Take home slide

We can utilize the mechanisms of Software Transactional Memory to greatly improve security policy enforcement.
Security policy enforcement

Common difficulties with policy enforcement:

**Difficulty 1**
Time of check vs. time of use

**Difficulty 2**
Correctly handling failures

**Difficulty 3**
Ensuring complete mediation
Policy enforcement difficulty 1
Time of check vs. time of use

if (allowed(principal, resource, operation)) {
    perform operation on resource
}
Policy enforcement **difficulty 1**

Time of check vs. time of use

```java
if (allowed(principal, resource, operation)) {
    perform operation on resource
}
```
if (allowed(principal, resource, operation)) {
    perform operation on resource
}
Policy enforcement difficulty 1
Time of check vs. time of use

if (allowed(principal, resource, operation)) {
    Other thread may run here!
    perform operation on resource
}
if (allowed(principal, resource, operation)) {
    Other thread may run here!
    perform operation on resource
}

Policy enforcement difficulty 1
Time of check vs. time of use

Interleaving code may invalidate the check
Need a synchronization mechanism.
Policy enforcement difficulty 1
Time of check vs. time of use

if (allowed(principal, resource, operation)) {
    perform operation on resource
}
Solution for **difficulty 1**
Software Transactional Memory (STM)

```c
if (allowed(principal, resource, operation)) {
    perform operation on resource
}
```
Solution for **difficulty 1**
Software Transactional Memory (STM)

```java
atomically {
    if (allowed(principal, resource, operation)) {
        perform operation on resource
    }
}
```
Solution for **difficulty 1**
Software Transactional Memory (STM)

```plaintext
atomically {
    if (allowed(principal, resource, operation)) {
        perform operation on resource
    }
}

Uses parallel, speculative execution
```
Solution for difficulty 1
Software Transactional Memory (STM)

atomically {
  if (allowed(principal, resource, operation)) {
    perform operation on resource
  }
}

Uses parallel, speculative execution
Monitors all access to memory
Solution for **difficulty 1**

Software Transactional Memory (STM)

atomically {
    if (allowed(principal, resource, operation)) {
        perform operation on resource
    }
}

Uses parallel, speculative execution

Monitors all access to memory

Can roll back and retry on conflict
Solution for **difficulty 1**

Software Transactional Memory (STM)

```java
atomically {
    if (allowed(principal, resource, operation)) {
        perform operation on resource
    }
}
```

STM guarantees **atomicity** and **isolation** of atomic blocks.
Transactional Memory Introspection (TMI)

Utilizes the mechanisms of Software Transactional Memory — such as its bookkeeping — to greatly improve security policy enforcement.
Where TMI fits in with STM

- TX body
- Contention Mgr.
- Log
- Commit

Flow:
- TX body to Contention Mgr.
- Contention Mgr. to Log
- Log to Commit
- Conflict to Rollback & Retry
- OK from Contention Mgr.
Where TMI fits in with STM

TX body → Contention Mgr. → OK

Log

Conflict

Rollback & Retry

Commit
Where TMI fits in with STM
Where TMI fits in with STM
Policy enforcement difficulty 2
Error handling

if (allowed(principal, resource1, op1)) {
    perform op1 on resource1
} else {
    clean up and report error
}
Policy enforcement **difficulty 2**

Error handling

```java
if (allowed(principal, resource1, op1)) {
    perform op1 on resource1
} else {
    clean up and report error
}

if (allowed(principal, resource2, op2)) {
    perform op2 on resource2
} else {
    clean up after op1;
    clean up after op2 and report error
}
```
 Policy enforcement difficulty 2

Error handling

if (allowed(principal, resource1, op1)) {
    perform op1 on resource1
} else {
    clean up and report error
}

if (allowed(principal, resource2, op2)) {
    perform op2 on resource2
} else {
    clean up after op1;
    clean up after op2 and report error
}

This quickly becomes hard to manage
Policy enforcement difficulty 2
Error handling (cont’d)

- Error handling accounts for a large fraction of server software, over two-thirds [IBM’87]
- Exception handling code itself is prone to errors [Fetzer and Felber ’04]
- SecurityException is the one most often handled incorrectly [Weimer & Necula OOPSLA’04]
Policy enforcement **difficulty 3**

Complete mediation

```java
if (allowed(principal, resource1, op1)) {
    perform op1 on resource1
} else {
    clean up and report error
}

if (allowed(principal, resource2, op2)) {
    perform op2 on resource2
} else {
    clean up after op1;
    clean up after op2 and report error
}
```
Policy enforcement difficulty 3
Complete mediation

if (allowed(principal, resource1, op1)) {
    perform op1 on resource1
} else {
    clean up and report error
}

if (allowed(principal, resource2, op2)) {
    perform op2 on resource2
} else {
    clean up after op1;
    clean up after op2 and report error
}

Easy to forget or miss checks in complex code
Policy enforcement difficulty 3
Complete mediation (cont’d)

• A real problem in current practice

• Bugs of this kind found in the Linux kernel, `page_cache_read` did not check for file permissions [Zhang et al. USENIX Security ‘02]

• Decentralized, ad-hoc hard-coded access checks, leads to errors when code changes.

• Also a problem in Linux [Jaeger et al. ’04]
TMI solves these difficulties and simplifies application code
Handling errors with STM abort

atomically {
    if (allowed(principal, resource1, op1)) {
        perform op1 on resource1;
    } else {
        clean up and report error
    }

    if (allowed(principal, resource2, op2)) {
        perform op2 on resource2;
    } else {
        clean up after op1;
    clean up after op2 and report error
    }
}
Handling errors with STM abort

atomically {
    if (allowed(principal, resource1, op1)) {
        perform op1 on resource1;
    }
    if (allowed(principal, resource2, op2)) {
        perform op2 on resource2;
    }
}
Handling errors with STM abort

atomically {
    if (allowed(principal, resource1, op1)) {
        perform op1 on resource1;
    }
    if (allowed(principal, resource2, op2)) {
        perform op2 on resource2;
    }
} on abort {
    report error;  // no cleanup necessary
}
Complete mediation and decoupling of enforcement code

atomically {
  if (allowed(principal, resource1, op1)) {
    perform op1 on resource1;
  }
  if (allowed(principal, resource2, op2)) {
    perform op2 on resource2;
  }
} on abort {
  report error;  // no cleanup necessary
}
Complete mediation and decoupling of enforcement code

atomically [principal] {
  if (allowed(principal, resource1, op1)) {
    perform op1 on resource1;
  }
  if (allowed(principal, resource2, op2)) {
    perform op2 on resource2;
  }
} on abort {
  report error; // no cleanup necessary
}
Complete mediation and decoupling of enforcement code

atomically [principal] {
    perform op1 on resource1;
    perform op2 on resource2;
} on abort {
    report error; // no cleanup necessary
}
Complete mediation and decoupling of enforcement code

atomically [principal] {
    perform op1 on resource1;
    perform op2 on resource2;
} on abort {
    report error;  // no cleanup necessary
}

The TMI reference monitor is invoked
- on every security-relevant memory access
- before every transaction commit
Complete mediation and decoupling of enforcement code

\[
\text{atomically [principal] \{ }
\]
\[
\text{perform op1 on resource1; ★★}
\]
\[
\text{perform op2 on resource2; ★★}
\]
\[
\text{\}} \text{ on abort} \{ \\
\text{report error; // no cleanup necessary}
\]
\[
\text{\}}
\]

The TMI reference monitor is invoked
- on every security-relevant memory access ★★
- before every transaction commit
Complete mediation and decoupling of enforcement code

atomically [principal] {
    perform op1 on resource1;
    perform op2 on resource2;
}

} on abort {
    report error;  // no cleanup necessary
}

The TMI reference monitor is invoked
- on every security-relevant memory access
- before every transaction commit
Complete mediation and decoupling of enforcement code

atomically [principal] {
    perform op1 on resource1; ⭐
    perform op2 on resource2; ⭐
}

on abort {
    report error; // no cleanup necessary
}

The TMI reference monitor is invoked
- on every security-relevant memory access ⭐
- before every transaction commit ⭐

A TMI authorization manager must make a policy decision — but can do so at any time before commit.
Policy evaluation in TMI

- **TMI authorization manager** evaluates the policy
- Supplied by the programmer, decoupled from application logic
- Invoked on *all* accesses to sensitive resources. *Complete mediation for free.*

before commit of each transaction $T$ {
  for (resource, op) in $T$.log {
    if (not allowed($T$.principal, resource, op)
      abort $T$;
  }
}
Variants of authorization managers
Variants of authorization managers

**Eager**

- TX body
- Contention Mgr.
- Commit

- Validate access
  - Denied
  - Log

- OK
- Conflict

- Abort & Stop
- Rollback & Retry

- Denied
- Log

- OK
Variants of authorization managers

Lazy

Log metadata

TX body → Contention Mgr. → Commit

Validate log

OK → Abort & Stop

Conflict

Rollback & Retry
Variants of authorization managers

Overlapped

Send metadata
TX body
Contention Mgr.
Log
Commit
OK
Conflict
Denied
Abort & Stop
Rollback & Retry
Send decision
Authorization Thread
Implementation

- Extends Sun’s DSTM2 library for Java
  [Herlihy, Luchango and Moir OOPSLA’06]
- Programmer specifies *security metadata*
- Pluggable authorization mgr. receives metadata
- Eager, lazy, overlapped or custom authoriz. mgr.
- Adds less than 500 LOC to DSTM2
Evaluation

**GradeSheet**: 900 LOC, simple enforcement code, 1 atomic block

**Tar**: 5,000 LOC, Java Stack Inspection, 1 atomic block

**FreeCS**: 22,000 LOC, XACML policy enforcement, 47 atomic blocks

**WeirdX**: 27,000 LOC, XACML policy enforcement, 108 atomic blocks
Evaluation

Average execution time of a request

- **STM only**
- **Eager TMI**
- **Lazy TMI**
- **Overlapped TMI**

<table>
<thead>
<tr>
<th>System</th>
<th>STM only</th>
<th>Eager TMI</th>
<th>Lazy TMI</th>
<th>Overlapped TMI</th>
</tr>
</thead>
<tbody>
<tr>
<td>GradeSheet</td>
<td>100</td>
<td>110</td>
<td>120</td>
<td>130</td>
</tr>
<tr>
<td>Tar</td>
<td>150</td>
<td>155</td>
<td>160</td>
<td>165</td>
</tr>
<tr>
<td>FreeCS</td>
<td>200</td>
<td>205</td>
<td>210</td>
<td>215</td>
</tr>
<tr>
<td>WeirdX</td>
<td>250</td>
<td>255</td>
<td>260</td>
<td>265</td>
</tr>
</tbody>
</table>

10.8x improvement in WeirdX compared to STM only.
Evaluation

Average execution time of a request
Evaluation

Average execution time of a request

- GradeSheet: 0.3%
- Tar: -15.8%
- FreeCS: 10.8x

Legend:
- STM only
- Eager TMI
- Lazy TMI
- Overlapped TMI
Evaluation

Average execution time of a request

STM only  Eager TMI  Lazy TMI  Overlapped TMI

GradeSheet  0.3%  -15.8%  4.3%  10.8x
Tar
Evaluation

Average execution time of a request

- STM only
- Eager TMI
- Lazy TMI
- Overlapped TMI

GradeSheet: 0.3%
Tar: -15.8%
FreeCS: 4.3%
WeirdX: 10.8x (11%)
Summary

- TMI is a new reference monitor architecture
- **Decouples application logic from policy enforcement**
- Freedom from TOCTTOU bugs
- Easier handling of authorization failures
- Easier to ensure complete mediation
Take home slide

TMI utilizes the mechanisms of Software Transactional Memory to greatly improve security policy enforcement.
TMI utilizes the mechanisms of Software Transactional Memory to greatly improve security policy enforcement.

Thank you for your attention!