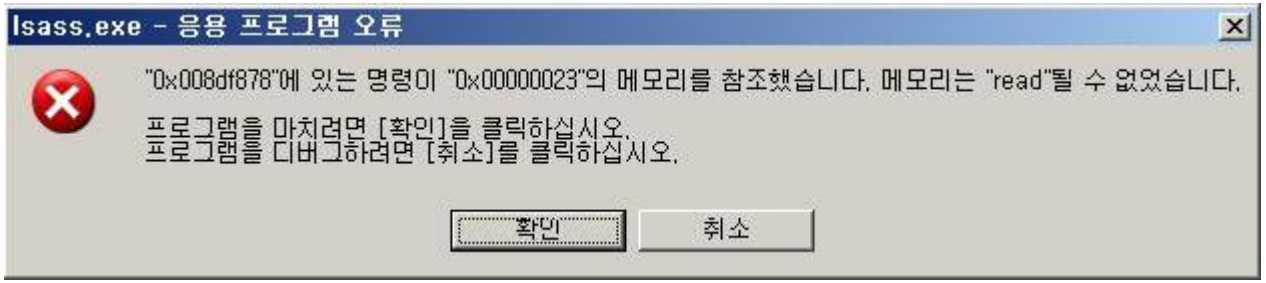
[ROSAEC 3rd Workshop]

Filtering Redundant Alarms in Static Buffer-Overflow Analysis

Youil Kim
<youil.kim@arcs.kaist.ac.kr>

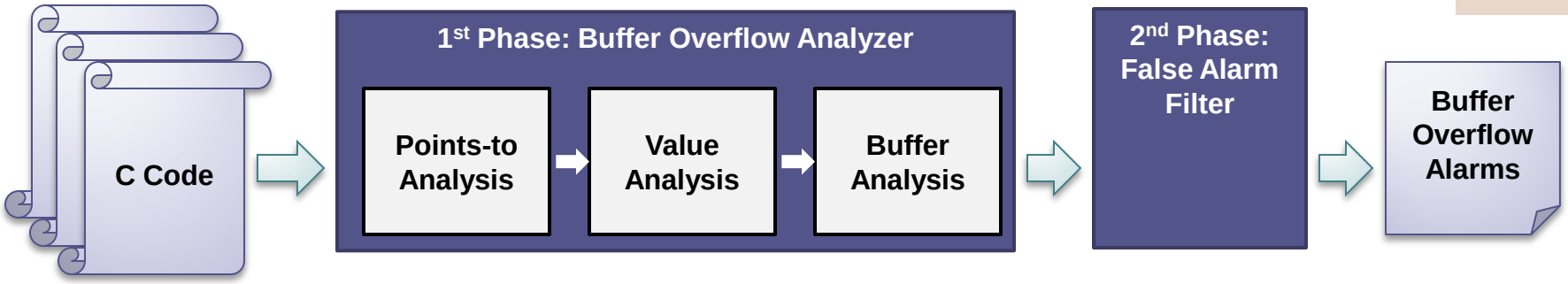
Department of CS, KAIST

Introduction



- Buffer overflows are often in C programs, and some of them remain even after extensive tests.
- A static analyzer can be a powerful tool to find buffer-overflow errors in C programs.

Two Phase Buffer-Overflow Analysis

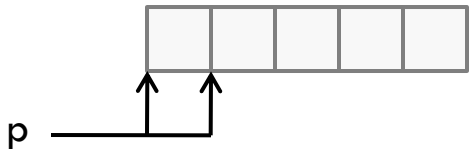


$$\begin{array}{lcl} \mathrm{Mem_v} & : & \mathrm{AbsLoc} \mapsto \mathrm{Interval} \\ \mathrm{Interval} & = & \mathrm{Integer} \times \mathrm{Integer} \end{array}$$

$$\begin{array}{lcl} \mathrm{Mem_p} & : & \mathrm{AbsLoc} \mapsto \mathrm{Pointer} \\ \mathrm{Pointer} & : & \mathrm{BufSize} \times \mathrm{PtrOffset} \\ \mathrm{BufSize} & = & \mathrm{Interval} \\ \mathrm{PtrOffset} & = & \mathrm{Interval} \end{array}$$

Buffer Analysis Example:

p = {size: [5, 5], offset: [0, 1]}



State Refinements at Array Accesses

```
01:  position_set grps[256];

02:  MALLOC(grps[ngrps].elems, position, d->nleaves);
03:  grps[ngrps].nelem = 1;
04:  grps[ngrps].elems[0] = pos;
```

A code snippet from dfa.c in Grep 2.5.1

ngrps = [-oo,+oo]

ngrps = [0,255]

State Refinement:

$0 \leq ngrps < sizeof(grps)$

State Refinements at Pointer Dereferences

```
01:  while (*optarg && *optarg >= '0' && *optarg <= '9')
02:    val = val * 8 + *optarg++ - '0';
```

A code snippet from ftp.c in Wu-ftpd 2.6.2

optarg = {offset:[-oo,+oo], size:[0,+oo]}

State Refinement:
$0 \leq \text{offset} < \text{size}$

optarg = {offset:[0,+oo], size:[0,+oo], safe:[0,0]}

It provides a limited form of relational analysis:
offset + safe $<$ size

Experimental Result

- State refinements at array and pointer accesses remove 27% of alarms from 12 target programs.
- Analysis time increases less than 4%.
- The target programs are from BugBench and GNU system software.

Shan Lu et al., BugBench: Benchmarks for Evaluating Bug Detection Tools, In Workshop on the Evaluation of Software Defect Detection Tools, 2005.

Correctness

- If an alarm is a false positive:
  - It is sound to assume only states which do not cause buffer overflows.
  - State refinements improve the accuracy.
- If an alarm indicates a possible error:
  - It is possible that some of hidden alarms still remain after the error correction.

Error Correction Example

```
01:  int Score[10];
02:  int Age[10];

/* i = [0, 10] */

03:  Score[i] = score; /* Cause a buffer overflow */

/* i = [0, 9] by State Refinements */

04:  Age[i] = age;
```

- Two choices for the error correction:
  - Correcting the value of the index variable – the refinement is valid.
  - Correcting the size of the array in the declaration.

Conclusion

- We introduce state refinements at buffer overflows, which are analogous to compilers' error recovery techniques.
- Our approach effectively removes redundant alarms in our fast buffer-overflow phase; it can remove 27% of alarms.

The Effect of State Refinements on # alarms

Software	# Accesses	Before	After	% Reduced
polymorph-0.4.0	28	27	21	22%
ncompress-4.2.4	82	49	39	20%
man-1.5h1	357	313	215	31%
gzip-1.2.4	697	416	359	14%
bc-1.0.6	1,322	1,263	834	34%
tar-1.13	2,210	1,735	1,220	30%
099.go	9,843	8,972	6,888	23%
129.compress	92	64	62	3%
sed-4.0.8	3,479	2,327	1,516	35%
grep-2.5.1	1,976	1,410	1,054	25%
bison-1.875	5,024	2,357	1,854	21%
wu-ftpd-2.62	2,404	1,978	1,208	30%
Total	27,514	20,911	15,270	27%

The Effect of State Refinements on Time

Software	SLOC	Before	After	% Increment
polymorph-0.4.0	1,357	0.05s	0.05s	0.0%
ncompress-4.2.4	2,195	1.06s	1.07s	0.9%
man-1.5h1	7,232	8.63s	8.76s	1.5%
gzip-1.2.4	11,213	35.53s	36.11s	1.6%
bc-1.0.6	12,830	141.59s	146.83s	3.7%
tar-1.13	21,891	779.68s	781.31s	0.2%
099.go	47,337	966.46s	871.69s	-9.8%
129.compress	5,585	0.95s	0.96s	1.1%
sed-4.0.8	18,687	363.91s	327.00s	2.2%
grep-2.5.1	20,843	153.19s	154.22s	0.7%
bison-1.875	31,203	201.47s	207.25s	2.9%
wu-ftpd-2.62	18,071	609.31s	618.62s	1.5%