Security of Cloud Computing

Fabrizio Baiardi
f.baiardi@unipi.it

Syllabus

- Cloud Computing Introduction
  - Definitions
  - Economic Reasons
  - Service Model
  - Deployment Model
- Supporting Technologies
  - Virtualization Technology
  - Scalable Computing = Elasticity
- Security
  - New Threat Model
  - New Attacks
  - Countermeasures

Cartography: a basis for several attacks

A system where legal user and attacker share the same architecture is the target of new attacks that try to discover and monitor the flows of information

- among VMs, application, platforms
- between the browser and the cloud

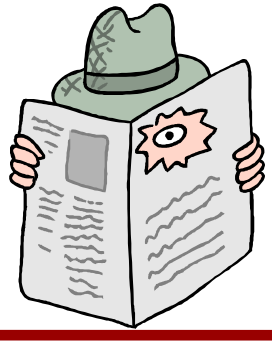
- a precondition of the previous attacks is the discovery of the allocation of VMs onto physical node to deduce physical resources shared among VM = cartography
- a further class of attacks targets the tools to interact with the cloud system (= the browser) and will be discussed in the following

Challenges for the attacker

How to find out **where** the target is located

How to be **co-located** with the target in the same (physical) machine

How to gather information about the target



Cloud Cartography

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

Ristenpart et al., CCS 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

Amazon EC2 (from the site)

To use Amazon EC2, you simply:

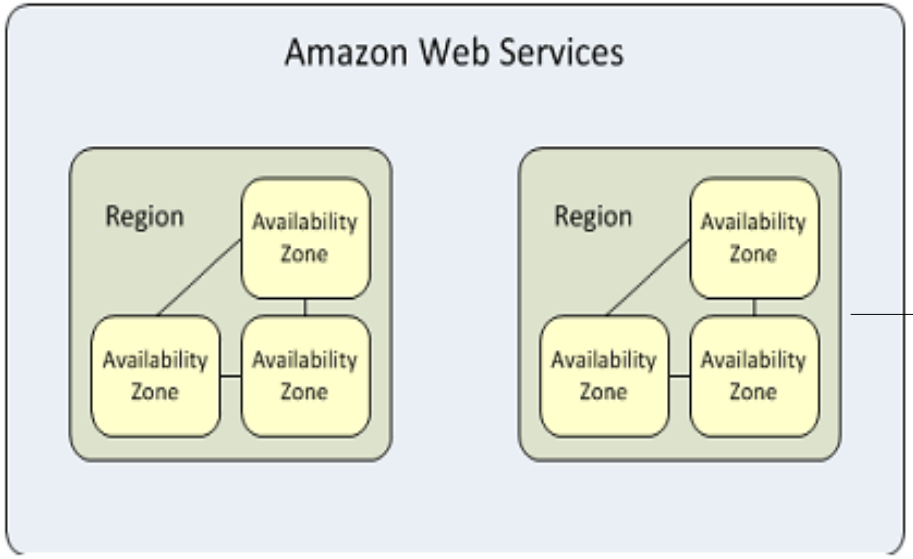
- Select a pre-configured, templated image to get up and running immediately. Or create an Amazon Machine Image (AMI) containing your applications, libraries, data, and associated configuration settings.
- Configure security and network access on your Amazon EC2 instance.
- Choose which instance type(s) and operating system you want, then
  - start,
  - terminate,
  - monitor

  as many instances of your AMI as needed, using the web service APIs or the variety of management tools provided.
- Determine whether you want to run in multiple locations, utilize static IP endpoints, or attach persistent block storage to your instances.
- Pay only for the resources that you actually consume, like instance-hours or data transfer.

Amazon EC2 (from the site)

- Amazon Elastic Block Store – Amazon Elastic Block Store (EBS) offers persistent storage for EC2 instances. EBS volumes persists independently from the life of an instance
- The SLA commitment is 99.95% availability (4 hours a year) for each of five Region.US East (Northern Virginia), US West (Northern California), EU (Ireland), Asia Pacific (Singapore), and Asia Pacific (Tokyo).
- Multiple Locations – Amazon EC2 provides the ability to place instances in multiple locations that are composed of Regions and Availability Zones.
  - Availability Zones = distinct locations engineered to
    - Be insulated from failures in other Zones
    - Provide low latency network connectivity to other Zones in the same Region.
    - By launching instances in separate Zones, you protect your applications from failure of a single location.
  - Regions
    - consist of one or more Availability Zones,
    - are geographically dispersed,
    - will be in separate geographic areas or countries.

Amazon EC2 (from the site)



Code	Name
ap-northeast-1	Asia Pacific (Tokyo) Region
ap-southeast-1	Asia Pacific (Singapore) Region
ap-southeast-2	Asia Pacific (Sydney) Region
eu-west-1	EU (Ireland) Region
sa-east-1	South America (Sao Paulo) Region
us-east-1	US East (Northern Virginia) Region
us-west-1	US West (Northern California) Region
us-west-2	US West (Oregon) Region

- [EC2-Classic] Select one of the Availability Zone options from the list, or select **No Preference** to enable us to select the best one for you.

Availability Zone (i) No preference

- [EC2-VPC] Select one of the subnet options from the list, or select **No preference (default subnet in any Availability Zone)** to enable us to select the best one for you.

Subnet (i) No preference (default subnet in any Availability Zone) Create new subnet

Amazon EC2 (from the site)

- Amazon CloudWatch
  - a web service that monitors AWS cloud resources, starting with Amazon EC2. It provides you with visibility into
    - resource utilization,
    - operational performance,
    - overall demand patterns—including metrics such as CPU utilization, disk reads and writes, and network traffic.
- Elastic IP Addresses
  - static IP addresses designed for dynamic cloud computing.
  - An Elastic IP address is associated with your account not a particular instance, and you control that address until you choose to explicitly release it.
  - Elastic IP addresses allow you to mask failures of instance or Availability Zone by programmatically remapping your public IP addresses to any instance in your account

Amazon EC2 (from the site)

- Auto Scaling –
  - allows you to automatically scale your Amazon EC2 capacity up or down according to conditions you define
  - ensures that the number of Amazon EC2 instances you're using scales up seamlessly during demand spikes to maintain performance, and scales down automatically during demand lulls to minimize costs.
- Elastic Load Balancing
  - It automatically
    - distributes application traffic across multiple Amazon EC2 instances
    - reroutes traffic to healthy instances until the unhealthy instances have been restored.
  - It enables you to achieve even greater fault tolerance in your applications
  - It detects unhealthy instances within a pool

Amazon EC2 (from the site)

Type	Description
Basic	Data is available automatically in 5-minute periods at no charge.
Detailed	Data is available in 1-minute periods at an additional cost. To get this level of data, you must specifically enable it for the instance. For the instances where you've enabled detailed monitoring, you can also get aggregated data across groups of similar instances. For information about pricing, see the Amazon CloudWatch product page.

CPU utilization, network, disk, swapin swapout
15 days of data available to discover trends and
to evaluate current computational load

Amazon EC2 (from the site)

- High Performance Computing (HPC) Clusters –

  Customers with complex computational workloads such as tightly coupled parallel processes, or with applications sensitive to network performance, can achieve the same high compute and network performance provided by custom-built infrastructure while benefiting from the elasticity, flexibility and cost advantages of Amazon EC2.

- VM Import

  It enables you to easily import virtual machine images from your existing environment to Amazon EC2 instances. Preserves your existing investments by seamlessly bringing those virtual machines into Amazon EC2 as ready-to-use instances.

Amazon EC2 (from the site)

- Placement Groups = A logical grouping of instances within a single Availability Zone (cannot span several zones).
- Placement groups enables applications to participate in a low-latency, 10 Gbps network and are recommended for applications that benefit from low network latency, high network throughput, or both.
- First, you create a placement group and then you launch the number of instances that you need in the placement group in a single launch request. If you try to add more instances to the placement group later, you increase your chances of getting an insufficient capacity error.

Carthography: Strategy

1. Map the cloud infrastructure to find where the target is located
2. Use various heuristics to determine co-residency of two VMs
3. Launch probe VMs trying to be co-resident with target VMs
4. Exploit cross-VM leakage to gather info about target

Attacker (threat) model

- Cloud infrastructure provider is trustworthy
- Cloud insiders are trustworthy
- Other clients are not trustworthy = Attacker is a malicious third party who legitimately accesses the cloud as a client

Assets

Confidentiality aware services running on the cloud

Availability of services running on the cloud

Tools of the trade

Network probing

a) **Nmap** = security scanning and fingerprinting to discover hosts and services on a computer network nodes to create a network "map“

b) **hping** = packet generator and analyzer for TCP/IP.

c) **Wget** = program that retrieves content from web servers.

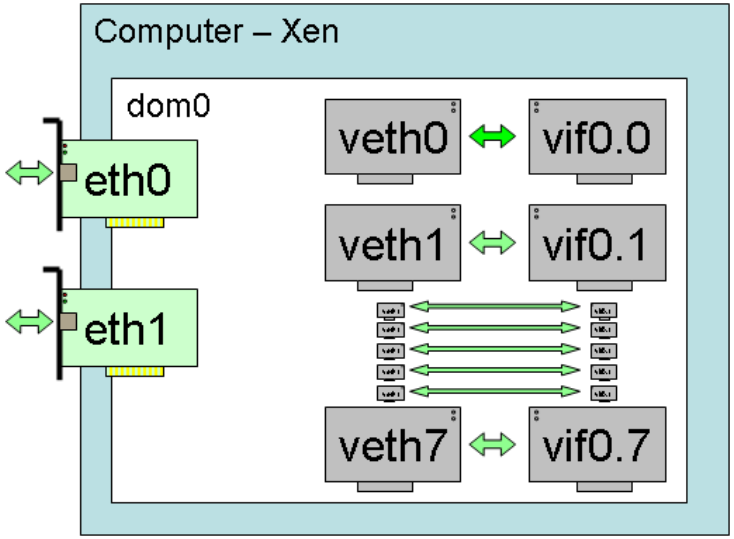
External probes = originated from outside EC2

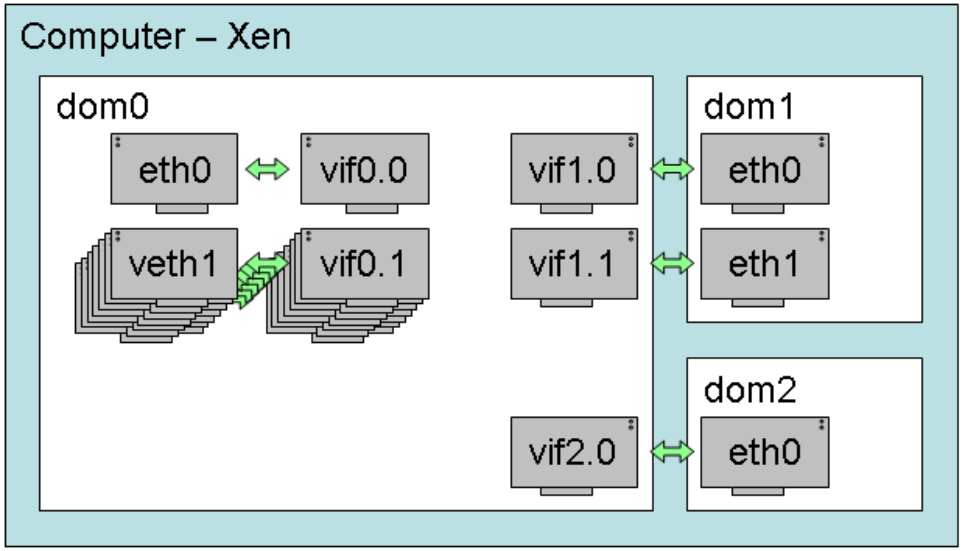
Internal probes = originated from EC2

Amazon EC2’s own DNS to map dns names to IPs

Sidenote: EC2 configuration

EC2 uses **Xen**, with up to 8 instances per physical machine



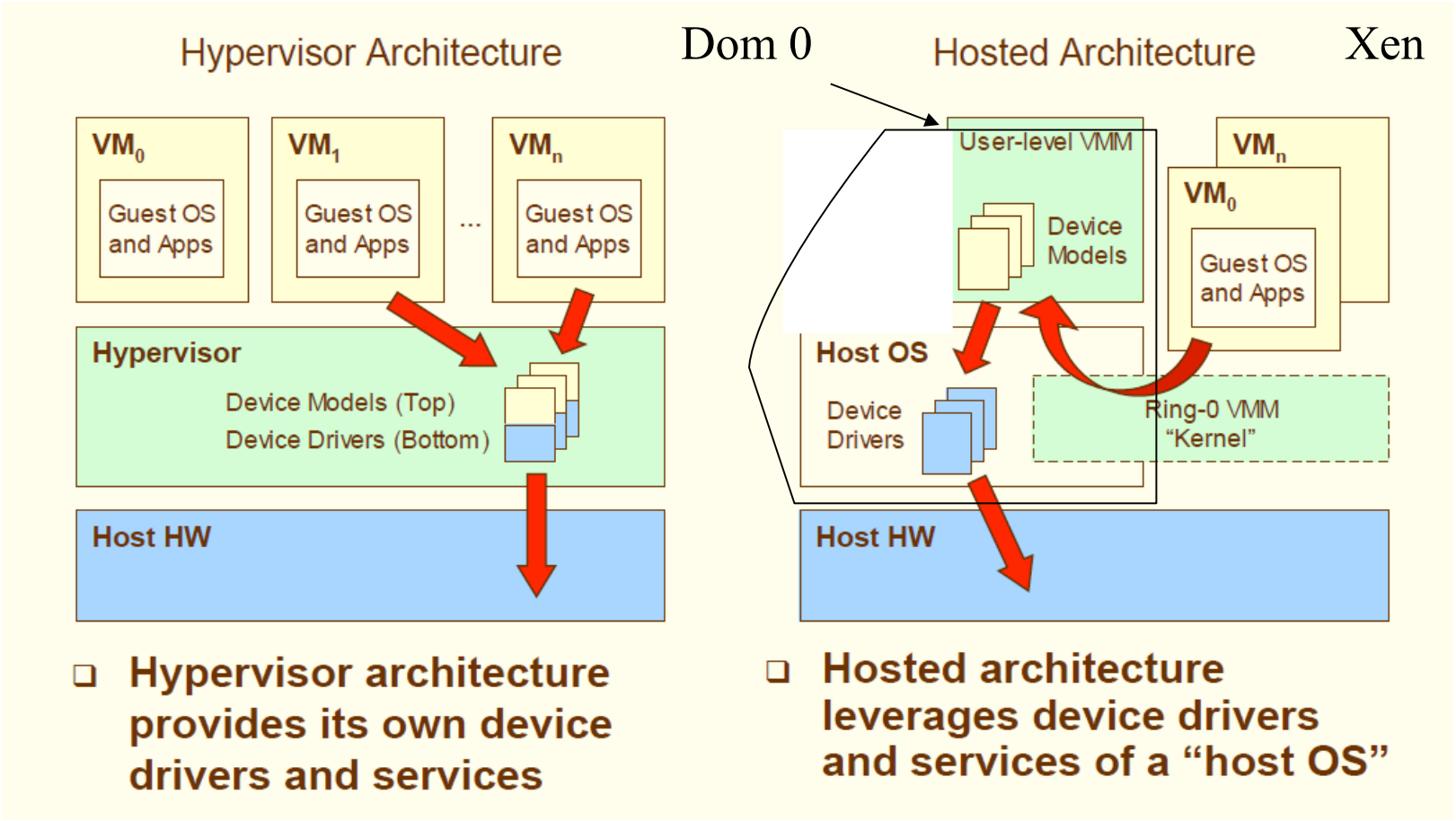


Dom0 is the first instance on the machine, connected to physical adapter

All other instances route to external world via dom0

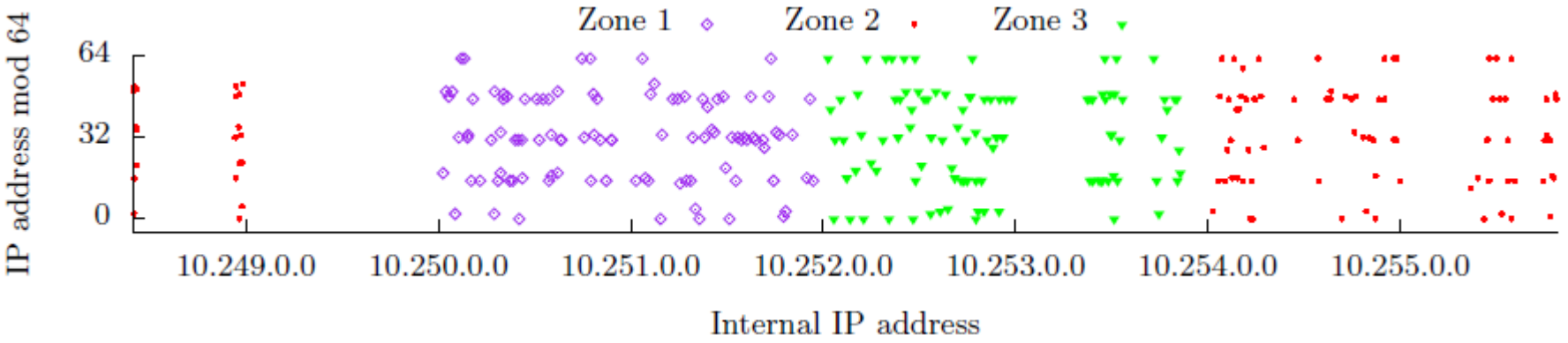
[Figures from Xen Wiki]

Xen I/O and Routing



- Hypervisor architecture provides its own device drivers and services
- Hosted architecture leverages device drivers and services of a "host OS"

Task 1: Mapping the cloud



Different availability zones in the same region use different IP regions.

Each instance has one internal IP and one external IP. Both are static.

For example:

External IP: *75.101.210.100*

External Name: *ec2-75-101-210-100.computer-1.amazonaws.com*

Internal IP: *10.252.146.52*

Internal Name: *domU-12-31-38-00-8D-C6.computer-1.internal*

Reverse engineering the VM placement schemes provides useful heuristics about EC2's strategy

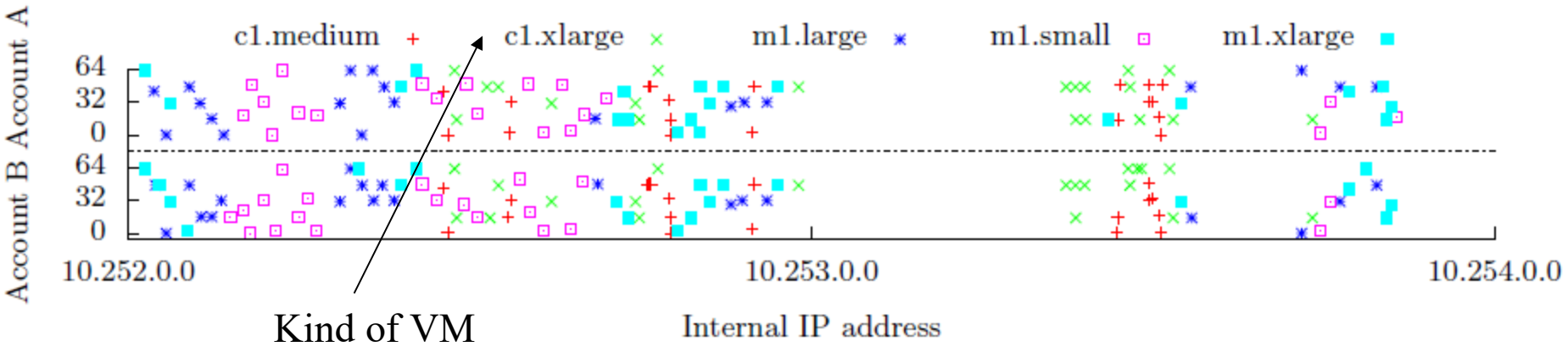
Task 1: Mapping the cloud

1. We identified four distinct IP address prefixes, a /16, /17, /18, and /19, as being associated with EC2. The last three contained public IPs observed as assigned to EC2 instances.

2. Nmap has been used to discover IP pubblic addresses by a scanning on the port 80 or 443

3. Via an appropriate DNS lookup from within EC2, we translated each public IP address that responded to either the port 80 or port 443 scan into an internal EC2 address.

Task 1: Mapping the Cloud

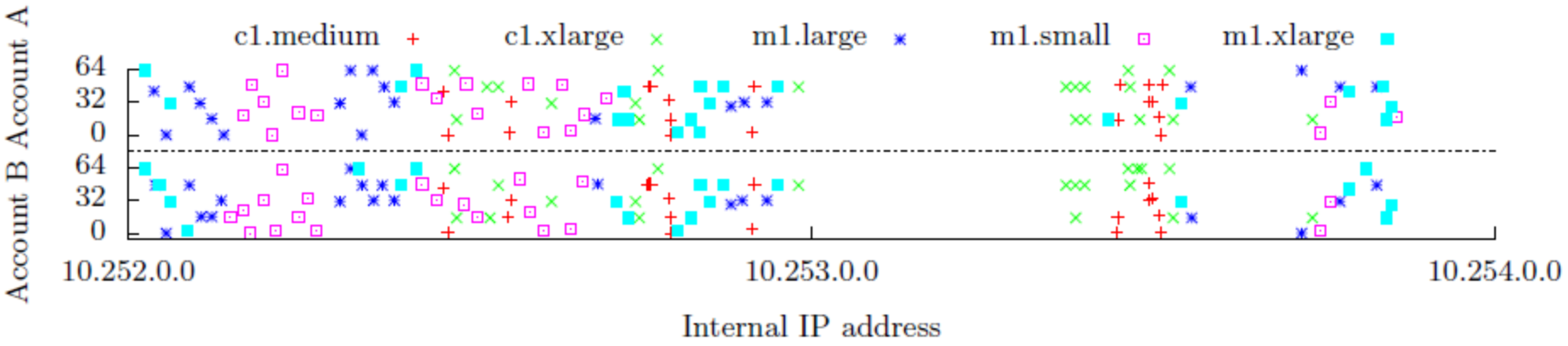


- We used a single account, call it "Account A". The top graph depicts a plot of the internal IP address assigned to each of the 300 instances, partitioned according to availability zone.

  Samples from each zone are assigned IP addresses from disjoint portions of the observed internal address space.

- We launched 100 instances (20 of each type, 39 hours after terminating the Account A instances) in Zone 3 from a second account, "Account B".

Task 1: Mapping the Cloud



Finding: same instance type within the same zone = similar IP regions

Reverse engineered mapping decision heuristics:

A /24 inherits any included sampled instance type.

A /24 containing a Dom0 IP address only contains Dom0 IP address.

All /24's between two consecutive Dom0 /24's inherit the former's associated type.

Task #2: Determining co-residence

Co-residence: Check to determine if a given VM is placed in the same physical machine as another VM

Network based check:

Instances are likely co-resident if they have

(1) matching Dom0 IP address,

(2) small packet round-trip times,

or

(2) numerically close internal IP addresses (e.g. within 7).

An instance owner can determine its Dom0 IP from the first hop on any route out from the instance.

One can determine an uncontrolled instance's Dom0 IP by performing a TCP SYN traceroute to it (on some open port) from another instance and inspecting the last hopNo false positives found during experiments

Task #2: Determining co-residence

Round Trip Times of Coresident VMs

Number of pairs

	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

For any pair A,B at first A probes B (control A) and then B probe A (control B)

Brute force scheme

Idea: figure out target's availability zone and type

Launch many probe instances in the same area

Success rate: 8.4%

Smarter strategy: utilize locality

Idea: VM instances launched right after target are likely to be co-resident with the target

Paper claims 40% success rate due to strong locality in the allocation of instances

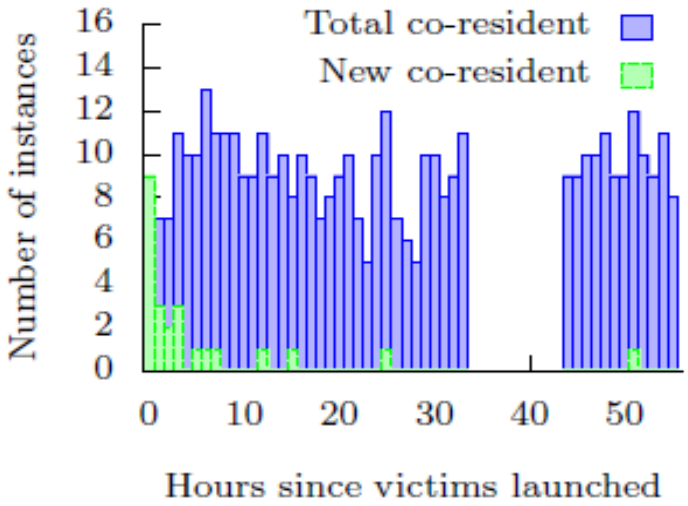
Task #3: Making a probe VM co-resident with target VM

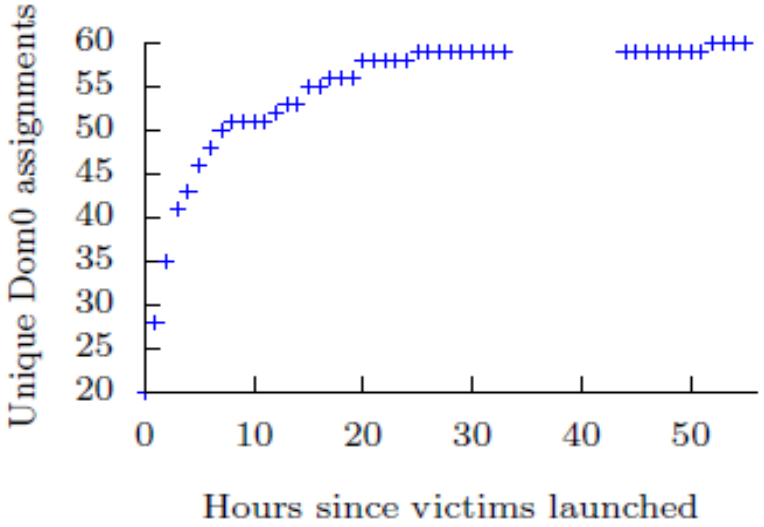
	# victims v	# probes p	coverage
Zone 1	1	20	1/1
	10	20	5/10
	20	20	7/20
Zone 2	1	20	0/1
	10	18	3/10
	20	19	8/20
Zone 3	1	20	1/1
	10	20	2/10
	20	20	8/20

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.

Task #3: Making a probe VM co-resident with target VM



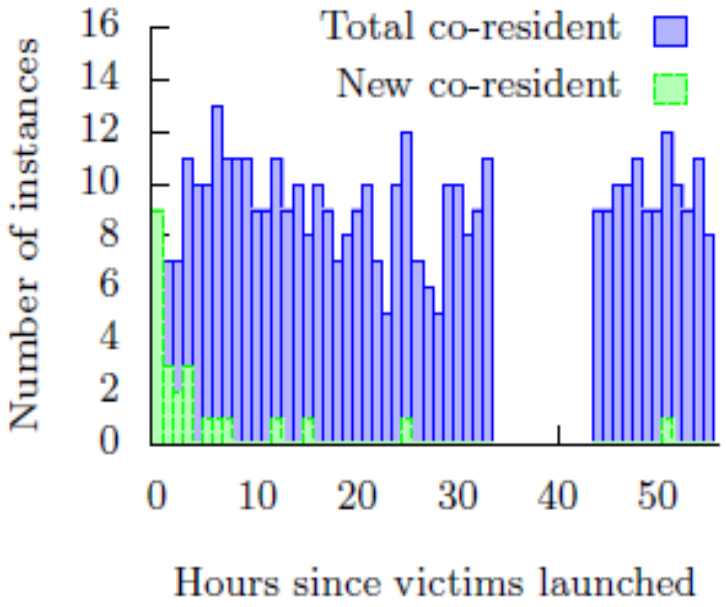


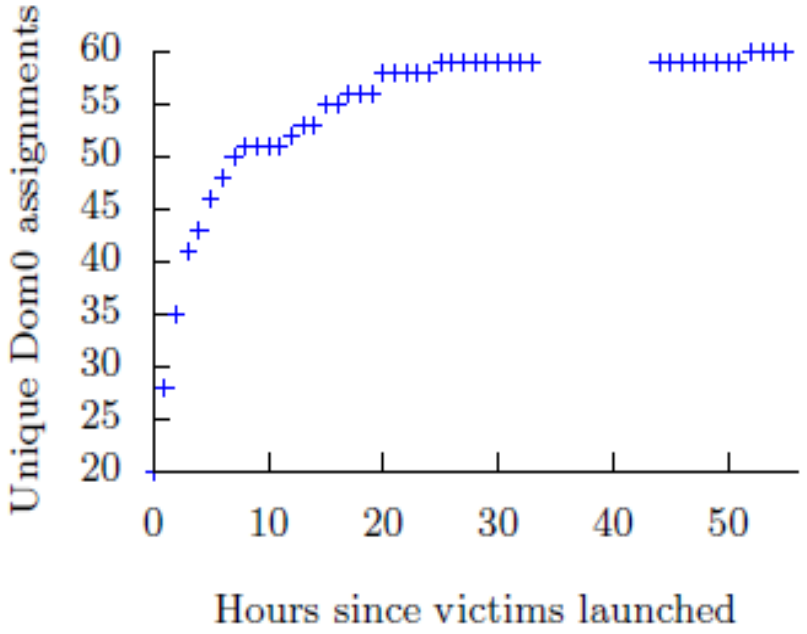
Results for the experiment measuring the effects of increasing time lag between victim launch and probe launch. Probe instances were not run for the hours 34–43.

"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.

Task #3: Making a probe VM co-resident with target VM

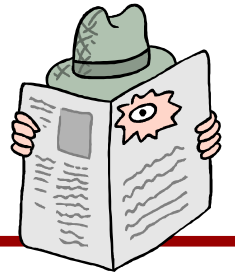




Window of opportunity is quite large, measured in days

The right graph shows the cumulative number of unique Dom0 IP addresses seen by the probes over the course of the experiment. The growth in the number of machines probes were placed on levels off rapidly—quantitative evidence of sequential placement locality.

Task #4: Gather leaked information



Now that the VM is co-resident with target, what can it do?

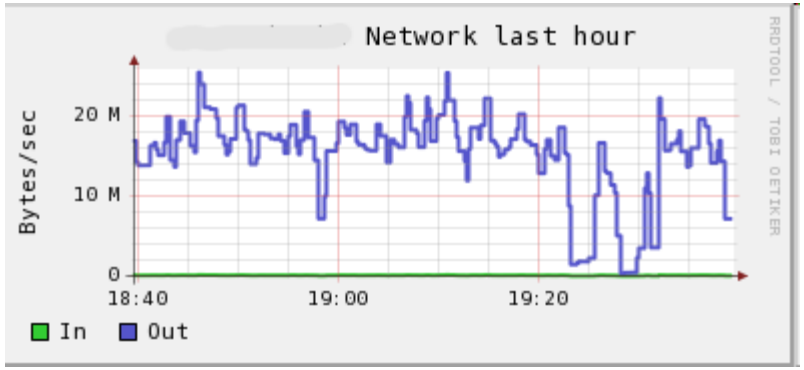
1. Gather information via side channels
   - Cooperating vm
   - Unwilling vm
2. Perform DoS

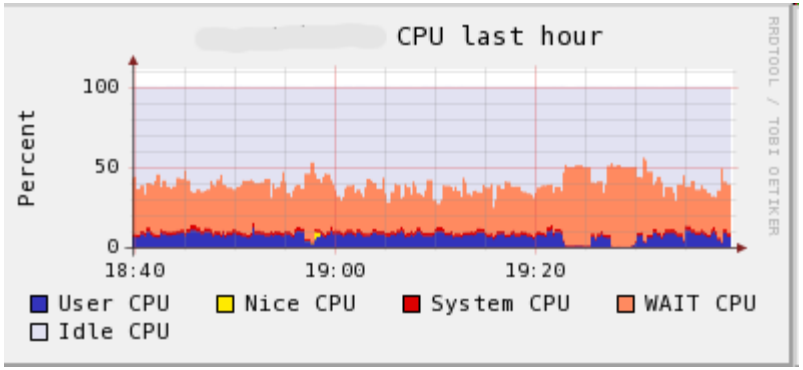
DOS

Two VMs

- One creating a backup
- The other execute the command

  dd if=/dev/zero of=testfile bs=1M count=15000 eg creates a 15G file with zeroes





Average network bandwidth and CPU for the one creating the backup reduced from 15/20 M to 1-2

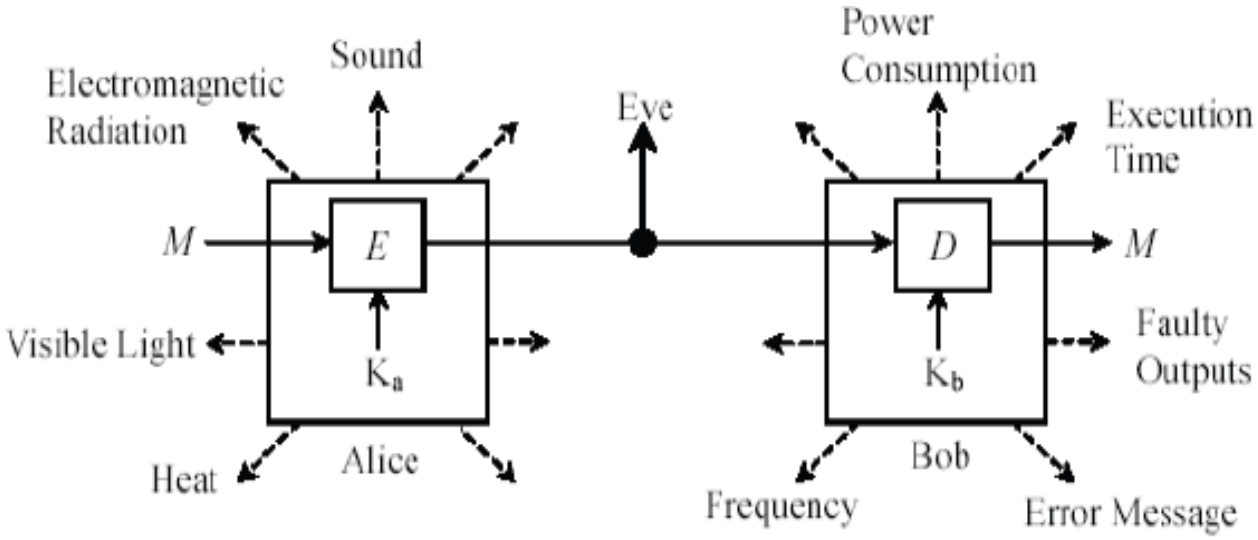
Side Channel Attack (wikipedia ;-))

a **side channel attack** is any attack based on information gained from the physical implementation of a cryptosystem rather than from brute force or algorithm weaknesses as in cryptoanalysis

Classified into

- Timing : measuring time to perform various computations
- Power : make use of varying power consumption by the hardware during computation.
- Electromagnetic: based on leaked electromagnetic radiation which can directly provide plaintexts. Cryptographic keys can be deduced from measurements through techniques equivalent to those in power analysis, or from non-cryptographic attacks, e.g. radiation monitoring
- Acoustic : exploit sound produced during a computation rather like power
- Cache : exploiting state shared between processes on a computer
- Differential Fault : secrets are discovered by introducing faults in a computation.

Side Channel Attack

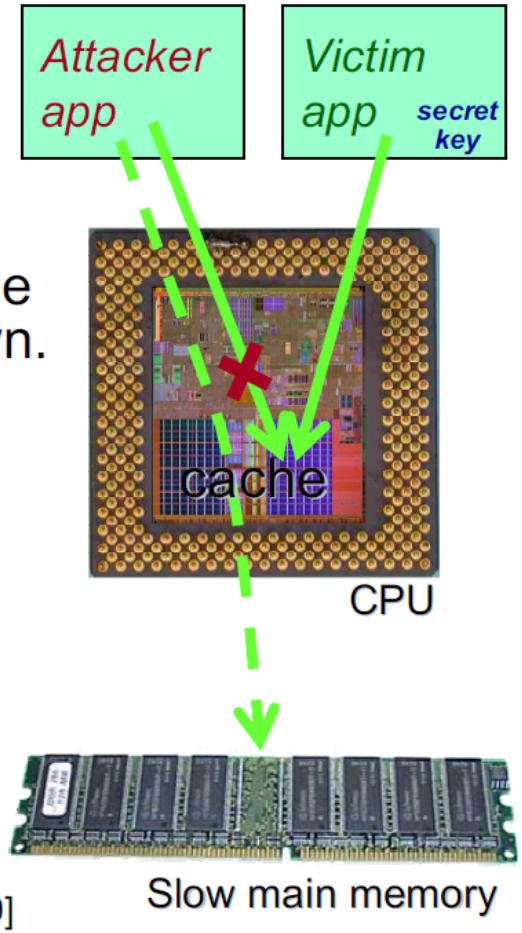


Side Channel Attack

- CPU contains small, fast memory **cache** shared by all applications.
- If only *Attacker* accesses memory, it is served from the fast cache.
- If *Victim* also accesses memory, the cache fills up… and *Attacker* notices a slow-down.
- From this, *Attacker* can deduce the memory access patterns of *Victim*.
- This is sensitive information!

  Example: if *Victim* is performing RSA or AES decryption, the access patterns are determined by the *secret key*.

→ unprivileged *Attacker* can steal AES secret in 65 milliseconds (non-cloud settings) [Tromer Osvik Shamir 09]



Task 4.1: Gathering information

If VM's are separated and secure, the **best** the attacker can do is to gather information

- Measure latency of cache loads
- Use that to determine
  - Co-residence
  - Traffic rates
  - Keystroke timing

Leaking Information

- A VM may be willing to transmit information to another one
- An insider willing to transmit some information to a competitor
- The competitor may run a VM on the cloud to receive the information through a covert channel
- A covert channel (not to be confused with a side channel) transmits information between two colluding partners through pattern of usage on a shared resource
- The receiver monitors the usage of the resource to input the information
- “if I am printing at 10 then 1 else 0”

Finding a Good Covert Channel

- Find a place that random data is being transmitted naturally

  Ex. Initial Sequence Numbers, complex timing of network transmissions
- Replace that random data with your own ‘random data’ which is actually an encrypted message

Example

- Alice wants to send a message to Bob
- Alice FTPs Bob a couple of old vacation pictures, meanwhile Bob records all traffic
- Alice encodes the secret message byte by byte in the padding of several TCP segment headers
- Bob looks at padding of recorded traffic

Unseen Path of Communication

Useful if you do not want your association with a node to be known

- Communicating with a closely scrutinized node
- Accessing ‘forbidden’ material
- Malicious activity

Use another node to relay information for you

- Alice and Bob both are allowed to make requests to the same small webserver
- To transmit a one, Alice pounds the webserver with heavy traffic
- To transmit a zero, Alice doesn’t make any requests
- Bob makes requests to the webserver and measures latencies

Finding a Good Channel

Use a hard to monitor node as an intermediary

Protection is possible even if all networks are watched

- Mixes, onion routing
  - Prevents association of incoming and outgoing
- Also use extra communication covert channel
  - Prevents detection that a node is relaying info

Evaluating a Covert Channel

- Bandwidth
  - bits per TCP connection
  - bits per packet
- Ease of Detection
- Permissibility
  - How often will it be permitted?
- Prevention
- Difficulty of Implementing
- Special Cases or Restrictions

Shared resource CPU

K. Okamura, Keisuke , Y. Oyama, Yoshihiro, Load-based covert channels between Xen VMs, 2010 ACM Symp. on Applied Computing

- A bit is transmitted by executing/not executing in a given time interval

S:R	3:1	2.5:1	2:1	1.5:1	1:1	1:1.5	1:2	1:2.5	1:3
Success	92	98	100	98	100	61	54	16	4
Sync failure	8	2	0	2	0	39	46	84	96
Bit error	0	0	0	0	0	0	0	0	0

- Only the two VMs are executed, S:R = ratio between the priorities of the two VMs
- Further load = noise

CPU load (%)	5-9	8-12	13-17
Success	92	91	74
Sync failure	7	9	13
Bit error	1	0	13

Mitigation strategies

Mapping

- Use a randomized scheme to allocate IP addresses
- Block some tools (nmap, traceroute)

Co-Residence Check

- Prevent traceroute (i.e. prevent identification of dom0)

Prevent Co-Location

- Not allow co-residence at all
- Beneficial for cloud user
- Not efficient for cloud provider

Is carthography required?

- Any time an user access an application through the web, some information leaking is possible even if the interaction occurs through an encrypted connection
- This leaking is critical when the user privacy is involved
- However, since the goverments and the public administration are moving to the cloud, the case of services such as
  - tax declaration
  - managing of health records

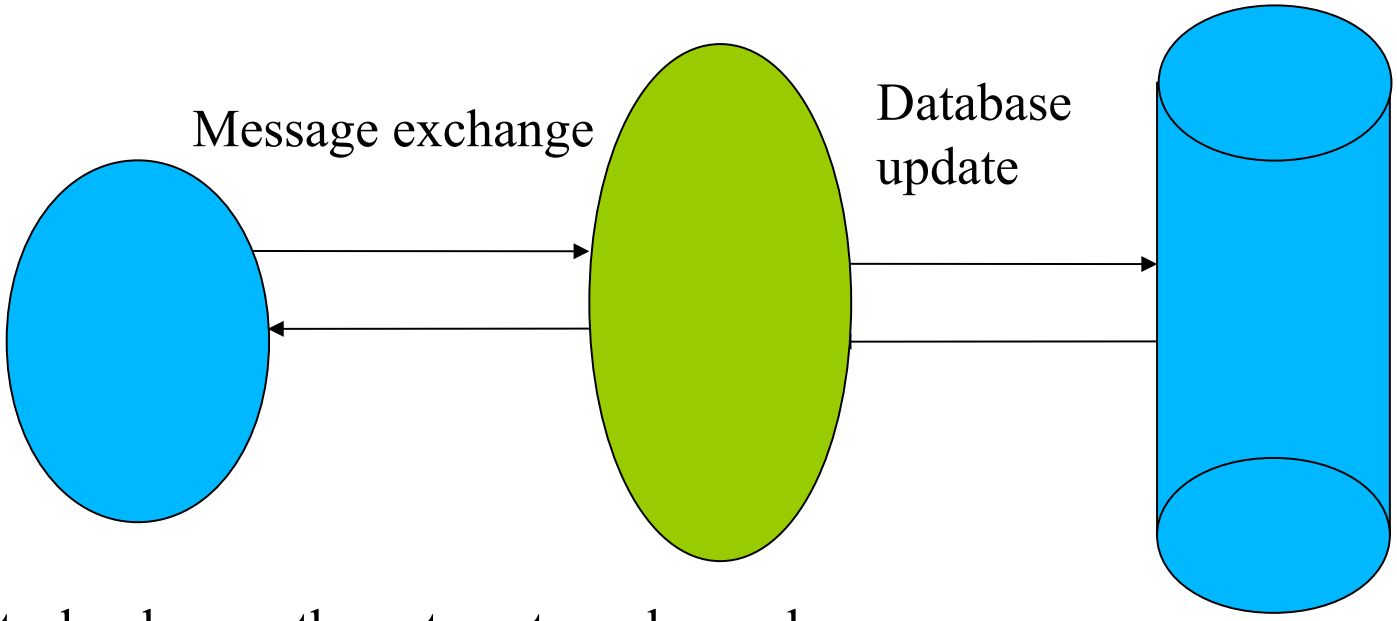
  is more and more popular
- In this case, the program that is offering the service is almost known since it is shared among a large number of users
- We consider attack against Software-as-a-Service

An abstract model

A user

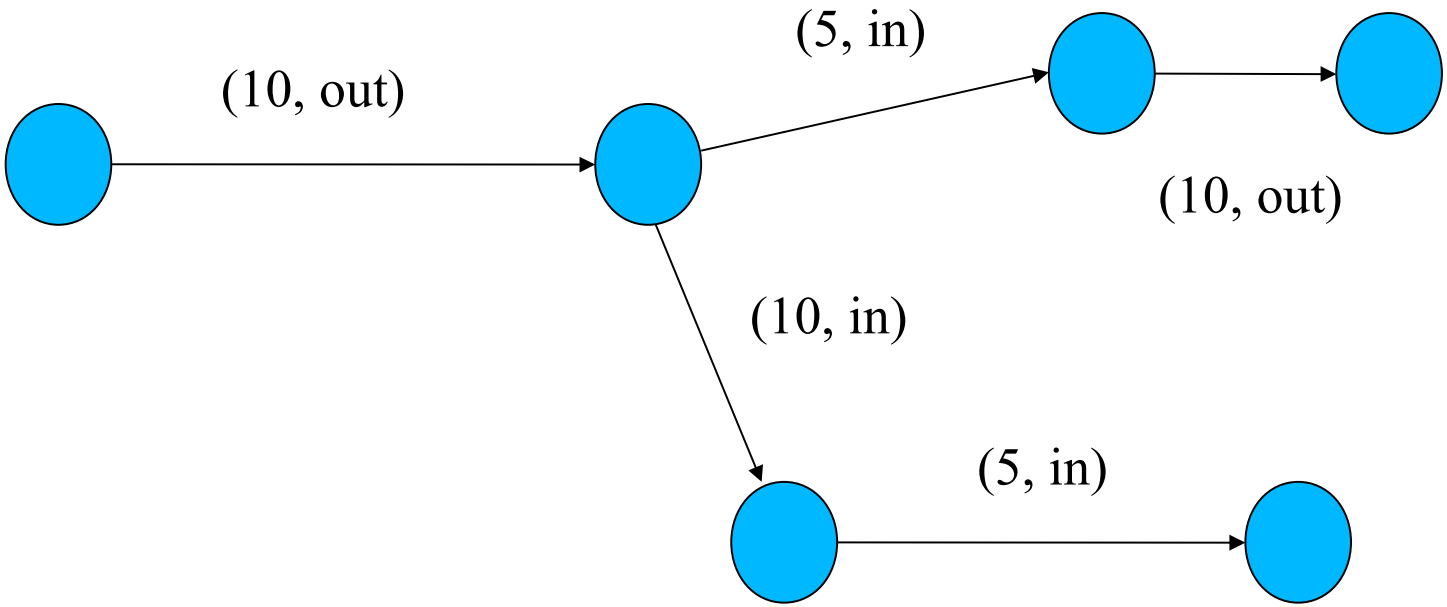
The program (= a finite state automata) interacting with the user

A database



The attacker knows the automata and can observe some properties of the messages that are exchanged and that belong to a finite domain (size, content, IP addresses, MAC addresses)

An abstract model



Each state transition is paired with the transmission (out) or the reception (in) of a message with a given property that may be observable or not observable
The automa may be deterministic or nondeterministic

Abstract Model

- The problem that is considered here is whether we can deduce some non observable property by analyzing the observable ones
- As an example, can we deduce some information on the value of an encrypted message by observing the size of the message?
- In general this is impossible but now we can exploit further information $\leftrightarrow$ the knowledge of the automata that describe the program behaviour
- The data is encrypted but the program that generates and manipulates this data is known

An important property - 1

- P is an observable property and *v an observable value*,
- The automata exchanges k messages where the value of the property of interest is **v**
- The messages transmitted by the automata may assume M distinct values of the property in a domain D according to the program point where it is transmitted

- The density of messages with a property value **v**
  - is the ratio k/M
  - it may larger than 1 because we count the occurences of each value of the property,
- Example
  - The observable property is the message size
  - The automata may send several times a 100 bytes message

An important property - 2

As density of an observable value decreases, it increase the probability of discovering

- the automata transaction that generates the message with such a property value
- other information about the value sent by the automata

In tax return, there is one path if the user is married and another if it is not married. If messages have a low density it is easy to discover this information

This is a case of ambiguity reduction enabled by the knowledge of the program

Entropy (1)

Definition of Entropy:

- Measure of the *uncertainty* of a random variable.
- Measure of the amount of *information* required on the average to describe the random variable
- Notation: *H(X)*

Entropy (2)

Given a discrete random variable, X, that can take N possible values with probability greater than zero, ($p_1 \ldots p_N$), the entropy of X is defined as:

$$H(X) = -\sum_{i=1}^{N} p_i \cdot \log_2(p_i)$$

Entropy (3)

- The more equally distributed, the more information we gain from an event (greater *H(X)*); the closer to a deterministic distribution, the less information (smaller *H(X)*)
- The closer to a deterministic distribution, the less information (smaller *H(X)*)
- The entropy of X is a functional of the distribution of X, it does not depend on the values X takes (X: set of possible senders; p_i: probability that $X = x_i$)

SIDE-CHANNEL-LEAKS IN WEB APPLICATIONS: A REALITY TODAY, A CHALLENGE TOMORROW

Shuo Chen

Microsoft Research

Rui Wang, XiaoFeng Wang and Kehuan Zhang

INDIANA UNIVERSITY

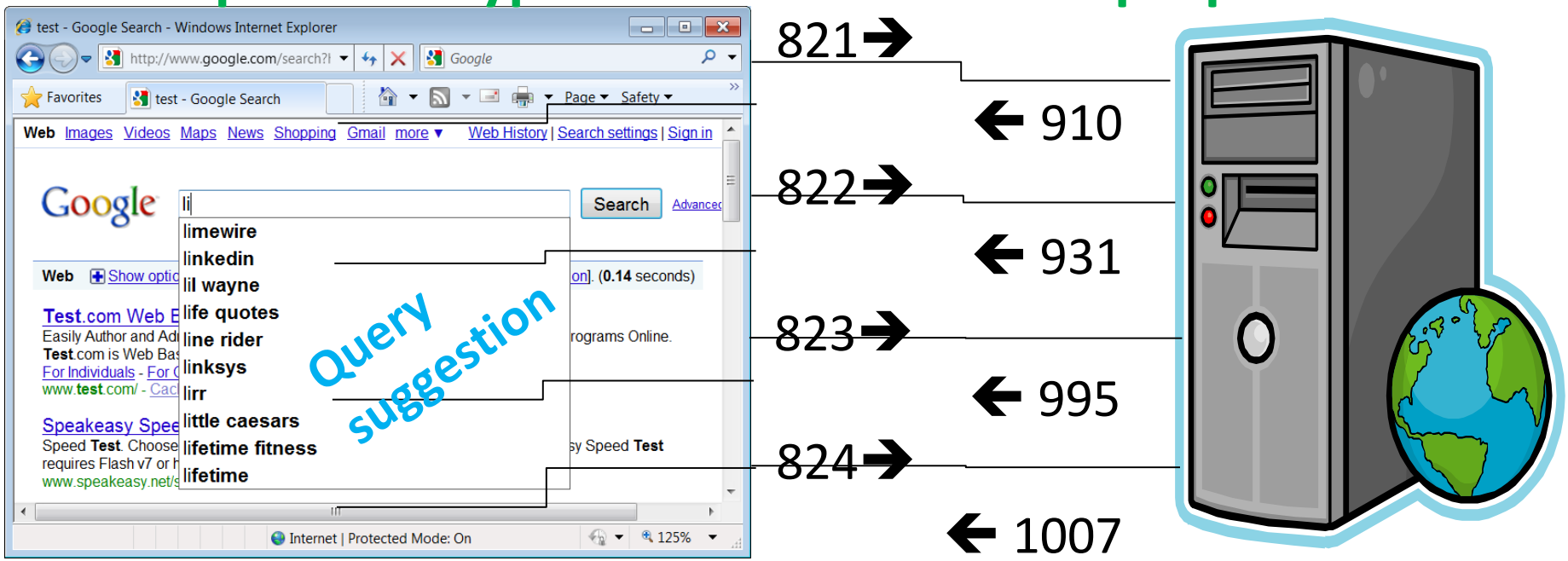
IEEE Symposium on Security and Privacy
Oakland, California
May 17th, 2010

Our Main Findings

- Detailed user information is being leaked out from several high-profile web applications
  - personal health data, family income, investment details, search queries
- Anonymized app names per requests from related companies. The root causes are some fundamental characteristics in today's web apps
  - stateful communication, low entropy input and significant traffic distinctions.
- Defense is non-trivial
  - effective defense needs to be application specific.
  - calls for a disciplined web programming methodology.

Scenario: search using encrypted Wi-Fi WPA/WPA2.

Example: user types "*list*" on a WPA2 laptop.



Attacker's effort: linear, not exponential.

Consequence: Anybody on the street knows our search queries.

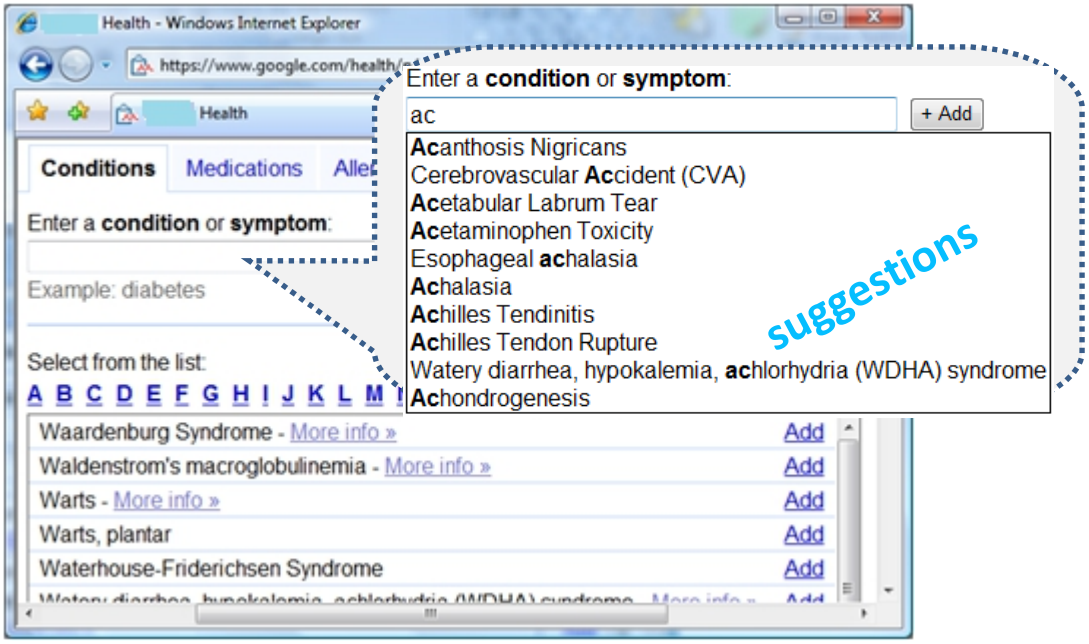
OnlineHealth[A]

("A" denoting a pseudonym)

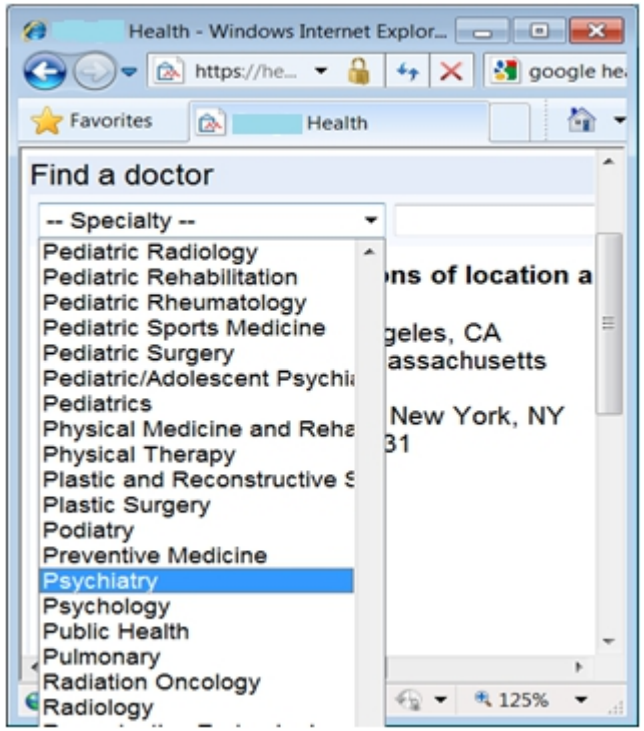
- A web application by one of the most reputable companies of online services
- Illness/medication/surgery information is leaked out, as well as the type of doctor being queried.
- Vulnerable designs
- Entering health records
  - By typing – auto suggestion
  - By mouse selecting – a tree-structure organization of elements
- Finding a doctor
  - Using a dropdown list item as the search input

Attacker's power

Entering health records: no matter keyboard typing or mouse selection, attacker has a 2000× ambiguity reduction power.



Find-A-Doctor: attacker can uniquely identify the specialty.



Ed Felten comment

- …. this inference-from-size problem gets much, much worse when pages are using the now-standard AJAX programming methods, in which a web "page" is really a computer program that makes frequent requests to the server for information. With more requests to the server, there are many more opportunities for an eavesdropper to make inferences about what you're doing -- to the point that common applications leak a great deal of private information

- In other words, with Ajax some complexities become linear rather than polinomial.

- It's becoming increasingly clear that securing web-based applications is very difficult, and that the basic tools for developing web apps don't do much to help. The industry, and researchers, will be struggling with web app security issues for years to come.

The Register

The threat is significant because it stems from fundamental characteristics of software-as-a-service applications that have been in vogue for about a decade. Among other things, apps built on AJAX and other Web 2.0 technologies are usually "stateful," meaning they keep track of unique configuration information. Such data often has "low entropy," making it easy for attackers to make educated guesses about its contents

Software-as-a-Service = cloud computing

Open Source + Cloud+Carthography = ????

Side Leak Attack

Countermeasure

- **Prevent Traffic Analysis**
- **Sidebuster: Automated Detection and Quantification of Side Channel Leaks in Web Application Development**

Side Leak Attack

Countermeasure

- **Prevent Traffic Analysis**
- **Sidebuster: Automated Detection and Quantification of Side Channel Leaks in Web Application Development**

SideBuster

- Automatic detection of side-channel leaks in web applications
- Novel technique for quantifying information leaks
- Implementation and evaluation

Traffic Analysis Prevention

What is Traffic Analysis Prevention (TAP)?

Problems, challenges with TAP

Analysis attacks, typical system vulnerabilities

Methods of TAP

Conclusion

Introduction to TAP

Traffic Analysis Prevention –

- *Traffic Analysis* - an *adversary* who monitors or compromises parts of a system, in order to match message senders with recipients
- *Network Unobservability* - Masking comm. patterns which emerge over time

Achieving network unobservability implies ineffective traffic analysis

Challenges for TAP

Loss of efficiency/performance from adding TAP mechanisms to a system

Number of possible senders/recipients bounded

Analysis Attacks, cont…

Brute-force

- Following every possible path a message might have taken
- Typically results in a list of *possible* sender/recipient pairs

Node flushing

- A mix node** which sends out messages after receiving *N* messages;
  - Attacker sends *N-1* messages to a node, can then match his inputs with messages leaving the node
- Encryption, authentication at each mix node combats node flushing
- Can be mounted by *active* adversaries (able to manipulate network, as opposed to simply listen)

Timing attack

- If routes taken by messages have different latencies, attacker may use arrival and departure times from a network to correlate the path taken

Analysis Attacks

Contextual attack

- Looking at distinguishing features of traffic patterns, such as partners taking turns communicating, counting numbers of packets in arriving and departing messages, etc

DoS attack

- Destroying some intermediate nodes will affect the behavior of some users but not others, revealing information about which users take which paths

Exploitation of user reactions

- Intercept a message leaving a mix-node, and forward to many recipients
- Users expecting to receive the message will behave differently than those not expecting the message

Many others

Methods of TAP

- Mixing
- Dummy messages
- Routing

Methods of TAP: Mixing

A packet travels from source to destination through several intermediate nodes (*mix nodes*) (onion routing)

User encrypts the message to be sent

Each *mix* node collects packets/messages, releases in some different (random) fashion

The longer a node can hold and collect messages, the better the security

Mixing helps to hide sender-recipient relationship by masking the route taken (any one node only knows the previous and next hop, not entire route)

Methods of TAP: Dummy Msg.

Injecting dummy traffic/packets into the network (a.k.a. “padding”)

- Dummy packets should not be distinguishable from “real” packets
- Padding within the packet

Tradeoff:

- Dummy messages may reduce amount of time real packets are saved up in mixed nodes, if dummy messages are used in conjunction with a mixed network
- Dummy messages obviously decrease overall network efficiency, increase network overhead

Methods of TAP: Routing

Altering routes that packets travel to make traffic following difficult

- Varying number of hops

Onion Routing

- Each intermediate node only knows about previous and next node
- As mentioned in mixing, encryption/decryption at each node alters message's appearance

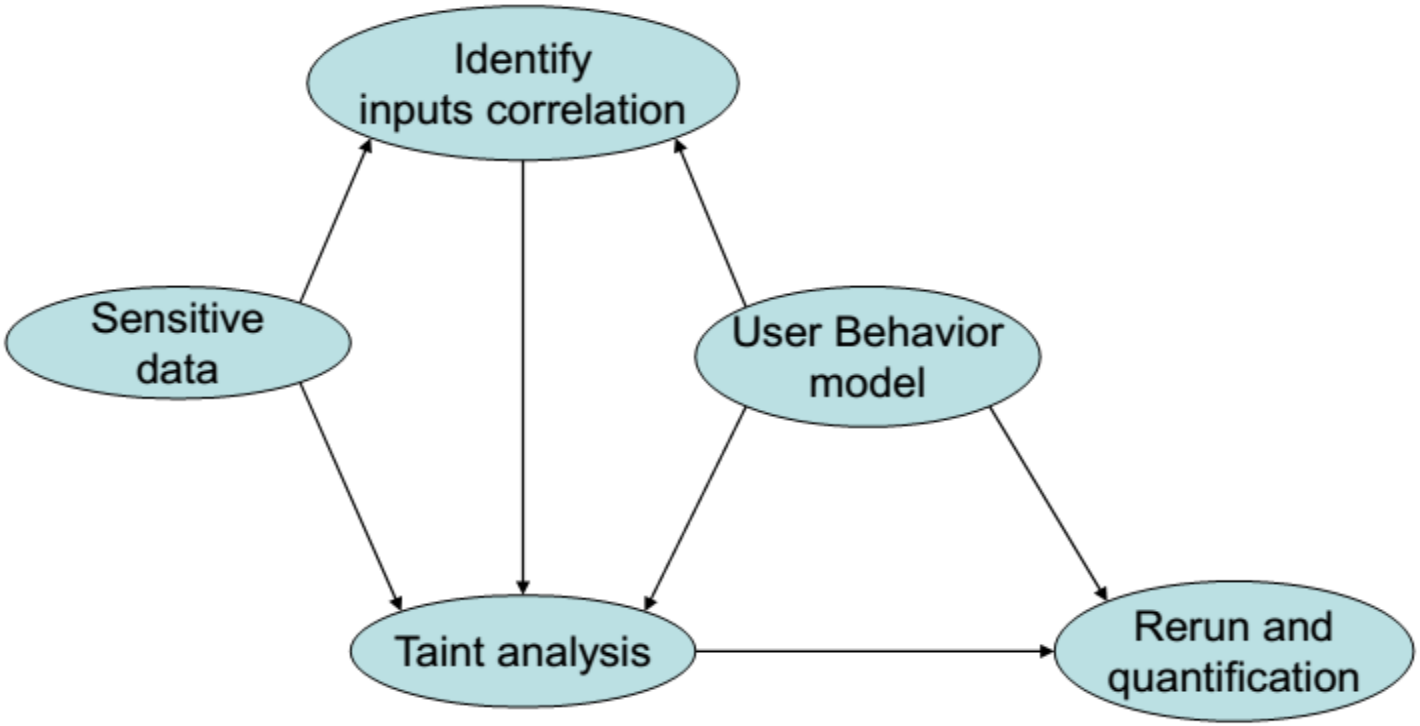
Conclusions

Traffic analysis attacks can be mounted from a variety of adversaries

Current methods of TAP *reduce* the ability of adversaries, but fool-proof prevention methods are undiscovered or impractical

TAP methods typically provide security at a relatively high cost of added overhead

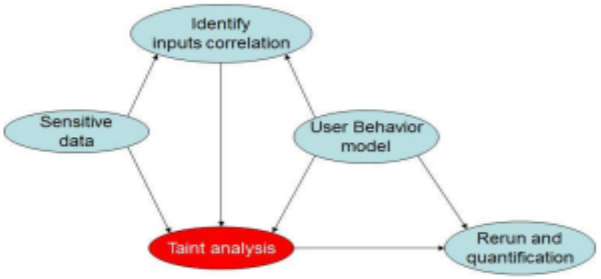
SideBuster


Identify
inputs correlation
Sensitive
data
User Behavior
model
Taint analysis
Rerun and
quantification

SideBuster

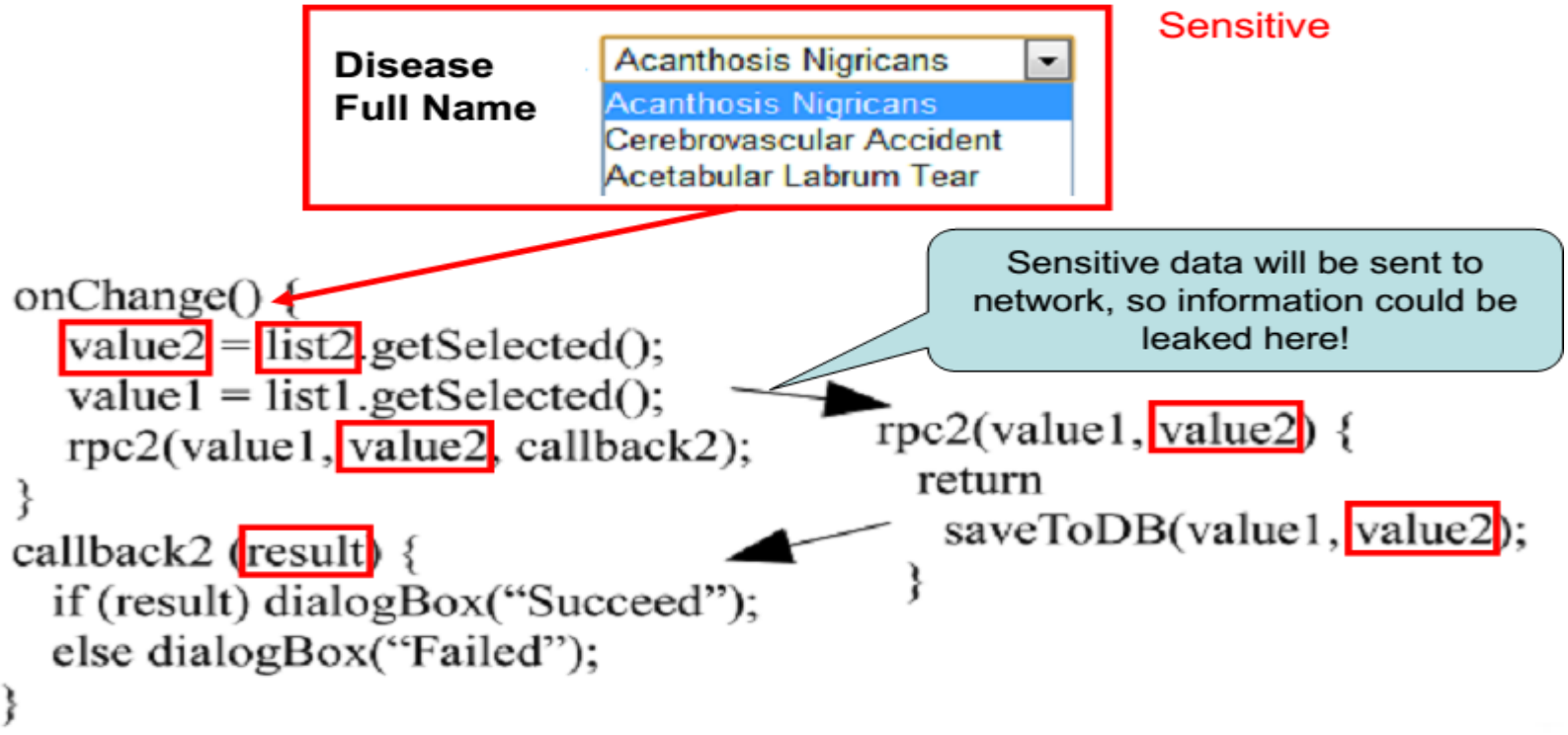
Data Flow Analysis

- Using existing “taint analysis” techniques
  - Mark the sensitive user input as taint source
  - Mark the network communication APIs as taint sink
  - Define a set of taint propagation rules

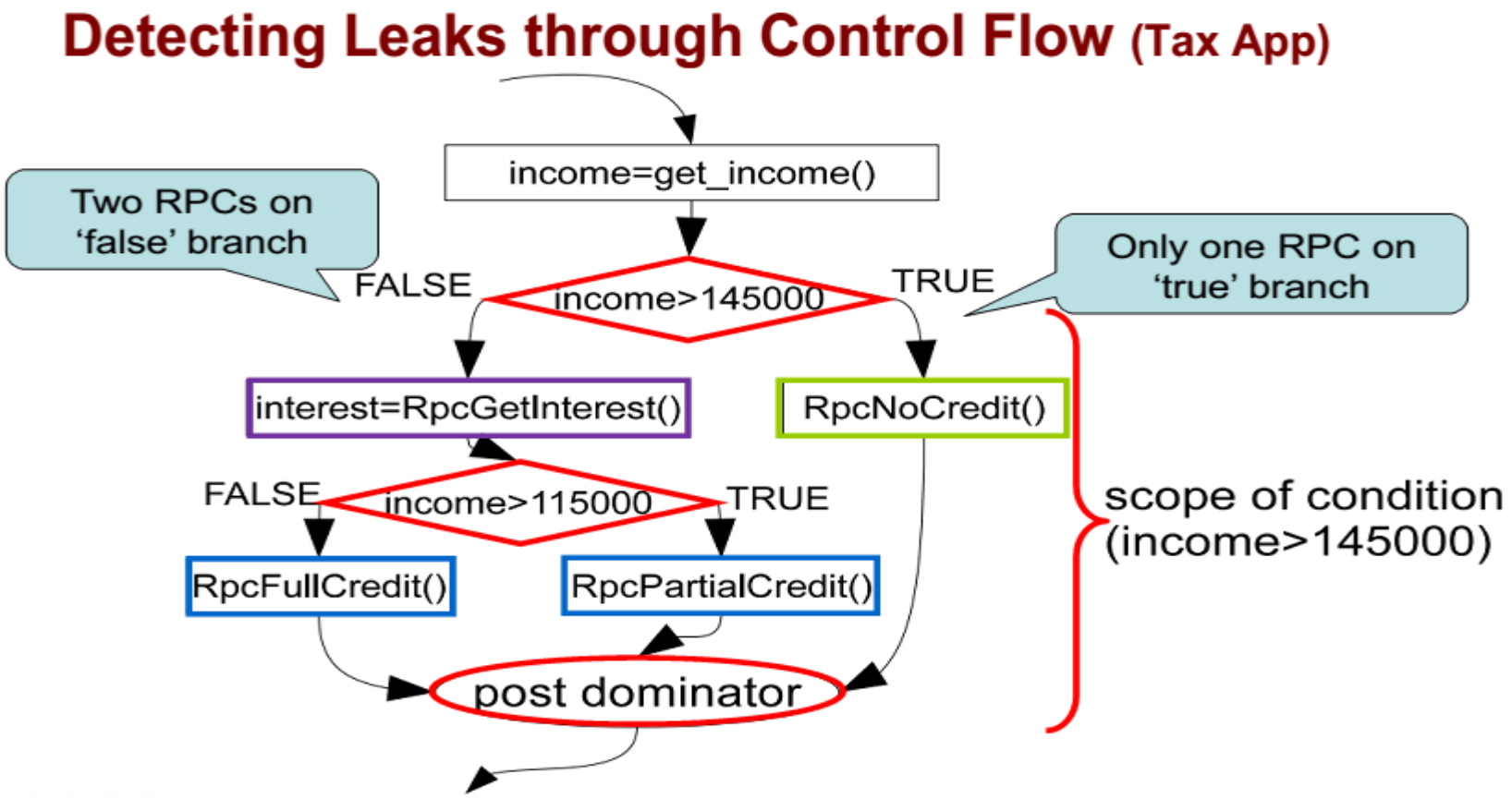


SideBuster

Trace the propagation of sensitive data (Health App)



SideBuster

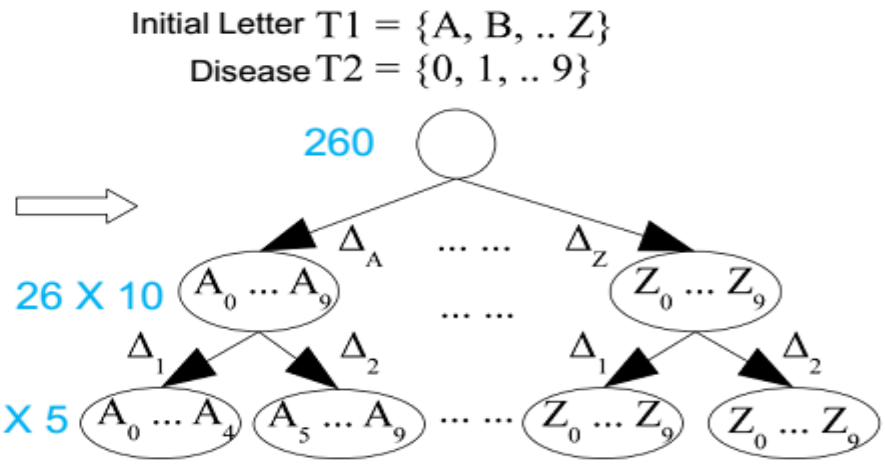

Detecting Leaks through Control Flow (Tax App)
income=get_income()
Two RPCs on 'false' branch
FALSE
income>145000
TRUE
Only one RPC on 'true' branch
interest=RpcGetInterest()
RpcNoCredit()
FALSE
income>115000
TRUE
scope of condition (income>145000)
RpcFullCredit()
RpcPartialCredit()
post dominator

SideBuster

Examples: leaks from multiple source

Input T1	A	B	C	…	Y	Z
Output 1	Δ_A	Δ_B	Δ_C	…	Δ_Y	Δ_Z

Input T2	0	…	4	5	…	9
Output 2	Δ_1	…	Δ_1	Δ_2	…	Δ_2



- Input T2 depends on input T1
- The original information is: $\log_2(|T1|x|T2|) = \log_2(260) = 8.02$ bits
- The information remained is: $52x(1/52)x\log_2 5 = 2.32$ bits
- 5.7 bits of information has been leaked

All together now

- WIth cartography you can allocate a VM that measures some properties on the node running two VMs that exchange some interesting informatio
- With open source, SaaS ect. discovering the automata that corresponds to a given application is not too complex
- *It is even simpler if you are the provider ...*