



PES UNIVERSITY

Department of Computer Science & Engineering

Software Engineering

UE23CS341A

Assignment 6 Submission

Name of the Student	Pranav Hemanth
SRN	PES1UG23CS433
Section	G
Department	CSE
Campus	RR

FUZZING FOR INPUT VALIDATION BUGS

Department of Computer Science & Engineering
Software Engineering

UE23CS341A

Lab 5: FUZZING FOR INPUT VALIDATION BUGS

Objective:

To explore fuzz testing as a security and quality assurance technique by identifying and fixing bugs in input processing functions using Python and the Hypothesis fuzzing library.

Learning Outcomes

By the end of this lab, we will

- Understand the purpose and methodology of fuzz testing.
- Apply property-based fuzzing using Hypothesis.
- Detect and analyse bugs caused by edge-case inputs.
- Improve code robustness through defensive programming.

Folder Structure

```
[pranavhemanth@Pranavs-MacBook-Pro-M3 SE-S5 %cd Lab6  
[pranavhemanth@Pranavs-MacBook-Pro-M3 Lab6 %tree  
.  
└── PES1UG23CS433  
    ├── processor.py  
    └── test_processor.py
```

- `processor.py` - Given: The buggy code that to test and fix
- `test_processor.py` - Complete the code in this using hypothesis

Tasks:

1. Install dependencies
 - a. pip install hypothesis

Aug -Dec 2025 Assignment SUBMISSION_UE23CS341A

```
● pranavhemanth@Pranavs-MacBook-Pro-M3 SE-S5 %source .venv/bin/activate
● (.venv) pranavhemanth@Pranavs-MacBook-Pro-M3 SE-S5 %pip install hypothesis
Collecting hypothesis
  Downloading hypothesis-6.147.0-py3-none-any.whl.metadata (5.6 kB)
Collecting sortedcontainers<3.0.0,>=2.1.0 (from hypothesis)
  Using cached sortedcontainers-2.4.0-py2.py3-none-any.whl.metadata (10 kB)
  Downloading hypothesis-6.147.0-py3-none-any.whl (535 kB)
  ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━................................................................ 535.6/535.6 kB 10.7 MB/s 0:00:00
Using cached sortedcontainers-2.4.0-py2.py3-none-any.whl (29 kB)
Installing collected packages: sortedcontainers, hypothesis
Successfully installed hypothesis-6.147.0 sortedcontainers-2.4.0

[notice] A new release of pip is available: 25.2 -> 25.3
[notice] To update, run: pip install --upgrade pip
● (.venv) pranavhemanth@Pranavs-MacBook-Pro-M3 SE-S5 %cd Lab6
● (.venv) pranavhemanth@Pranavs-MacBook-Pro-M3 Lab6 %cd PES1UG23CS433
○ (.venv) pranavhemanth@Pranavs-MacBook-Pro-M3 PES1UG23CS433 %
```

b. pip install pytest

```
● (.venv) pranavhemanth@Pranavs-MacBook-Pro-M3 PES1UG23CS433 %pip install pytest
Collecting pytest
  Downloading pytest-8.4.2-py3-none-any.whl.metadata (7.7 kB)
Collecting iniconfig>=1 (from pytest)
  Downloading iniconfig-2.3.0-py3-none-any.whl.metadata (2.5 kB)
Collecting packaging>=20 (from pytest)
  Using cached packaging-25.0-py3-none-any.whl.metadata (3.3 kB)
Collecting pluggy<2,>=1.5 (from pytest)
  Downloading pluggy-1.6.0-py3-none-any.whl.metadata (4.8 kB)
Collecting pygments>=2.7.2 (from pytest)
  Using cached pygments-2.19.2-py3-none-any.whl.metadata (2.5 kB)
Downloading pytest-8.4.2-py3-none-any.whl (365 kB)
Downloading pluggy-1.6.0-py3-none-any.whl (20 kB)
Downloading iniconfig-2.3.0-py3-none-any.whl (7.5 kB)
Using cached packaging-25.0-py3-none-any.whl (66 kB)
Using cached pygments-2.19.2-py3-none-any.whl (1.2 MB)
Installing collected packages: pygments, pluggy, packaging, iniconfig, pytest
Successfully installed iniconfig-2.3.0 packaging-25.0 pluggy-1.6.0 pygments-2.19.2 pytest-8.4.2

[notice] A new release of pip is available: 25.2 -> 25.3
[notice] To update, run: pip install --upgrade pip
diamond (.venv) pranavhemanth@Pranavs-MacBook-Pro-M3 PES1UG23CS433 %
```

2. Run processor.py:

Command: python3 processor.py

Aug -Dec 2025 Assignment SUBMISSION_UE23CS341A

```
● (.venv) pranavhemanth@Pranav's-MacBook-Pro-M3 PES1UG23CS433 % python3 processor.py
===== Running sanitize_string =====
Enter a string with special characters (!,@,#,$,%): Software@123#!$%
Sanitized String: Software123

===== Running parse_int_list =====
Enter a CSV of integers (e.g. 1,2,3,4): 5,6,7,8
Parsed Integer List: [5, 6, 7, 8]

===== Running reverse_words =====
Enter a sentence without punctuation: This is SE lab
Reversed Words Sentence: siHT si ES bal
○ (.venv) pranavhemanth@Pranav's-MacBook-Pro-M3 PES1UG23CS433 %
```

3. Complete the code in test_processor.py using hypothesis:

Goal: Use Hypothesis to uncover more edge cases automatically for all three functions.

- Use `@given(st.text() | st.none())` to generate a wide range of inputs, including `None`.
 - Import and test the following functions:
 - `sanitize_string`
 - `parse_int_list`
 - `reverse_words`

4. Run the test_processor.py:

Command: pytest test_processor.py

Or

`python3 -m pytest test_processor.py` (if pytest is installed but still showing not recognised)

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS zsh - PES1UG23CS433 + [ ]  
(.venv) pranavhemanth@Pranav's-MacBook-Pro-M3 PES1UG23CS433 % python3 -m pytest test_processor.py  
===== test session starts =====  
platform darwin -- Python 3.13.7, pytest-8.4.2, pluggy-1.6.0  
rootdir: /Users/pranavhemanth/Code/Academics/SE-S5/Lab6/PES1UG23CS433  
plugins: hypothesis-6.147.0  
collected 3 items  
  
test_processor.py FFF [100%]  
===== FAILURES =====  
_____  
test_sanitize_string_no_crash  
_____  
s = None  
  
@given(st.text() | st.none())  
def test_sanitize_string_no_crash(s):  
    # To-Do: Using try and except complete this function.  
    try:  
        >     sanitize_string(s)  
  
test_processor.py:9:  
-----  
  
data = None  
  
def sanitize_string(data):  
    """  
        Removes special characters and trims the input.  
        Assumes data is a non-empty string.  
    """  
    >     data = data.strip()  
          ^~~~~~  
E     AttributeError: 'NoneType' object has no attribute 'strip'  
  
processor.py:6: AttributeError  
  
During handling of the above exception, another exception occurred:  
  
    @given(st.text() | st.none())  
    >     def test_sanitize_string_no_crash(s):  
          ^~~  
  
test_processor.py:6:
```

Aug -Dec 2025 Assignment SUBMISSION_UE23CS341A

```
s = None
@given(st.text() | st.none())
def test_sanitize_string_no_crash(s):
    # To-Do: Using try and except complete this function.
    try:
        sanitize_string(s)
    except Exception as e:
        assert False, f"sanitize_string raised an exception: {e}"
>     AssertionError: sanitize_string raised an exception: 'NoneType' object has no attribute 'strip'
>     assert False
>     Falsifying example: test_sanitize_string_no_crash()
>         s=None,
>     )
>     Explanation:
>         These lines were always and only run by failing examples:
>             /Users/pranavhemanth/Code/Academics/SE-S5/Lab6/PES1UG23CS433/test_processor.py:10
test_processor.py:11: AssertionError
                                         test_parse_int_list_safe
+ Exception Group Traceback (most recent call last):
|   File "/Users/pranavhemanth/Code/Academics/SE-S5/Lab6/PES1UG23CS433/test_processor.py", line 15, in test_parse_int_list_safe
|     def test_parse_int_list_safe(s):
|         ^
|         File "/Users/pranavhemanth/Code/Academics/SE-S5/.venv/lib/python3.13/site-packages/hypothesis/core.py", line 2110, in wrapped_test
|             raise _error_hypothesis_found
|         ExplanationGroup: Hypothesis found 2 distinct failures. (2 sub-exceptions)
+-- 1
|   Traceback (most recent call last):
|     File "/Users/pranavhemanth/Code/Academics/SE-S5/Lab6/PES1UG23CS433/test_processor.py", line 18, in test_parse_int_list_safe
|       parse_int_list(s)
|       ^
|       File "/Users/pranavhemanth/Code/Academics/SE-S5/Lab6/PES1UG23CS433/processor.py", line 17, in parse_int_list
|         return [int(p) for p in parts]
|         ^
|         ValueError: invalid literal for int() with base 10: ''
|         During handling of the above exception, another exception occurred:
|             Traceback (most recent call last):
|               File "/Users/pranavhemanth/Code/Academics/SE-S5/Lab6/PES1UG23CS433/test_processor.py", line 20, in test_parse_int_list_safe
|                 assert False, f"parse_int_list raised an exception: {e}"
|             AssertionError: parse_int_list raised an exception: invalid literal for int() with base 10: ''
|             assert False
|             Falsifying example: test_parse_int_list_safe()
|                 s='',
|             )
|             Explanation:
|                 These lines were always and only run by failing examples:
|                     /Users/pranavhemanth/Code/Academics/SE-S5/Lab6/PES1UG23CS433/test_processor.py:19
+-- 2
|   Traceback (most recent call last):
|     File "/Users/pranavhemanth/Code/Academics/SE-S5/Lab6/PES1UG23CS433/test_processor.py", line 18, in test_parse_int_list_safe
|       parse_int_list(s)
|       ^
|       File "/Users/pranavhemanth/Code/Academics/SE-S5/Lab6/PES1UG23CS433/processor.py", line 16, in parse_int_list
|         parts = csv_string.split(',')
|         ^
|         AttributeError: 'NoneType' object has no attribute 'split'
|         During handling of the above exception, another exception occurred:
|             Traceback (most recent call last):
|               File "/Users/pranavhemanth/Code/Academics/SE-S5/Lab6/PES1UG23CS433/test_processor.py", line 20, in test_parse_int_list_safe
|                 assert False, f"parse_int_list raised an exception: {e}"
|             AssertionError: parse_int_list raised an exception: 'NoneType' object has no attribute 'split'
|             assert False
|             Falsifying example: test_parse_int_list_safe()
|                 s=None,
|             )
|             Explanation:
|                 These lines were always and only run by failing examples:
|                     /Users/pranavhemanth/Code/Academics/SE-S5/Lab6/PES1UG23CS433/test_processor.py:19
```

Aug -Dec 2025 Assignment SUBMISSION_UE23CS341A

```
(.venv) pranavhemanth@Pranav's-MacBook-Pro-M3 PES1UG23CS433 % python3 -m pytest test_processor.py
test_reverse_words_safe

s = None
@given(st.text() | st.none())
def test_reverse_words_safe(s):
    # To-Do: Using try and except complete this function.
    try:
        reverse_words(s)

test_processor.py:27:
-----
sentence = None

def reverse_words(sentence):
    """
    Reverses each word in a sentence.
    Assumes sentence is non-empty and contains no punctuation.
    """
    words = sentence.split()
E   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:24:
-----
s = None

@given(st.text() | st.none())
def test_reverse_words_safe(s):
    # To-Do: Using try and except complete this function.
    try:
        reverse_words(s)
    except Exception as e:
        assert False, f"reverse_words raised an exception: {e}"
E   AssertionError: reverse_words raised an exception: 'NoneType' object has no attribute 'split'
E   assert False
E   Falsifying example: test_reverse_words_safe(
E       s=None,
E   )
E   Explanation:
E       These lines were always and only run by failing examples:
E       /Users/pranavhemanth/Code/Academics/SE-S5/Lab6/PES1UG23CS433/test_processor.py:28

test_processor.py:29: AssertionError
=====
short test summary info =====
FAILED test_processor.py::test_SANITIZE_STRING_no_crash - AssertionError: sanitize_string raised an exception: 'NoneType' object has no attribute 'strip'
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 an exception: 'NoneType' object has no attribute 'split'
=====
3 failed in 0.74s
```

5. Fix the buggy processor.py code:

Goal: Harden the functions against all unexpected or invalid inputs, especially those discovered through fuzz testing.

Run the fixed processor.py –

Command : python processor.py

```
● (.venv) pranavhemanth@Pranav's-MacBook-Pro-M3 PES1UG23CS433 % python3 processor.py
===== Running sanitize_string =====
Enter a string with special characters (!,@,#,$,%): Software@123#!$%
Sanitized String: Software123

===== Running parse_int_list =====
Enter a CSV of integers (e.g. 1,2,3,4): 5,6,7,8
Parsed Integer List: [5, 6, 7, 8]

===== Running reverse_words =====
Enter a sentence without punctuation: This is SE lab
Reversed Words Sentence: sihT si ES bal
○ (.venv) pranavhemanth@Pranav's-MacBook-Pro-M3 PES1UG23CS433 %
```

6. Re-Run the test_processor.py wrt to fixed processor and take the ss of the output:

Command: pytest test_processor.py

Or

python3 -m pytest test_processor.py (if pytest is installed but still showing not recognised)

```
(.venv) pranavhemanth@Pranav's-MacBook-Pro-M3 PES1UG23CS433 % python3 -m pytest test_processor.py
=====
platform darwin -- Python 3.13.7, pytest-8.4.2, pluggy-1.6.0
rootdir: /Users/pranavhemanth/Code/Academics/SE-55/Lab6/PES1UG23CS433
plugins: hypothesis-6.147.0
collected 3 items

test_processor.py ...

[100%]

(.venv) pranavhemanth@Pranav's-MacBook-Pro-M3 PES1UG23CS433 %
```

7. Reflection:

1. How did Hypothesis help?

Hypothesis significantly streamlined the process of discovering and fixing edge-case bugs that I would have otherwise missed through manual testing. It provided several key benefits:

1. Automated Generation of Unexpected Inputs:

Instead of manually writing test cases for special scenarios- like empty strings, unusual characters, None values, or extremely long strings, Hypothesis automatically generated them using strategies such as @given(st.text() | st.none()). This ensured that my functions were tested against a broad and diverse set of inputs.

2. Revealing Breaking Points:

When I ran pytest in Step 4, Hypothesis systematically fed these generated inputs into functions like sanitize_string, parse_int_list, and reverse_words. This process quickly uncovered “falsifying examples” specific inputs that caused the original code to fail, making it easy to pinpoint weaknesses and unhandled edge cases.

3. Guiding the Debugging Process:

The failing tests highlighted exactly what kinds of data caused errors (for example, a TypeError when the function received None). These insights directly guided my fixes in processor.py, leading me to add proper error handling through try...except blocks and conditional checks to make the functions more resilient.

4. Verifying the Fixes:

After implementing the fixes, I re-ran Hypothesis tests in Step 6. When all tests passed, it confirmed that the updated code not only handled the previously failing inputs but was also robust across a wide range of automatically generated cases.

In essence, Hypothesis acted as an intelligent and creative quality assurance partner, automating edge-case discovery, exposing hidden bugs, and verifying that my fixes truly strengthened the reliability of the code.

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

From this lab, I learned how valuable fuzzing can be when integrated into a CI/CD pipeline as an automated security and quality control step. Whenever new code is committed, the CI pipeline would automatically trigger. After the standard unit tests pass, a dedicated stage would run fuzz tests (for example, `test_processor.py`). If the fuzzer detects any input that leads to a crash or unhandled exception (like the issues identified in SS2), the pipeline would immediately flag the build as failed. This mechanism would stop the defective code from being merged or deployed to production.

The key benefit of this approach is continuous automation and early prevention. By embedding fuzzing into the CI process, developers no longer need to remember to run tests manually. The system consistently checks every code change, ensuring that existing robustness is maintained and that new input-handling issues are caught as soon as they arise.

3. What do you observe from the screenshots SS2 and SS4? Justify your answer.

In SS2, several test cases failed, revealing important issues in the code.

Test 1: `test_SANITIZE_STRING_no_crash`

This test produced an `AttributeError`: '`NoneType`' object has no attribute '`strip`'. The failure occurred because the function received `None` as input (`s=None`) and directly called `data.strip()`, which only works on strings. Since `None` does not have a `strip()` method, the function crashed.

Test 2: `test_PARSE_INT_LIST_safe`

Hypothesis identified two separate failures in this test:

1. A `ValueError`: invalid literal for `int()` with base 10, which occurred when splitting an empty string using `.split(',')`. This operation produced `[]`, and attempting to convert `"` to an integer caused the error.
2. An `AttributeError`: '`NoneType`' object has no attribute '`split`', which happened because the function attempted to call `.split(',')` on `None`, an invalid operation.

Test 3: `test_reverse_words_safe`

This test failed with an `AttributeError`: '`NoneType`' object has no attribute '`split`'. The issue arose because the test passed `None` as input, and the function directly invoked `sentence.split()`, which cannot be applied to `None`.

The clean test pass in SS4 demonstrated that I successfully used the insights from SS2 to strengthen my code. I modified `processor.py` by adding appropriate error handling, such as

Aug -Dec 2025 Assignment SUBMISSION_UE23CS341A

try...except blocks and checks for None, to address these AttributeError and ValueError cases.

In summary, SS2 highlighted the vulnerabilities discovered by Hypothesis, while SS4 confirmed that my fixes effectively resolved them.