

데이터 의존 관계를 이용한 지향성 퍼징

발표: 2023.07.05 김태은
연구: 김태은, 최재승, 허기홍, 차상길

정적 분석

프로그램의 행동을 어림잡아 어떤 오류가 발생할지 예측하는 기술

정적 분석

프로그램의 행동을 어림잡아 어떤 오류가 발생할지 예측하는 기술

한계: 어림잡는 특성으로 인해 오탐이 필연적으로 동반된다

정적 분석

프로그램의 행동을 어림잡아 어떤 오류가 발생할지 예측하는 기술

한계: 어림잡는 특성으로 인해 오탐이 필연적으로 동반된다

Q. 어떻게 하면 확실한 알람을 알아낼 수 있을까?

정적 분석

프로그램의 행동을 어림잡아 어떤 오류가 발생할지 예측하는 기술

한계: 어림잡는 특성으로 인해 오탐이 필연적으로 동반된다

Q. 어떻게 하면 확실한 알람을 알아낼 수 있을까?

A. 데이터 의존 관계를 이용한 지향성 퍼징

퍼징(Fuzzing)이란?

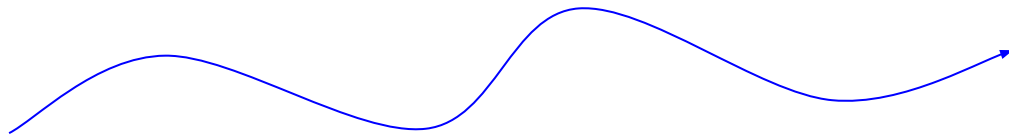
퍼징, 혹은 퍼즈 테스트:

자동으로 생성한 입력을 통해 프로그램을 테스트하는 기법

퍼징(Fuzzing)이란?

퍼징, 혹은 퍼즈 테스트:

자동으로 생성한 입력을 통해 프로그램을 테스트하는 기법

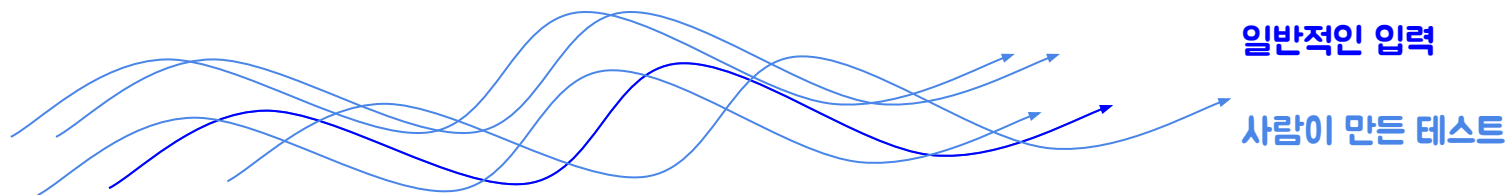


일반적인 입력

퍼징(Fuzzing)이란?

퍼징, 혹은 퍼즈 테스트:

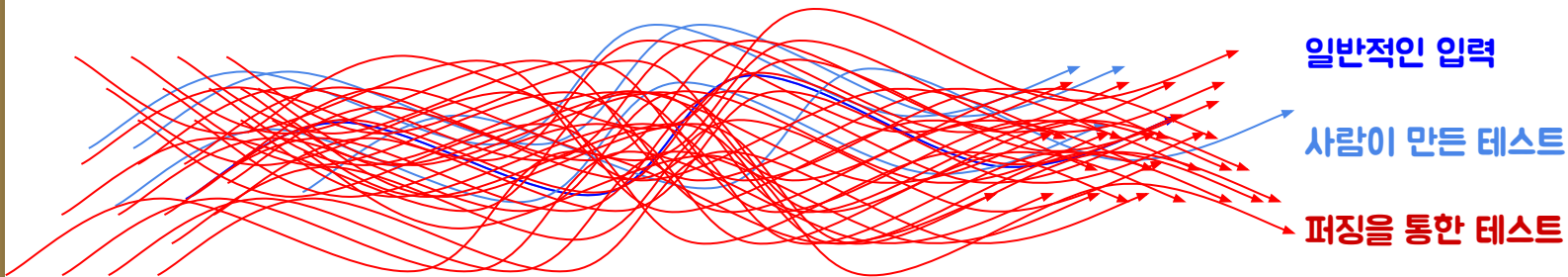
자동으로 생성한 입력을 통해 프로그램을 테스트하는 기법



퍼징(Fuzzing)이란?

퍼징, 혹은 퍼즈 테스트:

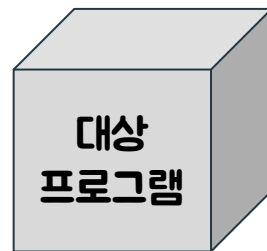
자동으로 생성한 입력을 통해 프로그램을 테스트하는 기법



변형 기반 반투명 퍼징 (Mutation-based Greybox fuzzing)



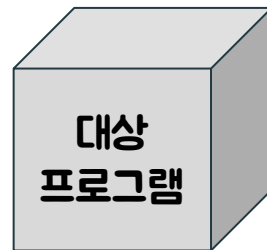
Seed 리스트



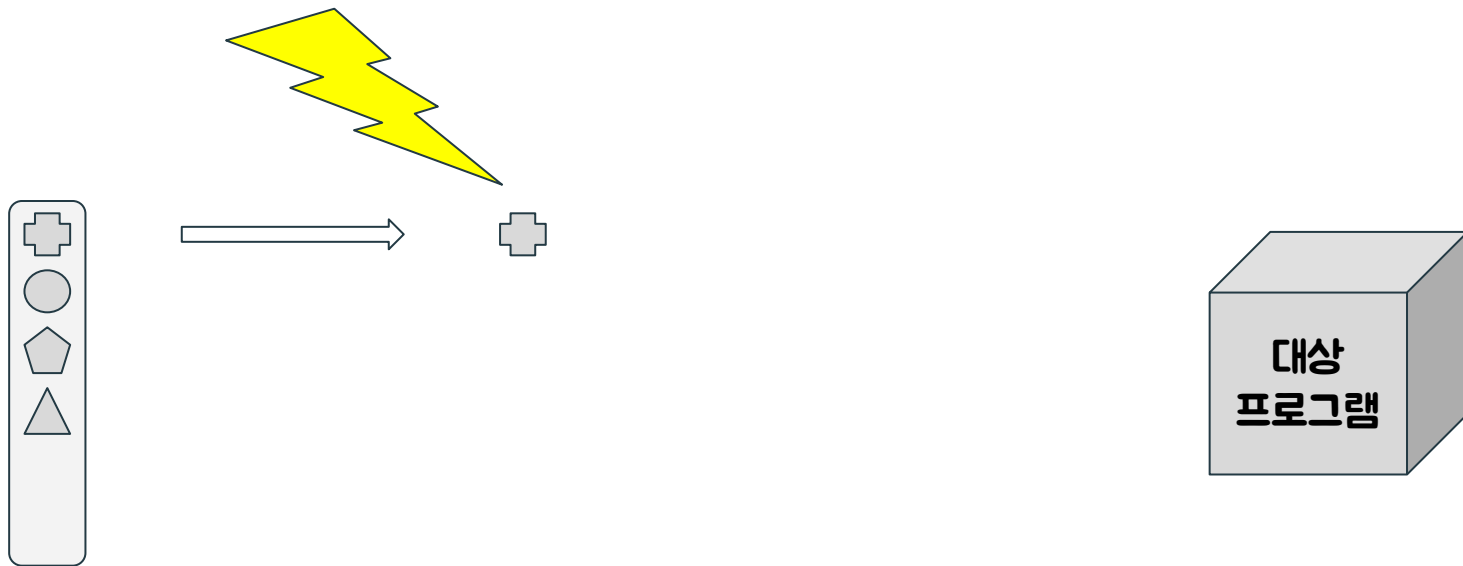
변형 기반 반투명 퍼징 (Mutation-based Greybox fuzzing)



Seed 리스트

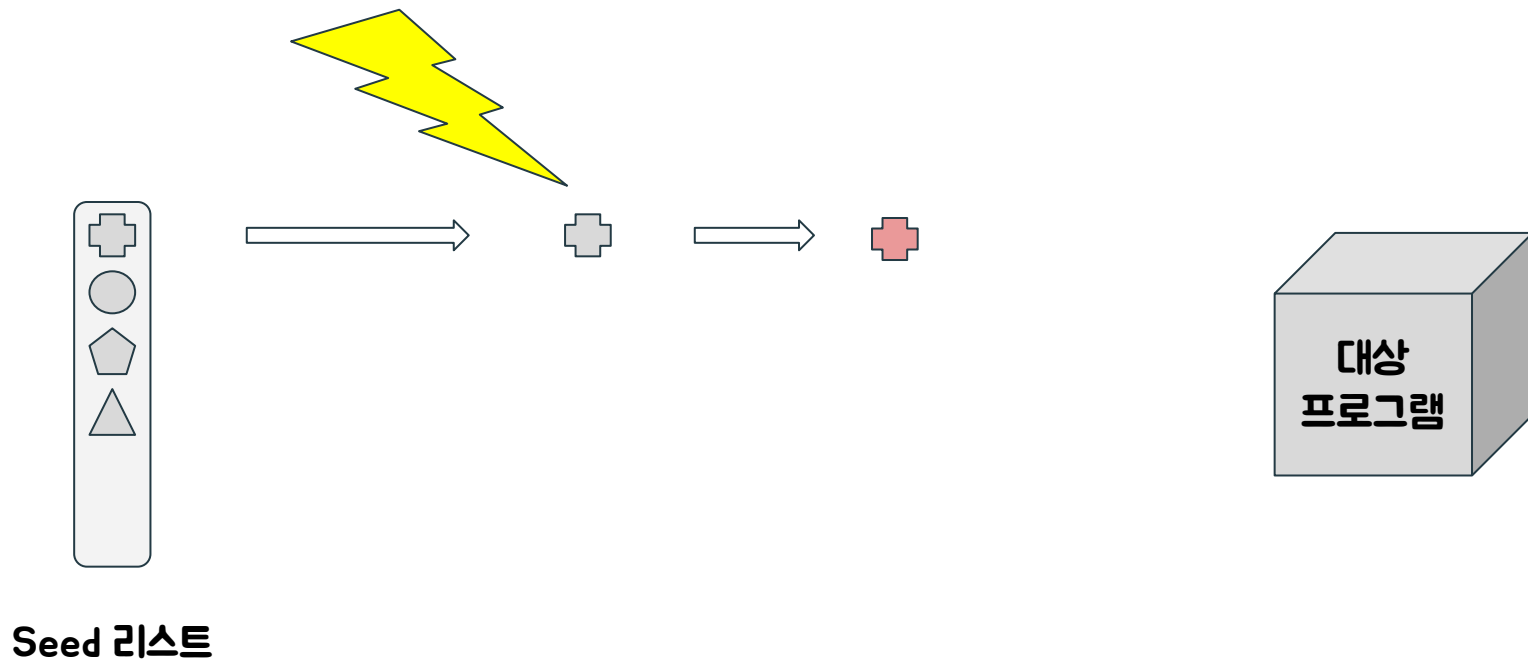


변형 기반 반투명 퍼징 (Mutation-based Greybox fuzzing)

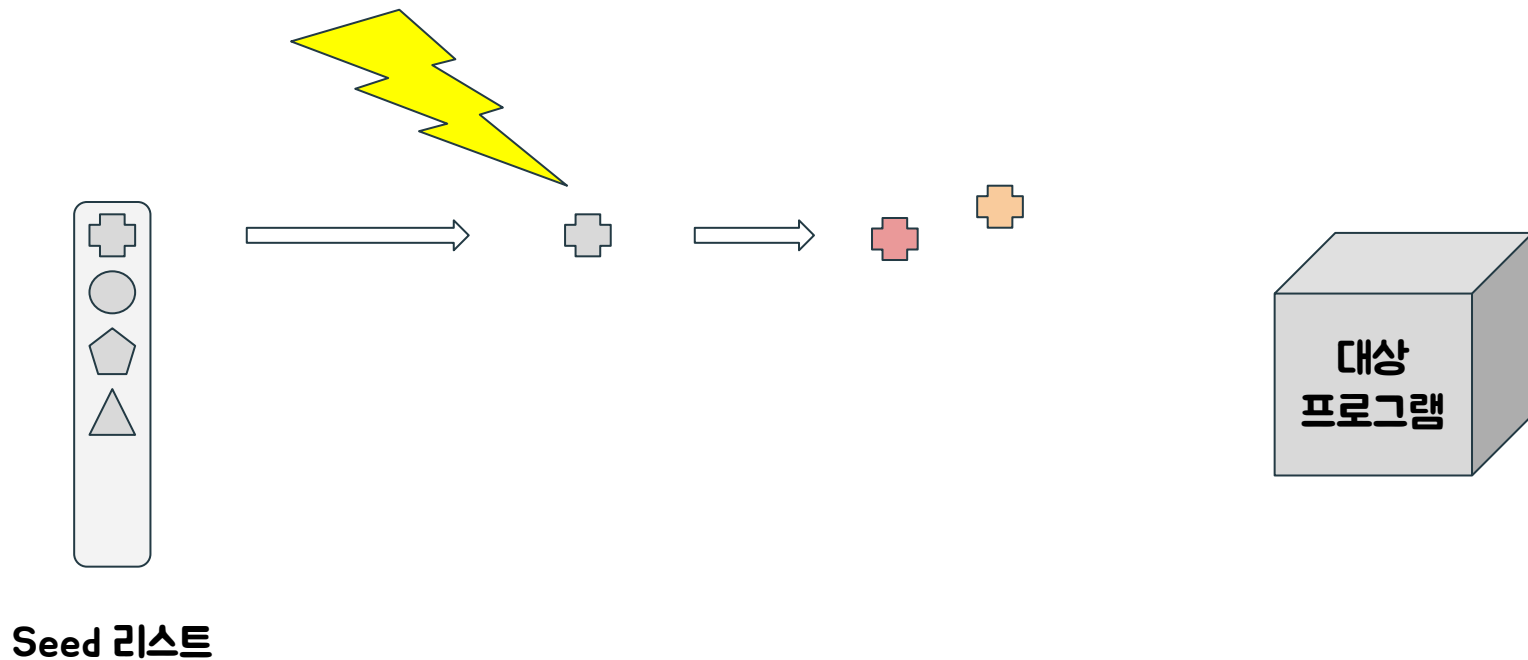


Seed 리스트

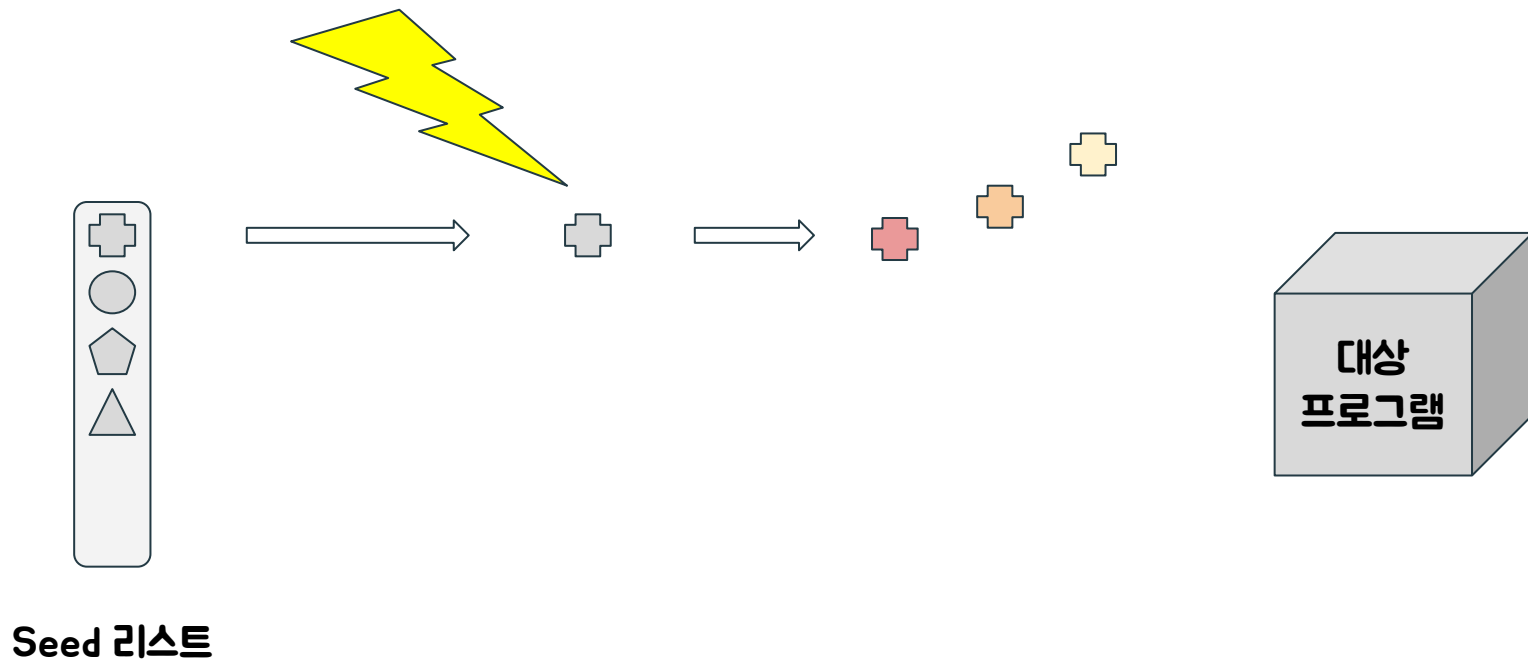
변형 기반 반투명 퍼징 (Mutation-based Greybox fuzzing)



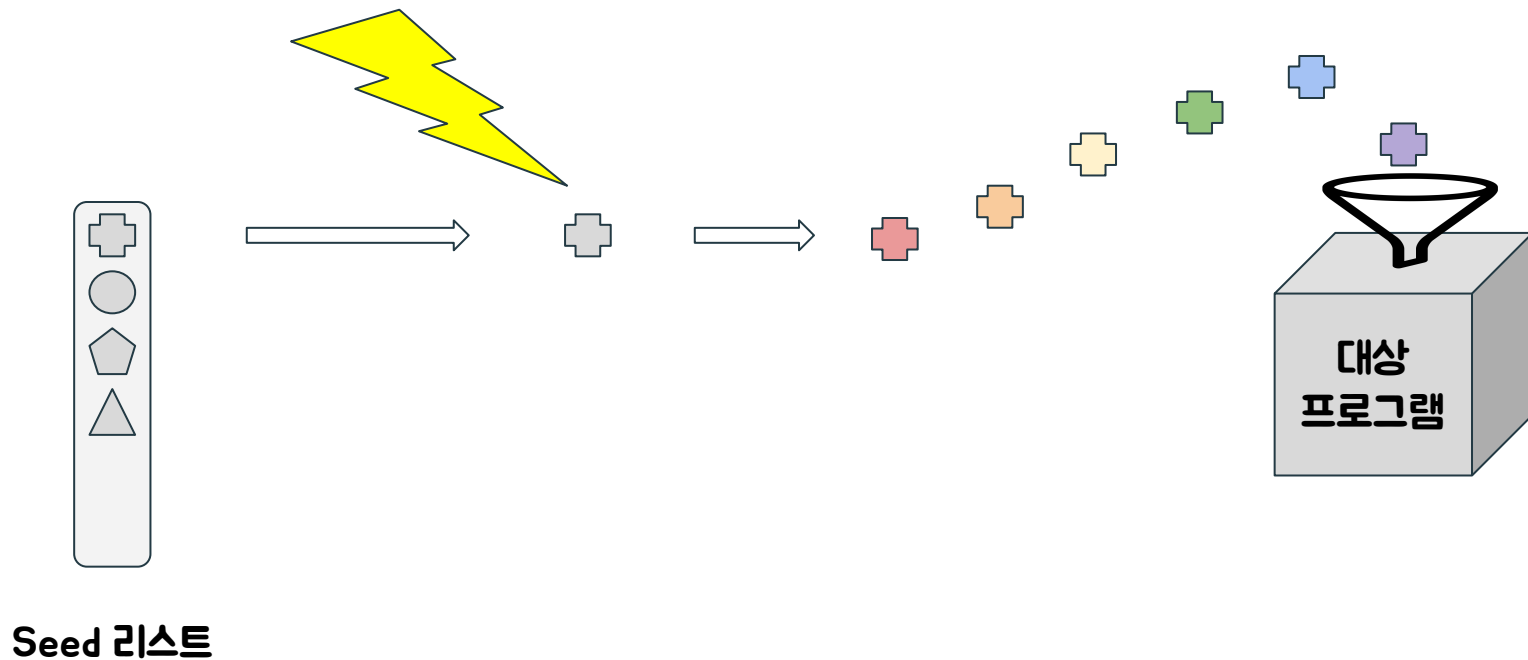
변형 기반 반투명 퍼징 (Mutation-based Greybox fuzzing)



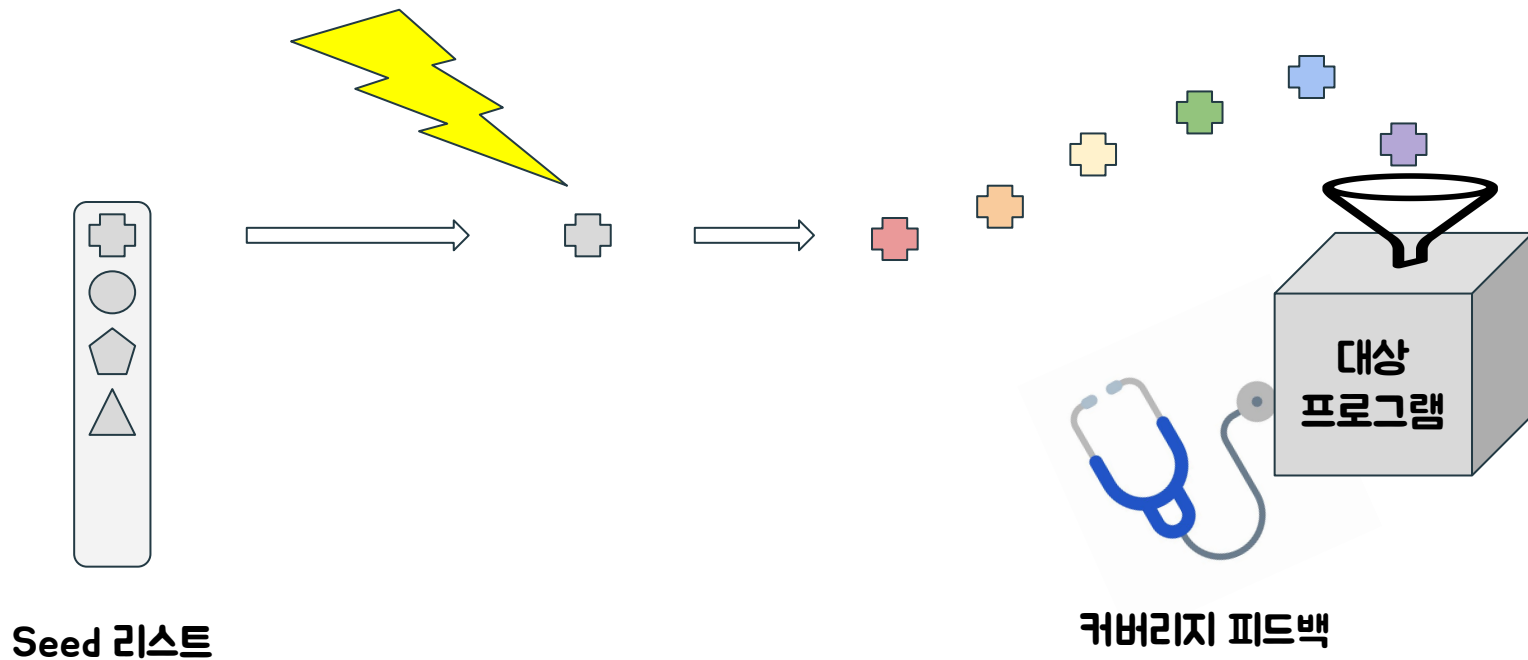
변형 기반 반투명 퍼징 (Mutation-based Greybox fuzzing)



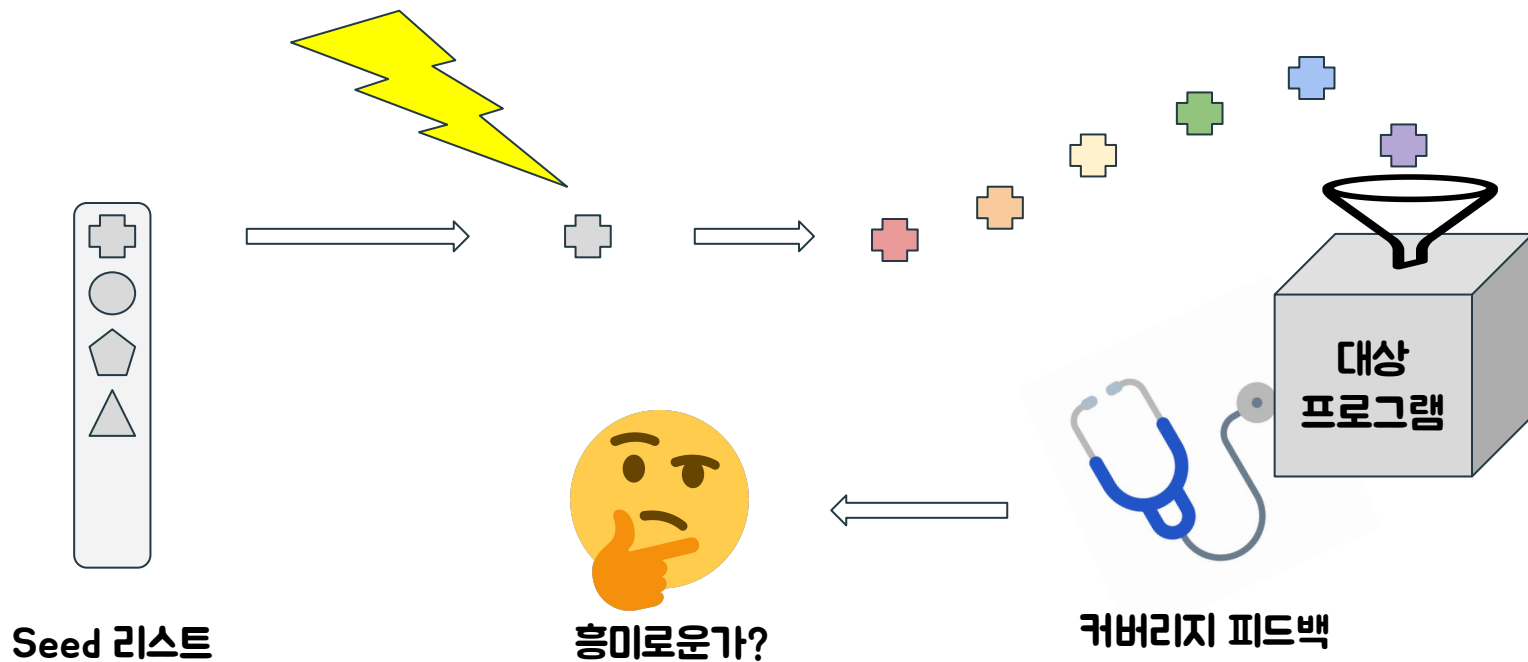
변형 기반 반투명 퍼징 (Mutation-based Greybox fuzzing)



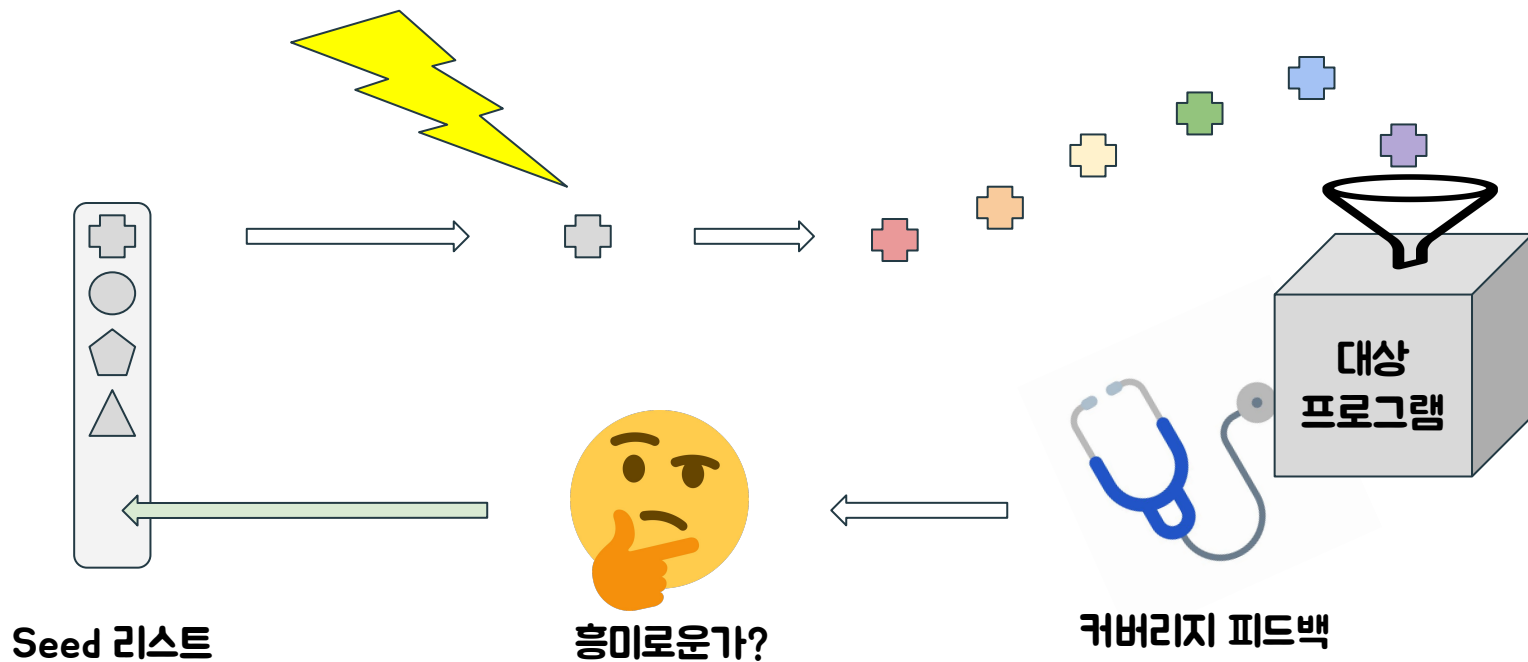
변형 기반 반투명 퍼징 (Mutation-based Greybox fuzzing)



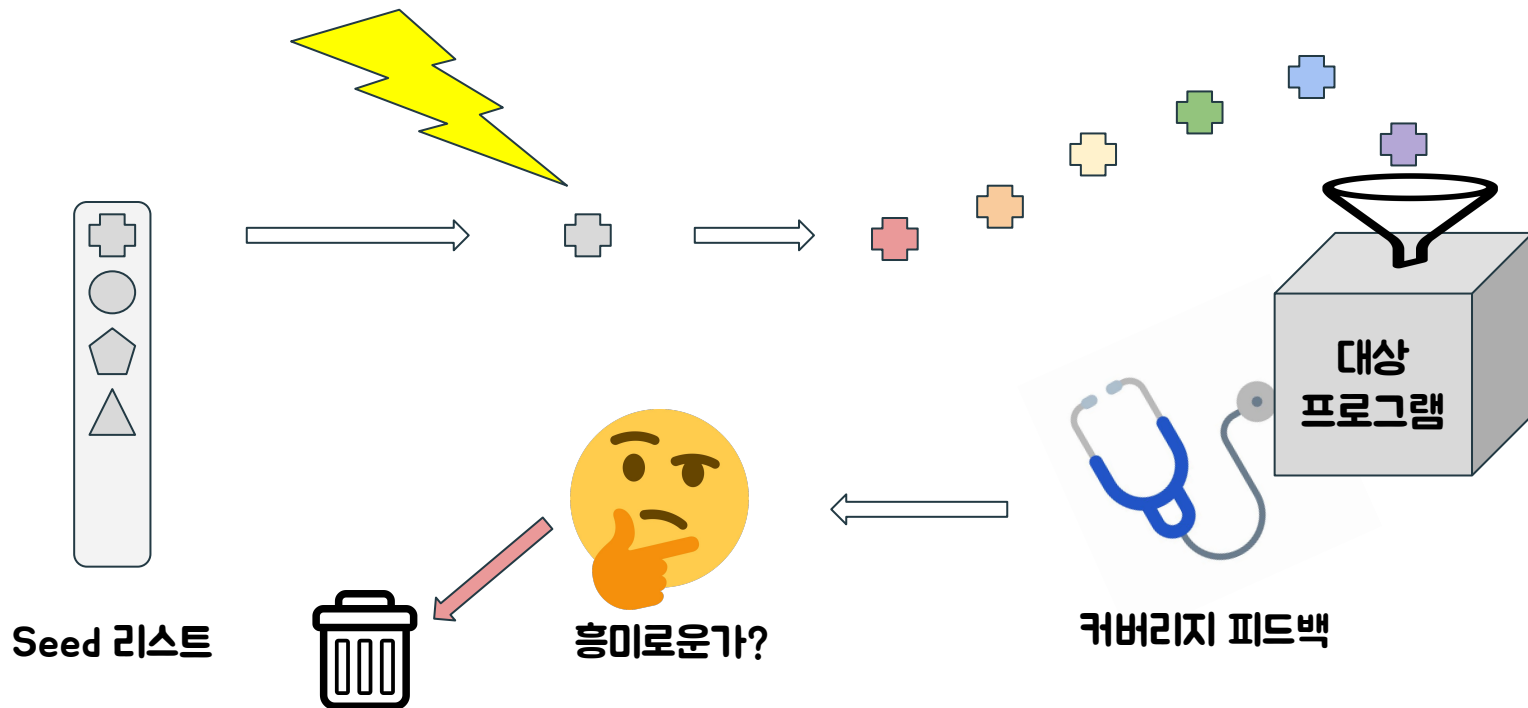
변형 기반 반투명 퍼징 (Mutation-based Greybox fuzzing)



변형 기반 반투명 퍼징 (Mutation-based Greybox fuzzing)



변형 기반 반투명 퍼징 (Mutation-based Greybox fuzzing)



OSS-Fuzz: 지구방위대 구글

퍼징의 유용성을 인지한 구글,

자사의 넘치는 컴퓨팅 파워를 기반으로 한 퍼징 플랫폼을 운영 중.



OSS-Fuzz: 지구방위대 구글

퍼징의 유용성을 인지한 구글,

자사의 넘치는 컴퓨팅 파워를 기반으로 한 퍼징 플랫폼을 운영 중.



23년 7월 4일 기준, 1000개 오픈소스 프로젝트에서 약 45000건의 결함 발견!!

OSS-Fuzz: 지구방위대 구글

퍼징의 유용성을 인지한 구글,

자사의 넘치는 컴퓨팅 파워를 기반으로 한 퍼징 플랫폼을 운영 중.



23년 7월 4일 기준, 1000개 오픈소스 프로젝트에서 약 45000건의 결함 발견!!

하루 평균 20건

그러나...

우리의 목표는?

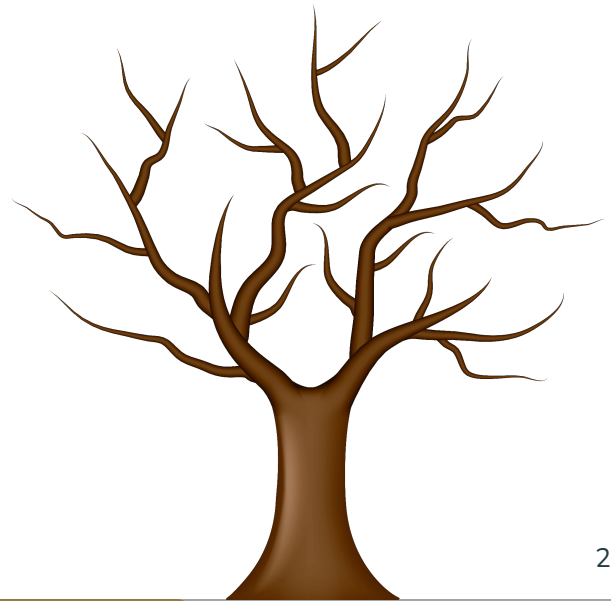
그러나...

우리의 목표는?

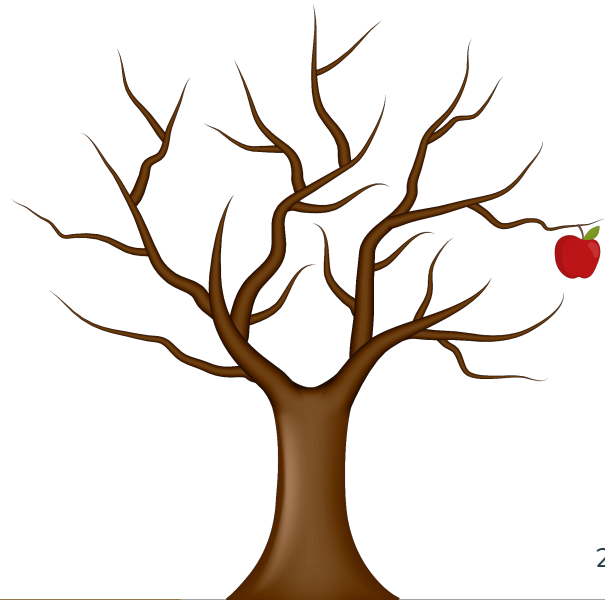
정적분석의 결과 검사!

정적분석의 결과 검사!

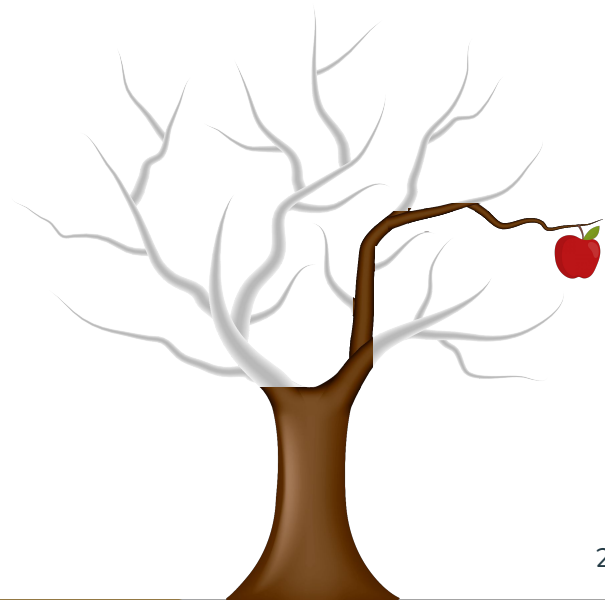
정적분석의 결과 검사!



정적분석의 결과 검사!



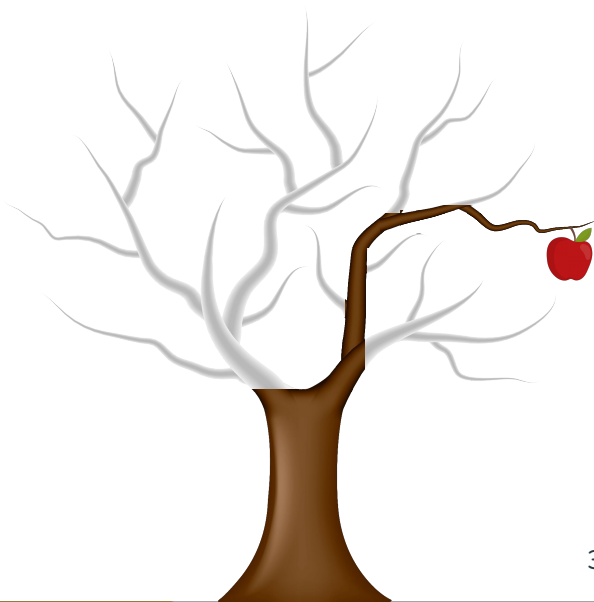
정적분석의 결과 검사!



지향성 퍼징 (Directed Fuzzing)

사용자가 관심 있는 곳에 닿게 하는 Fuzzing

정적분석의 결과 검사!



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

새로운 접근

DAFL (Directed AFL): 정적 분석을 활용한 지향성 퍼징 도구 (Usenix Security '23)

새로운 접근

DAFL (Directed AFL): 정적 분석을 활용한 지향성 퍼징 도구 (Usenix Security '23)

41개의 실제 결함에 대해 성능 평가

새로운 접근

DAFL (Directed AFL): 정적 분석을 활용한 지향성 퍼징 도구 (Usenix Security '23)

41개의 실제 결함에 대해 성능 평가

기존 도구들보다 약 9배 더 빠르게 목표 결함 발견!!

DAFL

핵심

- 선택하기

프로그램에서 의미 있는 부분을 선별하기

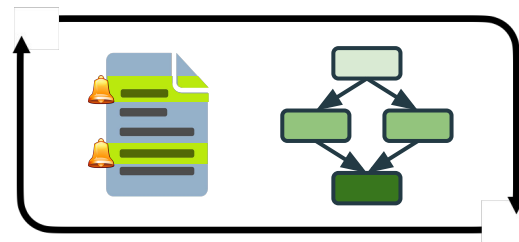
- 집중하기

선별된 프로그램 지점들로부터만 커버리지 피드백을 받기

- 품종개량

퍼징 과정에서 특정 시드에 우선순위 부여하기 (with 의미 기반 연관성 점수)

DAFL



선택하기

집중하기

품종개량

선택하기

선택하기

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

선택하기

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

결론: 의미 있는 함수를 선별해야 함

선택하기

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

결론: 의미 있는 함수를 선별해야 함

방법: 정의-사용(Def-Use) 관계를 고려하여 프로그램 지점 간추리기 (Slicing)

목표 지점과 정의-사용 관계로 이어진 라인들을 추적

선택하기

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

결론: 의미 있는 함수를 선별해야 함

방법: 정의-사용(Def-Use) 관계를 고려하여 프로그램 지점 간추리기 (Slicing)

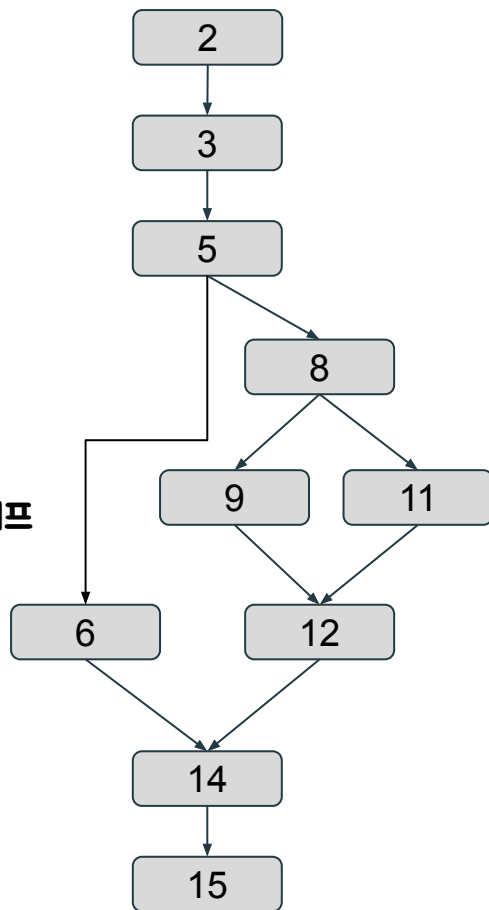
목표 지점과 정의-사용 관계로 이어진 라인들을 추적

프로그램 전체가 아닌 의미적으로 연관된 프로그램 지점만 고려할 수 있다!

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

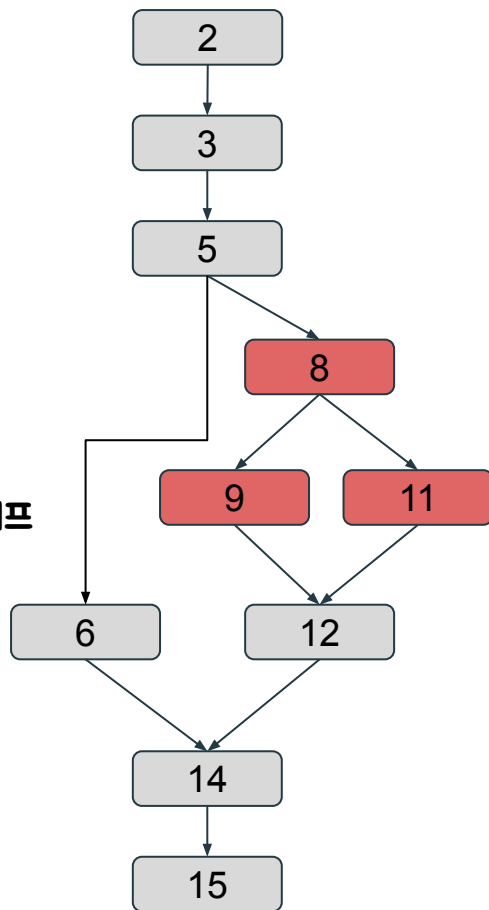
```
1: int x, a;  
2: x = input();  
3: a = input();  
4:  
5: if( ... )  
6:     x++;  
7: else {  
8:     if( ... )  
9:         a++;  
10:    else  
11:        a--;  
12:    x--;  
13: }  
14: if(x>3)  
15:     foo(x);
```

실행 흐름 그래프



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

실행 흐름 그래프

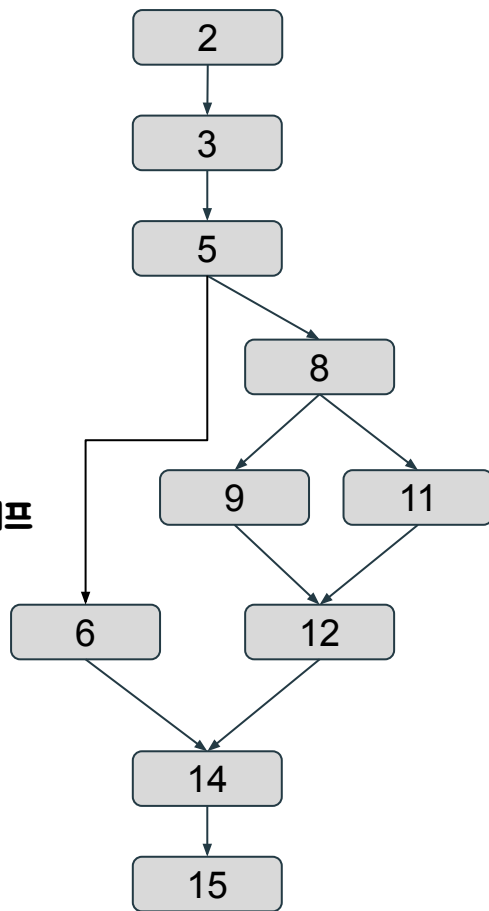


```

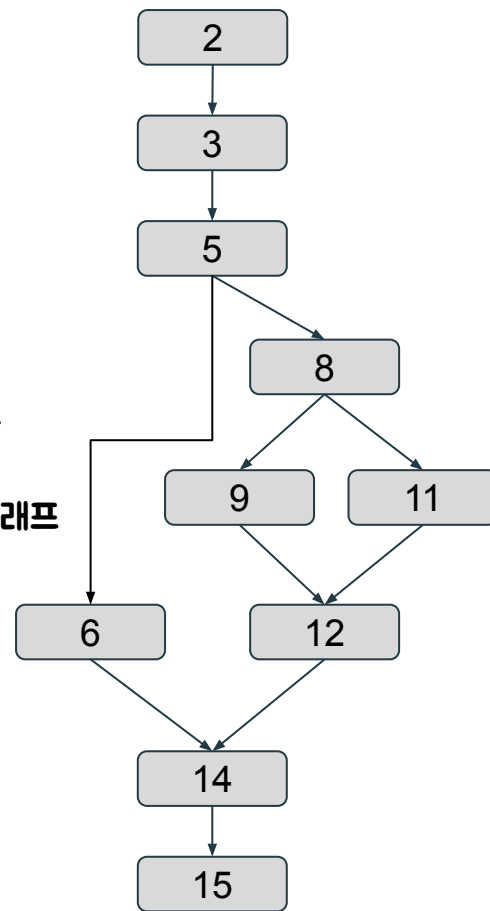
1:  int x, a;
2:  x = input();
3:  a = input();
4:
5:  if( ... )
6:      x++;
7:  else {
8:      if( ... )
9:          a++;
10:     else
11:         a--;
12:     x--;
13: }
14: if(x>3)
15:     foo(x);

```

실행 흐름 그래프



정의-사용 그래프

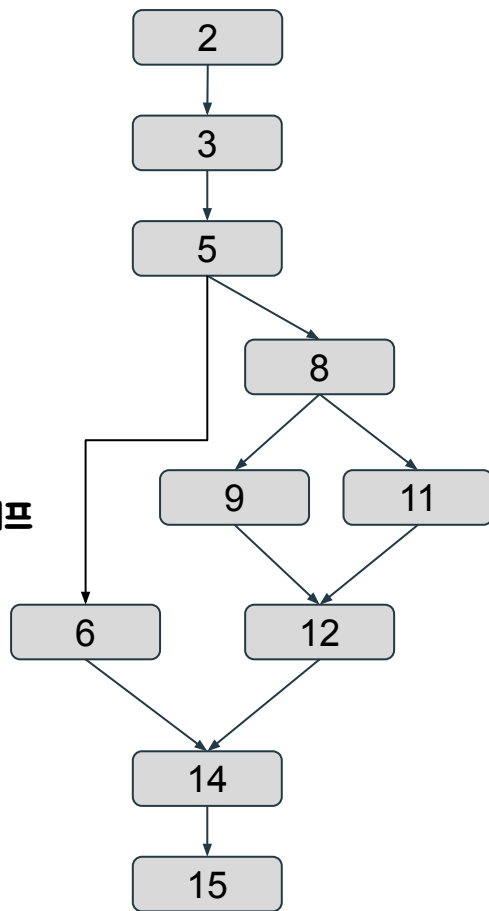



```

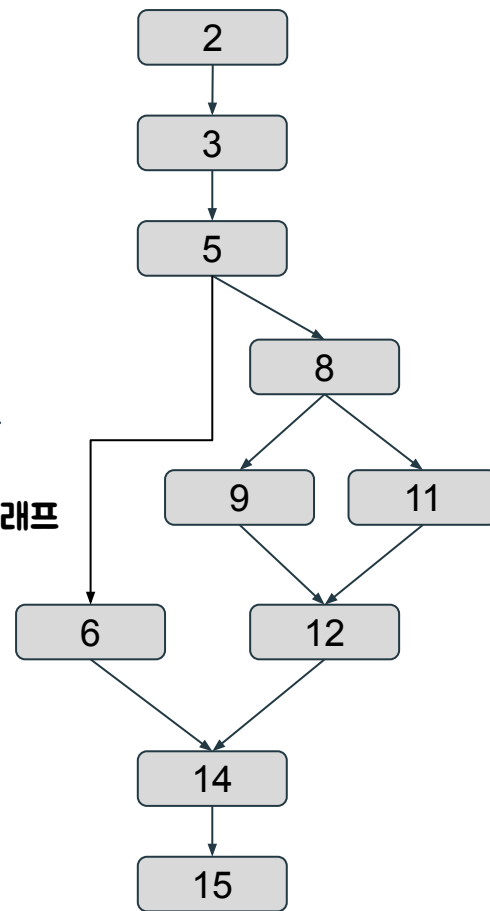
1: int x, a;
2: x = input();
3: a = input();
4:
5: if( ... )
6:     x++;
7: else {
8:     if( ... )
9:         a++;
10:    else
11:        a--;
12:    x--;
13: }
14: if(x>3)
15:     foo(x);

```

실행 흐름 그래프



정의-사용 그래프

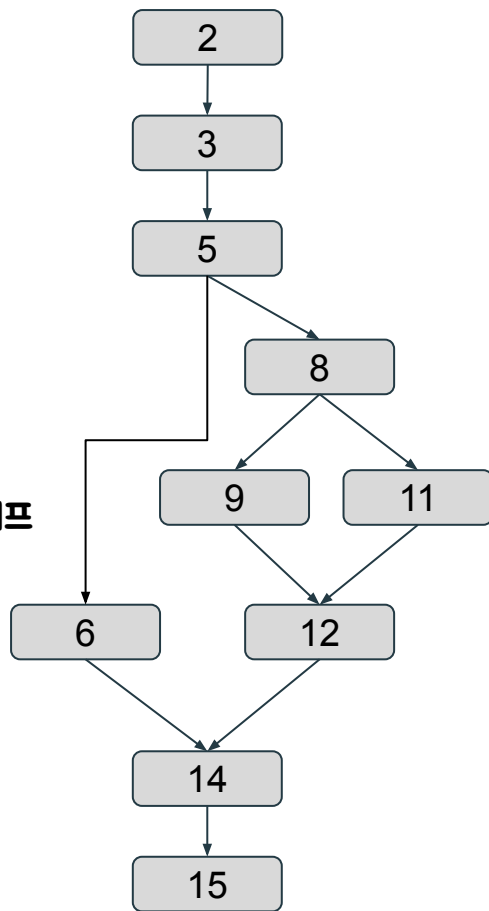


```

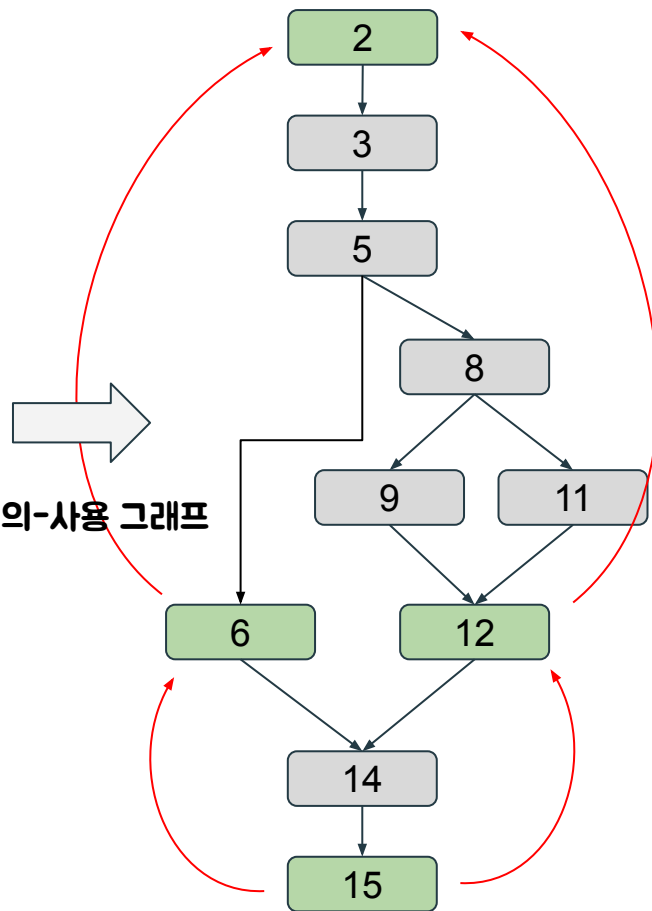
1: int x, a;
2: x = input();
3: a = input();
4:
5: if( ... )
6:     x++;
7: else {
8:     if( ... )
9:         a++;
10:    else
11:        a--;
12:    x--;
13: }
14: if(x>3)
15:     foo(x);

```

실행 흐름 그래프



정의-사용 그래프

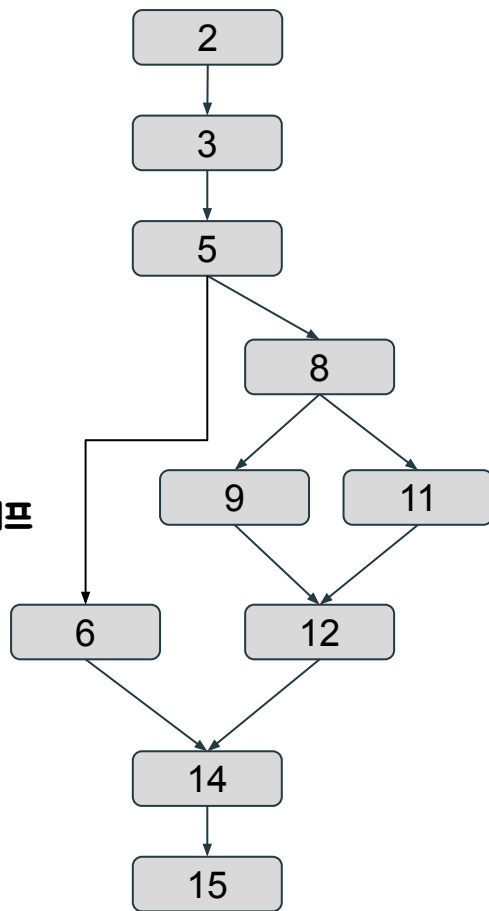


```

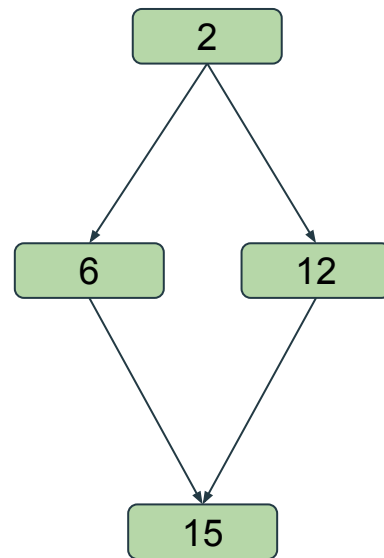
1: int x, a;
2: x = input();
3: a = input();
4:
5: if( ... )
6:     x++;
7: else {
8:     if( ... )
9:         a++;
10:    else
11:        a--;
12:    x--;
13: }
14: if(x>3)
15:     foo(x);

```

실행 흐름 그래프



정의-사용 그래프

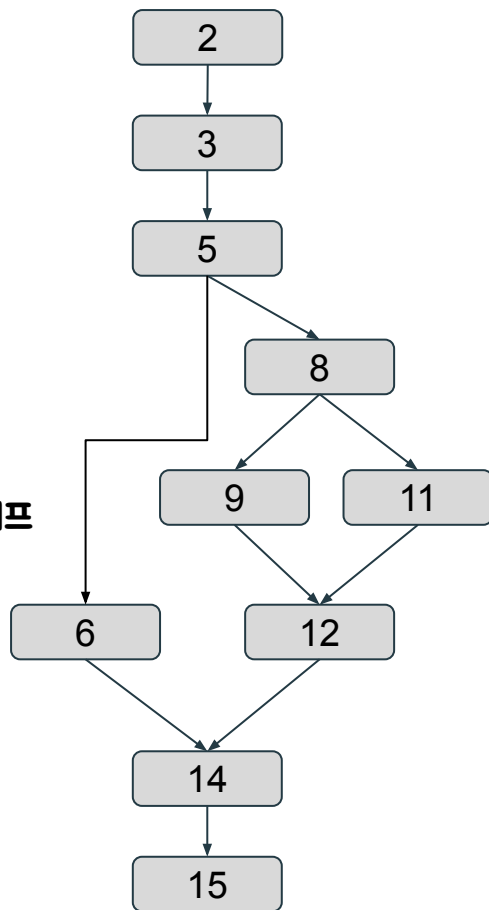


```

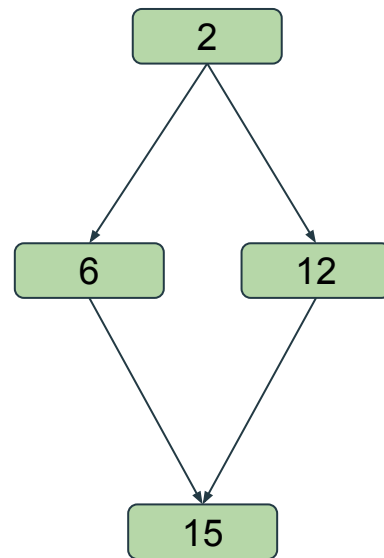
1: int x, a;
2: x = input();
3: a = input();
4:
5: if( ... )
6:     x++;
7: else {
8:     if( ... )
9:         a++;
10:    else
11:        a--;
12:    x--;
13: }
14: if(x>3)
15:     foo(x);

```

실행 흐름 그래프



정의-사용 그래프



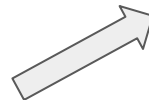
정의 사용 그래프를 통해
연관된 함수들 파악 가능!



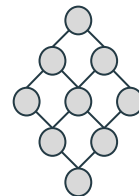
선택하기



선택하기

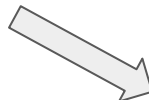
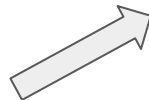


정의-사용 그래프

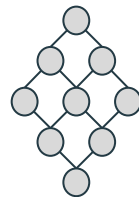




선택하기



정의-사용 그래프



함수 목록




집중하기 & 품종개량

집중하기: 선별된 프로그램 지점들로부터만 커버리지 피드백을 받기

=> 함수 목록 사용 

품종개량: 퍼징 과정에서 특정 시드에 우선순위 부여하기

=> 정의-사용 그래프를 그대로 활용 

집중하기

집중하기

원하는 지점의 실행 정보만 선별적으로 기록 (Selective Instrumentation)

집중하기

원하는 지점의 실행 정보만 선별적으로 기록 (Selective Instrumentation)

= 관련된 함수 목에 방울 달기 🔔

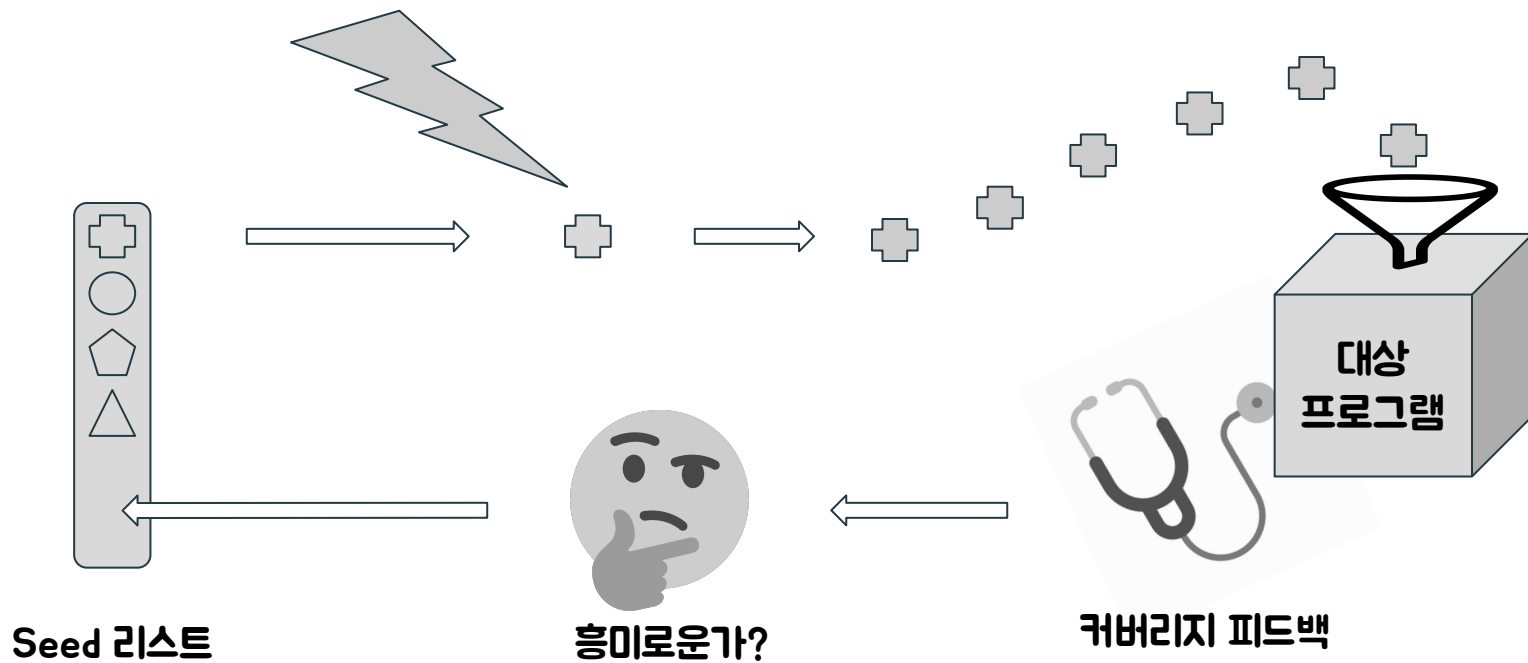
집중하기

원하는 지점의 실행 정보만 선별적으로 기록 (Selective Instrumentation)

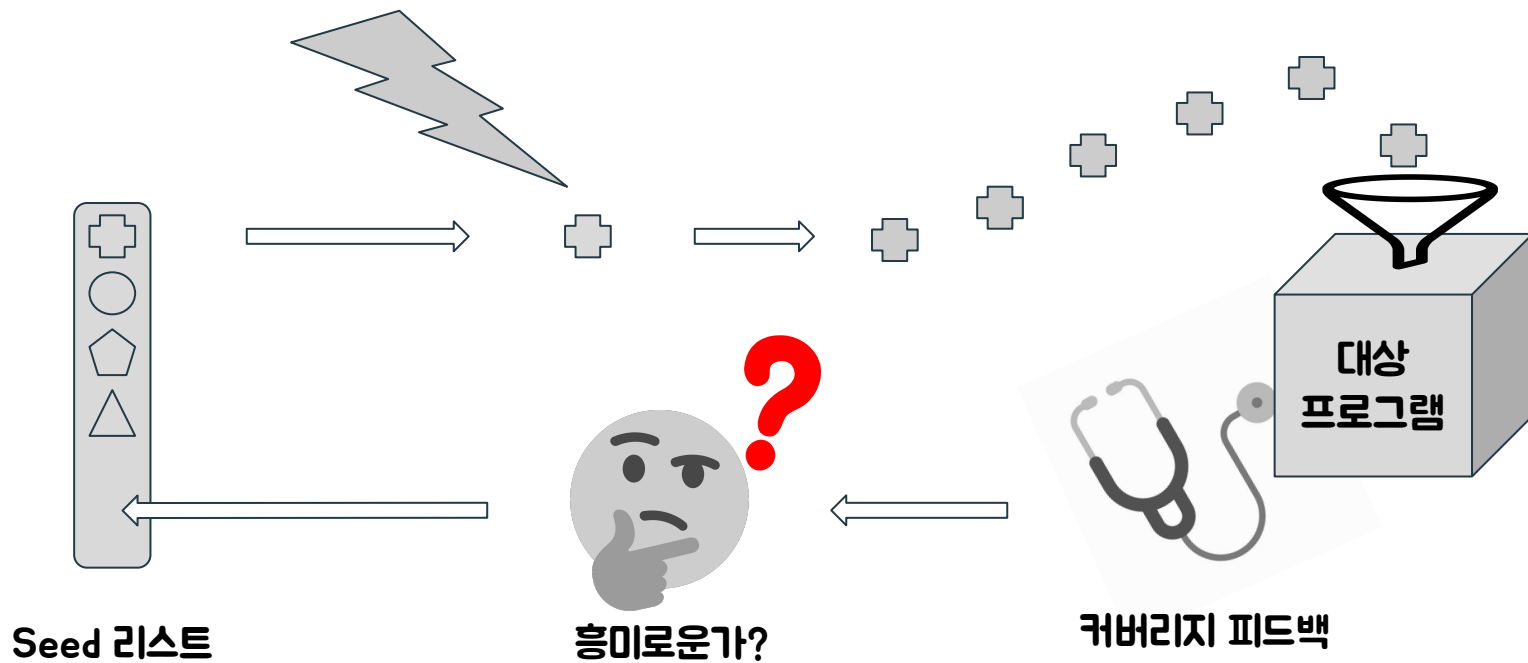
= 관련된 함수 목에 방울 달기 🛎



집중하기



집중하기



품종개량

품종개량

두 가지 결정:

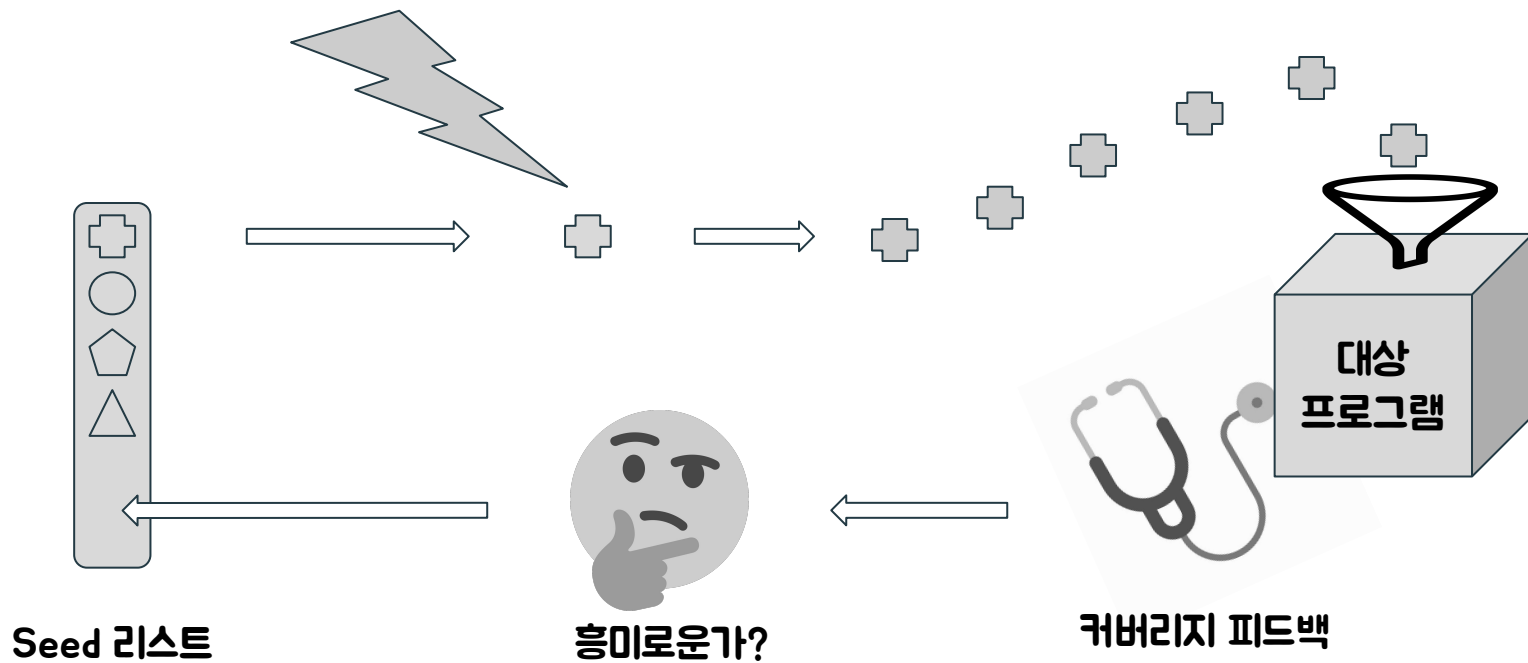
- 가지고 있는 종자 중, 무엇을 먼저 심을 지 결정
- 심을 때, 얼마나 많이 심을 지 결정

품종개량

두 가지 결정:

- Seed 중 무엇을 먼저 사용할 지 결정
- 선택된 Seed를 얼마나 많이 사용할 지 결정

품종개량

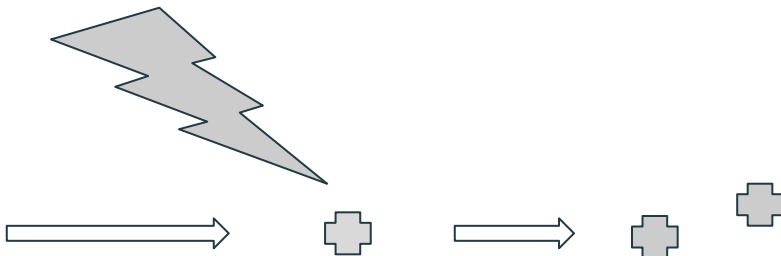


품종개량

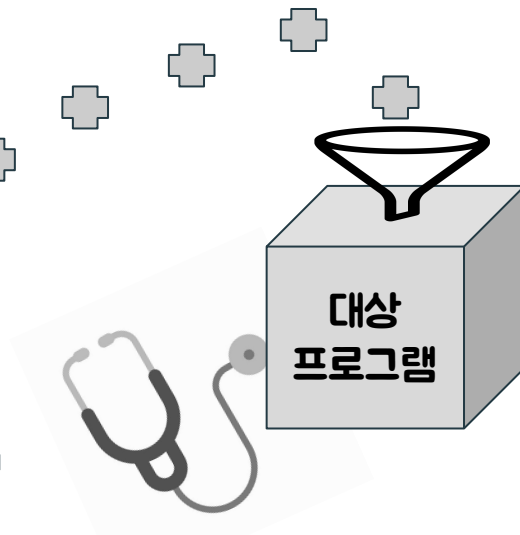
무엇을 먼저 사용할까?



Seed 리스트

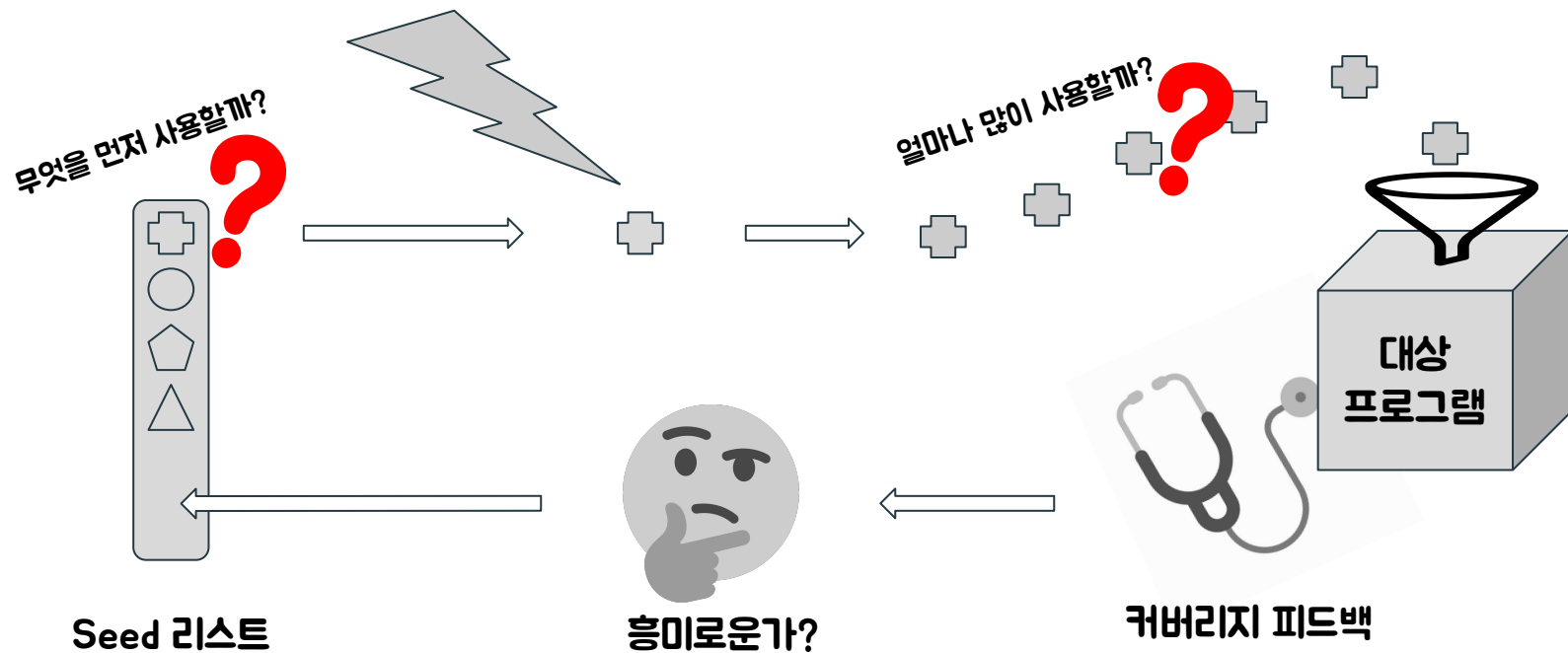


흥미로운가?



커버리지 피드백

품종개량



품종개량

두 가지 선택:

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

=> 두 선택 모두 '의미 기반 연관성 점수(Semantic Relevance Score)'를 고려!

품종개량

두 가지 선택:

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

=> 두 선택 모두 '의미 기반 연관성 점수(Semantic Relevance Score)'를 고려!

의미 기반 연관성 점수란?

정의-사용 그래프를 기반으로 계산된 각 Seed별 점수

의미 기반 연관성 점수 (Semantic Relevance Score)

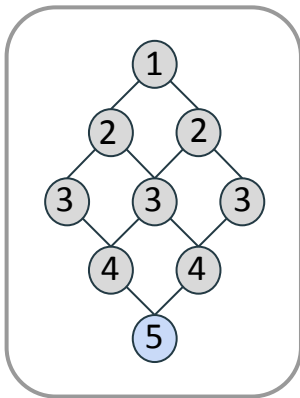
의미 기반 연관성 점수 (Semantic Relevance Score)

Seed A

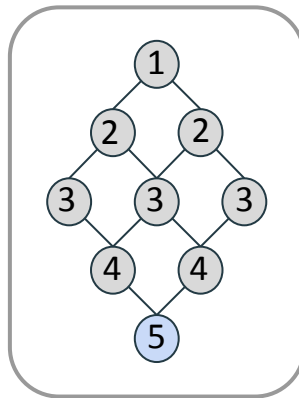
Seed B

의미 기반 연관성 점수 (Semantic Relevance Score)

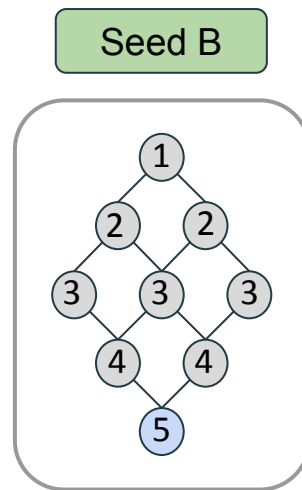
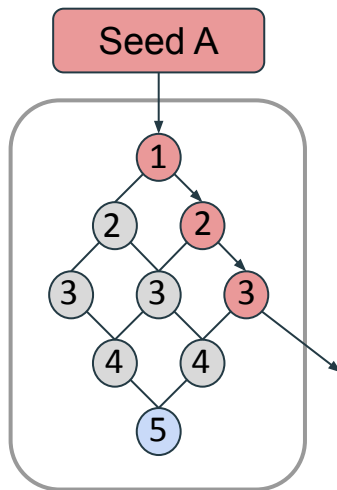
Seed A



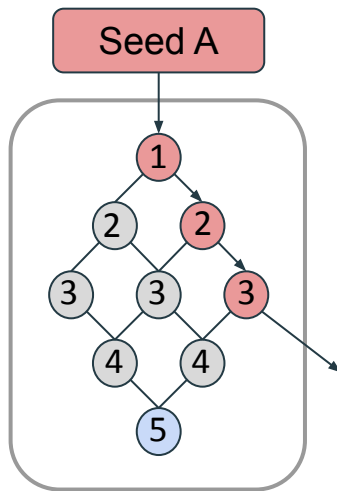
Seed B



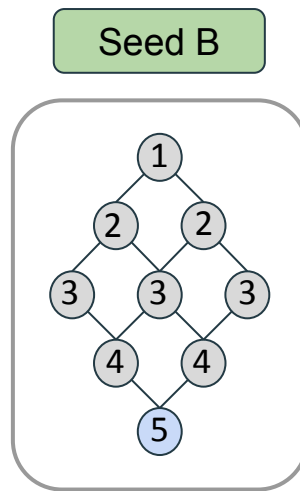
의미 기반 연관성 점수 (Semantic Relevance Score)



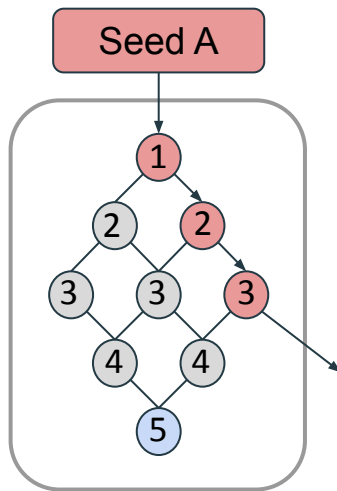
의미 기반 연관성 점수 (Semantic Relevance Score)



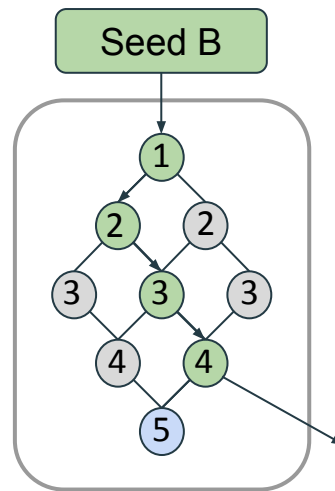
6점



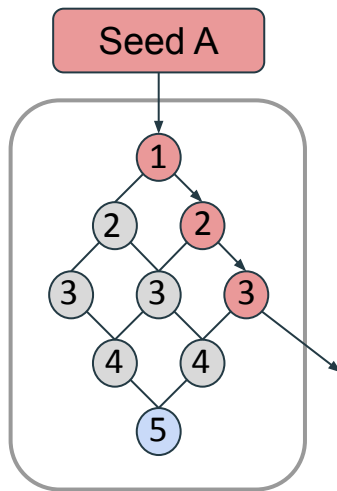
의미 기반 연관성 점수 (Semantic Relevance Score)



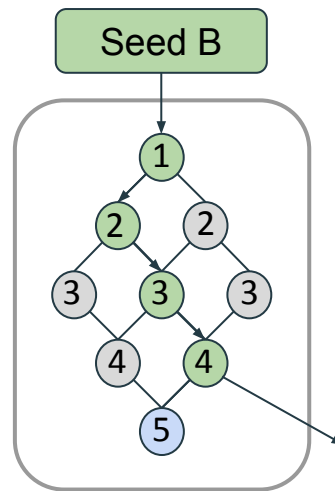
6점



의미 기반 연관성 점수 (Semantic Relevance Score)

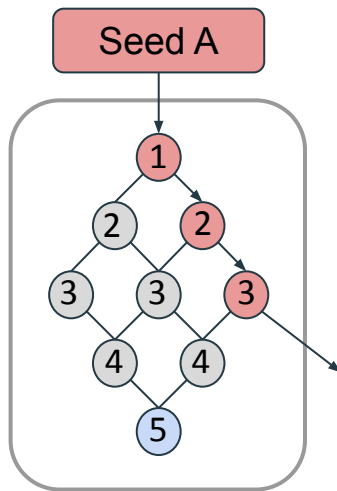


6점

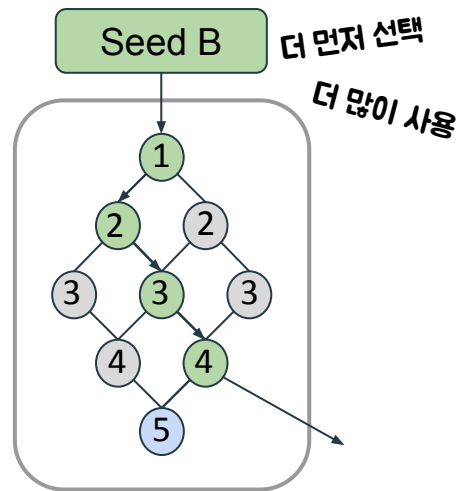


10점

의미 기반 연관성 점수 (Semantic Relevance Score)

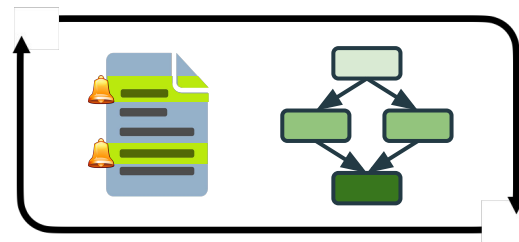


6점



10점

DAFL 정리



선택하기
with
프로그램 지점 간추리기

집중하기
with
선별적 실행 정보 기록

품종개량
with
의미 기반 연관성 점수

실험

실험 대상: 10개 프로그램의 41개 결함

비교 대상:

- AFL: 무지향성 퍼저
- AFLGo^[1]: AFL 기반 지향성 퍼저
- WindRanger^[2]: AFL 기반 지향성 퍼저

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

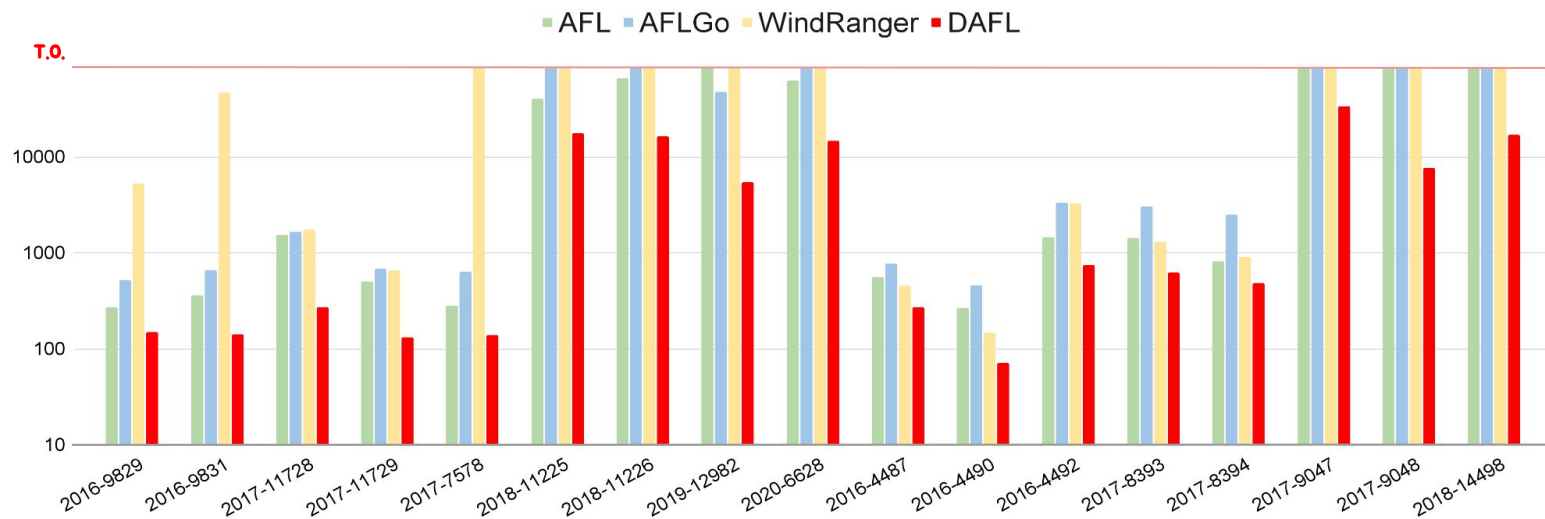
[1] "Directed Greybox Fuzzing", Bohme et al., CCS 2017

[2] "WindRanger: a directed greybox fuzzer driven by deviation basic blocks", Du et al., ICSE 2022

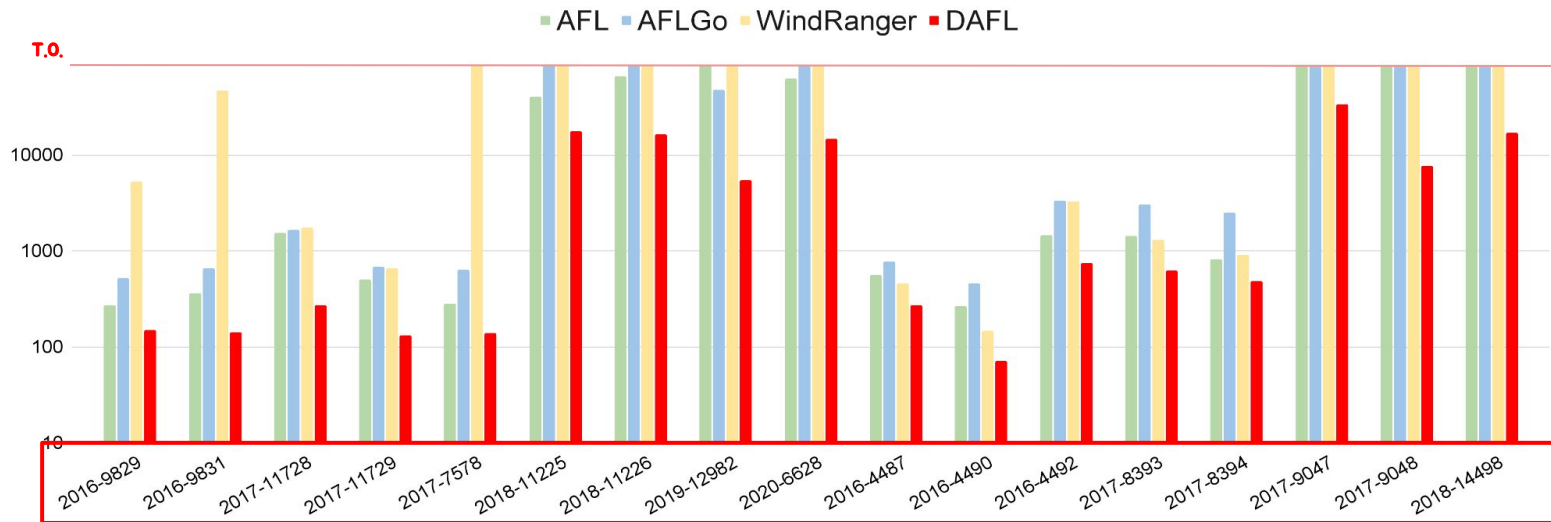
실험 결과

1. 41개 중 **27개** 결함을 가장 빨리 발견!
2. 다른 도구와 비교했을 때, 최소 **2배**, 최대 **20배**의 성능 향상!
3. 다른 도구는 제한 시간안에 (거의) 찾지 못하는 결함 **5개** 발견!

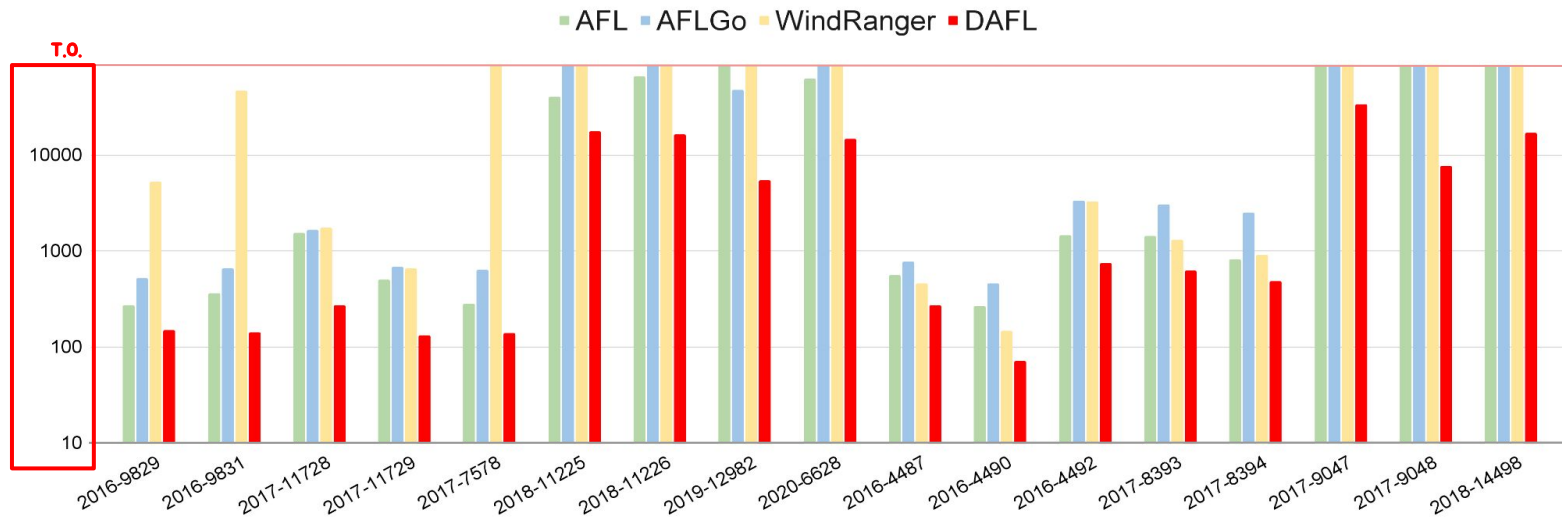
실험 결과



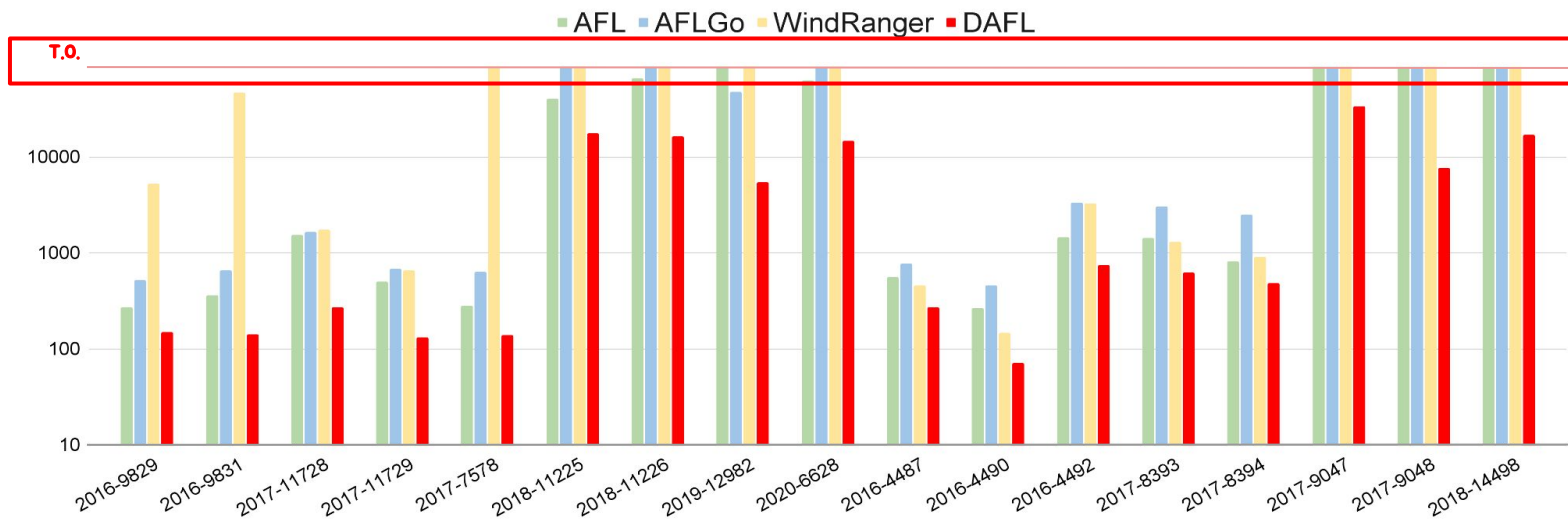
실험 결과



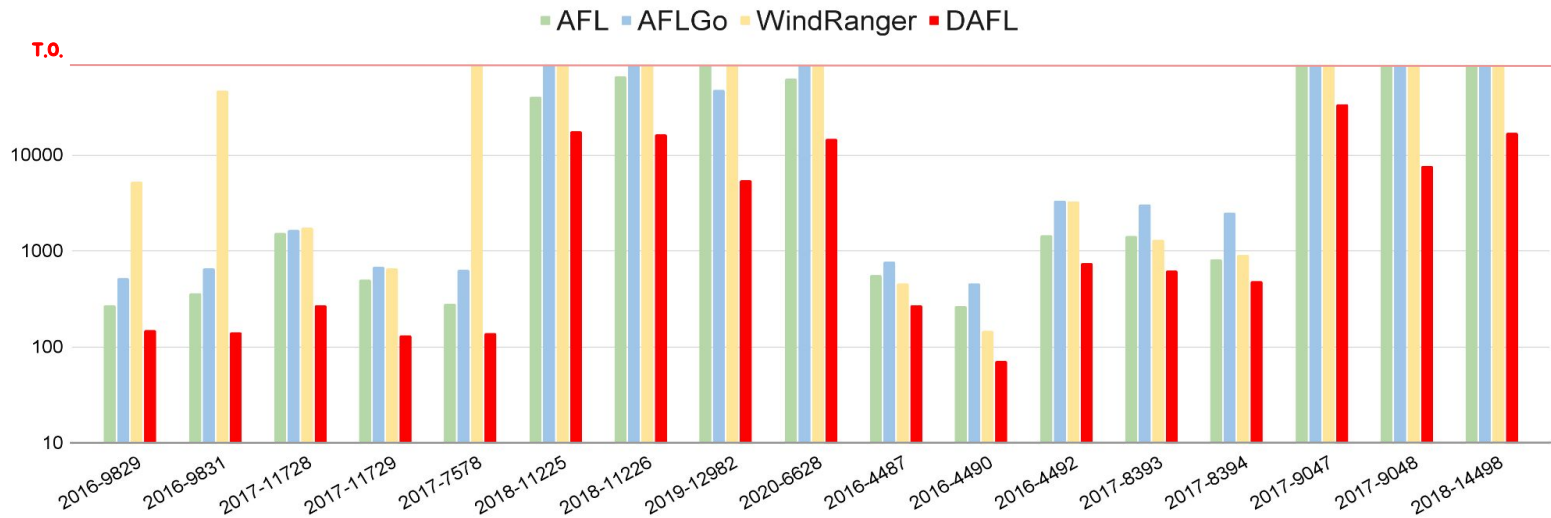
실험 결과



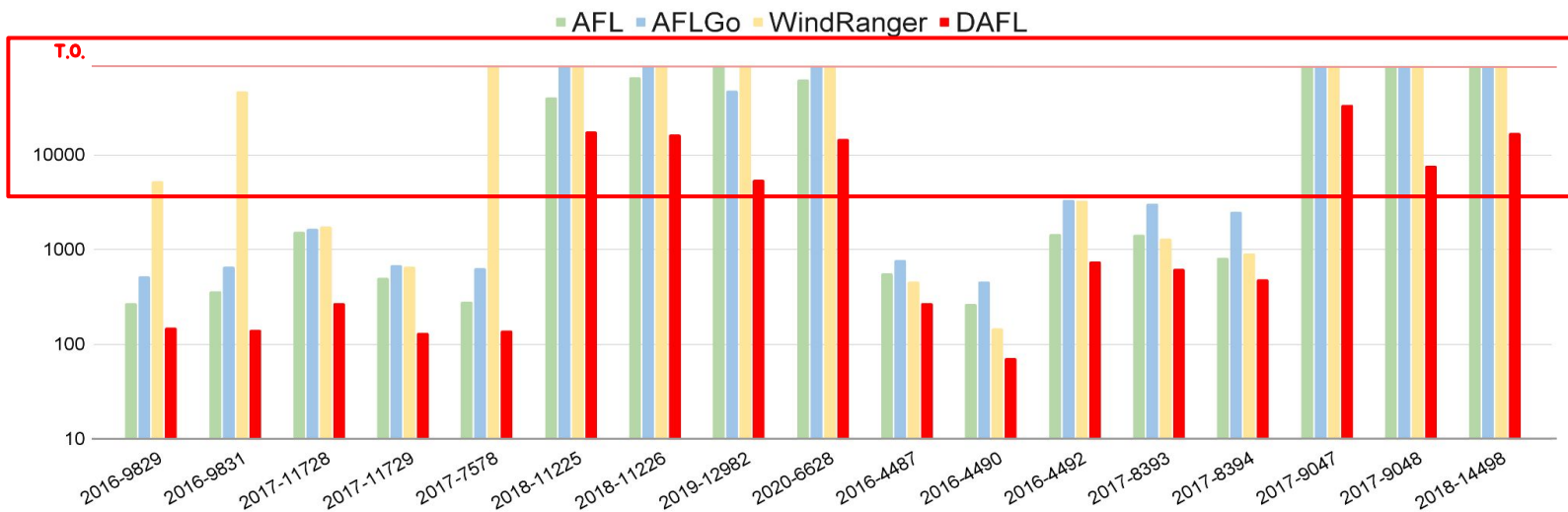
실험 결과



실험 결과



실험 결과



발표 요약

- **지향성 퍼징이란?**
 - 주어진 목표지점에 도달하는 입력을 생성하는 퍼징
- **효과적인 지향성 퍼징을 하는 법**
 - 데이터 의존 관계를 분석해 목표 지점과 연관된 지점 파악 및 집중하기
- **목표 지점과 연관된 지점에 집중하는 방법**
 - 선택적으로 커버리지 피드백 받기
 - 의미 기반 점수를 활용하여 Seed 품종 개량하기
- **최종 목표**
 - 정적 분석 알람으로부터 참 알람 선별
 - 결함 유발 입력까지 생성하여 오류 데이터베이스 구축

보충 슬라이드

선별하기

평균 약 24%의 함수만 선별

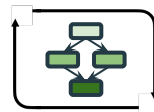
Program	# CVE	All	Naive	Thin
swftophp	17	786	429	234
lrzip	2	154	112	77
cxxfilt	6	211	40	35
objcopy	3	1478	1184	891
objdump	5	1787	1347	875
strip	1	1485	3	3
nm	1	1306	1017	50
readelf	1	441	185	112
xmllint	3	2414	1215	416
cjpeg	2	101	52	43

세부 실험: 집중하기 & 품종 개량

집중하기: 선별된 지점으로부터만 커버리지 피드백 받기



품종개량: 의미 기반 점수를 활용하여 Seed 선택 및 사용하기

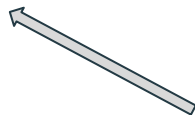


세부 실험: 집중하기 & 품종 개량

세부 실험: 집중하기 & 품종 개량



집중하기: 1.73배 성능 향상

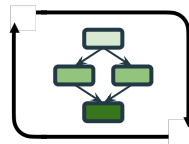


AFL

세부 실험: 집중하기 & 품종 개량



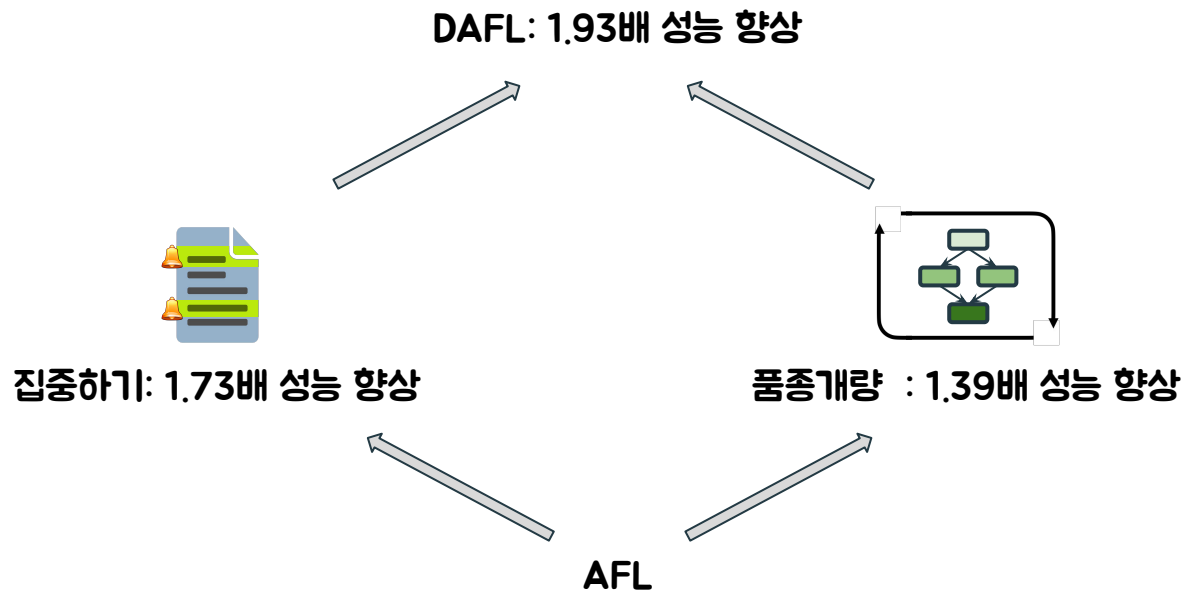
집중하기: 1.73배 성능 향상



품종개량 : 1.39배 성능 향상

AFL

세부 실험: 집중하기 & 품종 개량



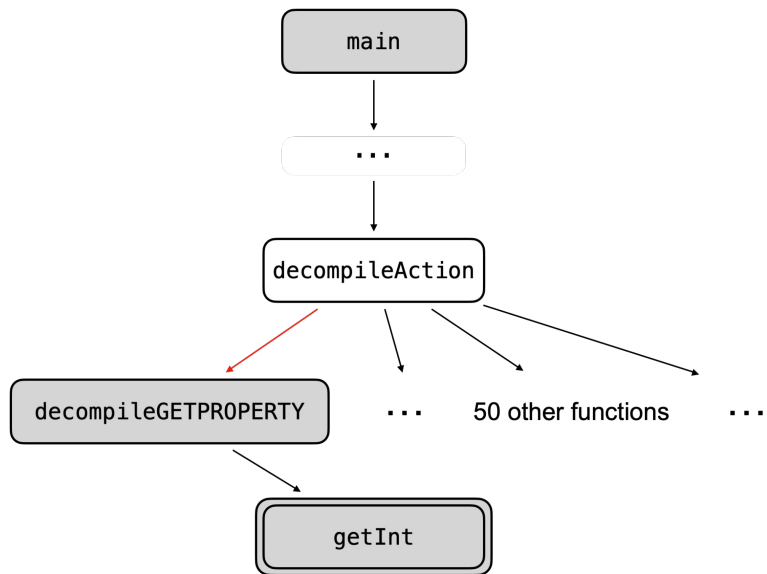
한계

4개의 결함에서 성능 하락

주요 원인: 결함 발생에 관여하는 중요한 함수가 누락되는 문제 발생

한계

사례: CVE-2018-20427



한계

누락된 주요 함수들을 추가해준 결과,

- 3개 결함은 다른 도구들보다 월등히 빨리 발견
- 1개 결함은 다른 도구들과 비슷한 속도로 발견