IIT KHARAGPUR

NPTEL ONLINE CERTIFICATION COURSES

NPTEL

CLOUD COMPUTING

CLOUD SECURITY III

PROF. SOUMYA K. GHOSH
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
IIT KHARAGPUR

Research Article

- Research Paper:
  - *Hey, You, Get Off of My Cloud! Exploring Information Leakage in Third-Party Compute Clouds*. by Thomas Ristenpart, Eran Tromer, Hovav Shacham, and Stefan Savage. In Proceedings of CCS 2009, pages 199–212. ACM Press, Nov. 2009.
  - First work on *cloud cartography*
    - Attack launched against commercially available “real” cloud (Amazon EC2)
    - Claims up to 40% success in co-residence with target VM

New Risks in Cloud

- Trust and dependence
  - Establishing new trust relationship between customer and cloud provider
  - Customers must trust their cloud providers to respect the privacy of their data and integrity of their computations
- Security (multi-tenancy)
  - Threats from other customers due to the subtleties of how physical resources can be transparently shared between virtual machines (VMs)

Multi-tenancy

- Multiplexing VMs of disjoint customers upon the same physical hardware
  - Your machine is placed on the same server with other customers
  - Problem: you don't have the control to prevent your instance from being co-resident with an adversary
- New risks
  - Side-channels exploitation
    - Cross-VM information leakage due to sharing of physical resource (e.g., CPU's data caches)
    - Has the potential to extract RSA & AES secret keys
  - Vulnerable VM isolation mechanisms
    - Via a vulnerability that allows an "escape" to the hypervisor
  - Lack of control who you're sharing server space

Attack Model

- Motivation
  - To study practicality of mounting cross-VM attacks in existing third-party compute clouds
- Experiments have been carried out on real IaaS cloud service provider (Amazon EC2)
- Two steps of attack:
  - *Placement*: adversary arranging to place its malicious VM on the same physical machine as that of the target customer
  - *Extraction*: extract confidential information via side channel attack

Threat Model

- Assumptions of the threat model:
  - Provider and infrastructure to be trusted
  - Do not consider attacks that rely on subverting administrator functions
  - Do not exploit vulnerabilities of the virtual machine monitor and/or other software
  - Adversaries: non-providers-affiliated malicious parties
  - Victims: users running confidentiality-requiring services in the cloud
- Focus on new cloud-related capabilities of the attacker and implicitly expanding the attack surface

Threat Model *(contd...)*

- Like any customer, the malicious party can run and control many instances in the cloud
  - Maximum of 20 instances can be run parallel using an Amazon EC2 account
- Attacker's instance might be placed on the same physical hardware as potential victims
- Attack might manipulate shared physical resources to learn otherwise confidential information
- Two kinds of attack may take place:
  - Attack on some known hosted service
  - Attacking a particular victim's service

Addresses the Following…

- *Q1*: Can one determine where in the cloud infrastructure an instance is located?
- *Q2*: Can one easily determine if two instances are co-resident on the same physical machine?
- *Q3*: Can an adversary launch instances that will be co-resident with other user’s instances?
- *Q4*: Can an adversary exploit cross-VM information leakage once co-resident?

Amazon EC2 Service

- Scalable, pay-as-you-go compute capacity in the cloud
- Customers can run different operating systems within a virtual machine
- Three degrees of freedom: *instance-type, region, availability zone*
- Different computing options (instances) available
  - m1.small, c1. medium: 32-bit architecture
  - m1.large, m1.xlarge, c1.xlarge: 64-bit architecture
- Different regions available
  - US, EU, Asia
- Regions split into availability zones
  - In US: East (Virginia), West (Oregon), West (Northern California)
  - Infrastructures with separate power and network connectivity
- Customers randomly assigned to physical machines based on their instance, region, and availability zone choices

Amazon EC2 Service *(contd...)*

- Xen hypervisor
  - Domain0 (Dom0): privileged virtual machine
    - Manages guest images
    - Provisions physical resources
    - Access control rights
    - Configured to route packets for its guest images and reports itself as a hop in traceroutes.
  - When an instance is launched, it is assigned to a single physical machine for its lifetime
- Each instance is assigned internal and external IP addresses and domain names
  - *External IP*: public IPv4 address [IP: **75.101.210.100**/domain name: **ec2-75-101-210-100.compute-1.amazonaws.com**]
  - *Internal IP*: RFC 1918 private address [IP: **10.252.146.52**/domain name: **domU-12-31-38-00-8D-C6.compute-1.internal**]
- Within the cloud, both domain names resolve to the internal IP address
- Outside the cloud, external name is mapped to the external IP address

Q1: Cloud Cartography

- Instance placing is not disclosed by Amazon but is needed to launch co-residency attack
- Map the EC2 service to understand where potential targets are located in the cloud
- Determine instance creation parameters needed to attempt establishing co-residence of an adversarial instance
- Hypothesis: *different availability zones and instance types correspond to different IP address ranges*

Network Probing

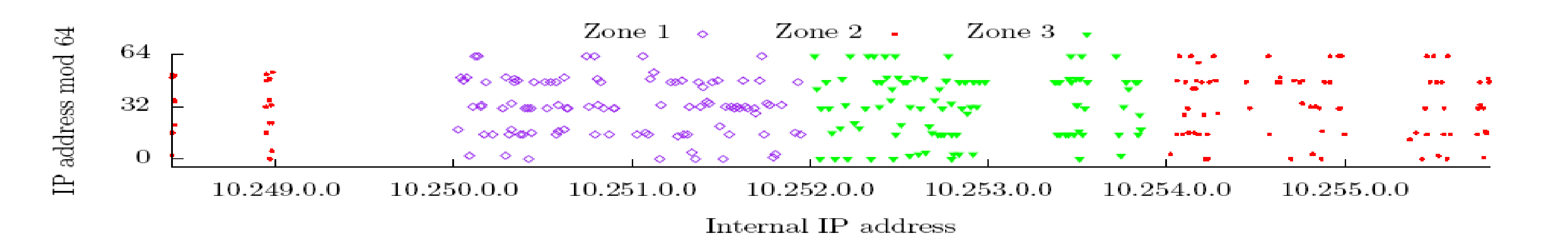
- Identify public servers hosted in EC2 and verify co-residence
- Open-source tools have been used to probe ports (80 and 443)
  - **nmap** – perform TCP connect probes (attempt to complete a 3-way hand-shake between a source and target)
  - **hping** – perform TCP SYN traceroutes, which iteratively sends TCP SYN packets with increasing TTLs, until no ACK is received
  - **wget** – used to retrieve web pages
- *External probe*: probe originating from a system outside EC2 and has an EC2 instance as destination
- *Internal probe*: originates from an EC2 instance, and has destination another EC2 instance
- Given an external IP address, DNS resolution queries are used to determine:
  - External name
  - Internal IP address

Survey Public Servers on EC2

- Goal: to enable identification of the instance type and availability zone of one or more potential targets
- *WHOIS:* used to identify distinct IP address prefixes associated with EC2
- EC2 public IPs: /17, /18, /19 prefixes
  – 57344 IP addresses
- Use external probes to find responsive IPs:
  – Performed *TCP connect probe* on port 80
    - 11315 responsive IPs
  – Followed up with *wget* on port 80
    - 9558 responsive IPs
  – Performed a *TCP scan* on port 443
    - 8375 responsive IPs
- Used DNS lookup service
  – Translate each public IP address that responded to either the port 80 or 443 scan into an internal EC2 address
  – 14054 unique internal IPs obtained

Instance Placement Parameters

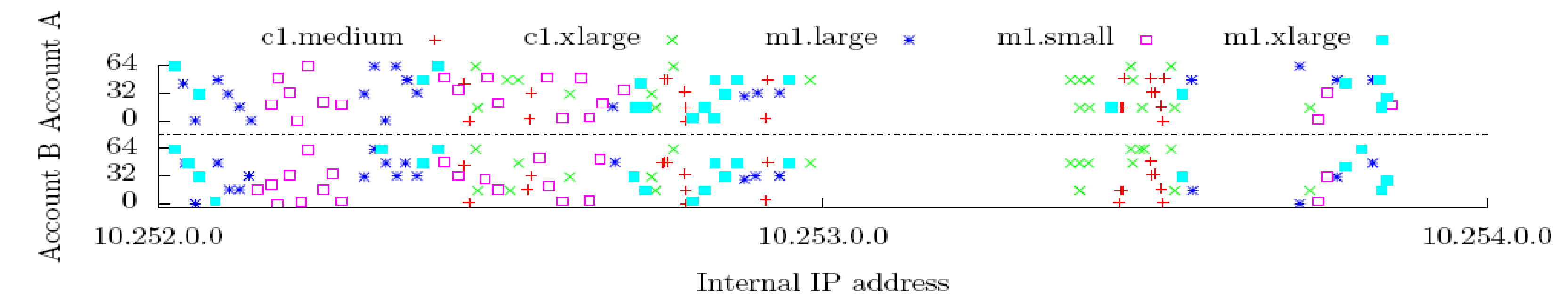
- EC2's internal address space is cleanly partitioned between availability zones
  - Three availability zone; five instance-type/zone
  - 20 instances launched for each of the 15 availability zone/instance type pairs from a particular account (Say, Account A)



- Samples from each zone are assigned IP addresses from disjoint portions of the observed internal address space
- **Assumption**: internal IP addresses are statically assigned to physical machines
  - To ease out IP routing
- Availability zones use separate physical infrastructure

Instance Placement Parameters *(contd...)*

- 100 instances have been launched in Zone 3 using two different accounts: A & B (39 hours after terminating the Account A instances)



- Of 100 Account A Zone 3 instances
  - 92 had unique /24 prefixes
  - Four /24 prefixes had two instances each
- Of 100 Account B Zone 3 instances
  - 88 had unique /24 prefixes
  - Six of the /24 prefixes had two instances each
- A single /24 had both an m1.large and m1.xlarge instance
- Of 100 Account B IP's, 55 were repeats of IP addresses assigned to instances for Account A

Q2: Determining Co-residence

- Network-based co-residency checks: instances are likely to be co-resident if they have-
  - **Matching Dom0 IP address**: determine an uncontrolled instance's Dom0 IP by performing a *TCP SYN* traceroute to it from another instance and inspect the last hop
  - **Small packet round-trip times**: 10 probes were performed and the average is taken
  - **Numerically close internal IP addresses (e.g., within 7)**: the same Dom0 IP will be shared by instances with contiguous sequence of internal IP addresses

Verifying Co-residency Check

- If two (under self-control) instances can successfully transmit via the covert channel, then they are co-resident, otherwise not
- Experiment: hard-disk-based covert channel
  - To send a 1, sender reads from random locations on a shared volume, to send a 0 sender does nothing
  - Receiver times reading from a fixed location on the disk: longer read times mean a 1 is set, shorter a 0
- 3 m1.small EC2 accounts: *control, victim, probe*
  - 2 control instances in each of 3 availability zones, 20 victim and 20 probe instances in Zone 3
- Determine *Dom0* address for each instance
- For each ordered pair (A, B) of 40 instances, perform co-residency checks
- After 3 independent trials, 31 (potentially) co-resident pairs have been identified - 62 ordered pairs
- *5 bit* message from A to B was successfully sent for 60 out of 62 ordered pairs

Effective Co-residency Check

- For checking co-residence with target instances:
  - Compare internal IP addresses to see if they are close
  - If yes, perform a TCP SYN traceroute to an open port on the target and see if there is only a single hop (Dom0 IP)
    - Check requires sending (at most) two *TCP SYN* packets
      - No full TCP connection is established
    - Very "quiet" check (little communication with the victim)

Q3: Causing Co-residence

- Two strategies to achieve “good” coverage (co-residence with a good fraction of target set)
  - Brute-force placement:
    - run numerous *probe* instances over a long period of time and see how many targets one can achieve co-residence with.
    - For co-residency check, the probe performed a wget on port 80 to ensure the target was still serving web pages
    - Of the 1686 target victims, the brute-force probes achieved co-residency with 141 victim servers (8.4% coverage)
    - Even a naïve strategy can successfully achieve co-residence against a not-so-small fraction of targets
  - Target recently launched instances:
    - take advantage of the tendency of EC2 to assign fresh instances to small set of machines

Leveraging Placement Locality

- Placement locality
  - Instances launched simultaneously from same account do not run on the same physical machine
  - *Sequential placement locality*: exists when two instances run sequentially (the first terminated before launching the second) are often assigned to the same machine
  - *Parallel placement locality*: exists when two instances run (from distinct accounts) at roughly the same time are often assigned to the same machine.
- *Instance flooding*: launch lots of instances in parallel in the appropriate availability zone and of the appropriate type

Leveraging Placement Locality (contd...)

- Experiment
  - Single victim instance is launched
  - Attacker launches 20 instances within 5 minutes
  - Perform co-residence check
  - 40% of the time the attacker launching just 20 probes achieves co-residence against a specific target instance

Q4: Exploiting Co-residence

- Cross-VM attacks can allow for information leakage
- How can we exploit the shared infrastructure?
  - Gain information about the resource usage of other instances
  - Create and use covert channels to intentionally leak information from one instance to another
  - Some applications of this covert channel are:
    - Co-residence detection
    - Surreptitious detection of the rate of web traffic a co-resident site receives
    - Timing keystrokes by an honest user of a co-resident instance

Exploiting Co-residence (contd…)

- Measuring cache usage
  - Time-shared cache allows an attacker to measure when other instances are experiencing computational load
  - Load measurement: allocate a contiguous buffer B of *b* bytes, *s* is cache line size (in bytes)
    - *Prime*: read B at *s*-byte offsets in order to ensure that it is cached.
    - *Trigger*: busy-loop until CPU's cycle counter jumps by a large value
    - *Probe*: measure the time it takes to again read B at *s*-byte offset
  - Cache-based covert channel:
    - Sender idles to transmit a 0 and frantically accesses memory to transmit a 1
    - Receiver accesses a memory block and observes the access latencies
    - High latencies are indicative that "1" is transmitted

Exploiting Co-residence (contd…)

- Load-based co-residence check
  - Co-residence check can be done without network- base technique
  - Adversary can actively cause load variation due to a publicly-accessible service running on the target
  - Use a priori knowledge about load variation
  - Induce computational load (lots of HTTP requests) and observe the differences in load samples

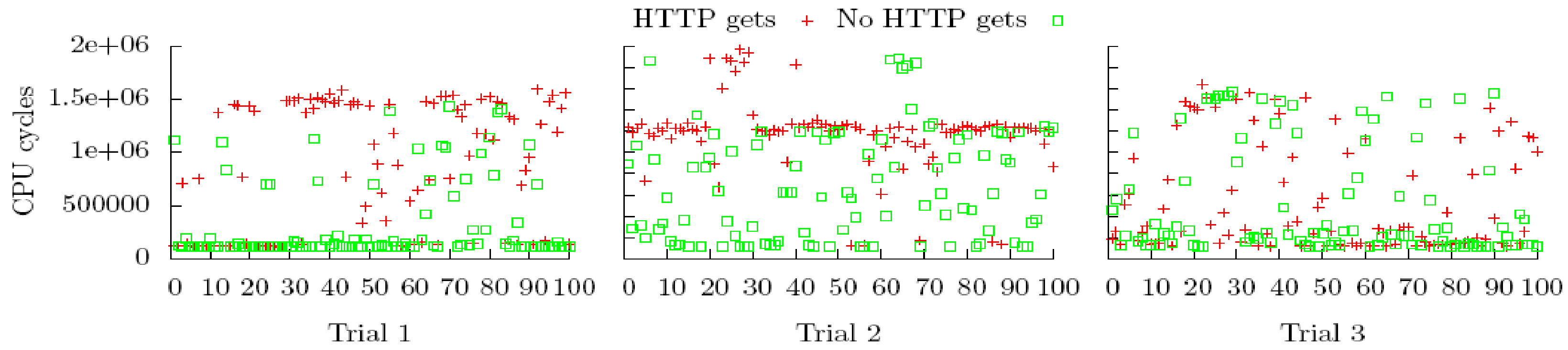


Figure 5: Results of executing 100 Prime+Trigger+Probe cache timing measurements for three pairs of m1.small instances, both when concurrently making HTTP get requests and when not. Instances in Trial 1 and Trial 2 were co-resident on distinct physical machines. Instances in Trial 3 were not co-resident.

- Instances in Trial 1 and Trial 2 were co-resident on distinct physical machines; instances in Trial 3 were not co-resident

Exploiting Co-residence (contd...)

- Estimating traffic rates
  - Load measurement might provide a method for estimating the number of visitors to a co-resident web server
  - It might not be a public information and could be damaging
  - Perform 1000 cache load measurements in which
    - no HTTP requests are sent
    - HTTP requests sent at a rate of (i) 50 per minute, (ii) 100 per minute, (iii) 200 per minutes

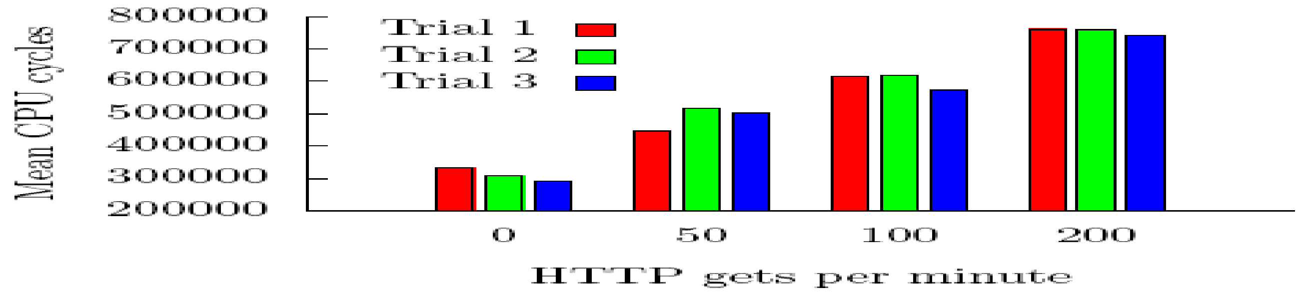


Figure 6: Mean cache load measurement timings (over 1 000 samples) taken while differing rates of web requests were made to a 3 megabyte text file hosted by a co-resident web server.

Exploiting Co-residence (contd...)

- Keystroke timing attack
  - The goal is to measure the time between keystrokes made by a victim typing a password (or other sensitive information)
  - Malicious VM can observe keystroke timing in real time via cache-based load measurements
  - Inter-keystroke times if properly measures can be used to perform recovery of the password
  - In an otherwise idle machine, a spike in load corresponds to a letter being typed into the co-resident VM's terminal
  - Attacker does not directly learn exactly which keys are pressed, the attained timing resolution suffices to conduct the password-recovery attacks on SSH sessions

Preventive Measures

- Mapping
  - Use a randomized scheme to allocate IP addresses
  - Block some tools (nmap, traceroute)
- Co-residence checks
  - Prevent identification of Dom0
- Co-location
  - Not allow co-residence at all
    - Beneficial for cloud user
    - Not efficient for cloud provider
- Information leakage via side-channel
  - No solution

Summary

- New risks from cloud computing
- Shared physical infrastructure may and most likely will cause problems
  - Exploiting software vulnerabilities not addressed here
- Practical attack performed
- Some countermeasures proposed

Thank You!