A study of bugs found in the Ansible configuration management system

Matas Rastenis (m.rastenis@student.tudelft.nl)[1]

Abstract

- Research that focuses on examining software bugs is critical when developing tools for preventing and for fixing software issues.
- The foundational aim is to help improve the quality of the Ansible open source configuration management system[1]
- This study defines a data pipeline and custom tools to extract and analyze 100 Ansible bugs. Common classifications are determined, and the bugs are manually classified, revealing common patters within the bugs.
- Results: most bug prone areas are executor and connectivity components, fuzz testing for vulnerable input configurations and genetic algorithm testing for expanding coverage is recommended.

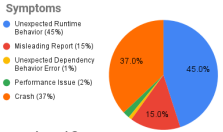
Research Questions

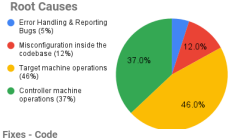
- "What common patterns can be extracted from the bugs found and what are the root causes, symptoms, triggers, system-dependence factors, fixes, and the impact of the most frequent types of bugs in the Ansible configuration management system.

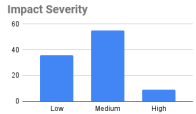
1. What are the common patterns of Ansible bugs?
2. What are the main symptoms of common Ansible bugs?
3. What are the main root causes of common Ansible bugs?
4. What is the impact of common Ansible bugs?
5. What are the triggers of common Ansible bugs?
6. What is the impact of common Ansible bugs?
7. Are these bugs system dependent?

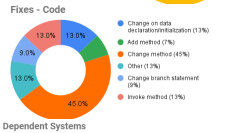
Results

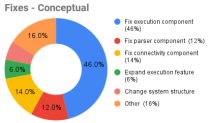
59448 total issues.
28511 Bugs.
15338 Bugs with PRs.
71 Verified bugs.
735 Bugs with the 'has_pr' label.
581 Documentation Bugs
2610 Unmerged PRs.
3605 Bugs with merged fixes.

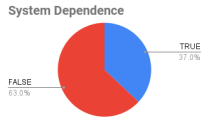


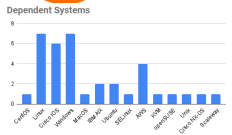


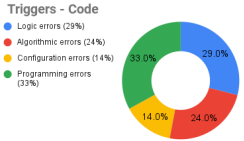


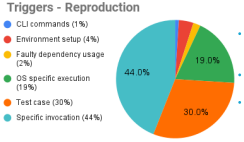












Conclusions:

- Most bug-prone components are execution components and connectivity components.
- Fuzz testing recommended due to the prevolence of specific invocation trigger [2].
- Code review process enhancements proposed.
- Recommendation to invest in genetic algorithm test case discovery for improving test coverage [3].

Method

The officially recognized source for bug tracking, Github Issues, was used to fetch all of the relevant data. In total, 100 Ansible bugs were analyzed. Custom tooling was created to pull, filter and serialize raw data. Then, manual analysis of the bugs produced the classifications. The next steps are to analyze the bugs in accordance with the research question.

The method is as follows:

1. Use Perceval to fetch raw data from Github.
2. Pre-process the data into a parsable format.
3. Perform filtering and extract summary from the pre-processed data.
4. Prune unneeded data (post-filtering).
5. Serialize the bugs into a MySQL database with a standardized schema.
6. Sample an amount of bugs for analysis.
7. Manually analyze bugs and create common characterizations.
8. Assign categories to answer all of the research questions.
9. Derive insights and make recommendations.

References

1. Red Hat Ansible. Ansible is Simple IT Automation. 2022. url: https://www.ansible.com/?hsLang=en-us
2. Sanjeev Das et al. "A Flexible Framework for Expediting Bug Finding by Leveraging Past (Mis-)Behavior to Discover New Bugs". In: Annual Computer Security Applications Conference (2020). doi: 10.1145/3427228.3427269.
3. Rizal Broer Bahaweres et al. "Analysis of statement branch and loop coverage in software testing with genetic algorithm". In: 2017 4th International Conference on Electrical Engineering, Computer Science and Informatics (EECSI) (2017). doi: 10.1109/eecsi.2017.8239088

[1]: Supervised by Thodoris Sotiropoulos (theosotr@aueb.gr).