

Lab 6: Fuzzing for Input Validation Bugs

C Kaustubh — SRN: PES1UG23CS154 — Section : C

Execution Screenshots

Step 2: Initial Run (Valid Inputs)

```
[C:\Kaustubh from 0 LAPTOP-JQ8E9NE6][0 0.229s][0 RAM: 13/15GB][0 Friday at 9:09:25 AM][0 ?main =]
[C:\Kaustubh\Sem-5\SE\Lab-6]
uv run .\processor.py
==== Running sanitize_string ====
Enter a string with special characters (!,@,#,$,%): Hello@World!
Sanitized String: HelloWorld

==== Running parse_int_list ====
Enter a CSV of integers (e.g. 1,2,3,4): 1,2,3,4
Parsed Integer List: [1, 2, 3, 4]

==== Running reverse_words ====
Enter a sentence without punctuation: hello world
Reversed Words Sentence: olleh dlrow
```

Observation: The buggy code works correctly with valid inputs (Hello@World!, 1,2,3,4, hello world).

Step 4: Tests Failing on Buggy Code

```
[C:\Kaustubh from 0 LAPTOP-JQ8E9NE6][0 0.001s][0 RAM: 13/15GB][0 Friday at 9:11:03 AM][0 ?main = 0 -1]
[C:\Kaustubh\Sem-5\SE\Lab-6]
uv run pytest test_processor.py -v
===== test session starts =====
platform win32 -- python 3.12.3, pytest-8.4.2, pluggy-1.6.0 -- C:\Kaustubh\Sem-5\SE\Lab-6\venv\Scripts\python.exe
cachedir: .pytest_cache
hypothesis profile 'default'
rootdir: C:\Kaustubh\Sem-5\SE\Lab-6
plugins: hypothesis-6.142.4
collected 3 items

test_processor.py::test_sanitize_string_no_crash FAILED [ 33%]
test_processor.py::test_parse_int_list_safe FAILED [ 66%]
test_processor.py::test_reverse_words_safe FAILED [100%]

===== FAILURES =====
test_sanitize_string_no_crash
s = None
@given(st.text() | st.none())
def test_sanitize_string_no_crash(s):
    # Call sanitize_string with arbitrary text or None and assert it doesn't raise
    try:
        res = sanitize_string(s)
        #####
test_processor.py:9:

test_processor.py:32:
s = None
@given(st.text() | st.none())
def test_reverse_words_safe(s):
    # Call reverse_words with arbitrary text or None and assert it doesn't raise
    try:
        res = reverse_words(s)
        #####
test_processor.py:35:
sentence = None
def reverse_words(sentence):
    """
    Reverses each word in a sentence.
    Assumes sentence is non-empty and contains no punctuation.
    """
    words = sentence.split()
    #####
AttributeError: 'NoneType' object has no attribute 'split'
processor.py:24: AttributeError
During handling of the above exception, another exception occurred:
@given(st.text() | st.none())
def test_reverse_words_safe(s):
    #####
test_processor.py:32:

test_processor.py:32:
s = None
@given(st.text() | st.none())
def test_reverse_words_safe(s):
    # Call reverse_words with arbitrary text or None and assert it doesn't raise
    try:
        res = reverse_words(s)
    except Exception as e:
        assert False, f"reverse_words raised {e!r} for input {s!r}"
AssertionError: reverse_words raised AttributeError("'NoneType' object has no attribute 'split'") for input None
assert False
Falsifying example: test_reverse_words_safe(
Explanation:
These lines were always and only run by failing examples:
C:\Kaustubh\Sem-5\SE\Lab-6\test_processor.py:36

test_processor.py:37: AssertionError
===== short test summary info =====
FAILED test_processor.py::test_sanitize_string_no_crash - AssertionError: sanitize_string raised AttributeError("'NoneType' object has no attribute 'strip'") for...
FAILED test_processor.py::test_parse_int_list_safe - ExceptionGroup: Hypothesis found 2 distinct failures. (2 sub-exceptions)
FAILED test_processor.py::test_reverse_words_safe - AssertionError: reverse_words raised AttributeError("'NoneType' object has no attribute 'split'") for 1...
===== 3 failed in 1.83s =====
[C:\Kaustubh from 0 LAPTOP-JQ8E9NE6][0 2.689s][0 RAM: 13/15GB][0 Friday at 9:11:14 AM][0 ?main = 0 -1][0 Error, check your command]
[C:\Kaustubh\Sem-5\SE\Lab-6]
```

Bugs Found: All 3 tests failed. Hypothesis discovered that functions crash with None inputs, empty strings, and invalid data types. Errors: AttributeError, ValueError.

Step 5: Fixed Code Running

```
[C:\Kautubh from LAPTOP-JQ8E9NE6] [0.001s] [RAM: 13/15GB] [Friday at 9:15:18 AM] [main = 0 ~2]
[C:\Kautubh\Sem-5\SE\Lab-6]
uv run .\processor.py
==== Running sanitize_string ====
Enter a string with special characters (!,@,#,$,%): Hello@World!
Sanitized String: HelloWorld

==== Running parse_int_list ====
Enter a CSV of integers (e.g. 1,2,3,4): 1,2,3,4
Parsed Integer List: [1, 2, 3, 4]

==== Running reverse_words ====
Enter a sentence without punctuation: hello world
Reversed Words Sentence: olleh dlrow
```

Fix Applied: Added None checks, type validation, and exception handling to all three functions.

Step 6: All Tests Passing

```
[C:\Kautubh from LAPTOP-JQ8E9NE6] [0.0s] [RAM: 13/15GB] [Friday at 9:16:15 AM] [main = 0 ~2]
[C:\Kautubh\Sem-5\SE\Lab-6]
uv run pytest test_processor.py -v
===== test session starts =====
platform win32 -- Python 3.12.3, pytest-8.4.2, pluggy-1.6.0 -- C:\Kautubh\Sem-5\SE\Lab-6\.venv\Scripts\python.exe
cachedir: .pytest_cache
hypothesis profile 'default'
rootdir: C:\Kautubh\Sem-5\SE\Lab-6
plugins: hypothesis-6.142.4
collected 3 items

test_processor.py::test_sanitize_string_no_crash PASSED [ 33%]
test_processor.py::test_parse_int_list_safe PASSED [ 66%]
test_processor.py::test_reverse_words_safe PASSED [100%]

===== 3 passed in 0.99s =====
```

Result: All 3 tests passed after fixes. Code now handles hundreds of edge case inputs gracefully.

Reflections

1. How did Hypothesis help?

Hypothesis automatically generated thousands of test cases including edge cases (**None**, empty strings, Unicode). It found bugs in seconds that manual testing missed, minimized failing inputs for easy debugging, and forced defensive programming practices.

2. What would you use Fuzzing in CI/CD Pipelines?

Use fuzzing for: (a) Automated regression testing on every commit, (b) Security vulnerability detection (SQL injection, DoS), (c) API endpoint testing with malformed requests, (d) Pre-deployment quality gates to block buggy releases, (e) Data validation for parsers and input sanitizers, (f) Continuous security monitoring with nightly fuzz campaigns.

3. What do you observe from screenshots (Before vs After)?

Before (Tests Failing): All 3 tests failed. Hypothesis exposed critical bugs with **None** and empty string inputs causing **AttributeError** and **ValueError**. The code made unsafe assumptions and only worked for "happy path" inputs.

After (Tests Passing): All 3 tests passed. The fixed code handles edge cases gracefully with proper validation, type checking, and error handling. Hundreds of random inputs tested successfully without crashes.

Justification: The transformation proves that (1) Original code was fragile and production-unsafe, (2) Manual testing alone is insufficient as it misses edge cases, (3) Fuzz testing reveals hidden vulnerabilities automatically, (4) Defensive programming is essential for robustness. This demonstrates why fuzzing must be integrated into development workflows to build secure, reliable software.