# COSE419: Software Verification

## Lecture 2 — Greybox Fuzzing

Hakjoo Oh
2024 Spring

# Setup

- $P$: Program under test (PUT)
- Program execution:

$$\llbracket P \rrbracket : \Sigma^* \to \mathcal{R}$$

  - $\Sigma$: input characters
  - $\mathcal{R}$: execution results
- Test oracle:

$$\textsc{Oracle} : \Sigma^* \times \mathcal{R} \to \{\bot, \top\}$$

  - $\top$: the program has run correctly (expected outcome)
  - $\bot$: the program has run incorrectly (unexpected outcome)
  - E.g., "crash oracle", reference implementation, etc

# Random Fuzzing

```
procedure $\textsc{RandomFuzzer}(P)$
    $bugs \leftarrow \emptyset$
    repeat
        $inp \leftarrow \textsc{Sample}(\Sigma^*)$
        $res \leftarrow [\![P]\!](inp)$
        if $\textsc{Oracle}(inp, res) = \bot$ then
            $bugs \leftarrow bugs \cup \{inp\}$
        end if
    until time budget expires
    return $bugs$
end procedure
```

# Limitations

- Programs typically expect inputs in specific languages ($L \subseteq \Sigma^*$)
  - e.g., web browsers, image processors, compilers, etc
- Random inputs are unlikely to exercise deep program paths

# Mutation-based Blackbox Fuzzing

- Mutation-based fuzzers start with valid inputs and then subsequently mutate them to generate test inputs
  - $S : \wp(\Sigma^*)$: a seed corpus
  - $M = \{m_1, m_2, \dots\}$: a set of mutators ($m_i : \Sigma^* \rightarrow \Sigma^*$), e.g.,
    - ⋆ Deleting a random character, e.g., $abc \rightarrow ac$
    - ⋆ Inserting a random character, e.g., $abc \rightarrow ab\$c$
  - $\textsc{Select}(S)$: randomly pick a seed in $S$
  - $\textsc{Mutate}(inp, M)$:

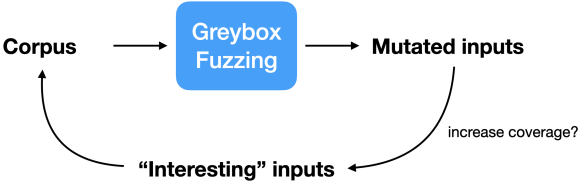
```
for $k \leftarrow 1$ to $n$ do                    ▷ $n$: random integer
    $m_i \leftarrow \textsc{Sample}(M)$
    $inp \leftarrow m_i(inp)$
end for
return $inp$
```

# Mutation-based Blackbox Fuzzing

```
procedure $\textsc{BlackBoxFuzzer}(P, S, M)$
    $bugs \leftarrow \emptyset$
    repeat
        $inp \leftarrow \textsc{Mutate}(\textsc{Select}(S), M)$
        $res \leftarrow [\![P]\!](inp)$
        if $\textsc{Oracle}(inp, res) = \bot$ then
            $bugs \leftarrow bugs \cup \{inp\}$
        end if
    until time budget expires
    return $bugs$
end procedure
```

# Greybox Fuzzing

- A mutation-based, coverage-guided approach
    - Mutation-based: use a set of valid inputs and randomly mutate them to preserve the input format as much as possible
    - Coverage-guided: add the generated input to the seed corpus only when coverage increase is observed



- Instrumented program execution:

$$[\![P]\!] : \Sigma^* \to \mathcal{R} \times Coverage$$

# (Structural) Code Coverage

A metric to measure the extent to which a program has been tested: e.g.,

- Function coverage: $\frac{\text{\# of executed functions}}{\text{\# of functions}}$
- Statement coverage: $\frac{\text{\# of executed statements}}{\text{\# of statements}}$
- Branch (decision) coverage: $\frac{\text{\# of executed branches}}{\text{\# of branches}}$
- Condition coverage: $\frac{\text{\# of conditions evaluated to both true and false}}{\text{\# of atomic conditions}}$
- (Modified) condition/decision coverage, path coverage, . . .

# (Structural) Code Coverage

```
int foo (int x, int y) {
    int z = 0;
    if (x > 3 && y < 6) {
        z = x;
    }
    return z; }
```

Test inputs for 100%

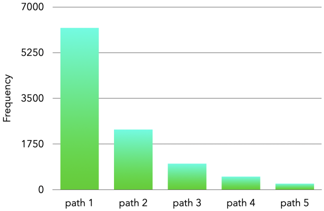
- function coverage:
- statement coverage:
- branch coverage:
- condition coverage:
- condition/decision coverage:
- modified condition/decision coverage:

# Greybox Fuzzing

```
procedure $\textsc{GreyBoxFuzzer}(P, S, M)$
    $corpus, covered, bugs \leftarrow S, \emptyset, \emptyset$
    repeat
        $inp \leftarrow \textsc{Mutate}(\textsc{Select}(corpus), M)$
        $res, cov \leftarrow [\![P]\!](inp)$
        if $\textsc{Oracle}(inp, res) = \perp$ then
            $bugs \leftarrow bugs \cup \{inp\}$
        end if
        if $cov \notin covered$ then
            $corpus, covered \leftarrow corpus \cup \{inp\}, covered \cup \{cov\}$
        end if
    until time budget expires
    return $bugs$
end procedure
```

# Boosted Greybox Fuzzing

- Observation: Most tests exercise few “high-frequency” paths



- Idea: Prefer to choose seeds that exercise “low-frequency” paths
- SELECT chooses a seed $s$ with probability

$$\frac{1}{freq(p)^a}$$

  - $p$: the path exercised by $s$
  - $freq(p)$: the number of times $p$ is exercised
  - $a$: a given exponent, e.g., $a = 5$
- Seed is associated with coverage, i.e., $(s, c)$ where $\_, c = [\![P]\!](s)$

# Boosted Greybox Fuzzing

```
**procedure** $\textsc{BoostedGreyBoxFuzzer}(P, S, M)$
    $corpus, covered, bugs, freq \leftarrow \{(seed, \emptyset) \mid seed \in S\}, \emptyset, \emptyset, \lambda p.0$
    **repeat**
        $(inp, \_) \leftarrow \textsc{Mutate}(\textsc{Select}(corpus, freq), M)$
        $res, cov \leftarrow [\![P]\!](inp)$
        **if** $\textsc{Oracle}(inp, res) = \bot$ **then**
            $bugs \leftarrow bugs \cup \{inp\}$
        **end if**
        **if** $cov \notin covered$ **then**
            $corpus, covered \leftarrow corpus \cup \{(inp, cov)\}, covered \cup \{cov\}$
        **end if**
        $freq(pathid(cov)) \leftarrow freq(pathid(cov)) + 1$
    **until** time budget expires
    **return** $bugs$
**end procedure**
```

# Implementation in Python[1]

- Example programs:

```
def crashme(s: str) -> None:
    if len(s) > 0 and s[0] == ’b’:
        if len(s) > 1 and s[1] == ’a’:
            if len(s) > 2 and s[2] == ’d’:
                if len(s) > 3 and s[3] == ’!’:
                    raise Exception()

def html_parser(inp: str) -> None:
    parser = HTMLParser()
    parser.feed(inp)
```

- Instrumented execution:

```
def run(function: Callable, inp: str):
    with Coverage() as cov:
        try:
            result = function(inp)
            return True, result, cov.coverage()
        except Exception:
            return False, None, cov.coverage()
```

[1] http://https://prl.korea.ac.kr/courses/cose419/2024/greybox.py

# Random Fuzzing

```
def random_fuzzer(function: Callable, trials: int, max_length : int = 100,
                  char_start : int = 32, char_range : int = 32):
    data = []
    for i in range(trials):
        length = random.randrange(0, max_length + 1)
        inp = ""
        for i in range(length):
            inp += chr(random.randrange(char_start, char_start + char_range))
        outcome, result, coverage = run(function, inp)
        data.append((inp, outcome, result, coverage))
    return data


random_fuzzer(crashme, 5000, max_length = 30)
```

# Mutation-based Blackbox Fuzzing

```
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)
    return s[:pos] + s[pos + 1:]

def insert_random_character(s: str) -> str:
    """Returns s with a random character inserted"""
    pos = random.randint(0, len(s))
    random_character = chr(random.randrange(32, 127))
    return s[:pos] + random_character + s[pos:]

def flip_random_character(s: str) -> str:
    """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)
    return s[:pos] + new_c + s[pos + 1:]
```

# Mutation-based Blackbox 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)
    return mutator(s)

def create_candidate(population, schedule):
    seed = schedule.choose(population)
    candidate = seed.data

    trials = min(len(candidate), 1 << random.randint(1, 5))
    for i in range(trials):
        candidate = mutate(candidate)
    return candidate
```

# Mutation-based Blackbox Fuzzing

```
def blackbox_fuzzer(function: Callable, seeds : List[str], schedule, trials : int):
    data = []
    population = list(map(lambda x: Seed(x), seeds))
    seed_index = 0

    for i in range(trials):
        if seed_index < len(seeds):
            inp = seeds[seed_index]
            seed_index += 1
        else:
            inp = create_candidate(population, schedule)

        outcome, result, coverage = run(function, inp)
        data.append((inp, outcome, result, coverage))

    return data
```

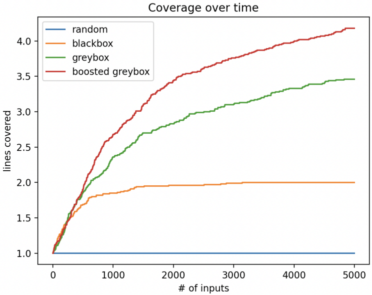
# Greybox Fuzzing

```
def greybox_fuzzer(function: Callable, seeds : List[str], schedule, trials : int):
    coverages_seen : Set[frozenset] = set()
    population = []
    data = []
    seed_index = 0

    for i in range(trials):
        if seed_index < len(seeds):
            inp = seeds[seed_index]
            seed_index += 1
        else:
            inp = create_candidate(population, schedule)

        outcome, result, coverage = run(function, inp)
        data.append((inp, outcome, result, coverage))

        new_coverage = frozenset(coverage)
        if new_coverage not in coverages_seen:
            seed = Seed(inp)
            seed.coverage = new_coverage
            coverages_seen.add(new_coverage)
            population.append(seed)

    return data
```

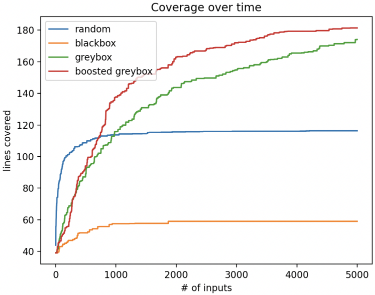
# Boosted Greybox Fuzzing

```
def boosted_greybox_fuzzer(function: Callable, seeds : List[str], schedule, trials
    coverages_seen, population, data, seed_index = set(), [], [], 0
    schedule.path_frequency = {}
    for i in range(trials):
        if seed_index < len(seeds): ...
        else:
            inp = create_candidate(population, schedule)

        outcome, result, coverage = run(function, inp)
        data.append((inp, outcome, result, coverage))

        new_coverage = frozenset(coverage)
        if new_coverage not in coverages_seen:
            seed = Seed(inp)
            seed.coverage = new_coverage
            coverages_seen.add(new_coverage)
            population.append(seed)

        path_id = getPathID(coverage)
        if path_id not in schedule.path_frequency:
            schedule.path_frequency[path_id] = 1
        else:
            schedule.path_frequency[path_id] += 1
    return data
```

# Comparison



crashme
(avg. over 100 runs)



html_parser
(avg. over 10 runs)

# cf) Instrumenting C Programs

```
$ gcc test.c
$ gcc --coverage test.c
$ ./a.out
$ gcov test.c
File 'test.c'
Lines executed:75.00% of 4
Creating 'test.c.gcov'

$ cat test.c.gcov
        -:    0:Source:test.c
        -:    0:Graph:test.gcno
        -:    0:Data:test.gcda
        -:    0:Runs:1
        1:    1:int main(int argc, char *argv[]) {
        1:    2:    if (argc >= 2) {
    #####:    3:        return 1;
        -:    4:    }
        -:    5:    else {
        1:    6:        return 0;
        -:    7:    }
        -:    8:}
```

# cf) LLVM Address Sanitizer

```
#include <stdlib.h>
#include <string.h>

int main(int argc, char** argv) {
    char *buf = malloc(100);
    int index = atoi(argv[1]);
    char val = buf[index];  // potential buffer overflow
    free(buf);
    return val;
}

$ clang -g -o program program.c
$ ./program 100
$ clang -fsanitize=address -g -o program program.c
$ ./program 100
=================================================================
==3657147==ERROR: AddressSanitizer: heap-buffer-overflow on address 0x60b00000015e a
READ of size 1 at 0x60b00000015e thread T0
    #0 0x4c31ca in main /home/vagrant/test/program.c:11:16
    #1 0x7fe5783fa082 in __libc_start_main (/lib/x86_64-linux-gnu/libc.so.6+0x24082
    #2 0x41b2dd in _start (/home/vagrant/test/program+0x41b2dd)
...
```

# Summary

- Greybox fuzzing:
  - One of the most successful approaches to fining bugs
  - Active research area:
    - ★ How to effectively instrument programs?
    - ★ How to effectively mutate programs?
    - ★ How to enhance fuzzing with AI?
    - ★ How to combine program analysis with fuzzing?
    - ★ ...
- Applications:
  - Finding bugs in compilers, databases, deep learning frameworks, quantum computing platforms, ...
- Other approaches to fuzzing:
  - Grammar-based fuzzing, search-based fuzzing, concolic fuzzing, ...
- Reference: The Fuzzing Book (`https://www.fuzzingbook.org`)