

Test Case Generation Framework for Client-Server Apps: Reverse Engineering Approach

Emine Dumlu Demircioğlu[1(✉)] and Oya Kalıpsız[2]

[1] Yıldız Technical University-Borsa Istanbul, 34467 Istanbul, Turkey
eminedumlu@gmail.com
[2] Yıldız Technical University, 34220 Istanbul, Turkey
oyakalipsiz@gmail.com

Abstract. In this paper, the message-driven test case generation framework is proposed for client-server architecture systems which are communicated with each other by using API messages. Basically proposed framework automatically generates test cases, which are extracted with reverse engineering mechanism from captured network packets between client-server communications over networks. We propose a novel approach to record packets for regression testing. The studying is motivated due to the fact that there is a lack of regression test automation framework in a specific domain: client-server apps which uses API messages for the communication, such as financial applications, banking applications, trading systems, real-time apps. Our approach supports the reuse of test cases and will ensure 100% automation from test case generation to execution in API regression testing for client-server applications. Our work reports the significant reuse capability and reduction of effort on a real-word financial system example.

Keywords: Software testing frameworks · Automation testing · API testing

1 Introduction

Software failures may cause havoc at banks, airlines, trading systems in the stock exchange and resulting in billions of pounds of damage. To avoid this and to develop reliable and high- quality software products, software testing is a major step within the software development process. Testing can be implemented either manually or automatically. Especially in companies using incremental development process model, the software should be developed incrementally. New requirements at each increment or changes necessitate changes in the software. Thus, testers should verify the new function as soon as possible; and before going live, effective regression tests must be applied [7]. Regression testing is selective re-testing of a system to verify that changes in the software systems have not caused side effect [10]. Accomplishing this manually is very difficult and laborious particularly in medium-size or large-size business software systems, since such systems contain hundreds of business functions and there is a need to consider a huge amount of test scenarios. So the manual creation and execution of

© Springer Nature Switzerland AG 2020
O. Gervasi et al. (Eds.): ICCSA 2020, LNCS 12253, pp. 674–683, 2020.
https://doi.org/10.1007/978-3-030-58814-4_54

these test scenarios would necessitate an enormous investment in human resources [5, 12]. It is also difficult to do manual API tests repeatedly because manual API testing is very time-consuming, error-prone and tedious [3, 13, 14]. Moreover, based on common experience, it is known that the success of manual testing can be affected by the personal problems or the mood of the tester on any given day. These can affect the productivity and can harm the whole project; the effects are from delay, lack of attention, decreased focus [15, 16].Therefore, we want to eliminate the human impact from repetitive testing for the systems based on client-server architecture and to speed up the release cycle. Our aim is to improve the efficiency of software testing and shorten the test cycle.

Test automation should be considered as end to end automation from test case generation to execution and defect reporting [2]. Studying on how to automatically generate test cases is a key problem in automatic software testing [11, 17]. In this study, we decided to automate the manual API regression testing process for the systems based on client-server architecture. To be able to do this, we have divided our studying into the three parts: The first part is test case generation automatically, the second one is test execution of created test cases automatically, and the last part is validation and defect reporting. Our approach supports the reuse of test cases and will ensure 100% automation in API regression testing for client-server applications. Such automation takes the cumbersome of doing manual tests from testers. In this paper, how test cases are generated automatically will be shown.

In Sect. 2, we continue by providing necessary background information for understanding the proposed systems and discuss the related work. Next, Sect. 3 describes the proposed framework, Sect. 4 describes the empirical studies, and Sect. 5 presents concluding remarks and future work.

2 Background and Related Work

Figure 1 shows the targeting system architecture. There can be many client applications connecting to server app on the predefined ports. There is no GUI for the targeting server app. An established connection from client to server is uniquely identified by the combination of client-side and server-side IP/Port pairs. The server and clients talk over TCP or UDP using a well-defined messaging format. TCP is a reliable connection-oriented protocol designed to send data packets over the Internet. UDP is an unreliable connectionless protocol that run on top of IP. Communication between client and server is established by using TCP or UDP data packets. Each data packet has header and payload sections. The header section contains Source/Destination Port information. In addition, the TCP header also has sequence numbers to keep track of how much data has been sent. The payload section contains data (API messages) carried for the application [11].

The approach uses information at the packet level and produces test cases. So, we will use the Wireshark tool to obtain communication data packets between client-server applications. This tool is a free and open source packet analyzer tool.

Packets between client and server are captured in a PCAPs (Packet Captures) file by using Wireshark. The PCAPs file is a data file composed of packet data of network traffic and captures client's profile (actual usage data produced by the client). Each packets in the

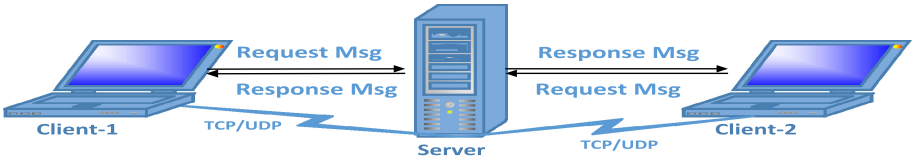

Fig. 1. Client-server architecture

PCAPs file displays the communication data packets including API messages between client and server applications.

A large number of large enterprises such as airlines (e.g. real-time flight updates), banks (e.g. mobile apps) or government apps have been using APIs to provide services such as data queries and functionalities [21]. However, dynamic characteristics of APIs raise new threats to the quality of systems. A defect in an open API may cause software failures in a large scale. API testing is thus becoming necessary to ensure the quality and reliability of APIs for individual services. API testing is a type of testing that involves testing Application Programming Interface and of great importance to ensure functionality, reliability and delivery of business logic. API testing is performed at the message layer because APIs lack a graphical user interface. The existing API testing tools are mostly for Web APIs which can be accessed using HTTP protocol; especially REST (Representational State Transfer) API are most widely employed APIs [4, 6, 14]. REST-based web applications basically use JSON or XML formats to deliver REST API responses [14]. These are not suitable for our proposed targeting systems and its message structure in terms of usability. The proposed system is not a Web Server and does not support JSON or XML formats. It can have its own message structure and can have its own application layer protocols instead of HTTP protocol. In many cases, the input messages to an API call depends on the response messages obtained from other API call. Existing API testing tools does not support extracting input from response [14], hence not provide 100% automation. In proposed framework, we will parse response message coming from server and use it for the next input message when specifying input data. So we will increase the automation.

According to the purpose of testing, there are different types of test automation tools. For example, if there is a web application and functional tests are conducted to validate the application, the Selenium tool can be selected. To do a functional test, the tester writes test scripts code using Selenium IDE and performs automation testing or JMeter IDE can be selected to conduct load testing. These are mostly for web, mobile and desktop applications. In order to implement third-party testing tools, the tester should develop test script code to adapt the existing test automation to the desired system. Test script code requires a maintenance process which is a costly one [3] and is very difficult for the tester. The tester needs to update the test script code when there are changes in any requirement. We would like to eliminate test script code development and the maintenance process.

In addition, existing API tools have some shortcomings like high overhead, long test data preparation and difficulty in reuse [18–20]. Creation of test cases is done manually by existing API tools. TCP packets which payload section contains API messages are send to the targeting systems. But the API messages in the test cases can have volatile

values. So these values should be extracted and changed not to face assertion failures when executing test cases by using such tools. Our approach solves this problem.

In this endeavor, we have proposed a regression testing framework for large-scale client-server applications based on a capturing packet at the network level, followed with the analysis and re-constructing of the information in the messages. Thus, this paper suggests a reverse engineering approach for information at the packet level to create regression testing test cases. The idea here is to capture the usage data at packet level to ensure they are more application independent. Actually, this process is the easiest method to obtain different requests from various clients.

3 Message-Driven Regression Test Case Generation Framework

In this section, we have described the main workflow of proposed framework. It is developed in Java by using pkts.io open source library in [1] to read the PCAPs file. The input file to the framework is a PCAPs file in which data packets between client-server apps are captured. This file is then analyzed, and reverse engineering is used to determine values sent back and forth between the server and the client. Basically, the framework reads the PCAPs file and produces test suite, which is a collection of test cases, to replay PCAPs file through the targeting test system. 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. The framework does this mapping according to the Source IP/Port, Destination IP/Port information to create test suite.

When the time data packet in the PCAPs file is read, the framework starts to analyze the packet. Initially, it detects which transport layer protocols (TCP or UDP) were used in the communication. If TCP is used, the framework analyses the packet to determine whether the packet is retransmitted. Each packet has sequence number and payload section which stores the data carried for the packet. To be able to detect whether or not the packet is retransmitted, sequence numbers on the packet and payload section length are used. The sequence number of the next package is calculated by summing the sequence number of the current packet and the length of the payload section. If the sequence number of the next packet is less than calculated sequence number, this packet is marked as a retransmitted packet and retransmitted data portion is calculated. This means that the original version of the retransmitted data portion is already stored in the memory. Hence, we ignored retransmitted packets when creating the list of test cases. Since UDP is a connectionless protocol, retransmissions are not handled. After detecting communication protocol, heuristic search techniques are applied to detect which application layer protocols for the targeting systems are used. The determined message patterns according to the targeting systems are searched in the payload section of the packet. For example, according to the FIX protocol [8], each protocol message starts with "8 = FIXT" and ends with "10 = XYZ". When determining the used protocol (for example FIX), below search pattern in java can be applied:

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

Other search patterns can be specified for the targeting systems. After detecting used application layer protocols, the payload section which contains data in the packet is extracted and parsed according to the predefined messaging format of the software

system in question. When composing a message, we took into consideration the fact that the payload section may have been fragmented, because the payload section has the maximum size of data payload. If the fragmentation is detected in the packet, the payload section will first be stored in the memory and then reassembled with the next packet's payload section. After reassembly, data in the payload section is parsed according to the predefined messaging format of the software system in question. Finally, information such as arrival time, Src/Dst IP/Port of the packet, and messages in that packet will be included within the packet. Consequently, the final obtained packet is mapped to understand whether the packet is a client message which is a request message sent by the client to the server, or a server message which is a response message sent by the server to the client. Given the previous discussion our basic algorithm should now be easy to follow. Algorithm 1 describes the main createTestSuite routine that loops until the end of PCAPs file, making a call to getNextPacket to get a packet in the file once per iteration. After extracting packet details like Arrival Time, Src/Dst IP/Port and used protocol (TCP or UDP) by calling extractPacketDetail (packet), it makes a call to constructMessagesInThePacket to create messages in the packet according to the pre-defined messaging format of the targeting software system. Finally, we add the created messages to the test suite by mapping as request or response messages making a call to addToTestSuite (messages). These approach can be applied to different business domains by implementing only constructMessagesInThePacket (packet) method for the targeting system's message structure. With this approach, test cases can be generated at packet level automatically. This was the first motivation of our study.

Algorithm 1 – createTestSuite

Input: pcapFile: Pcap File

1. **repeat**
2. packet ← getNextPacket();
3. extractPacketDetail(packet);
4. messages ← constructMessagesInThePacket(packet);
5. addToTestSuite(messages);
6. **until** end of pcapFile

Some protocol contains various packets which are related to an environment thus they cannot be reused. For example, HTTP response contains last-modified time of target page, response date, or session information which are volatile. Saving whole packets without any analysis may cause assertion failures because of these volatile values. To realize record and replay automation, it is required to extracting these kinds of environment related values. Therefore, the extraction of application layer packets is done instead of saving and resending whole TCP/UDP packets. Hence, in the second part of this studying which will be replaying part, volatile values will be identified and extracted for these test cases not to face assertion failures.

4 Case Study

We conducted a set of empirical studies to evaluate the usability of the proposed test automation framework and applied it to the trading system in the Borsa Istanbul Stock Exchange. FIX and OUCH Protocols are used as an API Protocols by trading system.

Table 1. Selected Sample Test Cases from Testsuite.pcap File for OUCH protocol

Test id	Send message from client to server	Receive messages from server
1	Soup packet type: login request [ABCEH4,XXXXXX,TR2602280,3]	Soup packet type: login accepted [TR2602280,0]
2	Soup packet type: client heartbeat	Soup packet type: server heartbeat
3	Soup packet type: unsequenced data [**O**, 253588, 70798, Buy, 10, 600,,NoChange, MRG, REF, 12345, 0, Client, Normalhours]	Soup packet type: sequence data [**A**, 253588, 70798, Buy, 7153059811473522662, NoChange, MRG, OnBook, REF, 12345, 10, 0, Client, Normalhours]
4	Soup packet type: client heartbeat	Soup packet type: server heartbeat
5	Soup packet type: unsequenced data [**U**, 253588, 253589, 100, 0, NoChange, 0, Empty]	Soup packet type: sequence data [**U**, 253589, 253588, 70798, Buy, 7153059811473522662, 100, 600, NoChange, MRG, OnBook, REF, 12345, 100, 0, Client]
6	Soup packet type: unsequenced data [**X**, 253588]	Soup packet type: sequence data [**C**, 253590, 70798, Buy, 7153059811473522662, Canceled]
7	Soup packet type: unsequenced data [**Y**, 70798, Buy, 7153059811473522662]	Soup packet type: sequence data [**J**, Rejected]
8	Soup packet type: logout request	Soup packet type: logout response

FIX is the standard protocol for exchanges within the finance world as it is considered the standard method to communicate trading information electronically between brokers, buy-side institutions, and markets. The FIX Protocol language is comprised of a series of messaging specifications used in trade communications [8]. OUCH is a simple protocol that allows trading users to enter orders, cancel and replace existing orders and receive executions. The OUCH protocol is composed of logical messages passed between the OUCH host and the client application [9]. There is a huge number of functionalities/scenarios in order to test systems which use the FIX and OUCH protocols as a message infrastructure. These two protocols have completely different message structures and these are application layer protocols.

Table 2. Selected Sample Test Cases from TestSuite.pcap File for FIX protocol

Test id	Send message from client to server	Receive messages from server
1	***8 = FIXT.1.1*** 9 = 160 **35 = D** 34 = 39 49 = BIABC 50 = F61 52 = 20190719-14:31:06.843 56 = TEST 1 = ACC 11 = 40359 38 = 20 40 = 2 44 = 11.000 54 = 1 55 = XXX.E 59 = 0 60 = 20190719-14:31:06.843 70 = AAA 528 = A ***10 = 044***	***8 = FIXT.1.1*** 9 = 264 35 = 8 34 = 45 49 = TEST 52 = 20190719-14:31:07.002 56 = BIABC 57 = F61 1 = ACC 6 = 0 11 = 40359 14 = 0 17 = 152 22 = M 37 = 6363BF0000008101 38 = 20.0000000 39 = 0 40 = 2 44 = 11.0000000 48 = 70616 54 = 1 55 = XXX.E 151 = 20.0000000 528 = A ***10 = 032***
2	**8 = FIXT.1.1** 9 = 160 **35 = D** 34 = 40 49 = BIABC 50 = F61 52 = 20190719-14:31:09.810 56 = TEST 1 = ACC 11 = 40360 38 = 20 40 = 2 44 = 11.000 54 = 1 55 = XXX.E 59 = 0 60 = 20190719-14:31:09.810 70 = AAA 528 = A **10 = 022**	**8 = FIXT.1.1** 9 = 264 35 = 8 34 = 48 49 = TEST 52 = 20190719-14:31:09.967 56 = BIABC 57 = F61 1 = ACC 6 = 0 11 = 40360 14 = 0 17 = 157 22 = M 37 = 6363BF0000008105 38 = 20.0000000 39 = 0 40 = 2 44 = 11.0000000 48 = 70616 54 = 1 55 = XXX.E 59 = 0 60 = 20190719-14:31:09.962 70 = AAA 119 = 220.0000000 150 = 0 151 = 20.0000000 528 = A **10 = 054**
3	**8 = FIXT.1.1** 9 = 188 **35 = G** 49 = BIABC 56 = TEST 50 = F61 34 = 42 52 = 20190719-14:31:16.492 43 = N 37 = 6363BF0000008105 41 = NONE 11 = 40362 55 = XXX.E 60 = 20190719-14:31:16.492 38 = 27 59 = 0 54 = 1 40 = 2 44 = 11.000 **10 = 035**	**8 = FIXT.1.1** 9 = 000302 35 = 8 49 = TEST 56 = BIABC 34 = 50 57 = F61 52 = 20190719-14:31:16.645 37 = 6363BF0000008105 11 = 40362 17 = 89485 150 = 5 39 = 0 1 = ACC 55 = XXX.E 48 = 423253 22 = M 54 = 1 38 = 27.0000000 40 = 2 44 = 11.000 59 = 0 151 = 27.0000000 14 = 0 6 = 0 60 = 20190719-14:31:16.492 **10 = 166**
4	**8 = FIXT.1.1** 9 = 149 **35 = F** 34 = 41 49 = BIABC 50 = F61 52 = 20190719-14:31:15.492 56 = TEST 11 = 40361 37 = 6363BF0000008105 38 = 20 41 = NONE 54 = 1 55 = XXX.E 60 = 20190719-14:31:15.492 **10 = 184**	**8 = FIXT.1.1** 9 = 224 35 = 8 34 = 49 49 = TEST 52 = 20190719-14:31:15.645 56 = BIABC 57 = F61 1 = ACC 6 = 0 11 = 40361 14 = 0 17 = 158 22 = M 37 = 6363BF0000008105 38 = 20.0000000 39 = 4 41 = 40360 48 = 70616 54 = 1 55 = XXX.E 60 = 20190719-14:31:15.644 70 = AAA 150 = 4 151 = 0 528 = A **10 = 252**

4.1 Preparation of Data

Market experts and testers in Borsa Istanbul designate a list of test cases via exploratory testing. Test case design conducted manually based on human knowledge is known as exploratory testing [2]. While testers execute prepared test cases manually towards to the system under test (SUT), the network packets are captured via the Wireshark tool. Thus, we obtained PCAPs files and introduced them to the proposed framework.

Another method of obtaining the PCAPs file is getting it from production environment instead of creating it manually. So real data is obtained and test cases are created from real data.

4.2 Analysis of Proposed Framework

In this study, we have one PCAPs file named TestSuite.pcap which contains both FIX and OUCH protocol messages. The proposed framework absorbs this file as input and produces separate test suites for each different protocols. Table 1 demonstrates the selected output of the test case generation framework for the OUCH protocol. 8 test cases are selected to show output of the framework. Each line shows the send message from client to server and received messages from server to client as a response. In other words, when a tester deals with test case 1, if a Login Request packet with the specified data within the table is sent to the server; the Login Accepted Packet is received as a response. In test case 3, when Unsequenced Data Packet with the specified data within the table is sent to the server, the server should send the Sequence Data Packet with the specified values in the table. The first value of the message data is "O". It shows message type of the OUCH message. According to the OUCH protocol specification, "O" means "Enter Order" message. The response to a successful Enter Order is an Order Accepted message. According to the table, the first value of the response message is "A" which means "Order Accepted Message". The list of OUCH protocol message structure is available in [9]. Table 2 also illustrates the selected test cases produced from TestSuite.pcap file for the FIX protocol.

5 Concluding Remarks and Future Work

Software testing is a means of detecting bugs within the process of evaluating whether the software product meets specified requirements to ensure that software products have no defects and high-quality software products can be produced. Repetitive manual testing is time consuming, tedious and error prone. Automation testing overcomes the difficulties of manual testing and is the best method to improve accuracy, test coverage, efficiency and speed up the testing process. Hence, we have aimed to automate the manual API regression tests for client-server applications. With this objective, the message-driven regression test case generation framework was developed. With this framework, test case generation has become automatically for the systems based on client server architecture such as trading systems in the stock exchange and banking applications. The rationale behind this message-driven test automation framework is the fact that test data is stored in the "PCAPs File. With this framework, we increase the automation in API testing by generating test cases automatically. We can replay whole PCAPs files to the system being tested and subsequently simulate and validate the client's behavior by using real data in production environment using production PCAPs file towards the system under test. Replaying and validating part will be next part of the study. We are planning to replay the whole PCAPs file to the SUT with validation by using the produced test case list from the proposed framework. In order to accomplish this, we intend to determine validation methods for the system.

Acknowledgments. This study was supported by Borsa Istanbul of Turkey.

References

1. Java Library for reading and writing PCAPs. https://github.com/aboutsip/pkts
2. Garousi, V., Elberzhager, F.: Test automation: not just for test execution. IEEE Softw. **34**(2), 90–96 (2017). https://doi.org/10.1109/ms.2017.34
3. Liu, Z., Chen, Q., Jiang, X.: A maintainability spreadsheet-driven regression test automation framework. In: IEEE 16th International Conference on Computational Science and Engineering, Sydney, NSW, pp. 1181–1184 (2013). https://doi.org/10.1109/cse.2013.175
4. Sneha, K., Malle, G.M.: Research on software testing techniques and software automation testing tools. In: 2017 International Conference on Energy, Communication, Data Analytics and Soft Computing (ICECDS), Chennai, pp. 77–81 (2017). https://doi.org/10.1109/icecds.2017.8389562
5. Sharma, R.M.: Quantitative analysis of automation and manual testing. Int. J. Eng. Innovative Technol. (IJEIT) **4**(1), 1–6 (2014)
6. Bhateja, N.: A study on various software automation testing tools. Int. J. Adv. Res. Comput. Sci. Softw. Eng. **5**(6), 1250–1252 (2015)
7. Sultanía, A.K.: Developing software product and test automation software using agile methodology. In: Proceedings of the 2015 Third International Conference on Computer, Communication, Control and Information Technology (C3IT), Hooghly, pp. 1–4 (2015)
8. FIX Protocol. https://www.borsaistanbul.com/docs/default-source/nasdaq-dokuman/genium-inet-fix-protocol-specification-.pdf?sfvrsn=46
9. OUCH Protocol. https://www.borsaistanbul.com/docs/default-source/nasdaq-dokuman/bistech-ouch-protocol-specification-va-2410.pdf?sfvrsn=8
10. Dalal, S., Solanki, K.: Challenges of regression testing: a pragmatic perspective. Int. J. Adv. Res. Comput. Sci. **9**(1), 499–503 (2018)
11. Itkonen, J., Mantyla, M.V.: Are test cases needed? Replicated comparison between exploratory and test-case based software testing. Empirical Softw. Eng. **19**(2), 303–342 (2014). https://doi.org/10.1007/s10664-013-9266-8
12. Han, X., Zhang, N., He, W., Zhang, K., Tang, L.: Automated warship software testing system based on loadrunner automation API. In: 2018 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C), Lisbon, pp. 51–55 (2018)
13. Bangare, S., Bangare, S., Borse, S., Bangare, P., Nandedkar, S.: Automated API testing approach. Int. J. Eng. Sci. Technol. **4**, 1–4 (2012)
14. Isha, A.S., Revathi, M.: Automated API testing. In: 2018 3rd International Conference on Inventive Computation Technologies (ICICT), Coimbatore, India, pp. 788–791 (2018). https://doi.org/10.1109/icict43934.2018.9034254
15. Gonçalves, W., de Almeida, C.B., Araújo, L., Ferraz, M., Xandú, R., Junior, I.: The impact of human factors on the software testing process: the importance of these factors in a software testing environment. J. Inf. Syst. Eng. Manage. **2**, 24 (2017). https://doi.org/10.20897/jisem.201724
16. Engström, E., Runeson, P.: A qualitative survey of regression testing practices. In: Ali Babar, M., Vierimaa, M., Oivo, M. (eds.) PROFES 2010. LNCS, vol. 6156, pp. 3–16. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-13792-1_3
17. Fernandez-Sanz, L., Misra, S.: Practical application of UML activity diagrams for the generation of test cases. Proc. Romanian Acad. Ser. A **13**(3), 251–260 (2012)

18. Xu, D., Xu, W., Kent, M., Thomas, L., Wang, L.: An automated test generation technique for software quality assurance. IEEE Trans. Reliab. **64**(1), 247–268 (2015). https://doi.org/10.1109/tr.2014.2354172
19. Chen, Y., Gao, Y., Zhou, Y., Chen, M., Ma, X.: Design of an automated test tool based on interface protocol. In: 2019 IEEE 19th International Conference on Software Quality, Reliability and Security Companion (QRS-C), Sofia, Bulgaria, pp. 57–61 (2019). https://doi.org/10.1109/qrs-c.2019.00024
20. Aiya, K.V., Verma, H.: Keyword driven automated testing framework for web application. In: 2014 9th International Conference on Industrial and Information Systems (ICIIS), Gwalior, pp. 1–6 (2014). https://doi.org/10.1109/iciinfs.2014.7036478
21. Wang, J., Bai, X., Li, L., Ji, Z., Ma, H.: A model-based framework for cloud API testing. In: 2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC), Turin, pp. 60–65 (2017). https://doi.org/10.1109/compsac.2017.24