Automated Generation of Test Oracles for RESTful APIs

Juan C. Alonso
SCORE Lab, Universidad de Sevilla, Seville, Spain

ABSTRACT

Test case generation tools for RESTful APIs have proliferated in recent years. However, despite their promising results, they all share the same limitation: they can only detect crashes (i.e., server errors) and disconformities with the API specification. In this paper, we present a technique for the automated generation of test oracles for RESTful APIs through the detection of invariants. In practice, our approach aims to learn the expected properties of the output by analysing previous API requests and their corresponding responses. For this, we extended the popular tool Daikon for dynamic detection of likely invariants. A preliminary evaluation conducted on a set of 8 operations from 6 industrial APIs reveals a total precision of 66.5% (reaching 100% in 2 operations). Moreover, our approach revealed 6 reproducible bugs in APIs with millions of users: Amadeus, GitHub and OMDb.

CCS CONCEPTS

• Software and its engineering → Software testing and debugging; **• Information systems → RESTful web services**.

KEYWORDS

RESTful APIs, Oracle problem, Invariant detection

ACM Reference Format:
Juan C. Alonso. 2022. Automated Generation of Test Oracles for RESTful APIs. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '22), November 14–18, 2022, Singapore, Singapore.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3540250.3559080

1 RESEARCH PROBLEM AND MOTIVATION

Web *Application Programming Interfaces (APIs)* allow heterogeneous software systems to interact over the network [10, 16]. Companies such as Google, Microsoft and Apple provide web APIs to enable their integration into third-party systems. Most web APIs adhere to the REpresentational State Transfer (REST) architectural style, being known as RESTful APIs [8]. RESTful APIs are commonly specified using languages such as the OpenAPI Specification (OAS) [2], which defines a standard notation for describing *RESTful APIs* in terms of operations, input parameters, and output formats, among others.

Automated test case generation for RESTful APIs is a thriving research topic due to the key role they play in software integration [5, 12, 19]. Despite the promising results of existing proposals, they are all limited by the types of errors they can detect: uncontrolled server failures (labelled with a 500 status code) and disconformities with the API specification (e.g., the API response contains a field that is not present in the specification). Generally, there are several (in the order of dozens or hundreds) possible test oracles for every API operation. For example, if we perform a search for songs in the Spotify API by setting a maximum number of results (`limit` parameter), the size of the response field containing an array of songs should be equal to or smaller than the value of `limit`. Currently, creating these kinds of test oracle is a costly and manual endeavour that requires domain knowledge.

In this paper, we present an approach for the automated generation of test oracles for RESTful APIs. Our approach is based on the detection of *invariants*, that is, properties of the output that should always hold, e.g., a JSON property from the response should always have the same value as one of the input parameters. For this, we developed an extension (so-called instrumenter) of Daikon [7], a popular tool for dynamic detection of likely invariants. Generated (likely) invariants can be used as potential test oracles, and they can also help to identify unexpected behaviour (bugs) in the software under test.

Preliminary results on a dataset of 8 operations from 6 commercial APIs show the capability of the proposal to generate hundreds of valid test oracles, obtaining a total precision of 66.5% (up to 100% in 2 of the operations), More importantly, the generated invariants uncovered 6 reproducible bugs in the APIs of Amadeus, GitHub, and OMDb.

2 BACKGROUND AND RELATED WORK

Most approaches for automated testing of RESTful APIs generate test cases from an OAS specification by using model based testing [12, 18] or property based testing [9]. Other proposals aim to infer sequences of operations from the specification [5, 19]. According to a recently published comparative study [15], current tools for automated testing of RESTful APIs are limited by the types of oracles that they support (server errors and response format analysis). This reveals that, despite the significant advances in test case generation, current techniques are still limited by their fault detection capabilities. Other authors have proposed using metamorphic testing [17], but this technique is only applicable in specific operations and requires a manually generated specification.

An invariant is a property that is always satisfied at one or more points in the execution of a program, such as its inputs and outputs in the context of black-box testing, whereas a likely invariant is a property that is satisfied by a set of program executions but that could not be satisfied by a different execution with, for example, different input values. Automated detection of likely invariants has shown promising results in different contexts such as learning postconditions in Java programs [14], relational databases [6], web applications [13], automated program repair [21], cyber-physical systems [3, 20] or robotics [11], among others. Nonetheless, this technique has not been applied yet in the context of RESTful APIs, despite its potential.

There are several systems for the detection of likely invariants in program executions, with Daikon [7] being the most widely used. Daikon detects likely invariants by analyzing an instrumentation of the execution of a program. This instrumentation process is performed by an *instrumenter*, a software that transforms the execution of a program into a format that can be processed by Daikon. There are several Daikon instrumenters available for different programming languages, such as Java, Perl or C++, and file formats, such as csv files [1]. Daikon observes the properties of a program through different executions, returning those properties (i.e., likely invariants) that are always satisfied.

3 APPROACH

Our approach is divided into two steps:

Instrumentation. We implemented a Daikon instrumenter, which receives three inputs: 1) the OAS specification of the API under test, 2) a set of automatically generated test cases (API requests), and 3) the corresponding test case observed outputs (API responses). This instrumentation consists of a `decls` file (describing the format of the API operations inputs and outputs) and a `dtrace` file (values assigned to each field of the `decls` file in each test case).

Invariant detection. The test suite instrumentation is processed by a modified version of Daikon, resulting in a set of likely invariants that can be used as test oracles. Specifically, we modified Daikon by suppressing 25 invariants unlikely to reveal relevant information in our context (e.g., comparison of the length of string properties) and adding 22 domain specific invariants based on an analysis of a dataset of 48 APIs systematically collected for a previous paper by the author [4]. This version of Daikon supports a total of 140 different types of invariants that fall into different categories such as arithmetic relations (`return.track_number>=1`), array properties (`return.hotelId in input.hotelIds[]`), specific values (`return.visibility one of {"private", "public"}`) or formats (`return.href is Url`), among many others. New types of invariants may be targeted in the future. The application of this approach allows the detection of domain-specific test oracles and bugs that are not contemplated by existing techniques.

4 RESULTS AND CONTRIBUTIONS

For the evaluation, we selected a set of 8 operations from 6 industrial APIs. For every operation, we automatically generated 1000 successful test cases (i.e., responses returning a 2XX status code) using the RESTest framework [12]. This set of test cases, along with their observed output and the corresponding OAS specification, were fed to the instrumenter, resulting in a set of likely invariants that we manually classified as true positives, false positives, or bugs.

To check the impact of the size of the test suite in the precision, each test suite was divided in subsets of 1000, 500, 100 and 50 test cases, computing the precision achieved for each subset. These experiments allowed to answer the following Research Questions:

RQ1: Effectiveness of the proposal to generate test oracles. Table 1 shows the number of invariants (column 3), the precision (column 4) and the number of bug revealing invariants detected (column 5) when using the set of 1000 test cases. Our approach achieved a precision of up to 100% in two operations (and a total precision of 66.5%), detecting tens of domain-specific oracles for every operation. Most of the false positives (57.7% of them) are arithmetic comparisons between numeric fields with different scales, such as the duration in miliseconds of a Spotify song and its number of artists (`return.duration_ms>size(return.artists[])`).

Table 1: Experimental results. I=Invariants, P=Precision

API	Operation	I	P (%)	Bugs
Amadeus Hotel	Find hotel offers	99	59.6	1
GitHub	List organization repositories	94	74.5	9
OMDb	By ID or Title	17	82.4	1
OMDb	By Search	6	100	1
Spotify	Create Playlist	28	100	0
Spotify	Get Album tracks	51	80.4	0
Yelp	Search businesses	19	42.1	0
YouTube	List videos	187	57.2	0
Total		**501**	**66.5**	**12**

RQ2: Impact of the size of the test suite on the precision. The total precision achieved by the approach ranged between 65.3% on the set of 50 test cases and 66.5% on the set of 1000 test cases. Thus, our preliminary results suggest that an increment in the size of the test suite does not have a significant impact on the precision achieved.

RQ3: Error detection capabilities. Although the test oracles can be used to generate assertions, the faulty behavior of an API could be reflected on an invariant. This allowed our approach to detect 6 domain specific and reproducible bugs in 4 operations from 3 APIs with millions of users worldwide, namely Amadeus, GitHub and OMDb.

In the Amadeus Hotel search API, one of the detected invariants (`return.room.typeEstimated.beds >= 0`) allowed to identify 55 hotel offers in which the room had zero beds, even though the API documentation states that every room must have at least one. This bug has been confirmed and fixed by the API providers.

According to the OAS specification of the "List organization repositories" operation of the GitHub API, one of the response fields specifies the template repository used for creating each repository, if any. Our approach detected that this property was never present (`return.template_repository==null`). The API providers confirmed that the presence of this field is an inconsistency in the documentation and created an internal issue to fix it.

The parameter `type` of the OMDb API operations is used to filter the obtained results to one media type ("movie", "episode" or "series", according to the documentation). However, one of the invariants (`return.Type one of {"game", "movie", "series" }`) revealed a new value for this parameter that was not specified in the documentation ("game") and that the "By Search" operation does not return elements of type "episode".

The dataset of our preliminary evaluation is available at https://doi.org/10.5281/zenodo.6874668

REFERENCES

[1] 2022. DAIKON instrumenters. https://plse.cs.washington.edu/daikon/download/doc/daikon.html#Front-ends-_0028instrumentation_0029. Accessed July 2022.
[2] 2022. OpenAPI Specification. https://www.openapis.org accessed July 2022.

[3] Afsoon Afzal, Claire Le Goues, and Christopher Steven Timperley. 2021. Mithra: Anomaly Detection as an Oracle for Cyberphysical Systems. *IEEE Transactions on Software Engineering* (2021), 1–1. https://doi.org/10.1109/TSE.2021.3120680

[4] Juan C. Alonso, Alberto Martin-Lopez, Sergio Segura, Jose Maria Garcia, and Antonio Ruiz-Cortes. 2022. ARTE: Automated Generation of Realistic Test Inputs for Web APIs. *IEEE Transactions on Software Engineering* (2022). https://doi.org/10.1109/TSE.2022.3150618

[5] V. Atlidakis, P. Godefroid, and M. Polishchuk. 2019. RESTler: Stateful REST API Fuzzing. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. 748–758. https://doi.org/10.1109/ICSE.2019.00083

[6] Jake Cobb, James A. Jones, Gregory M. Kapfhammer, and Mary Jean Harrold. 2011. Dynamic Invariant Detection for Relational Databases. In *Proceedings of the Ninth International Workshop on Dynamic Analysis* (Toronto, Ontario, Canada) *(WODA '11)*. Association for Computing Machinery, New York, NY, USA, 12–17. https://doi.org/10.1145/2002951.2002955

[7] Michael D. Ernst, Jeff H. Perkins, Philip J. Guo, Stephen McCamant, Carlos Pacheco, Matthew S. Tschantz, and Chen Xiao. 2007. The Daikon system for dynamic detection of likely invariants. *Science of Computer Programming* 69, 1 (2007), 35–45. https://doi.org/10.1016/j.scico.2007.01.015 Special issue on Experimental Software and Toolkits.

[8] Roy Thomas Fielding. 2000. *Architectural Styles and the Design of Network-based Software Architectures*. Ph. D. Dissertation.

[9] Zac Hatfield-Dodds and Dmitry Dygalo. 2021. Deriving Semantics-Aware Fuzzers from Web API Schemas. *arXiv preprint arXiv:2112.10328* (2021).

[10] Daniel Jacobson, Greg Brail, and Dan Woods. 2011. *APIs: A Strategy Guide*. O'Reilly Media, Inc.

[11] Deborah S Katz, Christopher S Timperley, and Claire Le Goues. 2022. Using Dynamic Binary Instrumentation to Detect Failures in Robotics Software. *arXiv preprint arXiv:2201.12464* (2022).

[12] Alberto Martin-Lopez, Sergio Segura, and Antonio Ruiz-Cortés. 2020. RESTest: Black-Box Constraint-Based Testing of RESTful Web APIs. In *International Conference on Service-Oriented Computing*. 459–475.

[13] Ali Mesbah, Arie van Deursen, and Danny Roest. 2012. Invariant-Based Automatic Testing of Modern Web Applications. *IEEE Transactions on Software Engineering* 38, 1 (2012), 35–53. https://doi.org/10.1109/TSE.2011.28

[14] Facundo Molina, Pablo Ponzio, Nazareno Aguirre, and Marcelo Frias. 2021. EvoSpex: An Evolutionary Algorithm for Learning Postconditions. In *2021 IEEE/ACM 43st International Conference on Software Engineering (ICSE)*. 1223–1235. https://doi.org/10.1109/ICSE43902.2021.00112

[15] Saurabh Sinha Myeongsoo Kim, Qi Xin and Alessandro Orso. 2022. Automated Test Generation for REST APIs: No Time to Rest Yet. In *Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis*.

[16] Leonard Richardson, Mike Amundsen, and Sam Ruby. 2013. *RESTful Web APIs*. O'Reilly Media, Inc.

[17] Sergio Segura, José A. Parejo, Javier Troya, and Antonio Ruiz-Cortés. 2018. Metamorphic Testing of RESTful Web APIs. *IEEE Transactions on Software Engineering* 44, 11 (2018), 1083–1099. https://doi.org/10.1109/TSE.2017.2764464

[18] Dimitri Stallenberg, Mitchell Olsthoorn, and Annibale Panichella. 2021. Improving Test Case Generation for REST APIs Through Hierarchical Clustering. In *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 117–128. https://doi.org/10.1109/ASE51524.2021.9678586

[19] E. Viglianisi, M. Dallago, and M. Ceccato. 2020. RESTTESTGEN: Automated Black-Box Testing of RESTful APIs. In *2020 IEEE 13th International Conference on Software Testing, Validation and Verification (ICST)*. 142–152. https://doi.org/10.1109/ICST46399.2020.00024

[20] Xiaodong Yang, Omar Ali Beg, Matthew Kenigsberg, and Taylor T. Johnson. 2022. A Framework for Identification and Validation of Affine Hybrid Automata from Input-Output Traces. *ACM Trans. Cyber-Phys. Syst.* 6, 2, Article 13 (apr 2022), 24 pages. https://doi.org/10.1145/3470455

[21] Yuntong Zhang, Xiang Gao, Gregory J Duck, and Abhik Roychoudhury. 2022. Program Vulnerability Repair via Inductive Inference. (2022).