Secure Popcorn:

Using Machine Boundaries to Harden Applications

Rob Lyerly
Systems Software Research Group at Virginia Tech

Cambridge, UK
September 11-13, 2017

The Problem

- Current security mechanisms provide inter-process protection or coarse-grained randomization
  - SELinux: “bag of permissions”, who can access which files
  - ASLR: load-time virtual address space randomization
- Many exploits circumvent these mechanisms to co-opt execution & leak information
  - Heartbleed: malicious crypto packets read arbitrary memory from server
  - Rowhammer: flip DRAM bits by “hammering” data cells
  - FLUSH+RELOAD: read memory of co-located processes through shared cache
  - Return-oriented programming (ROP): construct arbitrary executions using buffer overflow and “gadgets” from application code

- How do we provide stronger inter-/intra-process security?

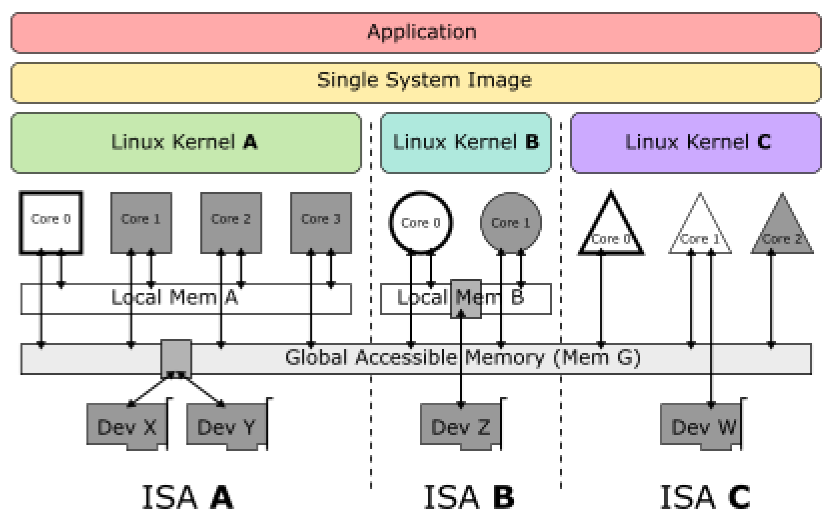
The Impact

- Eliminate several classes of security exploits
  - Information leakage: enforce programmer intent by preventing cross-component memory accesses in the page-fault handler
    - "My image library shouldn't access my crypto data!"
  - Memory side-channel attacks: physically isolate sensitive memory
- Mitigate impact of other security exploits
  - Information leakage: randomize virtual address space during execution to hide application structure from "owned" threads
  - ROP-based attacks: adjust stack layout to destroy "gadgets"

- End-users get security benefits while still being able to write applications using shared-memory programming model
  - Don't have to rewrite applications!

Overview of the approach

- Secure Popcorn: an OS, compiler and runtime for secure application execution
  – Based on Popcorn Linux, a replicated-kernel OS, and ELFbac, a memory access control mechanism
- Per-thread execution migration across machine boundaries
  – Single system image (SSI) across machines, which provides distributed shared memory & file descriptor migration (e.g., filesystem & network interface)
  – Migration between heterogeneous-ISA processors, e.g., ARMv8 and x86-64
- Compiler builds multi-ISA binaries
  – Align code/data symbols across compilations of application for all ISAs
- Runtime performs dynamic state translation for stack & registers between ISA-specific ABIs



Overview of the approach

- Group application components into ELF sections
  - Describes programmer intent, i.e., which code should access which data
  - Use page fault handler to prevent cross-component memory access
- Randomize memory layout during migration
  - Transform stack layout between ABIs – disrupts ROP gadgets
  - Randomize layout of global code/data
    - Memory leak does not reveal address space layout
    - "Owned" thread cannot discover information about other threads
- Isolate sensitive application data across machines
  - Hide sensitive information, e.g., cryptographic keys, in "private rooms"
  - Prevent memory side-channel attacks