RESTest: Automated Black-Box Testing of RESTful Web APIs

Alberto Martin-Lopez
alberto.martin@us.es
SCORE Lab, I3US Institute,
Universidad de Sevilla
Seville, Spain

Sergio Segura
sergiosegura@us.es
SCORE Lab, I3US Institute,
Universidad de Sevilla
Seville, Spain

Antonio Ruiz-Cortés
aruiz@us.es
SCORE Lab, I3US Institute,
Universidad de Sevilla
Seville, Spain

ABSTRACT

Testing RESTful APIs thoroughly is critical due to their key role in software integration. Existing tools for the automated generation of test cases in this domain have shown great promise, but their applicability is limited as they mostly rely on random inputs, i.e., fuzzing. In this paper, we present RESTest, an open source black-box testing framework for RESTful web APIs. Based on the API specification, RESTest supports the generation of test cases using different testing techniques such as fuzzing and constraint-based testing, among others. RESTest is developed as a framework and can be easily extended with new test case generators and test writers for different programming languages. We evaluate the tool in two scenarios: offline and online testing. In the former, we show how RESTest can efficiently generate realistic test cases (test inputs and test oracles) that uncover bugs in real-world APIs. In the latter, we show RESTest's capabilities as a continuous testing and monitoring framework. Demo video: https://youtu.be/1f_tjdkaCKo.

KEYWORDS:

REST,
black-box testing,
web APIs

1 INTRODUCTION

Web APIs enable the consumption of services and data over the network, typically using web services. Modern web APIs generally adhere to the REpresentational State Transfer (REST) architectural style [16], being referred to as RESTful web APIs. *RESTful web APIs* are usually decomposed into multiple RESTful web services [23], each of which implements one or more create, read, update or delete (CRUD) operations over a specific resource (e.g., a video in the YouTube API). RESTful APIs are commonly described using languages such as the OpenAPI Specification (OAS) [4]. OAS provides a structured description of a RESTful web API that allows both humans and computers to discover and understand the capabilities of a service without requiring access to the source code or additional documentation. The widespread use of RESTful APIs is reflected in the size of popular API directories such as ProgrammableWeb [7], currently indexing over 24K APIs.

RESTful APIs have become key for the development and seamless integration of heterogeneous systems, therefore their testing deserves special attention. A faulty API can have a huge impact in the many applications using it. In recent years, several approaches and tools have been proposed to automate the testing of RESTful APIs. When the source code is available, white-box approaches can be applied [11]. However, this is not often the case for this type of systems, and so it is necessary to resort to black-box testing techniques. Black-box approaches leverage the API specification (e.g., OAS) to automatically derive test cases from it. Essentially, these approaches exercise the API under test using (pseudo) random test data, including random and default values [12, 15], input data dictionaries [13], test data generators [18], data observed in previous responses from the API [26] and malformed inputs [15, 26].

While current approaches show promising results in the automated detection of bugs, their effectiveness is limited for real-world APIs which may require input data to be semantically complex or to satisfy certain input constraints [20]. For instance, the search operation of the YouTube API [10] imposes a total of 16 constraints involving 25 out of its 31 parameters. As another example, the operation to create a draft invoice in the PayPal API [5] requires as input a JSON object composed of more than 200 properties. Standard fuzzing techniques may not suffice to generate realistic test inputs that can actually exercise the inner functionality of such APIs. This is especially critical when resources are limited, and it is crucial to ensure that every request sent to the API is of some use.

In this paper, we present RESTest,[1] a framework for automated black-box testing of RESTful web APIs. RESTest receives as input the specification of the API under test in OAS format, and supports the generation, and optionally execution, of test cases using state-of-the-art techniques including fuzzing, adaptive random testing and constraint-based testing. For the generation of input data, RESTest relies on custom test data generators which automatically generate realistic data such as email addresses, language codes or strings matching a regular expression. The test cases can be instantiated into several frameworks and libraries such as REST Assured [9] and Postman [6]. Test case generation and execution can be performed

[1] https://github.com/isa-group/RESTest

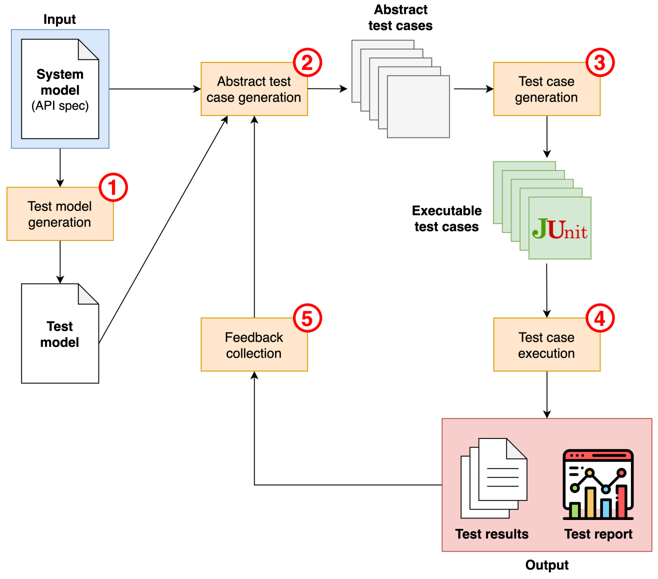

Figure 1: RESTest workflow.

in isolation (offline testing) or they can be interleaved (online testing). Graphical test reports are automatically generated using the Allure framework [1].

RESTest was firstly introduced as the only tool supporting constraint-based testing of RESTful APIs [22]. In this work, we present RESTest as a complete framework which integrates multiple testing techniques beyond constraint-based testing. RESTest is open source and can be easily extended with new test case generation strategies, test data generators and test writers.

2 RESTEST OVERVIEW

In what follows, we explain the basic workflow of RESTest, depicted in Figure 1.

(1) *Test model generation.* RESTest follows a model-based testing approach. Two models are used: the *system model* (i.e., the API specification), and the so-called *test model*, consisting of a configuration file in YAML notation. The test model contains all test-related configuration settings for the API under test, and it may be manually augmented to tailor the testing process, for example, to specify authentication details (e.g., API keys). The test model also specifies the test data to be used for each parameter, which may include data dictionaries or test data generators (e.g., airport or currency codes).

(2) *Abstract test case generation.* Test cases are derived from the system and test models using one or more testing techniques. These test cases are *abstract* or platform-independent, meaning that they can be later transformed into executable test cases for specific testing frameworks and programming languages.

(3) *Test case generation.* Abstract test cases are *instantiated* into executable test cases using specific testing frameworks and libraries such as REST Assured [9].

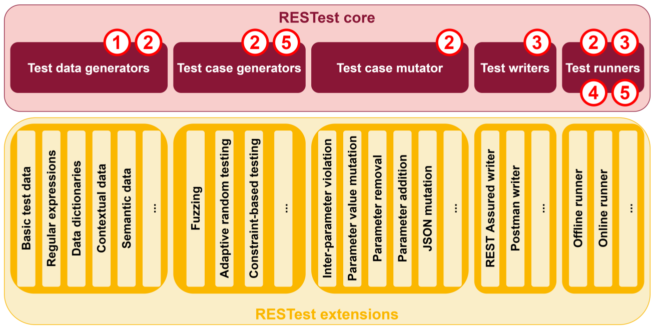

Figure 2: RESTest architecture.

(4) *Test case execution.* Test cases are optionally executed and the test results are exported to a machine-readable format and reported to the user, e.g., in a dashboard, using a test reporting framework like Allure [1].

(5) *Feedback collection.* Test generators can react to the test outputs to create more sophisticated test cases, for example, applying search-based techniques in order to maximize the API coverage [21] (e.g., status codes and response bodies).

3 RESTEST ARCHITECTURE

Figure 2 illustrates the architecture of RESTest. For each main component, Figure 2 shows the step of the testing process where it is involved, as explained in Section 2 and depicted in Figure 1. Next, we describe the main components of RESTest.

3.1 Test Data Generators

Test data generators in RESTest are automatically configured when generating the test model (step ①), and they generate test inputs (step ②). Testing a RESTful API operation such as `GET /books` involves generating values for the available operation parameters, e.g., `isbn` and `author`. Random values are unlikely to return any result in this case. RESTest automatically generates realistic values for these parameters following different strategies: (1) extracting values from knowledge bases like DBpedia [3] (*semantic data* generator); (2) reusing values observed in previous API responses (*contextual data* generator); or (3) leveraging manually-defined domain-specific generators (e.g., strings conforming to a regular expression) or data dictionaries.

3.2 Test Case Generators

Test case generators create test cases (step ②) according to different strategies, and they may leverage the feedback provided by previous executions (step ⑤). In RESTest, a test case represents a single call to an API operation and a set of assertions in the response. Stateful interactions (e.g., creating a resource with a `POST` request and then retrieving it with a `GET` request) can be achieved by testing multiple operations at the same time. RESTest currently supports the following test case generation strategies:

- *Fuzzing.* Test cases are built by assigning random values to each parameter of the operation under test. It is possible to create more sophisticated test cases by configuring specific test data generators for each parameter, instead of using purely random or malformed inputs.

- *Adaptive random testing*. Test cases are evenly distributed within the input space, with the hope of covering more API functionality and uncovering more failures [14].
- *Constraint-based testing*. This strategy is applicable to APIs containing inter-parameter dependencies, which, according to a recent study [20], account for 85% of industrial APIs. An inter-parameter dependency is a constraint between two or more input parameters of an API operation. For example, in the YouTube API, when searching for videos in high definition (`videoDefinition='high'`), the parameter `type` must be set to `'video'`, otherwise an error is returned. This testing approach leverages constraint programming solvers to automatically generate requests satisfying the inter-parameter dependencies present in the API operation [19]. Specifically, RESTest integrates IDLReasoner,[2] an analysis library developed by the authors.

3.3 Test Case Mutator

The test case mutator enables the creation of new test cases (step ②) by applying changes to existing ones (i.e., mutating them). This is typically done, for example, for transforming nominal test cases into faulty ones. Nominal test cases test the API under valid inputs (those conforming to the API specification). Faulty test cases check how the API handles invalid inputs, i.e., they expect a client error as a response.

3.4 Test Writers

Test writers transform abstract test cases into platform-specific ready-to-execute test cases (step ③). RESTest currently supports the generation of executable test cases for the frameworks REST Assured [9] and Postman [6].

3.5 Test Runners

Test runners allow to automate the whole testing process, i.e., the generation of test cases (steps ② and ③), their execution (step ④) and the collection and reporting of results (step ⑤). RESTest provides two working modes: offline and online testing. In *offline testing*, test case generation and execution are independent tasks. This has certain benefits. For example, test cases can be generated once, and then be executed many times as a part of regression testing. Also, test generation and test execution can be performed on different machines and at different times. In *online testing*, test case generation and execution are interleaved. This enables, for example, fully autonomous testing of RESTful web APIs, e.g., generating and executing test cases 24/7.

4 VALIDATION

In what follows, we show the potential of RESTest for offline and online testing of RESTful APIs.

[2]https://github.com/isa-group/IDLReasoner

4.1 Offline Testing

In this experiment, we automatically tested three RESTful services with inter-parameter dependencies [20]. Testing this kind of services with random approaches is generally inefficient or simply infeasible, since randomly generated requests are very unlikely to satisfy all the input constraints of the service [22]. For every service under test, we generated 2,000 test cases using the constraint-based (CBT) and random (RT) test case generators integrated into RESTest. Then, we counted the number of failures uncovered by each technique. Failures can occur due to several reasons such as server errors (5XX status codes) or unexpected client errors (4XX status codes) in response to valid inputs.

Table 1 provides a summary of the APIs under test and the results, including API name, operation tested, number of input parameters (P), number of dependencies (D), number (and percentage) of different parameters involved in at least one dependency (PD), and number of failures uncovered by each generator, where the CBT generator clearly outperformed the RT one. This highlights the fact that fuzzing may not suffice for testing complex APIs thoroughly. Among other failures, we found 500 status codes, disconformities with the OAS specification, and incorrect handling of valid and invalid inputs. For more detailed insights about these and more experiments, we refer the reader to our previous work [22].

Table 1: Characteristics and failures found in each API.

API	Operation	P	D	PD (%)	Failures	
					RT	CBT
Stripe	Create product	18	6	11 (61%)	0	535
Yelp	Search businesses	14	3	7 (50%)	67	161
YouTube	Search	31	16	25 (81%)	0	513

4.2 Online Testing

One of the key features of RESTest is that it can be set up to continuously test multiple APIs. The test results can be checked live in a multi-dashboard graphical user interface (GUI) based on the Allure test reporting framework [1].

For our second evaluation, we deployed 10 instances of RESTest in a server and left them continuously testing 15 RESTful services of 7 APIs: GitHub, Foursquare, Marvel, Stripe, Tumblr, Yelp and YouTube. We configured the test runners accordingly so that the quota limitations would never be exceeded (e.g., 1000 requests/hour in the Tumblr API). After 5 days, RESTest generated more than 90K test cases, 30% of which uncovered failures in all APIs under test. Specifically, the APIs of Foursquare, Marvel, Tumblr and Yelp exposed 5XX status codes (server errors), while the APIs of Foursquare, Stripe and YouTube returned 400 status codes (client errors) in response to valid inputs. We found this was because their inter-parameter dependencies were not correctly specified in the documentation (or correctly implemented in the API itself) [22]. In most APIs, mismatches between the API specification (i.e., the OAS) and the actual implementation were found too.

Figure 3 depicts the GUI[3] where the test results can be checked,

[3]This dashboard is available at http://betty.us.es/restest-showcase-demo.

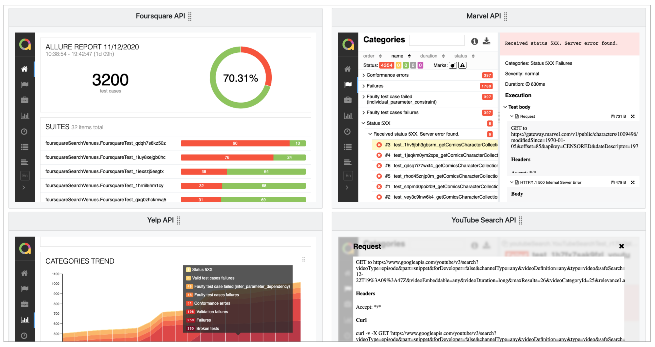

Figure 3: RESTest dashboard to monitor multiple APIs.

including the number of tests run, failures grouped by category and severity, test suites and timelines, among others.

5 RELATED WORK

Most black-box approaches for testing RESTful APIs generate test cases based on the API specification, e.g., in OAS format. For the generation of test data, they use random and default values [12, 15], user-defined data dictionaries [13], test data generators [18], data observed in previous responses from the API [26] and malformed inputs [15, 26]. RESTest integrates all these test data generation strategies, and provides an additional technique to extract realistic test data from knowledge bases like DBpedia [3]. As for the test oracles, most approaches rely on the absence of 5XX status codes and the conformance with the OAS specification. RESTest employs these and more complex oracles, such as checking the status code based on the input data used [22] (e.g., asserting that a valid request obtains a successful response).

Regarding white-box testing, Arcuri [11] proposed a search-based approach, where test cases are generated aiming to maximize code coverage and fault finding. White-box techniques may lead to better results than black-box [12], however, they require access to the source code of the API, which is not always available. Moreover, white-box techniques complicate online testing and monitoring of web APIs, which may hinder adoption by potential users.

Several commercial tools exist for the generation of test cases for RESTful APIs such as Postman [6], REST Assured [9], ReadyAPI [8] and API Fortress [2]. These tools, nevertheless, offer little automation and customization degree, as they are mostly intended for the automated execution of *manually written* test cases.

6 CONCLUSION AND FUTURE WORK

This paper presents RESTest, an open source framework for automated black-box testing of RESTful web APIs. RESTest implements several testing strategies and test data generation techniques, and can be integrated into continuous integration (CI) setups to continuously test and monitor multiple RESTful services. The framework can be easily extended with new test data generators, test case generation techniques and test writers, among others. RESTest has already proved useful in the automated detection of real-world bugs in commercial APIs used by millions of users worldwide [22].

In future work, we plan to extend RESTest with more test data generators and test case generation strategies. In particular, we are currently integrating search-based techniques for the generation of test cases optimized towards one or more objectives such as maximum API coverage [21], minimum test suite size or maximum inputs' diversity. We also aim to automate metamorphic testing of RESTful APIs [25] based on existing metamorphic relation patterns for query-based systems [24]. On the other hand, we intend to support non-functional testing of RESTful APIs by leveraging existing specifications such as SLA4OAI [17]. In doing so, we aim to make RESTest a full-fledged framework for online testing and monitoring of RESTful APIs.

ACKNOWLEDGMENTS

This work has been partially supported by the European Commission (FEDER) and Junta de Andalucia under projects APOLO (US-1264651) and EKIPMENT-PLUS (P18-FR-2895), by the Spanish Government under project HORATIO (RTI2018-101204-B-C21), and by the FPU scholarship program, granted by the Spanish Ministry of Education and Vocational Training (FPU17/04077).

REFERENCES

[1] [n.d.]. Allure - Test report framework. http://allure.qatools.ru/ accessed April 2021.
[2] [n.d.]. API Fortress. https://apifortress.com/ accessed April 2021.
[3] [n.d.]. DBpedia. https://www.dbpedia.org/ accessed April 2021.
[4] [n.d.]. OpenAPI Specification. https://www.openapis.org accessed April 2021.
[5] [n.d.]. PayPal API. https://developer.paypal.com/docs/api/ accessed April 2021.
[6] [n.d.]. Postman. https://www.getpostman.com accessed April 2021.
[7] [n.d.]. ProgrammableWeb API Directory. http://www.programmableweb.com/ accessed April 2021.
[8] [n.d.]. ReadyAPI. https://smartbear.com/product/ready-api/overview/ accessed April 2021.
[9] [n.d.]. REST Assured. http://rest-assured.io accessed April 2021.
[10] [n.d.]. YouTube Data API. https://developers.google.com/youtube/v3/ accessed April 2021.
[11] Andrea Arcuri. 2019. RESTful API Automated Test Case Generation with EvoMaster. *ACM TOSEM* 28, 1 (2019), 1–37.
[12] Andrea Arcuri. 2021. Automated Blackbox and Whitebox Testing of RESTful APIs With EvoMaster. *IEEE Software* (2021).
[13] Vaggelis Atlidakis, Patrice Godefroid, and Marina Polishchuk. 2019. RESTler: Stateful REST API Fuzzing. In *ICSE*. 748–758.
[14] T. Y. Chen, H. Leung, and I. K. Mak. 2005. Adaptive Random Testing. In *ASIAN*. 320–329.
[15] Hamza Ed-douibi, Javier Luis Cánovas Izquierdo, and Jordi Cabot. 2018. Automatic Generation of Test Cases for REST APIs: A Specification-Based Approach. In *EDOC*. 181–190.
[16] Roy Thomas Fielding. 2000. *Architectural Styles and the Design of Network-based Software Architectures*. Ph.D. Dissertation.
[17] Antonio Gamez-Diaz, Pablo Fernandez, and Antonio Ruiz-Cortes. 2019. Automating SLA-Driven API Development with SLA4OAI. In *ICSOC*. 20–35.
[18] Stefan Karlsson, Adnan Causevic, and Daniel Sundmark. 2020. QuickREST: Property-based Test Generation of OpenAPI Described RESTful APIs. In *ICST*.
[19] Alberto Martin-Lopez, Sergio Segura, Carlos Müller, and Antonio Ruiz-Cortés. 2020. Specification and Automated Analysis of Inter-Parameter Dependencies in Web APIs. *IEEE TSC* (2020).
[20] Alberto Martin-Lopez, Sergio Segura, and Antonio Ruiz-Cortés. 2019. A Catalogue of Inter-Parameter Dependencies in RESTful Web APIs. In *ICSOC*. 399–414.
[21] Alberto Martin-Lopez, Sergio Segura, and Antonio Ruiz-Cortés. 2019. Test Coverage Criteria for RESTful Web APIs. In *A-TEST*. 15–21.
[22] Alberto Martin-Lopez, Sergio Segura, and Antonio Ruiz-Cortés. 2020. RESTest: Black-Box Constraint-Based Testing of RESTful Web APIs. In *ICSOC*.
[23] Leonard Richardson, Mike Amundsen, and Sam Ruby. 2013. *RESTful Web APIs*. O'Reilly Media, Inc.
[24] Sergio Segura, Amador Durán, Javier Troya, and Antonio Ruiz-Cortés. 2019. Metamorphic Relation Patterns for Query-Based Systems. In *MET*. 24–31.
[25] Sergio Segura, José A Parejo, Javier Troya, and Antonio Ruiz-Cortés. 2018. Metamorphic Testing of RESTful Web APIs. *IEEE TSE* 44, 11 (2018), 1083–1099.
[26] Emanuele Viglianisi, Michael Dallago, and Mariano Ceccato. 2020. RestTestGen: Automated Black-Box Testing of RESTful APIs. In *ICST*.