Sandboxing: Software Fault Isolation

Scenarios for Sandboxing

- Web Browser Plug-ins
  - Security hole in plug-in compromises browser
  - Impose restrictions on plug-in
- Downloaded executable applications from untrusted sources
- Packet Filter
  - Can use callbacks to copy and filter packets in the application
    - Incurs a lot of overhead from context switching
  - Put filtering logic in kernel
    - Danger of untrusted code running in supervisor mode

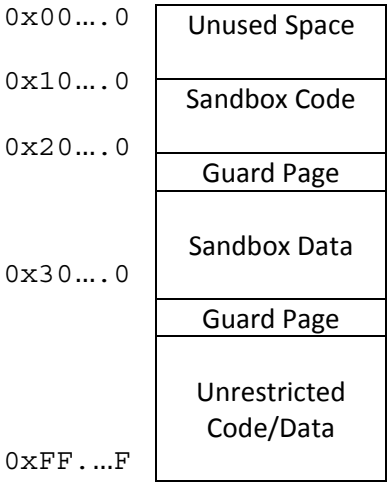
Reference Monitor

- Use Reference monitor for access control between untrusted code and trusted code.
- Policy enforcement is implemented in the reference monitor
- Performance costs aren't as good as expected

Disadvantages of Separate Processes

- IPC overhead is expensive (context switches)
- The new process must still be sandboxed

SFI



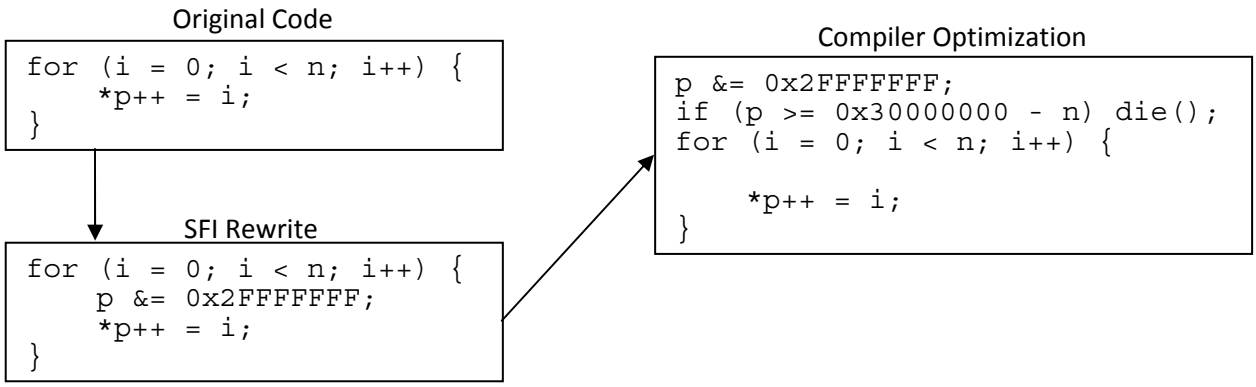
Code Rewriting

1. `mov %eax, (%ebx)`
   a. `and 0x2FFFFFFF, %ebx`
      `mov %eax, (%ebx)`
2. `mov (%ebx), %eax`
   a. No change
3. `jmp 0x12341234`
   a. Check address location statically
4. `jmp (%ebx)`
   a. Chunk code into 16 bytes
      i. Creates fixed length instructions with noop padding
   b. `and 0x1FFFFFFC, %ebx`
      `jmp %ebx`

Reference Monitor in SFI

- How do we access the reference monitor from SFI code?
  - Add reference monitor at fixed address, augment call checks to allow the new address
  - Only allow call (and not jmp) due to prevent fake stack frames
- Reference monitor does not trust the caller
  - Stack might be near a guard page
  - Parameters might cause TOCTOU bugs
- Reference monitor must save the environment and work in a trusted address space
  - Similar to an OS context switch but no syscalls and no page table remapping
- Multithreaded applications must be able to pass data back and forth
  - Data indirection in reference monitor, must become a scheduler also
- End up replicating operating system functionality
- Advantage of using SFI?
  - Memory access protection implemented in software on systems without hardware MMU

Compiler Optimizations



- Indirect jump can go anywhere, including past the address check in the optimized binary.
  - Must keep the address check in the same control execution block as the write
    - Performance impact due to aligning blocks with nops

Ideas Behind SFI

- Very hard to prove that x86 code is safe
- Proving a small subset of x86 code is easy
  - Minimize the trusted computing base
  - Must be able to map all of x86 code into this subset
  - How do we find this small subset?
- Statically verify rewritten code before execution
  - If it is not obviously safe, reject the code