CSci 5271

Navid Emamdoost
navid@cs.umn.edu

Oct-16-2014

UNIVERSITY OF MINNESOTA
Driven to Discover℠

Software-based Fault Isolation

UNIVERSITY OF MINNESOTA
Driven to Discover℠

Need for extensibility

UNIVERSITY OF MINNESOTA

- UNIX vnode interface
  - Add new file system
- Postgres database
  - User-defined data type
- Browser plugins
  - Incorporate plugins (possibly from untrusted sources)

Problem with extensions

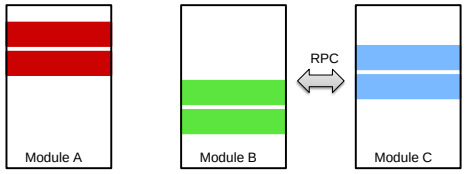
UNIVERSITY OF MINNESOTA

- Security!
- Extensions may be
  - Malicious
  - Vulnerable
  - Faulty
- Solution:
  - Isolate from other codes

Isolation options

UNIVERSITY OF MINNESOTA

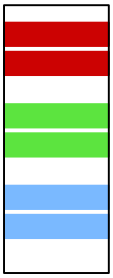
- Hardware-based isolation
  - Different virtual address space
  - Communicate via RPC



Isolation options (cont'd)

UNIVERSITY OF MINNESOTA

- Software-based isolation
  - All modules in same virtual address
  - Protect them from each other
  - Provide an efficient communication



Efficient Software-based Fault Isolation

Robert Wahbe, Steven Lucco, Thomas E. Anderson, Susan L. Graham
SOSP 1993

UNIVERSITY OF MINNESOTA
Driven to Discover℠

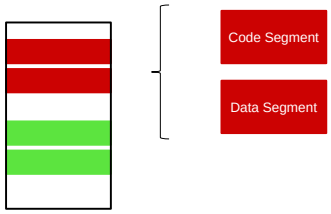
Goal

UNIVERSITY OF MINNESOTA

- Protect the rest of an application from a buggy/malicious module on RISC architecture
- Separate distrusted code
  - Define a fault domain
  - Prevent the module from jumping or writing outside of it
  - While letting efficient communications
- Security Policy:
  - No code is executed outside of fault domain
  - No data changed outside of fault domain

Fault Domain

- Load untrusted extension into its own fault domain
  - Code Segment
  - Data Segment



Segment ID

- Within a segment
  - Addresses share unique pattern of upper bits

Unsafe Instruction

- Jump or store instructions
- Addressing issue
  - jmp 10001e0
- or
  - mov %eax,0x11020028

Unsafe Instruction

- Jump or store instructions
- Addressing issue
  - jmp 10001e0
  - **jmp *%ecx**
- or
  - mov %eax,0x11020028
  - **mov $0x11018b80,%ecx**

Segment Matching

- Insert checking code before unsafe insn
  - check segment ID of target address
- Use dedicated registers

dedicated-reg ⇐ target-address
scratch-reg ⇐ (dedicated-reg >> shift-reg)
if scratch-reg == segment-reg:
    jmp/mov using dedicated-reg

Segment Matching

- Needs 4 dedicated registers
- Checking code must be atomic
- Exact location of fault can be detected
- Runtime overhead
  - 4 extra instructions

Address Sandboxing

- Ensure, do not check!
- Before each unsafe instruction
  - Set upper bit of target address to correct segment ID

dedicated-reg ⇐ target-address & and-mask
dedicated-reg ⇐ dedicated-reg | segment-reg
jmp/mov using dedicated-reg

Address Sandboxing

- Prevents faults
- Needs 5 dedicated registers
- 2 extra instructions
  - less overhead compared to segment matching

Process Resources

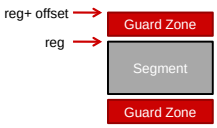
UNIVERSITY OF MINNESOTA

- No direct syscall
- A trusted fault domain receives the syscall
- Determine if it is safe
- If so, make the syscall and return the result to distrusted code

Optimizations

UNIVERSITY OF MINNESOTA

- register-plus-offset mode
  - store value, offset(reg)
    - offset is in the range of -64K to +64K
  - mov %esi,0x8(%edx)



Optimizations

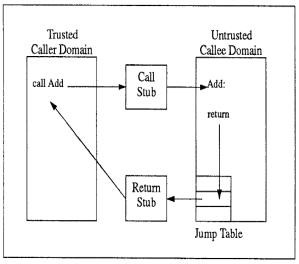
UNIVERSITY OF MINNESOTA

- Stack pointer
  - just sandbox it when it is set
  - ignore sandboxing for small changes
    - push, pop
  - Works because of guard zones

Cross Fault Domain Call

UNIVERSITY OF MINNESOTA

- How to call from another fault domain



Cross Fault Domain Call

UNIVERSITY OF MINNESOTA

- Trusted call/return stub
  - copy parameters
  - switch execution stack
  - maintain values of CPU registers
  - no traps or address space switching
    - faster
  - returns via jump table
    - jump targets are immediates
    - a legal address in target fault domain

Implementation

UNIVERSITY OF MINNESOTA

- Change the compiler
  - emit encapsulation code into distrusted code
- At the load time
  - check the integrity of encapsulation code
  - Verifier

Verifier

UNIVERSITY OF MINNESOTA

- Responsible for checking encapsulation instructions just before execution start
- Challenge:
  - indirect jump
- Hint:
  - every store/jump uses dedicated registers
- Look for changes in dedicated registers
  - any change means beginning of a check region
  - verify the integrity of check region

Verifier

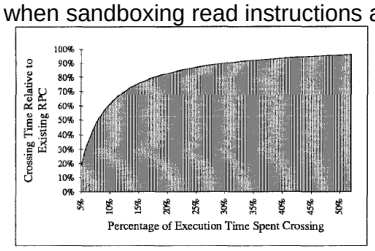
UNIVERSITY OF MINNESOTA

- Divide program into unsafe regions
  - any modification to store dedicated register
    - start of store unsafe region
  - the store unsafe region ends when:
    - next instruction be a store (uses dedicated register)
    - next instruction be control flow change
    - next instruction is not guaranteed to be executed
    - no more instructions be in the code
  - at the end if dedicated register is not sandboxed correctly, reject the code

Performance Overhead

UNIVERSITY OF MINNESOTA

- 4.3% on average
- 21.8% when sandboxing read instructions as well



UNIVERSITY OF MINNESOTA

What about CISC architectures?!
x86

Evaluating SFI for a CISC Architecture
(PittSFIeld)

Stephen McCamant, Greg Morrisett
USENIX 2005

UNIVERSITY OF MINNESOTA
Driven to Discover℠

CISC Architectures

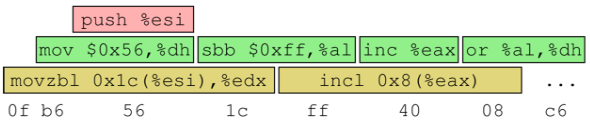
UNIVERSITY OF MINNESOTA

- RISC Architecture
  - Fixed length instructions
  - More CPU registers
- Intel IA-32 (aka x86-32)
  - Variable length instructions
  - Less CPU registers
- Classical SFI is not applicable here

CISC Architectures

UNIVERSITY OF MINNESOTA

- Processor can jump to any byte
- Hard to make hidden instructions safe



Solution

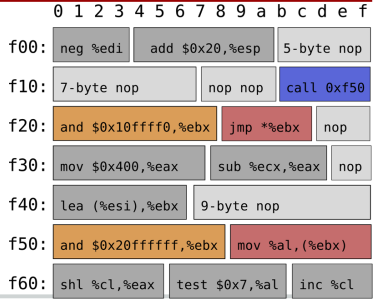
UNIVERSITY OF MINNESOTA

- Alignment
  - Divide memory into 16-byte *chunks*
  - No instruction is allowed to cross chunk boundary
  - Target of jumps placed at the beginning of chunks
  - Call instructions placed at the end of chunk

Alignment

UNIVERSITY OF MINNESOTA

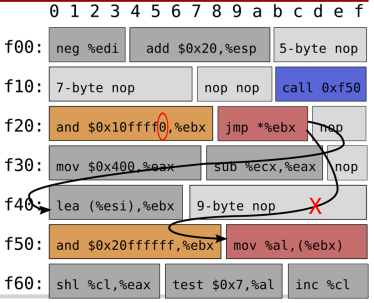
- Use NOP
  - for padding
- No separation of an unsafe instruction and its check



Jumps

UNIVERSITY OF MINNESOTA

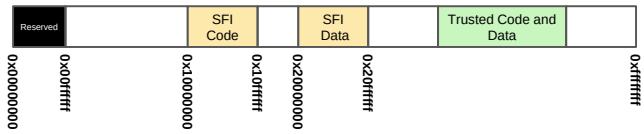
- Chunks are atomic
- Jump destinations are checked to be 16-byte aligned



Optimization: AND-only Sandboxing

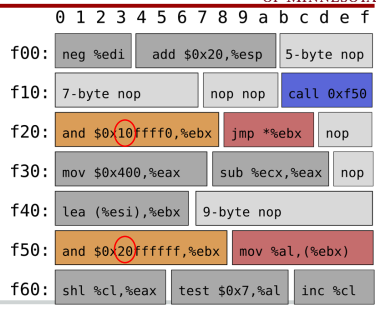
UNIVERSITY OF MINNESOTA

- Reduces sandboxing sequence to just one instruction
  - o choose code and data region addresses carefully
  - o Their ID just has one bit set



Example

UNIVERSITY OF MINNESOTA



Verification

UNIVERSITY OF MINNESOTA

- Statically check
  - o No jump to outside of code region
  - o No store to outside of data region
- Before each unsafe jump or store there should be a sandboxing AND
- The sandboxing AND should not be the last instruction in a chunk

Performance overhead

UNIVERSITY OF MINNESOTA

- Implemented prototype
  - o named PittSFIeld
- Average module overhead: 21%
- But the overall execution can be improved because of faster communications
  - o no trap, RPC, etc

Native-client: A Sandbox for Portable, Untrusted x86 Native Code

Bennet Yee, et al.
IEEE S&P, 2009

UNIVERSITY OF MINNESOTA
Driven to Discover℠

Google Native Client

UNIVERSITY OF MINNESOTA

- Browser Plugin (Google Chrome)
  - o Allows execution of untrusted native code in browser
- Browser?! Native Code?!
  - o Yes! browsers are new platform for applications
- Gives Browser plugins performance of native code
- Ships by default with Chrome 14
- Very complex architecture
  - o Focus on sandboxing technique

Sandboxing

UNIVERSITY OF MINNESOTA

- Inner Sandbox
  - o Like PittSFIeld
  - o Alignment and address sandboxing
    - No cross boundary instructions
    - jump target must be aligned
- Outer Sandbox
  - o Controls system calls issued by native code
  - o Whitelist

Inner Sandbox

UNIVERSITY OF MINNESOTA

- On x86-32 bit architecture
- Use segmented memory to guaranty data sandboxing
- Use 32-byte alignment to sandbox jumps
  - o jump
  - o call
  - o retturn

```
and   $0xffffffe0,%ecx
jmp   *%ecx
```

Outer Sandbox

UNIVERSITY OF MINNESOTA

- Second layer of defense for native code
- Filters system calls
- On linux uses *ptrace*
- Block any sys call not in whitelist
- For some, perform special argument checking
  - SYS_OPEN: can access to a whitelisted set of files
- Any violation from outer sandbox policy will terminate native code execution

Native Client Toolchain

UNIVERSITY OF MINNESOTA

- Modified GCC and GAS
  - To emit sandboxing instructions
- Final executable has .nexe extension
  - compiled and linked as ELF file
- Can be disassembled using standard tools
  - objdump -d

naclcall	%ebx	→	and call	$0xfffffffe0,%ebx *%ebx
nacljmp	%ecx	→	and jmp	$0xfffffffe0,%ecx *%ecx

Performance Evaluation

UNIVERSITY OF MINNESOTA

- Imposes in average %5 overhead
- Sources of overhead
  - Inner sandbox
    - alignment and padding
  - Outer sandbox
    - syscall capturing and whitelisting

Recap

UNIVERSITY OF MINNESOTA

- Sandboxing
  - Execute untrusted code in a fault domain
- RISC
  - Simple instructions
  - Address Sandboxing
- CISC
  - Complex instructions
  - Address alignment
- Browser plugin
  - Benefit native performance in browser

UNIVERSITY OF MINNESOTA

Thank you

Questions?