DIFFER

DIFFER: Detecting Inconsistencies in Feature or Function Evaluations of Requirements

Setup

Installing Dependencies

1. Install Python 3.9 and dependencies. For Ubuntu 20.04, the deadsnakes PPA can be used: `$ sudo add-apt-repository ppa:deadsnakes/ppa $ sudo apt update $ sudo apt-get install python3.9 python3.9-venv libfuzzy-dev lftp lighttpd memcached tcpdump binutils $ sudo systemctl stop memcached $ sudo systemctl disable memcached`
2. Install `pipenv`, which manages the virtual environment. `$ python3.9 -m pip install pipenv`
3. Create the virtual environment and install development dependencies: `$ pipenv sync --dev`
4. Install Node.js, which is required by the type checker (pyright). On Linux, use the node version manager and on Windows install Node.js 18+ and add `node.exe` to the `PATH`.

Allow current user to execute tcpdump

The current user will need to be able to run `tcpdump` without `sudo` in order for the packet capture functionality to work properly. This is not necessary if DIFFER is being run as `root`.

1. Enable the network traffic capture capabilities for the `tcpdump` binary. `$ sudo setcap cap_net_raw,cap_net_admin=eip /usr/sbin/tcpdump`

2. Verify that you can run `tcpdump` without sudo. The following command should work properly and produce a pcap file. ``` $ tcpdump -i lo -w test.pcap # wait a few seconds # ctrl+c

$ ls -l test.pcap # verify that the file exists and is not empty ```

Running Differ

Sample Project Configuration

```
# Sample project configuration file: project.yml

# Unique name
name: coreutils_echo

# Path to the original binary
original: /usr/bin/echo
# List of debloated binaries to test against. The key is the debloater name and the value
# is the path to the debloated version of the original binary.
```

```
debloaters:
  # Replace this path to the debloated version
  binrec: /usr/bin/echo

# List of templates to generate, run, and compare against the original binary
templates:
  # command line arguments (supports Jinja2 templates from variables)
  - arguments: '{{left}} + {{right}}'

    # Fuzzing variables. The variables are generated and populated into the command line
    # arguments and any template input files for each run.
    variables:
      left:
        type: int
        range:
          # generate 5 integers in the range of 0-99 (inclusive)
          minimum: 0
          maximum: 99
          count: 5

      right:
        type: int
        # Use the following 3 int values
        values:
          - -1
          - 0
          - 1

    # List of comparators that verify the debloated version
    comparators:
      # verify stdout matches
      - stdout

      # verify stderr matches
      - id: stderr

      # verify the exit code is identical
      - exit_code
```

To run this project:

```
$ pipenv run python -m differ --verbose project.yml
```

The output is stored in the `./reports` directory by default and only errors are recorded. To change the output directory and output all reports, including successful runs:

```
$ pipenv run python -m differ --verbose --report-successes --report-dir ./output project.yml
```

Reports are stored in `{report_dir}/{project.name}/report-{engine}-[success|error]-{trace.id}.yml`. For example, a trace of the `binrec` debloater for the `coreutils_echo` project that failed would have a report located at:

```
$ cat ./reports/coreutils_echo/report-binrec-error-001.yml

arguments:
- '70'
- +
- '-1'
binary: /usr/bin/echo-binrec
results:
- comparator: stdout
  details: stdout content does not match
  status: error
```

```
- comparator: stderr
  details: ''
  status: success
- comparator: exit_code
  details: ''
  status: success
trace_directory: /home/user/Projects/differ/reports/coreutils_echo/trace-001/binrec
values:
  left: 70
  right: -1
```

In this example, the stdout content did not match the original's.

Development

Formatting, Linting, and CI

The CI pipeline runs multiple tools that can also be run locally:

- **Formatting**
- blue - code formatting
- isort - import sorting
- **Static Analysis**
- pyright - type checking
- flake8 - static code analysis
- **Unit Tests**
- pytest - unit testing
- coverage - code coverage
- **Documentation**
- sphinx - API documentation
- **Spell Checking**

- cspell - Code spell checker.

    - This is a NodeJS package that must be installed, outside of `pipenv`, in a Node v14 or newer environment if you want to run this locally. `bash $ npm install -g cspell`

    - The custom dictionary is located in `./cspell-words.txt` and can be updated as needed.

These tools can be run locally using `pipenv`:

```
# Run linting checks (static analysis)
$ pipenv run lint

# Format Python code
$ pipenv run format

# Run unit and integration tests
$ pipenv run tests

# Run spell checking (requires cspell)
$ pipenv run spell-check
```

```
# Run all CI checks (lint, spell check, test)
$ pipenv run ci
```