Modern Systems: Security

PRESENTED BY ANDREW FERRAIUOLO
CORNELL UNIVERSITY CS 6510
Two Secure Systems Approaches

Fabric (Myers)
- Information flow

Nexus (Sirer & Schneider)
- Credentials-based authentication
- Hardware root-of-trust
Information Flow

Track and constrain the propagation of data throughout the system.
Information Flow Background

Lattice Model (Denning 1976)

Confidentiality and integrity are duals (Biba 1977)

Noninterference (Goguen 1982) – If an attacker can observe data in set L but not H, execution looks deterministic regardless of H.
Applications of Information Flow

Operating Systems
- DStar, HiStar (Zeldovich 2008)

Digital logic
- Gate-level information flow tracking – (Tiwari 2009)

Hardware architecture
- Dynamic information flow tracking – (Suh 2003)

Type Systems
- A lot of work (Survey: Sabelfield 2003)
Hardware Authentication: TPM

Hardware support for providing remote attestation of software

Problems:
- Cannot describe state-dependent trust
- Compromises privacy
- Provides purely axiomatic trust
Trust Bases

**Axiomatic** – A principal is assumed to be trusted
- Trusted platform module

**Synthetic** – Transforms untrusted code into trustworthy code
- Reference monitors, sandboxing

**Analytic** – Predicts whether certain program behaviors are possible
- Proof-carrying code (Necula 1996), information flow!
Credentials Based Authorization

Permit access when a principal satisfies some property (Appel 1999)

\[
P_1, P_2, ..., P_n \quad \frac{}{\mathcal{F}}
\]

Goal Formula: conditions under which resource can be accessed
Credentials: a proof that satisfies a goal formula

Nexus adds features to support all three bases of trust.
- Can Nexus be used to perform information flow analysis? Would you want it to?
Fabric

Goal: Secure distributed computation

Approach:
- *Secure language* – information flow type system enforces noninterference
- *Decentralized security* – multiple distrusting parties, no central authority
Principals

Represent entities (e.g., users, groups, nodes) and privileges.

Acts-for relation: supports delegation, can be used for access control

```plaintext
principal user, priv;

... if (user actsfor priv) {
    ... do action ...
}
```
Labels

Express policies in the code
- labels are associated with data
- describe which principals can act on them and how

Policies are enforced mostly statically at compile-time
- Some checks, e.g. for remote procedure calls are checked dynamically
Labels

Decentralized label model (Myers 2000)
- Written in terms of principals
- Confidentiality \{ alice \rightarrow bob \}
- Integrity \{ alice \leftarrow bob \}

Flow ordering
- permitted flow among labels

Trust ordering
- restrictiveness of policy
- (new in this paper)
Label Example

```c
1  int {alice→bob} x;
2  int {alice→bob, charlie} y;
3  x = y; // OK: bob ⊨ (bob ∨ charlie)
4  y = x; // Invalid
5  if (charlie actsfor bob) {
6        y = x; // OK: (bob ∨ charlie) ⊨ bob
7    }
```
Remote Calls

```java
1  void m1{alice←} () {  
2     Worker w = findWorker("bob.cs.cornell.edu");  
3     if (w actsfor bob) {  
4         int{alice→bob} data = 1;  
5         int{alice→} y = m2@w(data);  
6     }  
7  }  
8
9  int{alice→bob} m2{alice←} (int{alice→bob} x) {  
10     return x+1;  
11  }
```
System Architecture

Fabric nodes take one of three forms
- Storage nodes – persistent objects
- Worker nodes – perform computation
- Dissemination nodes – data replication

A single host can have multiple nodes
Evaluation: CMS

<table>
<thead>
<tr>
<th></th>
<th>Page Latency (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Course</td>
</tr>
<tr>
<td>EJB</td>
<td>305</td>
</tr>
<tr>
<td>Hilda</td>
<td>432</td>
</tr>
<tr>
<td>FabIL</td>
<td>35</td>
</tr>
<tr>
<td>FabIL/memory</td>
<td>35</td>
</tr>
<tr>
<td>Java</td>
<td>19</td>
</tr>
</tbody>
</table>

Table 1: CMS page load times (ms) under continuous load.

2X-4X slowdown compared to regular Java
Evaluation: OO7 Benchmark

<table>
<thead>
<tr>
<th></th>
<th>total</th>
<th>app</th>
<th>tx</th>
<th>log</th>
<th>fetch</th>
<th>store</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cold</td>
<td>9153</td>
<td>10%</td>
<td>2%</td>
<td>12%</td>
<td>74%</td>
<td>2%</td>
</tr>
<tr>
<td>Warm</td>
<td>6043</td>
<td>27%</td>
<td>3%</td>
<td>6%</td>
<td>61%</td>
<td>3%</td>
</tr>
<tr>
<td>Hot</td>
<td>840</td>
<td>46%</td>
<td>14%</td>
<td>24%</td>
<td>0%</td>
<td>17%</td>
</tr>
</tbody>
</table>

Table 2: Breakdown of OO7 traversal time (times in ms)

The regular Java implementation is 10X faster.
Nexus

Goal: Support authorization with all three trust bases.

Approach:
- Nexus authorization logic (NAL) – describes principals and their beliefs
- Reference monitor – general-purpose proof-checker enforces NAL
- OS and hardware features for each basis of trust.
Nexus & Trust Bases

Axiomatic
- *trusted platform module* – supports attested storage /w Merkle trees

Analytic
- *Introspection* – exposes view of kernel state
  - *e.g. process lookup tables, IPC ports, application state*

Synthetic
- *Interpositioning* – reference monitor can respond to untrusted I/O
Nexus Authorization Logic

NAL formulas have form “P says S” (see Schneider 2009)

- Principal P believes S to be true
- Predicates have application-defined meanings

Delegation

- “A speaksfor B” means that if A says S then B says S.

Examples:

- “Typechecker says isTypeSafe(PGM)”
- “Server says NTP speaksfor Server on TimeNow”
Caching and State

Decision cache - caches authorization decisions by the guard

Guard cache – caches partial proof-checking results.

State and Authorities – stateful predicates cannot be cached!
  • Authorities – listens on bound IPC, answers queries about stateful predicates.
Proof Evaluation Cost
Interpositioning Overhead
## Comparison of Approaches

<table>
<thead>
<tr>
<th>Information flow</th>
<th>Credentials based authentication</th>
</tr>
</thead>
<tbody>
<tr>
<td>Comprehensive/transitive</td>
<td>Constrains at resource interface</td>
</tr>
<tr>
<td>◦ Noninterfer-ish (declassification)</td>
<td>Policies</td>
</tr>
<tr>
<td>◦ Confidentiality/integrity, not availability</td>
<td>◦ Written by OS writer</td>
</tr>
<tr>
<td>Policies</td>
<td>◦ General propositional logic</td>
</tr>
<tr>
<td>◦ Written by app developer</td>
<td>TCB: OS(25k), theorem-prover, crypto</td>
</tr>
<tr>
<td>◦ Trust relationships</td>
<td>TCB: compiler(33k), crypto</td>
</tr>
</tbody>
</table>
Questions

Overall, which approach do you prefer?
  ◦ Do you see any value in using both?

What is a reasonable (performance) cost for security in practice?
  ◦ Is it 10%? 10X? 100X?

What do you think should be the root-of-trust?
  ◦ The hardware? OS? Compiler? Theorem-provers?
Perspective

Noninterference is powerful, but too restrictive
  ◦ Practical systems rely on declassification/endorsement.
  ◦ No single answer. (Survey paper: Sabelfield 2009)

Need to address covert channels to be truly comprehensive.

Is formal verification the answer?
  ◦ seL4