Nomad: Mitigating Arbitrary Cloud Side Channels via Provider-Assisted Migration

Soo-Jin Moon, Vyas Sekar, Michael K. Reiter

Carnegie Mellon University

THE UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

Project Silver
Co-residency side-channel attacks in clouds

Stealing secrets (e.g., keys)

• Many different vectors
  (e.g., L2/L3 cache, storage, main memory)

Demonstrated side-channel attacks are not limited to:
Y. Zhang et al., CCS2012; T. Ristenpart et al., CCS2009; F. Liu et al., Oakland 2015
Limitations of Current Defenses

1. Requires significant/detailed upgrades
   - OS
   - OS
     - e.g., Noise injection
   - Hypervisor
     - e.g., Deterministic execution
   - Hardware
     - e.g., New cache design

2. Attack-specific

Proposed defense includes but not limited to: Y. Zhang et al., CCS2013; T. Kim et al., USENIXSec 2012; F. Liu and R. Lee, Micro 2014
Limitations of Current Defenses

1. Requires significant/detailed upgrades

- OS
- OS
- Hypervisor
- Hardware
  - e.g., Noise injection
  - e.g., Deterministic execution
  - e.g., New cache design

2. Attack-specific

What about future side-channel attacks?

Proposed defense includes but not limited to: Y. Zhang et al., CCS2013; T. Kim et al., USENIXSec 2012; F. Liu and R. Lee, Micro 2014
Ideal Properties

1) General

2) Immediately deployable
Ideal Properties

1) General

2) Immediately deployable

Single-tenancy?
Ideal Properties

1) General

2) Immediately deployable

Single-tenancy?
Nomad Ideas

1) General

2) Immediately deployable
Nomad Ideas

1) General
   Tackle root-cause
   → Minimize co-residency

2) Immediately deployable
Nomad Ideas

1) General
   Tackle root-cause
   \[\rightarrow\] Minimize co-residency

2) Immediately deployable
   Migration
Cloud Controller

- Provider-assisted

**Nomad Vision: Migration-as-a-Service**
Nomad Vision: Migration-as-a-Service

- Provider-assisted
**Nomad Vision: Migration-as-a-Service**

- **Opt-in Service**
  - Cloud Provider
  - Service offering
  - Opt-in?
  - Clients

- **Provider-assisted**
  - Cloud Controller
  - Move VMs {...}
Nomad Practical Challenges

Logic
Characterize information leakage due to co-residency
Nomad Practical Challenges

Logic
Characterize information leakage due to co-residency

Scalable Design
e.g., can Amazon EC2 run this?
Nomad Practical Challenges

Logic
Characterize information leakage due to co-residency

Scalable Design
e.g., can Amazon EC2 run this?

Practical Impact (cloud)
Minimal modifications?
**Nomad Practical Challenges**

**Logic**
Characterize information leakage due to co-residency

**Scalable Design**
e.g., can Amazon EC2 run this?

**Practical Impact (cloud)**
Minimal modifications?

**Cloud Controller**

**Practical Impact (applications)**
1) Advancement of VM migration techniques
2) Many cloud workloads with in-built resilience to migration
Our Work

1. Idea

General side-channel defense via migration
Our Work

1. Idea
   General side-channel defense via migration

2. Logic
   Characterize information leakage due to co-residency
Our Work

1. Idea
   General side-channel defense via migration

2. Logic
   Characterize information leakage due to co-residency

3. Scalable Design
   Scalable VM migration strategy that can handle large cloud deployments
Our Work

1. Idea
General side-channel defense via migration

2. Logic
Characterize information leakage due to co-residency

3. Scalable Design
Scalable VM migration strategy that can handle large cloud deployments

4. Practical Impact
Practical OpenStack implementation with minimal modifications
Our Work

1. Idea
   General side-channel defense via migration

2. Logic
   Characterize information leakage due to co-residency

3. Scalable Design
   Scalable VM migration strategy that can handle large cloud deployments

4. Practical Impact
   Practical OpenStack implementation with minimal modifications
Threat Model

Objective: Extract secrets via co-residency

- Can use any kind of resource
- Can launch/terminate VMs at will
- VMs of a given client can collaborate
Threat Model

Objective: Extract secrets via co-residency

- Can use any kind of resource
- Can launch/terminate VMs at will
- VMs of a given client can collaborate
- Cannot control VM placement
- No info. sharing across distinct clients
Threat Model

Objective: Extract secrets via co-residency

- Can use any kind of resource
- Can launch/terminate VMs at will
- VMs of a given client can collaborate
- Cannot control VM placement
- No info. sharing across distinct clients
- Don’t know which other clients are malicious
Information Leakage (InfoLeak) Model

Clients → InfoLeak?
Information Leakage (InfoLeak) Model

Clients

Replicated? (R or NR)

VM-level view

B1

B2

R

InfoLeak?
Information Leakage (InfoLeak) Model

Clients

Replicated? (R or NR)

VM-level view

R

NR
Information Leakage (InfoLeak) Model

Clients

Replicated? (R or NR)

Collaborating? (C or NC)

VM-level view

B1

B2

R

C

R1

R2

NR
Information Leakage (InfoLeak) Model

Clients

Replicated? (R or NR)

Collaborating? (C or NC)

VM-level view

R

C

NR

NC

B1

B2

B1

B2

R1

R2

R1

R2
Information Leakage (InfoLeak) Model

<table>
<thead>
<tr>
<th>Collaborating?</th>
<th>Replicated?</th>
</tr>
</thead>
<tbody>
<tr>
<td>NR</td>
<td>R</td>
</tr>
<tr>
<td>NC</td>
<td>&lt;NR, NC&gt;</td>
</tr>
<tr>
<td>C</td>
<td>&lt;NR, C&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;R, NC&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;R, C&gt;</td>
</tr>
</tbody>
</table>

Least InfoLeak

Most InfoLeak
Our Work

1. **Idea**
   General side-channel defense via migration

2. **Logic**
   Characterize information leakage due to co-residency

3. **Scalable Design**
   Scalable VM migration strategy that can handle large cloud deployments

4. **Practical Impact**
   Practical OpenStack implementation with minimal modifications
System Overview

Cloud Controller

Move VMs {...}
System Overview

Cloud Provider

Deployment model (e.g., <NR,NC>)

Opt-in?

Clients

Cloud Controller

Move VMs {...}

Machine

VM

Machine

VM

Machine

VM
Operational Timeline

1 epoch = D time units

Sliding Window of $\Delta$ epochs

Run placement algorithm every epoch
Operational Timeline

1 epoch = D time units

Sliding Window of Δ epochs

Run placement algorithm every epoch

Side-channel Parameters:
- K: Information leakage rate (i.e., bits per time unit)
- P: secret length (i.e., bits)
Operational Timeline

1 epoch = D time units

Sliding Window of $\Delta$ epochs

Run placement algorithm every epoch

Extracted secret (bits) if two VMs are co-resident for $\Delta$ epochs

Provider chooses D and $\Delta$ to AT LEAST satisfy:

$$D \times \Delta \times K < P$$
Placement Algorithm

Deployment Model (e.g., <NR,NC>)
Recent VM Placements
Client Workloads & Constraints

Placement Algorithm

VM Placement
Placement Algorithm

Goal (per epoch):
Minimize a global sum of a client-pair InfoLeak across past $\Delta$ epochs i.e.,
$$\sum_{c,c'}^{InfoLeak_{c \rightarrow c'}([t - \Delta, t])}$$
subject to a fixed migration budget
Placement Algorithm

Goal (per epoch):
Minimize a global sum of a client-pair InfoLeak across past $\Delta$ epochs i.e.,
$$\sum_{c,c'} \text{InfoLeak}_{c \rightarrow c'}([t - \Delta, t])$$
subject to a fixed migration budget

F(Deployment Model)

Deployment Model (e.g.,<NR,NC>) Recent VM Placements
Client Workloads & Constraints

Placement Algorithm

VM Placement
**Placement Algorithm**

**Deployment Model** (e.g., <NR, NC>)

**Recent VM Placements**

**Client Workloads & Constraints**

---

**Goal (per epoch):**

Minimize a *global* sum of a client-pair InfoLeak across past Δ epochs i.e.,

\[\sum_{c, c'} \text{InfoLeak}_{c \rightarrow c'}([t - \Delta, t])\]

subject to a fixed migration budget

---

**F** (Deployment Model)

**F** (Network Capacity)
Challenge: Scalability

Inputs

Should handle tens of thousands of servers

Placement Algorithm

VM Placement
Challenge: Scalability

Inputs

Placement Algorithm

Should handle tens of thousands of servers

- ILP (Integer Linear Programming)
  For 40 machines, $D > 1$ day
Challenge: Scalability

Inputs

Placement Algorithm

VM Placement

Should handle tens of thousands of servers

- ILP (Integer Linear Programming)
- For 40 machines, D > 1 day
Challenge: Scalability

Should handle tens of thousands of servers

Placement Algorithm

- ILP (Integer Linear Programming)
  For 40 machines, $D > 1$ day
- Basic Greedy
  For 400 machines, $D > 1$ day
Challenge: Scalability

Inputs

Should handle tens of thousands of servers

Placement Algorithm

- ILP (Integer Linear Programming)
  For 40 machines, $D > 1$ day
- Basic Greedy
  For 400 machines, $D > 1$ day

VM Placement
**Challenge: Scalability**

- Should handle tens of thousands of servers

**Inputs**

- Placement Algorithm
- VM Placement

**Options**

- ILP (Integer Linear Programming)
  - For 40 machines, $D > 1$ day
- Basic Greedy
  - For 400 machines, $D > 1$ day
- Basic Greedy with our optimizations
Why is Basic Greedy not scalable?

Generate Moves

Compute Benefit (total reduction in infoLeak)

Pick Best Move

Make Move

totalNumMove > Budget

Exit

Pairwise Swap: 1-2 -> 2-1 ...
N-way Swap: ...

No

Yes
Why is Basic Greedy not scalable?

Generate Moves

Compute Benefit
(total reduction in infoLeak)

Pick Best Move

Make Move

totalNumMove > Budget

Bottleneck #1: Large Search Space

N-way Swap: ...

Yes

Exit
Why is Basic Greedy not scalable?

1. **Generate Moves**
2. **Compute Benefit** (total reduction in infoLeak)
3. **Pick Best Move**
4. **Make Move**

**Bottleneck #1**: Large Search Space

**Bottleneck #2**: Computing InfoLeak across all clients

- totalNumMove > Budget

   - Yes: Exit
   - No: Repeat

---

Free Insert: 1
Pairwise Swap: 1 - 2 - 2 - 1
Exit Yes

---
Why is Basic Greedy not scalable?

1. **Generate Moves**

2. **Compute Benefit**
   - (total reduction in infoLeak)

3. **Pick Best Move**

4. **Make Move**

5. **totalNumMove > Budget**

- **Bottleneck #1:** Large Search Space
- **Bottleneck #2:** Computing InfoLeak across all clients
- **Bottleneck #3:** Re-generating move table after each move

Exit if totalNumMove > Budget.
Our Approach

Bottlenecks

- Large Search Space
- Computing InfoLeak across all clients
- Re-generating move table after each move

Our Approach

- Prune Search Space
- Incremental Benefit Computation
- Intra-Epoch Lazy Evaluation
Our Approach

Bottlenecks

Large Search Space
Computing InfoLeak across all clients
Re-generating move table after each move

Our Approach

Prune Search Space
Incremental Benefit Computation
Intra-Epoch Lazy Evaluation
Prune #1: Pruning Move Space

Sets of all moves

- Insert
  1 -> M1

- Pairwise Swap
  1-2 -> 2-1

- N-way Swap
  ....
Prune #1: Pruning Move Space

Sets of all moves

- Insert
  1 -> M1

- Pairwise Swap
  1-2 -> 2-1

- N-way Swap
  ....

Nomad sets of all moves

- Free Insert
  1 -> M1

- Pairwise Swap
  1-2 -> 2-1
Prune #2: Hierarchical Decomposition

Sets of all free inserts

Clients

Machines

C1

M1

C1000

M2

M50000
Sets of all free inserts

Clients

C1
C1000

Machines

M1
M2
M50000
Prune #2: Hierarchical Decomposition

Sets of all free inserts

Clients

M1

M2

M50000

C1

C1000

C

Machines

Nomad sets of all free inserts

C1

Cluster1

C1000

Cluster25
Prune #2: Hierarchical Decomposition

Sets of all free inserts

Clients

- C1
- .
- .
- C1000

Machines

- M1
- M2
- .
- .
- M50000

Nomad sets of all free inserts

C1
- .
- C1000

Cluster1

- .

Cluster25

M1

- M2

- .

- M2000

- .

- .

- .

Clients             Machines

Cluster1

Cluster1
Our Work

1. Idea
   General side-channel defense via migration

2. Logic
   Characterize information leakage due to co-residency

3. Scalable Design
   Scalable VM migration strategy that can handle large cloud deployments

4. Practical Impact
   Practical OpenStack implementation with minimal modifications
System Implementation

OpenStack v. Icehouse

Cloud Controller

Clients in Cluster 1

Cluster 1 Placement Algorithm

VM Placements for Cluster 1

Clients in Cluster N

Cluster N Placement Algorithm

VM Placements for Cluster N
System Implementation (One Cluster)

Cluster 1
Placement Algorithm

General Placement Computation

OpenStack-specific Migration Engine

VM Placement

Custom C++
~2000 LOC

OpenStack Icehouse:
~200 LOC in Controller Scheduler code
System Implementation (One Cluster)

Cluster 1
Placement Algorithm

General Placement Computation

OpenStack-specific Migration Engine

VM Placement

Custom C++
~2000 LOC

OpenStack Icehouse:
~200 LOC in Controller Scheduler code

Requires minimal modifications to existing deployments
Key Evaluation Questions

• Information leakage resilience
• Scalability
• Impact on cloud applications
• Benefit/Cost of each design idea
• Resilience to strategic adversary
Information Leakage resilience

\(<R,C>: \text{Problem size of 20-machines}\)

Metric:
\(\text{InfoLeak}_{c \rightarrow c'}([t - \Delta, t])\)

Nomad brings \(~4.5x\) reduction in InfoLeak for 98\(^{th}\) percentile compared to static w.r.t. ILP.
Nomad placement algorithm is scalable to large deployments
Replicated web-server (Wikibench)
• Each client: 3 replicated web servers, 1 worker
  – In one epoch, *at least* 1 server migrates

\[
\text{Norm. Throughput (Norm. } T) = \frac{T_{w/o} - T_w}{T_{w/o}} \times 100
\]

• Overhead (Norm. T)
  – \( \sim 0\% \) for 95\textsuperscript{th} Norm T.
  – 0.096\% for 50\textsuperscript{th} (median) Norm. T.
  – 1.8\% for 5\textsuperscript{th} Norm. T
Discussion

• Fast side-channel attacks
  – Need out-of-band defense
  – e.g., introduce cache noise, refresh secret

• Network Impact
  – With techniques like incremental diffs, the transfer size is much less than base VM image

• Incentives for adoption
  – Security-conscious clients opt-in
  – Providers have new revenue streams

• More opportunities
  – Fairness across clients
Conclusions

• Co-residency side-channel attacks: real/growing threats

Current World : No Migration
  1. Per-attack fixes
  2. Require significant upgrades

Nomad: “Migration-as-a-Service”
  1. General solution
  2. Needs minimal changes

• Nomad achieves:
  – Information leakage resilience close to the ILP
  – Scalable VM placement algorithm
  – Practical system atop OpenStack with minimal modifications