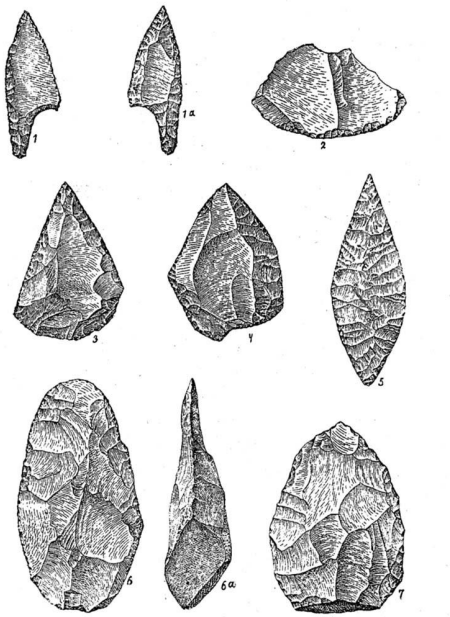
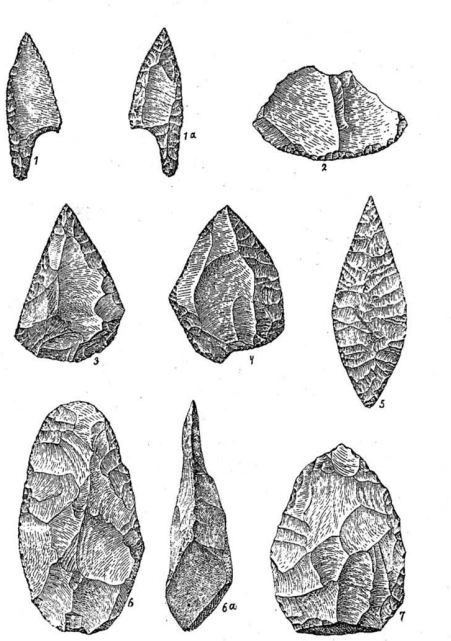
데이터 흐름 분석을 통한

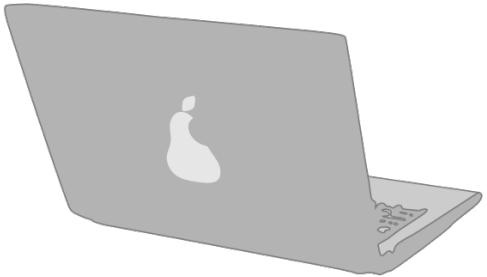
효율적인 지향성 퍼징

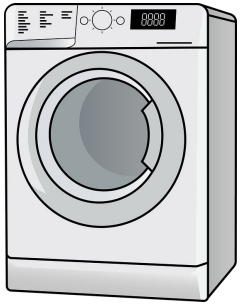
발표: 2022.07.14 김태은
연구: 김태은, 최재승, 허기홍, 차상길

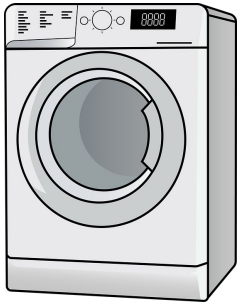
KAIST Programming Systems Laboratory

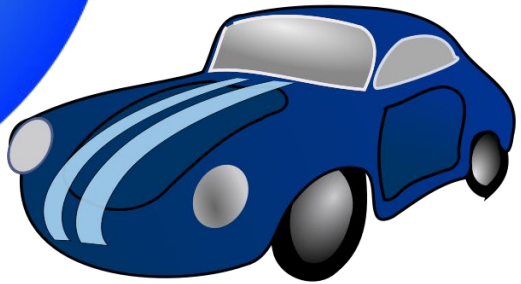


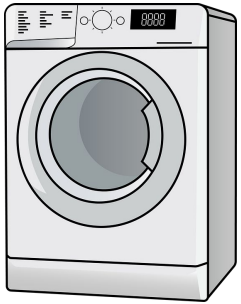


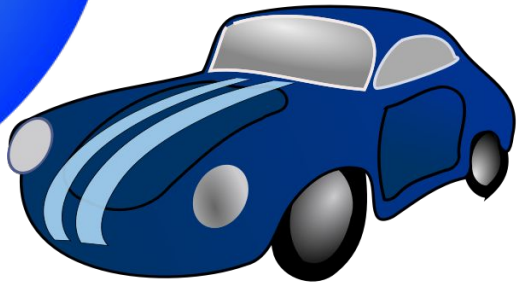


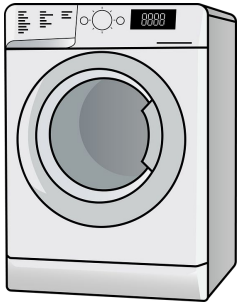


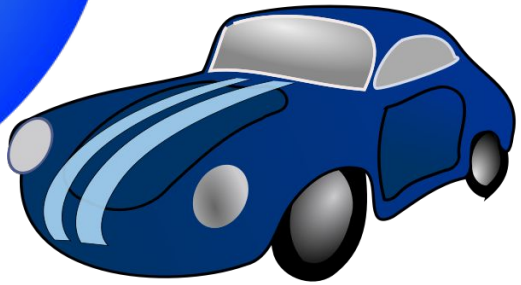


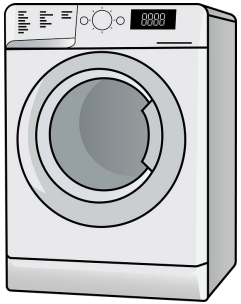


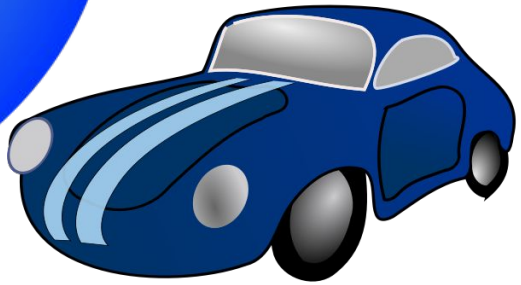




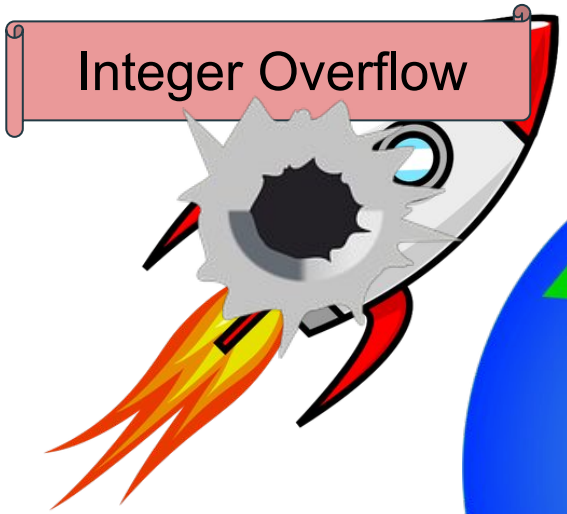


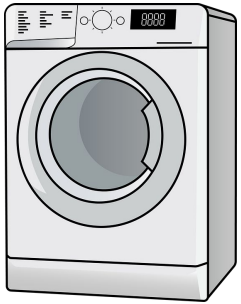


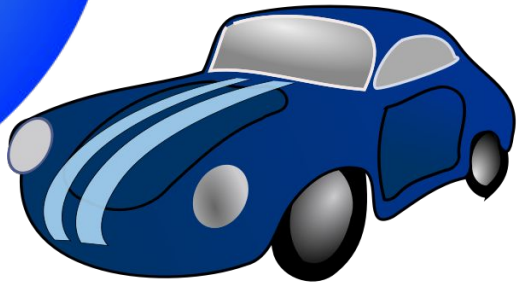




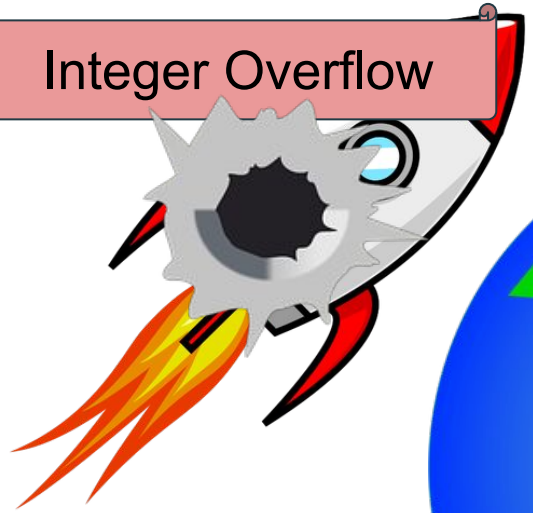
Integer Overflow



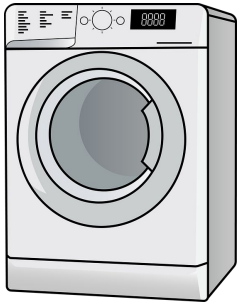


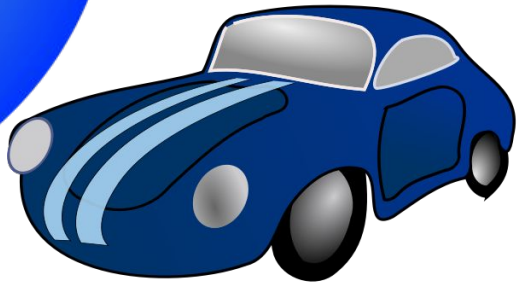


Integer Overflow

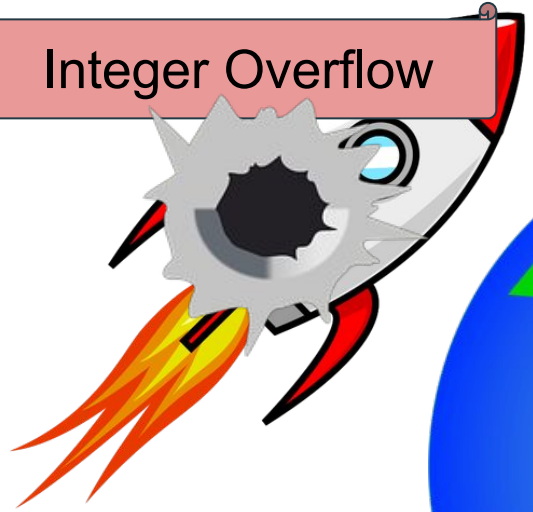


Buffer Overrun



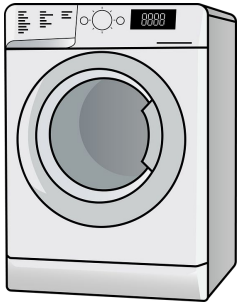


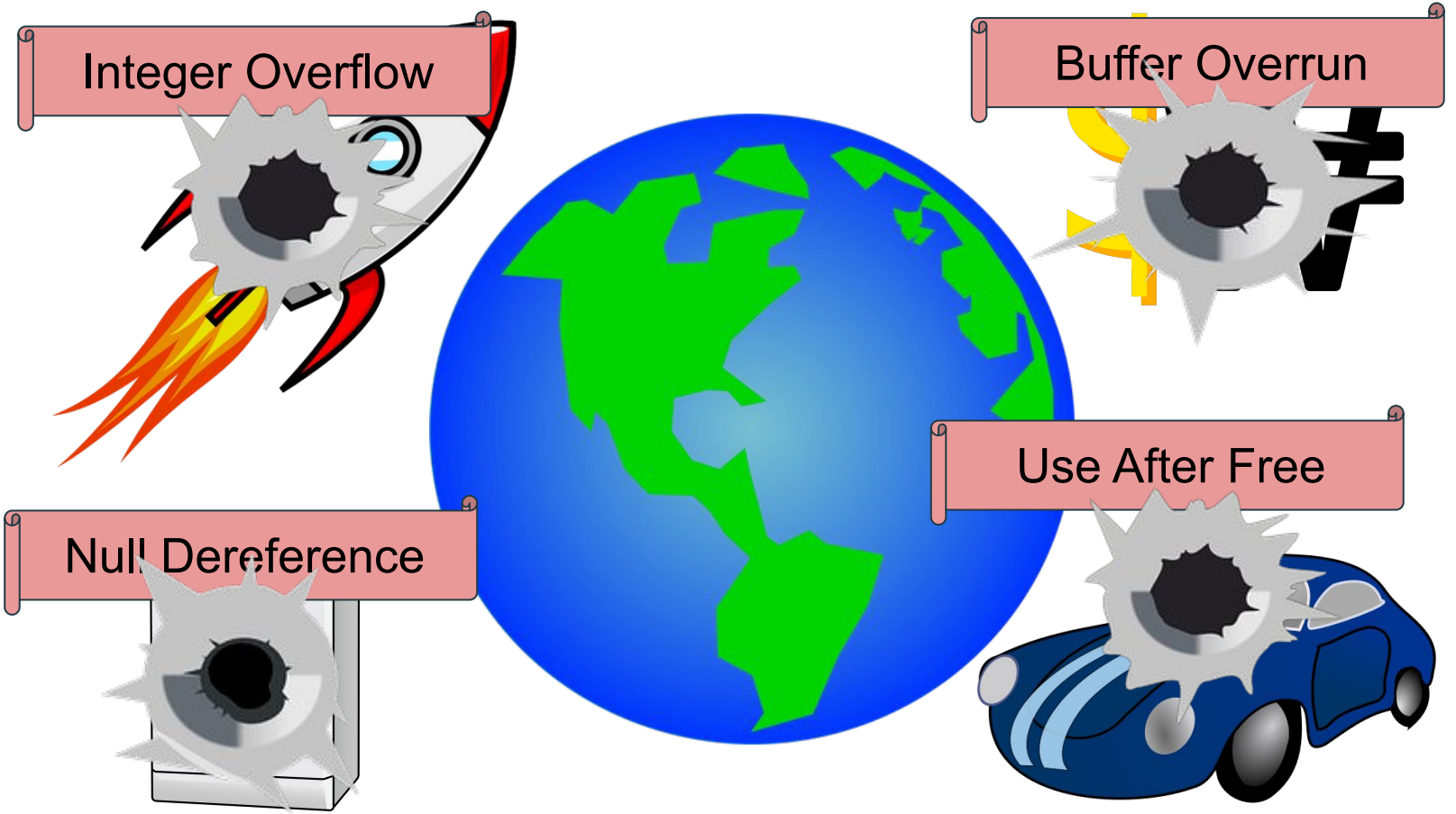
Integer Overflow



Buffer Overrun

Use After Free




Integer Overflow
Buffer Overrun
Use After Free
Null Dereference

OSS-Fuzz: 지구방위대 구글

구글의 넘치는 컴퓨팅 파워를 기반으로 한 퍼징 플랫폼.

OSS-Fuzz: 지구방위대 구글

구글의 넘치는 컴퓨팅 파워를 기반으로 한 퍼징 플랫폼.

- 누구나 자신의 퍼징 도구 / 오픈소스프로젝트를 등록 가능

OSS-Fuzz: 지구방위대 구글

구글의 넘치는 컴퓨팅 파워를 기반으로 한 퍼징 플랫폼.

- 누구나 자신의 퍼징 도구 / 오픈소스프로젝트를 등록 가능

현재 기준, 650개 오픈소스 프로젝트에서 약 37,000 건의 결함 발견

퍼징(Fuzzing)이란?

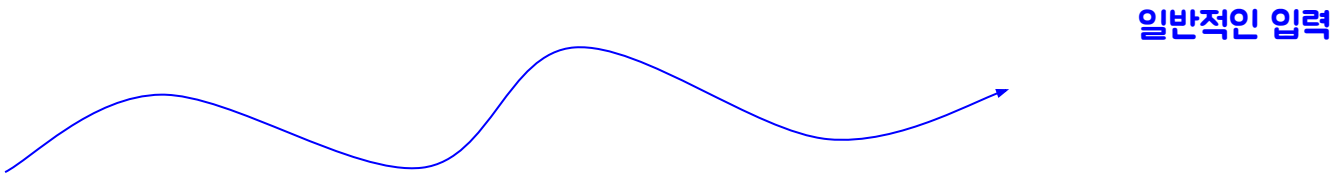
퍼징, 혹은 퍼즈 테스팅:

자동으로 생성한 입력을 통해 프로그램을 테스트하는 기법

퍼징(Fuzzing)이란?

퍼징, 혹은 퍼즈 테스팅:

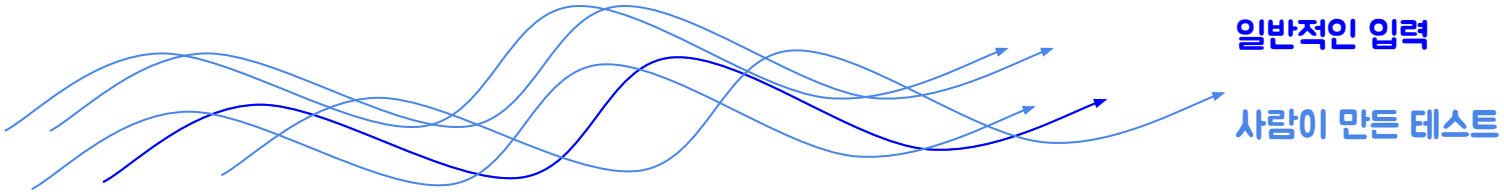
자동으로 생성한 입력을 통해 프로그램을 테스트하는 기법



퍼징(Fuzzing)이란?

퍼징, 혹은 퍼즈 테스팅:

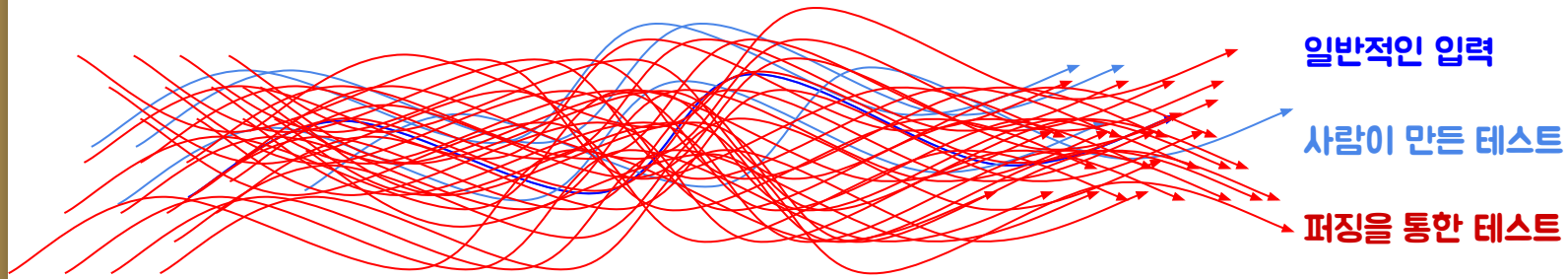
자동으로 생성한 입력을 통해 프로그램을 테스트하는 기법



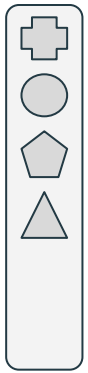
퍼징(Fuzzing)이란?

퍼징, 혹은 퍼즈 테스팅:

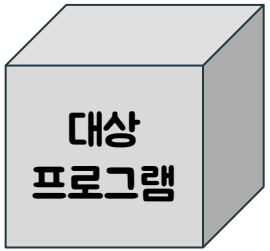
자동으로 생성한 입력을 통해 프로그램을 테스트하는 기법



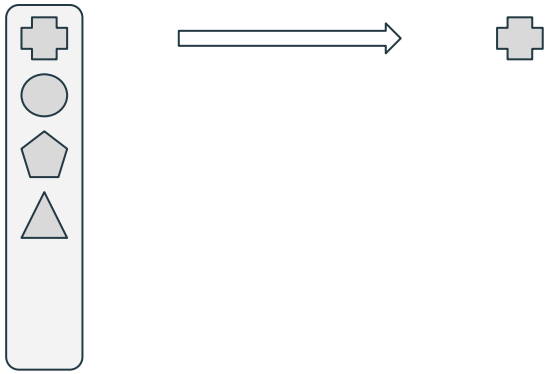
Mutation-based Greybox fuzzing



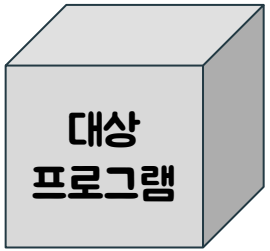
Seed 리스트



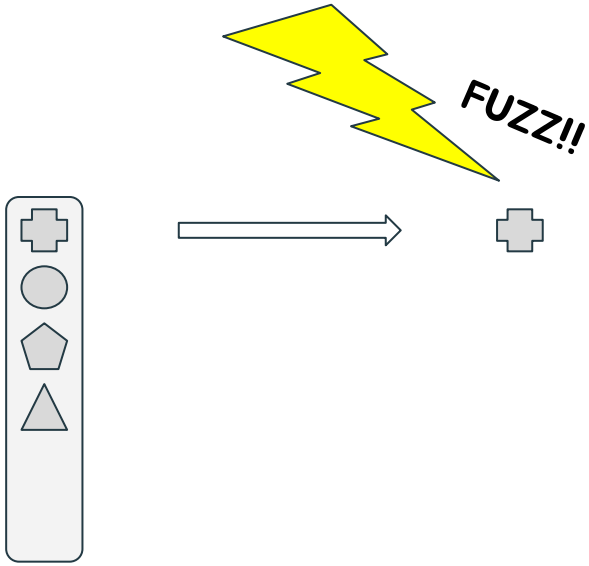
Mutation-based Greybox fuzzing



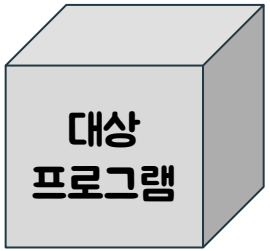
Seed 리스트



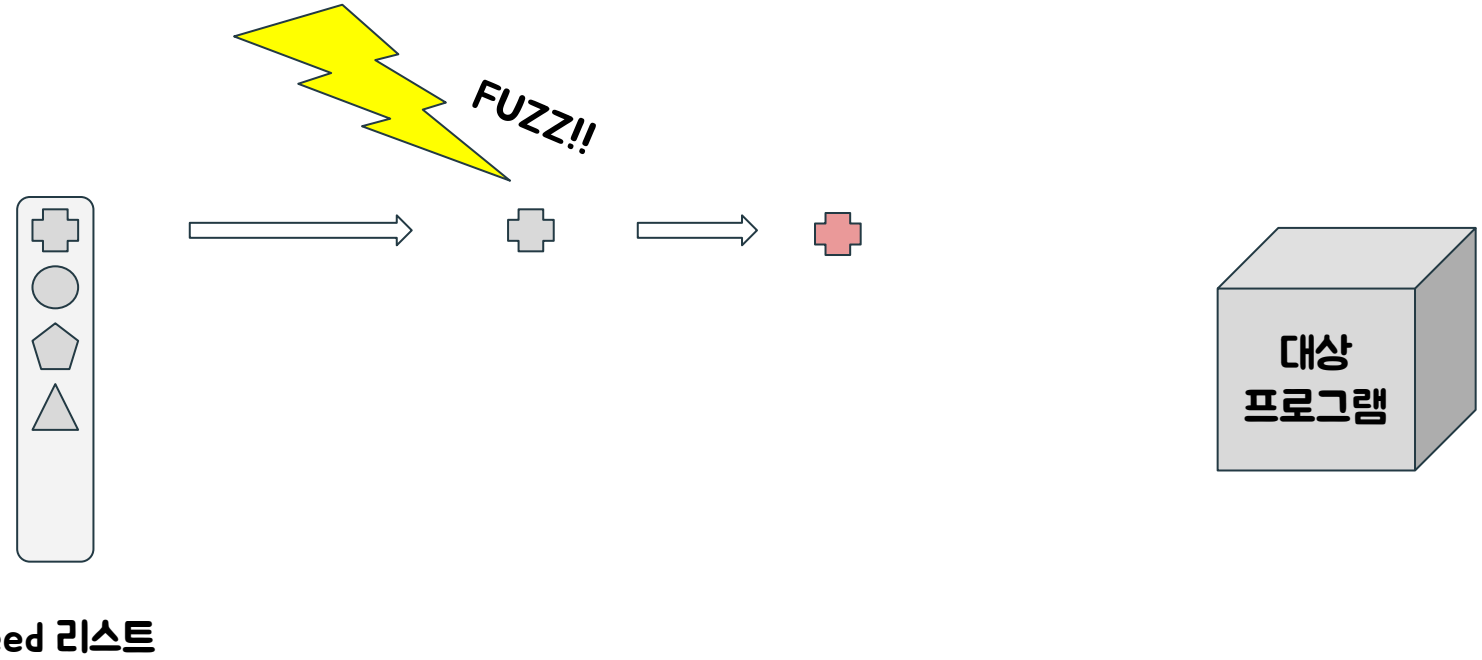
Mutation-based Greybox fuzzing



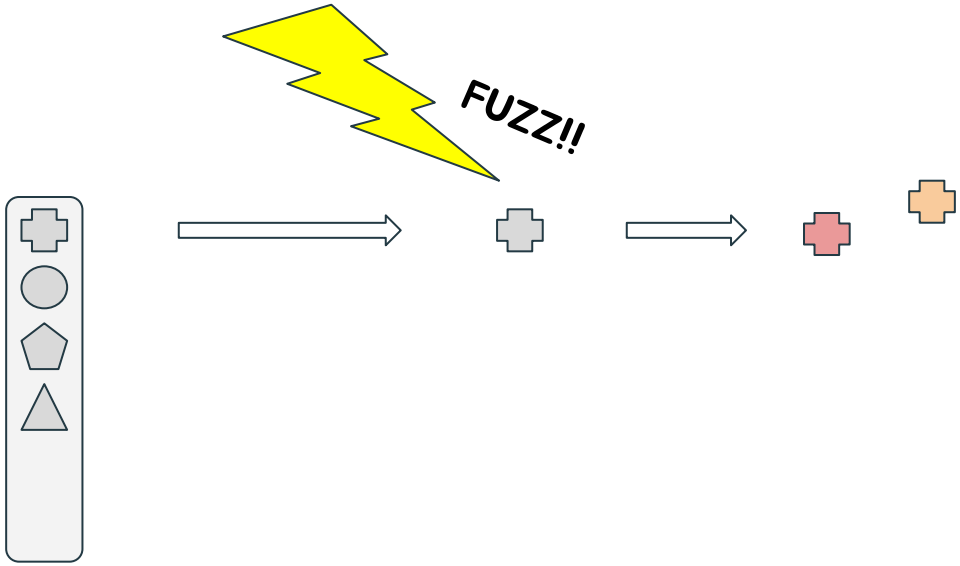
Seed 리스트



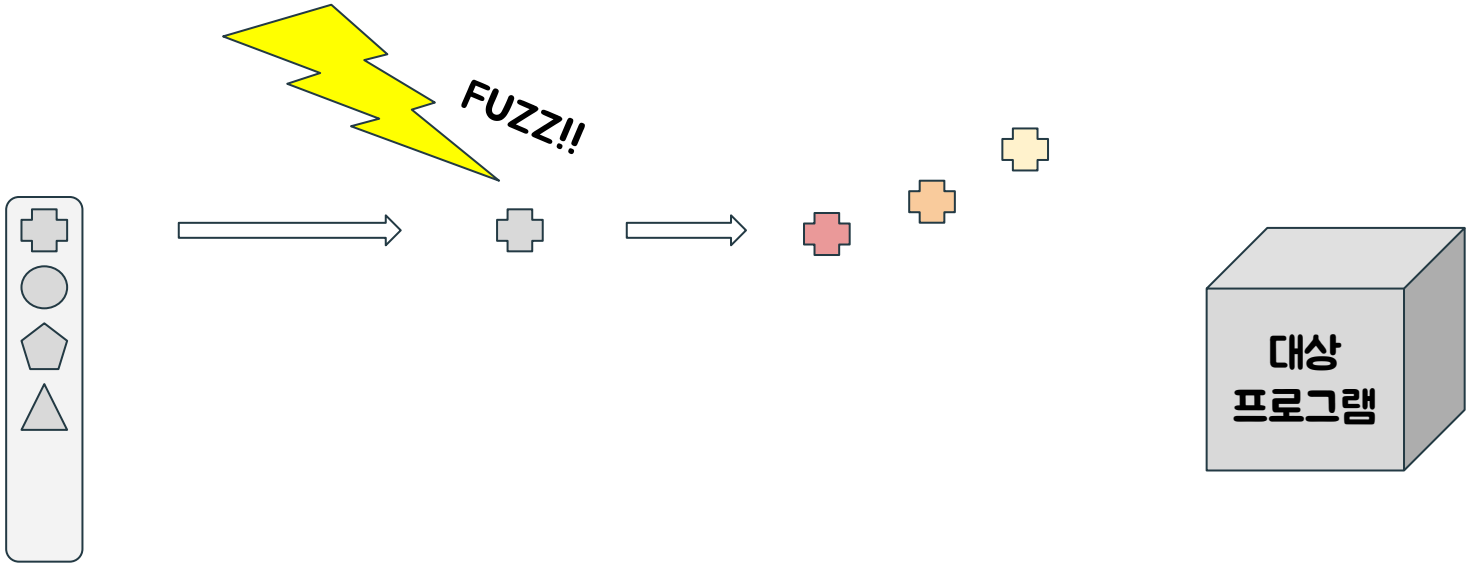
Mutation-based Greybox fuzzing



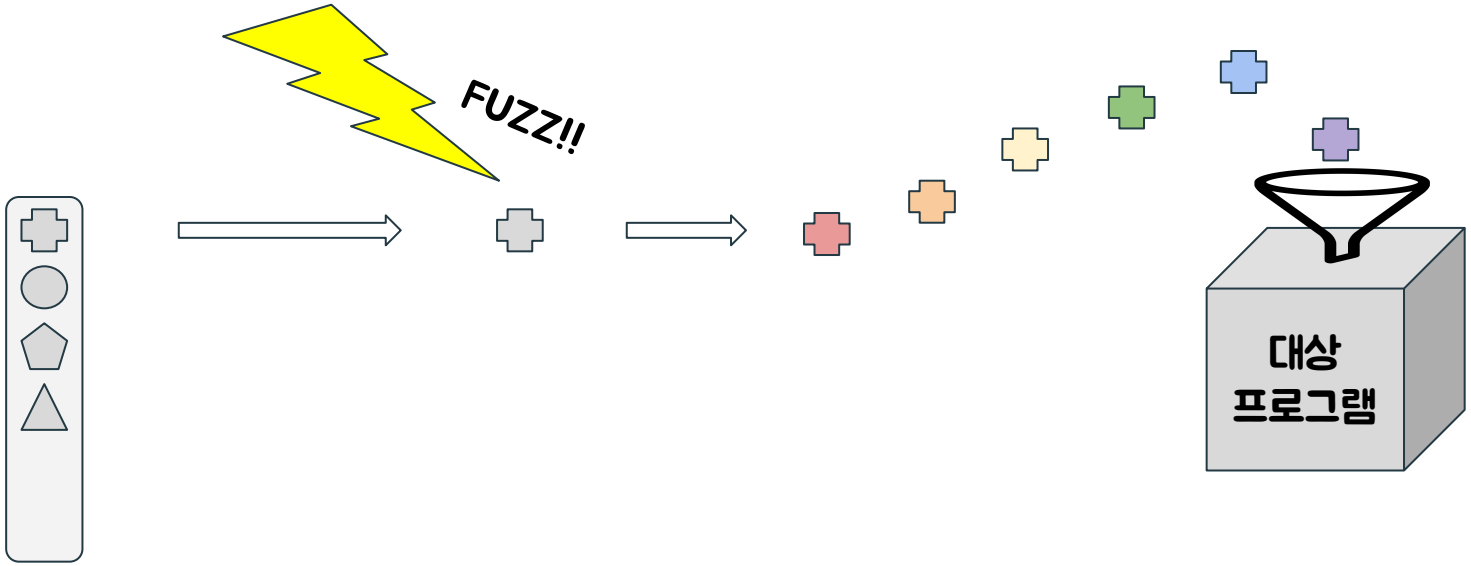
Mutation-based Greybox fuzzing



Mutation-based Greybox fuzzing

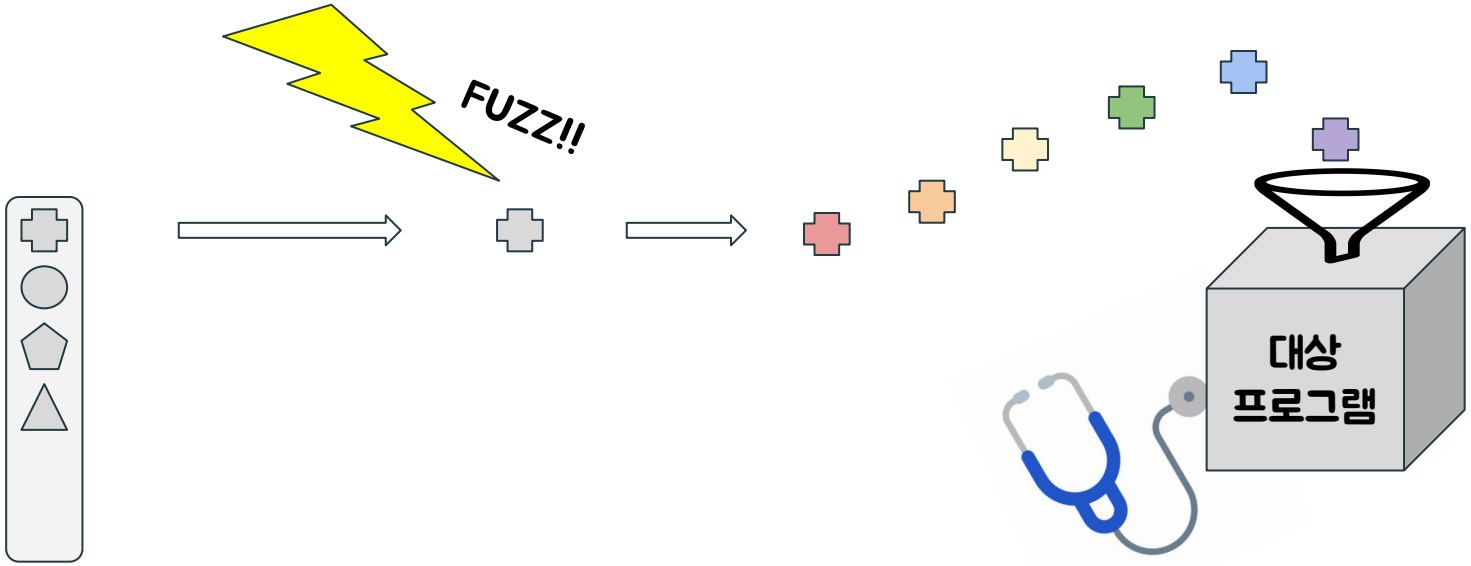


Mutation-based Greybox fuzzing

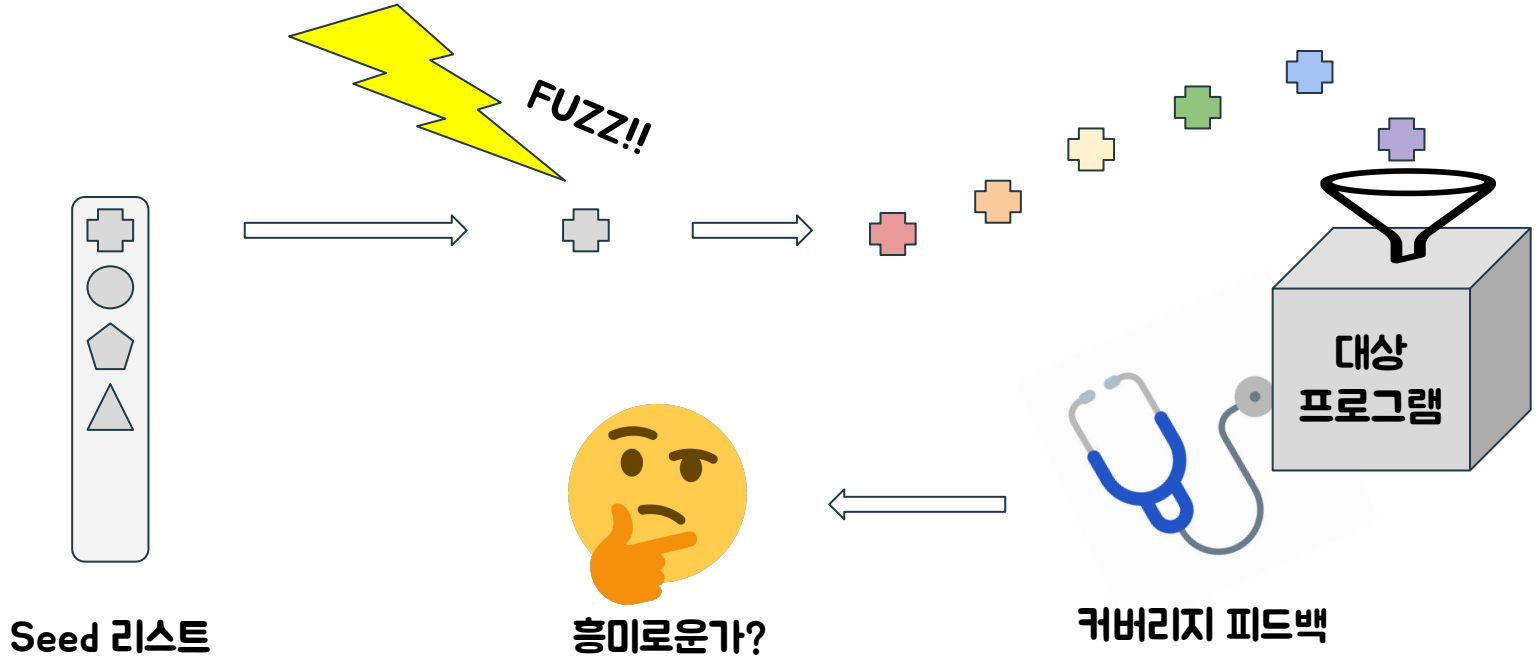


Seed 리스트

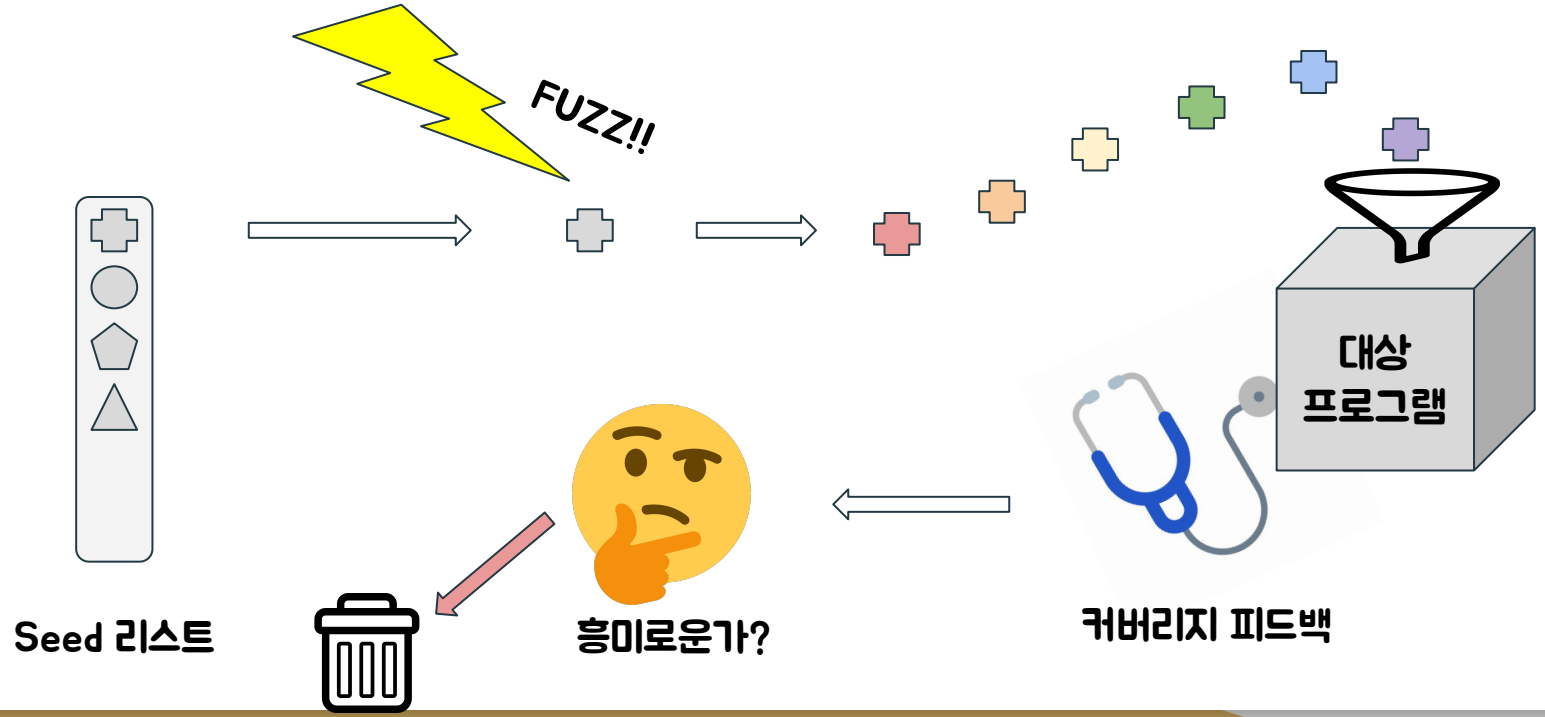
Mutation-based Greybox fuzzing



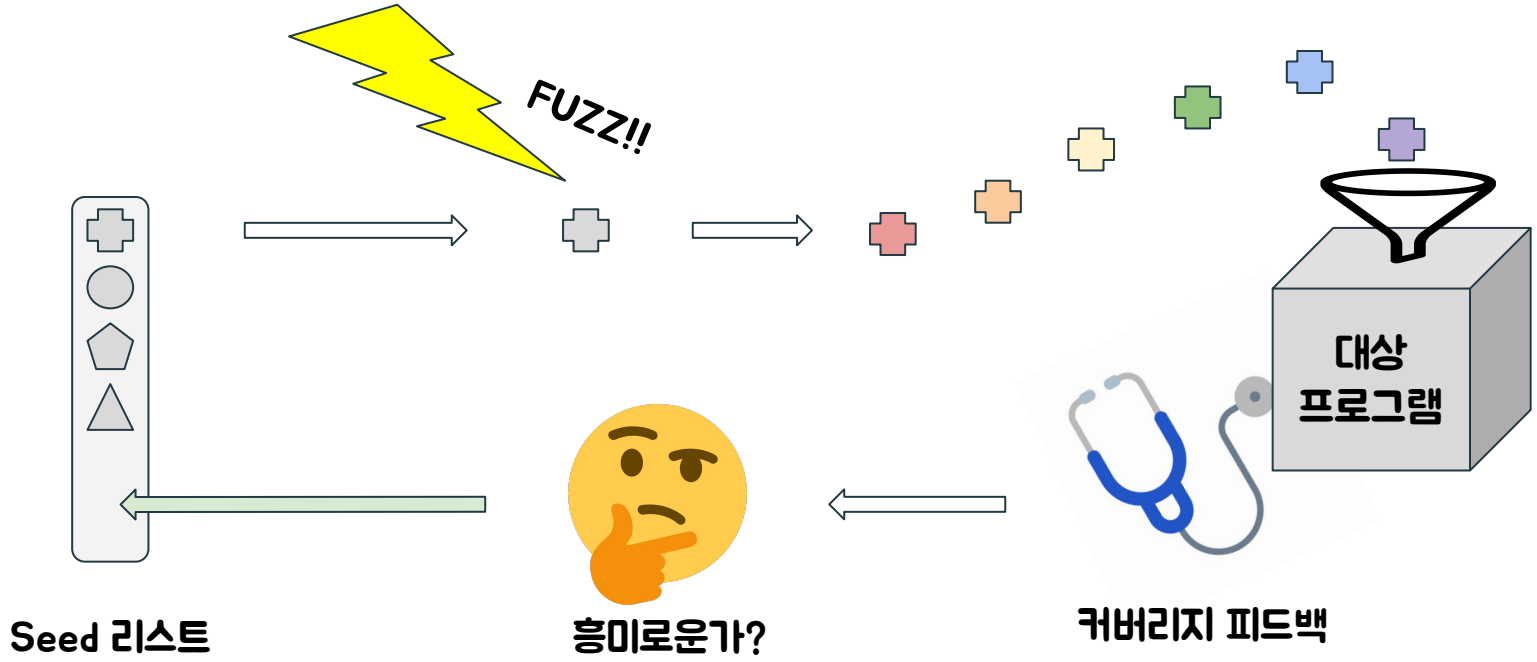
Mutation-based Greybox fuzzing



Mutation-based Greybox fuzzing



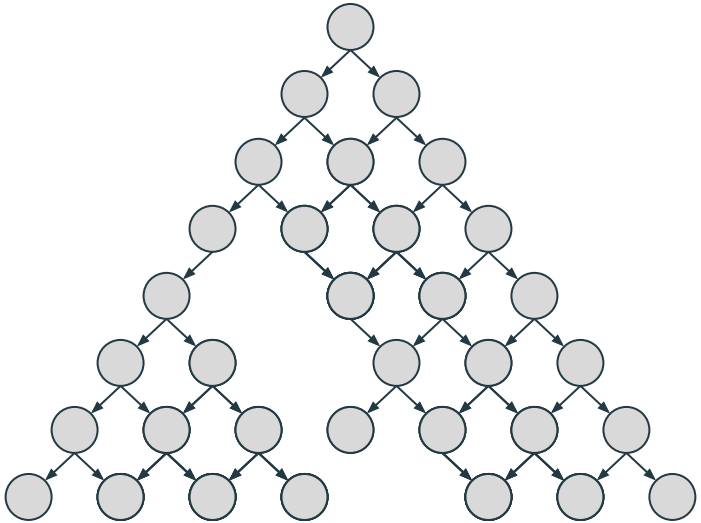
Mutation-based Greybox fuzzing



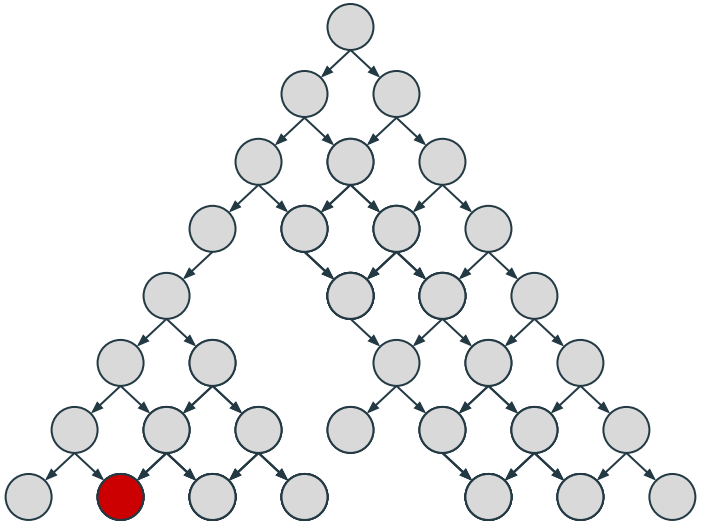
만약에...

- 커밋 검사
- 패치 검사
- 정적분석의 결과 검사

- 커밋 검사
- 패치 검사
- 정적분석의 결과 검사



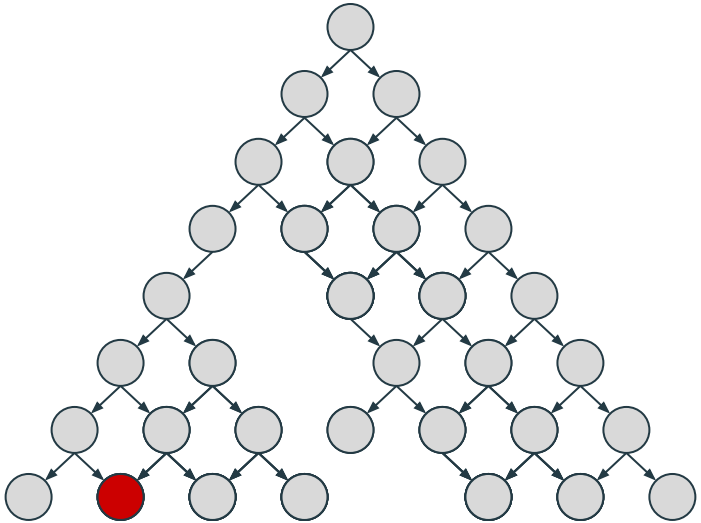
- 커밋 검사
- 패치 검사
- 정적분석의 결과 검사



지향성 퍼징 (Directed Fuzzing)

사용자가 관심 있는 곳에 닿게 하는 Fuzzing

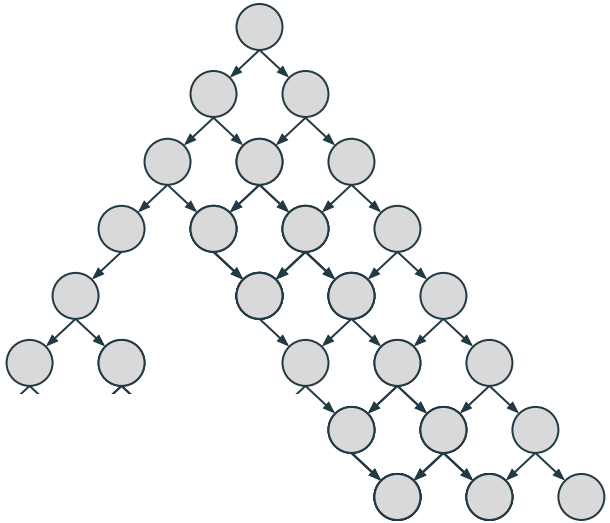
- 커밋 검사
- 패치 검사
- 정적분석의 결과 검사



지향성 퍼징 (Directed Fuzzing)

사용자가 관심 있는 곳에 닿게 하는 Fuzzing

- 커밋 검사
- 패치 검사
- 정적분석의 결과 검사



STAAR
Software disaster research center
Agile & Automated Response & repair

의심되는 프로그램 지점을 검사,

SW 재난 오류를 재현하는 입력 자동 생성 가능!

사례: 다양한 실제 결함들

Binutils 6개, libming 9개

평균적으로 13%의 함수만 실제로 실행된다!

=> 결함에 관여하는 프로그램 지점은 많지 않다

주어진 정보를 활용하여
의미있는 프로그램 지점에만 집중하면
더 빨리 특정 결함을 찾을 수 있을 것이다!

주어진 정보를 활용하여 (의심되는 목표 지점)
의미있는 프로그램 지점에만 집중하면 (관련된 함수들)
더 빨리 특정 결함을 찾을 수 있을 것이다!

새로운 접근

DAFL (Directed AFL): 정적 분석을 활용한 지향성 퍼징 도구

=> 15개의 실제 결함에 대해 성능 평가

새로운 접근

DAFL (Directed AFL): 정적 분석을 활용한 지향성 퍼징 도구

=> 15개의 실제 결함에 대해 성능 평가

기존 도구들보다 약 3배 더 빠르게 목표 결함 발견!!

DAFL

핵심 아이디어: 목표 지점과 연관된 프로그램 지점에 집중

1. 정적 분석을 통해 얻은 데이터 흐름 정보를 바탕으로
   목표 지점에 실제로 영향을 주는 지점들을 파악

2. 해당 정보를 적극 활용하여 퍼징을 수행

DAFL

핵심: 3S => Slicing, Selective instrumentation, Seed scheduling

DAFL

핵심: 3S => Slicing, Selective instrumentation, Seed scheduling

Slicing: 프로그램에서 의미 있는 부분을 선별하기

DAFL

핵심: 3S => Slicing, Selective instrumentation, Seed scheduling

Slicing: 프로그램에서 의미 있는 부분을 선별하기

Selective Inst.: 선별된 프로그램 지점들로부터만 커버리지 피드백을 받기

DAFL

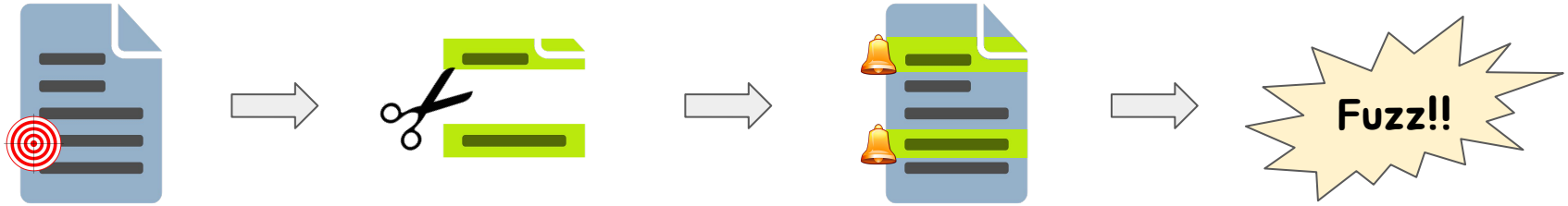
핵심: 3S => Slicing, Selective instrumentation, Seed scheduling

Slicing: 프로그램에서 의미 있는 부분을 선별하기

Selective Inst.: 선별된 프로그램 지점들로부터만 커버리지 피드백을 받기

Seed scheduling: 퍼징 과정에서 특정 시드에 우선순위 부여하기

DAFL



1. Slicing

2. Selective Instrumentation

3. Seed Scheduling

Slicing

실제로 실행되는, 즉 의미있는 함수는 전체 프로그램 중 일부에 불과하다.

=> 의미 있는 함수를 선별해야 함

Slicing

실제로 실행되는, 즉 의미있는 함수는 전체 프로그램 중 일부에 불과하다.

=> 의미 있는 함수를 선별해야 함

정의-사용(Def-Use) 관계를 고려하기

=> 목표 지점과 정의-사용 관계로 이어진 라인들을 추적

Slicing

실제로 실행되는, 즉 의미있는 함수는 전체 프로그램 중 일부에 불과하다.

=> 의미 있는 함수를 선별해야 함

정의-사용(Def-Use) 관계를 고려하기

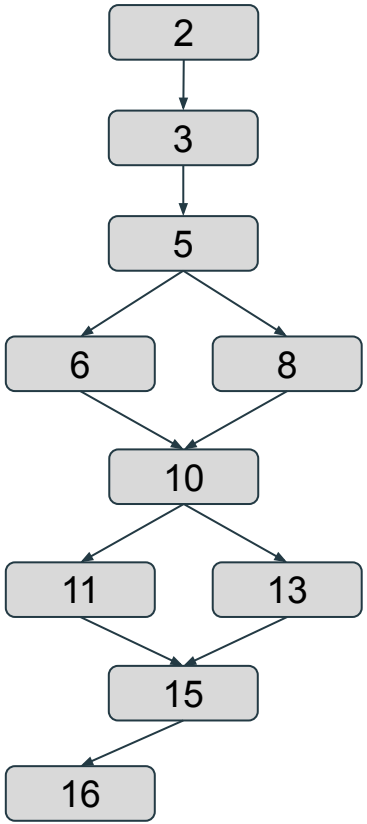
=> 목표 지점과 정의-사용 관계로 이어진 라인들을 추적

실행 흐름 그래프를 사용하는 것보다 더 적은 프로그램 지점을 고려할 수 있다!

```
 1:  int a; int x;
 2:  a = input();
 3:  x = input();
 4:
 5:  if( … )
 6:      x++;
 7:  else
 8:      x--;
 9:
10:  if(a<6)
11:      a++;
12:  else
13:      a--;
14:
15:  if(x>3)
16:      foo(x);
```

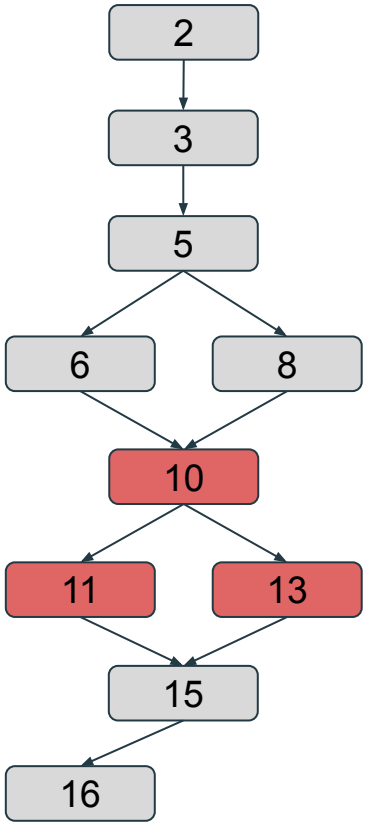
```
1:   int a; int x;
2:   a = input();
3:   x = input();
4:
5:   if( … )
6:       x++;
7:   else
8:       x--;
9:
10:  if(a<6)
11:      a++;
12:  else
13:      a--;
14:
15:  if(x>3)
16:      foo(x);
```

실행 흐름 그래프



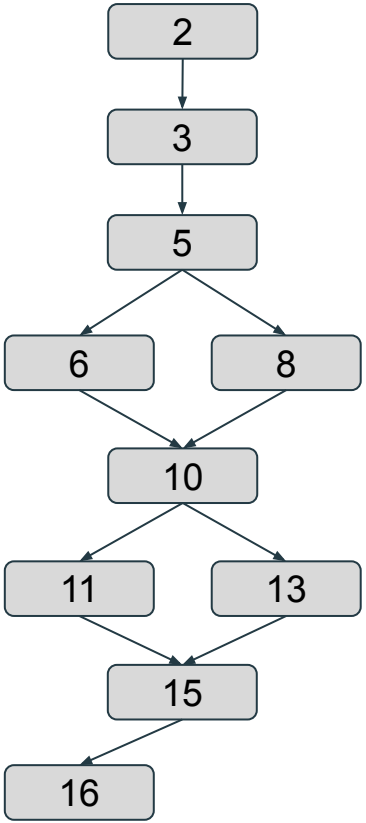
```
1:   int a; int x;
2:   a = input();
3:   x = input();
4:
5:   if( … )
6:       x++;
7:   else
8:       x--;
9:
10:  if(a<6)
11:      a++;
12:  else
13:      a--;
14:
15:  if(x>3)
16:      foo(x);
```

실행 흐름 그래프



```
1:   int a; int x;
2:   a = input();
3:   x = input();
4:
5:   if( … )
6:       x++;
7:   else
8:       x--;
9:
10:  if(a<6)
11:      a++;
12:  else
13:      a--;
14:
15:  if(x>3)
16:      foo(x);
```

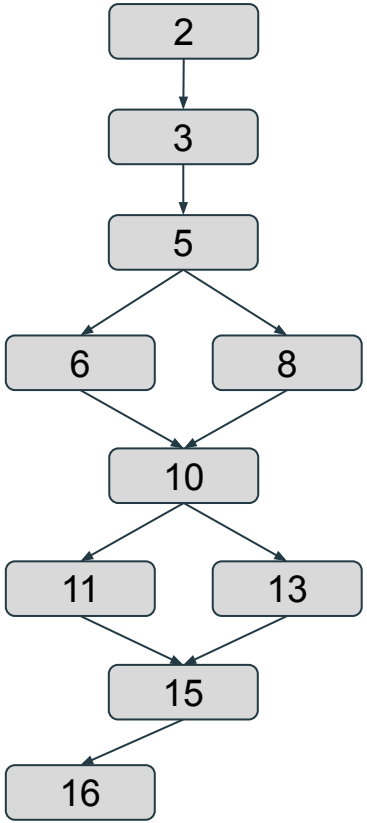
실행 흐름 그래프



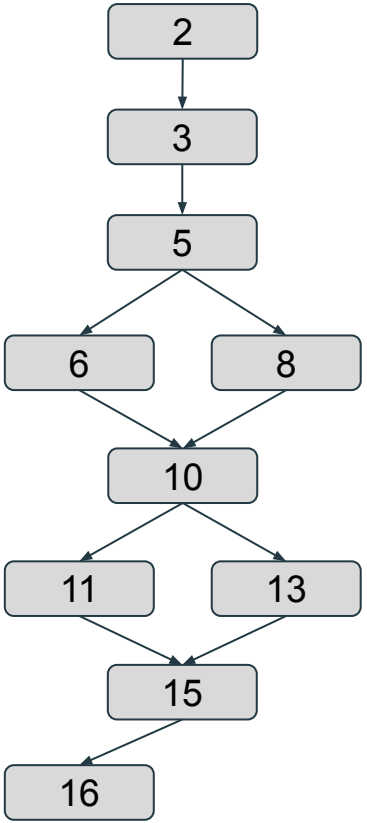
정의-사용 그래프

```
1:   int a; int x;
2:   a = input();
3:   x = input();
4:
5:   if( … )
6:       x++;
7:   else
8:       x--;
9:
10:  if(a<6)
11:      a++;
12:  else
13:      a--;
14:
15:  if(x>3)
16:      foo(x);
```

실행 흐름 그래프

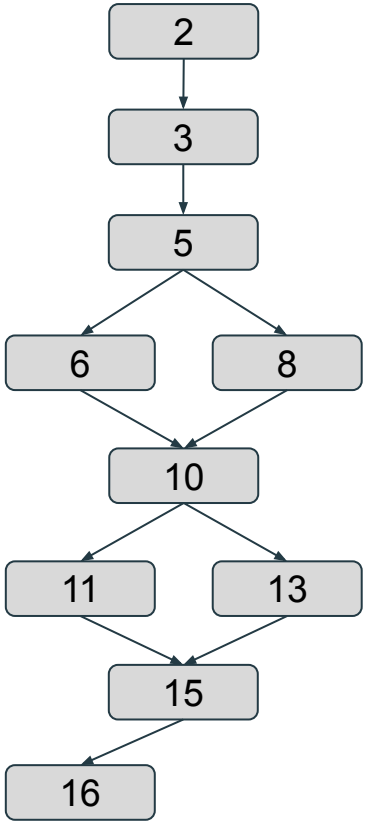


정의-사용 그래프

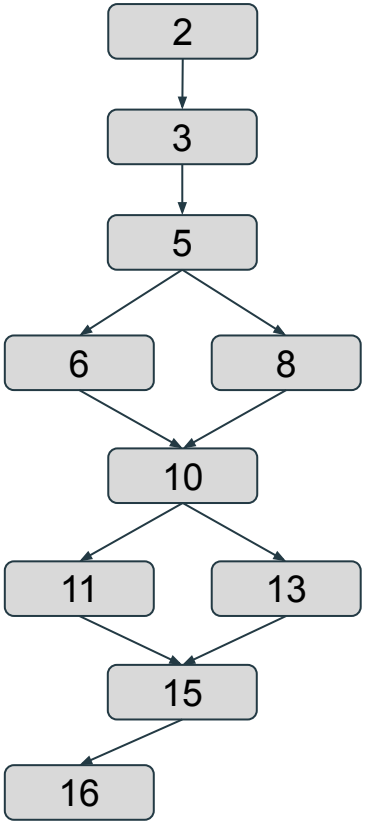


```
1:   int a; int x;
2:   a = input();
3:   x = input();
4:
5:   if( … )
6:       x++;
7:   else
8:       x--;
9:
10:  if(a<6)
11:      a++;
12:  else
13:      a--;
14:
15:  if(x>3)
16:      foo(x);
```

실행 흐름 그래프

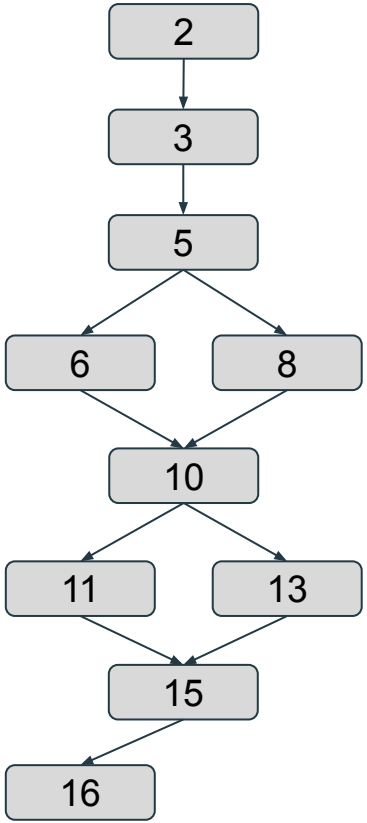


정의-사용 그래프

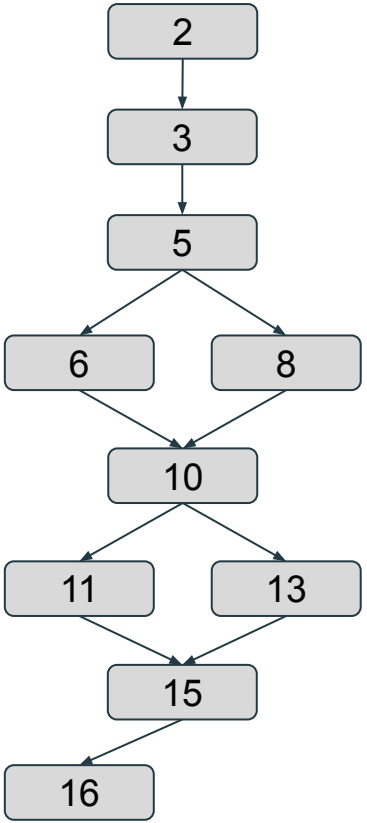


```
1:   int a; int x;
2:   a = input();
3:   x = input();
4:
5:   if( … )
6:       x++;
7:   else
8:       x--;
9:
10:  if(a<6)
11:      a++;
12:  else
13:      a--;
14:
15:  if(x>3)
16:      foo(x);
```

실행 흐름 그래프

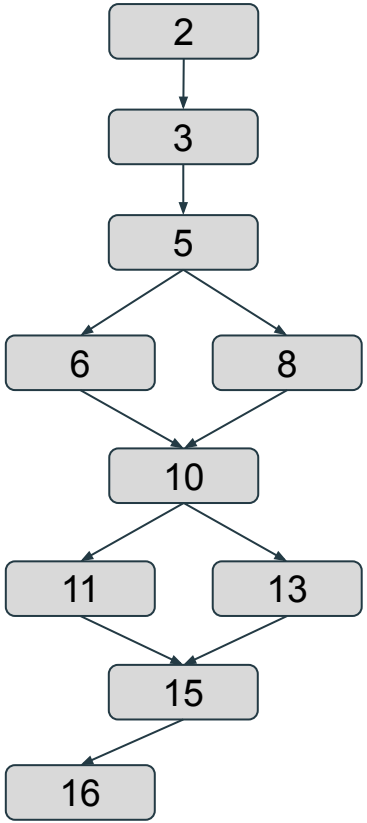


정의-사용 그래프

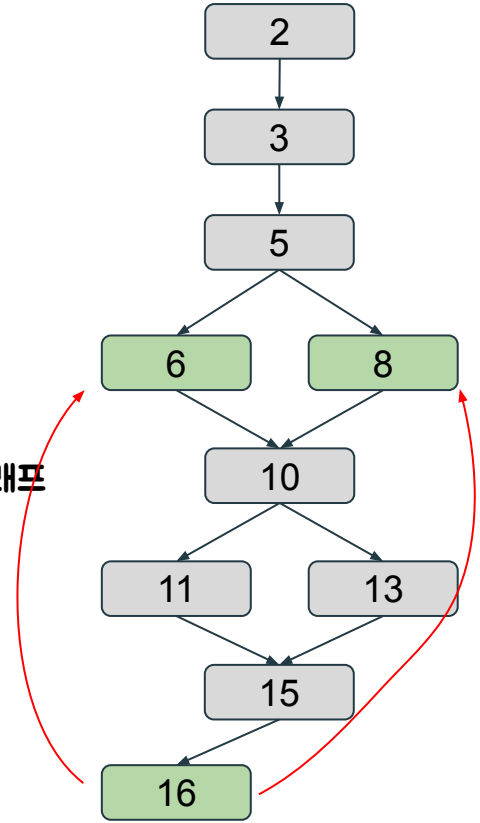


```
1:   int a; int x;
2:   a = input();
3:   x = input();
4:
5:   if( … )
6:       x++;
7:   else
8:       x--;
9:
10:  if(a<6)
11:      a++;
12:  else
13:      a--;
14:
15:  if(x>3)
16:      foo(x);
```

실행 흐름 그래프

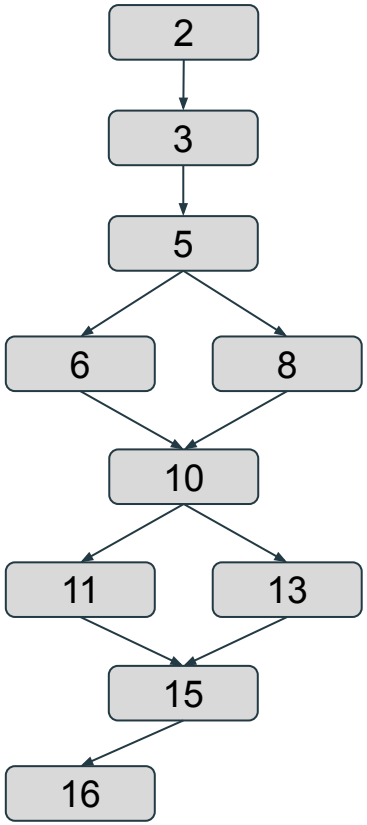


정의-사용 그래프

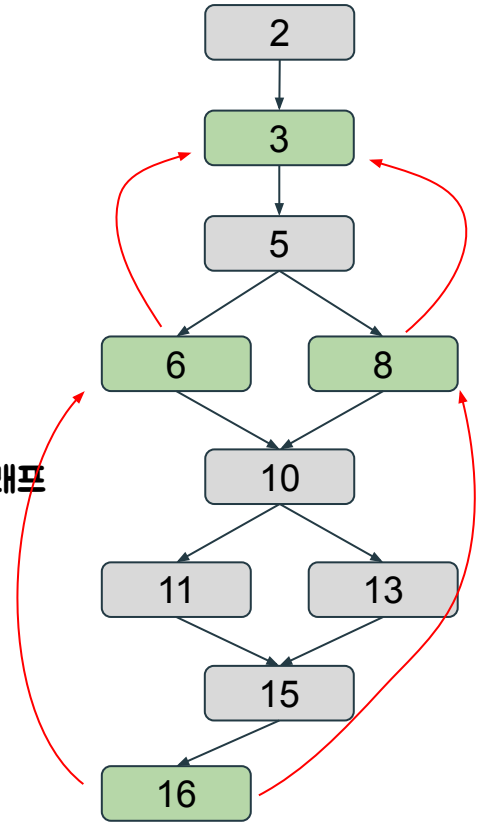


```
1:   int a; int x;
2:   a = input();
3:   x = input();
4:
5:   if( … )
6:       x++;
7:   else
8:       x--;
9:
10:  if(a<6)
11:      a++;
12:  else
13:      a--;
14:
15:  if(x>3)
16:      foo(x);
```

실행 흐름 그래프

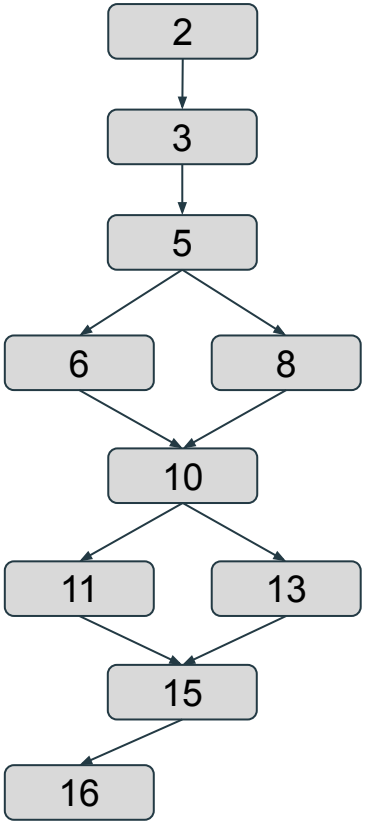


정의-사용 그래프

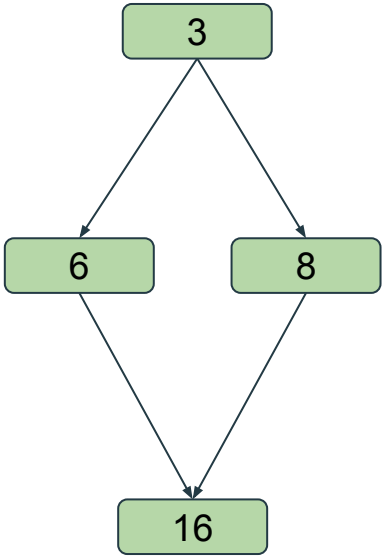


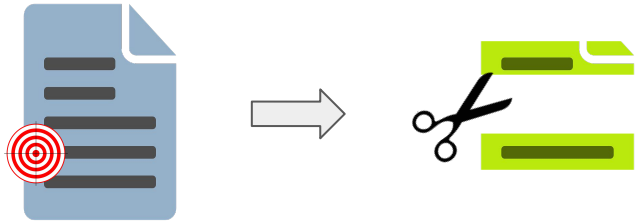
```
1:   int a; int x;
2:   a = input();
3:   x = input();
4:
5:   if( … )
6:       x++;
7:   else
8:       x--;
9:
10:  if(a<6)
11:      a++;
12:  else
13:      a--;
14:
15:  if(x>3)
16:      foo(x);
```

실행 흐름 그래프



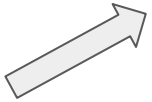
정의-사용 그래프



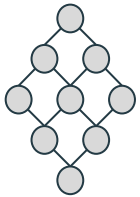


Slicing

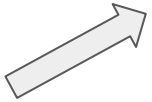
Slicing



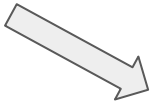
정의-사용 그래프



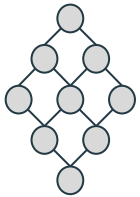
Slicing



정의-사용 그래프



함수 목록



Selective Inst. & Seed Scheduling

Selective Inst. & Seed Scheduling

Selective Inst.: 선별된 프로그램 지점들로부터만 커버리지 피드백을 받기

Selective Inst. & Seed Scheduling

Selective Inst.: 선별된 프로그램 지점들로부터만 커버리지 피드백을 받기

=> 함수 목록 사용

Selective Inst. & Seed Scheduling

Selective Inst.: 선별된 프로그램 지점들로부터만 커버리지 피드백을 받기

=> 함수 목록 사용

Seed scheduling: 퍼징 과정에서 특정 시드에 우선순위 부여하기

Selective Inst. & Seed Scheduling

Selective Inst.: 선별된 프로그램 지점들로부터만 커버리지 피드백을 받기

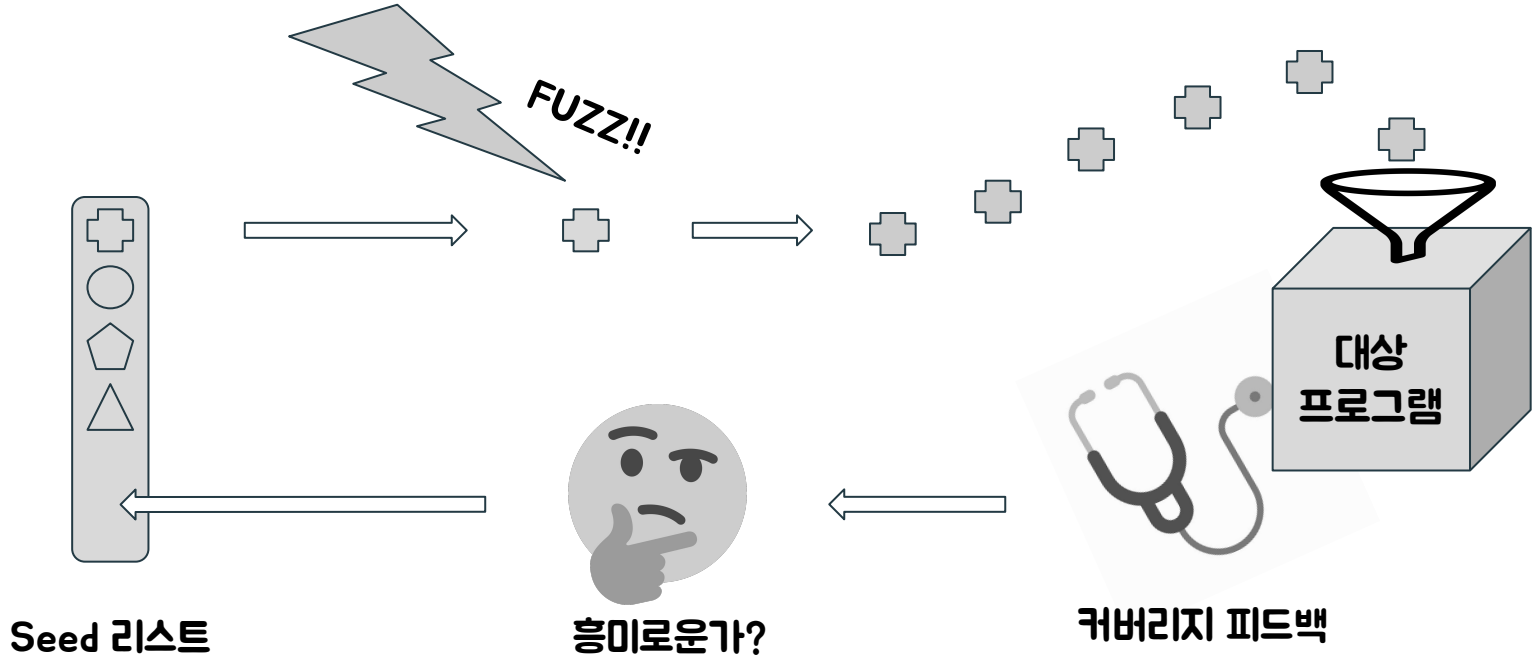
=> 함수 목록 사용

Seed scheduling: 퍼징 과정에서 특정 시드에 우선순위 부여하기

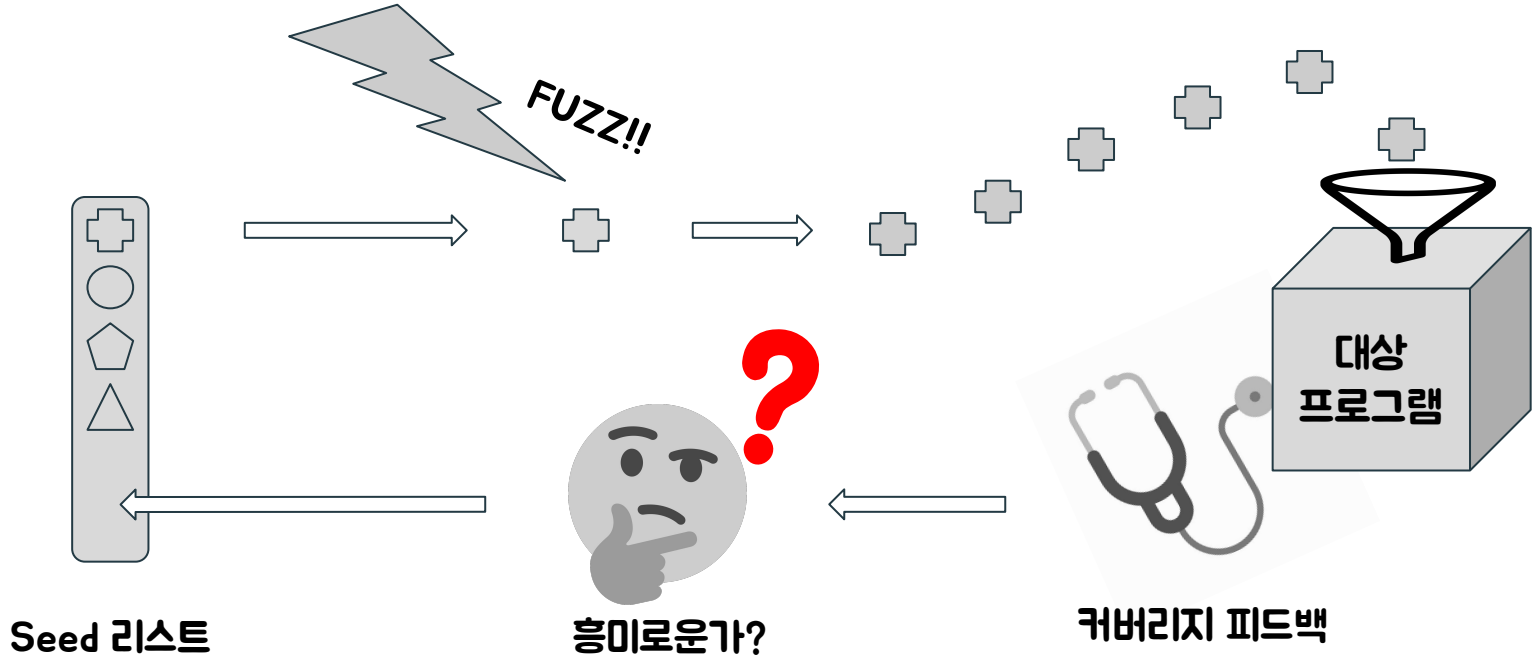
=> 정의-사용 그래프 사용

Selective Instrumentation

Selective Instrumentation



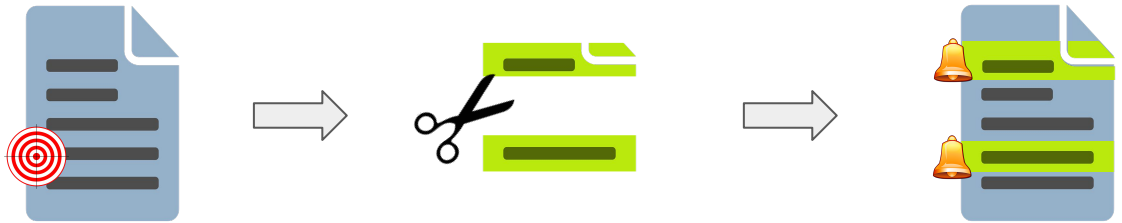
Selective Instrumentation



Selective Instrumentation

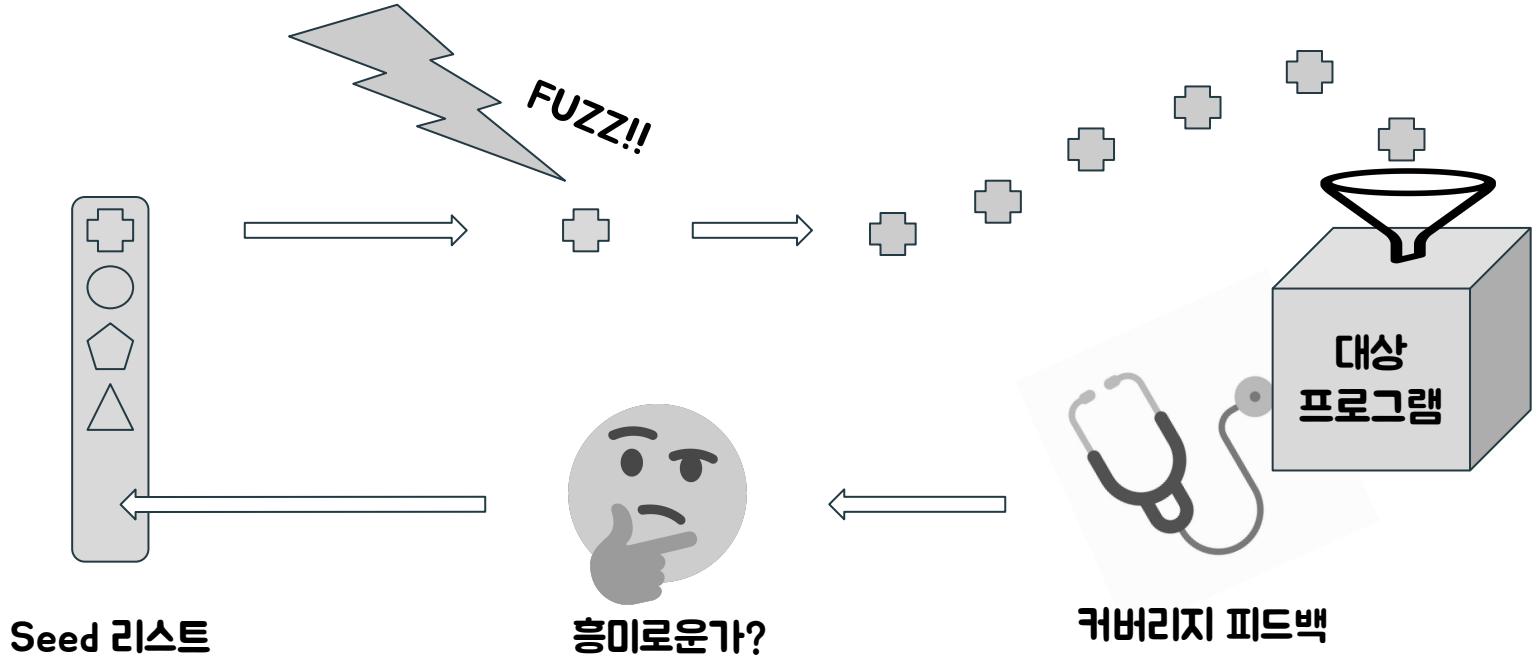
관련된 함수 목에 방울 달기:

=> 원하는 지점의 실행 정보만 기록 되도록!

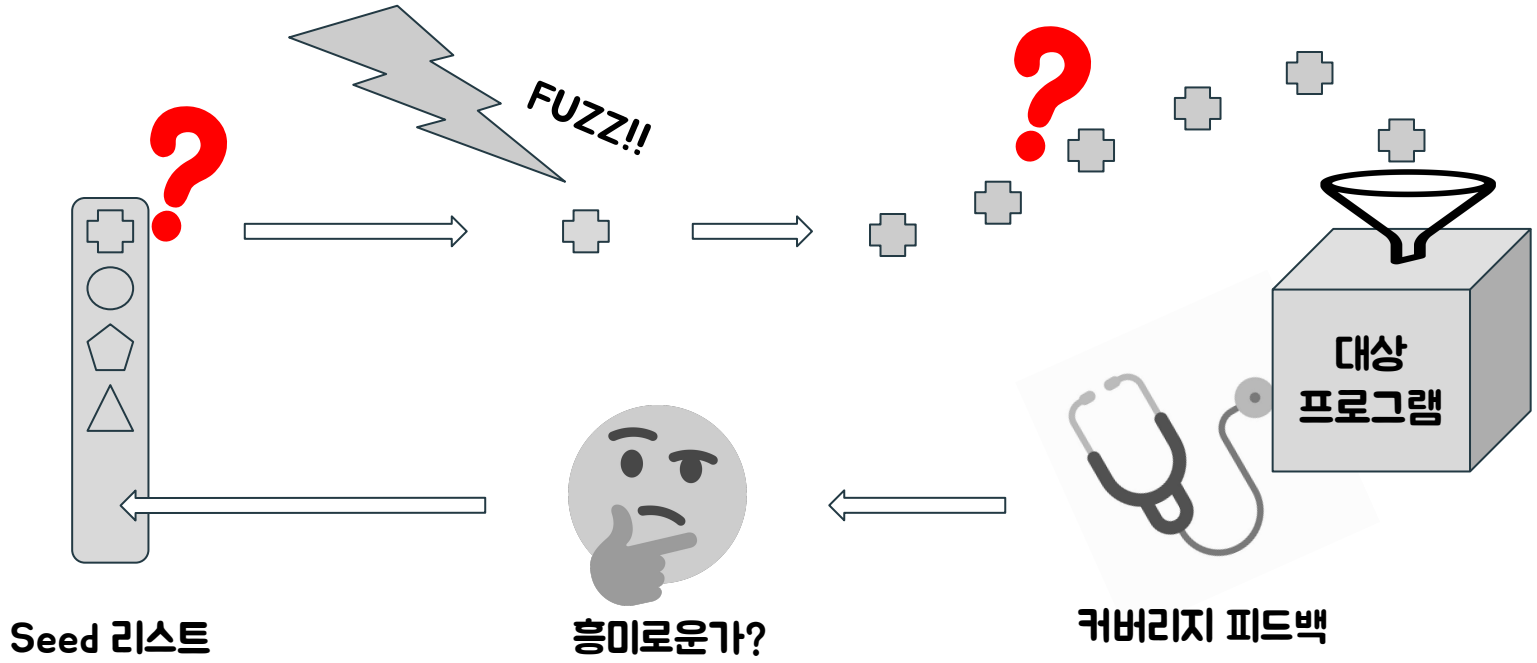


Seed Scheduling

Seed Scheduling



Seed Scheduling



Seed Scheduling

두 가지 선택:

- Seed 중 무엇을 먼저 사용할 지 선택
- 한번 뽑았을 때, 얼마나 오래 사용할 지 선택

Seed Scheduling

두 가지 선택:

- Seed 중 무엇을 먼저 사용할 지 선택
- 한번 뽑았을 때, 얼마나 오래 사용할 지 선택

=> 두 선택 모두 Proximity Score를 고려!

Seed Scheduling

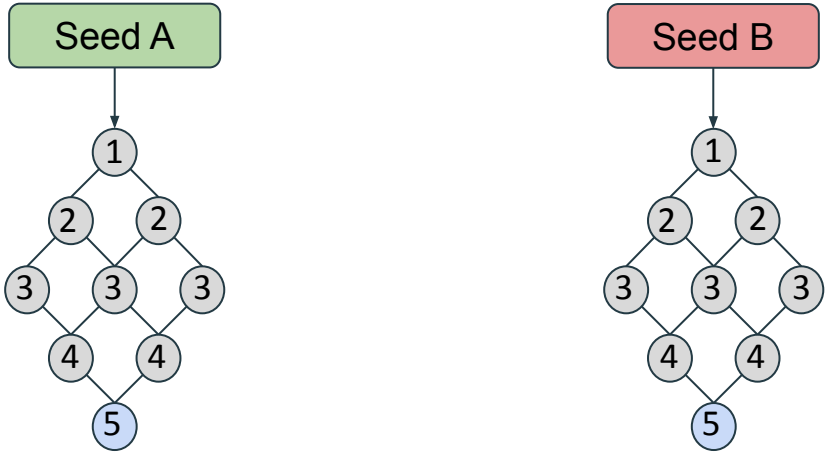
두 가지 선택:

- Seed 중 무엇을 먼저 사용할 지 선택
- 한번 뽑았을 때, 얼마나 오래 사용할 지 선택

=> 두 선택 모두 Proximity Score를 고려!

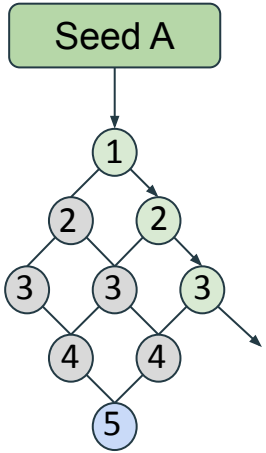
Proximity Score란? 정의-사용 그래프를 기반으로 계산된 각 Seed별 점수

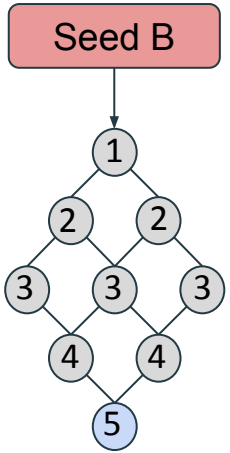
Seed Scheduling



Prox. Score:

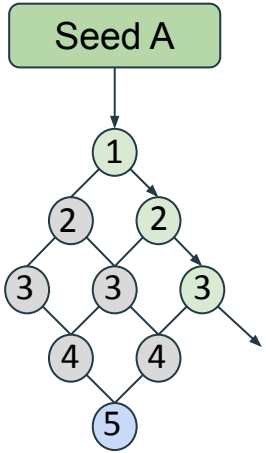
Seed Scheduling

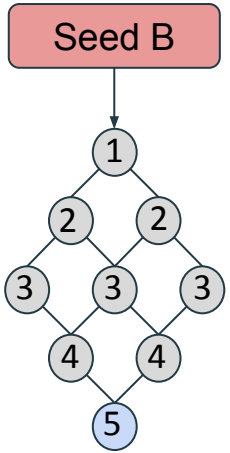




Prox. Score:

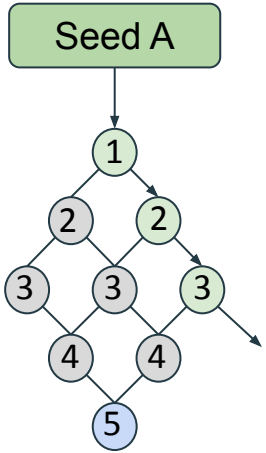
Seed Scheduling

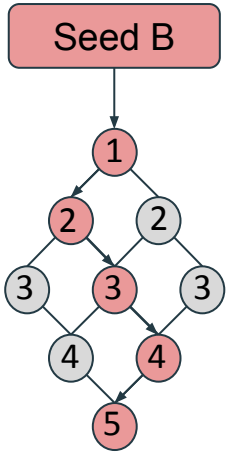




Prox. Score: 6점

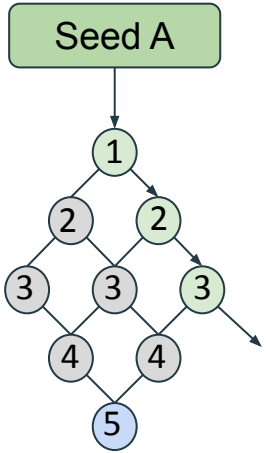
Seed Scheduling

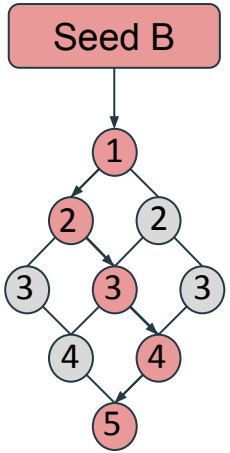




Prox. Score: 6점

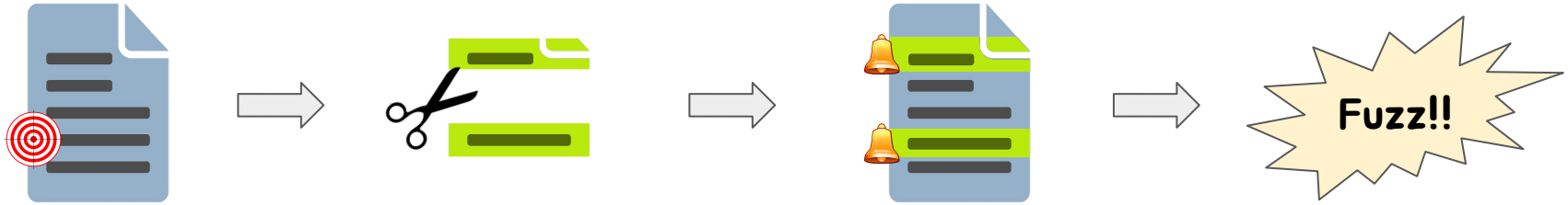
Seed Scheduling





Prox. Score: 6점 15점

DAFL 정리



1. Slicing

2. Selective Instrumentation

3. Seed Scheduling

실험

실험 대상: cxxfilt, swftophp 프로그램의 15개 결함

실험

실험 대상: cxxfilt, swftophp 프로그램의 15개 결함

비교 대상:

- AFL: 무지향성 퍼저
- AFLGo[1]: AFL 기반 지향성 퍼저

[1] Böhme et al., Directed Greybox Fuzzing, CCS '17

실험

실험 대상: cxxfilt, swftophp 프로그램의 15개 결함

비교 대상:

- AFL: 무지향성 퍼저
- AFLGo[1]: AFL 기반 지향성 퍼저

실험 방법: 40번 반복 실험 후 목표 결함을 발견한 시간의 중앙값 비교

[1] Böhme et al., Directed Greybox Fuzzing, CCS '17

실험 결과

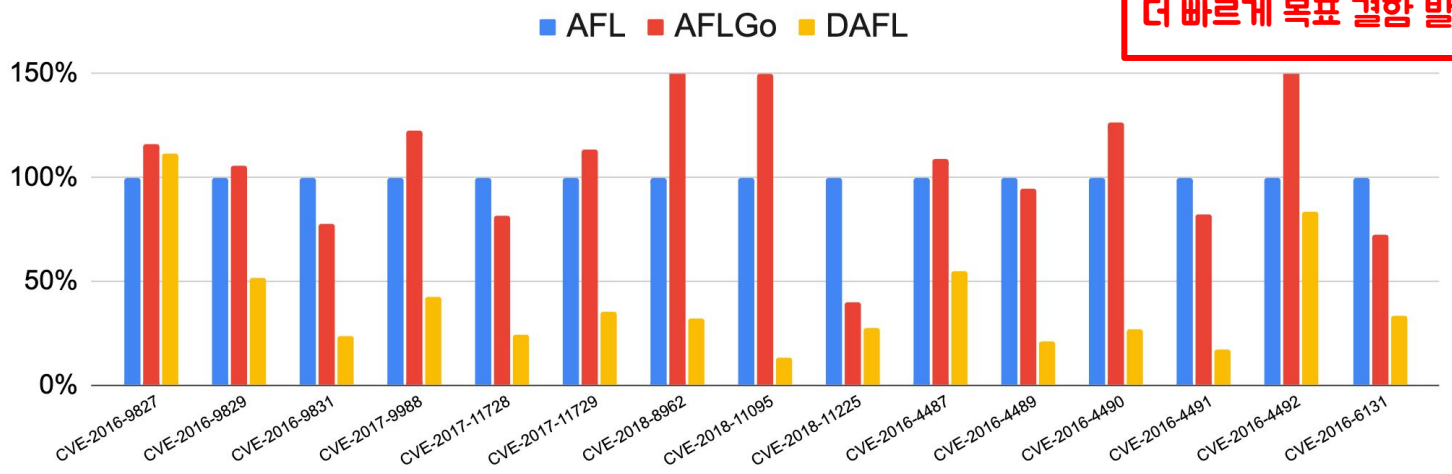
AFL의 결과를 기준으로 상대적 성능 비교

실험 결과

AFL의 결과를 기준으로 상대적 성능 비교

AFL보다 3.4배
AFLGo보다 3.6배

더 빠르게 목표 결함 발견!!



발표 요약

- 지향성 퍼징이란?
  - 주어진 목표지점에 도달하는 입력을 생성하는 퍼징
- 효과적인 지향성 퍼징을 하는 법
  - 더 적은 프로그램 지점에 집중하기
- 더 적은 프로그램 지점들을 구하는 방법
  - 데이터 흐름 분석을 통해 목표 지점과 연관된 함수들을 파악하기

향후 계획

1. 더 최신 지향성 퍼저들과 성능 비교

2. 지향성 퍼징 연구의 표준이 될 기준 수립

감사합니다

QnA