Using Address Watchpoints

Instrument data, not just code

Ashvin Goel
University of Toronto

Advanced Host-Level Security (AHLS)
Dec 10, 2014

Project Goal

- Goal is to protect operating system kernels against buggy module/driver code

- What types of bugs are we interested in?

Types of Bugs

- Bug detection
  - Memory bugs
    - Use-after-free, read-before-write, double-free
    - Buffer overflow detectors, memory leak detector
  - Concurrency (race, atomicity) bugs
  - Direct memory access (DMA) bugs
  - Semantic bugs
    - Object-specific invariant violations, access pattern violations

- Performance anomalies
  - False sharing detector

Approach

- Instrument all module code at runtime using Dynamic Binary Translation (DBT)
  - Rewrite module code during execution
  - Provides complete control over module execution
  - Built a prototype system called Granary
    - Think "Valgrind", but for the Linux kernel
- What about writing bug detectors using DBT?

Problems with Existing DBT Systems

- Instruments code at instruction level
  - Wrong abstraction, tools need to instrument data accesses
- All code is instrumented
  - High overhead, limits heavy instrumentation
- Hard to use
  - Have to deal with tricky instructions, worry about re-entrancy, safety, maintain illusion that DBT is not there

Ideally, We Want

- Data-centric instrumentation
  - You tell the hardware what objects your tool cares about
  - The hardware tells your tool when the objects is accessed
- Selective instrumentation
  - Otherwise, no instrumentation overhead
- High-level instrumention
  - Provide high-level API that handles concurrency, safety

Solution: Address Watchpoints

- Key insight
  - Hard to track objects, easy to track addresses!
  - Taint the address of “interesting” objects so that accesses to them always raise a fault, hence “address watchpoints"

- Address watchpoints
  - Relies on x86-64 48-bit address implementation in which 16 high-order bits are "free" to be changed
  - Kind of like getting a segfault when you read a bad pointer
  - On fault, use the tainted bits to identify what

Example

```
struct sk_buff *skb = alloc_skb(skb_size,
                                GFP_KERNEL);
// skb == 0xFFFFFFFFA092600
skb = add_watchpoint(skb, <meta-data>);
// skb == 0x7654FFFFA092600

...



dma_map_single(…,   skb->data,      , …);



do_general_protection(regs)
... regs->regs[...] == 0x7654FFFFA0926E0
```

Isn’t this slow?

Selective Instrumentation

- Approach
  - Take fault on first access to watched address
  - Turn on DBT
  - Turn off DBT when watched addresses are not expected to be accessed

- Benefits
  - Avoids faults on each watched addresss
  - Provides efficiency by taking advantage of locality of watched accesses
  - No overhead when watched addresses are not accessed

Initial Implementation

- Implemented address watchpoints using Granary DBT system [HotDep 2013]

- Applications
  - Buffer overflow detector
  - Use-after-free, read-before-write
  - Memory leak detector

Current Status

- Implementing Granary+
  - Learning from mistakes exposed by address watchpoints
- Building high-level instrumentation API
  - Tools are still hard to implement using address watchpoints
- Will enable more powerful watchpoint-based tools
  - Races, lock contention, false sharing detector

Example: Instruction Profiling

```
array div_count, div_p2_count

probe insn($opcode == "div") and function {
  div_count[$name]++                 // fn performs div
  if ((@op.2 & (@op.2 - 1)) != 0)
    div_p2_count[$name]++            // fn performs div
                                     // by power of 2
}

probe end {
  for (fname in div_count)
    printf(“%d | %d | %s\n”, div_count[fname],
           div_p2_count[fname], fname)
}
```

Example: Address Watchpoints

```
array accesses // # accesses of target objects
set targets    // handled by watchpoint framework

probe object.alloc and
  function($name == “skb_alloc") {
   add(@start..@end, targets) // track address range
}

probe object.access and
  function ($name =~ “dma_map_single") {
    if (@addr in targets) accesses[targets[@addr]]++
}
```

Conclusions

- Address watchpoints enable data-centric, selective instrumentation
- Initial implementation enabled several debugging tools for kernel modules

- Current Status
  - Reimplemening Granary/watchpoint implementation
  - Building higher-level instrumentation API
    - Will allow integrating tracepoints
  - Will enable more powerful watchpoint tools