INTRO TO OSS-FUZZ

BUILD, BREAK, AND HARDEN
OPEN SOURCE SOFTWARE

Mohammed Imaduddin

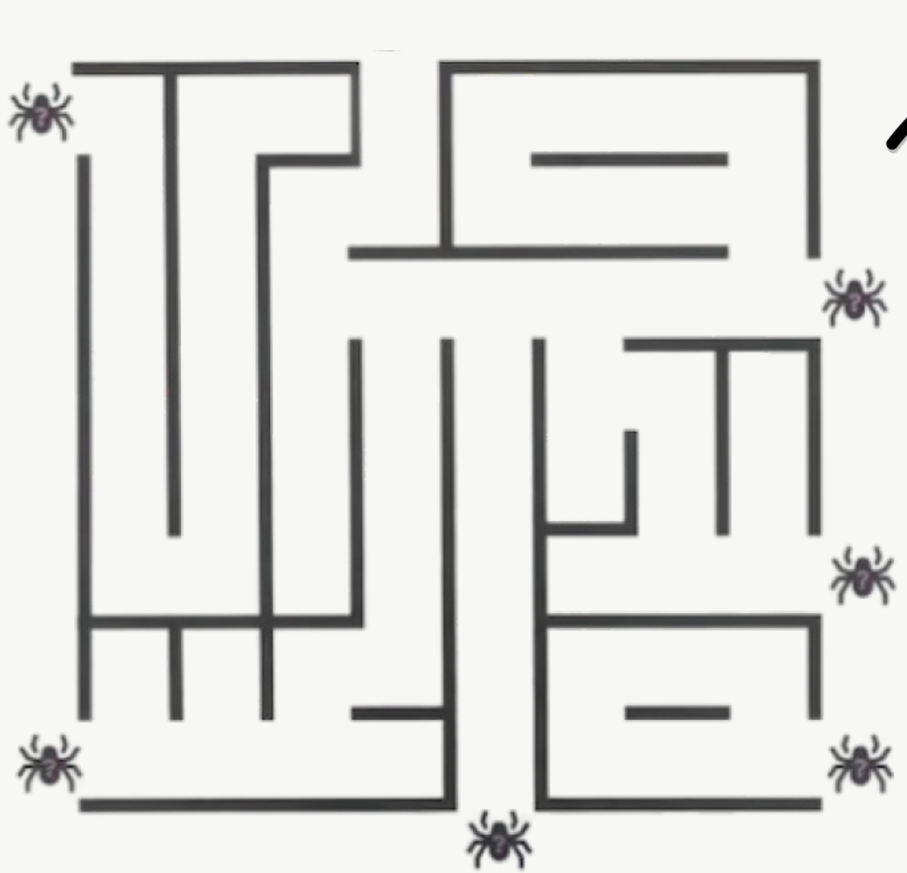
WORKSHOP AGENDA

- INTRODUCTION
- FUZZING BASICS
- TARGETING markdown2
- OSS-FUZZ WORKFLOW DEMO

WHAT IS FUZZING?

A dynamic software testing mechanism that runs (in an instrumented and optimised fashion) the software under test (through its harness) with tailored (eventually random) (formatted or unformatted) (eventually based on an initial corpus) inputs to detect unexpected behaviours such as crashes, assertions, memory leaks, race conditions and undefined behaviours

WHAT IS FUZZING?

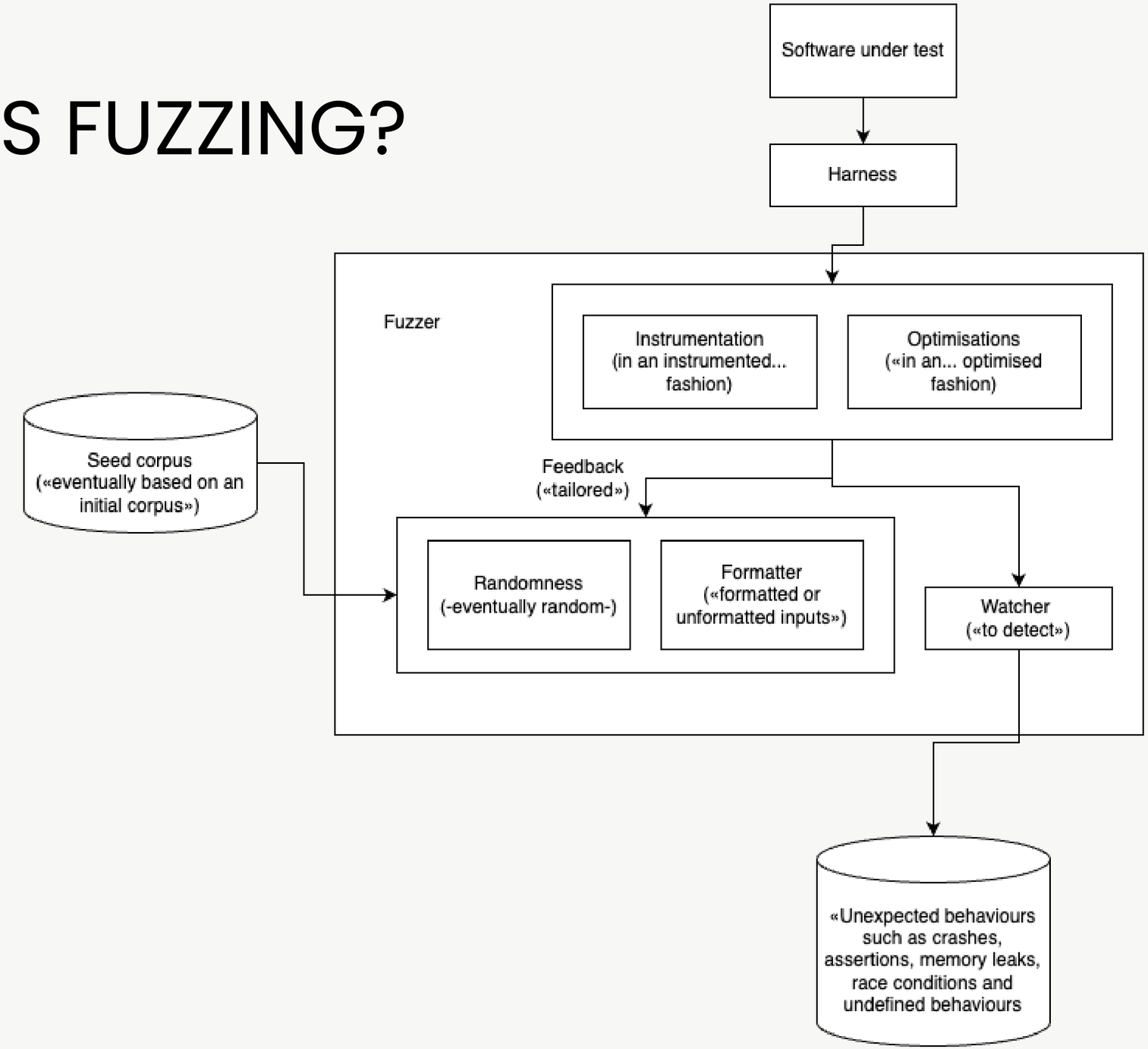


progra
m

Fuzzed Input:

- DDLDRD
- DDLDDD
- DDLDRRRD
- DDLDRRD
- DLRLRLRL

WHAT IS FUZZING?



WHAT IS FUZZING?

Fuzzing is a way of testing software by automatically feeding it lots of generated inputs (random or from examples) to see if it crashes or behaves unexpectedly.

Random vs Coverage-guided

- Random fuzzing
  - Feeds purely random inputs
  - Finds only shallow bugs
  - Inefficient, many invalid inputs
- Coverage-guided fuzzing
  - Tracks which code paths are exercised
  - Mutates inputs to reach unexplored branches
  - More efficient, finds deeper bugs

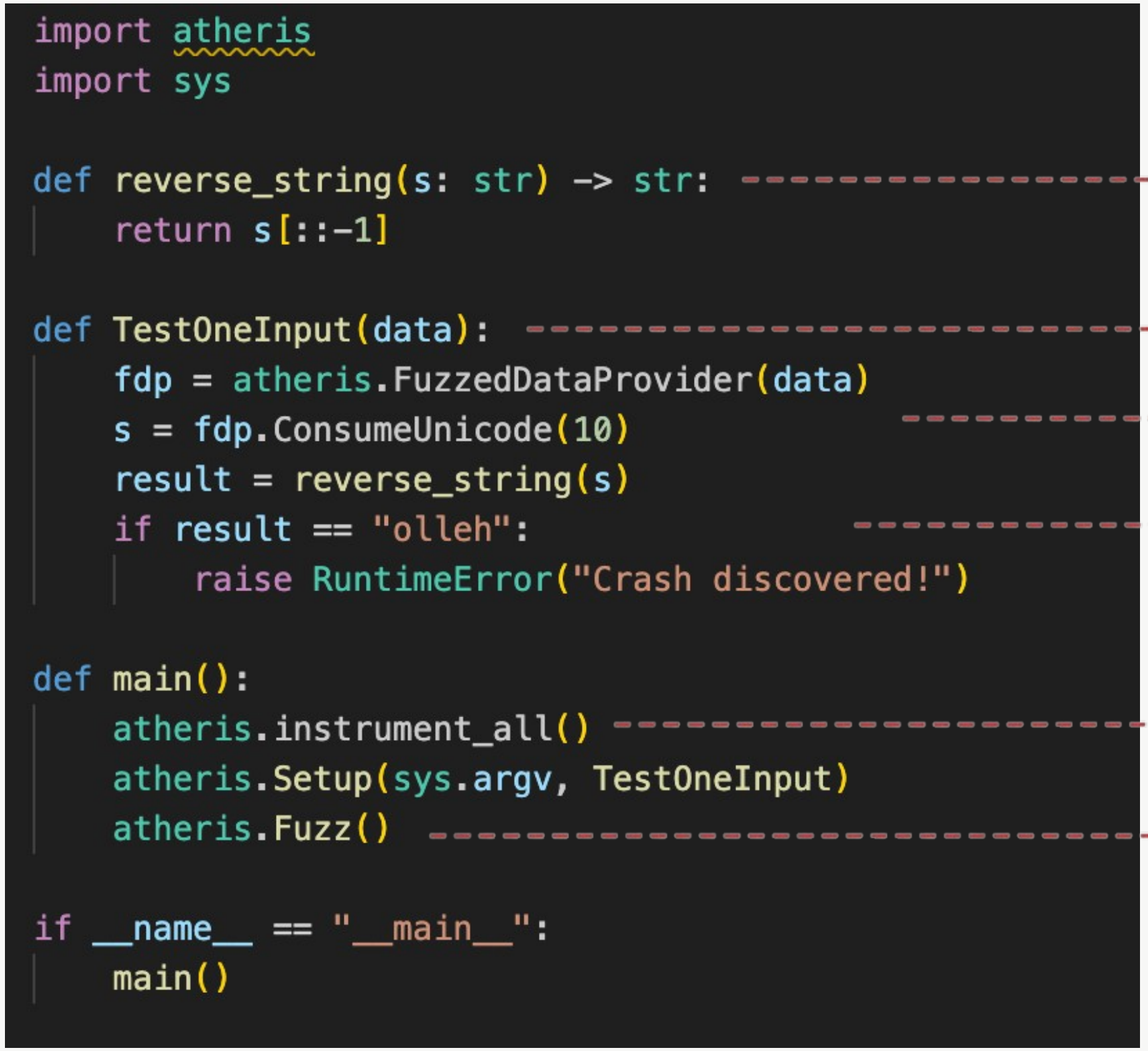
OSS-FUZZ FROM GOOGLE

- Continuous fuzzing infrastructure for critical open source projects
- Free
- OSS-Fuzz has helped identify and fix over 10,000 vulnerabilities and 36,000 bugs across 1,000 projects
- Supports:
  - Multiple fuzzers: libFuzzer, AFL++, Honggfuzz, and Centipede
  - Multiple languages: C, C++, Go, Java, Javascript, Python, Rust, and Swift
  - CPU architectures: x86_64, i386

ATHERIS: PYTHON FUZZING ENGINE

- Coverage-guided fuzzer built for Python
- Instruments Python bytecode at runtime
- Works with pure Python code and C extension modules
- Detects crashes, exceptions, and sanitizer issues
- Integrates seamlessly with OSS-Fuzz infrastructure

ATHERIS: PYTHON FUZZING ENGINE



Target function

fuzzing entrypoint

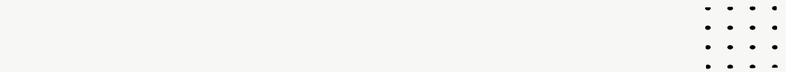
turns raw bytes into useful types

call the target function
with crash condition

instrument modules

setup the fuzzer and start fuzzing

fuzzing python-markdown2



popular Markdown parser

Processes user-controlled input (Markdown → HTML)

fuzzing python-markdown2

Even mature libraries can fail on unexpected input

Parsing libraries are common sources of crashes or security issues

target LinkProcessor

Handles links inside Markdown

Frequently executed in real usage, so bugs have high impact

target LinkProcessor

Malformed input can trigger crashes or security issues

Deals with brackets, regex, and nested structures, which are error-prone

Before we move ahead

Do you have the setup?

If not, install Python 3 + Docker, and run:

```
$ git clone https://github.com/google/oss-fuzz
$ git clone https://github.com/mdimado/oosc
```

Initializing the project

```
host $ cd oss-fuzz
host $ export PROJECT_NAME=python-markdown2
host $ export LANGUAGE=python
host $ python infra/helper.py generate \
$PROJECT_NAME \
--language=$LANGUAGE
host $ cd projects/$PROJECT_NAME
host $ ls
```

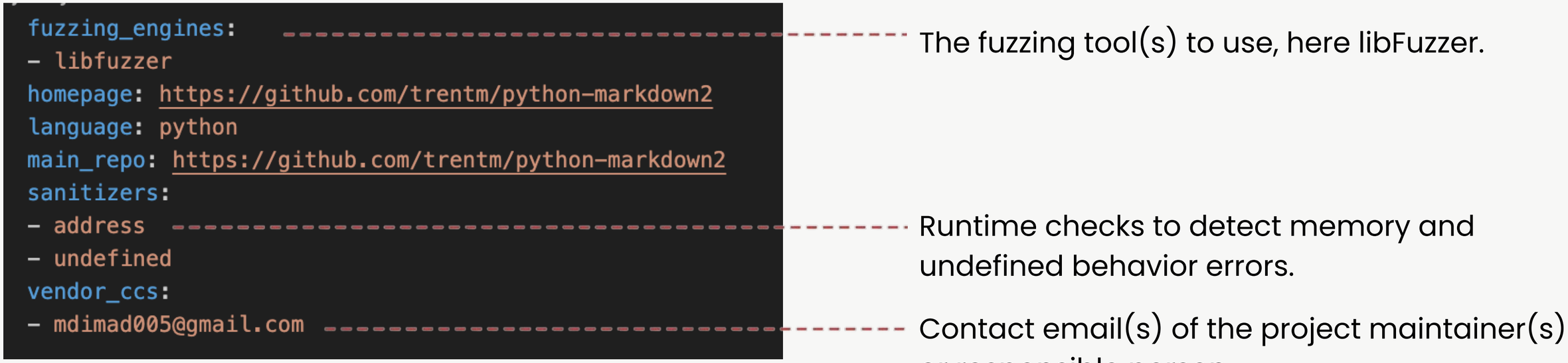
Registers your project with OSS-Fuzz

Filling up project.yaml

Provides metadata:
project name, language,
fuzzer executable,
sanitizer settings

```
host $ cd ..
host $ mv oosc/project.yaml \
oss-fuzz/projects/python-markdown2/
```

project.yaml



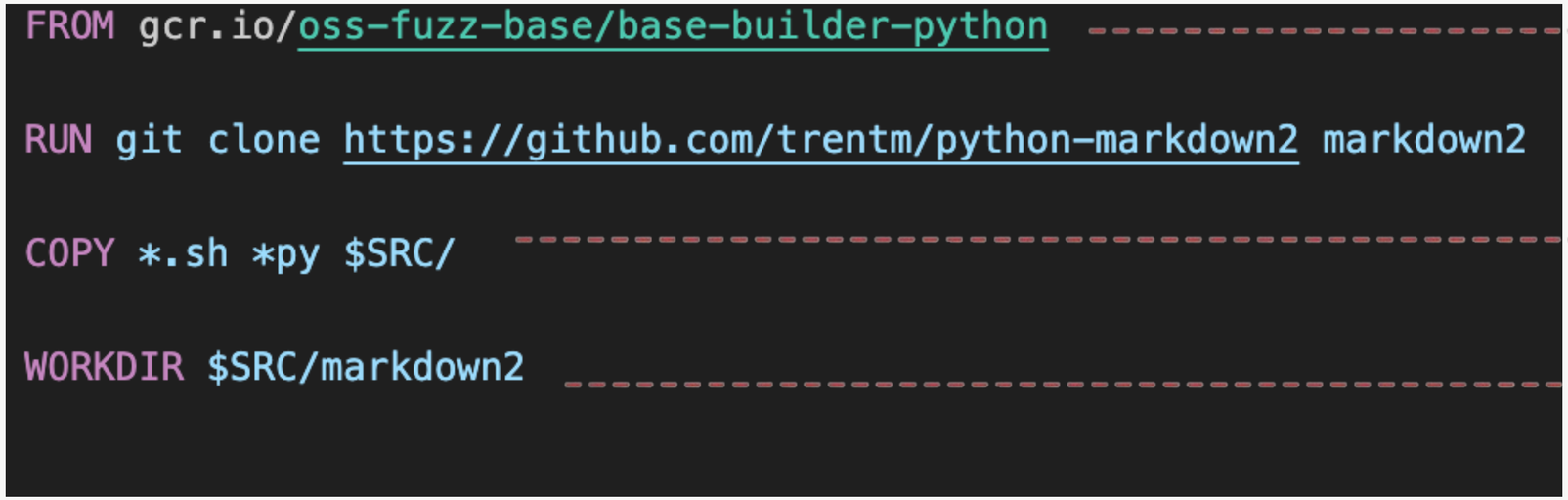
Filling up Dockerfile

Creates a controlled environment for building and running the fuzzer

Ensures all dependencies (Python, libraries, tools) are present

```
host $ cd ..
host $ mv oosc/Dockerfile \
oss-fuzz/projects/python-markdown2/
```

Dockerfile



OSS-Fuzz⍰s Python base Docker image

Copy all shell scripts and Python files

Set the working directory inside the container to the cloned markdown2 folder

Filling up build.sh

Script to compile and prepare your project for fuzzing

Can include installation of dependencies, compilation of C extensions, etc.

```
host $ cd ..
host $ mv oosc/build.sh \
oss-fuzz/projects/python-markdown2/
```

build.sh

```
pip3 install .

# Build fuzzers in $OUT.
for fuzzer in $(find $SRC -name 'fuzz_*.py'); do
  compile_python_fuzzer $fuzzer
done
```

Compile each fuzzing script into a binary that OSS-Fuzz can run.

Writing the fuzz harness

```
host $ cd ..
host $ mv oosc/fuzz_linkProcessor.py \
oss-fuzz/projects/python-markdown2/
```

fuzz_linkProcessor.py

```python
import sys
import atheris
import markdown2
from markdown2 import Markdown, LinkProcessor


def TestOneInput(data: bytes):
    fdp = atheris.FuzzedDataProvider(data)
    text = fdp.ConsumeUnicodeNoSurrogates(10000)

    md = Markdown(safe_mode=True)
    link_processor = LinkProcessor(md, None)

    # If test() matches, run it
    if link_processor.test(text):
        link_processor.run(text)

def main():
    atheris.instrument_all()
    atheris.Setup(sys.argv, TestOneInput, enable_python_coverage=True)
    atheris.Fuzz()


if __name__ == "__main__":
    main()
```

Target function

fuzzing entrypoint

turns raw bytes into useful types

create Markdown and LinkProcessor instances

Runs the link processor on the fuzzed input

instrument all modules

setup the fuzzer and start fuzzing

Building the fuzzers

```
host $ cd oss-fuzz
host $ python infra/helper.py \
build_fuzzers \
$PROJECT_NAME
```

Running the fuzzer

```
host $ cd oss-fuzz
host $ python infra/helper.py \
run_fuzzer \
$PROJECT_NAME \
fuzz_linkProcessor
```

The Crash

```
 === Uncaught Python exception: ===
AttributeError: 'Markdown' object has no attribute 'extra_classes'
Traceback (most recent call last):
  File "fuzz_linkProcessor.py", line 16, in TestOneInput
  File "markdown2.py", line 2723, in run
  File "markdown2.py", line 258, in inner
AttributeError: 'Markdown' object has no attribute 'extra_classes'

Exception ignored in: <function _removeHandlerRef at 0x7ffffc0d8e00>
Traceback (most recent call last):
  File "logging/__init__.py", line -1, in _removeHandlerRef
TypeError: 'NoneType' object is not callable
==16== ERROR: libFuzzer: fuzz target exited
    #0 0x7ffffedc005a in __sanitizer_print_stack_trace /root/llvm-project/compiler-rt/lib/asan/asan_stack.cpp:87:3
    #1 0x7ffffecc35f9 in fuzzer::PrintStackTrace() /root/llvm-project/compiler-rt/lib/fuzzer/FuzzerUtil.cpp:210:38
    #2 0x7ffffeca6946 in fuzzer::Fuzzer::ExitCallback() (.part.0) /root/llvm-project/compiler-rt/lib/fuzzer/FuzzerLoop.cpp:250:18
    #3 0x7ffffeca6a18 in fuzzer::Fuzzer::ExitCallback() /root/llvm-project/compiler-rt/lib/fuzzer/FuzzerLoop.cpp:210:1
    #4 0x7ffffeca6a18 in fuzzer::Fuzzer::StaticExitCallback() /root/llvm-project/compiler-rt/lib/fuzzer/FuzzerLoop.cpp:209:18
    #5 0x7ffffea598a6  (/lib/x86_64-linux-gnu/libc.so.6+0x468a6) (BuildId: 0323ab4806bee6f846d9ad4bccfc29afdca49a58)
    #6 0x7ffffea59a5f in exit (/lib/x86_64-linux-gnu/libc.so.6+0x46a5f) (BuildId: 0323ab4806bee6f846d9ad4bccfc29afdca49a58)
    #7 0x7ffffc7c9f28 in Py_Exit /tmp/Python-3.11.13/Python/pylifecycle.c:2944:5
    #8 0x7ffffc7cd0d9 in handle_system_exit /tmp/Python-3.11.13/Python/pythonrun.c:771:9
    #9 0x7ffffc7cc975 in _PyErr_PrintEx /tmp/Python-3.11.13/Python/pythonrun.c:781:5
    #10 0x40360f  (/out/fuzz_linkProcessor.pkg+0x40360f) (BuildId: 04804d3c31218f938502cbed5cdd1af09d59a8f0)
    #11 0x403ef4  (/out/fuzz_linkProcessor.pkg+0x403ef4) (BuildId: 04804d3c31218f938502cbed5cdd1af09d59a8f0)
    #12 0x7ffffea37082 in __libc_start_main (/lib/x86_64-linux-gnu/libc.so.6+0x24082) (BuildId: 0323ab4806bee6f846d9ad4bccfc29afdc
a49a58)
    #13 0x40250e  (/out/fuzz_linkProcessor.pkg+0x40250e) (BuildId: 04804d3c31218f938502cbed5cdd1af09d59a8f0)

DEDUP_TOKEN: __sanitizer_print_stack_trace--fuzzer::PrintStackTrace()--fuzzer::Fuzzer::ExitCallback() (.part.0)
SUMMARY: libFuzzer: fuzz target exited
MS: 4 ChangeBit-EraseBytes-InsertByte-CopyPart-; base unit: 6d84a1c97e6b07ab8fb42e15cb251c02474ec78c
0x5d,0x5b,0xa,0x5d,0x5b,
][\012][
artifact_prefix='./'; Test unit written to ./crash-79e0c306e74636c00449082cc68fc458b53edb84
```

What caused the Crash?

```
host $ cd build/out/python-markdown2
host $ ls
host $ less ./crash-
```

Why it matters?

- Even well-tested libraries can have hidden assumptions.
- Fuzzing uncovers these edge cases automatically.
- This crash is minor (AttributeError), but it shows the kind of unexpected behavior fuzzers can reveal.

Having a PR accepted inside the OSS-Fuzz repo

- Describe Project purpose and importance.
- List major users.
  - Highlight critical role of the project.
- Link maintainer's approval and fuzz target PR.

- Tip: Addressing these points helps our panel decide on accepting your integration.

Resources & Next Steps

- Atheris GitHub: https://github.com/google/atheris – Python fuzzing library
- OSS-Fuzz Documentation: https://google.github.io/oss-fuzz/ – Guide to continuous fuzzing
- Demo Repository: https://github.com/mdimado/oosc – Contains example harness, Dockerfile, and build scripts

- Start fuzzing your own Python projects using Atheris
- Explore fuzzing Python projects with C extensions
- Investigate crashes, reproduce them, and submit fixes or patches

Thank You

Any questions?

@mdimado