MINER: A Hybrid Data-Driven Approach for REST API Fuzzing

Chenyang Lyu1  Jiacheng Xu1  Shouling Ji1  Xuhong Zhang1  Qinyaing Wang1
Binbin Zhao2  Gaoning Pan1  Wei Cao3  Peng Chen3  Raheem Beyah2

1Zhejiang University  2Georgia Institute of Technology  3Ant Group
REST API is popular in cloud service but not secure.
## REST API Fuzzing

**RESTler**[1], the first stateful REST API fuzzer.

### Swagger Spec
- **POST** /groups
- **GET** /groups/{id}

### Request Grammar
- **POST** /groups?
  - name=abc
  - path=abc
  - HTTP/1.1
  - Host: 10.214.241.134
  - Response: 201 {"id": 23}

- **GET** /groups/23?
  - with_projects=true
  - HTTP/1.1
  - Host: 10.214.241.134
  - Response: 200 OK

---

Insight

- Parameters in a REST API request have explicit/implicit relations.
- Requests could be easily rejected irrespective of relations.
  - updated_before and order in GitLab Project API are compulsory.
  - import_url and initialize_with_readme in GitLab Project API are contradicting.
- Some parameter combinations could lead to server errors.
  - The co-existence of requirements_access_level, auto_cancel_pending_pipelines and initialize_with_readme in GitLab Group API could raise a server error.
Insight

<table>
<thead>
<tr>
<th>Gitlab API</th>
<th>Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td>/projects</td>
<td>72.01%</td>
</tr>
<tr>
<td>/groups</td>
<td>65.01%</td>
</tr>
<tr>
<td>/commits</td>
<td>53.52%</td>
</tr>
</tbody>
</table>

- **Low-quality Requests**
- **Short Sequences**

Swagger Spec → Request Grammar → Extend & Render → Tests & Bugs

![Graph showing distribution of request lengths](image)
How to **automatically** enhance testcases to pass syntax/sematic check and test deep logic?
MINER: a hybrid data-driven framework to fuzz REST APIs in cloud services and discover the corresponding logic vulnerabilities.
Collection Module

Requests Collection

- Filter the requests through corresponding responses.
- Collect effective request sequences (20X, 50X).
- Collect param-value pairs from effective requests (20X, 50X).
Training Module

注意力模型训练

- 学习参数之间从历史中隐含的关系。
- 将参数值对生成转换成文本生成。
- 将参数值对列表嵌入向量并训练模型。
Attention Model Usage

- Produce a set of lists of param-value pairs in which pairs are interrelated.
- The predicted pairs serve as candidates when MINER mutates requests.
- Training module is invoked periodically to update the model.
DataDriven Checker

- Target at the undefined parameter violation.
- The parameters are sampled from the collected pairs.
- Inject undefined parameters into the requests.
- Bug oracle: whether the mutate requests lead to error code.

Sequence Template Selection

- Prioritize the sequence templates whose lengths are longer
  \[ p = \log_{10}(l + 1) \]
  where \( l \) is the sequence length.
Experiment Settings

- Baselines includes RESTler, MINER_PART(without DataDriven Checker)
- Each evaluation lasts 48 hours.
- We evaluate MINER on GitLab, Bugzilla and WordPress with 11 APIs.

<table>
<thead>
<tr>
<th>Cloud Service</th>
<th>REST API</th>
<th># Request Templates</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GitLab 14.1.0-ce.0</td>
<td>Projects</td>
<td>29</td>
<td>REST API to interact with projects of GitLab</td>
</tr>
<tr>
<td></td>
<td>Groups</td>
<td>13</td>
<td>REST API related to groups of GitLab</td>
</tr>
<tr>
<td></td>
<td>Issues</td>
<td>23</td>
<td>REST API to interact with GitLab issues</td>
</tr>
<tr>
<td></td>
<td>Commits</td>
<td>14</td>
<td>REST API related to commits of GitLab</td>
</tr>
<tr>
<td></td>
<td>Branches</td>
<td>8</td>
<td>REST API to interact with GitLab branches</td>
</tr>
<tr>
<td>Bugzilla 5.0.4</td>
<td>Comments</td>
<td>8</td>
<td>REST API to maintain comments in Bugzilla</td>
</tr>
<tr>
<td></td>
<td>Bugs</td>
<td>11</td>
<td>REST API to maintain bug reports in Bugzilla</td>
</tr>
<tr>
<td></td>
<td>Groups</td>
<td>6</td>
<td>REST API to maintain user groups in Bugzilla</td>
</tr>
<tr>
<td>WordPress 5.8.1</td>
<td>Categories</td>
<td>5</td>
<td>REST API to maintain categories in WordPress</td>
</tr>
<tr>
<td></td>
<td>Posts</td>
<td>10</td>
<td>REST API to interact with posts in WordPress</td>
</tr>
<tr>
<td></td>
<td>Comments</td>
<td>8</td>
<td>REST API to maintain comments in WordPress</td>
</tr>
</tbody>
</table>
Fuzzing Performance Analysis

Results

- MINER_PART and MINER achieve 23% higher pass rate and covers 7% more unique request templates than RESTler.
- MINER discovers 15.27 unique errors while MINER_PART only discovers 9.91 and RESTler discovers 7.73 unique errors on average.
- MINER also uniquely finds 5.45 unique errors caused by misuse of undefined parameters using DataDriven Checker.
Significance of Designs

- Four variations: RESTler+Seq, RESTler+Rec1, RESTler+RecList, RESTler+Model

Results

- Both the model and the sequence construction improve pass rate and generate longer sequences.
- The model can improve the ability of error discovery.
- Recording and replaying the history mechanically are low efficiency.
Coverage Analysis

- Hook GitLab’s source code to trace line coverage using Coverband\textsuperscript{[2]}.

- Results
  - Both the model and the sequence construction improve the line coverage.
  - MINER\_PART achieves the most line coverage due to the data-driven designs.

\textsuperscript{[2]} Coverband. https://github.com/danmayer/coverband
Analysis on Reproducing Published Serious Bugs

_aspect Collect 4 published serious bugs manually as the ground truth.
_aspect Results

• MINER can reproduce all the bugs more effectively and efficiently compared to RESTler.
• MINER not only finds relatively shallow bugs, but also specializes in constructing complex sequences.

<table>
<thead>
<tr>
<th>No.</th>
<th>API</th>
<th># Request Templates</th>
<th>Minimum Sequence Length To Trigger Bug</th>
<th>RESTler</th>
<th>MINER</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Projects</td>
<td>3</td>
<td>3</td>
<td>N/A</td>
<td>66.5 mins</td>
</tr>
<tr>
<td>2</td>
<td>Projects</td>
<td>4</td>
<td>5</td>
<td>N/A</td>
<td>142.6 mins</td>
</tr>
<tr>
<td>3</td>
<td>Groups</td>
<td>1</td>
<td>1</td>
<td>27.6 mins</td>
<td>26.6 mins</td>
</tr>
<tr>
<td>4</td>
<td>Groups</td>
<td>1</td>
<td>1</td>
<td>13.7 mins</td>
<td>14.2 mins</td>
</tr>
</tbody>
</table>
Real Bug Analysis

**Results**

- MINER finds all the bugs discovered by RESTler, and also finds 10 extra bugs, including 4 parameter misuse bugs.
- MINER could discover bugs effectively, especially the ones requiring long request sequences and specific parameter combinations due to data-driven designs.
- Example: The third request updates a hook with three undefined parameters "requirements_access_level", "auto_cancel_pending_pipelines" and "initialize_with_readme" will raise a server error.

```
1. POST /api/v4/projects
2. POST /api/v4/projects/:proj_id/hooks
3. PUT /api/v4/projects/:proj_id/:hooks/:hook_id
```

```
1. 201 Created
2. 200 OK
3. 500 Internal Error
```
Analysis with Different Training Duration

Evaluate the impact of different schedules of the Training Module.

Results

• The training overhead increases when using a larger iteration duration.
• The pass rate slightly decreases when using a longer iteration duration.
• MINER’s data-driven approaches have minimal extra costs.
A novel data-driven REST API fuzzer named MINER, with 3 designs against low-quality requests, short sequences and parameter misuse.

Evaluate MINER on three cloud services via 11 REST APIs. MINER achieves 23% higher pass rate, find 100% more reproducible errors than RESTler. 17 new bugs are found by MINER (4 from DataDriven Checker).

Conduct comprehensive to demonstrate the outstanding performance of MINER.
Contact: puppet@zju.edu.cn
& stitch@zju.edu.cn