Plugging Side-Channel Leaks with Timing Information Flow Control

Bryan Ford
Yale University
http://dedis.cs.yale.edu/

USENIX HotCloud, June 13, 2012
The Long History of Timing Attacks

- **Cooperative attacks** – apply to:
  - Mandatory Access Control (MAC) systems [Kemmerer 83, Wray 91]
  - Decentralized Information Flow Control (DIFC) [Efstathopoulos 05, Zeldovich 06]

- **Non-cooperative attacks** – apply to:
  - Processes/VMs sharing a CPU core [Percival 05, Wang 06, Acııçmez 07, …]
  - Including VM configurations typical of clouds [Ristenpart 09]
Cooperative Attacks: Example

Trojan leaks **secret** information by modulating a **timing channel** observable by **unclassified** app.

**Secret** Level Trojan App

**Unclassified** Level Conspiring App

Timeshared Host

MAC/DIFC Protection Boundary

use a lot, use a little

how fast am I running?
Non-Cooperative Attacks: Example

Apps *unintentionally* modulate shared resources to reveal secrets when running standard code.

- **Acme Data, Inc.**
  - Crypto (AES, RSA, ...)
- **Eviltron**
  - Passive Attacker
- **Cloud Host**

Key-dependent usage patterns:
- Watch memory access timing

Discretionary Protection Boundary
Timing Attacks in the Cloud

The cloud *exacerbates* timing channel risks:

1. Routine co-residency
2. Massive parallelism
3. No intrusion alarms $\rightarrow$ hard to monitor/detect
4. Partitioning defenses defeat elasticity

"*Determinating Timing Channels in Compute Clouds*"
[CCSW '10]
Leak-Plugging Approaches

Two broad classes of existing solutions:

- *Tweak specific algorithms, implementations*
  - Equalize AES path lengths, cache footprint, …
- *Demand-insensitive resource partitioning*
  - Requires *new or modified hardware* in general
    - Partition CPU cores, cache, interconnect, …
  - Can't oversubscribe, stat-mux resources
    - Not economically feasible in an “elastic” cloud!
Information Flow Control

Explicitly *label* information, constrain propagation

- Old idea, recently (re-)popularized
  - DIFC, Asbestos/HiStar/Flume
  - Label variables, processes, messages, etc.
- So far, IFC avoids the timing channel issue
  - How would one “label time”?
  - What would we do with “timing labels”?
    - Hard to prevent programs from “taking time”!
- But could IFC apply to timing channels too?
Adapting IFC to Timing Analysis

Key idea: we need two kinds of labels

- **State labels** attached to *explicit program state*
  - Represent ownership of information in the *bits* of a variable, message, process, etc.

- **Time Labels** attached to *event channels*
  - Represent ownership of information affecting *time* or *rate* events occur in a program

**TIFC ≡ Timing Information Flow Control**

- Analyze, constrain both state & timing leaks
A "Timing-Hardened Cloud"
Flume IFC Model

Flume IFC model summary:

- **Tags** represent ownership/taint: “Alice”, “Bob”
- **Labels** are sets of tags:
  - \{Alice, Bob\} ≡ “contains Alice's & Bob's data”
- **Capabilities** enable adding/removing tags
  - e.g., If process P holds capability \{Alice\-\},
    P can *declassify* (remove) the Alice tag

P can send data to Q iff \((L_P \setminus L_Q) \subseteq (C^-_P \cup C^+_Q)\)
Adding Timings Labels to IFC

- **Timing Tag** is a tag with a frequency
  - Tag $A_f$ indicates a timing channel might leak A's information at up to $f$ bits per second
  - Tag $A_\infty$ indicates a timing channel might leak A's information at arbitrarily high rate
- **Labels** can contain both state and timing tags
  - Message channel labeled \{A/B$_f$\} indicates:
    - Message *bits* tainted with A's info
    - Message *arrival events* in channel tainted by B's info at up to rate $f$
Example 1: Dedicated Resources

Trivial case: physical partitioning of resources

Cloud Provider's Infrastructure

Alice's Compute Server

Job \( \{A/A_\infty\} \)  
Result \( \{A/A_\infty\} \)

Alice's Gateway \( \{A^+,A^-\} \)

Bob's Compute Server

Job \( \{B/B_\infty\} \)  
Result \( \{B/B_\infty\} \)

Bob's Gateway \( \{B^+,B^-\} \)

Alice

Bob
Informal “Schedule Analysis”

Bob's job is “short”

Alice submits \(\{A/A_\infty\}\) Alice's job \(\{A/A_\infty\}\) Alice's job completion time \textbf{not dependent} on Bob's job

Bob's job is “long”

Alice submits \(\{A/A_\infty\}\) Alice's job \(\{A/A_\infty\}\)
Demand-Insensitive Timesharing

Reservation-Based Scheduler
{-/-}

Shared Compute Server

Control
{-/-}

no demand feedback

Alice's Gateway
{A⁺, A⁻}

Result
{A/\infty}

Job
{A/\infty}

Alice

Bob's Gateway
{B⁺, B⁻}

Result
{B/\infty}

Job
{B/\infty}

Bob
Informal “Schedule Analysis”

Alice’s job completion time still not dependent on Bob’s job

Bob’s job is short

Bob’s job is long
Timing Control in Elastic Clouds

Need two additional facilities:

- **System-enforced deterministic execution** [OSDI '10]
  - OS/VMM ensures that a job's outputs depend *only* on job's explicit inputs

- **Pacing queues**
  - Input jobs/messages at any rate
  - Output jobs/messages on a *fixed schedule*
Elastic Cloud Scenario

- **Demand Scheduler**:
  \[\{A, B/A_{\infty}, B_{\infty}\}\]

- **Control**:
  \[\{A, B/A_{\infty}, B_{\infty}\}\]

- **Shared Deterministic Compute Server**

- **Job**:
  - Alice's Gateway:
    \[\{A^+, A^-, B_f\}\]
  - Bob's Gateway:
    \[\{B^+, B^-, A_f\}\]

- **Result**:
  - Alice:
    \[\{A/A_{\infty}\}\]
  - Bob:
    \[\{B/A_{\infty}\}\]

- **Pacer**:
  - freq \(f\)

- **Demand**:
  \[\{A, B/A_{\infty}, B_{\infty}\}\]

- **Control**:
  \[\{A, B/A_{\infty}, B_{\infty}\}\]

- **Result**:
  - Alice:
    \[\{A/A_{\infty}, B\}\]
  - Bob:
    \[\{B/A_{\infty}, B\}\]

- **Job**:
  - Alice:
    \[\{A/A_{\infty}\}\]
  - Bob:
    \[\{B/B_{\infty}\}\]

- **Result**:
  - Alice:
    \[\{A/A_{\infty}, B\}\]
  - Bob:
    \[\{B/A_{\infty}, B\}\]

- **Pacer**:
  - freq \(f\)

- **Result**:
  - Alice:
    \[\{A/A_f, B_f\}\]
  - Bob:
    \[\{B/A_f, B_f\}\]
Jobs: In Anytime, Out on a Schedule

For each customer (e.g., Alice):

- Deterministic execution ensures job output bits depend only on job input bits: \( O_j = f(I_j) \)
- Job outputs produced in same order as inputs
- At each “clock tick”, paced queue releases either next job output or says not ready yet
  - The single bit of information per clock tick that might leak other users' information
Informal “Schedule Analysis”

(b) Schedule: Bob's job short

(b) Schedule: Bob's job long
Key Challenges/Questions

- Formalize full TIFC model
  - Potentially applicable at systems or PL levels
  - Integrate Myers' “predictive mitigation” ideas
- Build TIFC-enforcing prototype
  - Ongoing, based on Determinator [OSDI '10]
- Explore flexibility, applicability of model
  - Can model support interactive applications?
  - Can model support transactional apps?
Conclusion

- TIFC = IFC extended to timing channels
- Several “timing-hardening” approaches
  - Physical partitioning
  - Demand-insensitive timesharing
  - Elastic computing via deterministic job model
- First general approach that could be both:
  - Feasible on unmodified hardware
  - Suitable for stat-muxed clouds

Further information: http://dedis.cs.yale.edu