Using .debug_line for optimized probe regions with kprobe

Preliminary results

Olivier Dion

Polytechnique Montréal
Dorsal laboratory

Summary

Summary

How kprobe works

- INT3 at probe region

How kprobe works

- INT3 at probe region
- Possible optimization with jump
  - No interruption means less overhead

How kprobe works

- INT3 at probe region
- Possible optimization with jump
  - No interruption means less overhead
- Limitations
  - Regions must be at least five bytes (x86)
  - Instructions of the region must be executed out of line
  - No jump can be made into the region
  - **No indirect jump instruction in the function**
  - No exception thrown by the function

Using DWARF's .debug_line

- DWARF defines the line program in .debug_line

Using DWARF's .debug_line

- DWARF defines the line program in .debug_line
- Interpreted by a consumer (e.g. GDB, kprobe)

Using DWARF's .debug_line

- DWARF defines the line program in .debug_line
- Interpreted by a consumer (e.g. GDB, kprobe)
- The program generates a matrix
  - Where rows are instructions
  - Where columns are attributes

Using DWARF's .debug_line

- DWARF defines the line program in .debug_line
- Interpreted by a consumer (e.g. GDB, kprobe)
- The program generates a matrix
  - Where rows are instructions
  - Where columns are attributes
- The basic_block attribute
  - A boolean indicating that the current instruction is the beginning of a basic block
  - Can improve the indirect jump limitation of kprobe

Summary

Hypothesis

For the check of jumps into a potential optimized region of a kprobe's probe, the usage of the basic_block attribute of the DWARF line program yield considerably better success rate than the current check of indirect jumps by kprobe.

Summary

Methodology

- Tool using Dyninst
  - Emit per function f
    - Number of instructions I_f
    - Number of instructions K_f currently optimizable by kprobe
    - Number of instructions B_f optimizable using the line program
  - Only check for indirect jump in functions
  - Only emit for functions with more than 1 instruction
  - Does not work on a kernel image

Methodology

- Tool using Dyninst
  - Emit per function f
    - Number of instructions I_f
    - Number of instructions K_f currently optimizable by kprobe
    - Number of instructions B_f optimizable using the line program
  - Only check for indirect jump in functions
  - Only emit for functions with more than 1 instruction
  - Does not work on a kernel image
- A corollary of this is that $K_f = B_f$ *or* $K_f = 0$
  - Which means our results are **overestimated**

Methodology

- Tool using Dyninst
  - Emit per function f
    - Number of instructions I_f
    - Number of instructions K_f currently optimizable by kprobe
    - Number of instructions B_f optimizable using the line program
  - Only check for indirect jump in functions
  - Only emit for functions with more than 1 instruction
  - Does not work on a kernel image
- A corollary of this is that $K_f = B_f$ *or* $K_f = 0$
  - Which means our results are **overestimated**
- We use the following formulas

$$Overall\ gain = \frac{\sum B_f - K_f}{\sum I_f}$$

$$Average\ gain\ per\ function = \frac{1}{N} \sum \frac{B_f - K_f}{I_f}$$

Summary

Results

Executable	kprobe success rate (%)	Our success rate (%)	Overall gain (%)	Average gain per function (%)
firefox	68.63	86.22	17.59	3.35
gcc	72.66	87.02	14.37	2.81
guix	74.39	87.15	12.75	2.65
kcachegrind	76.07	87.28	11.21	3.96
lttng	67.75	85.64	17.89	4.73
Average	71.90	86.66	14.76	3.50

Summary

Conclusion

- We've estimate the gain of using the line program for kprobe
  - It's an overestimation
  - We don't know the memory usage cost of the line program

Conclusion

- We've estimate the gain of using the line program for kprobe
  - It's an overestimation
  - We don't know the memory usage cost of the line program
- The line program can be consumed by different agents
  - LTTng
  - GDB
  - Dynamic instrumentation of uftrace from our other works

Conclusion

- We've estimate the gain of using the line program for kprobe
  - It's an overestimation
  - We don't know the memory usage cost of the line program
- The line program can be consumed by different agents
  - LTTng
  - GDB
  - Dynamic instrumentation of uftrace from our other works
- What lies ahead?
  - Merge the basic block attribute of .debug_line in GCC
  - Reproduce the experiment
  - Check the memory usage overhead
  - Integrate with tracing tools