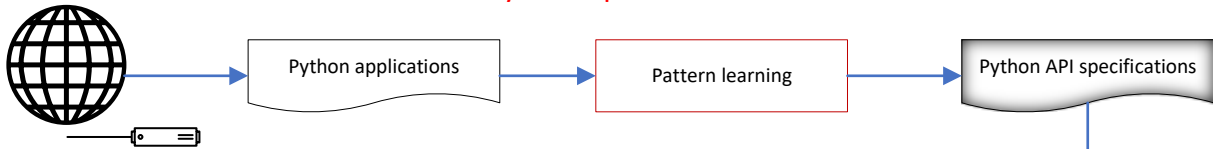


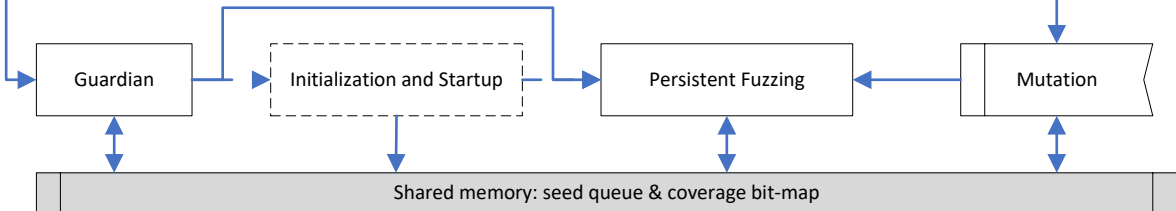
Overview

Learn Python API specifications



Seeds (python applications)

Persistent breakpoint-resume Fuzzing



Challenge 1: How to generate python applications with correct syntax and meaningful semantic?

-> **pattern learning for python API specifications**

Challenge 2: Efficiency

2.1 Why not AFL++

- > TOO low efficiency with large coverage bit-map (over 1500 K)
- > Only support non-persistent mode for Python, which means it will cost much time during startup phase (import libraries)

2.2 Why not Atheris

- > No global status for recovery after exiting triggered by error happens
- > Persistent mode for python applications, not support for python interpreter
- > Not support path coverage

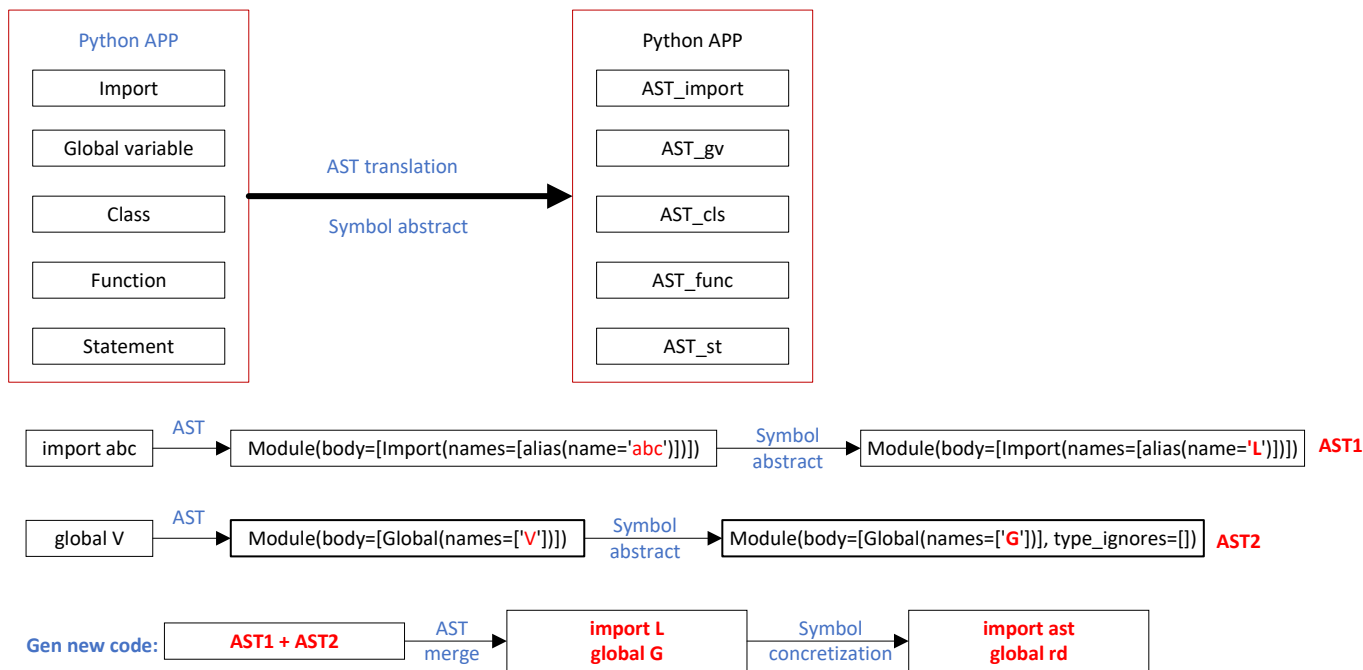
-> **Persistent fuzzing with breakpoint resume & path coverage & optimization of bit-map summary algorithm**

Python APP generation

1

General python APP generation

1. Learn the grammar from Python applications
 - translate APP into ASTs
 - for each type of element, **learn** the patterns and normalize the format through abstract symbol
2. Gen new APP through operation (ADD/DEL/MERGE/RECURSIVE...) of ASTs



2

Python APP generation focusing on usage of runtime libraries

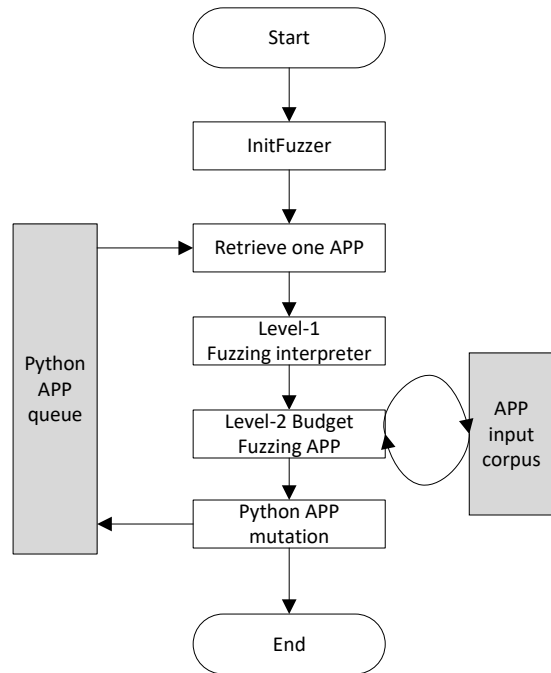
The usage of API is in form of call statement, hence we only need to learn all the possible call statements of these APIs and store them as ASTs

3

Possible dependencies of runtime library APIs

For possible dependencies among APIs, we can learn from the real world programs through static slicing

Two-level Fuzzing



Level-1 Fuzzing: Targeting interpreter Core. We use infinite budget at level-1.

Level-2 Fuzzing: Targeting runtime libraries. We use finite budget at level-2 until no favored path/block/feature found.