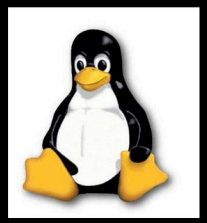
Linux Kernel Developer Responses to Static Analysis Bug Reports

Philip J. Guo and Dawson Engler
Stanford University

USENIX Annual Technical Conference
June 18, 2009



Questions

1. Which static analysis bug reports do developers **actually look at**?
2. How are triaged reports **clustered**?
3. Are static analysis bugs actually **meaningful**?

Methodology

- Quantitative
  - 2,125 bug reports in Linux kernel from static code analysis tool (Coverity)
  - Source control revision history (BK & GIT)
- Qualitative
  - Email questionnaire

Which static analysis bug reports do developers actually look at?

Which bug reports do developers actually look at?

1. depends on checker type

Checker type	# reports	% triaged	relative FP
dynamic buffer overrun			
read of uninitialized values			
dead code			
static buffer overrun			
unsafe use before negative test			
type/allocation size mismatch			
unsafe use before null test			
resource leak			
null pointer dereference			
unsafe use of null return value			
use resource after free			
unsafe use of negative return value			
Total			

Which bug reports do developers actually look at?

1. depends on checker type

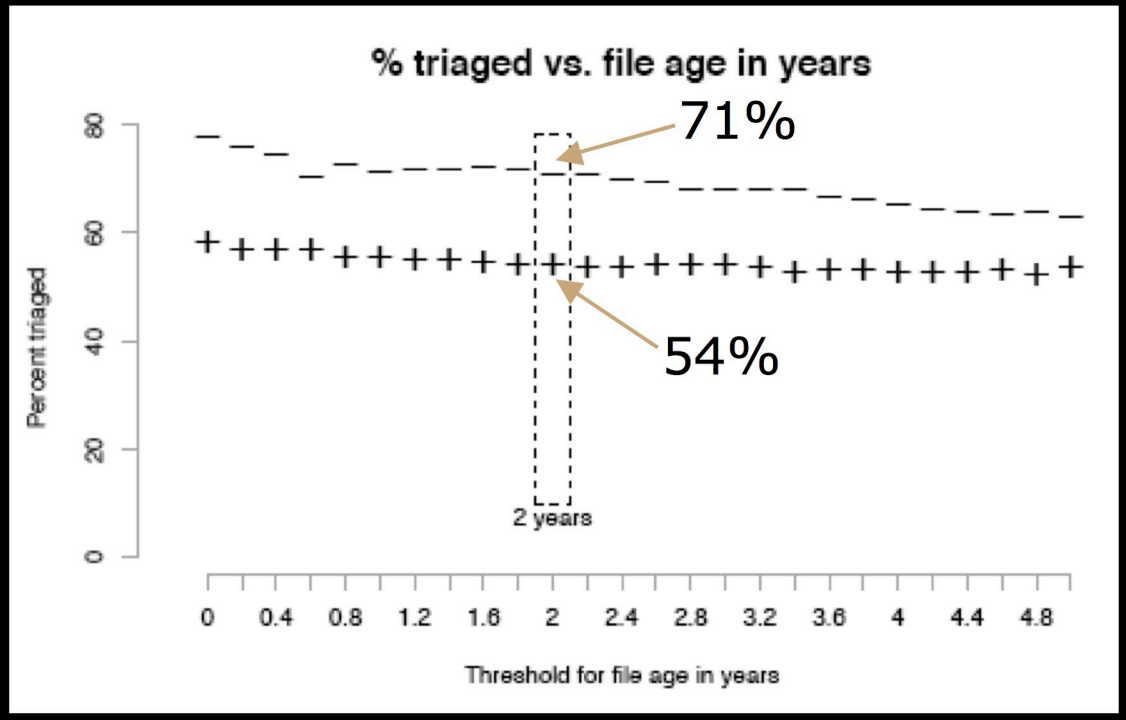
Checker type	# reports	% triaged	relative FP
dynamic buffer overrun	6	100%	3
read of uninitialized values	64	86%	5
dead code	266	82%	6
static buffer overrun	288	79%	8
unsafe use before negative test	13	69%	9
type/allocation size mismatch	5	60%	1
unsafe use before null test	256	57%	2
resource leak	302	54%	4
null pointer dereference	505	51%	7
unsafe use of null return value	153	50%	12
use resource after free	225	49%	11
unsafe use of negative return value	42	38%	10
Total	2,125	61%	

Most critical bugs

Most false positives

Which bug reports do developers actually look at?

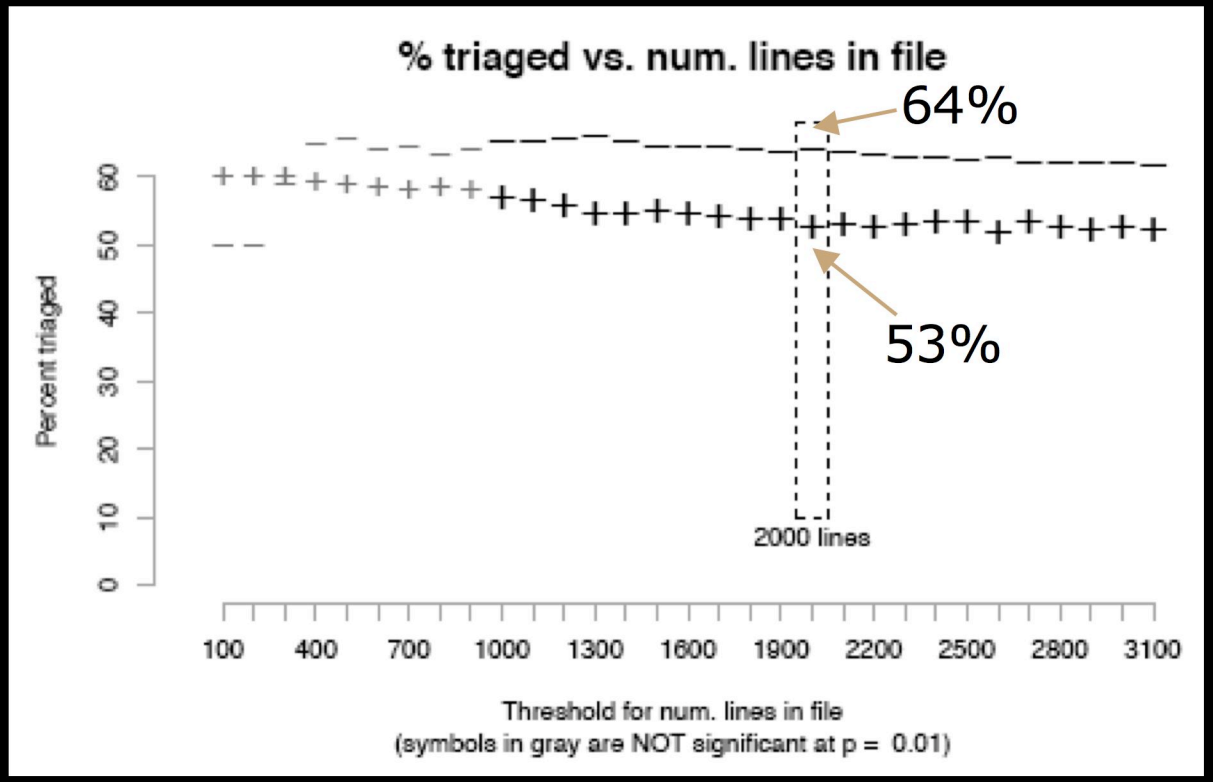
2. reports in younger files



"More often the people involved in creating those [younger] files will still be active kernel developers, and still interested in the area those files cover."

Which bug reports do developers actually look at?

3. reports in smaller files



"Possibly, perhaps due to the buried in warnings syndrome. Perhaps also because smaller files are easier to modify."

How are triaged reports clustered?

How are triaged reports clustered?

1. clustered in space

given:	Pr(all reports **triaged**)
unconditional	46%
$\geq$ 1 reports triaged	65%
$\geq$ 2 reports triaged	87%

} Triage all reports

How are triaged reports clustered?

1. clustered in space

given:	Pr(all reports **triaged**)
unconditional	46%
$\geq$ 1 reports triaged	65%
$\geq$ 2 reports triaged	87%

Triage all reports

given:	Pr(all reports **un-triaged**)
unconditional	30%
$\geq$ 1 reports un-triaged	55%
$\geq$ 2 reports un-triaged	79%

Triage no reports

How are triaged reports clustered?

2. clustered in time

What happened to reports in prev. scan:	Pr(triage)
0 reports triaged	50%

(counting all files with reports in >1 scan)

How are triaged reports clustered?

2. clustered in time

What happened to reports in prev. scan:	Pr(triage)
0 reports triaged	50%
≥ 1 reports triaged	59%

(counting all files with reports in >1 scan)

"triaging bug reports can be quite intimidating [...] Once a developer has got some confidence up in a subsystem they are more likely to step up to the plate and triage again."

How are triaged reports clustered?

2. clustered in time

What happened to reports in prev. scan:	Pr(triage)
0 reports triaged	50%
≥ 1 reports triaged	59%
≥ 1 marked true bug	67%
≥ 1 marked true bug and fixed	80%

(counting all files with reports in >1 scan)

"triaging bug reports can be quite intimidating [...] Once a developer has got some confidence up in a subsystem they are more likely to step up to the plate and triage again."

How are triaged reports clustered?

2. clustered in time

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

(counting all files with reports in >1 scan)

"triaging bug reports can be quite intimidating [...] Once a developer has got some confidence up in a subsystem they are more likely to step up to the plate and triage again."

"False positives tend to lower the maintainer's trust of the tool and are more likely then to let future reports from the same tool slip."

Are static analysis bugs actually meaningful?

Are static analysis bugs actually meaningful?

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*

Are static analysis bugs actually meaningful?

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	∞	entire lifetime
		Percent of files containing fixes for user-reported bugs					
no Coverity reports	7,504						
$\geq$ 1 reports	633						
$\geq$ 1 triaged reports	444						
$\geq$ 2 reports	197						

(counting all .c files alive during initial scan)

Are static analysis bugs actually meaningful?

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	∞	entire lifetime
		Percent of files containing fixes for user-reported bugs					
no Coverity reports	7,504	4%					
$\geq$ 1 reports	633	13%					
$\geq$ 1 triaged reports	444	14%					
$\geq$ 2 reports	197	17%					

(counting all .c files alive during initial scan)

Are static analysis bugs actually meaningful?

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	∞	entire lifetime
		Percent of files containing fixes for user-reported bugs					
no Coverity reports	7,504	4%	9%	17%	35%	45%	69%
$\geq$ 1 reports	633	13%	24%	39%	55%	66%	92%
$\geq$ 1 triaged reports	444	14%	25%	41%	58%	68%	92%
$\geq$ 2 reports	197	17%	28%	45%	65%	75%	96%

(counting all .c files alive during initial scan)

Are static analysis bugs actually meaningful?

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	∞	entire lifetime
		Percent of files containing fixes for user-reported bugs					
no Coverity reports	7,504	4%	9%	17%	35%	45%	69%
$\geq$ 1 reports	633	13%	24%	39%	55%	66%	92%
$\geq$ 1 triaged reports	444	14%	25%	41%	58%	68%	92%
$\geq$ 2 reports	197	17%	28%	45%	65%	75%	96%
		Mean number of fixes for user-reported bugs per file					
no Coverity reports	7,504	0.06					
$\geq$ 1 reports	633	0.17					
$\geq$ 1 triaged reports	444	0.18					
$\geq$ 2 reports	197	0.28					

(counting all .c files alive during initial scan)

Are static analysis bugs actually meaningful?

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	∞	entire lifetime
		Percent of files containing fixes for user-reported bugs					
no Coverity reports	7,504	4%	9%	17%	35%	45%	69%
$\geq$ 1 reports	633	13%	24%	39%	55%	66%	92%
$\geq$ 1 triaged reports	444	14%	25%	41%	58%	68%	92%
$\geq$ 2 reports	197	17%	28%	45%	65%	75%	96%
		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	2.8
$\geq$ 1 reports	633	0.17	0.38	0.72	1.35	2.17	7.4
$\geq$ 1 triaged reports	444	0.18	0.40	0.75	1.44	2.32	7.8
$\geq$ 2 reports	197	0.28	0.63	1.06	1.86	2.79	9.4

(counting all .c files alive during initial scan)

Are static analysis bugs actually meaningful?

*"Coverity and similar tools are a true opportunity for us to find out and **study suspect parts of our code**. Please do not misuse these tools! The goal is NOT to make the tools happy next time you run them, but to actually fix the problems, once and for all. If you focus too much on fixing the problems quickly rather than **fixing them cleanly**, then we forever lose the opportunity to clean our code, because the problems will then be hidden."*

Conclusions: How to make static analysis tools more effective

1. Rank and filter reports by likelihood of being triaged
2. Deeper analysis for important code
3. Use to direct attention to code more likely to contain user-reported bugs
4. Encourage finding deeper root causes rather than quick fixes

"The kernel is such a big project that triaging bug reports can be quite intimidating. Once a developer has got some confidence up in a subsystem they are more likely to step up to the plate and triage again."

"It's horrible, but after looking deeper, including looking at the callers, I'm now convinced it's correct (this code only gets called in 64bit kernels where longs are double the size of ints)."

"I have looked at a few coverity defects and skipped over them because a) they looked too hard to diagnose b) They looked like false positives but I didn't have enough knowledge about the code to be positive"

“Many maintainers have an inbox-is-todo-list mentality when it comes to bugfixes. If they receive a scan report and don’t act on it quickly then it’s likely it’s left the inbox and left the maintainer’s thoughts forever.”

"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, what needs to be done."