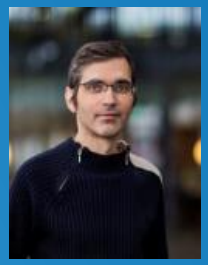
TATA CONSULTANCY SERVICES
Experience certainty.

Repositioning of Static Analysis Alarms

Tukaram Muske
TRDDC,
Tata Consultancy Services,
Pune, India
t.muske@tcs.com

Rohith Talluri
TRDDC,
Tata Consultancy Services,
Pune, India
rohith.talluri@tcs.com

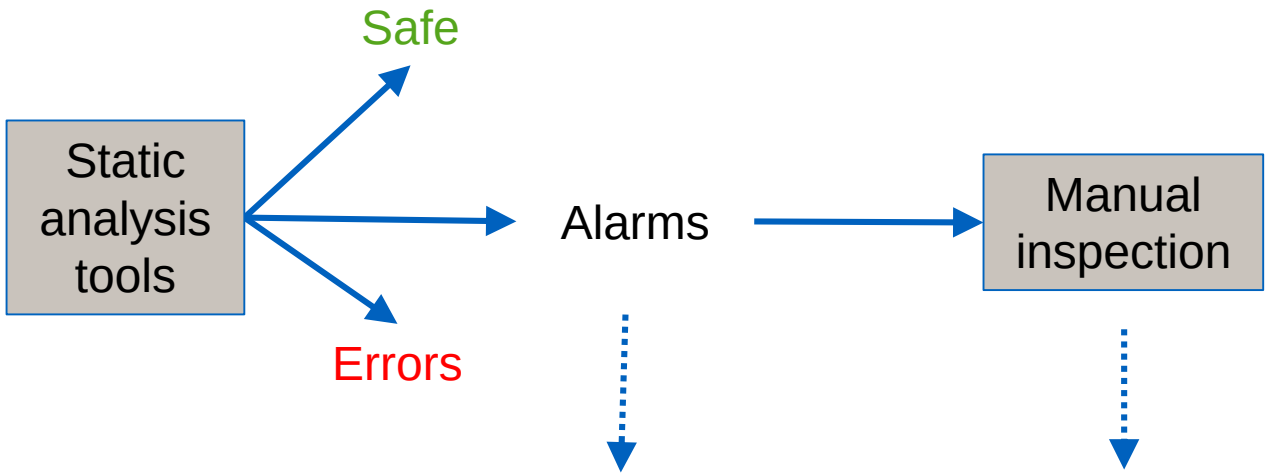


Alexander Serebrenik
Eindhoven University of Technology,
Eindhoven, The Netherlands
a.serebrenik@tue.nl

27th ACM SIGSOFT International Symposium on Software Testing and Analysis
Amsterdam, 16 - 18 July, 2018

Introduction

Usage of Static analysis tools



The problem we address

Tools underused in practice [1,2]

A very large number alarms

High effort in code traversals

We propose –
Repositioning of alarms (2 goals)

Primary Goal:
To reduce the number of alarms reported

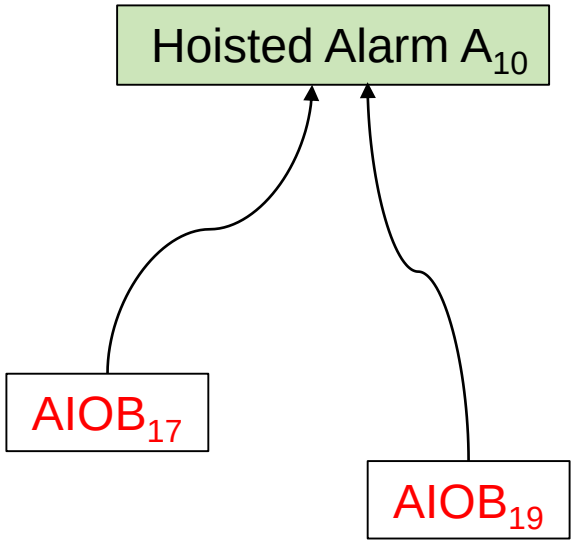
Secondary Goal:
To report alarms closer to their causes

[1] Maria Christakis and Christian Bird. What Developers Want and Need from Program Analysis: An Empirical Study. In ASE 2016.
[2] Brittany Johnson, Yoonki Song, Emerson Murphy-Hill, and Robert Bowdidge. Why Don't Software Developers Use Static Analysis Tools to Find Bugs? In ICSE 2013

Motivating Example - 1

- State-of-the-art grouping techniques fail to group the example AIOB alarms

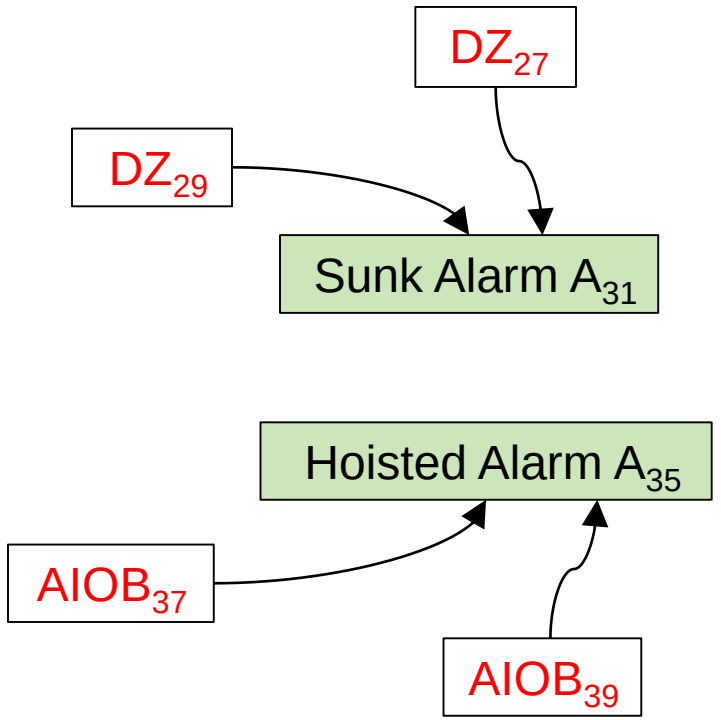
```
1.    void foo(){
2.        int arr[5], tmp = 1, i = 0;
3.
4.        if(...){
5.            if(...){
6.                i = lib1();
7.            }else{
8.                i = lib2();
9.            }
10.           //assert(0 ≤ i ≤ 4);
11.           tmp = 0;
12.       }else{
13.           tmp = lib3();
14.       }
15.
16.       if(i < tmp){
17.           arr[i] = 0;
18.       else
19.           arr[i] = 1;
20.   }
```



Motivating Example - 2

- Analysis for detecting Division by Zero (DZ) and AIOB errors

```
21.   void bar(int t1){
22.       int n, j, arr[5], tmp = 1;
23.
24.       n = lib1();
25.       if(...){
26.           n = lib2();
27.           t1 = t1 / n;
28.       }else{
29.           t1 = 10 / n;
30.       }
31.       //assert(n != 0);
32.       tmp = 0;
33.
34.       j = lib2();
35.       //assert(0 ≤ j ≤ 3);
36.
37.       t1 = arr[j];
38.       j++;
39.       tmp = arr[j];
40.   }
```



Repositioning Technique

- Two pass static analysis technique
  - Pass 1 - Backward analysis
    - Computes anticipable alarm conditions with related alarms
    - Obtains intermediate Repositioning (Secondary goal achieved)
  - Pass 2 - Forward analysis
    - Computes available alarm conditions with related alarms
    - Refines the intermediate Repositioning (Primary goal achieved)
- Maintains traceability links to original alarms

Intermediate Repositioning (Pass 1)

- Reposition anticipable condition <u>at a program point before which it is not anticipable</u>

```
1.    void foo(){
2.    int arr[5], tmp = 1, i = 0;
3.
4.       if(...){
5.         if(...){
6.            i = lib1();
              //assert(0 ≤ i ≤ 4);
7.         }else{
8.            i = lib2();
              //assert(0 ≤ i ≤ 4);
9.         }
10.
11.        tmp = 0;
12.      }else{
           //assert(0 ≤ i ≤ 4);
13.         tmp = lib3();
14.      }
15.
16.      if(i < tmp){
17.        arr[i] = 0;
18.      else
19.        arr[i] = 1;
20.   }
```

$$\text{Hoist}_{\text{exit(n)}} = \{\, c \mid c \in \text{Kill}_n(\text{AntOut}_n), \text{DepGen}_n(\{c\}) = \emptyset \,\}$$

0 ≤ i ≤ 4 is *anticipable* at *exit*(n_6) but not at *entry*(n_6)

0 ≤ i ≤ 4 is *anticipable* at *exit*(n_8) but not at *entry*(n_8)

$$\text{Hoist}_{\text{entry(n)}} = \text{AntIn}_n \setminus \bigcup_{m \in \text{pred(n)}} \text{AntOut}_m$$

0 ≤ i ≤ 4 is *anticipable* at *entry*(n_{13}) but not at *exit*(n_4)

Repositioning Refinement

- Use available condition at a program point after which it is not available

```
1.    void foo(){
2.    int arr[5], tmp = 1, i = 0;
3.
4.       if(...){
5.         if(...){
6.            i = lib1();
              //assert(0 ≤ i ≤ 4);
7.         }else{
8.            i = lib2();
              //assert(0 ≤ i ≤ 4);
9.         }

11.       tmp = 0;
12.       }else{
13.                  tmp = lib3();
14.       }

          ...
20.   }
```

Available alarm conditions with their associated values

$0 \le i \le 4 \rightarrow 0 \le i \le 4,\ exit(n_6)$

$0 \le i \le 4 \rightarrow 0 \le i \le 4,\ exit(n_8)$

$0 \le i \le 4 \rightarrow 0 \le i \le 4,\ entry(n_{11})$

Used for the final repositioning

$0 \le i \le 4 \rightarrow 0 \le i \le 4,\ entry(n_{11})$

- Post-processing of the repositioned alarms (more details in the paper)

Evaluation and Observations

- Experimental set up (primary goal)
  - 16 open source [1,2] and 4 industry applications (totaled over 450 KLOC)
  - Alarms for AIOB, DZ, OFUF, and UIV properties (total 33162)

- Observations
  - Alarms reduction by up to 20%, median 7.25%, and average 6.47%
  - Performance overhead 17.6%
  - Out of 31016, 1443 inter-func repositioning, 176 inter-func mergings
  - 60% cases, backward repositioning stopped due to conditions

- Future Work
  - Study manual effort reduction
  - Identify irrelevant conditions to improve repositioning further

[1] Dalin Zhang, Dahai Jin, Yunzhan Gong, and Hailong Zhang. Diagnosis-Oriented Alarm Correlations. In APSEC 2013.
[2] Woosuk Lee, Wonchan Lee, and Kwangkeun Yi. Sound Non-statistical Clustering of Static Analysis Alarms. In VMCAI 2012

Summary

Problem

- A very large number alarms generated
- Manual inspection of alarms requires performing code traversals

Our approach

- Reposition the alarms

(1) to reduce their count, and (2) report them closer to cause points

Technique

- Two pass analysis technique
- First obtains intermediate repositioning, and later refines it

Evaluation

- Reduction up to 20%, median reduction 7.25%
- Scope to improve the repositioning further