Linux kernel developer responses to static analysis bug reports

Philip J. Guo
Computer Systems Laboratory
Advisor: Prof. Dawson Engler

Computer Forum Poster Session
April 15, 2009

Objectives

Learn how programmers use static code analysis tools and suggest ways to make these tools more effective

Methodology

- Quantitative
  - 2,125 bug reports in **Linux kernel** from Coverity static code analysis tool
  - Source control revision history (GIT)
- Qualitative
  - Email questionnaire
  - Bug database and mailing list messages

Which reports are more likely to be triaged?

Reports from certain bug checkers

Reports in younger files

Reports in smaller files

Rank reports by likelihood of triaging

Bug checker types

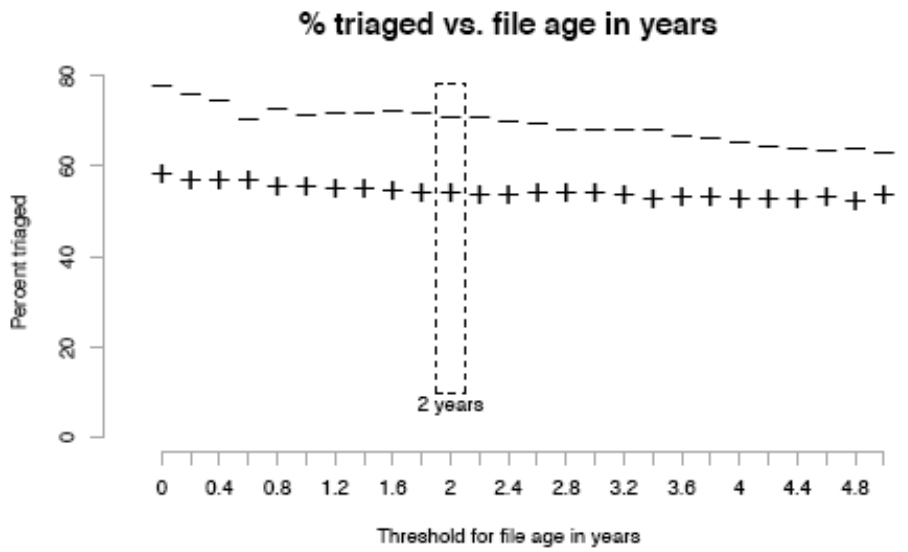
Checker type	# reports	% triaged	FP rank	Notes
dynamic buffer overrun	6	100%	10	security-critical
read of uninitialized values	64	86%	8	lead to non-deterministic bugs
dead code	266	82%	7	could indicate deep logic errors
static buffer overrun	288	79%	5	security-critical
unsafe use before negative test	13	69%	4	
type/allocation size mismatch	5	60%	12	
unsafe use before null test	256	57%	11	
resource leak	302	54%	9	e.g., memory, file handles
null pointer dereference	505	51%	6	
unsafe use of null return value	153	50%	1	inter-procedural
use resource after free	225	49%	2	e.g., memory, file handles
unsafe use of negative return value	42	38%	3	inter-procedural
Total	2,125	61%		

More critical bugs, more triaged

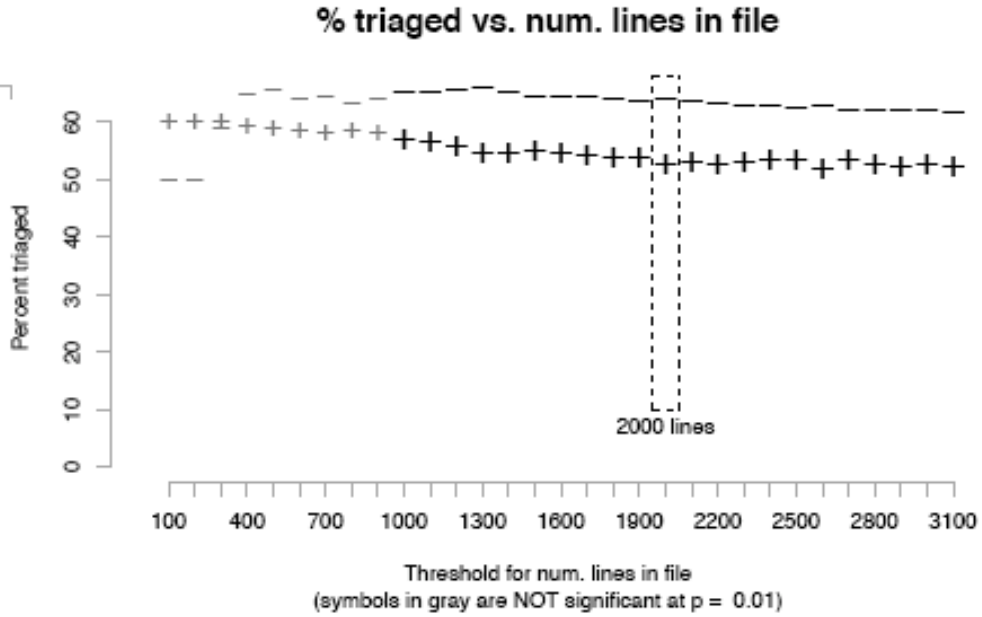
Easier to diagnose, more triaged

Fewer false positives, more triaged

File age and size

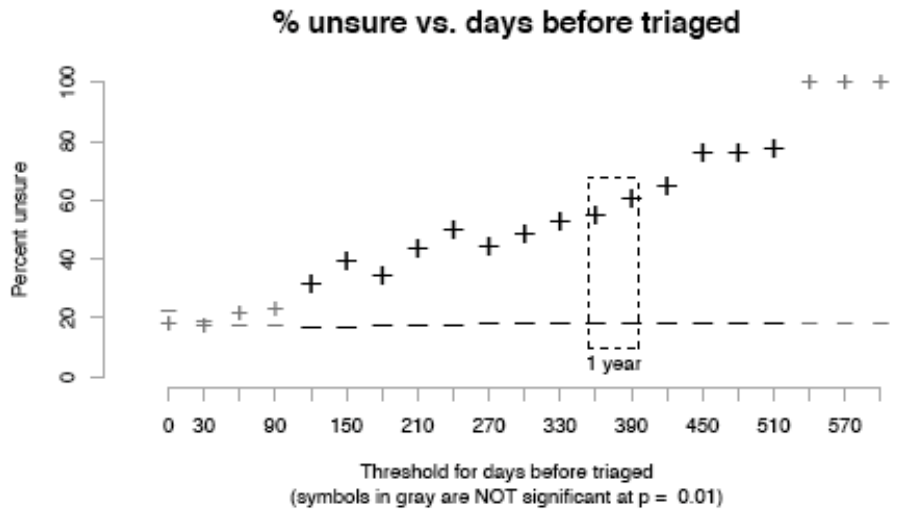


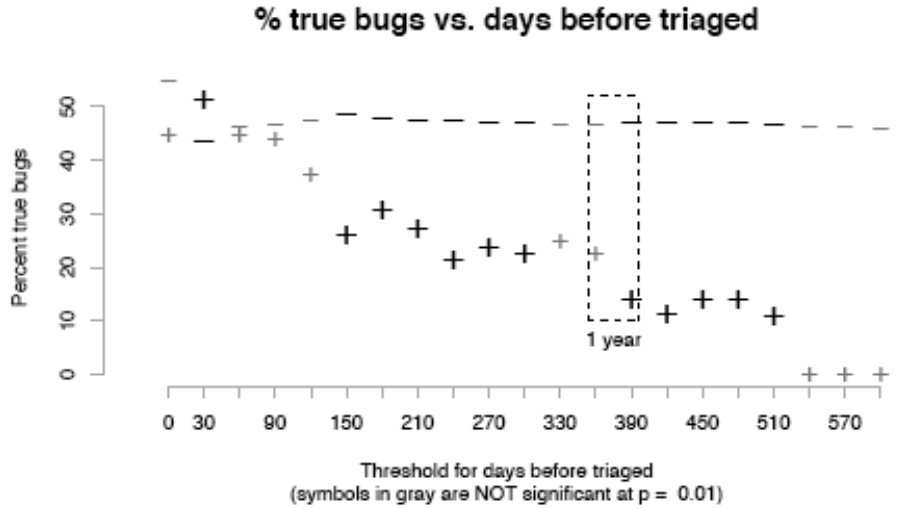
Younger files more actively maintained



Smaller files have less complex code

Triage quickly or forget





Developers first tackle easiest bugs

Useful to continuously run tools and present fresh results to developers

Triaged in	1 month	3 months	6 months	1 year
$<$	8.6%	8.7%	8.8%	8.4%
$\geq$	7.5%	6.3%	2.7%	2.9%

Percent of bugs fixed, of those triaged within and outside given time periods

Quick bug fixes harmful?

Bugs found by static analysis usually easy to fix, but might indicate deeper problems:

> *"Considering the very important flow of patches you are sending these days, I have to admit I am quite suspicious that you don't really investigate all issues individually as you should, but merely want to fix as many bugs as possible in a short amount of time. This is not, IMVHO [in my very humble opinion], what needs to be done."* - from developer mailing list

Triaging subsequent reports in the same file

Probability of triaging reports in a file during one scan, given what happened to reports in previous week's scan

What happened to reports in prev. scan:	Pr(triage)
0 reports triaged	50%
$\geq$ 1 reports triaged	59%
$\geq$ 1 marked true bug	67%
$\geq$ 1 marked true bug and fixed	80%
$\geq$ 1 marked false positive	56%
unconditional probability	54%

(only counting files with reports in at least 2 scans)

Static analysis bugs vs. user-reported bugs

Static analysis bug: *null pointer dereference on Line 36 of sound_driver.c*

User-reported bug: *Sound Blaster card emits weird tone when playing demo.wav*

Static analysis can flag dubious code that is more likely to have user-reported bugs

# Coverity reports	# bugfix patches per file	# bugfix patches per directory
total	**0.27**	**0.56**
triaged	0.23	0.53
un-triaged	0.20	0.46
false positives	0.15	0.42

Spearman's rank correlations

Static analysis bugs predict user-reported bugs

Files in initial scan with:	# files	Time elapsed since initial scan on Feb 24, 2006				
		1 month	3 months	6 months	1 year	lifetime
		Percent of files containing fixes for user-reported bugs				
no Coverity reports	7,504	4%	9%	17%	35%	45%
$\geq$ 1 reports	633	13%	24%	39%	55%	66%
$\geq$ 1 triaged reports	444	14%	25%	41%	58%	68%
$\geq$ 2 reports	197	17%	28%	45%	65%	75%
		Mean number of fixes for user-reported bugs per file				
no Coverity reports	7,504	0.06	0.12	0.27	0.61	0.98
$\geq$ 1 reports	633	0.17	0.38	0.72	1.35	2.17
$\geq$ 1 triaged reports	444	0.18	0.40	0.75	1.44	2.32
$\geq$ 2 reports	197	0.28	0.63	1.06	1.86	2.79

(counting all .c files alive during initial scan)

Making static analysis tools more effective

- Rank and filter reports by likelihood of being triaged
- Encourage finding deeper root causes rather than quick fixes
- Direct attentions to code more likely to have user-reported bugs