Abstractions for Usable Information Flow Control in Aeolus

Winnie Cheng
Victoria Popic
Dorothy Curtis

Dan R. K. Ports
Aaron Blankstein
Liuba Shrira

David Schultz
James Cowling
Barbara Liskov

MIT CSAIL
Motivation

Confidential information (e.g., financial data, medical records) is increasingly stored online.

Keeping this information secure is a high priority.

However, building secure software remains as difficult as ever.
Financial Management Service Example

Inspired by Mint.com

- users provide service with online banking credentials (username and password)
- system periodically downloads & aggregates transaction info from banks
- presents report to user
Financial Management Service Example

Security requirements

- don’t expose one user’s financial data to another
- bank passwords should only be used to log in to bank; should not even display them to user
Financial Management Service Example

Security requirements

- don’t expose one user’s financial data to another
- bank passwords should only be used to log in to bank; should not even display them to user
Financial Management Service Example

Security requirements

- don’t expose one user’s financial data to another
- bank passwords should only be used to log in to bank; should not even display them to user

Much code needs to be trusted to ensure these

- all application code that handles financial data
- third-party libraries (e.g., to parse data or draw graphs)
Aeolus

Platform for building new secure applications (available as a set of Java libraries)

Uses decentralized information flow control to avoid information leaks
Information Flow Control

Specify restrictions on when information can be released (instead of access control)

- allows untrusted code to access sensitive information but not to release it
- only trusted code allowed to remove restrictions on information flow (“declassify”)

Historically: military IFC systems (confidential, secret, etc.)

Decentralized IFC (DIFC) extends model to many users
Previous DIFC Work

Language-based approaches (e.g., Jif)

- static analysis: IFC enforced through type system, at fine granularity (individual variables)

DIFC operating systems (e.g., Asbestos, HiStar, Flume)

- dynamic: track information flow at the level of processes and files

**Aeolus**: information flow control in **language runtime**

- similar to OS approach, but provides higher-level abstractions
- implemented as library, so doesn’t require new OS or language (tradeoff is larger TCB size than DIFC OSes)
Aeolus Model

“inside” system tracks information flow dynamically

“outside” requires declassification to release info
Aeolus Contributions

New security model
  • designed to be understandable & easy to use
  • represents authority relationships in explicit graph

Programming model
  • abstractions for supporting principle of least privilege
  • threads with secure shared state
  • distributed computation support
Security Model Concepts

Principals: represent users or roles

Tags: categories of data with security requirements
  e.g., ALICE-FINANCIAL-DATA, ALICE-PASSWORD, …

Secrecy label: set of tags
  • objects (files) have immutable labels representing contamination of their contents
  • threads have mutable labels representing contamination of data accessed
Information Flow Rule
Information Flow Rule

Information can only flow to a destination more contaminated than the source
Information Flow Rule

Information can only flow to a destination more contaminated than the source

• Thread T can read object O only if $O.\text{label} \subseteq T.\text{label}$

• Thread T can write object O only if $T.\text{label} \subseteq O.\text{label}$
Information Flow Rule

Information can only flow to a destination more contaminated than the source

- Thread T can read object O only if $O.label \subseteq T.label$
- Thread T can write object O only if $T.label \subseteq O.label$

Thread T can communicate with outside only if $T.label$ is null
Label Manipulations

Thread labels can be changed with two operations

1. Add a tag to thread’s secrecy label
   - allows thread to read contaminated data
   - safe: increases restrictions on thread

2. Declassify: Remove a tag from thread’s label
   - unsafe: allows sensitive data to be released
   - requires authority
Authority

Each thread runs with an associated principal that determines what it can declassify.

Any thread can create a new tag
  • thread’s principal has authority to declassify that tag

Principals can delegate authority to other principals
  • acts-for relationships delegate all authority
  • grants delegate authority for a particular tag
  • either type can be revoked
Authority Structure
Authority Structure
Authority Structure
Authority Structure

![Diagram of authority structure]

- **PAT**
- **PAT-DR**
- **DR-BOB**
- **DR-TOM**
- **CLinic-ADMIN**
- **STATS**

- **pat**
- **PAT-DR**
- **PAT**
- **DR-TOM**
- **CLinic-ADMIN**
- **STATS**

**Tags and Compound Tags**:
- Principal
- Tag
- Compound tag

**Relationships**:
- Acts-for
- Grant
- Subtag
Authority

Aeolus uses explicit authority graph to manage authority

- models common authority relationships
- readily supports modification and revocation
- compare to capabilities as used in DIFC OSes
Programming Model

Abstractions for supporting:

• Principle of least privilege
• Secure sharing between threads
• Distributed computation
Principle of Least Privilege

Needs to be *easy* to drop and regain authority

Two mechanisms:

- **Reduced authority calls:**
  run function with different principal (lower authority)
  *e.g.*, drop all authority when invoking untrusted library

- **Authority closures:**
  Java object bound to principal during construction;
  object methods run with authority of that principal
  *e.g.*, grant authority to code that fetches bank transactions
Threads and Isolation

Each thread has security state: associated principal and secrecy label

Threads must be isolated to ensure information flow obeys label restrictions
- can’t allow threads to share memory directly

Threads can only share data through safe Aeolus mechanisms
- shared objects
- RPCs
- file system
Threads and Isolation

Each thread has security state: associated principal and secrecy label

Threads must be **isolated** to ensure information flow obeys label restrictions

- can’t allow threads to share memory directly

Threads can only share data through safe Aeolus mechanisms

- **shared objects**
- RPCs
- file system

} support distributed applications
(see paper)
Shared Objects

Can be referenced from multiple threads

Each shared object has a secrecy label (like files); Aeolus platform checks labels on access

- Simple built-in example: boxes
  shared objects with a get/put interface

- Developers can define new shared object types; Aeolus adds appropriate label checks
Boxes

Labeled object with get/put interface

```java
Box.get() {
    if (this.label ⊈ thread.label)
        throw InfoFlowException
    return copy(this.contents)
}
```

Allows thread to hold reference to sensitive data without being contaminated by its contents until read
User-Defined Shared Objects

Extending AeolusShared base class causes Aeolus platform to add runtime label check to all methods

class SharedHashTable<T> extends AeolusShared {
    public SharedHashTable(Label label) {
        super(label);
    }

    public T get(String key) {

        return data[key];
    }
}
User-Defined Shared Objects

Extending AeolusShared base class causes Aeolus platform to add runtime label check to all methods

class SharedHashTable<T> extends AeolusShared {
    public SharedHashTable(Label label) {
        super(label);
    }

    public T get(String key) {
        if (thread.label != object.label)
            throw InfoFlowException;
        return data[key] ;
    }
}

User-Defined Shared Objects

Extending AeolusShared base class causes Aeolus platform to add runtime label check to all methods

class SharedHashTable<T> extends AeolusShared {
    public SharedHashTable(Label label) {
        super(label);
    }

    public T get(String key) {
        if (thread.label != object.label)
            throw InfoFlowException;
        return data[key];
    }
}

Aeolus platform can’t tell if method is read-only, so assumes it both reads and writes
User-Defined Shared Objects

Extending AeolusShared base class causes Aeolus platform to add runtime label check to all methods.

class SharedHashTable\<T\> extends AeolusShared {
    public SharedHashTable(Label label) {
        super(label);
    }

    public T get(String key) {
        if (thread.label != object.label)
            throw InfoFlowException;

        return copy(data[key]);
    }
}

Aeolus platform can’t tell if method is read-only, so assumes it both reads and writes.
Implementing Isolation

Rely on memory safety of JVM

- copy all arguments passed to a newly-forked thread
- also, all arguments and result of shared object calls
- needs to be a deep copy, except references to shared objects OK

Disallow unsafe features via Java SecurityManager & bytecode verification

- native code (except in approved libraries)
- reflection
- static variables
Implementing Copying

Need to copy arguments to forks and shared object calls

- can’t use Java’s `Object.clone()`
  (user-provided clone functions might be unsafe)

- serialize to string then deserialize too slow:
  6.3 μs to copy empty obj (much validation, reflection)

- built new cloning library
  - lower-level, optimized: 93 ns per object copied
  - skips copying objects that are safely sharable:
    only contain immutable state or references to shared objs
# Performance (Micro)

<table>
<thead>
<tr>
<th>Call Type</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reduced authority call</td>
<td>51 ns</td>
</tr>
<tr>
<td>(if dropping <em>all</em> authority)</td>
<td>7.7 ns</td>
</tr>
<tr>
<td>Closure call</td>
<td>83 ns</td>
</tr>
<tr>
<td>Shared object call</td>
<td>8.9 ns + 93 ns per object copied</td>
</tr>
<tr>
<td>Java method call</td>
<td>4 ns</td>
</tr>
</tbody>
</table>
Performance (Macro)

Benchmark based on financial management service

- uses reduced authority calls, authority closures, shared state, label manipulations

323 ms/request; Aeolus adds 0.4 ms (0.15%) overhead

Overhead of security operations low in applications that do real work
Conclusion

Aeolus: platform for building secure applications with decentralized information flow control

- simplified DIFC model with explicit authority graph
- abstractions for supporting principle of least privilege: reduced authority calls & authority closures
- isolated threads with secure shared state