/atheris

Source: `~/.claude/skills/tob-testing-handbook-skills/skills/atheris/SKILL.md`

---

name: atheris type: fuzzer description: > Atheris is a coverage-guided Python fuzzer based on libFuzzer. Use for fuzzing pure Python code and Python C extensions.

Atheris

Atheris is a coverage-guided Python fuzzer built on libFuzzer. It enables fuzzing of both pure Python code and Python C extensions with integrated AddressSanitizer support for detecting memory corruption issues.

When to Use

Fuzzer	Best For	Complexity
Atheris	Python code and C extensions	Low-Medium
Hypothesis	Property-based testing	Low
python-afl	AFL-style fuzzing	Medium

Choose Atheris when:

- Fuzzing pure Python code with coverage guidance
- Testing Python C extensions for memory corruption
- Integration with libFuzzer ecosystem is desired
- AddressSanitizer support is needed

Quick Start

```
import sys
import atheris

@atheris.instrument_func
def test_one_input(data: bytes):
    if len(data) == 4:
        if data[0] == 0x46:  # "F"
            if data[1] == 0x55:  # "U"
                if data[2] == 0x5A:  # "Z"
                    if data[3] == 0x5A:  # "Z"
                        raise RuntimeError("You caught me")

def main():
    atheris.Setup(sys.argv, test_one_input)
    atheris.Fuzz()

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

Run:

```
python fuzz.py
```

Installation

Atheris supports 32-bit and 64-bit Linux, and macOS. We recommend fuzzing on Linux because it's simpler to manage and often faster.

Prerequisites

- Python 3.7 or later
- Recent version of clang (preferably latest release)
- For Docker users: Docker Desktop

Linux/macOS

```
uv pip install atheris
```

Docker Environment (Recommended)

For a fully operational Linux environment with all dependencies configured:

```
# https://hub.docker.com/_/python
ARG PYTHON_VERSION=3.11

FROM python:$PYTHON_VERSION-slim-bookworm

RUN python --version

RUN apt update && apt install -y \
    ca-certificates \
    wget \
    && rm -rf /var/lib/apt/lists/*

# LLVM builds version 15-19 for Debian 12 (Bookworm)
# https://apt.llvm.org/bookworm/dists/
ARG LLVM_VERSION=19

RUN echo "deb http://apt.llvm.org/bookworm/ llvm-toolchain-bookworm-$LLVM_VERSION main" >
/etc/apt/sources.list.d/llvm.list
RUN echo "deb-src http://apt.llvm.org/bookworm/ llvm-toolchain-bookworm-$LLVM_VERSION main" >>
/etc/apt/sources.list.d/llvm.list
RUN wget -qO- https://apt.llvm.org/llvm-snapshot.gpg.key >
/etc/apt/trusted.gpg.d/apt.llvm.org.asc

RUN apt update && apt install -y \
    build-essential \
    clang-$LLVM_VERSION \
    && rm -rf /var/lib/apt/lists/*
```

```
ENV APP_DIR "/app"
RUN mkdir $APP_DIR
WORKDIR $APP_DIR

ENV VIRTUAL_ENV "/opt/venv"
RUN python -m venv $VIRTUAL_ENV
ENV PATH "$VIRTUAL_ENV/bin:$PATH"

# https://github.com/google/atheris/blob/master/native_extension_fuzzing.md#step-1-compiling-your-extension
ENV CC="clang-$LLVM_VERSION"
ENV CFLAGS "-fsanitize=address,fuzzer-no-link"
ENV CXX="clang++-$LLVM_VERSION"
ENV CXXFLAGS "-fsanitize=address,fuzzer-no-link"
ENV LDSHARED="clang-$LLVM_VERSION -shared"
ENV LDSHAREDXX="clang++-$LLVM_VERSION -shared"
ENV ASAN_SYMBOLIZER_PATH="/usr/bin/llvm-symbolizer-$LLVM_VERSION"

# Allow Atheris to find fuzzer sanitizer shared libs
# https://github.com/google/atheris#building-from-source
RUN LIBFUZZER_LIB=$($CC -print-file-name=libclang_rt.fuzzer_no_main-$(uname -m).a) \
    python -m pip install --no-binary atheris atheris

# https://github.com/google/atheris/blob/master/native_extension_fuzzing.md#option-a-sanitizerlibfuzzer-preloads
ENV LD_PRELOAD "$VIRTUAL_ENV/lib/python3.11/site-packages/asan_with_fuzzer.so"

# 1. Skip memory allocation failures for now, they are common, and low impact (DoS)
# 2. https://github.com/google/atheris/blob/master/native_extension_fuzzing.md#leak-detection
ENV ASAN_OPTIONS "allocator_may_return_null=1,detect_leaks=0"

CMD ["/bin/bash"]
```

Build and run:

```
docker build -t atheris .
docker run -it atheris
```

Verification

```
python -c "import atheris; print(atheris.__version__)"
```

Writing a Harness

Harness Structure for Pure Python

```
import sys
import atheris

@atheris.instrument_func
def test_one_input(data: bytes):
    """
    Fuzzing entry point. Called with random byte sequences.

    Args:
        data: Random bytes generated by the fuzzer
    """
    # Add input validation if needed
    if len(data) < 1:
        return

    # Call your target function
    try:
        your_target_function(data)
    except ValueError:
        # Expected exceptions should be caught
        pass
    # Let unexpected exceptions crash (that's what we're looking for!)

def main():
    atheris.Setup(sys.argv, test_one_input)
    atheris.Fuzz()

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

Harness Rules

Do	Don't
Use `@atheris.instrument_func` for coverage	Forget to instrument target code
Catch expected exceptions	Catch all exceptions indiscriminately
Use `atheris.instrument_imports()` for libraries	Import modules after `atheris.Setup()`
Keep harness deterministic	Use randomness or time-based behavior

> **See Also:** For detailed harness writing techniques, patterns for handling complex inputs, and advanced strategies, see the **fuzz-harness-writing** technique skill.

Fuzzing Pure Python Code

For fuzzing broader parts of an application or library, use instrumentation functions:

```
import atheris
with atheris.instrument_imports():
    import your_module
    from another_module import target_function

def test_one_input(data: bytes):
    target_function(data)

atheris.Setup(sys.argv, test_one_input)
atheris.Fuzz()
```

Instrumentation Options:

- `atheris.instrument_func` - Decorator for single function instrumentation
- `atheris.instrument_imports()` - Context manager for instrumenting all imported modules
- `atheris.instrument_all()` - Instrument all Python code system-wide

Fuzzing Python C Extensions

Python C extensions require compilation with specific flags for instrumentation and sanitizer support.

Environment Configuration

If using the provided Dockerfile, these are already configured. For local setup:

```
export CC="clang"
export CFLAGS="-fsanitize=address,fuzzer-no-link"
export CXX="clang++"
export CXXFLAGS="-fsanitize=address,fuzzer-no-link"
export LDSHARED="clang -shared"
```

Example: Fuzzing cbor2

Install the extension from source:

```
CBOR2_BUILD_C_EXTENSION=1 python -m pip install --no-binary cbor2 cbor2==5.6.4
```

The `--no-binary` flag ensures the C extension is compiled locally with instrumentation.

Create `cbor2-fuzz.py`:

```
import sys
import atheris

# _cbor2 ensures the C library is imported
from _cbor2 import loads

def test_one_input(data: bytes):
    try:
        loads(data)
    except Exception:
        # We're searching for memory corruption, not Python exceptions
        pass

def main():
    atheris.Setup(sys.argv, test_one_input)
    atheris.Fuzz()

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

Run:

```
python cbor2-fuzz.py
```

> **Important:** When running locally (not in Docker), you must set `LD_PRELOAD` manually.

Corpus Management

Creating Initial Corpus

```
mkdir corpus
# Add seed inputs
echo "test data" > corpus/seed1
echo '{"key": "value"}' > corpus/seed2
```

Run with corpus:

```
python fuzz.py corpus/
```

Corpus Minimization

Atheris inherits corpus minimization from libFuzzer:

```
python fuzz.py -merge=1 new_corpus/ old_corpus/
```

> **See Also:** For corpus creation strategies, dictionaries, and seed selection, see the **fuzzing-corpus** technique skill.

Running Campaigns

Basic Run

```
python fuzz.py
```

With Corpus Directory

```
python fuzz.py corpus/
```

Common Options

```
# Run for 10 minutes
python fuzz.py -max_total_time=600

# Limit input size
python fuzz.py -max_len=1024

# Run with multiple workers
python fuzz.py -workers=4 -jobs=4
```

Interpreting Output

Output	Meaning
`NEW    cov: X`	Found new coverage, corpus expanded
`pulse  cov: X`	Periodic status update
`exec/s: X`	Executions per second (throughput)
`corp: X/Yb`	Corpus size: X inputs, Y bytes total
`ERROR: libFuzzer`	Crash detected

Sanitizer Integration

AddressSanitizer (ASan)

AddressSanitizer is automatically integrated when using the provided Docker environment or when compiling with appropriate flags.

For local setup:

```
export CFLAGS="-fsanitize=address,fuzzer-no-link"
export CXXFLAGS="-fsanitize=address,fuzzer-no-link"
```

Configure ASan behavior:

```
export ASAN_OPTIONS="allocator_may_return_null=1,detect_leaks=0"
```

LD_PRELOAD Configuration

For native extension fuzzing:

```
export LD_PRELOAD="$(python -c 'import atheris; import os;
print(os.path.join(os.path.dirname(atheris.__file__), "asan_with_fuzzer.so"))')"
```

> **See Also:** For detailed sanitizer configuration, common issues, and advanced flags, see the **address-sanitizer** and **undefined-behavior-sanitizer** technique skills.

Common Sanitizer Issues

Issue	Solution
`LD_PRELOAD` not set	Export `LD_PRELOAD` to point to `asan_with_fuzzer.so`
Memory allocation failures	Set `ASAN_OPTIONS=allocator_may_return_null=1`
Leak detection noise	Set `ASAN_OPTIONS=detect_leaks=0`
Missing symbolizer	Set `ASAN_SYMBOLIZER_PATH` to `llvm-symbolizer`

Advanced Usage

Tips and Tricks

Tip	Why It Helps
Use `atheris.instrument_imports()` early	Ensures all imports are instrumented for coverage
Start with small `max_len`	Faster initial fuzzing, gradually increase
Use dictionaries for structured formats	Helps fuzzer understand format tokens

Tip	Why It Helps
Run multiple parallel instances	Better coverage exploration

Custom Instrumentation

Fine-tune what gets instrumented:

```
import atheris

# Instrument only specific modules
with atheris.instrument_imports():
    import target_module
# Don't instrument test harness code

def test_one_input(data: bytes):
    target_module.parse(data)
```

Performance Tuning

Setting	Impact
`-max_len=N`	Smaller values = faster execution
`-workers=N -jobs=N`	Parallel fuzzing for faster coverage
`ASAN_OPTIONS=fast_unwind_on_malloc=0`	Better stack traces, slower execution

UndefinedBehaviorSanitizer (UBSan)

Add UBSan to catch additional bugs:

```
export CFLAGS="-fsanitize=address,undefined,fuzzer-no-link"
export CXXFLAGS="-fsanitize=address,undefined,fuzzer-no-link"
```

Note: Modify flags in Dockerfile if using containerized setup.

Real-World Examples

Example: Pure Python Parser

```
import sys
import atheris
import json

@atheris.instrument_func
def test_one_input(data: bytes):
    try:
        # Fuzz Python's JSON parser
        json.loads(data.decode('utf-8', errors='ignore'))
    except (ValueError, UnicodeDecodeError):
        pass

def main():
    atheris.Setup(sys.argv, test_one_input)
    atheris.Fuzz()

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

Example: HTTP Request Parsing

```
import sys
import atheris

with atheris.instrument_imports():
    from urllib3 import HTTPResponse
    from io import BytesIO

def test_one_input(data: bytes):
    try:
        # Fuzz HTTP response parsing
        fake_response = HTTPResponse(
            body=BytesIO(data),
            headers={},
            preload_content=False
        )
        fake_response.read()
    except Exception:
```

```
        pass

def main():
    atheris.Setup(sys.argv, test_one_input)
    atheris.Fuzz()

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

Troubleshooting

Problem	Cause	Solution
No coverage increase	Poor seed corpus or target not instrumented	Add better seeds, verify `instrument_imports()`
Slow execution	ASan overhead or large inputs	Reduce `max_len`, use `ASAN_OPTIONS=fast_unwind_on_malloc=1`
Import errors	Modules imported before instrumentation	Move imports inside `instrument_imports()` context
Segfault without ASan output	Missing `LD_PRELOAD`	Set `LD_PRELOAD` to `asan_with_fuzzer.so` path
Build failures	Wrong compiler or missing flags	Verify `CC`, `CFLAGS`, and clang version

Related Skills

Technique Skills

Skill	Use Case
fuzz-harness-writing	Detailed guidance on writing effective harnesses
address-sanitizer	Memory error detection during fuzzing
undefined-behavior-sanitizer	Catching undefined behavior in C extensions
coverage-analysis	Measuring and improving code coverage
fuzzing-corpus	Building and managing seed corpora

Related Fuzzers

Skill	When to Consider
hypothesis	Property-based testing with type-aware generation
python-afl	AFL-style fuzzing for Python when Atheris isn't available

Resources

Key External Resources

Atheris GitHub Repository Official repository with installation instructions, examples, and documentation for fuzzing both pure Python and native extensions.

Native Extension Fuzzing Guide Comprehensive guide covering compilation flags, LD_PRELOAD setup, sanitizer configuration, and troubleshooting for Python C extensions.

Continuously Fuzzing Python C Extensions Trail of Bits blog post covering CI/CD integration, ClusterFuzzLite setup, and real-world examples of fuzzing Python C extensions in continuous integration pipelines.

ClusterFuzzLite Python Integration Guide for integrating Atheris fuzzing into CI/CD pipelines using ClusterFuzzLite for automated continuous fuzzing.

Video Resources

Videos and tutorials are available in the main Atheris documentation and libFuzzer resources.

Revision #5
Created 2026-02-18 08:40:10 UTC by John
Updated 2026-06-21 20:01:20 UTC by John