# RESTest: automated black-box testing of RESTful web APIs

<https://doi.org/10.1145/3460319.3469082>

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>.

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 lan-  
guages such as the OpenAPI Specification (OAS) [ 4 ]. OAS provides  
a structured description of a RESTful web API that allows both hu-  
mans 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 seam-  
less 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 tech-  
niques. 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 dic-  
tionaries [ 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 auto-  
mated 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 op-  
eration to create a draft invoice in the PayPal API [ 5 ] requires as  
input a JSON object composed of more than 200 properties. Stan-  
dard 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

n isolation (offline testing) or they can be interleaved (online test-  
ing). 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 multi-  
ple testing techniques beyond constraint-based testing. RESTest is  
open source and can be easily extended with new test case genera-  
tion 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 de-  
tails (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, mean-  
ing that they can be later transformed into executable test  
cases for specific testing frameworks and programming lan-  
guages.  
(3) Test case generation. Abstract test cases are instantiated into  
executable test cases using specific testing frameworks and  
libraries such as REST Assured [9]

(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 out-  
puts 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 com-  
ponent, 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 re-  
sult 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 dif-  
ferent 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 re-  
sponse. 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 defini-  
tion (videoDefinition=‘high’), the parameter type must  
be set to ‘video’, otherwise an error is returned. This testing  
approach leverages constraint programming solvers to auto-  
matically generate requests satisfying the inter-parameter  
dependencies present in the API operation [19 ]. Specifically,  
RESTest integrates IDLReasoner,2 an analysis library devel-  
oped 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

4.1 Offline Testing  
In this experiment, we automatically tested three RESTful services  
with inter-parameter dependencies [20]. Testing this kind of ser-  
vices 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 tech-  
nique. 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 dif-  
ferent 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]

4.2 Online Testing  
One of the key features of RESTest is that it can be set up to contin-  
uously 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 un-  
der 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 GUI3 where the test results can be checked

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

6 CONCLUSION AND FUTURE WORK  
This paper presents RESTest, an open source framework for auto-  
mated 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 contin-  
uously 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 pat-  
terns 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.