 17 = 2 \| 22 = M \| 37 = 6054100010C68 \| 38 = 20.0000000 \| 39 = 0 \| 40 = 2 \| 41 = 22102 \| 44 = 21.0000000 \| 48 = 70616 \| 54 = 1 \| 55 = ACS.E \| 59 = 0 \| 60 = 14:25:05.170 \| 70 = AAA \| 119 = 220.0000000 \| 150 = 5 \| 151 = 20.0000000 \| 528 = A \| 10 = 081 \| ] |

Since the reuse of the information in the response message is an effective technique in the testing of OD tests, our approach applies a strategy based on the response dictionary [14]. Actually, it is a map which stores key value pairs. After a test case execution, key values show the field value extracted from the old response message, and the value shows the actual field value extracted from the actual response message from the test server. In cases where some information from the previous message is needed, the response dictionary is used. For example, when the update operation needs to be tested, the new operation must have already been executed. For example, In FIX API protocol, to be able to test an update operation, the framework needs to have prior knowledge, which has been given by the previous new operation. Therefore, the matching operation of the old response message and the new response message is determined and stored in the response dictionary. Consequently, the execution of the OD test is guaranteed. Table 3 provides an example of an update operation extracted from the pcap File. To be able to replay/send update messages to the test server, the original new message would have already been sent to the server, as is an OD test. When a new order message is sent to the server, the server replies with a response message with a new unique ID (new ID: 7777BF00101; old ID: 6054100010C68, stored in response message). This ID is stored in a response dictionary, with the key showing the old ID, and the value showing the new ID. By using this information, before sending an update message to the server, the old ID is changed to a new ID to prevent a rejection message from the server. In this case, the ID field in a request API message for an update operation is updated from 6054100010C68 to 7777BF00101 before sending it to the test server. This technique is also applied to all OD tests.

### *3.3. Validation and Defect Reporting*

A regression in a software program is a bug which causes a function that initially functioned correctly to stop working after any changes, such as a software update. A software update is a change in the version of the software from n to n + 1 [29]. To detect a regression after a software update using our framework, the reference testing approach is used in which validation is performed on the actual messages received from the test server and the expected messages which are stored in the captured network log file. Our reference data is stored in a pcap file. Each test case, extracted from the pcap file, has an expected API message as a reference message. After replaying a test case through the test environment, the actual API response message is obtained. Finally, validation is performed on the actual messages received from the test server and the expected messages which are stored in the captured network log file by comparing them. An API message in a test case can have volatile values. Such values can take different values in each run. Thus, the framework should not validate all message fields in the API message because of the volatile values, such as date and timestamp fields. Hence, these values should be extracted and changed in order to prevent the inclusion of failures during the validation phase. To solve this problem, the specification-based validation approach is used to determine which message field will be validated. In this study, the diffing method was not used when comparing two API messages because of these types of volatile values. Instead, the framework uses the JSON schema to specify which fields will be excluded/included during validation of API messages. When defining the JSON Schema for the business domain-specific API message, all the required fields which show whether validation will be conducted or not are defined. The template file for API message definition is provided in Table 4.

**Table 4.** Template file for API message definition.

| JSON File Template | JSONDataType |
|---|---|
| {<br>“description”: “API Message Description”,<br>“properties”: {<br>“field”: {<br>“type”: “**JSONDataType**”,<br>“required”: true / false<br>}<br>}<br>} | String<br>boolean<br>object<br>integer<br>number<br>file<br>array<br>null |

The JSON files are created by applying the template file structure for each API message. After building the framework, the corresponding Java classes are generated. The sample creates JSON files using the template file are given in Table A1. Validation is conducted according to the generated Java classes. The fields, which will be validated in the messages, are specified by using “required: true” annotation information. During the validation phase, the expected and actual messages are compared for the required message fields which are determined in the JSON file. At the end, the framework generates a test result report with all passed/failed test cases, with the detailed reasons of failure.

## 4. Industrial Case Study

In this section, we present an industrial case study for evaluating our framework. The subject system is a trading system. The two different API protocols used by a financial domain are applied and adapted to demonstrate the applicability and efficiency of our proposed framework. These protocols are the FIX (financial information exchange) API protocol, which is text-based [41], and the OUCH API protocol, which is binary-based [42]. The application layer protocols are FIX-API and OUCH-API, which use TCP as a transport layer protocol. Both are active, widely used protocols in the trading systems. These are completely different API protocols which enable the exchange of financial data, and they have completely different message structures. FIX stands for financial information exchange, which is a communication protocol for real-time information exchange within the finance world. The FIX protocol language is comprised of a series of messaging specifications used in trade communications [43]. The FIX protocol defines many standards, such as how a connection should be established, message and field types, etc. OUCH is a simple protocol that allows traders to enter, cancel, and replace existing orders. The OUCH protocol consists of logical messages between the OUCH server and the client application. All the supported messages for the API protocols in question are listed in specification documents. The proposed framework will be used to test the correctness of these two different API protocols running on the test server. In the following, we explain our experimental setup and give the details related to the execution. We conclude the section with a discussion of threats regarding the validity our evaluation.

### *4.1. Experimental Setup and Data Preparation*

We conducted our case study on the two API protocols belonging to financial domain of the trading company. The current regression test suites of the company are used for the two API protocols in question. The quality of the test suite was defined in the company as the full coverage of all API messages listed in the API specification document, since it is a business-critical system. Thus, all API messages listed in the specification document should be covered and validated during the testing phase. While the testers execute the prepared test cases for the first time on the prod-like version of the system, the network data packets were captured by using the Wireshark tool and then saved into a pcap file. Therefore, that file will be a reference file for the testing of other subsequent versions of the system.

### *4.2. Execution and Analysis of the Results*

In our case study, FIX API test cases and OUCH API test cases are obtained, which are stored in the TestSuite.pcap file, as shown in Figure 2. Each line in the pcap file is a network packet including API messages. After preparing the pcap file, the framework is started. There is a command line option that takes the pcap file as an input parameter to start the framework. After starting, our framework reads this file and produces test suites for each different API protocol by applying the reverse engineering approach. Table 5. shows the selected test cases extracted from the network file using the OUCH API protocol. When considering the test case 1, if a login request message with the specified data within the table is sent to the server, the login response message is received as a response message. For test case 2, when "Enter Order Request Message" is sent to the server with the values specified in the table, the server should send the "Order Accepted Message" with the values specified in the table. Table 6 also provides the selected test cases for the FIX API protocol generated from the TestSuite.pcap file.

**Table 5.** Selected OUCH-API test cases extracted from TestSuite.pcap.

| Test ID | Dest IP | Dest Port | Request Message | Response Messages/Expected Messages |
|---|---|---|---|---|
| 1 | 10.40.X.X | 8020 | **Login Request Message**:<br>[AEH1,XXXXXX,TR16092650,1] | **Login Response Message**:<br>[TR16092650,0] |
| 2 | 10.40.X.X | 8020 | **Enter Order Request Message**:<br>[O \| 1a \| 10769 \| B \| 30 \| 4000 \| 0 \| 0 \| GMR \| REF \| 12345 \| 0 \| 1 \| 0] | **Order Accepted Response Message**:<br>[A \| 163586118732 \| 1a \| 10769 \| B \| 9427703165229781 \| 30 \| 4000 \| 0 \| 0 \| GMR \| 1 \| REF \| 12345 \| 30 \| 0 \| 1 \| 0] |
| 3 | 10.40.X.X | 8020 | **Replace Order Request Message**:<br>[U \| 1a \| 3407884 \| 25 \| 0 \| 0 \| \| \| \| 0 \| 0] | **Order Replaced Response Message**:<br>[U \| 163586167732 \| 3407884 \| 1a \| 10769 \| B \| 9427703165229781 \| 25 \| 4000 \| 0 \| 0 \| GMR \| 1 \| REF \| 12345 \| 25 \| 0 \| 1] |
| 4 | 10.40.X.X | 8020 | **Cancel Order Request Message**:<br>[X \| 1a] | **Order Canceled Response Message**:<br>[C \| 163586177305 \| 3407884 \| 10769 \| B \| 9427703165229781 \| 1] |
| 5 | 10.40.X.X | 8020 | **Enter Order Request Message**:<br>[O \| 3407888 \| 7076 \| B \| 22 \| 7000 \| 3 \| 0 \| GRM \| REF \| 12345 \| 0 \| 1 \| 0] | **Order Accepted Response Message**:<br>A \| 1635862229859 \| 3407888 \| 7076 \| B \| 4277031652297829 \| 22 \| 7000 \| 3 \| 0 \| GRM \| 2 \| REF \| 12345 \| 22 \| 0 \| 1 \| 0]<br>**Order Cancelled Response Message**:<br>C \| 16358622298593 \| 3407888 \| 7076 \| B \| 4277031652297829 \| 9] |

**Table 6.** Selected FIX-API test cases extracted from TestSuite.pcap.

| Test ID | Dest IP | Dest Port | Request Message | Response Messages / Expected Messages |
|---|---|---|---|---|
| 1 | 10.40.X.X | 17090 | **Login Request Message**: [8 = FIXT.1.1 \| 35 = A \| 34 = 1 \| 49 = BIA \| 50 = FX91 \| 52 = 13:49:33.730 \| 56 = BI \| 98 = 0 \| 108 = 300 \| 141 = Y \| 553 = USER_FIX \| 554 = XXX \| 1137 = 9 \| 10 = 190 \| ] | **Login Response Message**: [8 = FIXT.1.1 \| 35 = A \| 49 = BI \| 56 = BIA \| 34 = 1 \| 52 = 13:49:33.969 \| 98 = 0 \| 108 = 300 \| 141 = Y \| 1409 = 0 \| 1137 = 9 \| 10 = 219 \| ] |
| 2 | 10.40.X.X | 17090 | **Enter Order Request Message**: [8 = FIXT.1.1 \| 35 = D \| 34 = 7 \| 49 = BIA \| 50 = FX91 \| 52 = 14:16:54.058 \| 56 = BI \| 1 = XYZ \| 11 = 22102 \| 38 = 20 \| 40 = 2 \| 44 = 21.000 \| 54 = 1 \| 55 = ACS.E \| 59 = 0 \| 60 = 14:16:54.057 \| 70 = AAA \| 528 = A \| 10 = 048 \| ] | **Order Response Message**: [8 = FIXT.1.1 \| 35 = 8 \| 34 = 7 \| 49 = BI \| 52 = 14:16:54.335 \| 56 = BIA \| 57 = FX91 \| 1 = XYZ \| 6 = 0 \| 11 = 22102 \| 14 = 0 \| 17 = 1 \| 22 = M \| 37 = 6054100010C68 \| 38 = 20.0000000 \| 39 = 0 \| 40 = 2 \| 44 = 21.0000000 \| 54 = 1 \| 55 = ACS.E \| 59 = 0 \| 60 = 14:16:54.297 \| 70 = AAA \| 119 = 220.0000000 \| 150 = 0 \| 151 = 20.0000000 \| 528 = A \| 10 = 188 \| ] |
| 3 | 10.40.X.X | 17090 | **Replace Order Request Message**: [8 = FIXT.1.1 \| 35 = G \| 34 = 9 \| 49 = BIA \| 50 = FX91 \| 52 = 14:25:04.944 \| 56 = BI \| 11 = 22103 \| 37 = 6054100010C68 \| 38 = 20 \| 40 = 2 \| 41 = NONE \| 44 = 21.00000 \| 54 = 1 \| 55 = ACS.E \| 60 = 14:25:04.944 \| 70 = AAA \| 10 = 031 \| ] | **Order Replaced Response Message**: [8 = FIXT.1.1 \| 35 = 8 \| 34 = 9 \| 49 = BI \| 52 = 14:25:05.177 \| 56 = BIA \| 57 = FX91 \| 1 = ACC \| 6 = 0 \| 11 = 22103 \| 14 = 0 \| 17 = 2 \| 22 = M \| 37 = 6054100010C68 \| 38 = 20.0000000 \| 39 = 0 \| 40 = 2 \| 41 = 22102 \| 44 = 21.0000000 \| 48 = 70616 \| 54 = 1 \| 55 = ACS.E \| 59 = 0 \| 60 = 14:25:05.170 \| 70 = AAA \| 119 = 220.0000000 \| 150 = 5 \| 151 = 20.0000000 \| 528 = A \| 10 = 081 \| ] |
| 4 | 10.40.X.X | 17090 | **Cancel Order Request Message**: [8 = FIXT.1.1 \| 35 = F \| 34 = 12 \| 49 = BIA \| 50 = FX91 \| 52 = 14:38:28.668 \| 56 = BI \| 11 = 22104 \| 37 = 6054100010C68 \| 38 = 20 \| 41 = NONE \| 54 = 1 \| 55 = ACS.E \| 60 = 14:38:28.668 \| 10 = 252] | **Order Canceled Response Message**: [8 = FIXT.1.1 \| 35 = 8 \| 34 = 12 \| 49 = BI \| 52 = 14:38:28.900 \| 56 = BIA \| 57 = FX91 \| 1 = ACC \| 6 = 0 \| 11 = 22104 \| 14 = 0 \| 17 = 3 \| 22 = M \| 37 = 6054100010C68 \| 38 = 20.0000000 \| 39 = 4 \| 41 = 22103 \| 54 = 1 \| 55 = ACS.E \| 58 = Canceled by user \| 60 = 14:38:28.894 \| 70 = AAA \| 150 = 4 \| 151 = 0 \| 528 = A \| 10 = 092 \| ] |
| 5 | 10.40.X.X | 17090 | **Enter Order Request Message**: [8 = FIXT.1.1 \| 35 = D \| 34 = 15 \| 49 = BIA \| 50 = FX91 \| 52 = 14:44:21.663 \| 56 = BI \| 1 = XYZ \| 11 = 22106 \| 38 = 99 \| 40 = 2 \| 44 = 22.000 \| 54 = 1 \| 55 = GAR.E \| 59 = 3 \| 60 = 14:44:21.663 \| 70 = AAA \| 528 = A \| 10 = 115 \| ] | **Order Accepted Response Message**: [8 = FIXT.1.1 \| 35 = 8 \| 34 = 16 \| 49 = BI \| 52 = 14:44:21.892 \| 56 = BIA \| 57 = FX91 \| 1 = XYZ \| 6 = 0 \| 11 = 22106 \| 14 = 0 \| 17 = 7 \| 22 = M \| 37 = 1054100010C75 \| 38 = 99.0000000 \| 39 = 0 \| 40 = 2 \| 44 = 22.0000000 \| 48 = 74196 \| 54 = 1 \| 55 = GAR.E \| 59 = 3 \| 60 = 14:44:21.888 \| 70 = AAA \| 119 = 1089.0000000 \| 150 = 0 \| 151 = 99.0000000 \| 528 = A \| 10 = 099 \| ]<br>**Order Cancelled Response Message**: [8 = FIXT.1.1 \| 35 = 8 \| 34 = 17 \| 49 = BI \| 52 = 14:44:21.893 \| 56 = BIA \| 57 = FX91 \| 1 = XYZ \| 6 = 0 \| 11 = 22106 \| 14 = 0 \| 17 = 9 \| 22 = M \| 37 = 1054100010C75 \| 38 = 99.0000000 \| 39 = 4 \| 40 = 2 \| 44 = 22.0000000 \| 48 = 74196 \| 54 = 1 \| 55 = GAR.E \| 58 = Unsolicited Cancel \| 59 = 3 \| 60 = 14:44:21.888 \| 70 = AAA \| 150 = 4 \| 151 = 0 \| 528 = A \| 10 = 022 \| ] |

After generating test suites, the framework starts to replay them to the test environment. Each test case has an IP address and Port information and based on these details, it connects to the destination IP/Port automatically and begins to send each API message to the test server. After receiving actual messages from the test environment, it starts validation by comparing actual and expected messages. At the end of this process, the framework produces test result reports, as shown in Table 7. Furthermore, test result statistics, including pass/fail counts, unexpected API messages, missing API messages counts, and detailed validation errors, are calculated and reported to the related teams.

It is known that a tester in the company can run a test in one minute. Considering that there is an 8 h working period during the day, a tester can run a maximum of 480 tests. In the case of 480 test cases in a test suite, this means that 480/480 = 1 man/day is required. For business domain critical systems, each API message, with different field message combinations, should be covered in the test suite. This figure highlights the enormity of the effort to run test cases manually. With our framework, this period was reduced, ensuring significant cost savings. With this automation, regressions can be caught very quickly. The framework ensures 100% automation and cost savings. This study supports the reuse of test cases, increases the automation level by taking into account OD tests, and alleviates the difficulties inherent in manual testing. All told, the test effort is reduced.

**Table 7.** Selected produced test result reports.

| Test ID | Test Result Report | |
|---|---|---|
| 1 | TestDate | 16 May 2022 |
| | TestResult | Passed |
| | RequestMsg | [AEH1,XXXXXX,TR16092650,1] |
| | ExpectedResponseMsg | [TR16092650,0] |
| | ActualResponseMsg | [TR16092650,0] |
| 2 | TestDate | 16 May 2022 |
| | TestResult | Passed |
| | RequestMsg | [O \| 1a \| 10769 \| B \| 30 \| 4000 \| 0 \| 0 \| GMR \| REF \| 12345 \| 0 \| 1 \| 0] |
| | ExpectedResponseMsg | [A \| 163586118732 \| 1a \| 10769 \| B \| 9427703165229781 \| 30 \| 4000 \| 0 \| 0 \| GMR \| 1 \| REF \| 12345 \| 30 \| 0 \| 1 \| 0] |
| | ActualResponseMsg | [A \| 163586120732 \| 20 \| 10769 \| B \| 7899889898999997 \| 30 \| 4000 \| 0 \| 0 \| GMR \| 1 \| REF \| 12345 \| 30 \| 0 \| 1 \| 0] |
| 3 | TestDate | 16 May 2022 |
| | TestResult | Passed |
| | RequestMsg | [U \| 1a \| 3407884 \| 25 \| 0 \| 0 \| \| \| \| 0 \| 0] |
| | ExpectedResponseMsg | [U \| 163586167732 \| 3407884 \| 1a \| 10769 \| B \| 9427703165229781 \| 25 \| 4000 \| 0 \| 0 \| GMR \| 1 \| REF \| 12345 \| 25 \| 0 \| 1] |
| | ActualResponseMsg | [U \| 163586208910 \| 3407884 \| 1a \| 10769 \| B \| 7899889898999997 \| 25 \| 4000 \| 0 \| 0 \| GMR \| 1 \| REF \| 12345 \| 25 \| 0 \| 1] |
| 4 | TestDate | 16 May 2022 |
| | TestResult | Passed |
| | RequestMsg | [X \| 1a] |
| | ExpectedResponseMsg | [C \| 163586177305 \| 3407884 \| 10769 \| B \| 9427703165229781 \| 1] |
| | ActualResponseMsg | [C \| 163586228910 \| 3407884 \| 10769 \| B \| 7899889898999997 \| 1] |
| 5 | TestDate | 16 May 2022 |
| | TestResult | Passed |
| | RequestMsg | [O \| 3407888 \| 7076 \| B \| 22 \| 7000 \| 3 \| 0 \| GRM \| REF \| 12345 \| 0 \| 1 \| 0] |
| | ExpectedResponseMsg | A \| 1635862229859 \| 3407888 \| 7076 \| B \| 4277031652297829 \| 22 \| 7000 \| 3 \| 0 \| GRM \| 2 \| REF \| 12345 \| 22 \| 0 \| 1 \| 0]<br>C \| 16358622298593 \| 3407888 \| 7076 \| B \| 4277031652297829 \| 9] |
| | ActualResponseMsg | A \| 163586239910 \| 3407888 \| 7076 \| B \| 1278931652297829 \| 22 \| 7000 \| 3 \| 0 \| GRM \| 2 \| REF \| 12345 \| 22 \| 0 \| 1 \| 0]<br>C \| 163586239910 \| 3407888 \| 7076 \| B \| 1278931652297829 \| 9] |

Apart from this, our framework has filled the gap for business domain-specific API testing, because there is currently a lack in the software test automation industry for the systems which contain business application layer protocols, except for the HTTP protocol. Moreover, the existing tools cannot be integrated into all business domains due to their dependence on applications. Therefore, there was a need for business domain-specific APIs. Based on this need, this paper proposed a novel approach and developed a testing framework. The developed regression testing framework is being actively used by the chosen company for regression tests. It is automatically executed for every service release, and it reports the test results to the related teams. Furthermore, our approach uses network packets for regression testing. The use of information at the packet level ensures application independence, as well as a more generic application. Different client behaviors can also be easily obtained by using network packets. A network log file contains all the messages/behaviors belonging to a client. Moreover, our approach can provide for the use of real data packets obtained from the production environment. This will not only provide for the creation of test scenarios from the obtained real data, but will also enable us to capture the behavior of a client towards the server. As a result, the behavior of a client in the real environment can be automatically simulated in the test environment, with validation. In addition, a bug that occurs in the production environment can be quickly simulated in the test environment by replaying the network packets obtained from the production environment. Such a tool does not exist in the literature. Without any modification, this tool can be used for other trading companies which use FIX and OUCH API protocols as communication protocols, and it can also be applied to the other business domains as well.

### *4.3. Threats to Validity*

A threat to the validity of our framework is that we performed a case study in the context of a trading company. In this case, we have only studied two API protocols. Therefore, our evaluation is subject to an external validity threat [44]. This may impact the generality of the framework. More case studies can be conducted in different contexts to increase the generalizability of the results. We intend to focus on this issue in future studies with different API protocol tuning. However, both protocols used in our experiment are widely used, non-trivial API protocols in the financial domain.

Another threat to validity is that the proposed framework uses a pcap file, which contains communication data packets between client server applications on the network, as an input file. It is thus subject to both a construction and an internal threat to validity. The file should be created by using a data packet analyzer tool, but this file is standard and not specific to our study. It includes API request-response message pairs.

### *4.4. Lessons Learned*

In this section, we have listed some lessons learned during the usage of the framework. The framework provided not only improvements in testing efficiency (as mentioned above), but also significant qualitative benefits. First, the framework extracts all API messages included in the network log file (pcap file) in order to use them in regression testing. With this functionality, it also helps us to examine a pcap file. The framework extracts API messages in a more humanly readable format because it parses and extract API messages which can exist in a binary format. It enables us to speed up the examination of API messages in a pcap file obtained from the production environment. Next, this automation alleviates the burden faced by manual testers created by repeated testing. After starting to use our framework, they could place a greater focus on their non-automated test cases.

## 5. Summary and Conclusions

Software testing can be performed manually, but this approach is not convenient, especially for large-sized client-server applications, such as real-time systems, financial applications, etc. In these applications, there are thousands of different scenarios which should simulate the client’s behaviors, and these scenarios may not be performed with precision each time, due to human error. Therefore, this type of testing is time-consuming and less reliable. Thus, test automation becomes a critical issue, as there are large number of test cases that need to be performed repeatedly in large-scale business applications. According to the literature review, there is a lack of test automation, except for the HTTP protocol, for the systems which have their own application layer protocols. Hence, we propose a new approach in which the network packets are used for communication between clients and servers. The usage of information at the packet level ensures application independence. Furthermore, our approach supports the reuse of test cases and ensures 100% automation in API regression testing for client-server applications. Moreover, by removing the difficulties inherent in manual testing, this automation alleviates the burden faced by manual testers. Particularly regarding critical systems, in which a 100% passing score constraint is applied for release, this automation increases testing accuracy. We believe that this type of test automation tools is also promising in that it can easily be adapted for different domains.

As a future study, we would like to extend our evaluation with case studies derived from various types of business domains, adding new API protocols. We intend to pursue this path and to add and adapt new API protocols for different business domain such as telecommunications, etc. Moreover, we would like to use real data packets obtained from the production environment. This will not only provide the ability to create test scenarios from the obtained real data, but will also enable us to capture the behavior of a client towards the server. As a result, the behavior of a client in the real environment will be automatically simulated in the test environment.

**Author Contributions:** Conceptualization, E.D.D. and O.K.; data curation, E.D.D.; formal analysis, E.D.D.; methodology, E.D.D. and O.K.; project administration, O.K.; software, E.D.D.; supervision, O.K.; validation, O.K.; visualization, E.D.D.; writing—original draft, E.D.D.; writing—review and editing, O.K. All authors have read and agreed to the published version of the manuscript.

**Funding:** This study was supported via the “Regression Test Automation Framework” project by the Research and Development Center of the Borsa Istanbul A.S. of Turkey.

**Conflicts of Interest:** The authors declare no conflict of interest.

## Appendix A. Sample JSON Files Created for a Financial Domain API Protocol

This appendix is an extension of Section 3. The JSON files which are required in the validation phase are created for each API message. One of these files is shown below:

**Table A1.** Created Json file for OUCH API—Order Rejected Message.

**OrderRejectedMessage.json**

```
{
    “description”: “OrderRejectedMessage”,
    “properties”: {
        “msgType”: {
        “type”: “string”,
         “required”: “true”
        },
       “timeStamp”: {
       “type”: “long”,
       “required”: “false”
        },
        “orderToken”: {
        “type”: “string”,
        “required”: “false”
        },
       “rejectCode”: {
       “type”: “integer”,
        “required”: “true”
        }
    }
}
```

## References

1. What Is an API? Available online: https://www.mulesoft.com/resources/api/what-is-an-api (accessed on 10 May 2022).
2. Arcuri, A. RESTful API Automated Test Case Generation. In Proceedings of the 2017 IEEE International Conference on Software Quality, Reliability and Security (QRS), Prague, Czech Republic, 25–29 July 2017; pp. 9–20. [CrossRef]
3. Dalal, S.; Solanki, K. Challanges of Regression Testing: A Pragmatic Perspective. *Int. J. Adv. Res. Comput. Sci.* **2018**, *9*, 499–503. [CrossRef]
4. Minhas, N.M.; Petersen, K.; Börstler, J.; Wnuk, K. Regression testing for large-scale embedded software development–Exploring the state of practice. *Inf. Softw. Technol.* **2020**, *120*, 106254. [CrossRef]
5. Khari, M.; Kumar, P. An extensive evaluation of search-based software testing: A review. *Soft Comput.* **2019**, *23*, 1933–1946. [CrossRef]
6. Khari, M. Emprical Evaluation of Automated Test Suite Generation and Optimization. *Arab. J. Sci. Eng.* **2019**, *45*, 2407–2423. [CrossRef]
7. Jain, A.; Tayal, D.K.; Khari, M.; Vij, S. A novel method for test path prioritization using centrality measures. *Int. J. Open Source Softw. Processes* **2016**, *7*, 19–38. [CrossRef]
8. Taley, D.S.; Pathak, B. Comprehensive Study of Software Testing Techniques and Strategies: A Review. *Int. J. Eng. Res.* **2020**, *9*, IJERTV9IS080373. [CrossRef]
9. Gonçalves, W.F.; de Almeida, C.B.; de Araújo, L.L.; Ferraz, M.S.; Xandú, R.B.; de Farias, I. The influence of human factors on the software testing process: The impact of these factors on the software testing process. In Proceedings of the 2017 12th Iberian Conference on Information Systems and Tech-nologies (CISTI), Lisbon, Portugal, 21–24 June 2017; pp. 1–6. [CrossRef]
10. Lam, W.; Shi, A.; Oei, R.; Zhang, S.; Ernst, M.D.; Xie, T. Dependent-test-aware regression testing techniques. In Proceedings of the International Symposium on Software Testing and Analysis (ISSTA), Virtual Event, USA, 18 July 2020; pp. 298–311. [CrossRef]
11. Arcuri, A. RESTful API Automated Test Case Generation with EvoMaster. *ACM Trans. Softw. Eng. Methodol.* **2019**, *28*, 1–37. [CrossRef]
12. Han, X.; Zhang, N.; He, W.; Zhang, K.; Tang, L. Automated Warship Software Testing System Based on LoadRunner Automation API. In Proceedings of the 2018 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C), Lisbon, Portugal, 16–20 July 2018; pp. 51–55. [CrossRef]

13. Ed-douibi, H.; Izquierdo, J.L.C.; Cabot, J. Automatic Generation of Test Cases for REST APIs: A Specification-Based Approach. In Proceedings of the 2018 IEEE 22nd International Enterprise Distributed Object Computing Conference (EDOC), Stockholm, Sweden, 16–19 October 2018; pp. 181–190. [CrossRef]
14. Viglianisi, E.; Dallago, M.; Ceccato, M. Resttestgen: Automated Black-Box Testing of RESTful APIs. In Proceedings of the 2020 IEEE 13th International Conference on Software Testing, Validation and Verification (ICST), Porto, Portugal, 24–28 October 2020; pp. 142–152. [CrossRef]
15. Atlidakis, V.; Godefroid, P.; Polishchuk, M. RESTler: Stateful REST API Fuzzing. In Proceedings of the 41st International Conference on Software Engineering (Montreal, Quebec, Canada) (ICSE), Montreal, QC, Canada, 25–31 May 2019; IEEE Press: Piscataway, NJ, USA, 2019; pp. 748–758. [CrossRef]
16. Postman | API Development Environment. Available online: https://www.getpostman.com/ (accessed on 21 August 2022).
17. vREST–Automated REST API Testing Tool. Available online: https://vrest.io/ (accessed on 24 August 2022).
18. The World's Most Popular Testing Tool | SoapUI. Available online: https://www.soapui.org (accessed on 21 August 2022).
19. Chen, Y.; Gao, Y.; Zhou, Y.; Chen, M.; Ma, X. Design of an Automated Test Tool Based on Interface Protocol. In Proceedings of the 2019 IEEE 19th International Conference on Software Quality, Reliability and Security Companion (QRS-C), Sofia, Bulgaria, 22–26 July 2019; pp. 57–61. [CrossRef]
20. Demircioğlu, E.D.; Kalıpsız, O. Test Case Generation Framework for Client-Server Apps: Reverse Engineering Approach. In *Computational Science and Its Applications–ICCSA 2020*; Gervasi, O., Murgante, B., Misra, S., Garau, C., Blečić, I., Taniar, D., Apduhan, B.O., Rocha, A.M.A.C., Tarantino, E., Torre, C.M., et al., Eds.; Lecture Notes in Computer Science; Springer: Cham, Switzerland, 2020; Volume 12253. [CrossRef]
21. TCP and UDP. Available online: https://www.cs.nmt.edu/~{}risk/TCP-UDP%20Pocket%20Guide.pdf (accessed on 20 June 2022).
22. Wireshark Tool. Available online: https://www.wireshark.org/ (accessed on 1 May 2022).
23. Tcpdump. Available online: https://www.tcpdump.org/ (accessed on 10 August 2022).
24. Arkime. Available online: https://arkime.com/index#home (accessed on 10 August 2022).
25. Isha; Sharma, A.; Revathi, M. Automated API Testing. In Proceedings of the 2018 3rd International Conference on Inventive Computation Technologies (ICICT), Coimbatore, India, 15–16 November 2018; pp. 788–791. [CrossRef]
26. Research on Market. Available online: https://www.researchandmarkets.com/ (accessed on 5 April 2022).
27. Garousi, V.; Tasli, S.; Sertel, O.; Tokgoz, M.; Herkiloglu, K.; Arkin, H.F.; Bilir, O. Automated Testing of Simulation Software in the Aviation Industry: An Experience Report. *IEEE Softw.* **2019**, *36*, 63–75. [CrossRef]
28. Wang, J.; Bai, X.; Li, L.; Ji, Z.; Ma, H. A Model-Based Framework for Cloud API Testing. In Proceedings of the 2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC), Turin, Italy, 4–8 July 2017; pp. 60–65. [CrossRef]
29. Godefroid, P.; Lehmann, D.; Polishchuk, M. Differential regression testing for REST APIs. In Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis, Virtual Event, USA, 18 July 2020; pp. 312–323. [CrossRef]
30. Xu, D.; Xu, W.; Kent, M.; Thomas, L.; Wang, L. An Automated Test Generation Technique for Software Quality Assurance. *IEEE Trans. Reliab.* **2015**, *64*, 247–268. [CrossRef]
31. Sneha, K.; Malle, G.M. Research on software testing techniques and software automation testing tools. In Proceedings of the 2017 International Conference on Energy, Communication, Data Analytics and Soft Computing (ICECDS), Chennai, India, 1–2 August 2017; pp. 77–81. [CrossRef]
32. Apache JMeter. Available online: https://jmeter.apache.org/ (accessed on 16 June 2022).
33. GoReplay. Available online: https://github.com/buger/goreplay (accessed on 10 June 2022).
34. Pollyjs. Available online: https://github.com/Netflix/pollyjs (accessed on 10 June 2022).
35. Node Replay. Available online: https://www.npmjs.com/package/replay (accessed on 15 June 2022).
36. Bhateja, N. A Study on Various Software Automation Testing Tools. *Int. J. Adv. Res. Comput. Sci. Softw. Eng.* **2015**, *5*, 1250–1252.
37. Liu, Z.; Chen, Q.; Jiang, X. A Maintainability Spreadsheet-Driven Regression Test Automation Framework. In Proceedings of the 2013 IEEE 16th International Conference on Computational Science and Engineering, Sydney, Australia, 3–5 December 2013; pp. 1181–1184. [CrossRef]
38. Malhotra, R.; Khari, M. Heuristic search-based approach for automated test data generation: A survey. *Int. J. Bio-Inspired Comput.* **2013**, *5*, 1–18. [CrossRef]
39. Khari, M.; Sinha, A.; Verdu, E.; Crespo, R.G. Performance analysis of six meta-heuristic algorithms over automated test suite generation for path coverage-based optimization. *Soft Comput.* **2020**, *24*, 9143–9160. [CrossRef]
40. Khari, M.; Kumar, P. An effective meta-heuristic cuckoo search algorithm for test suite optimization. *Informatica* **2017**, *41*, 363–377.
41. FIX API Protocol. Available online: https://www.borsaistanbul.com/files/genium-inet-fix-protocol-specification.pdf (accessed on 18 July 2022).
42. Ouch-API Protocol. Available online: https://www.borsaistan-bul.com/files/OUCH_ProtSpec_BIST_va2414.pdf (accessed on 18 July 2022).
43. FIX Protocol. Available online: https://www.fixtrading.org/ (accessed on 10 August 2022).
44. Wohlin, C.; Runeson, P.; Host, M.; Ohlsson, M.; Regnell, B.; Wesslen, A. *Experimentation in Software Engineering*; Springer: Berlin/Heidelberg, Germany, 2012.