DTrace based on BPF and tracing facilities: Challenges

dr. Kris Van Hees
Languages and Tools
Linux Engineering
<kris.van.hees@oracle.com>
DTrace and BPF

• D code compiled into BPF functions
• Dynamic generation of trampoline BPF programs
• Pre-compiled function library (C-to-BPF)
• Built-in linker to generate standalone BPF programs
• D supports local, global, and TLS variables
• D supports arrays, aggregations, dynamic variables, string functions, alloca/bcopy/...
Origins

- Bleeding edge functionality is cool
- Bleeding edge functionality solves many problems
- Production systems don’t run bleeding edge kernels
- Real life use cases usually originate on production systems
We can’t...

- ... tell customers to upgrade their systems
- ... tell customers to wait for new features
- ...
- ...
- ...
- ...
- ...

- ... tell customers to ignore reality!
Problem: Spill register to stack

- Store constant value from reg to stack
- Load it back → constant value in reg
- Store known (bounded) value from reg to stack
- Load it back → unknown value in reg

Fixed on Jul 13, 2021, first appeared in v5.14-rc4
Solution: Spill register to stack

- Store known (bounded) value from reg to stack
- Load it back → unknown value in reg
- Insert explicit bounds check(s) on the reg
  - Conditional jumps provide this info to the verifier
  - But be careful….!!!
Problem: branch prediction bug

- Conditional branch comparing %rD against %rS
- %rD = bounded value, %rS = constant value
- Prediction is attempted and bounds are updated
- %rD = constant value, %rS = bounded value
- No prediction is attempted and bounds are updated incorrectly

No patch for this problem in bpf-next (yet)!
Problem: branch prediction bug

[...]
BPF: 185: frame1: R0=invP0 R1_w=invP24 ...
R5=invP(id=0,umin_value=17,umax_value=20,var_off=(0x10; 0x7)) ...
BPF: 185: (3d) if r1 >= r5 goto pc+10
BPF: 240 frame1: R0=invP0 R1_w=invP24 ...
R5=invP(id=0,umin_value=25,umax_value=20,var_off=(0x10; 0x7)) ...
BPF: 186: frame1: R0=invP0 R1_w=invP24 ...
R5=invP(id=0,umin_value=25,umax_value=20,var_off=(0x10; 0x7)) ...
[...]
Solution: branch prediction bug

I will submit a patch for it this week.
Problem: resource limits

- Tracing scripts can get pretty complex
- String manipulation functions
- `alloca()`, `bcopy()`
- Associative arrays
- Dynamically allocated variables
- Need more memory than the stack provides
Problem: resource limits (cont.)

- BPF map, singleton element, large value size
- Use value as addressable memory
- Limitations:
  - Verifier cannot validate anything stored/loaded
  - Values are plain integers (can’t use as pointers)
  - Limited space (KMALLOC_MAX_SIZE)
Solution (?) - resource limits

• **Option 1:** Allow BPF maps with larger value size
• **Option 2:** Use multiple map values
  – A form of paged memory (map value is like a page)
  – Cumbersome (ptr + offset + addr translation vs ptr)
• **Option 3:** New (per-cpu) memory resource
  – Does not need to be visible to userspace
  – Large (bounded) size – needs to be preallocated
  – Possible bpf_malloc() / bpf_free() helper support?
Other issues...

- Complex scripts and functions need loops
  - More complex invariant state detection needed
  - Invariant relations between values in registers

- Why are the CPU registers (pt_regs) not accessible from the BPF context for some program types?

- Compilers (LLVM, GCC) generating code that is valid but cannot be validated by the verifier

- Userspace validation → signed BPF programs?