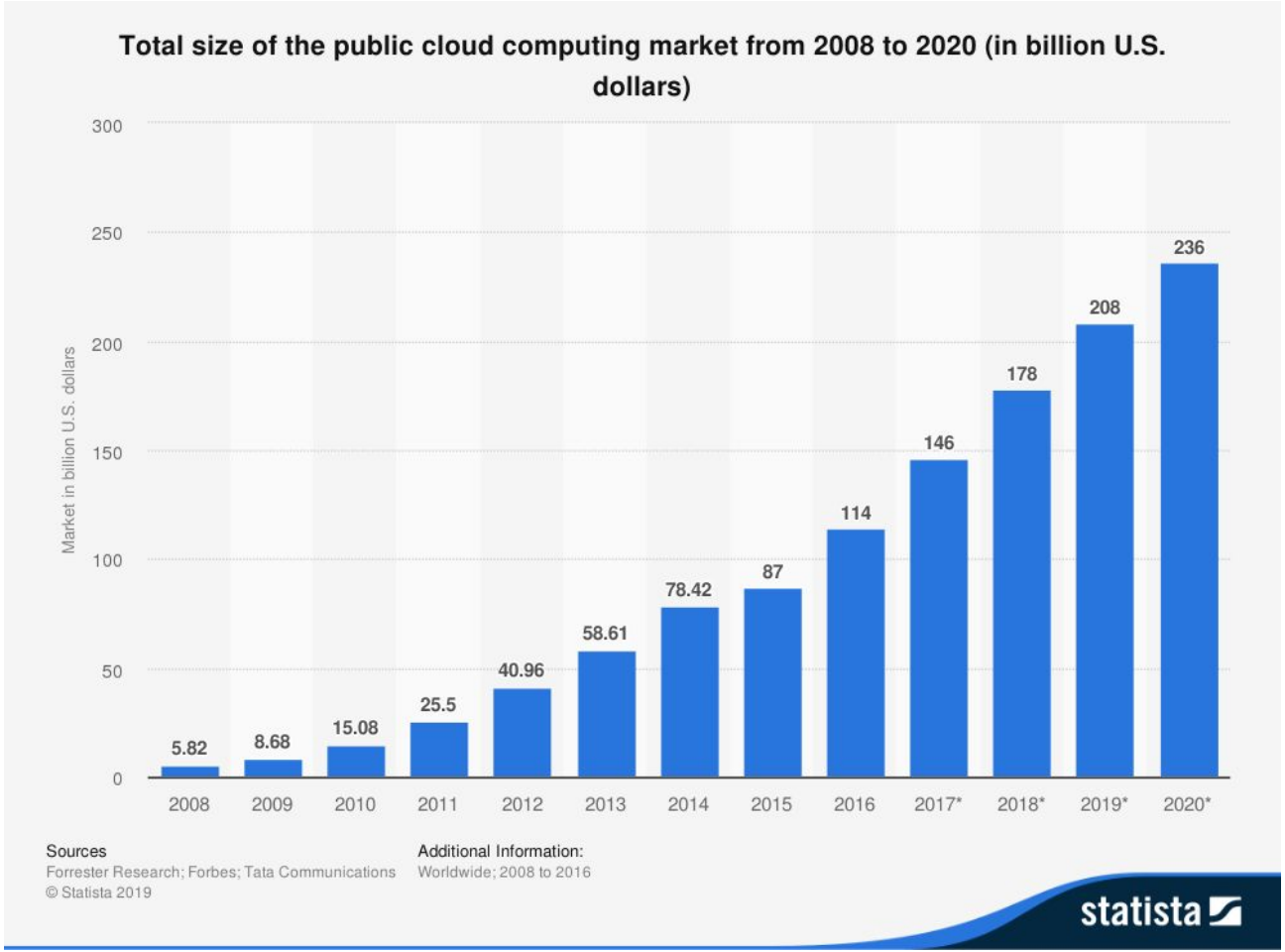
Structure and Feedback in Cloud Service API Fuzzing

Doctoral Thesis Defense

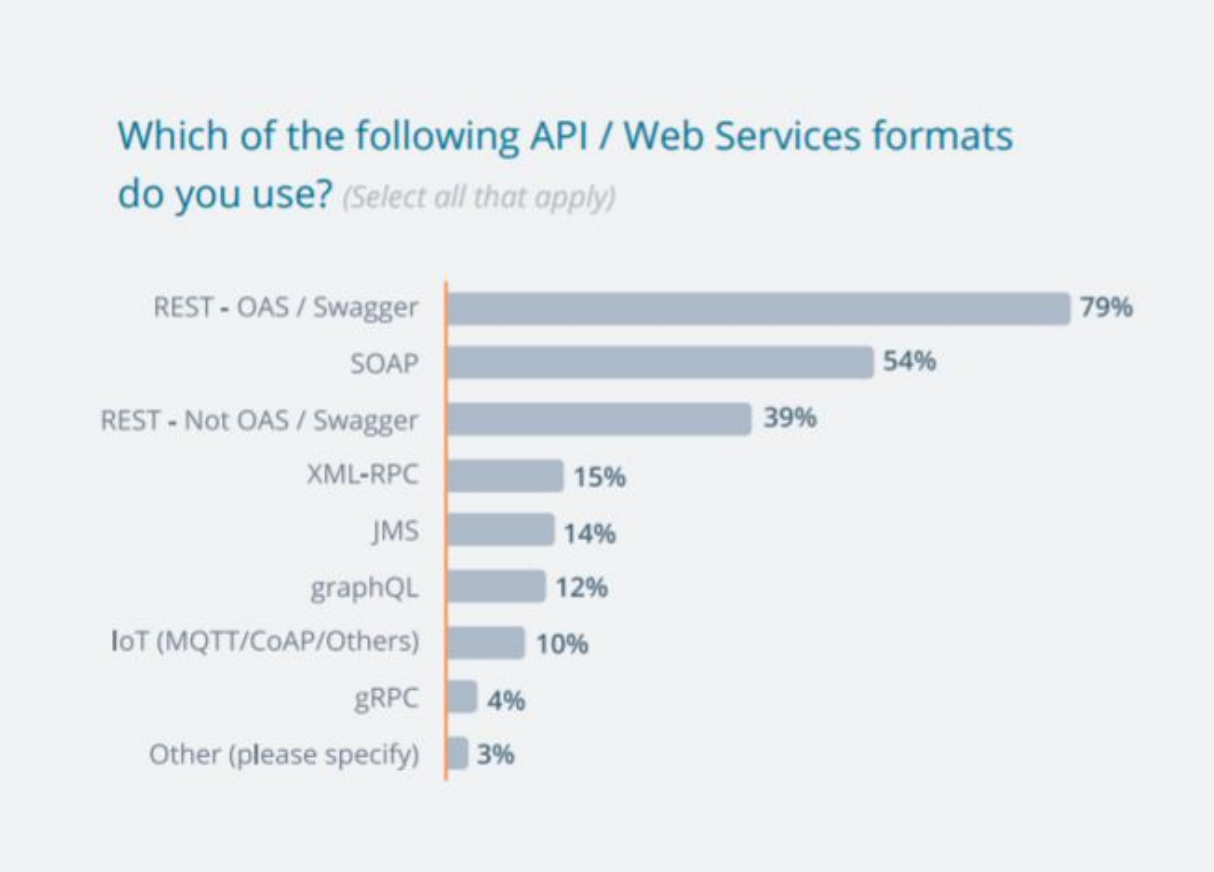
Vaggelis Atlidakis

Disclaimer: Part of the work discussed today was done while working at Microsoft Research in collaboration with Patrice Godefroid and Marina Polishchuk

Structure and Feedback in Cloud Service API Fuzzing



Structure and Feedback in Cloud Service API Fuzzing



Preface:

This survey was designed to establish benchmarks for the API industry regarding the methodologies, practices, and tools used by software teams to plan, design, develop, test, document, and monitor APIs

Methodology:

At SmartBear, we conducted a global online 52-question survey over the course of two months from November to December 2018 and collected a total of 3,372 responses. The primary audience for the survey were users of the open source, free, and commercial versions of the Swagger, SoapUI, and ReadyAPI

Testing REST APIs

- Grammar-based fuzzing
  - Producing grammar requires manual effort
  - No coverage feedback (How much fuzzing is enough?)
- HTTP fuzzers
  - Requires live traffic
  - Not Stateful (cannot reproduce sequences of events)
- Custom tools for specific APIs
  - Labour intensive
  - High maintenance cost (APIs evolve over time)

Thesis

We can leverage the structured usage of cloud services through REST APIs and feedback obtained during interaction in order to test cloud services in an automatic, efficient, and learning-based way

Contributions

- RESTler: Stateful REST API Fuzzing (ICSE'19)
- Checking Security Properties of Cloud Service REST APIs (ICST'20)
- Pythia: Grammar-Based Fuzzing of REST APIs with Coverage-guided Feedback and Learning-based Mutations (target submission ICSE'21)

Contributions

- RESTler: Stateful REST API Fuzzing (ICSE'19)
- Checking Security Properties of Cloud Service REST APIs (ICST'20)
- Pythia: Grammar-Based Fuzzing of REST APIs with Coverage-guided Feedback and Learning-based Mutations (target submission ICSE'21)

Challenge

- Testing individual API endpoints will miss errors that require sequences of events in order to be exposed

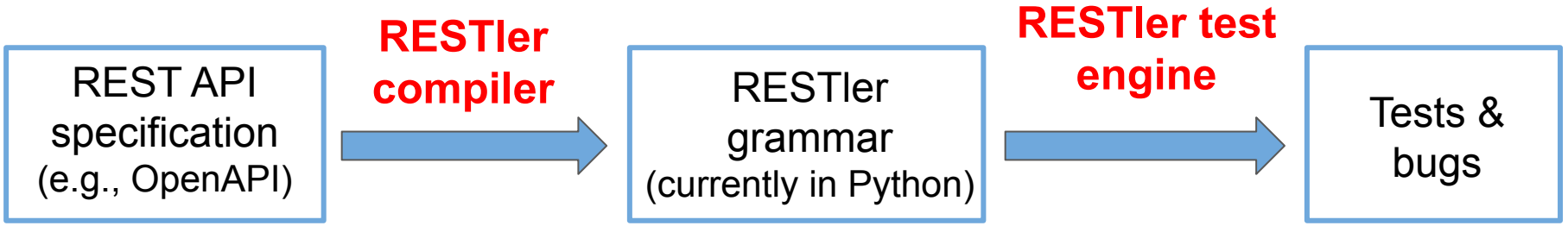
RESTler: Stateful REST API Fuzzing

- Dependency analysis between API requests
- Dynamic feedback loop that learns from past tests

Kinds of errors RESTler can find

- “500 Internal Server Error” (unhandled exceptions)

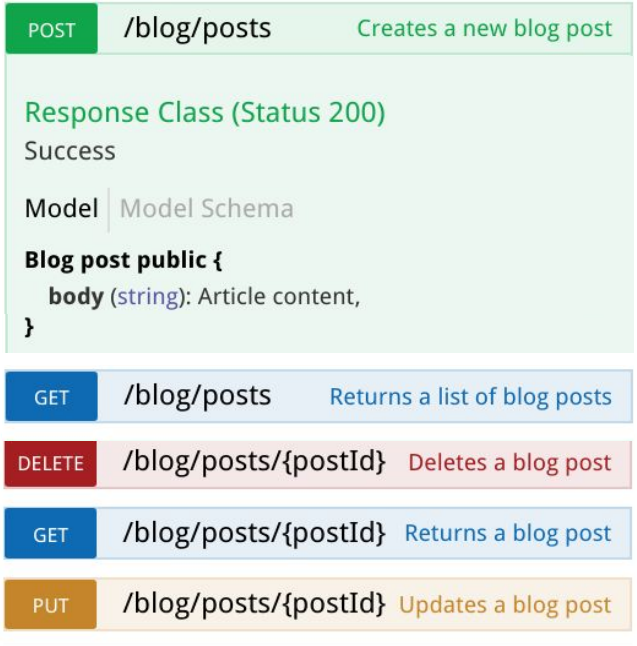
System overview



- Describe how to fuzz each request type
- Identify producer/consumer dependencies
- Generate code to parse responses

- Generate and execute tests: sequences of requests
- Systematic state-space exploration (breadth first search and others)
- Analyze test results: Dynamic feedback loop learns from service responses of past tests`

Example



Sample Swagger specification

```
fuzzable["string"] = {"SampleString", "␣"}

def parse_posts(data):
  post_id = data["id"]
  dependencies.set_var(post_id)

request = requests.Request(
  restler_static("POST"),
  restler_static("/api/blog/posts/"),
  restler_static("HTTP/1.1"),
  restler_static("{"),
  restler_static("body:"),
  restler_fuzzable("string"),
  restler_static("}"),
  'post_send': {
      'parser': parse_posts,
      'dependencies': [
          post_id.writer(),
      ]
  }
```

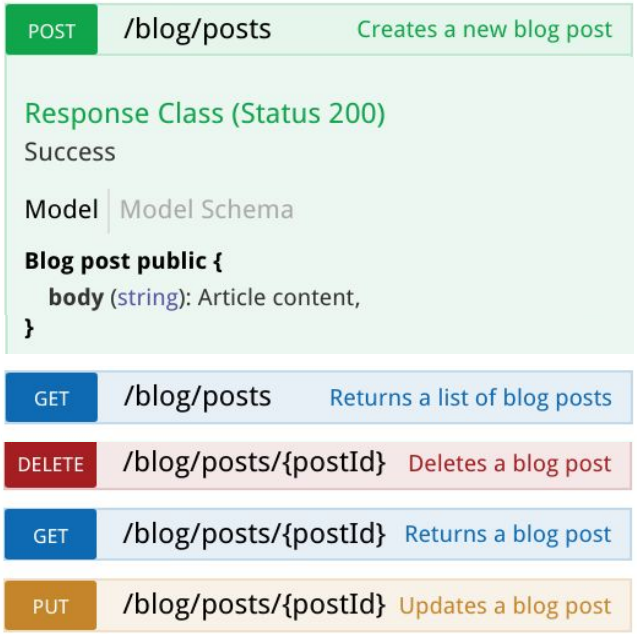
RESTler grammar fragment

```
Send: POST /api/blog/posts/ HTTP/1.1
Accept: application/json
Content−Type: application/json
{"body":"sampleString"}

Recv: HTTP/1.1 201 CREATED
Content−Type: application/json
Server: Werkzeug/0.14.1 Python/2.7.12
{"body": "sampleString", "id": 5889}
```

```
Send: POST /api/blog/posts/ HTTP/1.1
Accept: application/json
Content−Type: application/json
{"body":" "}

Recv: HTTP/1.1 400 Bad Request
Content−Type: application/json
Server: Werkzeug/0.14.1 Python/2.7.12
{"error": "body is empty"}
```

Sample tests

Test generation



Sample Swagger specification

API Requests = $\{ A_2^{\text{prod: post-id}}, B_1^{\text{none}}, C_1^{\text{cons: post-id}} \}$

Tests (Gen-0) = $\{ \varnothing \}$

Tests (Gen-1) = $\{ A_{1/2}^{\text{prod: post-id}}, A_{2/2}^{\text{prod: post-id}}, B_{1/1}^{\text{none}} \}$

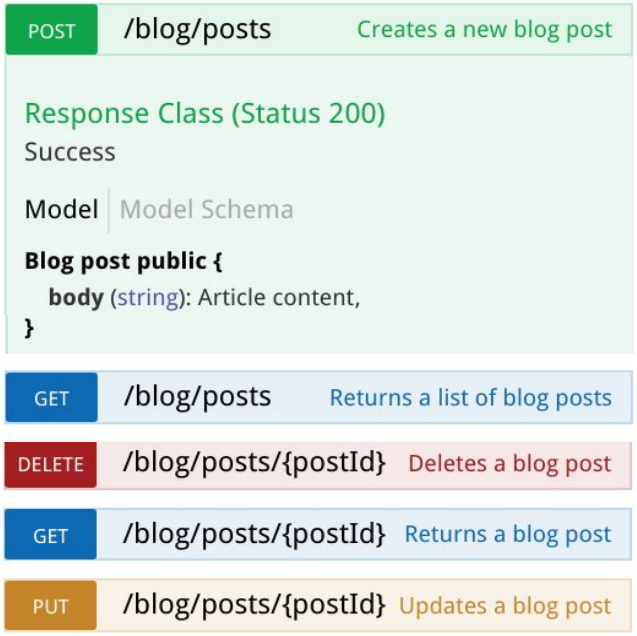
Tests (Gen-2) = $\{ A_{1/2}^{\text{prod: post-id}} A_{1/2}^{\text{prod: post-id}},$

$A_{1/2}^{\text{prod: post-id}} A_{2/2}^{\text{prod: post-id}}, A_{1/2}^{\text{prod: post-id}} B_{1/1}^{\text{none}},$

$A_{1/2}^{\text{prod: post-id}} C_{1/1}^{\text{cons: post-id}}, \ldots, B_{1/1}^{\text{none}} B_{1/1}^{\text{none}} \}$

➢ BFS total tests: **13**

Test generation



Sample Swagger specification

API Requests = $\{ A_2^{\text{prod: post-id}}, B_1^{\text{none}}, C_1^{\text{cons: post-id}} \}$

Tests (Gen-0) = $\{ \varnothing \}$

Tests (Gen-1) = $\{ A_{1/2}^{\text{prod: post-id}}, A_{2/2}^{\text{prod: post-id}}, B_{1/1}^{\text{none}} \}$

Tests (Gen-2) = $\{ A_{1/2}^{\text{prod: post-id}} A_{1/2}^{\text{prod: post-id}},$

$A_{1/2}^{\text{prod: post-id}} A_{2/2}^{\text{prod: post-id}}, A_{1/2}^{\text{prod: post-id}} B_{1/1}^{\text{none}},$

$A_{1/2}^{\text{prod: post-id}} C_{1/1}^{\text{cons: post-id}} \}$

➢ BFS-Fast total tests: **7**

Selected evaluation results

- Q1: Are tests generated by RESTler exercising deeper service-side code over time?
- Q2: Can RESTler find bugs in production-scale cloud services?

Study subjects: Gitlab, Spree, and Mastodon

- Open-source production-scale services
- Self-hosted git, e-commerce, and social networking
- Complex REST APIs & milions of users

Deeper service exploration (Q1)

API Family	Total API Requests	Seq. Len.	Code Coverage Increase (lines of code)	Tests
Gitlab Commits	15 (*11)	1	598	1
		2	1108	7
		3	1196	250
		4	1760	2220
		5	1760	3667
Spree Cart	8 (*11)	1	10	1
		2	208	2
		3	1473	47
		4	1943	6380
Mastodon Statuses	18 (*19)	1	4	60
		2	333	5908
		3	631	28926

- Longer sequences increase service-side code coverage

Testing APIs with RESTler (BFS 5h per API family)

Deeper service exploration (Q1)

API Family	Total API Requests	Seq. Len.	Code Coverage Increase (lines of code)	Tests
Gitlab Commits	**15 (*11)**	1	598	1
		2	1108	7
		3	1196	250
		4	1760	2220
		5	1760	3667
Spree Cart	8 (*11)	1	10	1
		2	208	2
		3	1473	47
		4	1943	6380
Mastodon Statuses	18 (*19)	1	4	60
		2	333	5908
		3	631	28926

Testing APIs with RESTler (BFS 5h per API family)

- Longer sequences increase service-side code coverage
- Sequences of at least three requests
- Progress in large search space

Deeper service exploration (Q1)

API Family	Total API Requests	Seq. Len.	Code Coverage Increase (lines of code)	Tests
Gitlab Commits	**15 (*11)**	**1**	**598**	**1**
		2	**1108**	**7**
		3	**1196**	**250**
		4	1760	2220
		5	1760	3667
Spree Cart	8 (*11)	1	10	1
		2	208	2
		3	1473	47
		4	1943	6380
Mastodon Statuses	18 (*19)	1	4	60
		2	333	5908
		3	631	28926

Testing APIs with RESTler (BFS 5h per API family)

- Longer sequences increase service-side code coverage
- Sequences of at least three requests
- Progress in large search space

Example: Testing Commits API (5 hours)

- **Brute-force**: 15 requests / 11 different payloads; that is: (15*11)^3 = 4.4 million sequences of length three
- **RESTler:** Sequence length three: 250 tests generated (request dependencies + service feedback)

New bugs found with RESTler (Q2)

API	BFS	BFS-Fast	RandomWalk	∩	∪
Commits	5	1	5	1	5
Cart	1	1	1	1	1
Statuses	1	1	0	0	1
...	...	...	...	...	...
Total	**18**	**15**	**22**	**10**	**24**

Bug buckets per search strategy

- BFS-Fast finds least bugs
- RandomWalk finds most bugs
- All 24 bugs are easily reproducible, confirmed, and fixed

Example Bug [#50268]

1. Create a gitlab project
2. Create a file with a proper commit message
3. Delete the file with an empty commit message

➢ "500 Internal Server Error"

Main take-aways (so far)

- ✔ Introduced stateful REST API fuzzing
  - ➢ 500 “Internal Server Error” (Unhandled exceptions)

Contributions

- RESTler: Stateful REST API Fuzzing (ICSE'19)
- Checking Security Properties of Cloud Service REST APIs (ICST'20)
- Pythia: Grammar-Based Fuzzing of REST APIs with Coverage-guided Feedback and Learning-based Mutations (target submission ICSE'21)

Structure and Feedback in Cloud Service API Fuzzing

Challenge

- How can we uncover errors that do not cause visible 500s?

Checking security properties

- Introduce rules that capture desirable security properties of cloud service REST APIs
- Augment stateful REST API fuzzing with checkers that test violation of these rules

Kinds of error

- Violations of security property rules

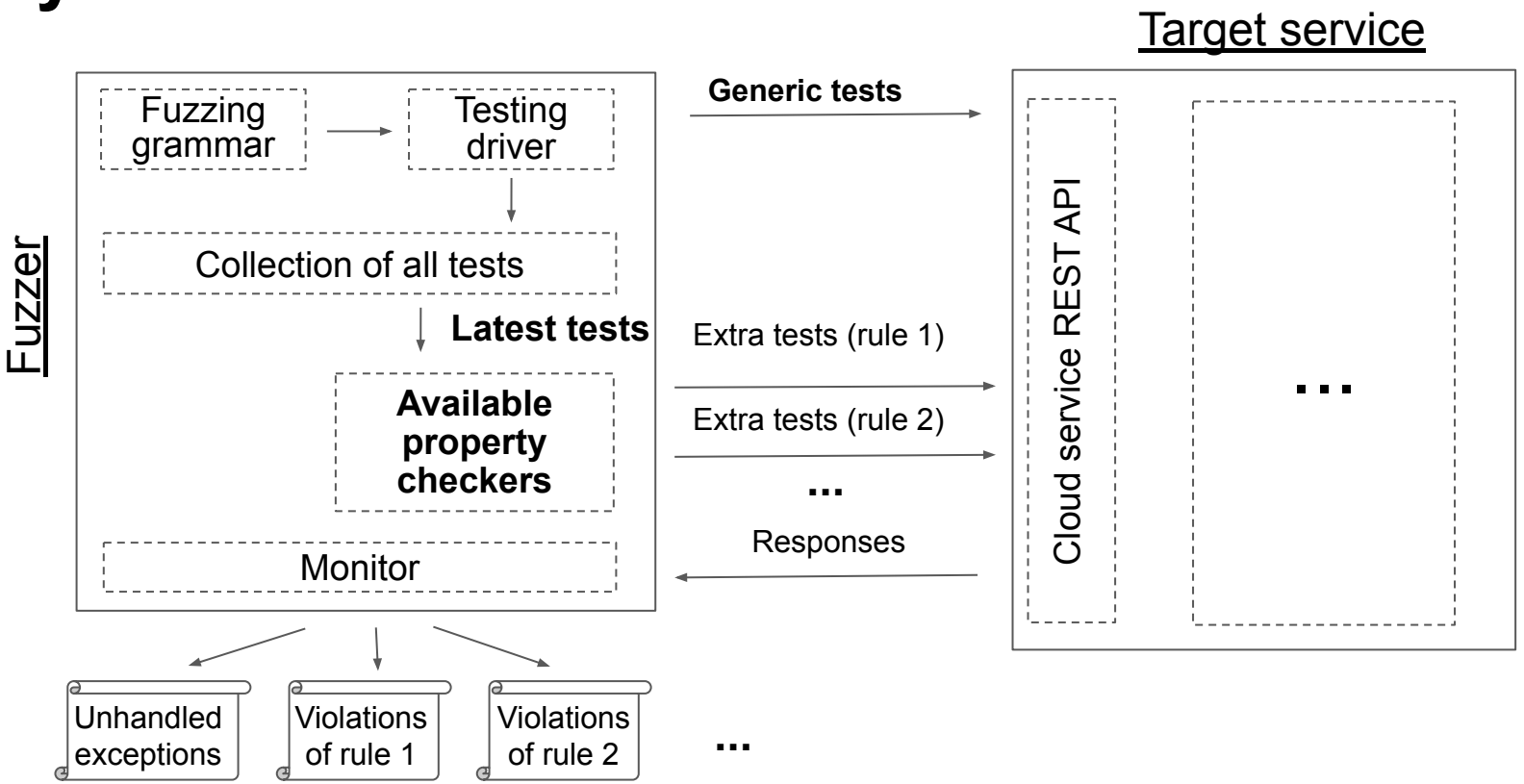
Selected security rules and desirable properties

- Use-after-free rule
  1. Delete /api/projects/1
  2. Access /api/projects/1 **MUST FAIL**

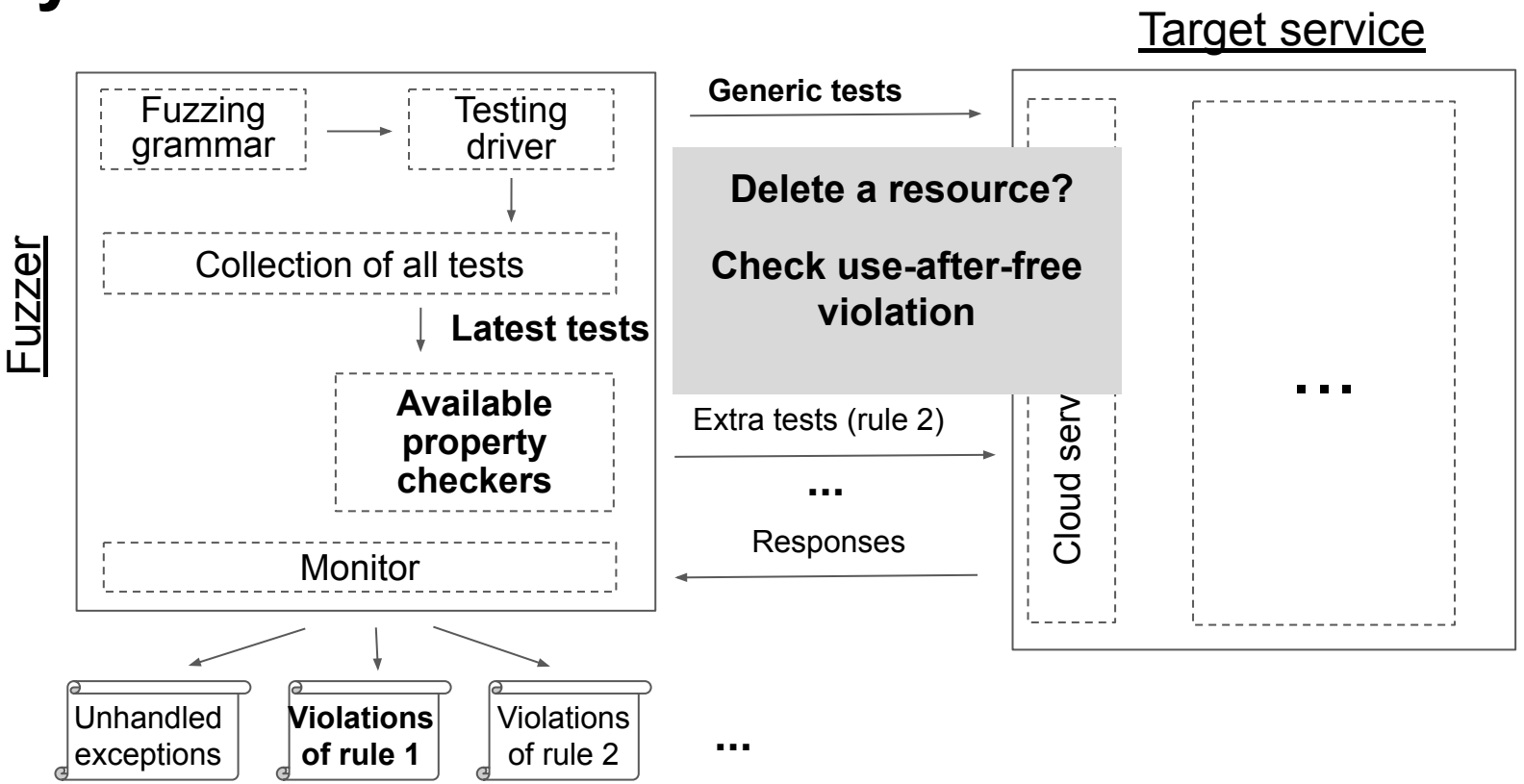
- Resource-hierarchy rule
  1. Create /api/projects/1 and /api/projects/2
  2. Create /api/projects/1/branches/1
  - ➢ Access /api/projects/2/branches/1 **MUST FAIL**

- Resource-leakage rule
  1. Create /api/projects/1 and receive error code (e.g., 404 or 500 HTTP status)
  - ➢ Access /api/projects/1 **MUST FAIL**

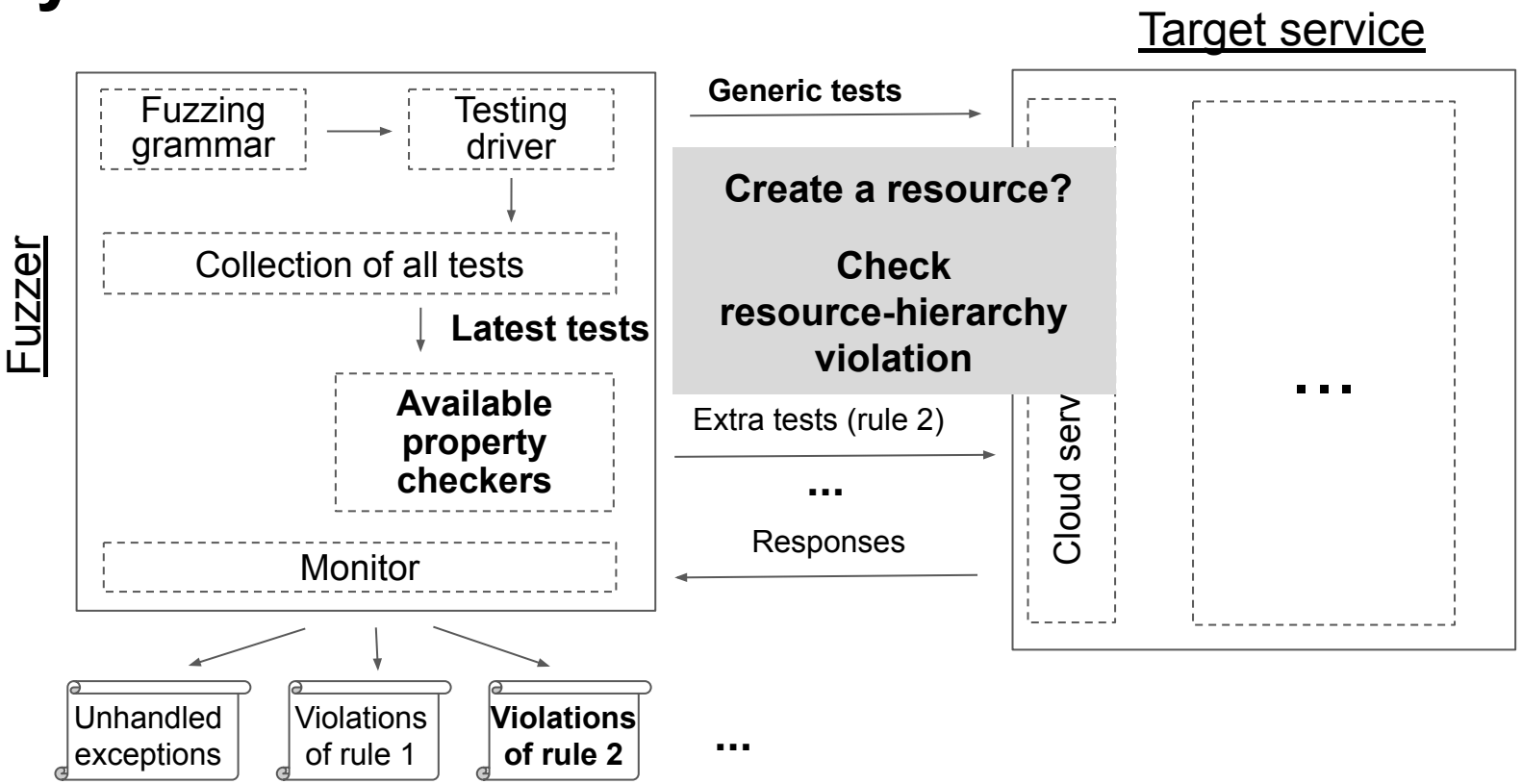
System overview



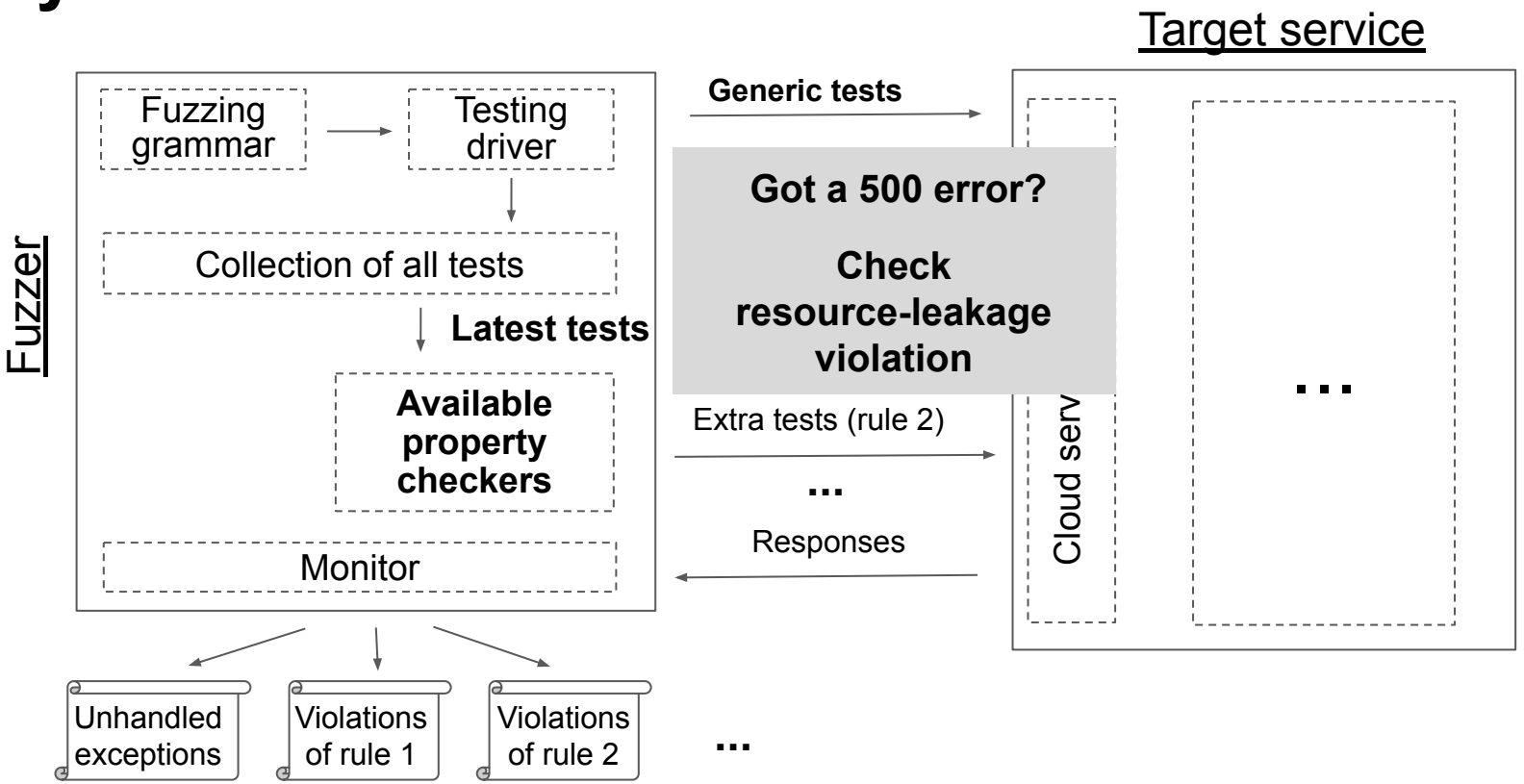
System overview



System overview



System overview



Selected errors found with checkers in Azure & O365

- Use-after-free rule violation
  1. Create a new resource R
  2. Delete resource R
  3. Create a new child resource as the deleted resource R
     - ➢ 500 "Internal Server Error" (should have been: 404 Not Found)
- Resource-hierarchy rule violation
  1. Create two messages (POST /api/posts/1212 and POST /api/posts/1313)
  2. Create a reply to the first message (POST /api/posts/1212/replies/12121)
  3. Edit the reply using the second message as parent (PUT /api/posts/1313/replies/12121)
     - ➢ 202 "Accepted" (should have been: 404 Not Found)
- Resource-leakage rule violation
  1. Create a resource of type T and name X with malformed body (this results in a 500 error)
  2. Get a list of all resource of type T: the returned result is empty
  3. Create a new resource of type T with the same name X in different region
     - ➢ 409 "Conflict" Inconsistent service state (should have been: 404 Not Found)

Main take-aways (so far)

- ✔ Introduced stateful REST API fuzzing
- ✔ Extended stateful REST API fuzzing to new classes of errors

Contributions

- RESTler: Stateful REST API Fuzzing (ICSE'19)
- Checking Security Properties of Cloud Service REST APIs (ICST'20)
- Pythia: Grammar-Based Fuzzing of REST APIs with Coverage-guided Feedback and Learning-based Mutations (target submission ICSE'21)

Structure and Feedback in Cloud Service API Fuzzing

Challenge

- Automatically-generated grammars have few, predefined fuzzing values and no code coverage feedback.

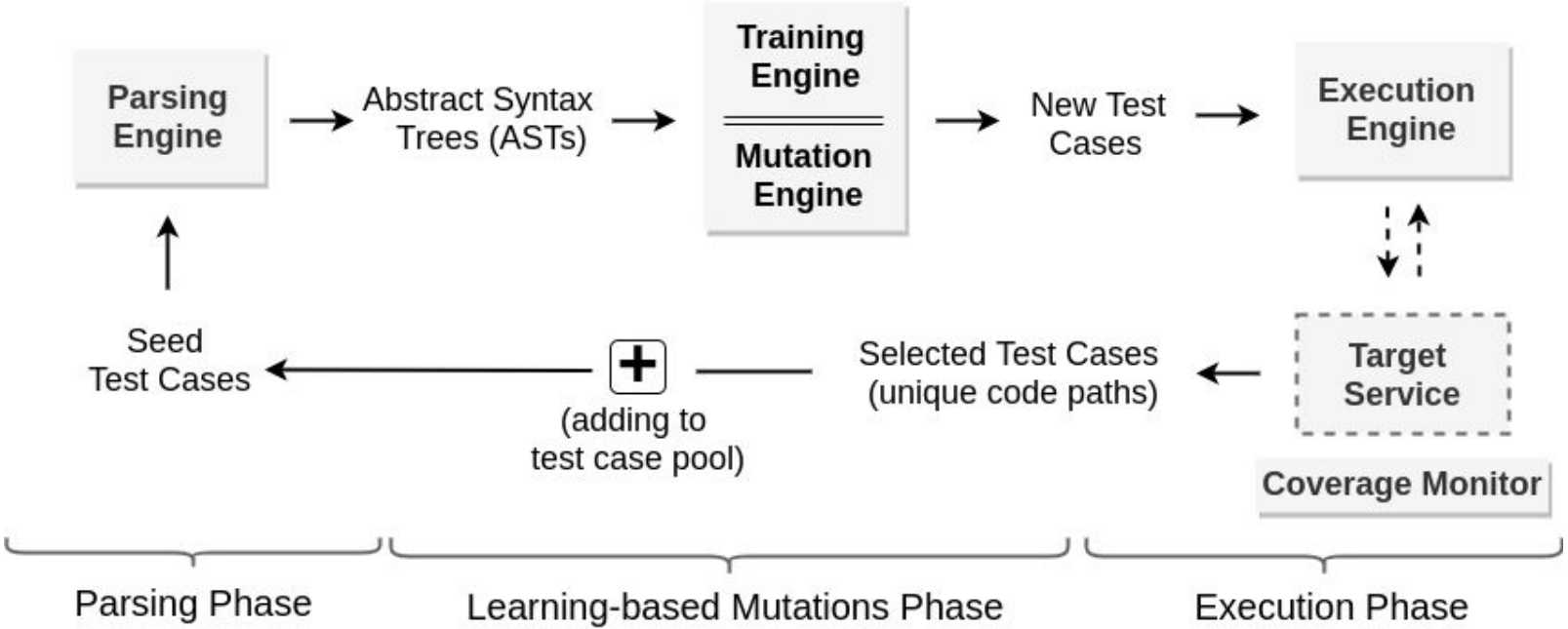
Pythia

- Learn common usage patterns from seed test cases
- Generate learning-based mutation & use code coverage to prioritize test cases

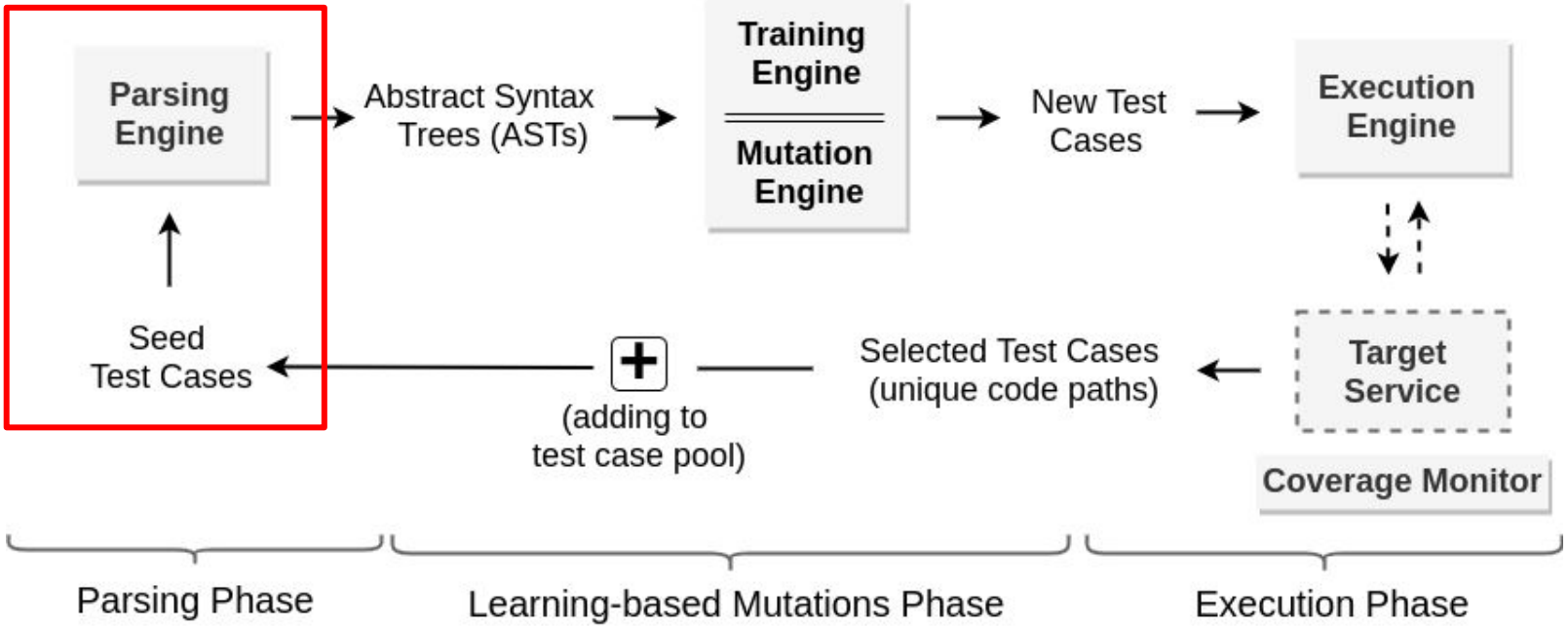
Kinds of errors RESTler can find

- “500 Internal Server Error” (unhandled exceptions)

System overview



System overview



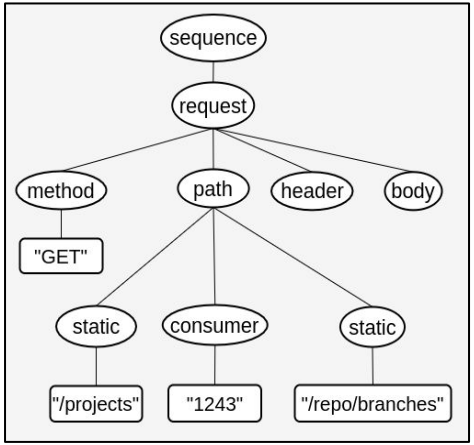
Parsing

- Parse seed test cases with a regular grammar and construct Abstract Syntax Trees (ASTs) from test cases
- Convert ASTs to sequences (of grammar rules)

GET /projects/1243/repo/branches

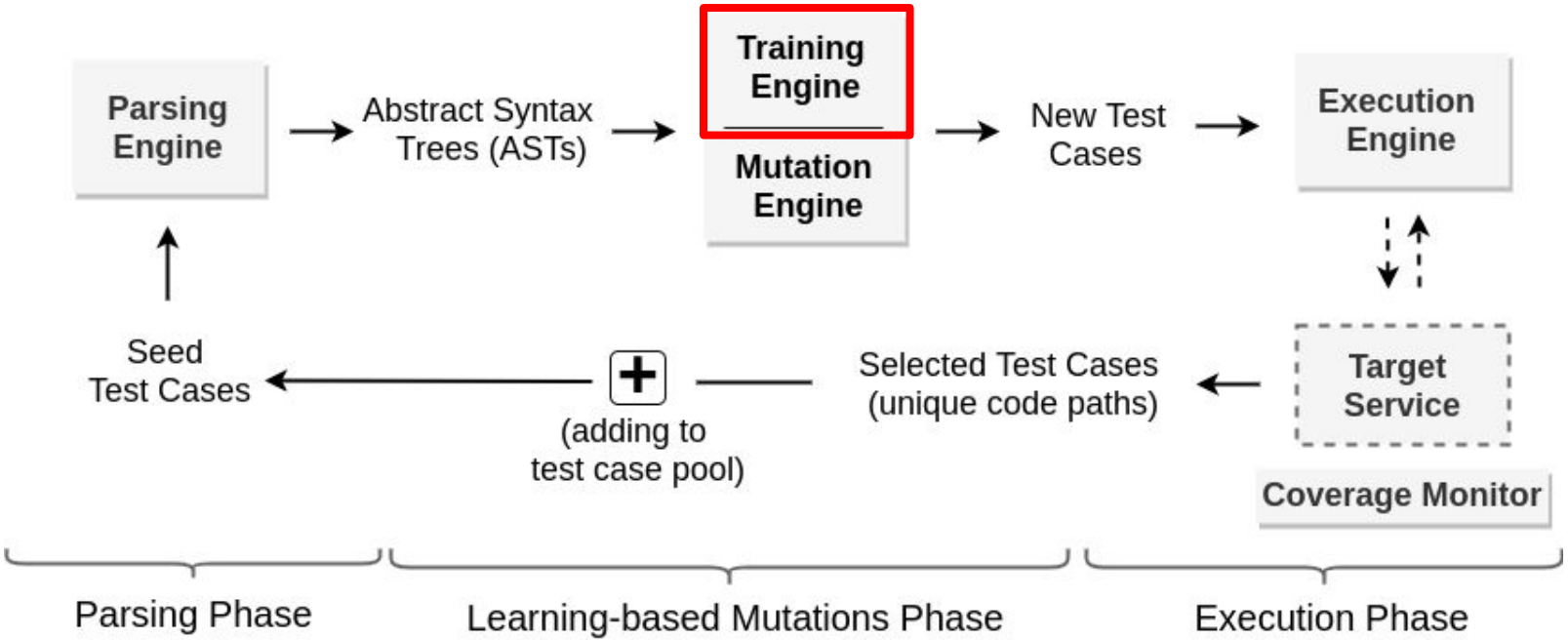
$G = (N, \Sigma, R, S)$

R1: sequence -> request + sequence
R2: request -> method + path + header + body
R3: method -> "GET"
R4: path -> static + path
R5: static -> "/projects"
R6: path -> consumer + path
R7: consumer -> "1243"
R8: path -> static + path
R9: static -> "/repo/branches"
R10: header -> e
R11: body -> e
R12: sequence -> e

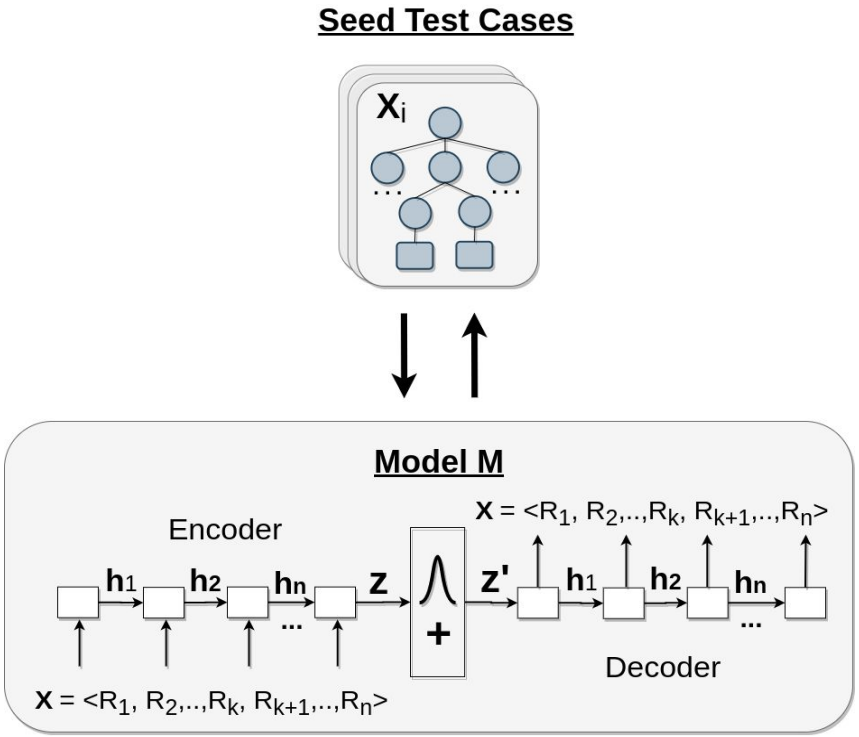


X = < R1,R2, R3, R4, R5, R6, R7, R8, R9, R10, R11, R12 >

System overview

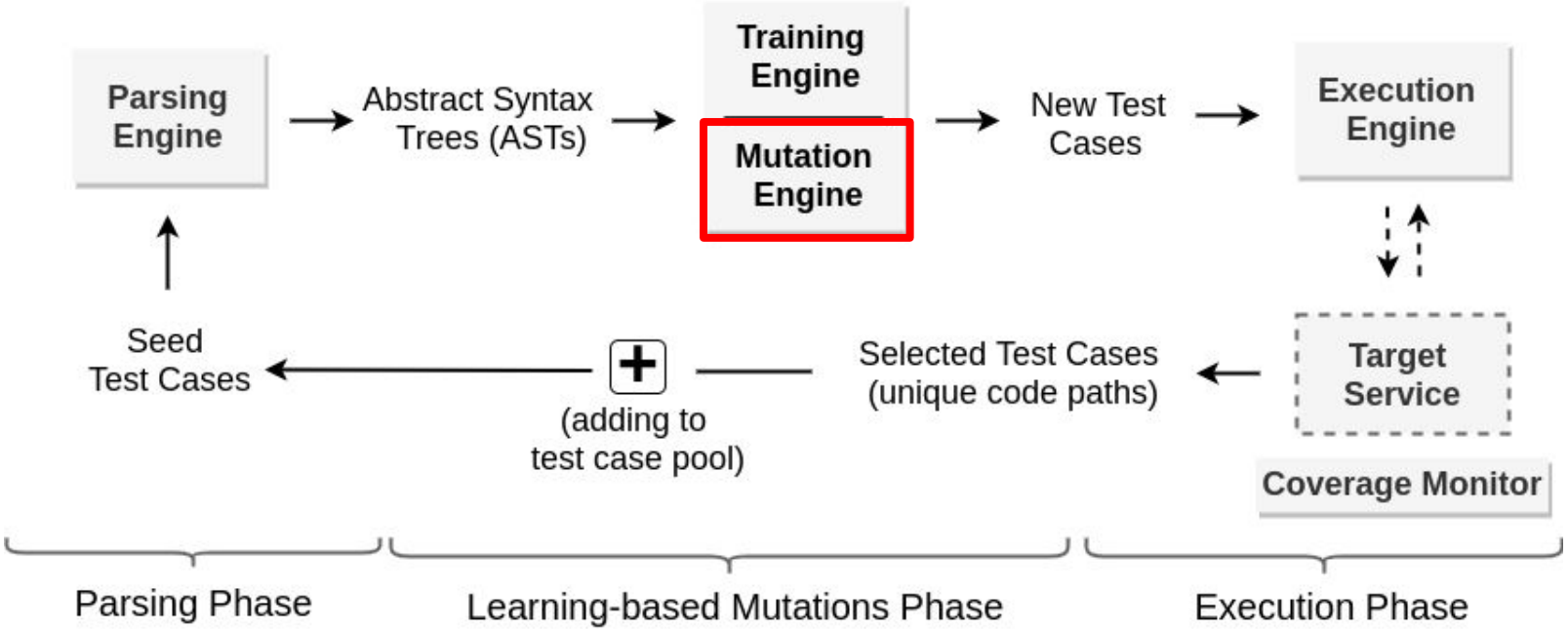


Training

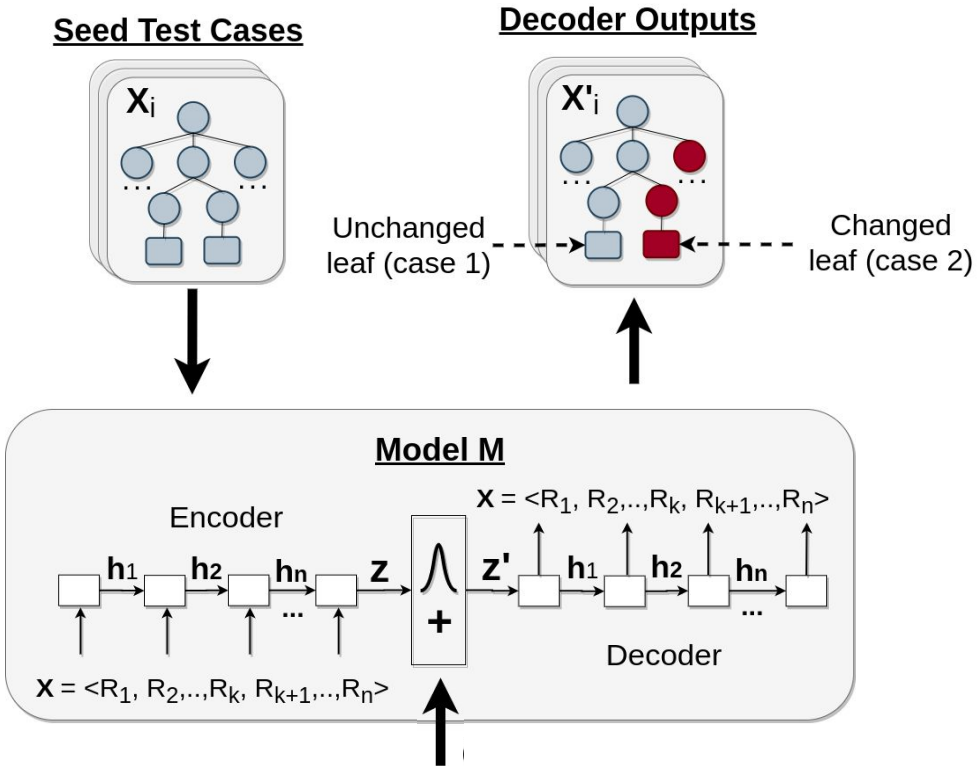


- Use grammatically valid seed test case sequences
- Train a seq2seq Recurrent Neural Network (RNN)
- Learn common usage patterns of a target cloud service

System overview

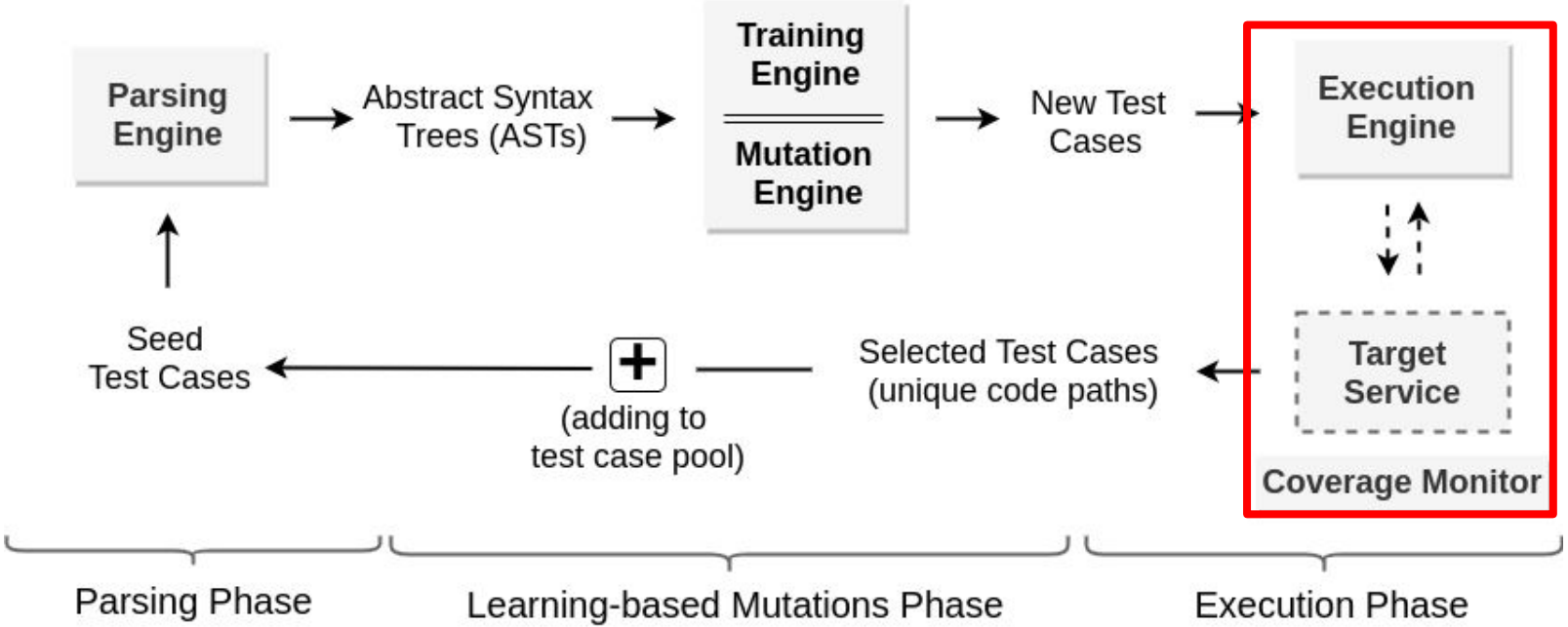


Mutations



- Feed seed test cases to encoder of trained model
- Perturb hidden state of encoder with random noise
- Compare decoder output with original test cases and generete new mutants

System overview



Selected evaluation results

- ➢ Q1: How does Pythia compare to other baselines w.r.t. code coverage?
- ➢ Q2: Can Pythia find bugs in production-scale cloud services?

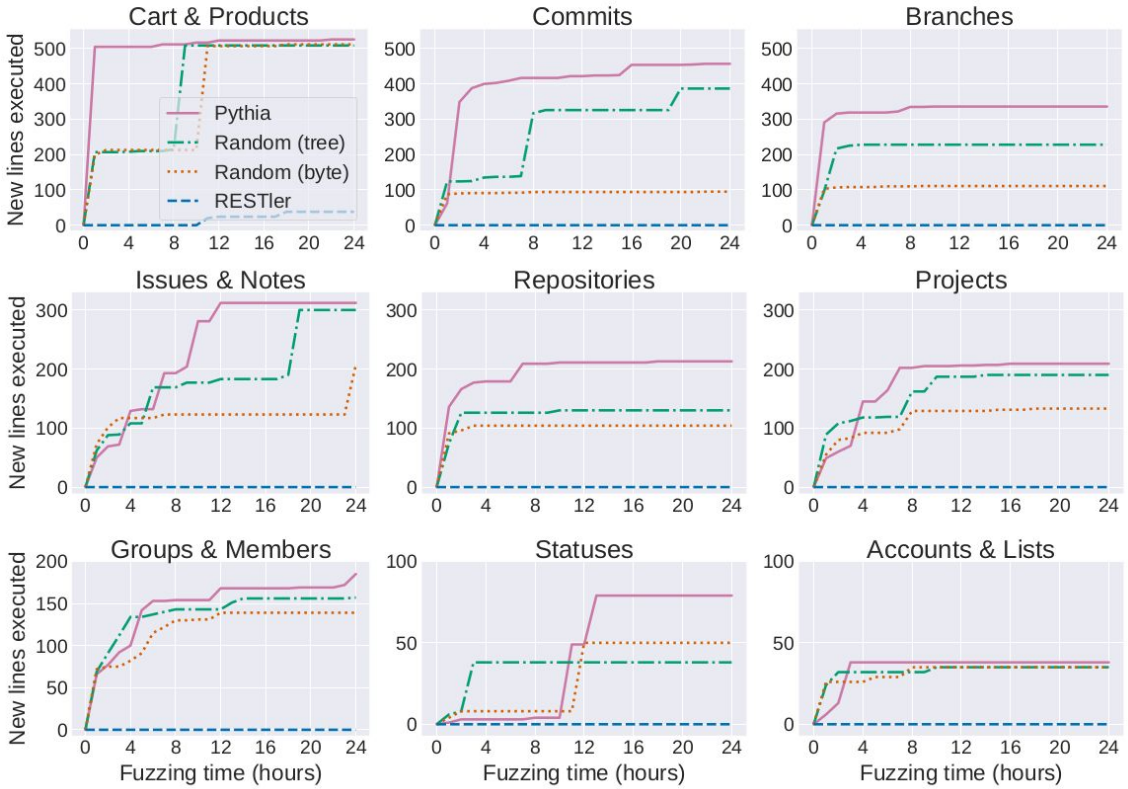
Study subjects

- ❖ Gitlab, Spree, and Mastodon APIs

Baselines

- ❖ RESTler: 24 hours initial seed creation & 24 hours fuzzing
- ❖ Byte-level mutations: Random byte alternations on seeds
- ❖ AST-level mutations: Mutations on AST representations

Code coverage (Q1)



- RESTler finds no new lines (plateau after the first 24 hours)
- Pythia finds new lines after RESTler plateau in all APIs
- Relative ordering is same in all APIs: Pythia > AST-level > byte-level mut.

New bugs found (Q2)

	RESTler			Pythia		
APIs	**Tests**	**500s**	**Bugs**	**Tests**	**500s**	**Bugs**
Commits	11.6K	0	0	10.7K	132	3
Branches	10.3K	0	0	12.3K	135	4
...	...	...	...	...	...	..
Statuses	58.8K	336	1	56K	962	1
Cart	15.5K	2018	1	18.7K	401	3
Total	194.5K	2.3K	**2**	220.5K	3.7K	**29**

Comparison of #tests and #errors after 24h fuzzing

- Both tools produce same order of magnitude of tests

New bugs found (Q2)

	RESTler			Pythia		
APIs	**Tests**	**500s**	**Bugs**	**Tests**	**500s**	**Bugs**
Commits	11.6K	0	0	10.7K	132	3
Branches	10.3K	0	0	12.3K	135	4
...	...	...	...	...	...	..
Statuses	58.8K	336	1	56K	962	1
Cart	15.5K	2018	1	18.7K	401	3
Total	194.5K	2.3K	**2**	220.5K	3.7K	**29**

Comparison of #tests and #errors after 24h fuzzing

- Both tools produce same order of magnitude of tests
- Pythia more 500s than RESTler

New bugs found (Q2)

	RESTler			Pythia		
APIs	**Tests**	**500s**	**Bugs**	**Tests**	**500s**	**Bugs**
Commits	11.6K	0	0	10.7K	132	3
Branches	10.3K	0	0	12.3K	135	4
...	...	...	...	...	...	..
Statuses	58.8K	336	1	56K	962	1
Cart	15.5K	2018	1	18.7K	401	3
Total	194.5K	2.3K	**2**	220.5K	3.7K	**29**

Comparison of #tests and #errors after 24h fuzzing

- Both tools produce same order of magnitude of tests
- Pythia more 500s than RESTler
- Pythia found previously-unknown bugs across all APIs

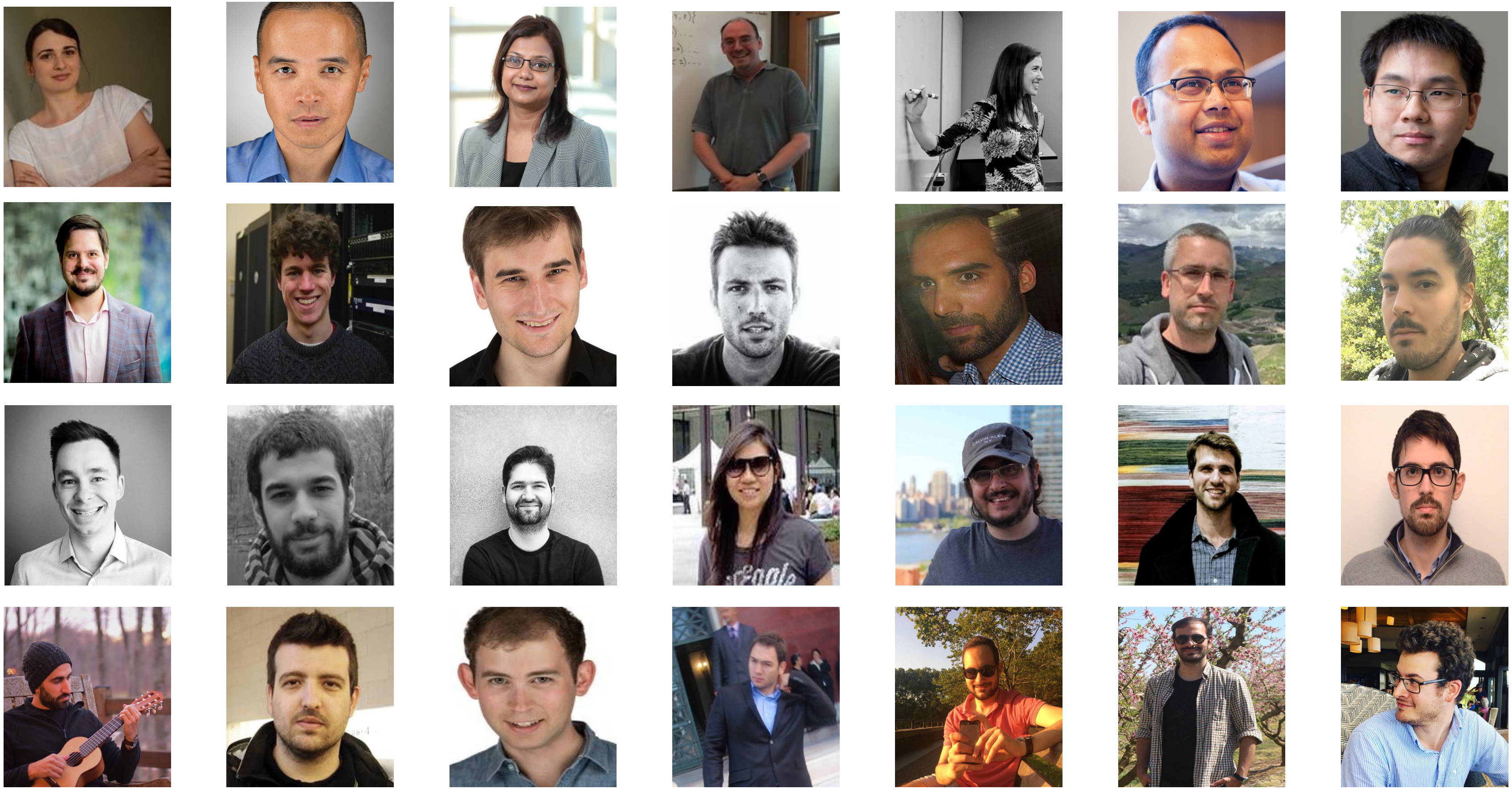
Main take-aways

- ✔ Introduced stateful REST API fuzzing
- ✔ Extended stateful REST API fuzzing to new classes of bugs
- ✔ Augmented stateful REST API fuzzing learning-based mutations and code coverage feedback

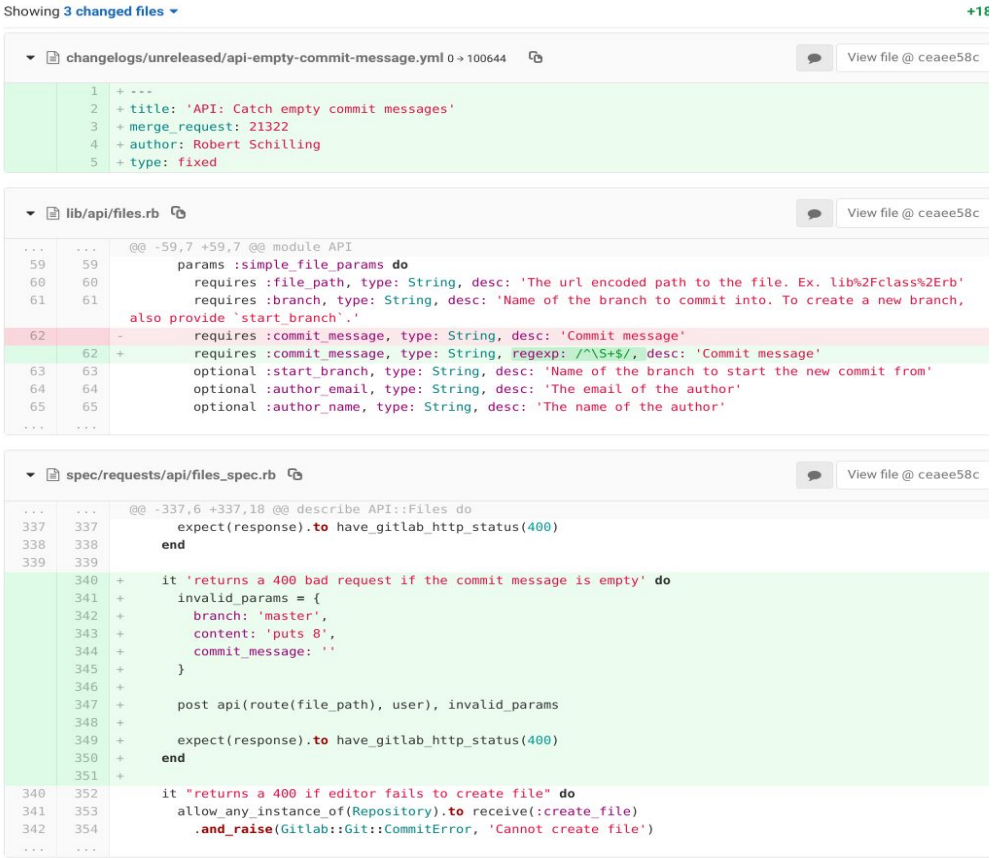
Conclusions

- Fuzzing cloud services by leveraging the structure of REST APIs and feedback obtained during interaction is new
- New types of bugs: Unlike buffer overflows, use-after-free bugs, or cross-site-scripting attacks, it is still unclear how serious the errors that hide behind REST APIs are
- Mark a clear path forward to test cloud services in automatic, efficient, and learning-based way

Thank you all!



Sample bugfix in Gitlab



Showing 3 changed files +18

changelogs/unreleased/api-empty-commit-message.yml 0 → 100644 — View file @ ceaee58c

```
+ ---
+ title: 'API: Catch empty commit messages'
+ merge_request: 21322
+ author: Robert Schilling
+ type: fixed
```

lib/api/files.rb — View file @ ceaee58c

```
@@ -59,7 +59,7 @@ module API
       params :simple_file_params do
         requires :file_path, type: String, desc: 'The url encoded path to the file. Ex. lib%2Fclass%2Erb'
         requires :branch, type: String, desc: 'Name of the branch to commit into. To create a new branch, also provide `start_branch`.'
-        requires :commit_message, type: String, desc: 'Commit message'
+        requires :commit_message, type: String, regexp: /^\S+$/, desc: 'Commit message'
         optional :start_branch, type: String, desc: 'Name of the branch to start the new commit from'
         optional :author_email, type: String, desc: 'The email of the author'
         optional :author_name, type: String, desc: 'The name of the author'
```

spec/requests/api/files_spec.rb — View file @ ceaee58c

```
@@ -337,6 +337,18 @@ describe API::Files do
       expect(response).to have_gitlab_http_status(400)
     end

+    it 'returns a 400 bad request if the commit message is empty' do
+      invalid_params = {
+        branch: 'master',
+        content: 'puts 8',
+        commit_message: ''
+      }
+
+      post api(route(file_path), user), invalid_params
+
+      expect(response).to have_gitlab_http_status(400)
+    end
+
     it "returns a 400 if editor fails to create file" do
       allow_any_instance_of(Repository).to receive(:create_file)
         .and_raise(Gitlab::Git::CommitError, 'Cannot create file')
```

Structure and Feedback in Cloud Service API Fuzzing

Developers' Responses

Patrice, thank you for reporting the bugs!
Plz provide instructions on how integrate the tool into the build

Please file VSO for each – these are real bugs. We already fixed a few in DNS.
•••