ROS Fuzzer Documentation

Release 1.0

Alias Robotics S.L.

Aug 30, 2020

Contents:

CHAPTER 1

Installation

Install the ROS fuzzer by directly downloading from PyPi:

```
pip install ros_fuzzer
```

Otherwise, the fuzzer can be directly installed by cloning the repository and calling:

```
pip install .
```

For development purposes, it's recommended to install the symlinked version of the application.

```
pip install -e .
```

CHAPTER 2

Fuzzer usage

The fuzzer works in a standalone manner, by calling it to fuzz a full message structure via CLI, or by designing custom tests that fuzz or exclude different fields of the messages.

2.1 CLI Usage

The fuzzer can be directly invoked from the command line. Ensure that the ROS workspace is sourced before proceeding. Message types follow the ROS naming scheme.

```
$ source /opt/ros/melodic/setup.bash
$ ros_fuzzer -t <topic> -m <message_type>
```

Here is a usage example for performing a Log message fuzzing over the /rosout topic:

```
$ source /opt/ros/melodic/setup.bash
$ ros_fuzzer -t /rosout -m rosgraph_msgs/Log
```

2.2 Usage as Unit Testing counterpart

Test cases can use the provided Hypothesis strategy generation functions to get fuzzed messages that can be modified and used for different purposes. Fuzzed test cases follow the same mechanisms as standard data test cases, obtaining the fuzzed message as a parameter to the test case. The following example shows a simple test case that makes use of a fuzzed Log message, that is then modified before being sent.

Listing 1: Example unittest test case

```
@given(log=map_ros_types(Log))
def test_fuzz_log_message_exclude(self, log):
    log.name = 'Fixed name'
    self.pub.publish(log)
```

The `ros1_fuzzer.ros_commons.map_ros_types()` function provides a dynamic strategy for the defined ROS Message class, that correctly sets up each of the elements of the message with the corresponding data type fuzzers. Examples can be extended to even fuzz different message types or sub-elements independently. Built in hypothesis `hypothesis.strategies` can be used as well. The `hypothesis.given()` decorator runs the decorated function with all the defined fuzz cases.

Listing 2: Example unittest with multiple parameters.

```
@given(log=map_ros_types(Log), header=map_ros_types(Header), name=st.text(min_
↪length=1, max_length=20))
def test_fuzz_log_message_parameters(log, header, name):
    log.name = name
    log.header = header
    self.pub.publish(log)
```

The following examples show a trajectory message fuzzer utilized for fuzzing the ABB control node of the ROS Industrial project. Notice the settings block, which serves as a way to set the fuzz cases to launch, and the output of the fuzzer. The `ros1_fuzzer.process_handling.FuzzedLocalProcessHandler` class serves to detect changes in the target node, detecting when this node has crashed.

Listing 3: Joint Trajectory message fuzzing used on REDROS-I ROSIN project for ABB node fuzzing.

```
@settings(max_examples=5000, verbosity=Verbosity.verbose)
@given(array(elements=float64(), min_size=6, max_size=6))
def test_fuzz_message_jointstate_effort(process_handler, fuzzed_fields):
    joint_state_message.effort = fuzzed_fields
    self.pub.publish(joint_state_message)
    assert self.process_handler.check_if_alive() is True

@settings(max_examples=5000, verbosity=Verbosity.verbose)
@given(array(elements=float64(), min_size=6, max_size=6),
       array(elements=float64(), min_size=6, max_size=6),
       array(elements=float64(), min_size=6, max_size=6))
def test_fuzz_message_jointstate_all(process_handler, positions, velocities, efforts):
    joint_state_message.position = positions
    joint_state_message.velocity = velocities
    joint_state_message.effort = efforts
    self.pub.publish(joint_state_message)
    assert self.process_handler.check_if_alive() is True
```

2.3 Usage of node health checkers

A health checker for detecting node process crashes has been implemented as well. This way, assertions on the node process status can be performed. The health checker is instantiated prior to starting the tests, passing the node name as an argument. Currently, a local process `ros1_fuzzer.process_handling.FuzzedLocalProcessHandler` health checker is supported.

Listing 4: Example TestSuite setup function with node health check initialization.

```
def setUp(self):
    self.process_handler = FuzzedLocalProcessHandler('/example_node')
```

The health checker can then be part of assert clauses on tests, by calling the `ros1_fuzzer.process_handling.FuzzedLocalProcessHandler.check_if_alive()` function.

Listing 5: Example test case with node health checking.

```
@given(array(elements=float64(), min_size=6, max_size=6))
def test_fuzz_message_jointstate_effort(process_handler, fuzzed_fields):
    joint_state_message.effort = fuzzed_fields
    self.pub.publish(joint_state_message)
    assert self.process_handler.check_if_alive() is True
```

CHAPTER 3

ROS fuzzer modules

CHAPTER 4

Acknowledgements

This project has been funded by the ROSIN Project as part of the REDROS-I project. Contract agreement No.: 732287

Based on the original idea and Hypothesis-ROS project by Florian Krommer

This is the documentation for the ROS1 Fuzzer developed by Alias Robotics. This fuzzer allows to test ROS applications against limit cases that could cause failures or pose a security risk. Currently the fuzzer allows to test ROS messages of any type by dynamically setting up the fuzz cases for each of the message elements.

CHAPTER 5

Indices and tables

- genindex
- modindex
- search