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

CCS '09: Proceedings of the 16th ACM conference on Computer and communications security
November 2009

Authors:

Thomas Ristenpart
Eran Tromer
Hovav Shacham
Stefan Savage

Presenters:

Deniz Aytemiz
Vikas Krishnan Radhakrishnan

What is this work about?

➔ Identifying the location of a target VM in the cloud.

➔ Instantiating co-resident VM with the target VM to enable side-channel attacks.

★ This paper is a case study of Amazon EC2.

★ Amazon EC2 is a third party cloud computing service where you can instantiate Virtual Machines.

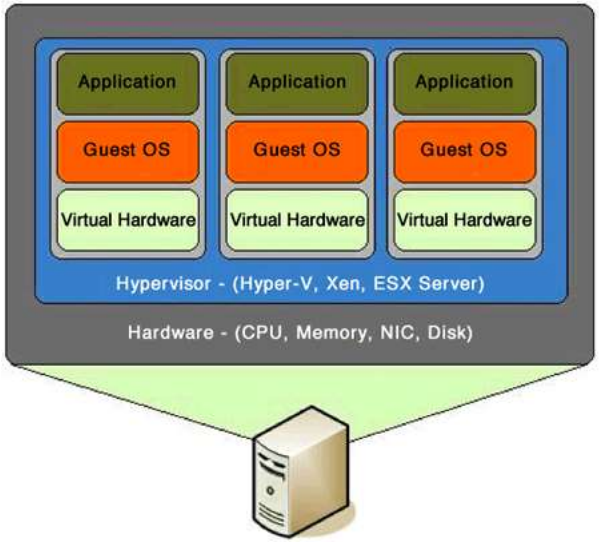


Image source: http://cis2.oc.ctc.edu/oc_apps/Westlund/xbook/xbook.php?unit=11&proc=page&numb=3

Outline

1. Introduction
2. Threat Model
3. EC2 Service
4. Network Probing
5. Cloud Cartography
6. Determining Co-residence
7. Exploiting Placement in EC2
8. Cross-VM Information Leakage
9. Conclusion & Discussion

Can one determine where in the cloud infrastructure an instance is located?

Can one easily determine if two instances are co-resident on the same physic machine?

Can an adversary launch instances that will be co-resident with other user's instances?

Can an adversary exploit cross-VM information leakage once co-resident?

Introduction

- ★ Multiplexing VMs enables cross-VM side channel attacks

Multiplexing: Two or more disjoint VMs sharing the same physical structure. (e.g. CPUs cache)

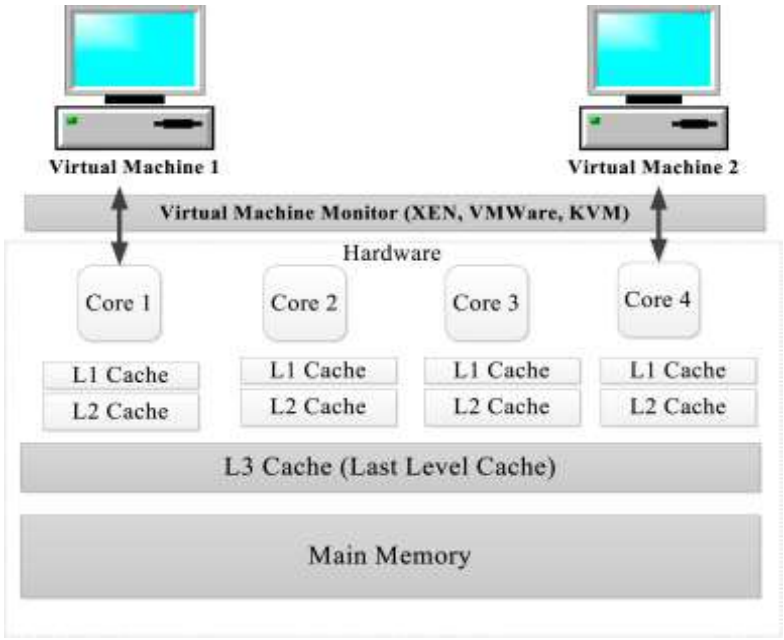


Image source: https://www.semanticscholar.org/paper/Scalable-virtual-machine-multiplexing-Gupta/ec29764f93fab42fd4b7bb6d8fca9f1bced19c97

Why is Multiplexing VMs Good AND Bad?

- ➔ Maximizes efficiency
- ➔ Attack case: Information leakage due to sharing of physical resources

Hypervisor provides isolation between VMs and intermediate access to physical hardware.

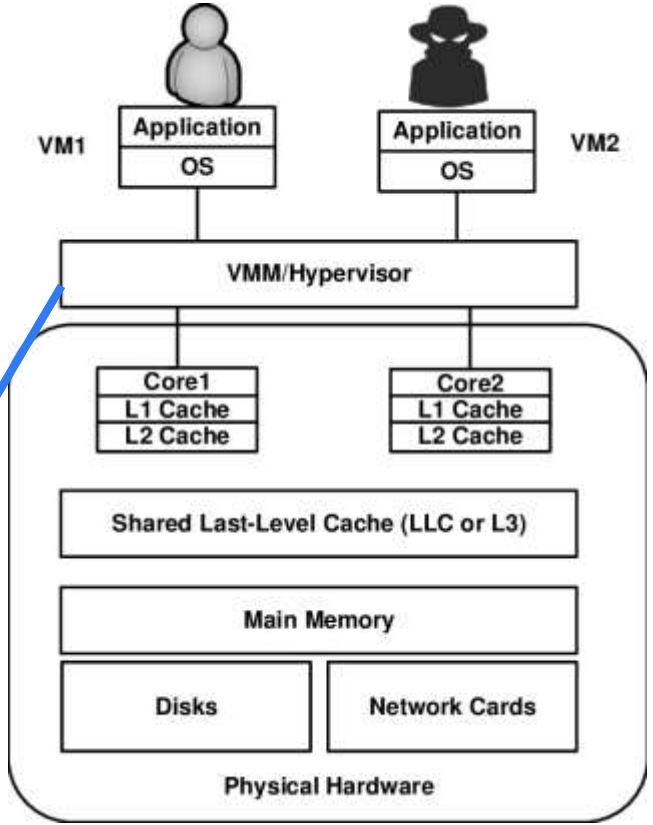


Image source: https://www.researchgate.net/figure/Cross-VM-side-channel-attack-using-a-shared-last-level-cache_fig2_327314423

Threat Model

Attacks involve two steps:

1. Place malicious VM in the same physical hardware as the target VM.
2. Extract confidential information from target VM through side channels.

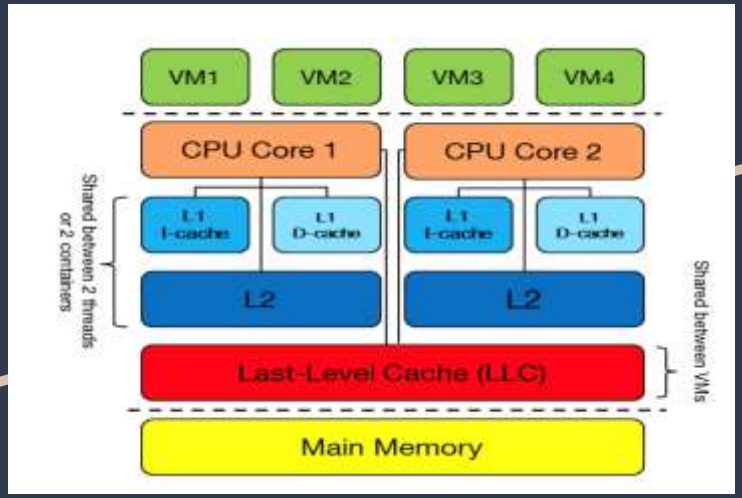


Image source:https://hal.inria.fr/hal-01591808/document

Only external attackers are considered in this work!

Attack to *some* known hosted service

Attack to a *particular* victim service

Third party cloud computing implicitly increases attack surface.

How relevant is this work and its results to the present?

This work is from late 2008/2009

Nascent hyperscale applications then!

Everything is cloudified now!

With that in mind, let’s revisit the origins of **cloud security**...

EC2 Service

"Third-party cloud computing platform" (IaaS)

Xen hypervisor (now known as Citrix hypervisor)

Guest VM: Linux, FreeBSD, OpenSolaris, Windows

Domain0 → privileged VM to manage guest VMs

How to use EC2?

Register with EC2:
Create Account with contact Email ID and CC information

Choose: Region
↓
Availability Zone
↓
Instance Type

Create a VM image
↓
Deploy on EC2 → *instance*

EC2 Infrastructure Availability

Regions: United States, Europe

Availability Zones: 3 per region → *Zone 1*, *Zone 2*, *Zone 3*

Instance Types (for Linux instance images in the US Region):

Instance Type	CPU Architecture	Instance Capability	Cost/Hour
m1.small	32-bit	1 vCPU, 1.7 GB memory, 160 GB storage	$0.10
c1.medium	32-bit	2 vCPUs, 1.7 GB memory, 350 GB storage	NA
m1.large	64-bit	2 vCPUs, 7.5 GB memory, 850 GB storage	$0.40
m1.xlarge	64-bit	4 vCPUs, 15 GB memory, 1680 GB storage	NA
c1.xlarge	64-bit	8 vCPUs, 7 GB memory, 1680 GB storage	NA

Source: .https://aws.amazon.com/ec2/previous-generation/

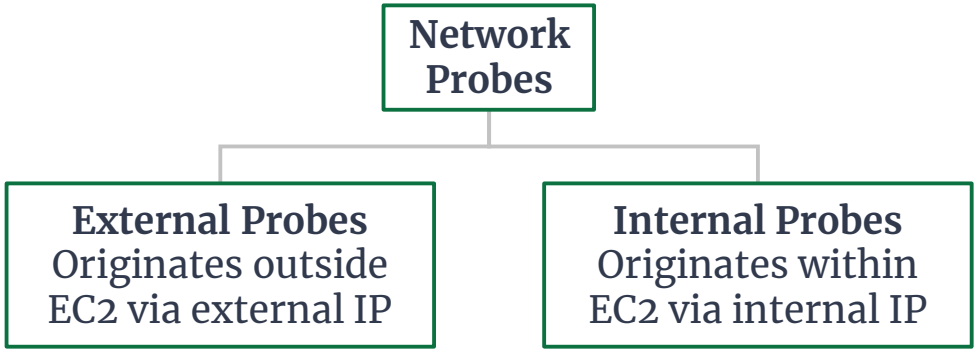
Network Probing

- ★ To estimate the VM placement pattern in the EC2 system
- ★ Used later to determine co-residence of an adversarial VM with a target VM

nmap → liveness probe (TCP Connect) to detect EC2 instances

hping → traceroute (iterative TCP SYN with increasing TTLs until no ACK) to determine co-resident VM placement

wget → retrieve web pages to establish public webserving target VMs in EC2



Cloud Cartography: Mapping

1. Surveying public servers on EC2

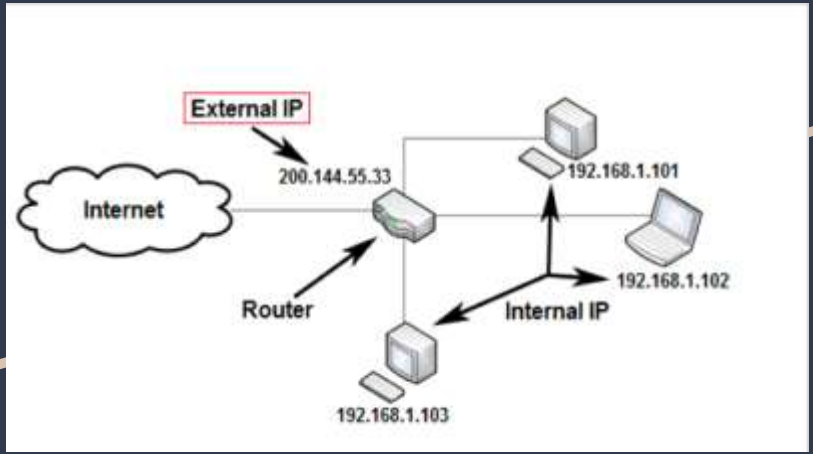
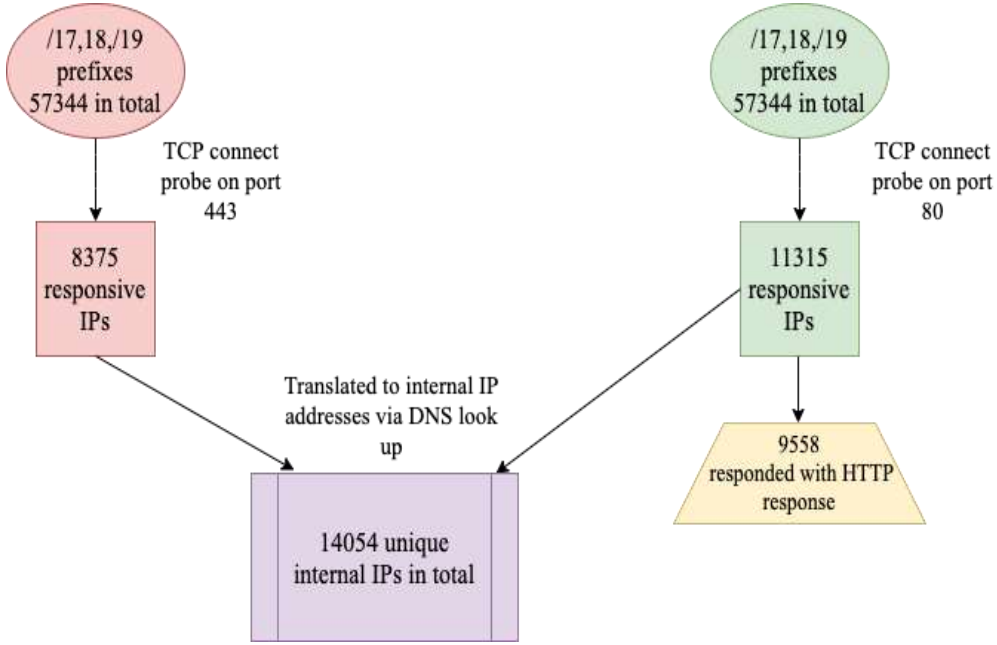


Image source: https://troypoint.com/internal-external-ip-address/

Experiment Conducted:

/16,/17,/18,/19 IPv4 prefixes associated with EC2

Enumerating public EC2-based web servers using external probes & translating responsive public IPs to internal IPs



Cloud Cartography:Mapping

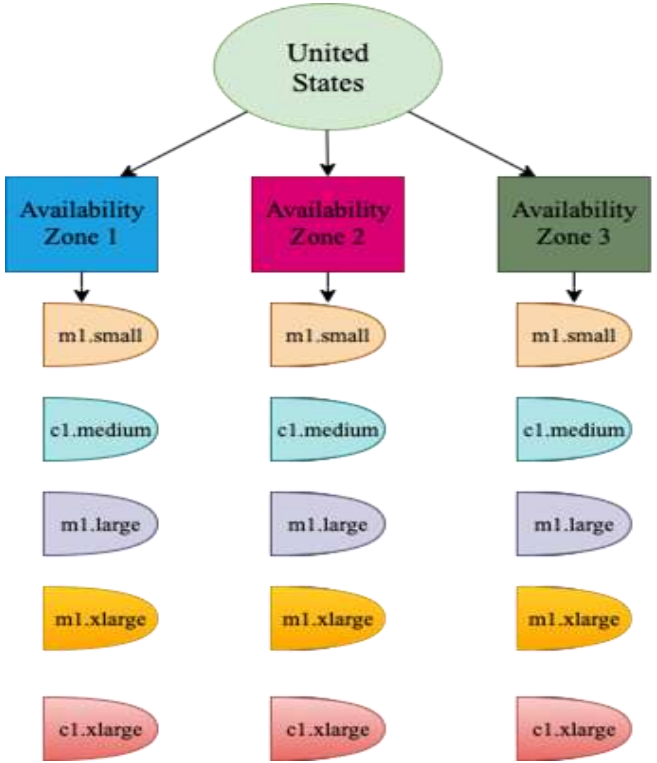
2. Instance Placement parameters

Binary Mask				Prefix Length	Subnet Mask
11111111	00000000	00000000	00000000	/8	255.0.0.0
11111111	10000000	00000000	00000000	/9	255.128.0.0
11111111	11000000	00000000	00000000	/10	255.192.0.0
11111111	11100000	00000000	00000000	/11	255.224.0.0
11111111	11110000	00000000	00000000	/12	255.240.0.0
11111111	11111000	00000000	00000000	/13	255.248.0.0
11111111	11111100	00000000	00000000	/14	255.252.0.0
11111111	11111110	00000000	00000000	/15	255.254.0.0
11111111	11111111	00000000	00000000	/16	255.255.0.0
11111111	11111111	10000000	00000000	/17	255.255.128.0
11111111	11111111	11000000	00000000	/18	255.255.192.0
11111111	11111111	11100000	00000000	/19	255.255.224.0
11111111	11111111	11110000	00000000	/20	255.255.240.0
11111111	11111111	11111000	00000000	/21	255.255.248.0
11111111	11111111	11111100	00000000	/22	255.255.252.0
11111111	11111111	11111110	00000000	/23	255.255.254.0
11111111	11111111	11111111	00000000	/24	255.255.255.0
11111111	11111111	11111111	10000000	/25	255.255.255.128
11111111	11111111	11111111	11000000	/26	255.255.255.192
11111111	11111111	11111111	11100000	/27	255.255.255.224
11111111	11111111	11111111	11110000	/28	255.255.255.240
11111111	11111111	11111111	11111000	/29	255.255.255.248
11111111	11111111	11111111	11111100	/30	255.255.255.252
11111111	11111111	11111111	11111110	/31	255.255.255.254
11111111	11111111	11111111	11111111	/32	255.255.255.255

Image source: https://www.benq.com/en-ap/business/support/faqs/application/publicdisplay-b2bfaq-k-00034.html

Account A:

20 instances for each, 300 in total.



Cloud Cartography

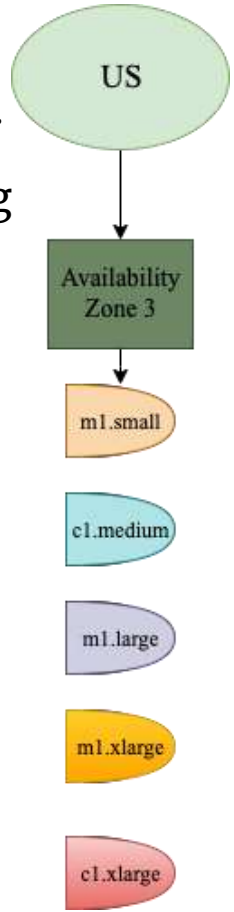
2. Instance Placement parameters

Binary Mask				Prefix Length	Subnet Mask
11111111	00000000	00000000	00000000	/8	255.0.0.0
11111111	10000000	00000000	00000000	/9	255.128.0.0
11111111	11000000	00000000	00000000	/10	255.192.0.0
11111111	11100000	00000000	00000000	/11	255.224.0.0
11111111	11110000	00000000	00000000	/12	255.240.0.0
11111111	11111000	00000000	00000000	/13	255.248.0.0
11111111	11111100	00000000	00000000	/14	255.252.0.0
11111111	11111110	00000000	00000000	/15	255.254.0.0
11111111	11111111	00000000	00000000	/16	255.255.0.0
11111111	11111111	10000000	00000000	/17	255.255.128.0
11111111	11111111	11000000	00000000	/18	255.255.192.0
11111111	11111111	11100000	00000000	/19	255.255.224.0
11111111	11111111	11110000	00000000	/20	255.255.240.0
11111111	11111111	11111000	00000000	/21	255.255.248.0
11111111	11111111	11111100	00000000	/22	255.255.252.0
11111111	11111111	11111110	00000000	/23	255.255.254.0
11111111	11111111	11111111	00000000	/24	255.255.255.0
11111111	11111111	11111111	10000000	/25	255.255.255.128
11111111	11111111	11111111	11000000	/26	255.255.255.192
11111111	11111111	11111111	11100000	/27	255.255.255.224
11111111	11111111	11111111	11110000	/28	255.255.255.240
11111111	11111111	11111111	11111000	/29	255.255.255.248
11111111	11111111	11111111	11111100	/30	255.255.255.252
11111111	11111111	11111111	11111110	/31	255.255.255.254
11111111	11111111	11111111	11111111	/32	255.255.255.255

Account B:

20 instances for each, 100 in total.

Created 39 hours after terminating account A.



Cloud Cartography: Results/ Fuller Map of EC2

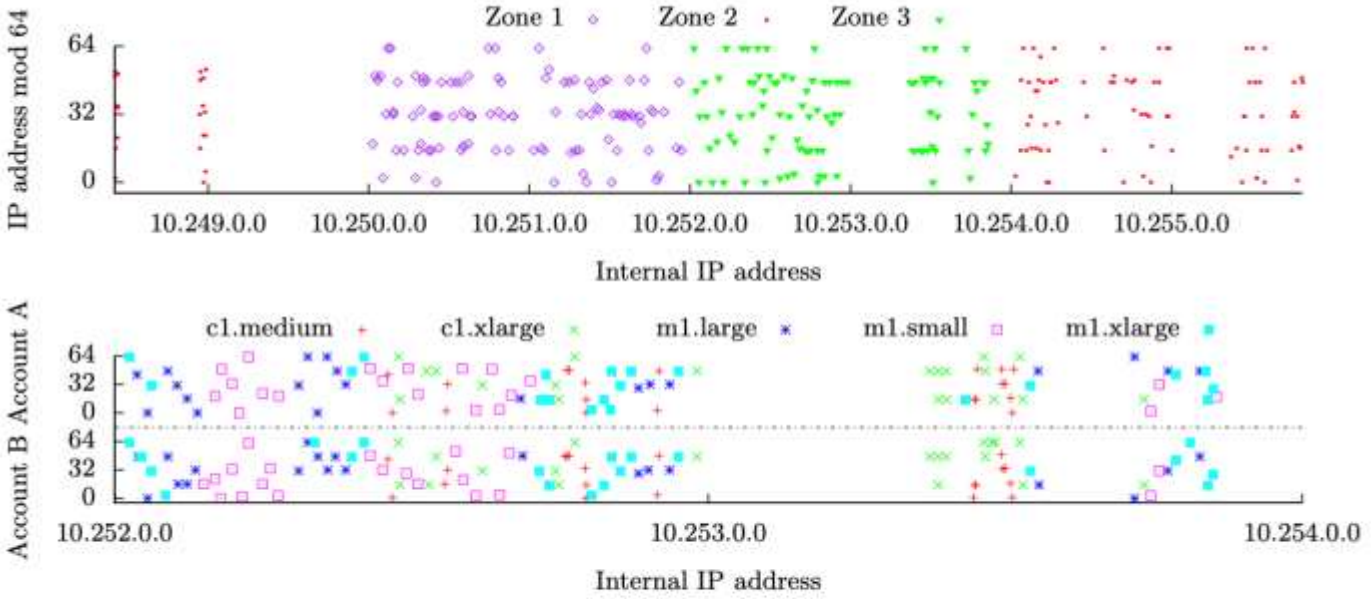


Figure 1: (Top) A plot of the internal IP addresses assigned to instances launched during the initial mapping experiment using Account A. (Bottom) A plot of the internal IP addresses of instances launched in Zone 3 by Account A and, 39 hours later, by Account B. Fifty-five of the Account B IPs were repeats of those assigned to instances for Account A.

Cloud Cartography: Results/ Fuller Map of EC2

- ★ No IP addresses were ever observed being assigned to more than one instance type.
- ★ All IPs from /16 are from same availability zone.
- ★ A /24 prefix inherits any included sampled instance type.
- ★ A /24 containing a Dom 0 IP address only contains Dom 0 IP addresses.
- ★ All /24's between two consecutive Dom 0 /24 inherit the former's associated type.

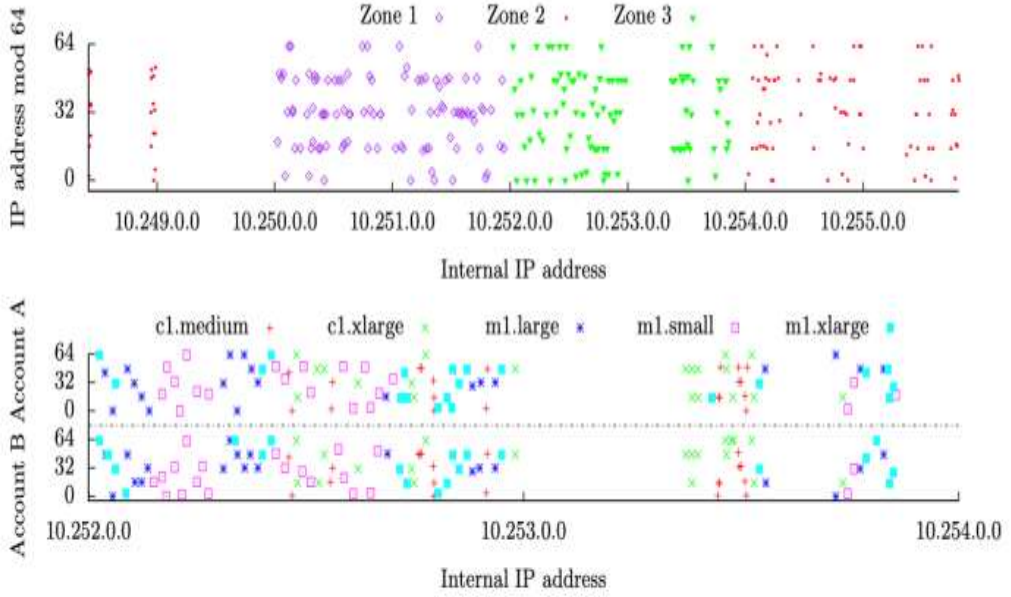


Figure 1: (Top) A plot of the internal IP addresses assigned to instances launched during the initial mapping experiment using Account A. (Bottom) A plot of the internal IP addresses of instances launched in Zone 3 by Account A and, 39 hours later, by Account B. Fifty-five of the Account B IPs were repeats of those assigned to instances for Account A.

Cloud Cartography: Results/ Fuller Map of EC2

- 869 /24 s in total
- Unique zone and type assigned to 723 of them
- Unique zone and two types assigned to 23 of these
- 123 left unlabeled

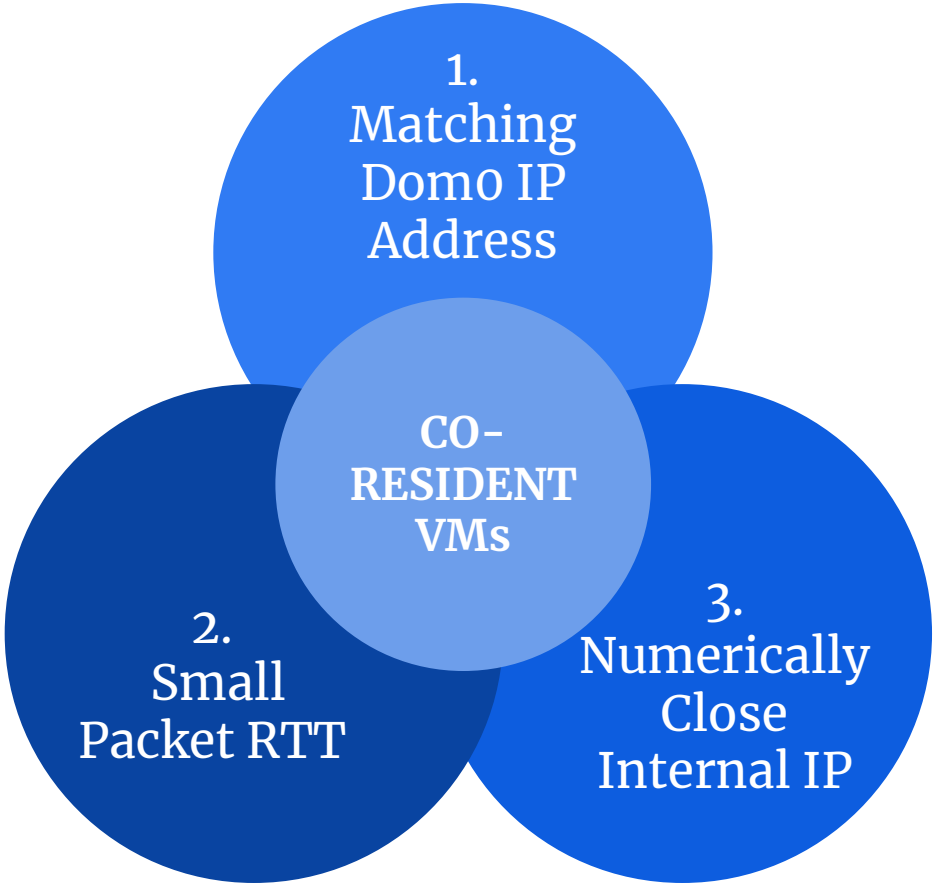
How to determine Co-residence of VM Instances?

Network-based co-residence checks

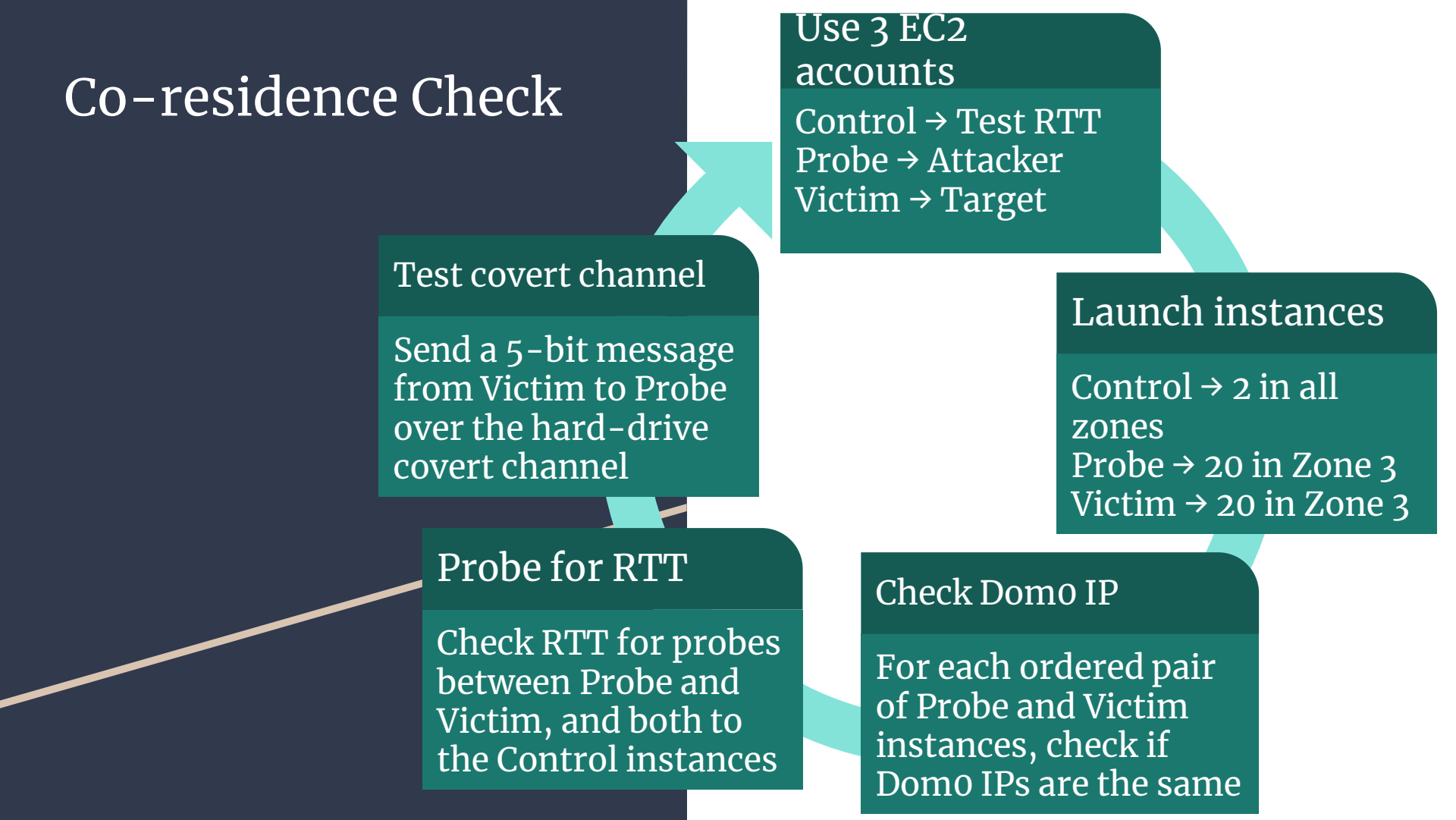
- ★ Instance's first hop is Dom0
- ★ Probe instance for Round Trip Time (RTT)
  - Use average RTT for 10 probes
- ★ Map internal IPs to Dom0 IP subnet
  - If IPs are close → co-resident instances

How to confirm co-residence?

- ★ Cross-VM covert channels
- ★ Hard disk-based



Co-residence Check



Co-residence Check (contd.)

1. Compare internal IP of instances for numerical closeness
2. Perform TCP SYN traceroute on an open port on the target
3. Check for a single hop through Domo IP address
4. The Probe instance is co-resident with the Target!

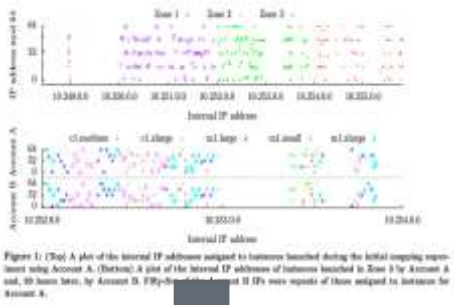
	Count	Median RTT (ms)
Co-resident instance	62	0.242
Zone 1 Control A	62	1.164
Zone 1 Control B	62	1.027
Zone 2 Control A	61	1.113
Zone 2 Control B	62	1.187
Zone 3 Control A	62	0.550
Zone 3 Control B	62	0.436

Figure 2: Median round trip times in seconds for probes sent during the 62 co-residence checks. (A probe to Zone 2 Control A timed out.)

Exploiting Placement in EC2:

1. Brute-Forcing Placement

Attacker enumerates a set of potential targets

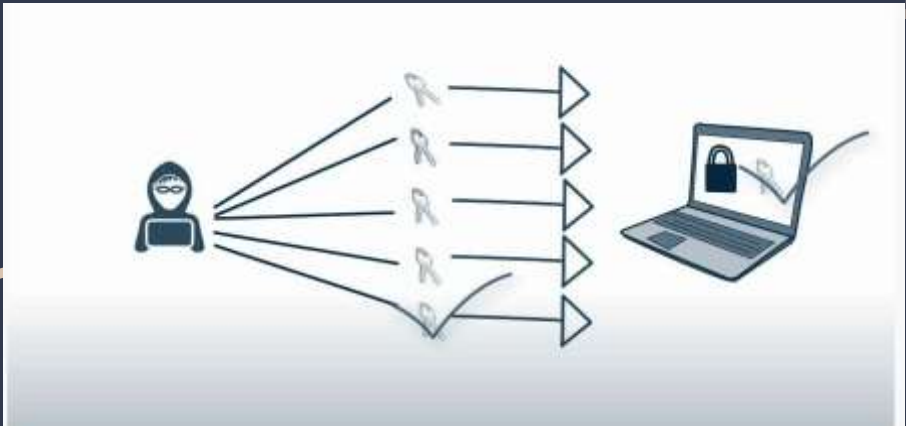


Attacker infers which of these targets belong to which availability zone and instance type by using the map

Attacker runs probe instances repeatedly in the target zone and of target type.
Each probe instances check co-residence and if not the instance is terminated quickly.

Exploiting Placement in EC2:

Results of Brute Forcing Strategy



1686 servers with zone 3 and m1.small instance type

Analyzed 1785 probe instances With 78 unique Dom 0 IPs

Achieved co-residency in 141 victim servers = 8.4% success rate

Exploiting Placement in EC2:

2. Abusing Placement Locality

- ★ Parallel placement locality
- ★ Instance flooding: running multiple instances in parallel.

Effect of time, account or zone?

Effect of increased time lag?

Exploiting Placement in EC2: Abusing Placement Locality (contd.)

- ★ M1.small instance type used for this experiment.
- ★ Availability zone does not meaningfully affect co-residency rates.

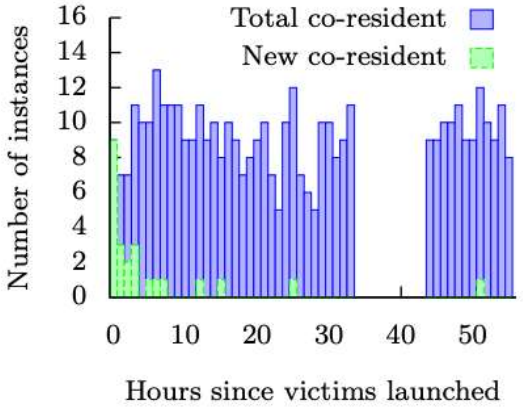
40% success rate

	# victims v	# probes p	coverage
Zone 1	1	20	1/1
	10	20	5/10
	20	20	7/20
Zone 2	1	20	0/1
	10	18	3/10
	20	19	8/20
Zone 3	1	20	1/1
	10	20	2/10
	20	20	8/20

Trial	Account A	Account B	Total
Midday (11:13 – 14:22 PST)	2 / 5	2 / 5	4/10
Afternoon (14:12 – 17:19 PST)	1 / 5	3 / 5	4/10
Night (23:18 – 2:11 PST)	2 / 5	2 / 5	4/10

Figure 3: (Left) Results of launching p probes 5 minutes after the launch of v victims. The rightmost column specifies success coverage: the number of victims for which a probe instance was co-resident over the total number of victims. (Right) The number of victims for which a probe achieved co-residence for three separate runs of 10 repetitions of launching 1 victim instance and, 5 minutes later, 20 probe instances. Odd-numbered repetitions used Account A; even-numbered repetitions used Account B.

Exploiting Placement in EC2: Results of Abusing Placement Locality



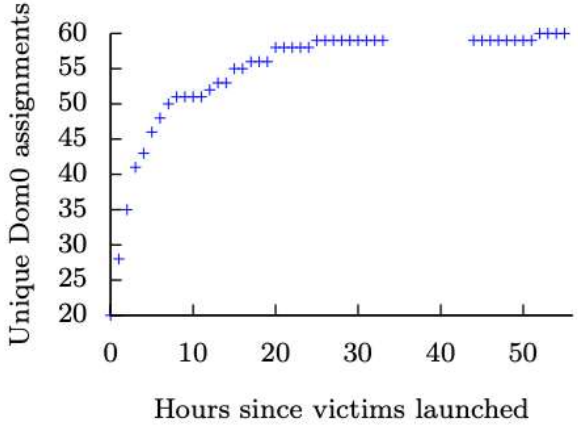


Figure 4: Results for the experiment measuring the effects of increasing time lag between victim launch and probe launch. Probe instances were not run for the hours 34–43. (Left) "Total co-resident" corresponds to the number of probe instances at the indicated hour offset that were co-resident with at least one of the victims. "New co-resident" is the number of victim instances that were collided with for the first time at the indicated hour offset. (Right) The cumulative number of unique Dom0 IP addresses assigned to attack instances for each round of flooding.

Exploiting Placement in EC2:

3. Patching Placement Vulnerabilities

- ★ Inhibit cartography
- ★ Inhibit co-residence checking
- ★ Offload placement choice to users

amazon web services™ | EC2

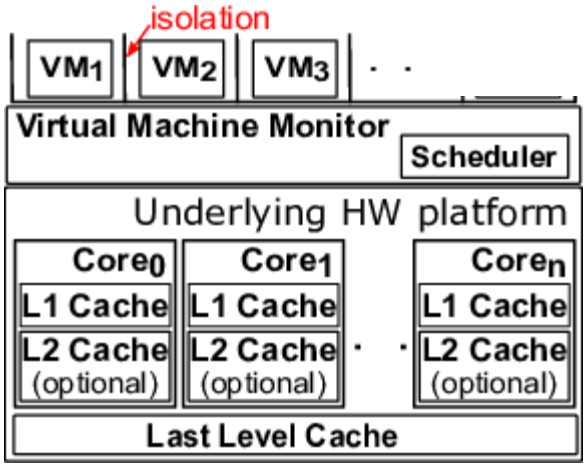


Image source: https://www.researchgate.net/figure/Isolation-in-a-virtualized-environment_fig1_292139909

Cross-VM Information Leakage

- ★ Exploit side-channels to learn information about co-resident instances.
  - Time-shared caches can leak information about computational load
- ★ Side-channel applications explored:
  - Robust co-residence detection (agnostic to network configuration)
  - Surreptitious detection of web traffic rates on a co-resident victim instance
  - Timing keystrokes by an unsuspecting user accessing a co-resident instance via SSH
    - Studied on an EC2-like virtualized environment
- ★ Coarse-grained side channel attacks!
  - No ability to extract cryptographic keys :(
  - Extract less bits of information :|
  - More robust; simpler to implement in noisy environments :)

How to Orchestrate Cross-VM Side-Channel Attacks?

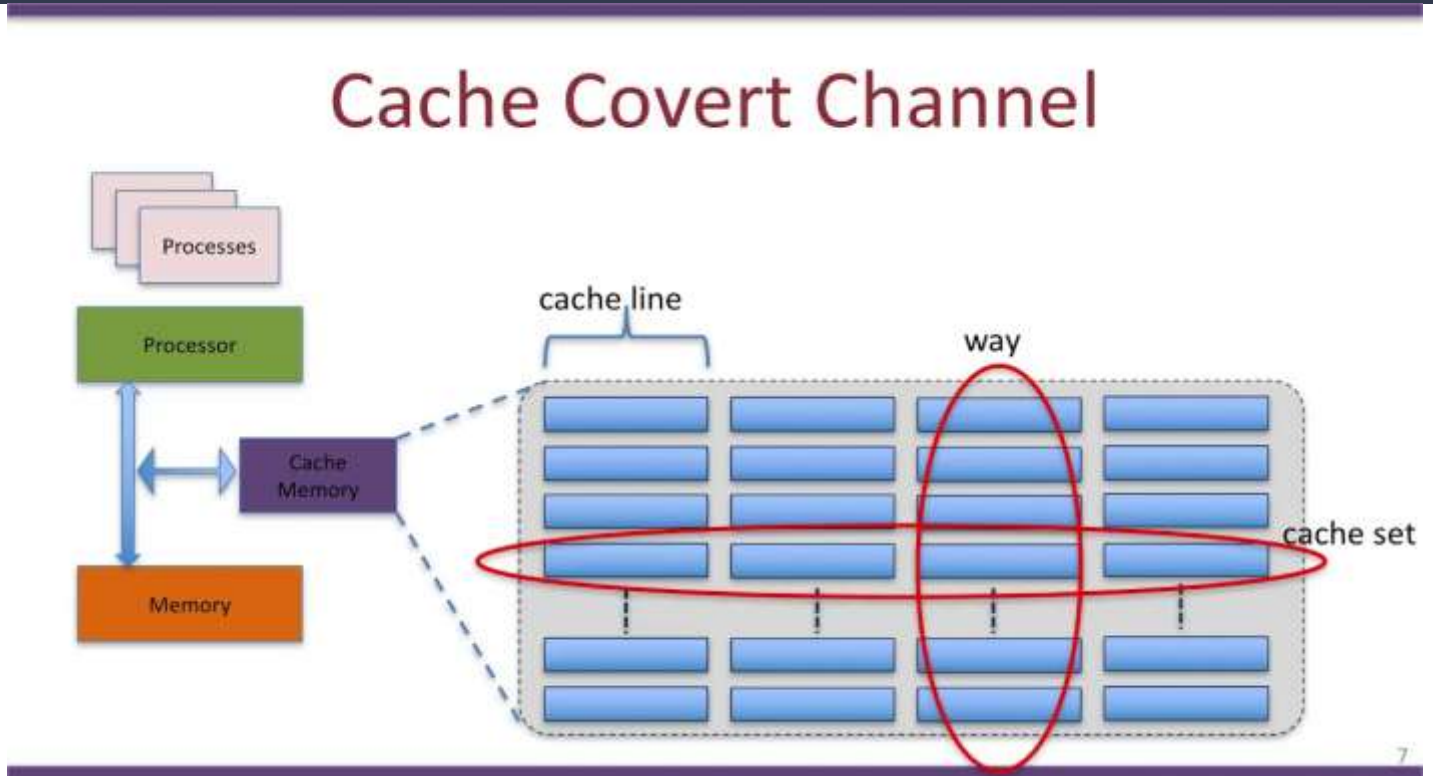


Image source: https://www.youtube.com/watch?v=RmJ4E4OwKXY (thumbnail)

CPU Load Measurement: Prime+Trigger+Probe

Cache utilization → good indicator of CPU load → activity in co-resident instances

Exploit statistical delay in cache miss times to establish covert channel.

Contiguous buffer B (b bytes) → b ≈ cache size; Cache line size (s bytes)

Enhanced P+T+P Technique

- ★ Use differential coding for noise resilience
- ★ Cache partitioning into associativity sets
  - “Odd sets” and “even sets”
- ★ Three parameters:
  - a → larger than attacked cache level
  - b → slightly smaller than attacked cache level
  - d → n^2 * (cache line size)

- ★ Even addresses → addresses == 0 mod $2d$
- ★ Odd addresses → addresses == d mod $2d$

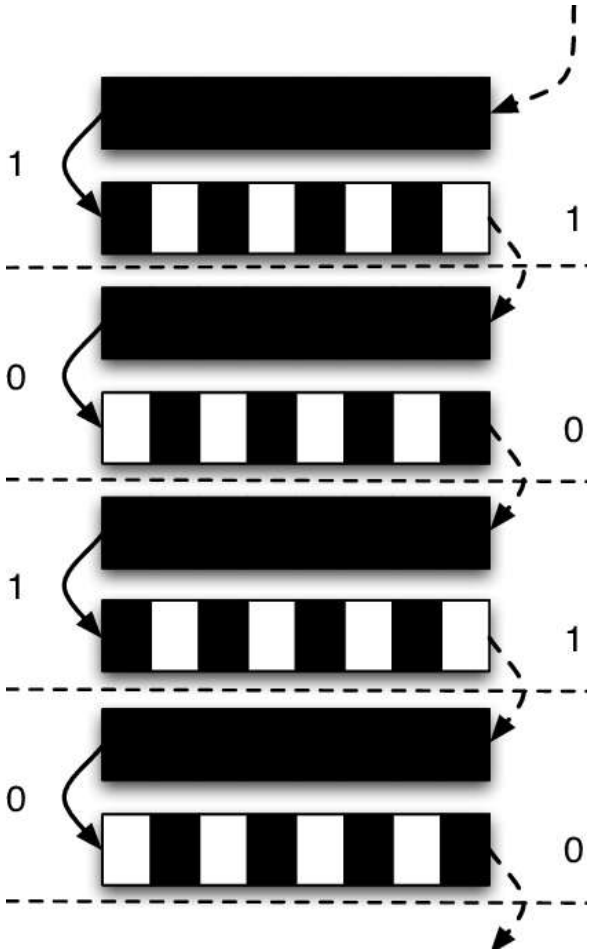


Image source: https://www.researchgate.net/figure/An-illustration-of-a-covert-channel-using-L2-cImage ache-to-encode-information-For-each-bit_fig2_221609235

Load-based Co-residence Detection

Setup: m1.small instances, Fedora Core 4 with Apache 2.0, 1024 byte text-only HTML page

1. Attacker VM takes 100 load samples (b= 768 * 1024, s = 128); ≈ 12 seconds
2. Pause for 10 seconds
3. Repeat 1 while making HTTP GET requests to the Target VM via jmeter 2.3.4 (100 user threads)

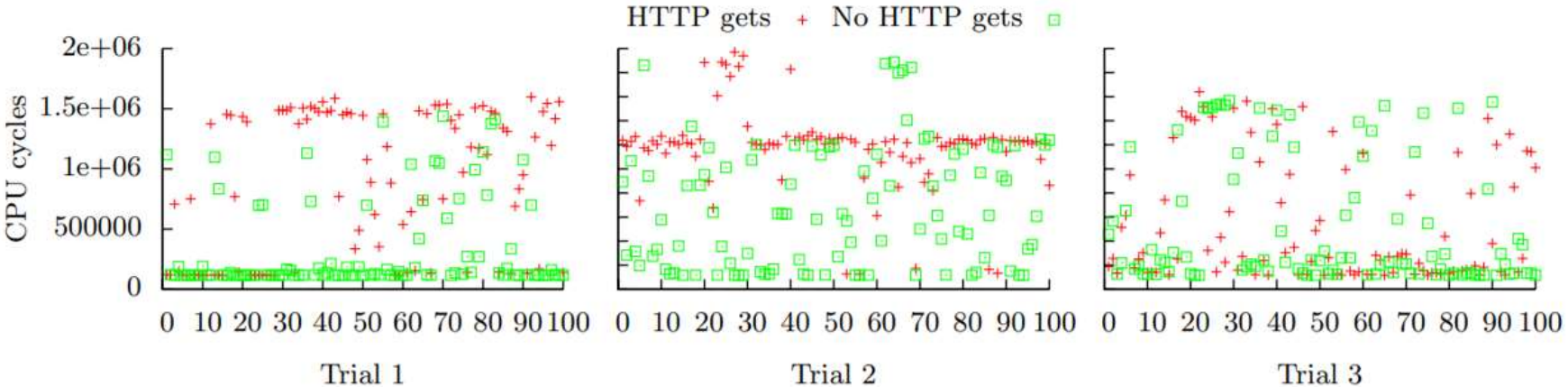


Figure 5: Results of executing 100 Prime+Trigger+Probe cache timing measurements for three pairs of m1.small instances, both when concurrently making HTTP get requests and when not. Instances in Trial 1 and Trial 2 were co-resident on distinct physical machines. Instances in Trial 3 were not co-resident.

Estimating Traffic Rates

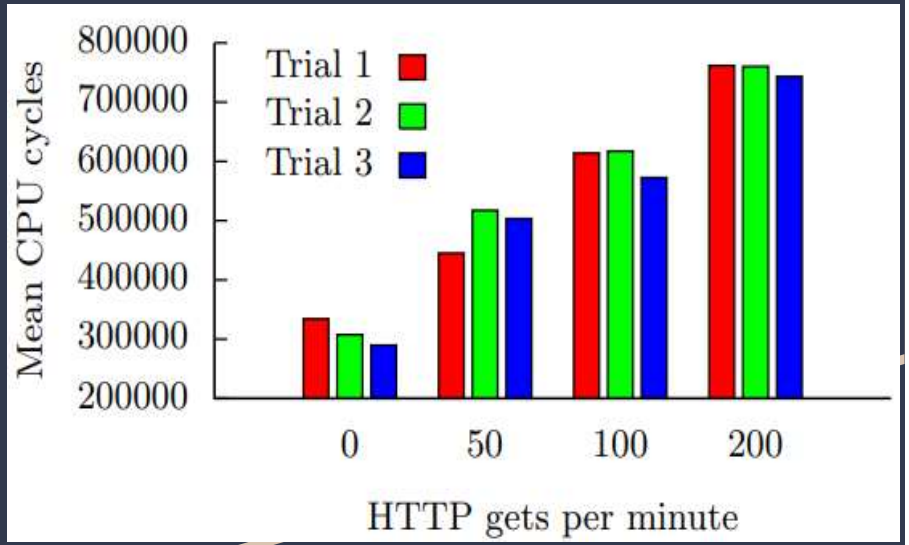


Figure 6: Mean cache load measurement timings (over 1 000 samples) taken while differing rates of web requests were made to a 3 megabyte text file hosted by a co-resident web server.

Experiment Setup:

- ➔ Two m1.small instances
- ➔ Four separate runs of 1000 cache load measurements
- ➔ 1000 measurements ≈ 90 seconds
- ➔ Requested web page: 3 MB text file
- ➔ jmeter with 20 users requesting
- ➔ Vary HTTP requests in each run:
  - ◆ No HTTP requests sent
  - ◆ HTTP requests sent @ 50 per minute
  - ◆ HTTP requests sent @ 100 per minute
  - ◆ HTTP requests sent @ 200 per minute
- ➔ Experiment repeated thrice; results averaged

Keystroke Timing Attack

Experiment Environment: Local testbed with EC2-like configuration

- ★ Opteron CPUs, Xen hypervisor
- ★ Linux kernels similar to EC2-hosted m1.small instances
- ★ VMs pinned to specific cores → no migration between vCPUs
  - Machine completely idle other than test code
  - Ability to discern relation between VMs involved in the attack (e.g time-shared cores or not)
- ★ Keystrokes detected using simple thresholding
  - E.g. reporting a keystroke when the probing measurement is between 3.1 µs and 9 µs
- ★ Variants can exploit either L1 or L2 cache contention
- ★ Attacker can observe a clear signal with 5% missed keystrokes & 0.3 false triggers/s
- ★ Timing resolution ≈ 13 ms

Inhibiting Side-channel Attacks:

Key Takeaways

- ★ Blinding techniques → minimize leakable information
  - E.g. cache wiping, random delay insertion, adjusting perception of time at each instance, etc.
- ★ Consider countermeasures to side-channel attacks
  - Difficult for practical implementation
    - High overhead of implementation or requires using non-standard hardware
    - Application-specific, or insufficient for complete risk mitigation
  - Requires discovery, anticipation, and patchability of *all* possible side-channels :(

- ★ AVOIDING co-residence, based on the *current* state of the art.

But... What is the *current* state of the art (in 2022?!)

AWS EC2 Infrastructure (Present)

- Regions
- Availability Zones
- Local Zones
- Wavelength Zones
- AWS Outposts

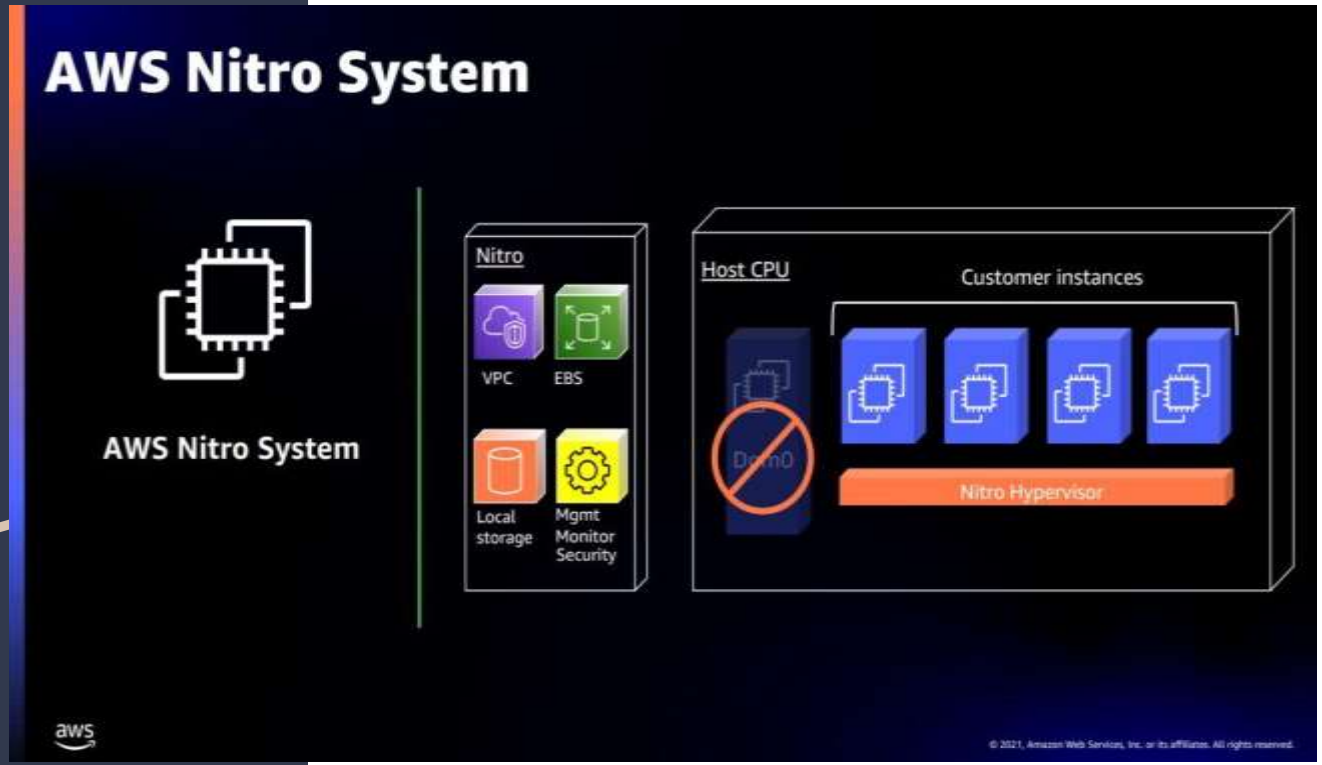
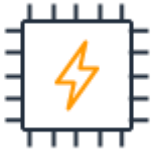


Image source: https://www.servethehome.com/aws-nitro-the-big-cloud-dpu-deployment-detailed/

AWS Nitro System: Key Features

Nitro Cards



Nitro Security Chip

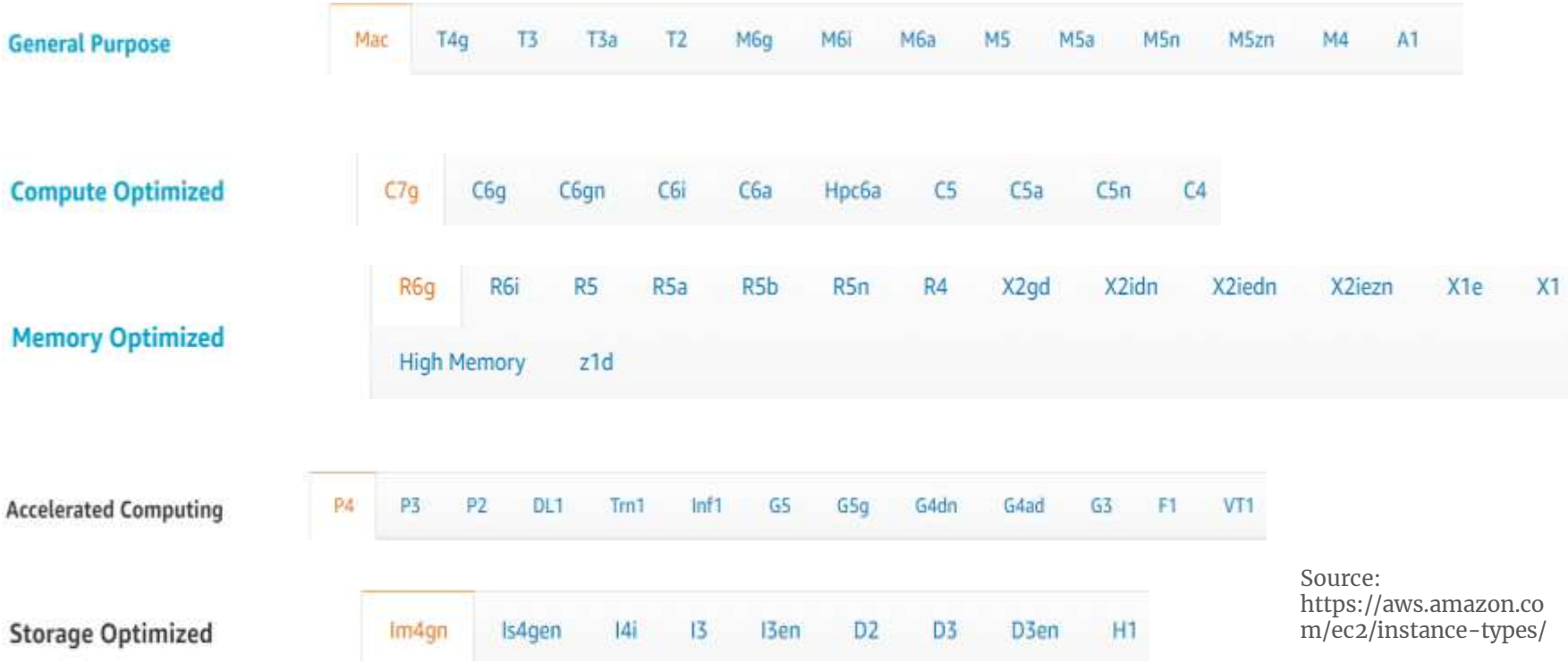
Nitro Hypervisor

AWS Nitro Enclaves

NitroTPM (Coming soon in 2022)

Source: https://aws.amazon.com/ec2/nitro/

AWS EC2: Current *Instance Types*

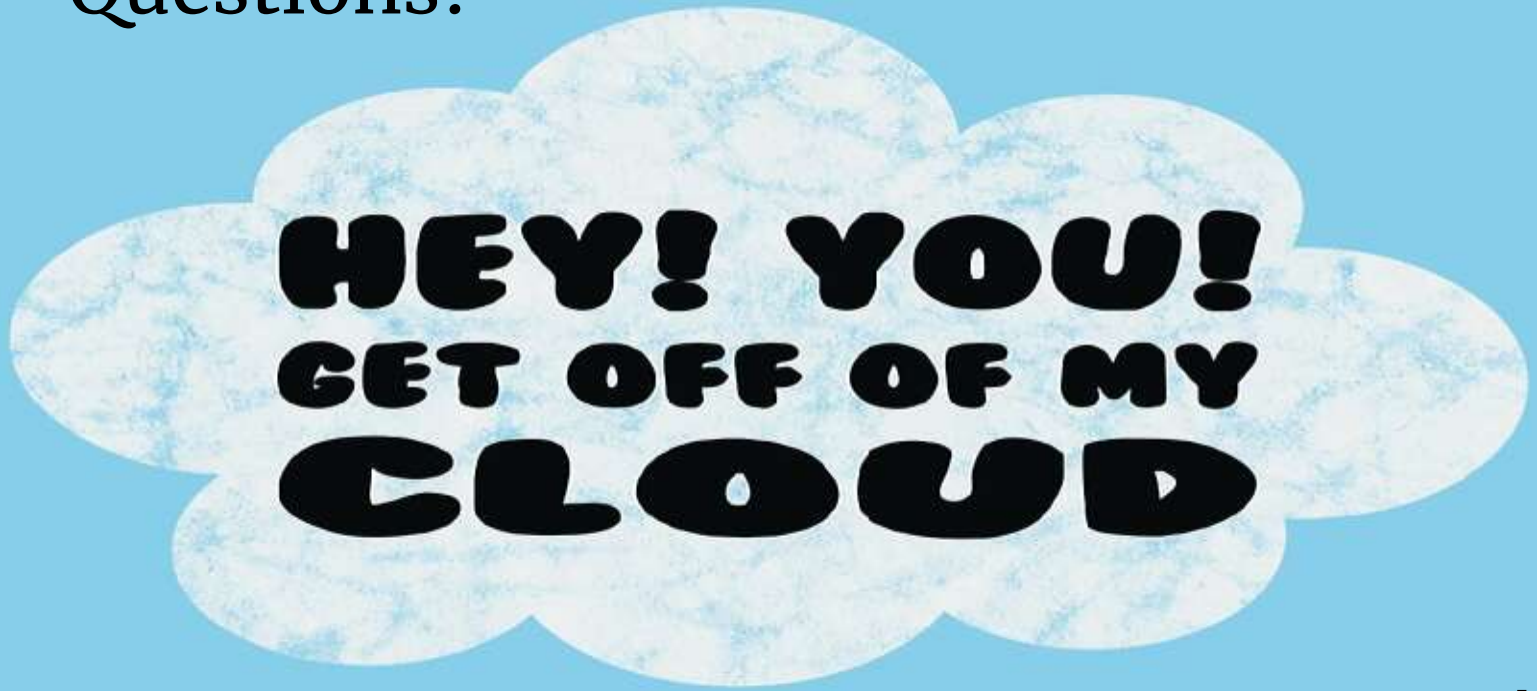


Source: https://aws.amazon.com/ec2/instance-types/

Conclusion & Discussion

- ★ Obfuscate internal structure of cloud infrastructure
- ★ Adopt an obscure/non-predictable instance placement policies
- ★ Inhibit simple network-based co-residence checks
- ★ Reduce the attack surface for side-channel vulnerabilities
- ★ Employ blinding techniques minimize information leakage
- ★ Expose any unmitigable risk to the user
- ★ Provide placement decisions directly to the users (dedicated resources)

Questions?



Thank you!