Enforcing Authorization Policies using Transactional Memory Introspection

Enforcing Authorization Policies using Transactional Memory Introspection

Arnar Birgisson†, Mohan Dhawan‡, Ulfar Erlingsson†, Vinod Ganapathy† and Liviu Iftode‡

†School of Computer Science-Reykjavik University, ‡Department of Computer Science-Rutgers University

Transactional Memory Introspection (TMI) Decouples Authorization Policy Enforcement from Program Functionality

Problems with existing Reference Monitors

I. Easy Violation of Complete Mediation
   Current practice:
   • Shared resource accesses identified manually
   • Accesses guarded using if..then..else
   Consequence:
   • Rarely-executed resource accesses are easily overlooked.

   Complete Mediation violation in Linux kernel

   ssize_t vfs.read(struct file *file,...) |
   ... |
   if(check.permission(file, MAT_READ) == ALLOWED) { file->f_op->read(file,...); |
   |
   int page.cache_read(struct file *file,...) |
   struct address_space *mapping = file->i.dentry->i.inode->i.mapping; |
   mapping->a_ops->readpage(file,...);
   ...

II. Time-of-Check to Time-of-Use Bugs
   Current practice:
   • Access checks and resource accesses must happen atomically. Ensured using locks.
   Consequence:
   • Complicates program logic. Error prone.
   • Bugs found in Linux kernel.

III. Exception-handling is Complicated
   Current practice:
   • Authorization failures are handled in an ad hoc fashion.
   Consequence:
   • Error-handling complicates program logic.
   • Accounts for large fraction of code-base.
   • Over two-thirds, according to IBM study
   • Error-handling code itself error-prone

Traditional versus TMI-based policy enforcement

Traditional Approach

Enforcement crosscuts program

TMI-based Approach

Enforcement decoupled from program

TMI Design

Execution

Authorization triggered automatically after transaction validation

Commit

Traditional Approach

TMI-based Approach

Implementation

• TMI reference monitor implemented using Sun’s DSTM2.
• Integrated with both XACML and Java Stack Inspection back-ends
• Enforced authorization policies on 4 servers, comprising a total 55,000 lines of code

Performance Overheads

Future Work

• Formal semantics of TMI: Interaction with STM implementation details.
• Exploring the use of hardware acceleration to reduce STM overheads.
• Using TMI to regulate information flow.