Hey, You, Get Off of My Cloud!
Exploring Information Leakage in Third-Party Clouds

Thomas Ristenpart, Eran Tromer, Hovav Shacham, Stefan Savage

UCSD       MIT       UCSD       UCSD
Today’s talk in one slide

Third-party clouds:

“cloud cartography” to map internal infrastructure

get malicious VM on same physical server as victim

side-channels might leak confidential data of victim

Exploiting a placement vulnerability:
only use cloud-provided functionality
A simplified model of third-party cloud computing

Users run Virtual Machines (VMs) on cloud provider’s infrastructure

Multitenancy (users share physical resources)

Virtual Machine Manager (VMM) manages physical server resources for VMs

To the VM should look like dedicated server

Owned/operated by cloud provider
Trust models in cloud computing

Users must trust third-party provider to

- not spy on running VMs / data
- secure infrastructure from external attackers
- secure infrastructure from internal attackers
Trust models in cloud computing

Users must trust third-party provider to
- not spy on running VMs / data
- secure infrastructure from external attackers
- secure infrastructure from internal attackers

Threats due to sharing of physical infrastructure?
- Your business competitor
- Script kiddies
- Criminals
- ...
We explore a new threat model:

Attacker identifies one or more victims VMs in cloud

1) Achieve advantageous placement
   - Attacker launches VMs
   - VMs each check for co-residence on same server as victim

2) Launch attacks using physical proximity
   - Exploit VMM vulnerability
   - DoS
   - Side-channel attack
Using Amazon EC2 as a case study:

1) Cloud cartography
   map internal infrastructure of cloud
   map used to locate targets in cloud

2) Checking for co-residence
   check that VM is on same server as target
   - network-based co-residence checks
   - efficacy confirmed by covert channels

3) Achieving co-residence
   brute forcing placement
   instance flooding after target launches

4) Side-channel information leakage
   coarse-grained cache-contention channels
   might leak confidential information
What our results mean is that

1) given no insider information
2) restricted by (the spirit of) Amazon’s acceptable use policy (AUP)

(using only Amazon’s customer APIs and very restricted network probing)

we can:

1. Pick target(s)
2. Choose launch parameters for malicious VMs
3. Each VM checks for co-residence
4. Given successful placement, spy on victim web server’s traffic patterns via side channels
Before we get into details of case study:

**Should I panic?**
No. We didn’t show how to extract cryptographic keys

But:
We exhibit side-channels to measure load across VMs in EC2
Coarser versions of channels used to extract cryptographic keys

**Other clouds?**
We haven’t investigated other clouds

**Problems only in EC2?**
EC2 network configuration made cartography and co-residence checking easy

But:
These don’t seem critical to success
Placement vulnerabilities seem inherent issue when using multitenancy
1 or more targets in the cloud and we want to achieve co-resident placement with any of them

Suppose we have an oracle for checking co-residence (we’ll realize it later)

Launch lots of instances (over time), each asking oracle if successful

If target set large enough or adversarial resources (time & money) sufficient, this might already work

In practice, we can do much better than this
Some info about EC2 service (at time of study)

Linux-based VMs available
Uses Xen-based VM manager

User account
3 “availability zones” (Zone 1, Zone 2, Zone 3)
5 instance types (various combinations of virtualized resources)

<table>
<thead>
<tr>
<th>Type</th>
<th>gigs of RAM</th>
<th>EC2 Compute Units (ECU)</th>
</tr>
</thead>
<tbody>
<tr>
<td>m1.small (default)</td>
<td>1.7</td>
<td>1</td>
</tr>
<tr>
<td>m1.large</td>
<td>7.5</td>
<td>4</td>
</tr>
<tr>
<td>m1.xlarge</td>
<td>15</td>
<td>8</td>
</tr>
<tr>
<td>c1.medium</td>
<td>1.7</td>
<td>5</td>
</tr>
<tr>
<td>c1.xlarge</td>
<td>7</td>
<td>20</td>
</tr>
</tbody>
</table>

1 ECU = 1.0-1.2 GHz 2007 Opteron or 2007 Xeon processor

Limit of 20 instances at a time per account.
Essentially unlimited accounts with credit card.
Our experiments indicate that internal IPs are **statically assigned** to physical servers.

Co-residence checking via Dom0: only hop on traceroute to co-resident target.
Cloud cartography

Map internal cloud structure to locate targets

Towards generating a map, we want to understand affects of launch parameters:

<table>
<thead>
<tr>
<th>Availability zone</th>
<th>Instance type</th>
<th>Account</th>
</tr>
</thead>
</table>

From “Account A”: launch 20 instances of each type in each availability zone

\[ 20 \times 15 = 300 \text{ instances launched} \]

Clean partition of internal IP address space among availability zones
From “Account A”: launch 20 instances of each type in each availability zone

\[ 20 \times 15 = 300 \] instances launched

From “Account B”: launch 20 instances of each type in Zone 3

\[ 20 \times 5 = 100 \] instances launched

39 hours apart

Cloud cartography

55 of 100 Account B instances had IP address assigned to Account A instance

Seems that user account doesn’t impact placement

Most /24 associated to single instance type and zone

Associate each /24 with Zone & Type
Data from 977 instances with unique internal IPs

simple heuristics based on EC2 network configuration

Ability to label /24’s with zone & instance type(s)

To locate a target in the cloud:

1) DNS lookup maps External IP to Internal IP
2) Check /24 to see what zone & instance type

Our map provides sufficiently precise estimate to use for mounting attacks.

Mapping might have other applications, as well (inferring types of instances used by a company)
Achieving co-residence

“Brute-forcing” co-residence

Attacker launches many VMs over a relatively long period of time in target’s zone and of target type

Experiment:

1,686 public HTTP servers as stand-in “targets” running m1.small and in Zone 3 (via our map)

1,785 “attacker” instances launched over 18 days

Each checked co-residence against all targets

Results:

78 unique Dom0 IPs

141 / 1,686 (8.4%) had attacker co-resident

Lower bound on true success rate

Sequential placement locality lowers success
Achieving co-residence

Can an attacker do better?

Launch many instances in parallel near time of target launch

Exploits parallel placement locality

Dynamic nature of cloud helps attacker:

Auto-scaling services (Amazon, RightScale, ...)

Cause target VM to crash, relaunch

Wait for maintenance cycles

...
Achieving co-residence

Can an attacker do better?

Launch many instances in parallel near time of target launch

Exploits parallel placement locality

Experiment:
Repeat for 10 trials:

1) Launch 1 target VM (Account A)

2) 5 minutes later, launch 20 “attack” VMs (alternate using Account B or C)

3) Determine if any co-resident with target

4 / 10 trials succeeded

In paper:
parallel placement locality good for >56 hours success against commercial accounts
Attacker has uncomfortably good chance at achieving co-residence with your VM

What can the attacker then do?
Side-channel information leakage

Cache contention yields cross-VM load measurement in EC2

Attacker measures time to retrieve memory data
Read times increase with Victim’s load

Measurements via Prime+Trigger+Probe:

1) Read an array to ensure cache used by attacker VM (Prime)
2) Busy loop until CPU’s cycle counter jumps by large value (Trigger)
3) Measure time to read array (Probe)

Extends [OST05] Prime+Produce technique
Load measurement uses coarse-grained side channel

- Simpler to mount
- More robust to noise
- Extract less information

Coarse side channels could be damaging in hands of clever attackers
Cache-based load measurement to determine co-residence

Repeated HTTP get requests

Running Apache server

Perform cache load measurements

3 pairs of instances, 2 pairs co-resident and 1 not
100 cache load measurements during **HTTP gets** (1024 byte page) and with **no HTTP gets**

![Graphs showing CPU cycles over trials for instances co-resident and not co-resident]
Cache-based load measurement of traffic rates

3 trials with 1 pair of co-resident instances:
1000 cache load measurements during
0, 50, 100, or 200 HTTP gets (3 Mbyte page) per minute for ~1.5 mins
Prime+Trigger+Probe combined with differential encoding technique gives high bandwidth cross-VM covert channel on EC2

Keystroke timing in experimental testbed similar to EC2 m1.small instances
More on cache-based physical channels

Prime+Trigger+Probe combined with differential encoding technique gives high bandwidth cross-VM covert channel on EC2

Keystroke timing in experimental testbed similar to EC2 m1.small instances
More on cache-based physical channels

Prime+Trigger+Probe combined with differential encoding technique gives high bandwidth cross-VM covert channel on EC2

Keystroke timing in experimental testbed similar to EC2 m1.small instances
More on cache-based physical channels

Prime+Trigger+Probe combined with differential encoding technique gives high bandwidth cross-VM covert channel on EC2

Keystroke timing in experimental testbed similar to EC2 m1.small instances

We show that cache-load measurements enable cross-VM keystroke detection

Keystroke timing of this form might be sufficient for the password recovery attacks of [Song, Wagner, Tian 01]
What can cloud providers do?

1) Cloud cartography

Customers can pay the (slight) extra operational costs to avoid multitenancy

2) Checking for co-residence

What can cloud providers do?

Possible counter-measures:

- Random Internal IP assignment
- Isolate each user’s view of internal address space
- Hide Dom0 from traceroutes
- Random Internal IP assignment
- Allow users to opt out of multitenancy

3) Achieving co-residence

- Hardware or software countermeasures to stop leakage [Ber05,OST05,Page02,Page03,Page05,Per05]

4) Side-channel information leakage

- Allow users to opt out of multitenancy
Placement vulnerability: attackers can knowingly achieve co-residence with target

Load measurement via side channels

Security threat seems inherent to any third-party cloud with multitenancy

More demands on virtual isolation due to multitenancy

Coarse-grained side channels already of use to some attackers