Milestone 5: Robustness Testing

Released: Wednesday, Apr 9, 2025
Due: **11:59 pm Friday, Apr 18, 2025**

Learning Objectives

- Test and evaluate a software component/service for robustness using an API specification.
- Redesign the component/service to improve its robustness.

Recommended Resources

- https://www.geeksforgeeks.org/robustness-testing/
- https://www.geeksforgeeks.org/security-testing/
- https://www.geeksforgeeks.org/software-testing-scalability-testing/
- Fuzz Testing:
  - https://www.freecodecamp.org/news/web-security-fuzz-web-applications-using-ffuf/
  - https://www.fuzzingbook.org/html/WebFuzzer.html
  - https://medium.com/@cuncis/fuzzing-made-easy-how-to-use-wfuzz-for-efficient-web-application-testing-d843e5b089bf
  - https://github.com/marmelab/gremlins.js
- Bug Report Writing: https://www.softwaretestinghelp.com/how-to-write-good-bug-report/

Milestone Tasks

In this milestone, your team will evaluate the robustness of the service developed by another team by trying to “break” it. Your team will be assigned the service to test in a separate message from the course staff.

Task 1: Designing Robustness Tests

Given the API specification of the service under test (SUT), identify faults that could break the SUT. These could be malformed requests, unexpected corner cases, malicious inputs, or creating an overload situation. Your goal is to identify multiple ways in which the SUT might break and describe potential ways for you to inject these faults into the system using the API provided to you.

Be **creative:** Think of many different ways to break the SUT. You can consider possible robustness issues, security attacks, or overload situations that expose scalability issues.

Try to think of **potential implementation shortcuts** the development team might have taken: Given the time constraints, the development team most likely identified certain inputs as not

likely to happen and did not consider them in their design. If you would have been on their team, which inputs would you have ignored? These are potential candidates for you to test!

Also, to increase your likelihood of finding issues, we recommend you to think of ways to **increase the coverage** of your robustness tests (i.e, efficiently covering many possible inputs - see the ressources on fuzz testing above). How can you generate a large range of inputs that are likely to break the SUT? We do not have requirements for minimum coverage. This note is just to help you find issues more efficiently.

Task 2: Executing Robustness Tests

Execute the tests that you described above. These can be automated, semi-automated, or manual. If possible, consider the use of a tool to automate the test execution. You may, for example, use a stress/robustness testing tool such as locust.io (quick start guide) or gremlins for this purpose.

Please be courteous to the team that has built the SUT. Before running your tests, please let them know ahead by sending them a message. If you actually manage to crash the SUT, request a restart of the service and try to avoid running the same test again to avoid forcing restarts more often than necessary.

You have successfully found an issue if:

- the SUT crashes
- the SUT returns factually incorrect results or results that are different from the API specification
- the response takes much longer than expected or the SUT does not respond at all
- the SUT returns an error message even though the inputs conformed to the API specification
- you gained unauthorized access to data within the service, commands within the service, or the VM inside which the service runs (please do not attempt to extract real personal information, such SSH keys, credentials, or personal files, from the VMs of other students)

If possible, please try to identify **at least two** issues in the SUT. The issues can be of the same general type (i.e., service crashes on incorrect input), but should be meaningfully different from each other (e.g., “failing on -1, -2, -3” and “failing on negative inputs” together are considered one issue instead of two, but “failing on negative inputs” is considered a different issue from “failing for extremely large inputs”). You will receive **bonus points** for discovering issues from more than **two** categories in the above list.

Task 3: Reporting Results

Send a report containing your tests and results to the team that built the SUT by **Tuesday, April 15**. Provide them with steps to reproduce the issues that you identified (e.g., sequence of API

requests or user inputs). If you used scripts to simulate requests, please also provide them with these scripts to make it easier for them to reproduce the results.

In the unlikely scenario that you were not able to find any issues with the SUT, send a report containing your tests only. In your report, provide an argument that your tests cover a diverse set of scenarios and inputs.

Task 4: Improving Robustness

For each of the two issues that the other team found in your service, describe a possible design modification for improving the robustness of your system. Provide a justification for how this modification improves robustness; this may include (although not necessary) a fault tree that shows how the modification increases the size of the minimum cut set for a particular failure. If applicable, refer to a design pattern or principle discussed in one of the lectures on robustness, scalability, or security.

Given time constraints, you are **NOT** required to actually implement the robustness improvements that you come up with. However, you will receive **bonus points** for any improvement that you implement and demonstrate by re-running the relevant test and showing that it no longer causes a failure.

Deliverables

Submit a report as a single PDF file to Gradescope that covers the following items in clearly labeled sections (ideally, each section should start on a new page). **Please correctly map the pages in the PDF to the corresponding sections.**

Test Report: Describe your test design, results that you found, and steps to reproduce the issues. Your test report should describe how you selected the inputs that you test, how you execute the tests, and how you assert that an issue is present. For creating bug reports and reproduction steps, please see these guidelines for writing effective bug reports: https://www.softwaretestinghelp.com/how-to-write-good-bug-report/. Please send this report to the development of the SUT by **Tuesday, April 15**, and submit a copy on Gradescope by the **Friday**, **April 18** deadline.

Robustness Improvement Report (**3 pg max**): Describe modifications to the design of your service that would improve its robustness against the issues found by the other team. For each of the two, justify how the modification improves robustness. Optionally, if the modification is implemented, provide a brief description of the implementation and a link to the part of your source code that contains the modification.

Grading

This assignment is out of **70** points. For full points, we expect:

- **(40 pt)** A testing report that clearly describes your approach to identify potential robustness issues with the service of another team, presents your findings (including at least two found issues), and clearly describes steps to reproduce the issue. **5 bonus points** for issues from more than two categories found.
- **(20 pt)** A report describing possible robustness improvements to your system and a discussion of alternative solutions (Task 4). **Bonus points** for actually implementing and demonstrating an improvement (5 bonus points per each improvement, up to 10 points in total).
- **(10 pt)** A team contract that describes (i) the division of tasks among team members and (ii) a set of intermediate milestones, their target date, and expected goals.
- **(5 pt)** Bonus social points (same as the previous milestones).