Bug-finder
From the reality

Toward pragmatic bug-finder

Daejun Park
Fasoo.com, Inc.

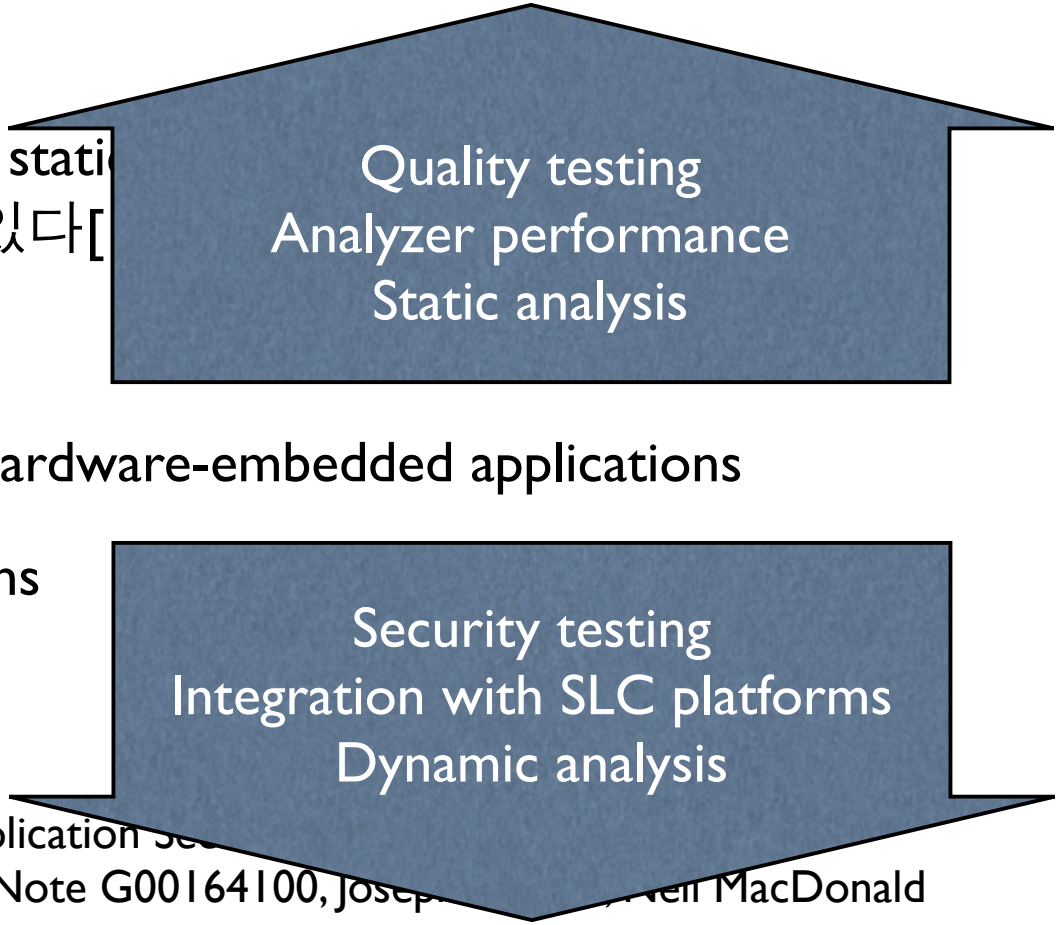
분석 시장 현황

- Quality/security testing, static/dynamic analysis 시장은 크게 다음과 같이 나눌 수 있다[1]:
  - Safety-critical system
  - Hardware vendors, hardware-embedded applications
  - Enterprise applications

[1] Magic Quadrant for Static Application Security Testing
Gartner RAS Core Research Note G00164100, Joseph Feiman, Neil MacDonald
6 February 2009

분석 시장 현황

- Quality/security testing, stati
  다음과 같이 나눌 수 있다[
  - Safety-critical system
  - Hardware vendors, hardware-embedded applications
  - Enterprise applications



[1] Magic Quadrant for Static Application Se
Gartner RAS Core Research Note G00164100, Josep Neil MacDonald
6 February 2009

분석 시장 현황

ROSAEC
Polyspace

Quality/security testing, stati
다음과 같이 나눌 수 있다[

- Safety-critical system
- Hardware vendors, hardware-embedded applications
- Enterprise applications

Sparrow
Coverity
Klocwork
Codesonar

Quality testing
Analyzer performance
Static analysis

Security testing
Integration with SLC platforms
Dynamic analysis

agic Quadrant for Static Application Se
Gartner RAS Core Research Note G00164100, Josep Neil MacDonald
6 February 2009

안전한(sound) 분석: 과연 가능할까?

- 전체 프로그램을 구성하는 모든 소스가 온전히 제공되는 경우가 드물다: 수많은 타사(third-party) 라이브러리들
  - `main` 함수부터 분석한다는 것은 거의 불가능
  - 글로벌 영역에 대한 분석 정확도가 매우 낮음
  - 특히 하드웨어 인터럽트 등
- 끊이지 않는 파싱 에러 문제
  - ARM CC, AIX CC, HP-UX CC, MSVC, GCC, GCC계열 크로스 컴파일러들, INTEL CC 등 수많은 개발 환경 지원
- 빌드 과정 이해
  - 매크로나 `#ifdef` 등과 같은 전처리 지시어들을 제대로 처리하는 문제

너무나 큰 소스코드

- 휴대폰 소스의 경우 매우 작은 모델(400만라인)도 리눅스 커널 소스보다 큼(300만라인)
  - 단일 파일 하나가 엄청 큰 경우도 있음
- 오픈소스처럼 집약적/효율적으로 구성되어 있지 않음: aliasing은 그리 많지 않은데, 변수의 개수는 상대적으로 많음
- 매크로의 과도한 사용: 한 줄의 실제 내용이 몇십줄에 해당함
- 우리에겐 서버급 단일 머신 하나가 주어질 뿐: 클러스터링까지 바라는 것은 아직은 무리

허위경보에 더 민감한 사용자들

- False negative보다는 false positive에 훨씬 더 민감함.
- 사용자는 개발자가 아니라 QA담당자들

매일 매일 조금씩 변하는 코드

- 변한 부분만 간단히 분석해서 결과를 보고 싶은 사용자
  - Make system vs. Incremental analysis
- 모듈러하게 분석하는 구조 필요
- 전체 분석과 부분 분석의 일관성 중요: 특히 재귀함수
- 같은 알람을 판별해 내는 기술 필요: 사용자가 허위경보라고 체크해 놓은 것이 계속 유지되도록

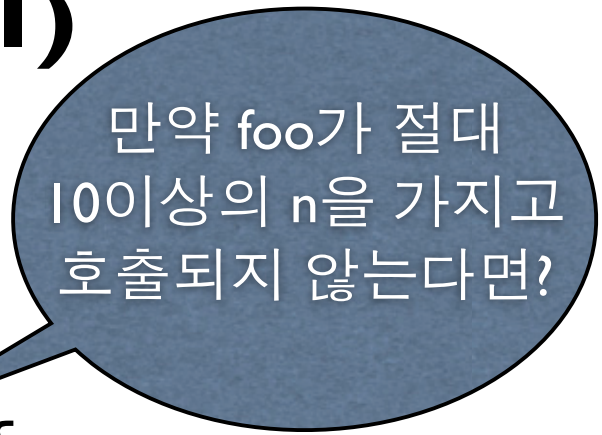
의미있는 오류란?

- 일부러 오류를 내는 코딩 이디엄들: 성능 및 구조 상의 이유로
- 지금 당장은 오류가 아니지만 나중에 위험한 것들
- 오류가 발생할 가능성이 있지만 그 가능성이 희박한 것들: 예외처리 등

의미있는 오류란? (예제 I)

```
1 void foo(int n) {
2   int buf[10];
3   if (n > 10) return;
4   buf[n] = ...;
5 }
```

의미있는 오류란? (예제 1)

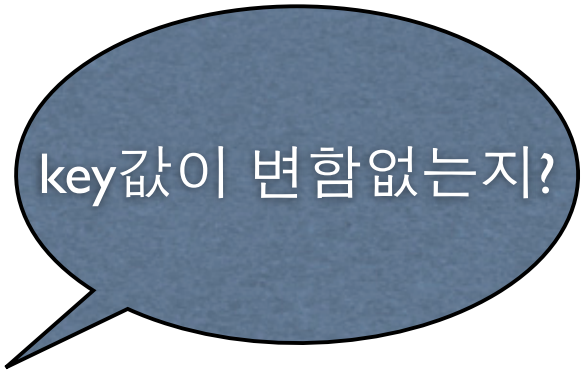



```
1 void foo(int n) {
2   int buf[10];
3   if (n > 10) return;
4   buf[n] = ...;
5 }
```

의미있는 오류란? (예제 2)

```
if (getValue(key)) p = malloc(...);
...
if (getValue(key)) free(p);
```

의미있는 오류란? (예제 2)

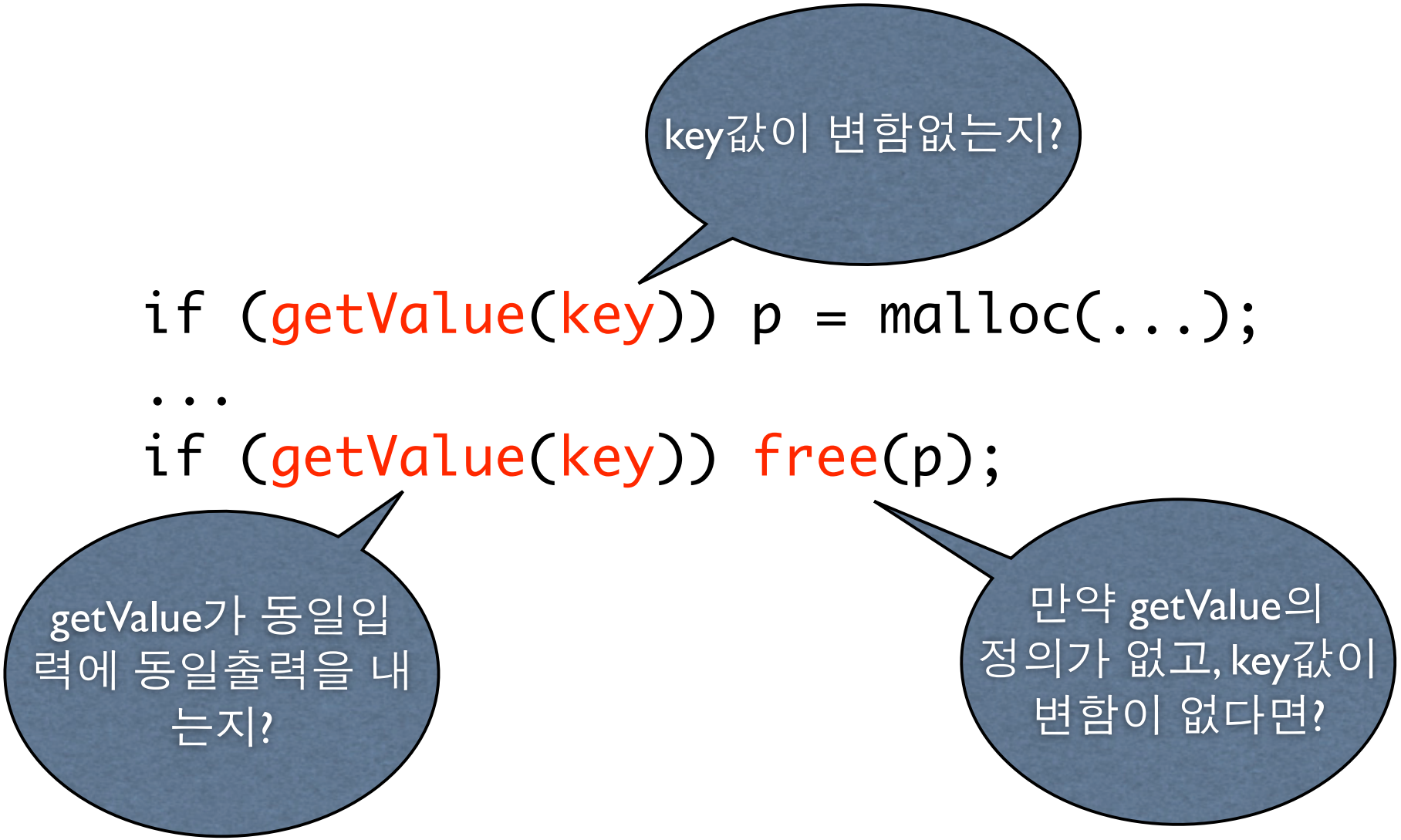



```
if (getValue(key)) p = malloc(...);
...
if (getValue(key)) free(p);
```

의미있는 오류란? (예제 2)


```
if (getValue(key)) p = malloc(...);
...
if (getValue(key)) free(p);
```

의미있는 오류란? (예제 2)

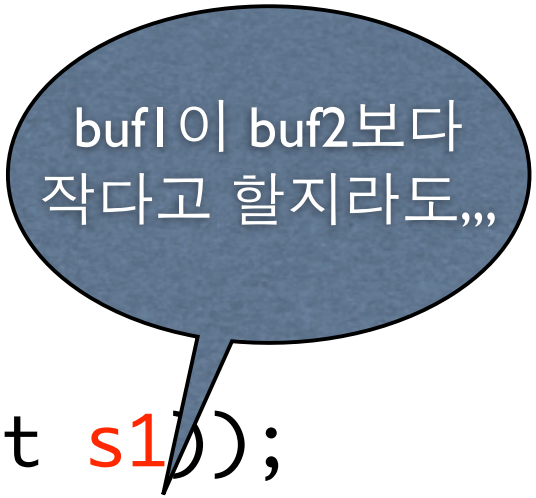

key값이 변함없는지?
if (getValue(key)) p = malloc(...);
...
if (getValue(key)) free(p);
getValue가 동일입력에 동일출력을 내는지?
만약 getValue의 정의가 없고, key값이 변함이 없다면?

의미있는 오류란? (예제 3)

```
1 struct s1 buf1;
2 struct s2 buf2;
3 bzero(buf1, sizeof(struct s1));
4 bzero(buf2, sizeof(struct s1));
```

의미있는 오류란? (예제 3)

```
1 struct s1 buf1;
2 struct s2 buf2;
3 bzero(buf1, sizeof(struct s1));
4 bzero(buf2, sizeof(struct s1));
```



의미있는 오류란? (예제 3)

```
1 struct s1 buf1;
2 struct s2 buf2;
3 bzero(buf1, sizeof(struct s1));
4 bzero(buf2, sizeof(struct s1));
```

buf1이 buf2보다 작다고 할지라도,,,

초기화루틴인지 아닌지?

알람 설명

- Collecting semantics는 그래프 형태로 보여주는 것이 좋음
  - 그렇지 않을 경우 패스를 나누어 하나씩 보여줘야 함
  - 다 보여주거나 대표적인 것 하나만
- 알람 발생의 직접적 원인에 해당하는 메모리 영역의 값을 그렇게 만드는데 영향을 준 모든 구문들 표시
  - Data dependency와 control dependency를 구분해서 표시
  - 직접적인 영향을 준 것과, 간접적인 영향을 준 것을 구분해서 표시
- 값의 변화에 전혀 영향을 주지 않은 구문들도 표시
- 쉬운 알람부터 보여주는 것도 중요함

허위 경보 걸러내기

- 분석기에 내장시키는 방법
  - 분석을 많이 하면 할수록 낮은 우선순위 부여
    - 분석이 진행된 경로가 길수록, 알람의 원인지점과 발생지점 사이의 거리가 멀수록, 그 사이에 알람의 발생과 관련된 영역의 주소값이 많이 사용될수록 (특히 함수의 인자로 넘어갈수록) 낮은 우선순위 부여
  - 알람의 원인이 분석기가 사용한 요약방식(abstraction)과 관련있을 경우 낮은 우선순위 부여
  - 특정 한가지 원인으로 인한 알람이 너무 많을 경우 낮은 우선순위 부여
- 사용자와의 상호작용을 통한 방법
  - 외부 라이브러리 정의 (scaffolding)
  - 소스 코드 주석 (annotation)
  - 비슷한 알람 판별해 내기: 알람 필터링 기술 필요

실용적인 분석기가 되려면„

- 분석 엔진 자체
  - 엔진을 최대한 말랑말랑하게
  - 특정 사이트에 민감하지 않게
  - 성능 개선의 길을 충분히 열어두는 것이„
- 분석 결과 후처리
  - 알람 분류를 쉽게
  - 허위 경보 제거를 쉽게
  - “사용자에게 유용한” 알람들을 쉽게 추출할 수 있도록