

Software Engineering
UE23CS341A
5th Semester, Academic Year 2023

Date: 05/11/2025

Name: Roshini Ramesh	SRN: PES1UG23CS488	Section: H
----------------------	--------------------	------------

LAB 6: 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, students 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:

PES1UG23CSXXX/

processor.py	→ Given: The buggy code that you'll test and fix
test_processor.py	→ Complete the code in this using hypothesis

Steps:

Step 1: Installing dependencies:

1. `pip install hypothesis`

```
rosh@LAPTOP-10GABJ15:~/Documents/PES/Sem 5/SE/Lab_6_FuzzTesting$ pip install hypothesis
Collecting hypothesis
  Downloading hypothesis-6.145.1-py3-none-any.whl.metadata (5.6 kB)
Requirement already satisfied: sortedcontainers<3.0.0,>=2.1.0 in c:\users\roshini\appdata\local\programs\python\python313\lib\site-packages (from hypothesis) (2.4.0)
Downloading hypothesis-6.145.1-py3-none-any.whl (534 kB)
 534.8/534.8 kB 10.0 MB/s eta 0:00:00
Installing collected packages: hypothesis
Successfully installed hypothesis-6.145.1

[notice] A new release of pip is available: 25.1.1 -> 25.3
[notice] To update, run: python.exe -m pip install --upgrade pip
❖ rosh@LAPTOP-10GABJ15:~/Documents/PES/Sem 5/SE/Lab_6_FuzzTesting$
```

2. `pip install pytest`

```
rosh@LAPTOP-10GABJ15:~/Documents/PES/Sem 5/SE/Lab_6_FuzzTesting$ pip install pytest
Collecting pytest
  Downloading pytest-8.4.2-py3-none-any.whl.metadata (7.7 kB)
Requirement already satisfied: colorama>=0.4 in c:\users\roshini\appdata\local\programs\python\python313\lib\site-packages (from pytest) (0.4.6)
Collecting iniconfig>=1 (from pytest)
  Downloading iniconfig-2.3.0-py3-none-any.whl.metadata (2.5 kB)
Requirement already satisfied: packaging>=20 in c:\users\roshini\appdata\local\programs\python\python313\lib\site-packages (from pytest) (25.0)
Collecting pluggy<2,>=1.5 (from pytest)
  Downloading pluggy-1.6.0-py3-none-any.whl.metadata (4.8 kB)
Requirement already satisfied: pygments>=2.7.2 in c:\users\roshini\appdata\local\programs\python\python313\lib\site-packages (from pytest) (2.19.2)
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)
Installing collected packages: pluggy, iniconfig, pytest
Successfully installed iniconfig-2.3.0 pluggy-1.6.0 pytest-8.4.2
```

Step 2: Run the buggy [processor.py](#):

Command: `python processor.py`

Provide the screenshot of the output (SS1)

```

● rosh@LAPTOP-10GABJ15:~/Documents/PES/Sem 5/SE/Lab_6_FuzzTesting$ python processor.py
==== Running sanitize_string ====
Enter a string with special characters (!,@,#,$,%): Hello@123!#$%
Sanitized String: Hello123

==== 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: Hi I am Roshini
Reversed Words Sentence: iH I ma inihsoR
❖ rosh@LAPTOP-10GABJ15:~/Documents/PES/Sem 5/SE/Lab_6_FuzzTesting$ 

```

Step 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`

Step 4: Run the test_processor.py :

Command: `pytest test_processor.py`

Or

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

Provide the screenshot of test cases failing (SS2).

```
rosh@LAPTOP-10GABJ15:~/Documents/PES/Sem 5/SE/Lab_6_FuzzTesting$ pytest test_processor.py
===== test session starts =====
platform win32 -- Python 3.13.1, pytest-8.4.2, pluggy-1.6.0
rootdir: C:\Users\Roshini\Documents\PES\Sem 5\SE\Lab_6_FuzzTesting
plugins: anyio-4.11.0, hypothesis-6.145.1
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):
        """Test sanitize_string with various inputs including None."""
        try:
>         result = 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:

s = None

@given(st.text() | st.none())

def test_sanitize_string_no_crash(s):

"""Test sanitize_string with various inputs including None."""

try:

result = sanitize_string(s)

If it succeeds, result should be a string

assert isinstance(result, str)

except (AttributeError, TypeError) as e:

Expected for None and non-string inputs

> pytest.fail(f"Expected error for input {repr(s)}: {type(e).__name__}")

E Failed: Expected error for input None: AttributeError

E Falsifying example: test_sanitize_string_no_crash(

E s=None,

E)

E Explanation:

E These lines were always and only run by failing examples:

E C:\Users\Roshini\Documents\PES\Sem 5\SE\Lab_6_FuzzTesting\test_processor.py:12

test_processor.py:14: Failed

test_parse_int_list_safe

+ Exception Group Traceback (most recent call last):

| File "C:\Users\Roshini\Documents\PES\Sem 5\SE\Lab_6_FuzzTesting\test_processor.py", line 20, in test_parse_int_list_safe

| def test_parse_int_list_safe(s):

| ^^^

| File "C:\Users\Roshini\AppData\Local\Programs\Python\Python313\Lib\site-packages\hypothesis\core.py", line 2114, in wrapped_test

| raise the_error_hypothesis_found

| BaseExceptionGroup: Hypothesis found 2 distinct failures. (2 sub-exceptions)

+----- 1 -----

| Traceback (most recent call last):

| File "C:\Users\Roshini\Documents\PES\Sem 5\SE\Lab_6_FuzzTesting\test_processor.py", line 23, in test_parse_int_list_safe

| result = parse_int_list(s)

| File "C:\Users\Roshini\Documents\PES\Sem 5\SE\Lab_6_FuzzTesting\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 "C:\Users\Roshini\Documents\PES\Sem 5\SE\Lab_6_FuzzTesting\test_processor.py", line 29, in test_parse_int_list_safe
|     pytest.fail(f"Expected error for input {repr(s)}: {type(e).__name__}")
|     ~~~~~
|   File "C:\Users\Roshini\AppData\Local\Programs\Python\Python313\Lib\site-packages\_pytest\outcomes.py", line 177, in fail
|     raise Failed(msg=reason, pytrace=pytrace)
| Failed: Expected error for input '': ValueError
| Falsifying example: test_parse_int_list_safe(
|     s='',
| )
| Explanation:
|   These lines were always and only run by failing examples:
|     C:\Users\Roshini\Documents\PES\Sem 5\SE\Lab_6_FuzzTesting\test_processor.py:27
+----- 2 -----
| Traceback (most recent call last):
|   File "C:\Users\Roshini\Documents\PES\Sem 5\SE\Lab_6_FuzzTesting\test_processor.py", line 23, in test_parse_int_list_safe
|     result = parse_int_list(s)
|   File "C:\Users\Roshini\Documents\PES\Sem 5\SE\Lab_6_FuzzTesting\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 "C:\Users\Roshini\Documents\PES\Sem 5\SE\Lab_6_FuzzTesting\test_processor.py", line 29, in test_parse_int_list_safe
|     pytest.fail(f"Expected error for input {repr(s)}: {type(e).__name__}")
|     ~~~~~
|   File "C:\Users\Roshini\AppData\Local\Programs\Python\Python313\Lib\site-packages\_pytest\outcomes.py", line 177, in fail
|     raise Failed(msg=reason, pytrace=pytrace)
| Failed: Expected error for input None: AttributeError
| Falsifying example: test_parse_int_list_safe(
|     s=None,
| )
| Explanation:
|   These lines were always and only run by failing examples:
|     C:\Users\Roshini\Documents\PES\Sem 5\SE\Lab_6_FuzzTesting\test_processor.py:27
+-----

```

test_reverse_words_safe

s = None

```

@given(st.text() | st.none())
def test_reverse_words_safe(s):
    """Test reverse_words with various inputs including None."""
    try:
>         result = reverse_words(s)
                ^^^^^^^^^^^^^^^^^^^

```

test_processor.py:38:

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:35:

s = None

```

@given(st.text() | st.none())
def test_reverse_words_safe(s):
    """Test reverse_words with various inputs including None."""
    try:
        result = reverse_words(s)
ame__}")
E         Failed: Expected error for input None: AttributeError
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         C:\Users\Roshini\Documents\PES\Sem 5\SE\Lab_6_FuzzTesting\test_processor.py:41

test_processor.py:43: Failed
===== short test summary info =====
FAILED test_processor.py::test_sanitize_string_no_crash - Failed: Expected error for input None: AttributeError
FAILED test_processor.py::test_parse_int_list_safe - BaseExceptionGroup: Hypothesis found 2 distinct failures. (2 sub-exce...
FAILED test_processor.py::test_reverse_words_safe - Failed: Expected error for input None: AttributeError
===== 3 failed in 1.41s =====
rosh@LAPTOP-10GABJ15:~/Documents/PES/Sem 5/SE/Lab_6_FuzzTesting$

```

Step 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](#)

Provide the screenshot of the output (SS3).

```

● rosh@LAPTOP-10GABJ15:~/Documents/PES/Sem 5/SE/Lab_6_FuzzTesting$ python processor
.py
==== Running sanitize_string ====
Enter a string with special characters (!,@,#,$,%): hello@123!#$%
Sanitized String: hello123

==== 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
❖ rosh@LAPTOP-10GABJ15:~/Documents/PES/Sem 5/SE/Lab_6_FuzzTesting$

```

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

Command: pytest test_processor.py

Or

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

Provide the screenshot of the output (SS4).

```
● rosh@LAPTOP-10GABJ15:~/Documents/PES/Sem 5/SE/Lab_6_FuzzTesting$ pytest test_processor.py
===== test session starts =====
platform win32 -- Python 3.13.1, pytest-8.4.2, pluggy-1.6.0
rootdir: C:\Users\Roshini\Documents\PES\Sem 5\SE\Lab_6_FuzzTesting
plugins: anyio-4.11.0, hypothesis-6.145.1
collected 3 items

test_processor.py ... [100%]

===== 3 passed in 0.79s =====
✧ rosh@LAPTOP-10GABJ15:~/Documents/PES/Sem 5/SE/Lab_6_FuzzTesting$
```

Reflection:

1. How did Hypothesis help?

Hypothesis helped by automating the discovery of edge-case bugs that I would not have found manually. It helped me in the following ways:

- Generating Unexpected Inputs: Instead of writing individual test cases for empty strings, weird characters, None values, or very long strings, Hypothesis did this automatically. By using `@given(st.text() | st.none())`, it generated a huge variety of inputs to test my functions with.
- Finding Breaking Points: When I ran pytest in Step 4, hypothesis fed all the inputs into `sanitize_string`, `parse_int_list`, and `reverse_words`. This quickly uncovered the "falsifying examples" (inputs) that caused my original, buggy code to crash, resulting in the failing tests.
- Guiding the Fix: The failing tests showed me exactly what kind of bad data would break the code (e.g., a `TypeError` when it received `None`). This allowed me to fix `processor.py` by adding proper error handling (like `try...except` blocks or `if` checks) to make the functions more robust.
- Verifies the Solution: After fixing the code, I used Hypothesis again in Step 6. When all the tests passed (SS4), it confirmed that my new code was not only fixed for the specific inputs that failed before but was also robust against the wide range of other inputs Hypothesis generated.

Essentially, hypothesis acted as a powerful and creative quality assurance tester, finding the exact vulnerabilities in my input validation so I could patch them.

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

Based on what I learned in this lab, I'd use fuzzing in a CI/CD pipeline as an automated security and quality gate. Every time my team and I commit new code, we would make sure to use the CI pipeline. Right after the basic unit tests pass, we would have a dedicated stage to run fuzz tests (like `test_processor.py`). If the fuzzer finds any new input that causes a crash or an unhandled exception (like

the failures in SS2), the pipeline would automatically fail the build. This failure would prevent the buggy code from being merged or deployed to production.

The main advantage of this approach is automation and prevention. Instead of us having to remember to run pytest manually, the pipeline would do it for us every single time. It ensures that the robust code stays robust, catching any new input validation bugs the moment they're introduced.

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

In SS2, we observe the following failed test cases.

Test 1: test_sanitize_string_no_crash:

- Here, we got an `AttributeError: 'NoneType' object has no attribute 'strip'`. The reason was that the test passed `None` as input (`s=None`), but the function directly calls `data.strip()` which only works on strings. `None` doesn't have `.strip()`.

Test 2: test_parse_int_list_safe

- Here, we got two separate failures found by Hypothesis.
 1. Here, we got a `ValueError: invalid literal for int() with base 10`. The reason was that when we split `"` with `.split(',')`, you get `[]`, and `int("")` raises a `ValueError`.
 2. Here, we got a `AttributeError: 'NoneType' object has no attribute 'split'`. The reason was that the function calls `.split(',')` on `None`, which is invalid.

Test 3: test_reverse_words_safe

- Here, we got an `AttributeError: 'NoneType' object has no attribute 'split'`. The reason was that the test passed `None`, but the function directly calls `sentence.split()`, which fails for `None`.

The clean pass in SS4 shows that I correctly used the information from SS2 to harden my code. I went back to `processor.py` and added error handling (like `try...except` blocks or checking if `sentence` is `None` to handle those exact `AttributeError` and `ValueError` cases.

In short, SS2 shows Hypothesis finding the vulnerabilities, and SS4 shows Hypothesis confirming my solution fixed them.