Mutation-Based Fuzzing

IIC3745 - Testing

Maximiliano Narea Carvajal

Recordatorio Clase Pasada

Fuzzing Básico

¿Qué es un Fuzzer?

Fuzzing Básico

```
from Fuzzer import fuzzer
sample = fuzzer()
sample

'!7#%"*#0=)$;%6*;>638:*>80"=</>(/*:-(2<4 !:5*6856&?""11<7+%
<%7,4.8,*+&,,$,."'
```

¿Y de qué sirve en este contexto?

Fuzzing Básico

```
1 with Coverage() as cov_fuzz:
2     try:
3         cgi_decode(sample)
4     except:
5         pass
6 cov_max.coverage() - cov_fuzz.coverage()
7
8 {('cgi_decode', 20), ('cgi_decode', 25),
9  ('cgi_decode', 26), ('cgi_decode', 32)}
```

¿Cómo lo usamos?

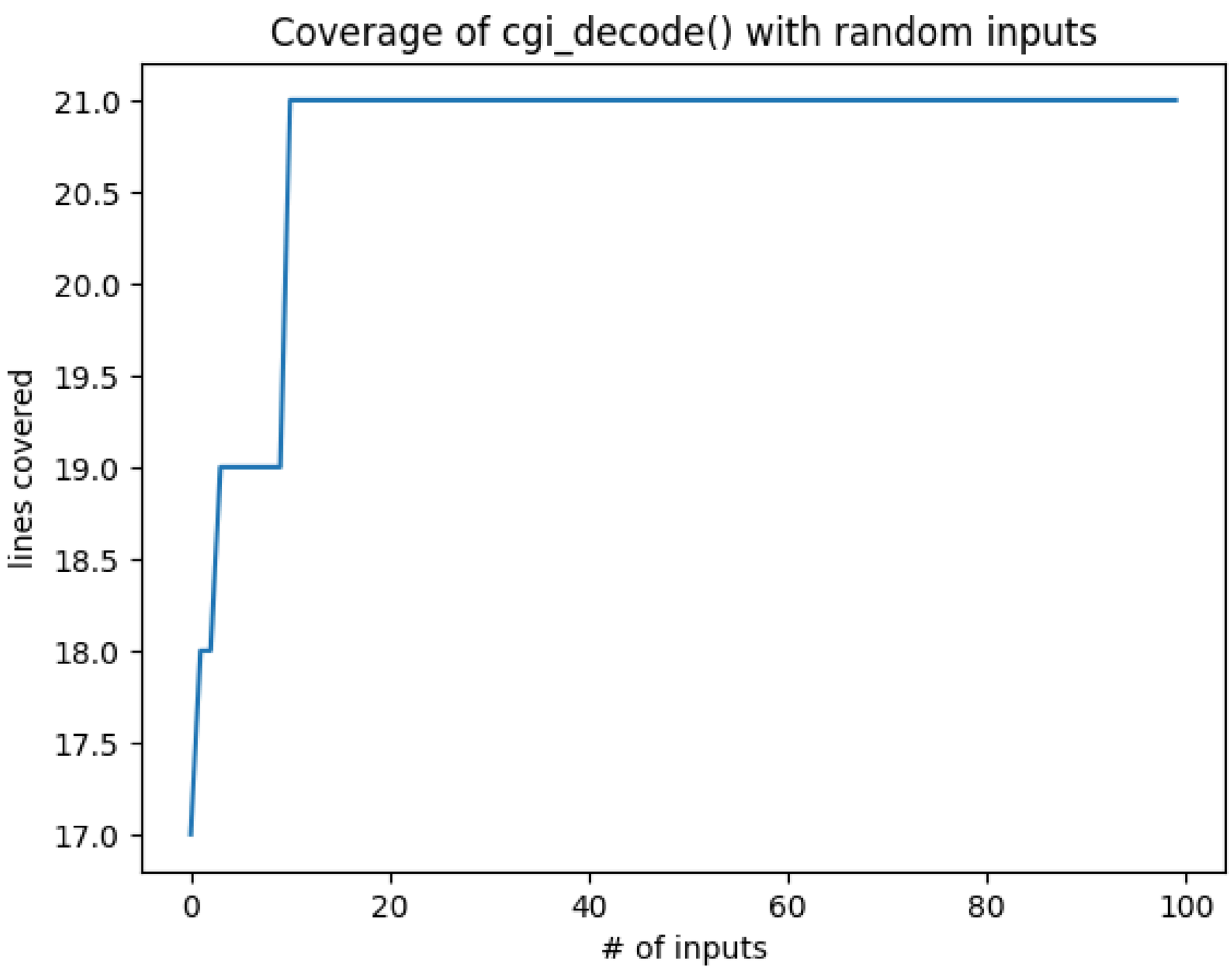
Fuzzing Básico

```
trials = 100
def population_coverage(population: List[str], function:
Callable):
    cumulative_coverage: List[int] = []
    all_coverage: Set[Location] = set()

    for s in population:
        with Coverage() as cov:
            try:
                function(s)
            except:
                pass
        all_coverage |= cov.coverage()
        cumulative_coverage.append(len(all_coverage))

    return all_coverage, cumulative_coverage

```

¿Y de qué sirve en este contexto?

Fuzzing Básico

```
1 all_coverage, cumulative_coverage = population_coverage(hundred_inputs(), cgi_decode)
2 plt.plot(cumulative_coverage)
3 plt.title('Coverage of cgi_decode() with random inputs')
4 plt.xlabel('# of inputs')
5 plt.ylabel('lines covered')
```


Coverage of cgi_decode() with random inputs
lines covered
# of inputs
21.0
20.5
20.0
19.5
19.0
18.5
18.0
17.5
17.0
0
20
40
60
80
100

Mutation-Based Fuzzing

¿Qué vamos a testear hoy?

Mutation-Based Fuzzing

```
def http_program(url: str) -> bool:
    supported_schemes = ["http", "https"]
    result = urlparse(url)
    if result.scheme not in supported_schemes:
        raise ValueError("Scheme must be one of " + repr(supported_schemes))
    if result.netloc == '':
        raise ValueError("Host must be non-empty")

    # Do something with the URL
    return True
```

¿Cómo lo haríamos con el approach random?

Mutation-Based Fuzzing

```
for i in range(1000):
    try:
        url = fuzzer()
        result = http_program(url)
        print("Success!")
    except ValueError:
        pass
```

¿Cómo lo haríamos con el approach random?

Mutation-Based Fuzzing

```
for i in range(1000):
    try:
        url = fuzzer()
        result = http_program(url)
        print("Success!")
    except ValueError:
        pass
```

¿Cómo lo haríamos con el approach random?

Mutation-Based Fuzzing

```
for i in range
    try:
        url = fuzzed()
        result = http_program(url)
        print("Success!")
    except ValueError:
        pass
```

```
"https" --> 1:96**8
96 ** 8
```

Una opción mejor, mutar lo bueno

Mutation-Based Fuzzing

- Para hacer Mutation-Based Fuzzing **comenzamos con un input válido.**
- Paso seguido, manipulamos el valor e **introducimos un cambio.**

```
import random
def delete_random_character(s: str) -> str:
    """Returns s with a random character deleted"""
    if s == "":
        return s

    pos = random.randint(0, len(s) - 1)
    # print("Deleting", repr(s[pos]), "at", pos)
    return s[:pos] + s[pos + 1:]
```

Una opción mejor, mutar lo bueno

Mutation-Based Fuzzing

```
seed_input = "A quick brown fox"
for i in range(10):
    x = delete_random_character(seed_input)
    print(repr(x))
'A uick brown fox'
'A quic brown fox'
'A quick brown fo'
'A quic brown fox'
'A quick bown fox'
'A quick bown fox'
'A quick brown fx'
'A quick brown ox'
'A quick brow fox'
'A quic brown fox'
```

Mutadores

Mutation-Based Fuzzing

- Corresponden a funciones que nos permiten introducir modificaciones.

```
import random
def delete_random_character(s: str) -> str:
    """Returns s with a random character deleted"""
    if s == "":
        return s

    pos = random.randint(0, len(s) - 1)
    # print("Deleting", repr(s[pos]), "at", pos)
    return s[:pos] + s[pos + 1:]
```

Mutadores

Mutation-Based Fuzzing

```
for i in range(10):
    print(repr(insert_random_character(seed_input)))
'A quick brvown fox'
'A quwick brown fox'
'A qBuick brown fox'
'A quick broSwn fox'
'A quick brown fvox'
'A quick brown 3fox'
'A quick brNown fox'
'A quick brow4n fox'
'A quick brown fox8'
'A equick brown fox'
```

Mutadores

Mutation-Based Fuzzing

```
def flip_random_character(s):
    """Returns s with a random bit flipped in a random
position"""
    if s == "":
        return s

    pos = random.randint(0, len(s) - 1)
    c = s[pos]
    bit = 1 << random.randint(0, 6)
    new_c = chr(ord(c) ^ bit)
    # print("Flipping", bit, "in", repr(c) + ", giving",
repr(new_c))
    return s[:pos] + new_c + s[pos + 1:]
```

Mutadores

Mutation-Based Fuzzing

```
for i in range(10):
    print(repr(flip_random_character(seed_input)))
'A quick bRown fox'
'A quici brown fox'
'A"quick brown fox'
'A quick brown$fox'
'A quick bpown fox'
'A quick brown!fox'
'A 1uick brown fox'
'@ quick brown fox'
'A quic+ brown fox'
'A quick bsown fox'
```

Aplicamos todos los mutadores en uno

Mutation-Based Fuzzing

```
def mutate(s: str) -> str:
    """Return s with a random mutation applied"""
    mutators = [
        delete_random_character,
        insert_random_character,
        flip_random_character
    ]
    mutator = random.choice(mutators)
    # print(mutator)
    return mutator(s)

for i in range(10):
    print(repr(mutate("A quick brown fox")))
```

Aplicamos todos los mutadores en uno

Mutation-Based Fuzzing

```
'A qzuick brown fox'
' quick brown fox'
'A quick Brown fox'
'A qMuick brown fox'
'A qu_ick brown fox'
'A quick bXrown fox'
'A quick brown fx'
'A quick!brown fox'
'A! quick brown fox'
'A quick brownfox'
```

Volvamos al problema original

Mutation-Based Fuzzing

- La idea general es que podamos tomar un punto de partida y que podamos generar candidatos alternativos que tengan una **alta probabilidad de ser válidos.**
- **Recordatorio: queremos testear la funcionalidad de tanto de http_program() como de urlparse().**

¿Qué vamos a testear hoy?

Mutation-Based Fuzzing

```
def http_program(url: str) -> bool:
    supported_schemes = ["http", "https"]
    result = urlparse(url)
    if result.scheme not in supported_schemes:
        raise ValueError("Scheme must be one of " + repr(supported_schemes))
    if result.netloc == '':
        raise ValueError("Host must be non-empty")

    # Do something with the URL
    return True
```

Volvamos al problema original

Mutation-Based Fuzzing

```
def is_valid_url(url: str) -> bool:
    try:
        result = http_program(url)
        return True
    except ValueError:
        return False

assert is_valid_url("http://www.google.com/search?q=fuzzing")
assert not is_valid_url("xyzzy")
```

Podemos notar una clara mejora

Mutation-Based Fuzzing

```
seed_input = "http://www.google.com/search?q=fuzzing"
valid_inputs = set()
trials = 20

for i in range(trials):
    inp = mutate(seed_input)
    if is_valid_url(inp):
        valid_inputs.add(inp)

len(valid_inputs) / trials
0.8
```

Multiple Mutations

MutationFuzzer

Multiple Mutations

```
class MutationFuzzer(Fuzzer):
    """Base class for mutational fuzzing"""

    def __init__(self, seed: List[str],
                 min_mutations: int = 2,
                 max_mutations: int = 10) -> None:
        """Constructor.
        `seed` - a list of (input) strings to mutate.
        `min_mutations` - the minimum number of mutations to apply.
        `max_mutations` - the maximum number of mutations to apply.
        """
        self.seed = seed
        self.min_mutations = min_mutations
        self.max_mutations = max_mutations
        self.reset()

    def reset(self) -> None:
        """Set population to initial seed.
        To be overloaded in subclasses."""
        self.population = self.seed
        self.seed_index = 0
```

```
class MutationFuzzer(Fuzzer):
    """Base class for mutational fuzzing"""

    def __init__(self, seed: List[str],
                 min_mutations: int = 2,
                 max_mutations: int = 10) -> None:
        """Constructor.
        `seed` - a list of (input) strings to mutate.
        `min_mutations` - the minimum number of mutations to apply.
        `max_mutations` - the maximum number of mutations to apply.
        """
        self.seed = seed
        self.min_mutations = min_mutations
        self.max_mutations = max_mutations
        self.reset()

    def reset(self) -> None:
        """Set population to initial seed.
        To be overloaded in subclasses."""
        self.population = self.seed
        self.seed_index = 0
```

```
class MutationFuzzer(MutationFuzzer):
    def mutate(self, inp: str) -> str:
        return mutate(inp)

    def create_candidate(self) -> str:
        """Create a new candidate by mutating a population member"""
        candidate = random.choice(self.population)
        trials = random.randint(self.min_mutations, self.max_mutations)
        for i in range(trials):
            candidate = self.mutate(candidate)
        return candidate

    def fuzz(self) -> str:
        if self.seed_index < len(self.seed):
            self.inp = self.seed[self.seed_index]
            self.seed_index += 1
        else:
            # Mutating
            self.inp = self.create_candidate()
        return self.inp
```

Coverage Guided Mutations

Para empezar

Coverage Guided Mutations

- Para mantener las cosas simples, vamos a **apuntar a testear la mayor cantidad de funcionalidades posibles (coverage).**
- Si apuntamos a poder recorrer todo el código buscando bugs, **¿Cómo podemos guiar la creación de tests tomando como base el coverage?**

- **La idea es simple, vamos a intentar hacer “evolucionar” los casos de prueba que nos permitan obtener nuevos coverages en cada ejecución.**

La clase Runner:

Coverage Guided Mutations

```
from Fuzzer import Runner

class FunctionRunner(Runner):
    def __init__(self, function: Callable) -> None:
        """Initialize.  `function` is a function to be executed"""
        self.function = function

    def run_function(self, inp: str) -> Any:
        return self.function(inp)

    def run(self, inp: str) -> Tuple[Any, str]:
        try:
            result = self.run_function(inp)
            outcome = self.PASS
        except Exception:
            result = None
            outcome = self.FAIL

        return result, outcome
```

```
from Fuzzer import Runner

class FunctionRunner(Runner):
    def __init__(self, function: Callable) -> None:
        """Initialize.  `function` is a function to be executed"""
        self.function = function

    def run_function(self, inp: str) -> Any:
        return self.function(inp)

    def run(self, inp: str) -> Tuple[Any, str]:
        try:
            result = self.run_function(inp)
            outcome = self.PASS
        except Exception:
            result = None
            outcome = self.FAIL

        return result, outcome
```

```
http_runner = FunctionRunner(http_program)
http_runner.run("https://foo.bar/")
(True, 'PASS')
```

También queremos que considere el Coverage

Coverage Guided Mutations

```
from Fuzzer import Runner

class FunctionRunner(Runner):
    def __init__(self, function: Callable) -> None:
        """Initialize.  `function` is a function to be executed"""
        self.function = function

    def run_function(self, inp: str) -> Any:
        return self.function(inp)

    def run(self, inp: str) -> Tuple[Any, str]:
        try:
            result = self.run_function(inp)
            outcome = self.PASS
        except Exception:
            result = None
            outcome = self.FAIL

        return result, outcome
```

```
from Coverage import Coverage, population_coverage, Location

class FunctionCoverageRunner(FunctionRunner):
    def run_function(self, inp: str) -> Any:
        with Coverage() as cov:
            try:
                result = super().run_function(inp)
            except Exception as exc:
                self._coverage = cov.coverage()
                raise exc

        self._coverage = cov.coverage()
        return result

    def coverage(self) -> Set[Location]:
        return self._coverage
```

Obtenemos las trazas de cada ejecución

Coverage Guided Mutations

```
http_runner = FunctionCoverageRunner(http_program)
http_runner.run("https://foo.bar/")
print(list(http_runner.coverage())[:5])

[('urlsplit', 460), ('urlsplit', 466), ('http_program', 2),
('urlparse', 394), ('urlparse', 400)]
```

Mutation Coverage Fuzzer

MutationCoverageFuzzer

Mutation Coverage Fuzzer

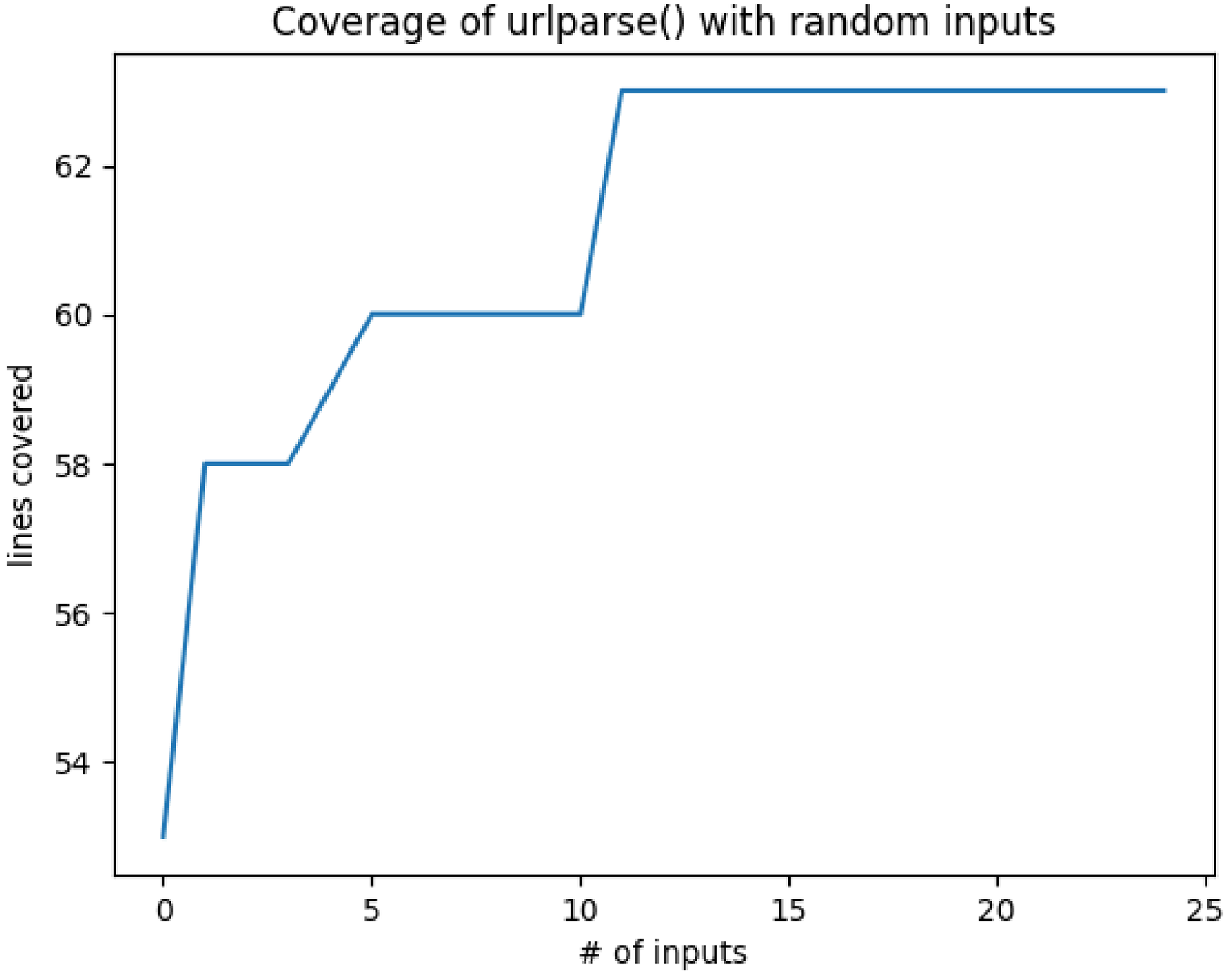
- Generamos un clase final que nos permita **crear inputs válidos y que apunten a mejorar el coverage.**
- Generamos **mutantes de los candidatos que nos permitan avanzar** y vamos haciendo nuevas generaciones.

```python
class MutationCoverageFuzzer(MutationFuzzer):
    """Realizamos Fuzzing con inputs mutados basados en el coverage"""

    def reset(self) -> None:
        super().reset()
        self.coverages_seen: Set[frozenset] = set()
        self.population = [] # Se va llenando a medida que itera

    def run(self, runner: FunctionCoverageRunner) -> Any:
        """Ejecutamos la función mientras obtenemos el coverage:
           Si obtenemos un coverage nuevo, agregamos al individuo
        """
        result, outcome = super().run(runner)
        new_coverage = frozenset(runner.coverage())
        if outcome == Runner.PASS and new_coverage not in self.coverages_seen:
            # Obtenemos un nuevo coverage y agregamos al individuo
            self.population.append(self.inp)
            self.coverages_seen.add(new_coverage)

        return result
```

```
seed_input = "http://www.google.com/search?q=fuzzing"
mutation_fuzzer = MutationCoverageFuzzer(seed=[seed_input])
mutation_fuzzer.runs(http_runner, trials=10000)
mutation_fuzzer.population
```

```
['http://www.google.com/search?q=fuzzing',
 'http://www.goog.com/search;q=fuzzilng',
 'http://ww.6goog\x0eoomosearch;/q=f}zzilng',
 'http://uv.Lboo.comoseakrch;q=fuzilng',
 'http://ww.6goog\x0eo/mosarch;/q=f}z{il~g',
 'http://www.googme.com/sear#h?q=fuzzing']
```

```
import matplotlib.pyplot as plt
all_coverage, cumulative_coverage = population_coverage( mutation_fuzzer.population, http_program)
plt.plot(cumulative_coverage)
plt.title('Coverage of urlparse() with random inputs')
plt.xlabel('# of inputs')
plt.ylabel('lines covered');
```


Coverage of urlparse() with random inputs
lines covered
# of inputs
54
56
58
60
62
0
5
10
15
20
25