

# 지향성 퍼징, 제대로 평가하고 있나요?

**Evaluating Directed Fuzzers: Are We Heading in the Right Direction?**

김태은, 최재승, 임성재, 허기홍, 차상길



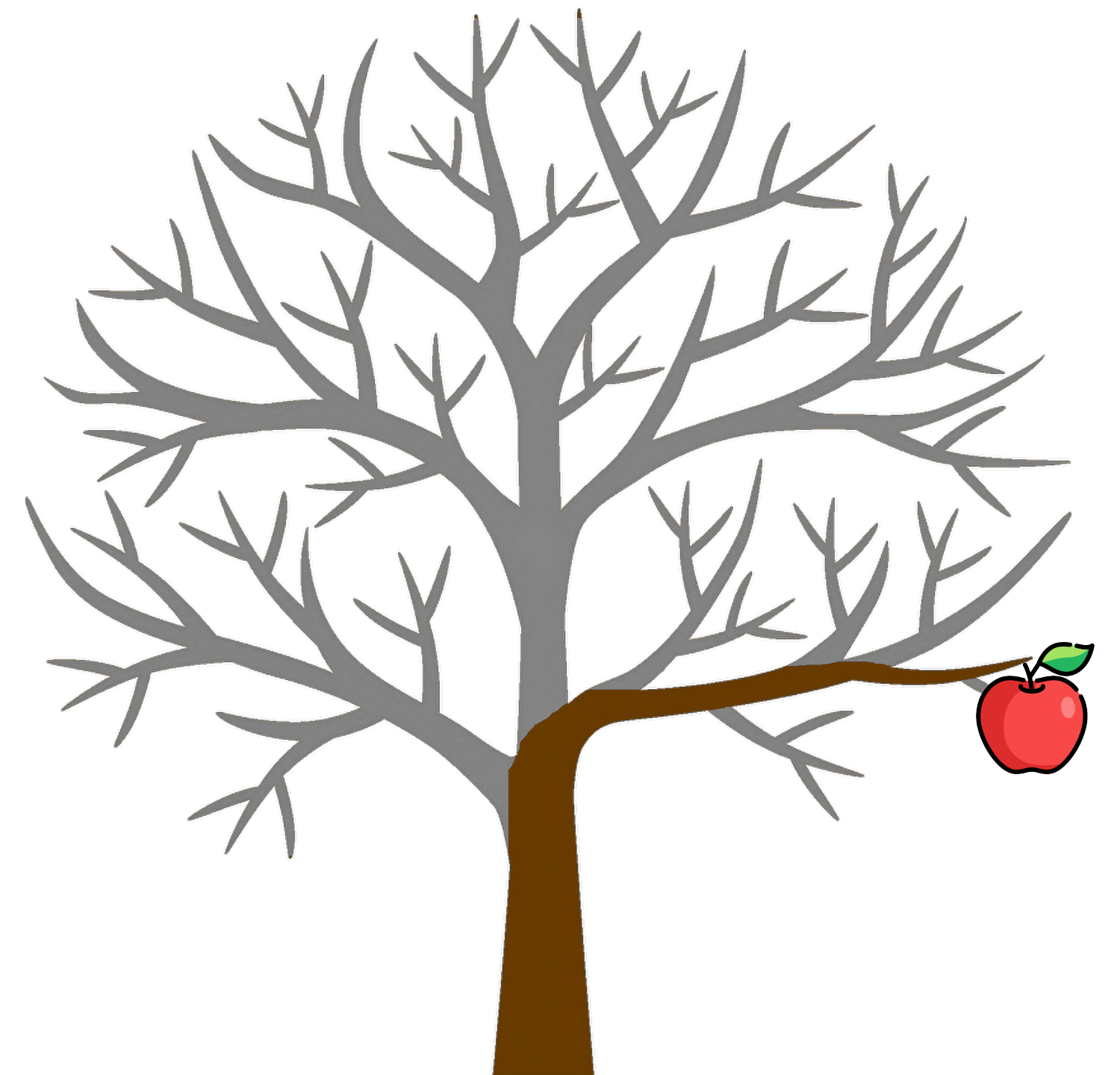
# 배경

## 퍼징

- 자동으로 생성한 입력으로 프로그램을 검사하는 기법
- 성공적인 사례들
  - e.g., AFL, 구글의 OSS Fuzz 프로젝트

## 지향성 퍼징

- 사용자가 관심있는 곳에 닿게 하는 퍼징
  - 예) 정적분석의 알람을 검사할때 유용



# 배경

## 지향성 퍼징 기술의 성능 평가

**핵심 지표:** 주어진 목표 오류를 얼마나 빨리 재현하는가?

→ Time-To-Exposure (TTE)

## 문제:

- 평가의 표준이 아직 정립되지 않음
  - 지향성 퍼징 평가에서 주의할 점들이 충분히 고려되지 않음
- 평가의 투명성과 재현성을 위협

# 지향성 퍼징 평가의 함정들

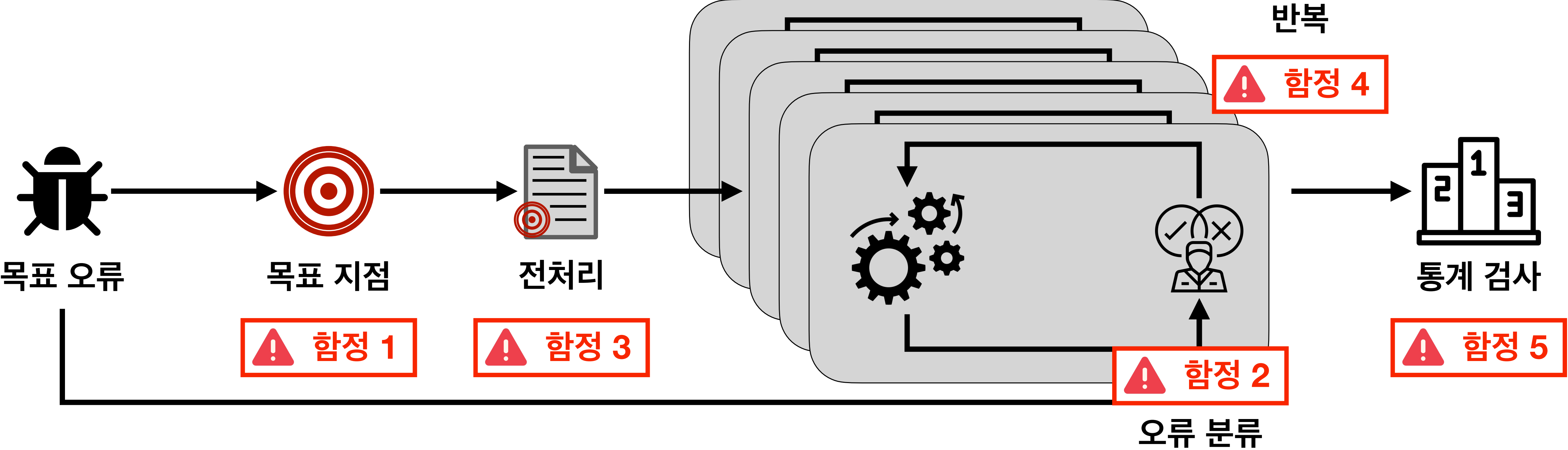
**조사:** 총 14개 지향성 퍼징 논문을 조사하여 성능 평가의 실태 파악

**실험:** 그 중 5개 도구의 실험 결과를 통해 성능 평가의 각 단계 검사

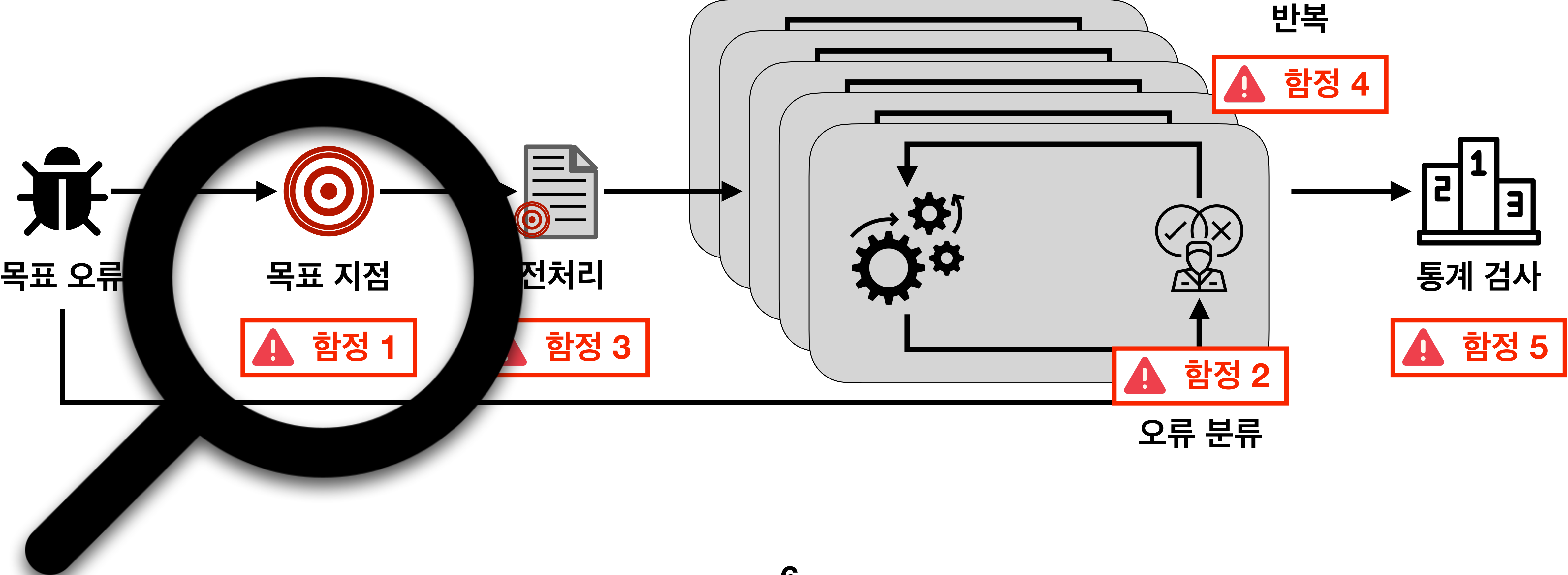
**결과:**

- 성능 평가의 각 단계에 존재하는 5개의 함정
- 투명하고 재현 가능한 성능 평가를 위한 5개의 교훈

# 함정 1: 목표 지점



# 함정 1: 목표 지점



# 함정 1: 목표 지점

목표 오류로부터 목표 지점을 선정하는 과정은 단순하지 않다

현황: 대부분의 논문은 목표 오류를 CVE 번호로 명시함 (14편 중 12편)

\*Common Vulnerabilities and Exposure

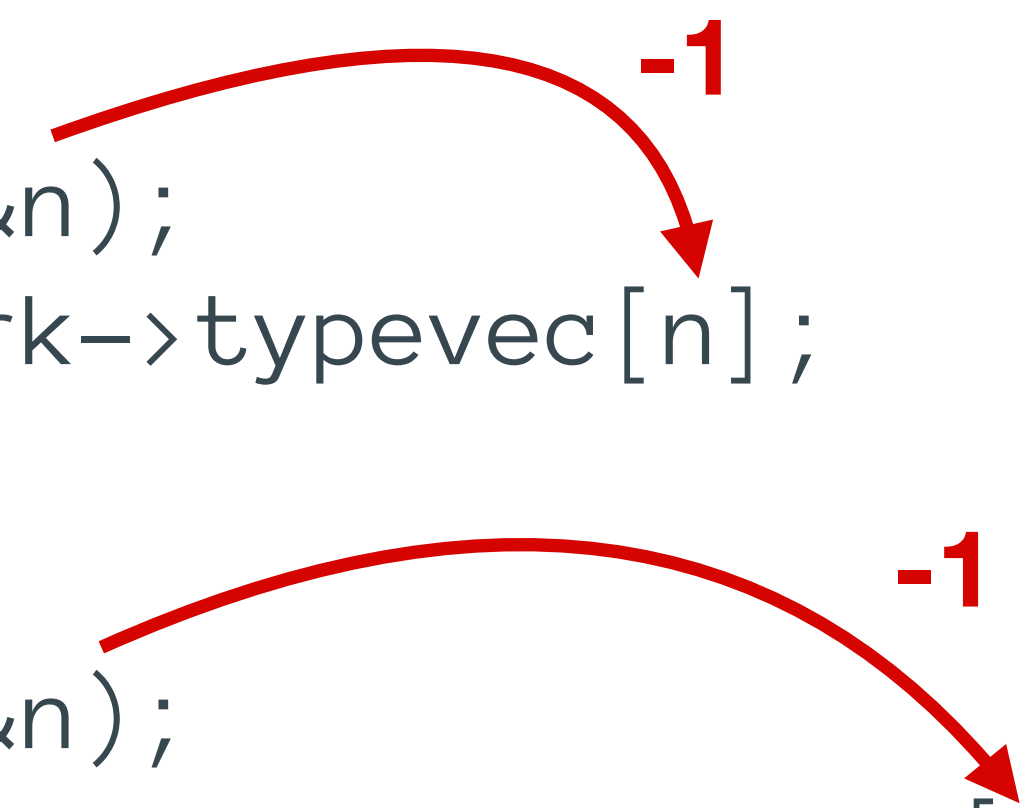
문제:

- 목표 오류는 평가의 목표지, 지향성 퍼징 도구의 목표는 아니다
- 대부분의 지향성 퍼징 도구는 목표 오류가 아닌 특정한 목표 라인을 입력으로 받는다

# 함정 1: 목표 지점

예) **CVE-2016-4492**: 두 개의 크래시 지점을 가진 오류

```
1 int do_type(work_stuff *work, char **mangled)
2     int n;
3     switch (**mangled) {
4         case 'T':
5             get_count (mangled, &n);
6             remembered_type = work->typevec[n];           // Crash Site 1
7             ...
8         case 'B':
9             get_count (mangled, &n);
10            string_append (result, work->btypevec[n]); // Crash Site 2
11    }
12 }
```





# 합정 1: 목표 지점

Q. 아무 라인이나 선택하면 안되나?

A. 아무 라인이나 선택하면 결과가 크게 달라진다

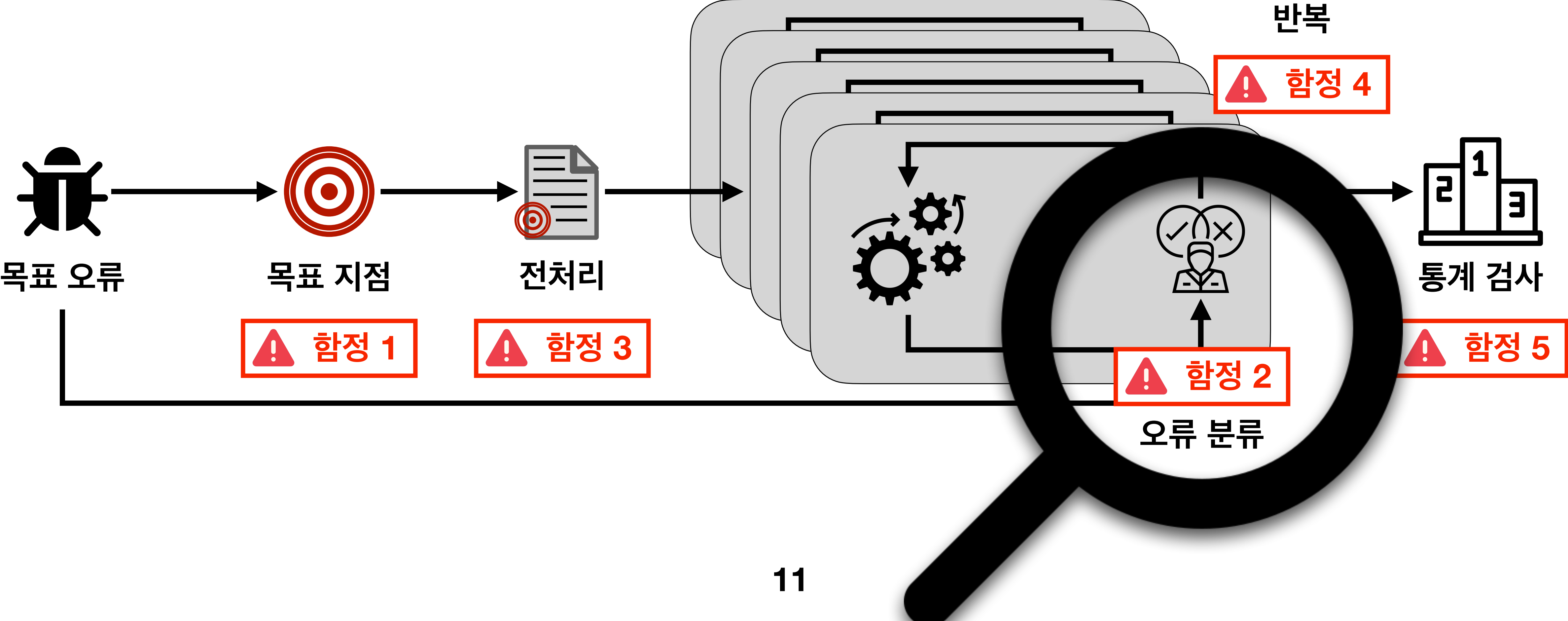
\* 160번 실험의 중앙값 TTE를 초 단위로 표시함

목표 라인	AFLGo	Beacon	WindRanger	SelectFuzz	DAFL
6번 라인	373	333	2,460	432	787
10번 라인	332	499	339	581	149

# 함정 1: 목표 지점

- ⚠ CVE 번호를 사용하는 12편 중 6편은 목표 지점을 따로 명시하지 않았다.
- ✅ 지향성 퍼징 도구에게 주어졌던 구체적인 목표 지점을 명시

# 함정 2: 오류 분류



# 함정 2: 오류 분류

```
ERROR: AddressSanitizer: heap-buffer-overflow ...  
#0 in parseSWF_RGBA parser.c:66  
#1 in parseSWF_MORPHGRADIENTRECORD parser.c:746  
...  
#6 in blockParse blocktypes.c:145  
#7 in readMovie main.c:265  
#8 in main main.c:350
```

TTE는 오류 분류의 사소한 차이에도 크게 달라진다

현황: 새니타이저 기반 분류

- ASAN 등의 새니타이저 로그를 활용하는 방식 (오류 유형 및 스택 트레이스)
- 발견한 오류 유발 입력을 CVE 보고서와 비교
  - 오류 설명
  - 오류 유발 입력 (POC)

문제: 어떻게 비교할 지 결정하는 것이 쉽지 않다

# 함정 2: 오류 분류

CVE 보고서:

“Heap-based buffer overflow in the parseSWF\_RGBA function”

예) CVE-2016-9831

```
1 void parseSWF_MORPHGRAD(FILE *f, SWF_MORPHGRAD *g) {
2     ...
3     g->NumGradients = readUInt8(f); <----- NumGradients 를 검사하지 않음
4     for (i = 0; i < g->NumGradients; i++)
5         parseSWF_MORPHGRADREC(f, &(g->GradientRecords[i]));
6 }
7
8 void parseSWF_MORPHGRADREC(FILE *f, SWF_MORPHGRADREC *r) {
9     r->StartRatio = readUInt8(f); <----- 같은 원인으로 가능한 또 다른 크래시 지점
10    parseSWF_RGBA(f, &r->StartColor);
11 }
12
13 void parseSWF_RGBA(FILE *f, SWF_RGBA *rgb) {
14     rgb->red      = readUInt8(f); <----- CVE 보고서의 POC는 여기서 크래시
15     rgb->green    = readUInt8(f); <----- CVE 보고서가 언급하는 또 다른 라인
16 }
```

# 합정 2: 오류 분류

예) CVE-2016-9831

CVE report:

“Heap-based buffer overflow in the parseSWF\_RGBA function”

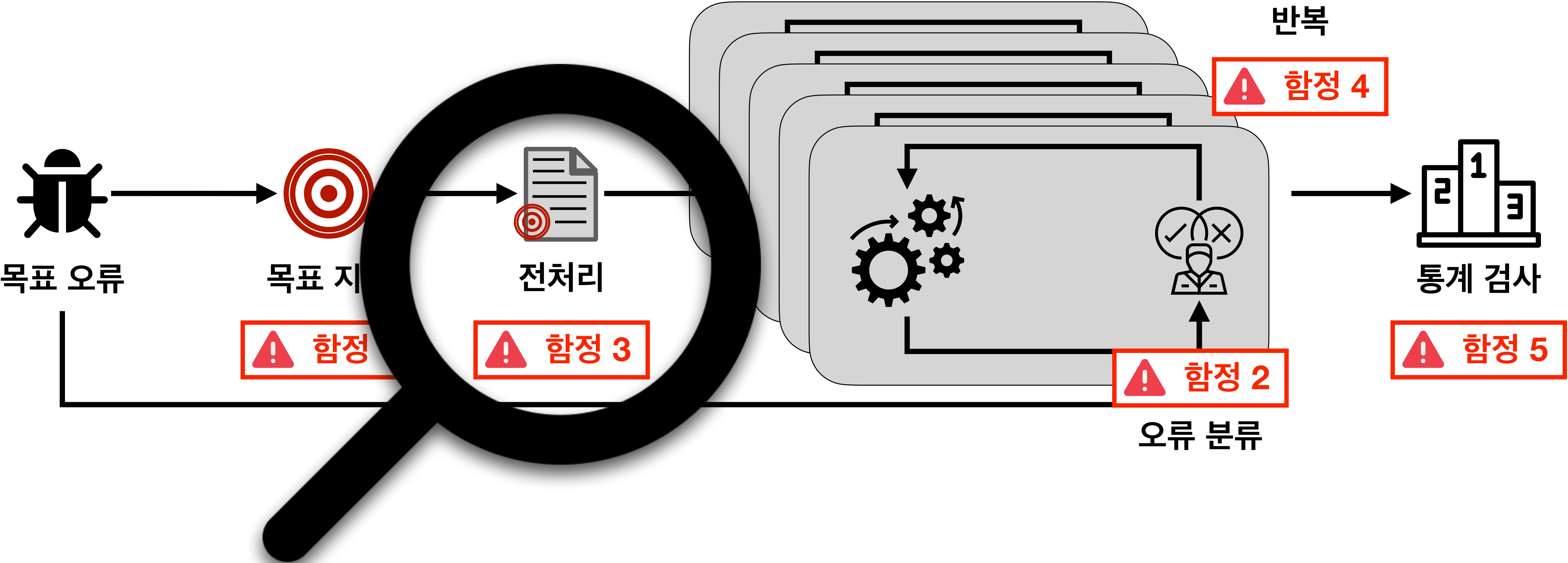
검사한 라인	AFLGo	Beacon	WindRanger	SelectFuzz	DAFL
14					
14,15					
14,15, 9					

```
9  r->StartRatio = readUInt8(f); <----- 같은 원인으로 가능한 또 다른 크래시 지점
10 parseSWF_RGBA(f, &r->StartColor);
11 }
12
13 void parseSWF_RGBA(FILE *f, SWF_RGBA *rgb) {
14     rgb->red      = readUInt8(f); <----- CVE 보고서의 POC는 여기서 크래시
15     rgb->green    = readUInt8(f); <----- CVE 보고서가 언급하는 또 다른 라인
16 }
```

# 함정 2: 오류 분류

- ⚠ 단 5편의 논문만이 오류 분류의 세부 사항을 공개함
- ✅ 오류 분류의 세부 사항 및 실제 사용한 스크립트 공개

# 함정 3: 전처리





# 함정 3: 전처리

전처리 비용을 무시하는 것은 정확한 성능 평가를 어렵게 한다

현황: 대부분의 지향성 퍼징 도구는 정적 분석을 활용 (14편 중 12편)

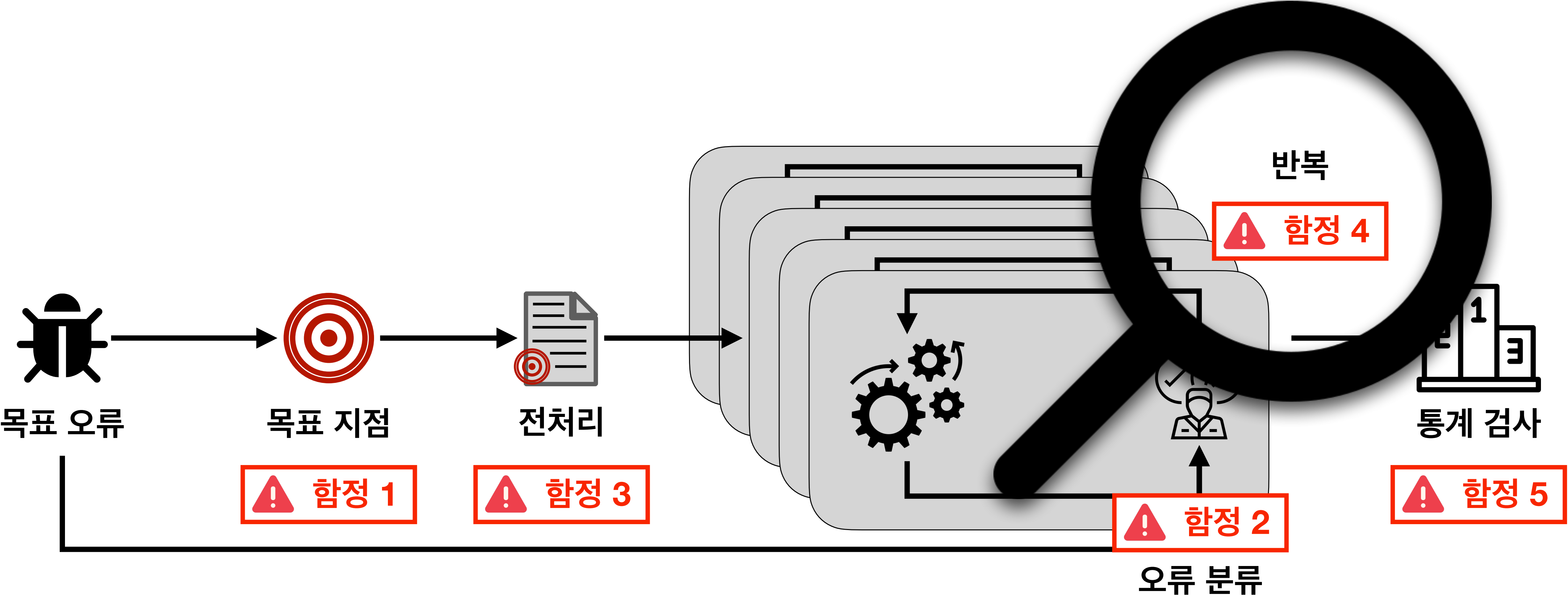
문제:

- 정적 분석은 한번 하고 마는 작업이 아니다
- 정적 분석의 비용이 실제 퍼징 비용보다 더 클 수 있다

⚠ 단 3편의 논문만이 정적 분석 시간을 구체적으로 공개함

✅ 처음부터 끝까지 소요되는 시간을 공개하여 각 지향성 퍼징 도구의 실질적인 성능을 평가

# 함정 4: 반복 횟수



# 합정 4: 반복 횟수

무작위성은 지향성 퍼징의 결과에 큰 영향을 준다

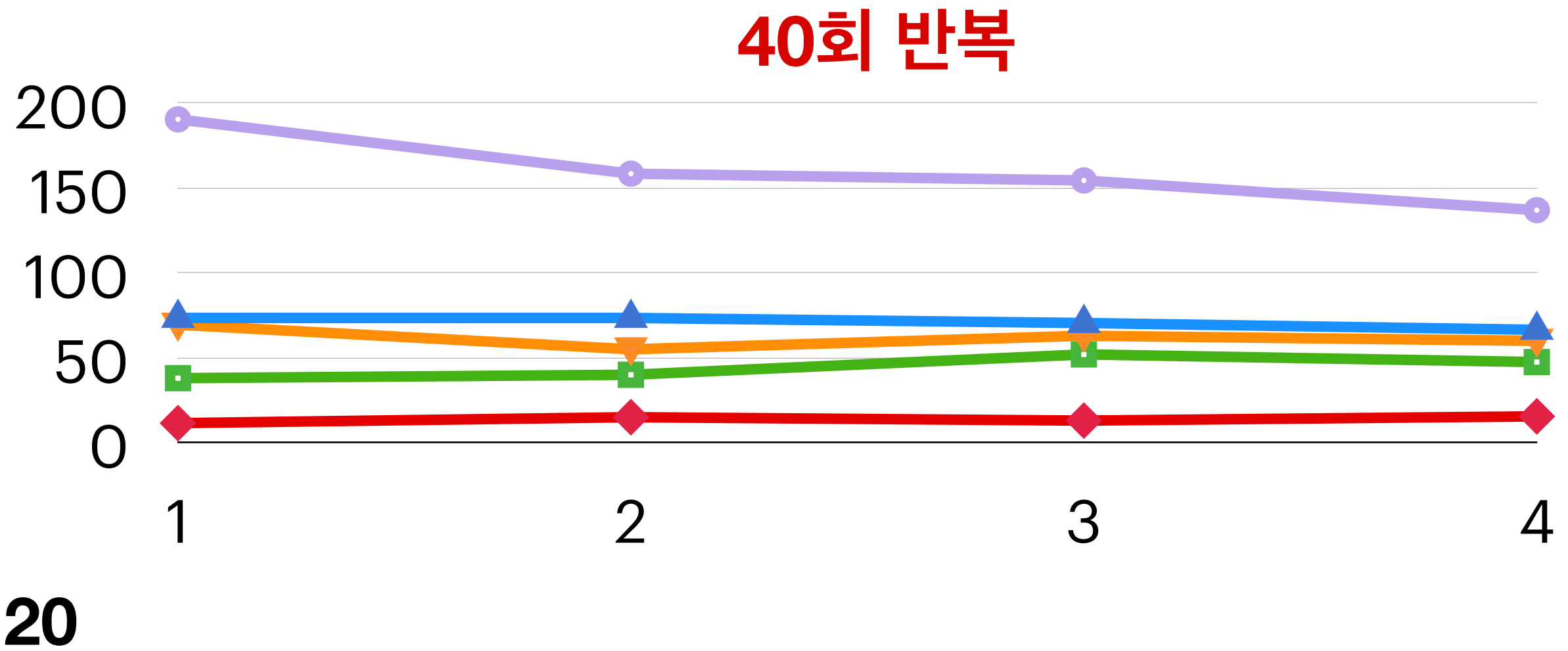
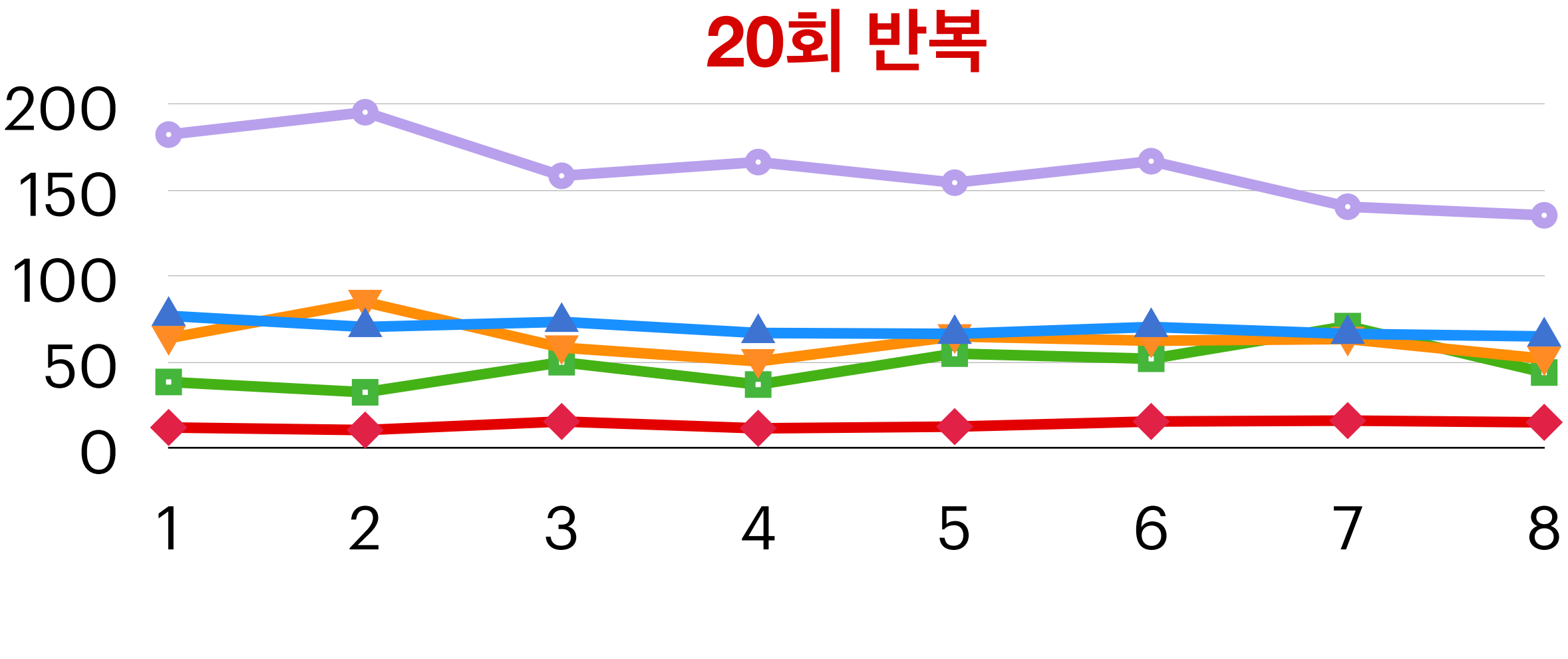
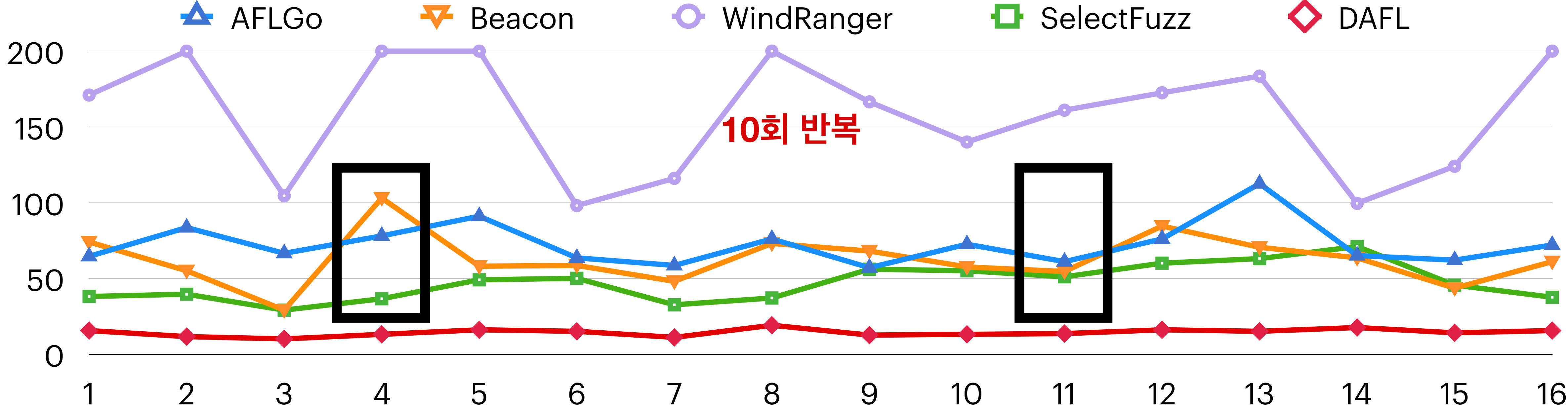
현황: 모든 논문이 반복 실험을 수행함

문제: 반복 횟수가 충분하지 않다

예) **CVE-2016-4490**: 타임아웃이 발생하지 않는 무난한 목표 오류

- 160회의 반복 실험
- 실험 결과를 10회, 20회, 40회씩 묶어 중앙값을 비교

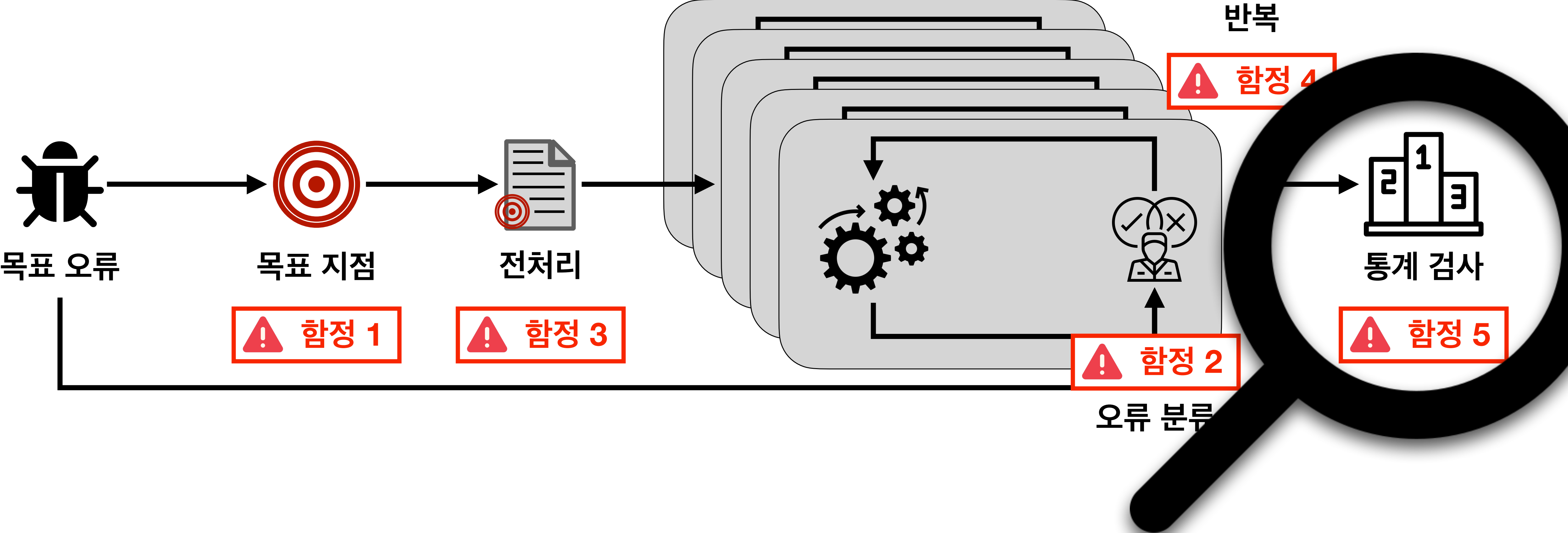
# 합정 4: 반복 횟수



# 함정 4: 반복 횟수

- ⚠ 기존 논문들의 평균 반복 횟수는 16회, 그 중 절반은 10회 이하
- ✅ 최소 20회 이상 반복 실험을 통해 안정적인 결과 도출

# 함정 5: 통계 검사



# 합정 5: 통계 검사

부적절한 통계 검사 기법은 잘못된 결론을 내리게 한다

현황:

결과의 유의미함을 확인하기 위해 Mann-Whitney U (MWU) 검사 수행

문제: MWU 검사는 관찰되지 않은 데이터를 다룰 수 없다 (예: 타임아웃)

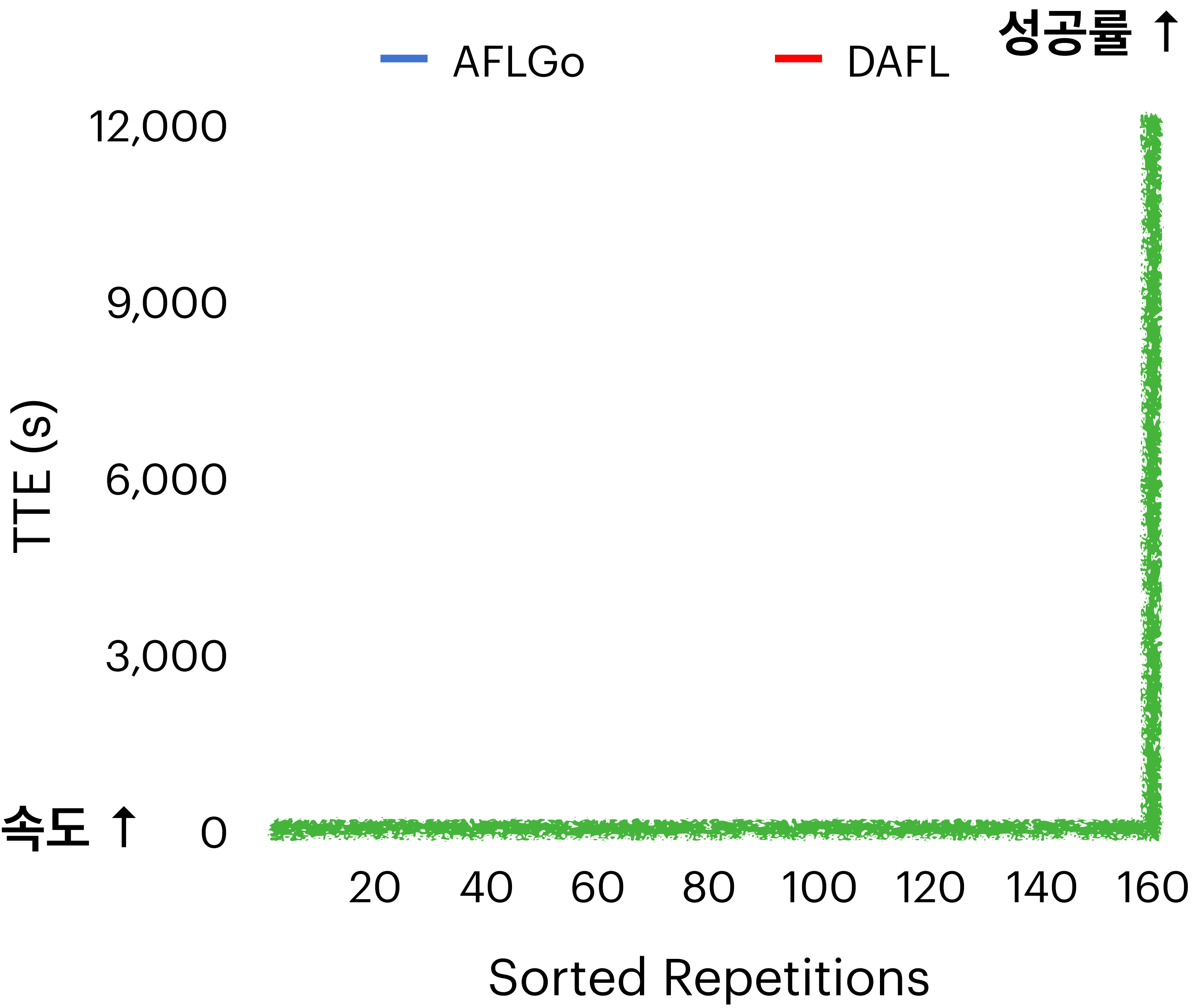
- 주어졌던 제한 시간을 TTE로 사용
- 타임아웃이 발생한 경우를 데이터에서 누락

# 합정 5: 통계 검사

예) CVE-2017-9988

지표	AFLGo	DAFL
중앙값 TTE	1,066	703
MWU 검사	p-value = 0.014	
타임아웃 횟수	1	17
로그순위 검사	p-value > 0.5	

\* 로그순위 검사: 생존 분석에서 사용되는 통계 기법  
타임아웃이 발생한 경우도 고려한다



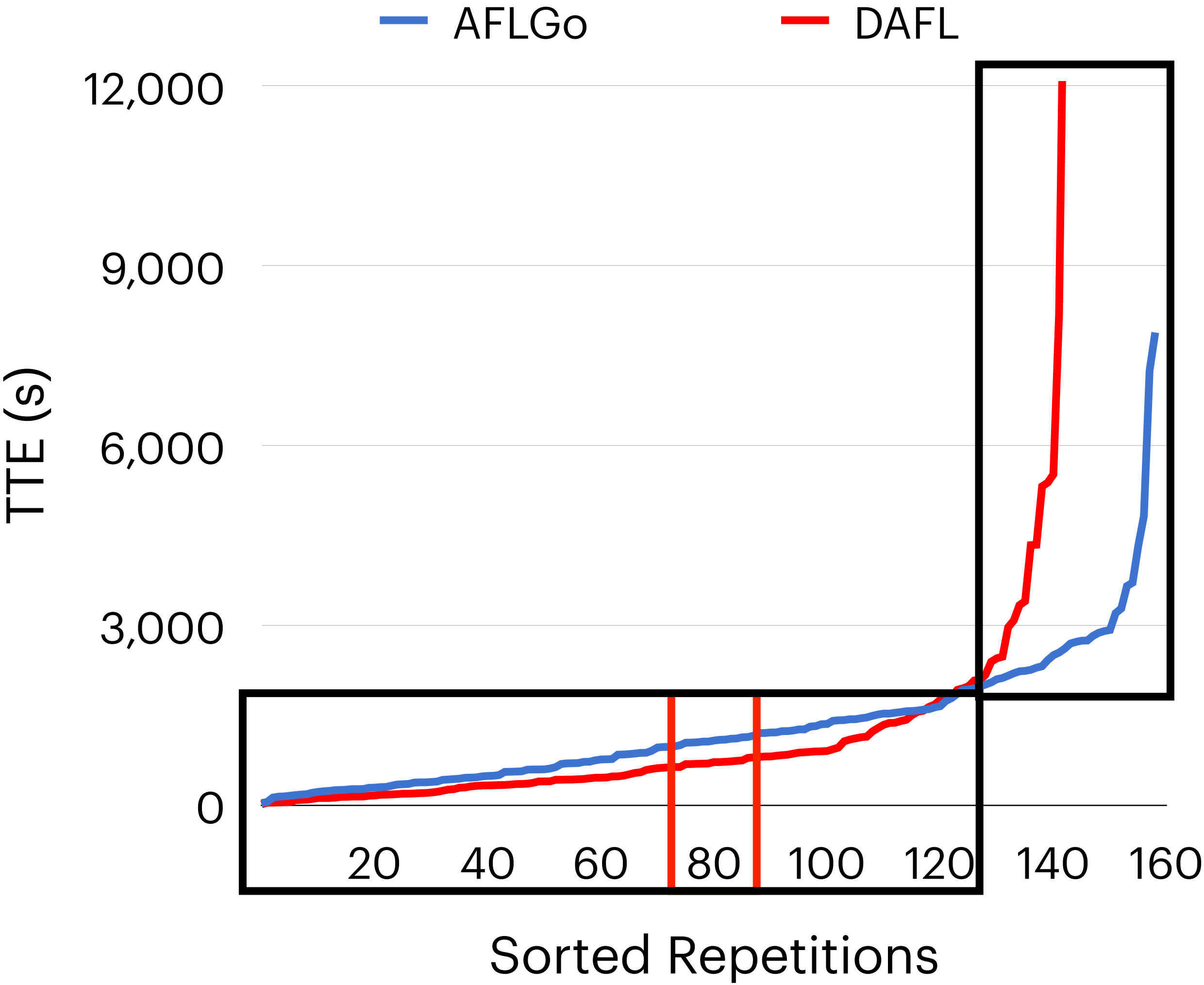


# 합정 5: 통계 검사

예) CVE-2017-9988

지표	AFLGo	DAFL
중앙값 TTE	1,066	703
MWU 검사	p-value = 0.014	
타임아웃 횟수	1	17
로그순위 검사	p-value > 0.5	

\* 로그순위 검사: 생존 분석에서 사용되는 통계 기법  
타임아웃이 발생한 경우도 고려한다



# 합정 5: 통계 검사

- ⚠ 8편의 논문은 MWU 검사에 의존함
- ✓ 로그순위 검사와 선인장 그래프를 사용하여 보다 정확한 결론 도출

# 요약



Artifact Link

## 지향성 퍼징의 성능 평가를 위한 교훈

- ✓ 지향성 퍼징 도구에게 주어졌던 구체적인 목표 지점을 명시
- ✓ 오류 분류의 세부 사항 및 실제 사용한 스크립트 공개
- ✓ 처음부터 끝까지 소요되는 시간을 공개하여 각 지향성 퍼징 도구의 실질적인 성능을 평가
- ✓ 최소 20회 이상 반복 실험을 통해 안정적인 결과 도출
- ✓ 로그순위 검사나 선인장 그래프를 사용하여 보다 정확한 결론 도출

더 자세한 내용은 논문과 포스터를 참고해주세요!