

Efficient Software-Based Fault Isolation

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

J. Garrett Morris, presenter

Software Extensibility

Operating Systems

- Kernel modules
- Device drivers
- Unix vNodes

Application Software

- PostgreSQL
- OLE
- Quark Xpress, Office

But:

Flaws in extension modules could cause flaws in the entire system

- Crashes
- Data corruption

Hardware Isolation

is slow

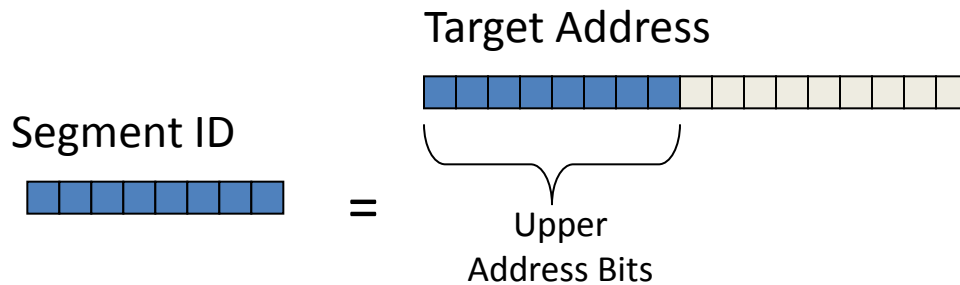
- Traps, address space switches, TLB flushes...
- Performance doesn't necessarily improve with integer performance

Software Isolation

- Load each untrusted module into its own **fault domain**
- Provide **write protection** so that untrusted code can't corrupt data
- **Limit execution** so that untrusted code can't hijack operating system resources or crash containing program

Implementation

- Fault domains are **segments**



- Untrusted code gets **code** and **data** segments
- Write protection
 - Segment matching
 - Address sandboxing

Segment Matching

store using target-address

Becomes:

dedicated-reg <= target-address

scratch-reg <= (dedicated-reg >> shift-reg)

compare scratch-reg segment-reg

trap if not equal

store using dedicated-reg

Address Sandboxing

store using target-address

Becomes:

```
dedicated-reg <= target-address & mask-reg  
dedicated-reg <= dedicated-reg | segment-reg  
store using dedicated-reg
```

Process Resources

- Need to protect file handles, other process resources.
 - Make operating system aware of fault domains
 - Require fault domains to access process resources through RPC

Implementation

Segment Matching

- Four dedicated registers
- Five extra instructions
- Trap indicates exact instruction that caused failure

Address Sandboxing

- Five dedicated registers
- Two extra instructions
- No indication of failure

Optimization

Compiler customization or object patching

Data Sharing

- All data is readable from fault domains
- Pages mapped into multiple fault domains allow cross-fault-domain communication

Cross-Domain RPC

- Generate stubs for interfaces in trusted code.
- Stubs responsible for:
 - Copying arguments
 - Preserving machine state
 - Trapping failures and time-outs
- But no traps or address space switching

Performance

- Encapsulation overhead
- Cross-fault-domain RPC cost
- Effect on user programs

Performance

Sequoia 2000 Query	Untrusted Function Manager Overhead	Software-Enforced Fault Isolation Overhead	Number of Cross-Domain Calls	DEC-MIPS-PIPE overhead (Predicted)
Query 6	1.4%	1.7%	60989	18.6%
Query 7	5.0%	1.8%	121986	38.6%
Query 8	9.0%	2.7%	121978	31.2%
Query 10	9.6%	5.7%	1427024	31.9%

Finis