Efficient Software-Based Fault Isolation

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

Possible Means of Isolating Faults in End-User Extensions

- Using an interpreted language to enable End-User Extensions
- Writing the system in a type safe language such as MODULA-3, tcl, or perl (e.g. *SPIN*).
- Hardware-based fault isolation methods such as setting protection bits in the MMU to restrict write access within the system's address space (e.g. NOOKS).
- Modifying modules themselves to avoid corruption outside of their address space (e.g. SFI).

Problem Description

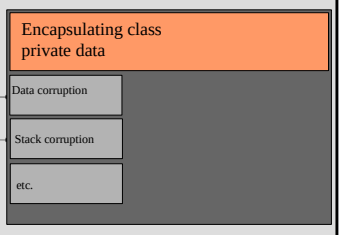
- Extensible applications demonstrate the value of allowing end-users to modify the behavior of the system.
  - Operating Systems
  - Web Browsers
  - Database Systems
- Extensible systems must be protected from possible instabilities in misbehaved end-user extensions.

MISFIT: A Tool for Constructing Safe Extensible C++ Systems
Cristopher Small and Margo Seltzer

Example cross-domain faults (in C++)

```
void unsafe()
{
  char *bad = (char *) this;
  bad -= <arbitrary>;
  memset(bad, 0, 30);
}
```

```
void unsafe()
{
  char name[20];
  memset(name, 0, 300);
}
```



Store & Jump considered only unsafe instructions across domains

Handling cross domain faults

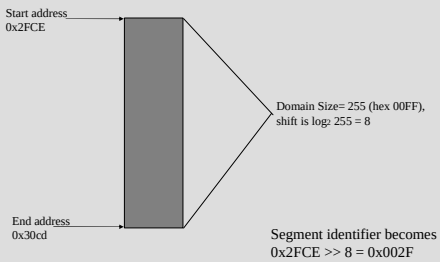
- Place domain data in a contiguous region.
- Ensure that each contiguous region's virtual addresses share a unique prefix.

Detection (segment matching):	Prevention (sandboxing):

dedicated := target
scratch := (dedicated>>shift)
if (scratch == segment)
store to dedicated
else
do error;
	dedicated := target&mask
	dedicated := dedicated| segment
	store to dedicated

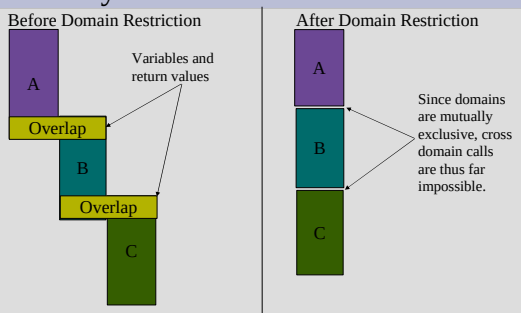
Dedicated, shift, mask, and segment are all dedicated registers.

Example Fault Domain (unaligned)



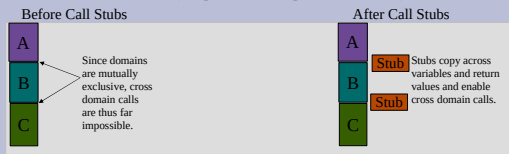
Segment identifier becomes
0x2FCE >> 8 = 0x002F

Protection Domains are Strictly Mutally Exclusive



6

Call Stubs (lightweight RPC)



- Call stubs enable functions to be called across domains by copying data directly from one domain into the other.
- Call stubs set dedicated registers and ensure properly aligned data for representative segment identifiers (context switching and alignment enforcement). (alignment could also be done by the compiler)

7

Hardware vs Software Based Fault Isolation

- Jump or Store Cost
  - Check protection bit in MMU / practically free
- Changing Domains
  - Reset protection bits in the MMU/ flush and reset the TLB

- Jump or Store Cost
  - Addition of a preamble to check the target of the Jump or Store
- Changing Domains
  - Copy data into dedicated registers (5 registers), fairly cheap.

8

Conclusion

- Fault isolation can be implemented in software.
- Software based fault isolation adds a little overhead to the common case.
- Software based fault isolation vastly improves the performance of IPC.
- Applications that cross fault domains a lot benefit a whole lot from software based fault isolation, but even applications that spend very little time crossing fault domains can benefit.

9

Caveats

- SFI is not enough alone when commonly used library functions such as bcopy, strcpy, read, write, close, printf, etc. have not been compiled using the SFI model.
- Safe versions of all commonly used library calls that modify memory must be implemented to avoid breaking the model.
- Safe languages like MODULA-3 may be able to accomplish the same task at nearly the same level with less overhead (but they are not as popular of languages).

10

Evaluation

- SFI could also be extended to provide security by extending isolation enforcement to loads at some additional cost.
- Hardware Based fault isolation cannot benefit from increasing or decreasing the level of security, the dominating cost of reprogramming the MMU and flushing the TLB remains constant regarless of protection type.
- SFI offers varying levels of protection at varying costs, and has fairly low overhead.

11

Questions?

Any questions at all.

Thank you

Thank you very much.