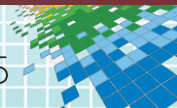


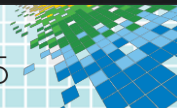
# CLOUD IS THE NEW OPERATING SYSTEM



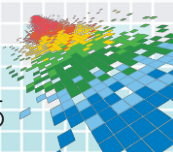
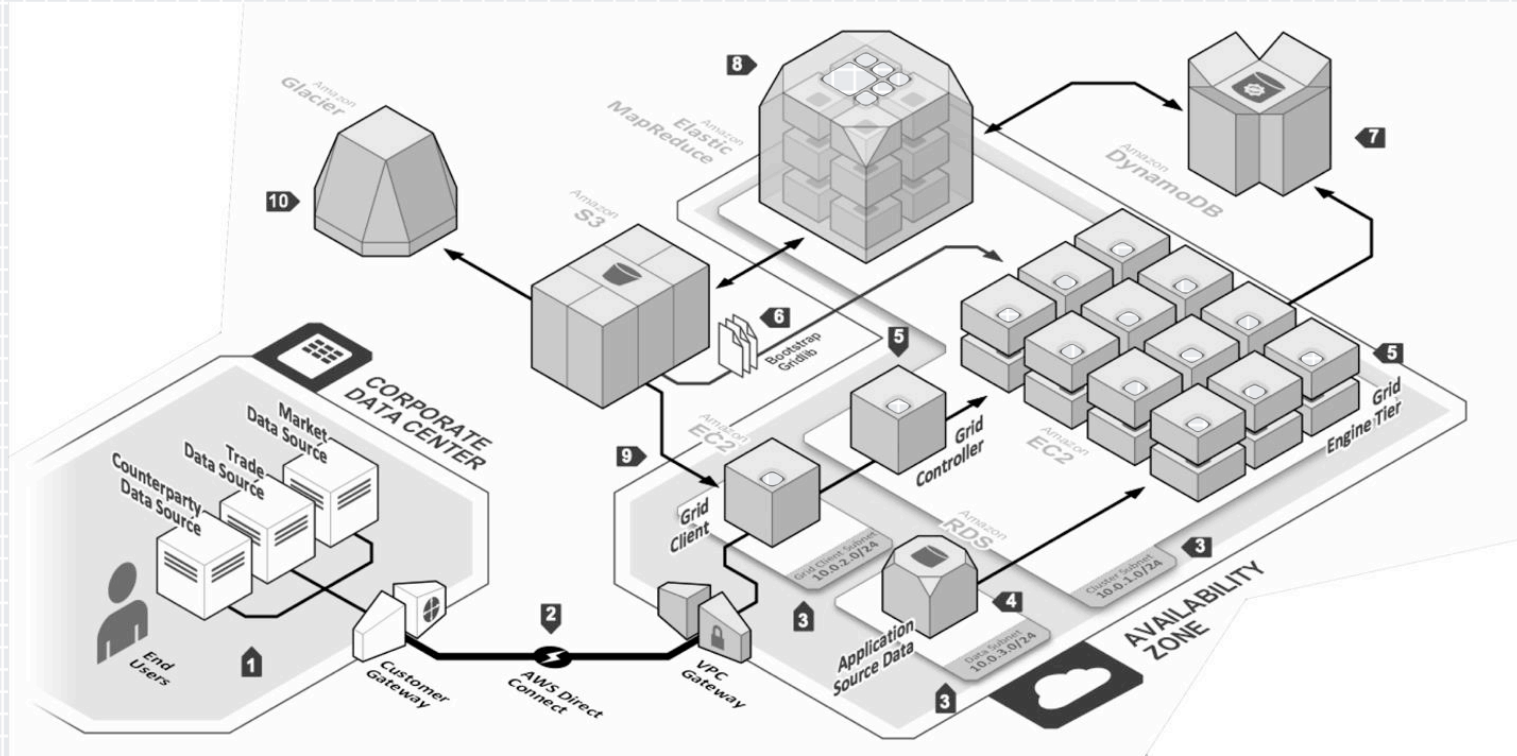
# CLOUD INFRASTRUCTURE IS CODE



# WHO'S SECURING THESE APPS?



# A Fully Realized Cloud Application (AWS)

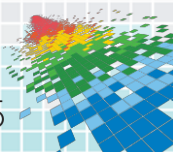
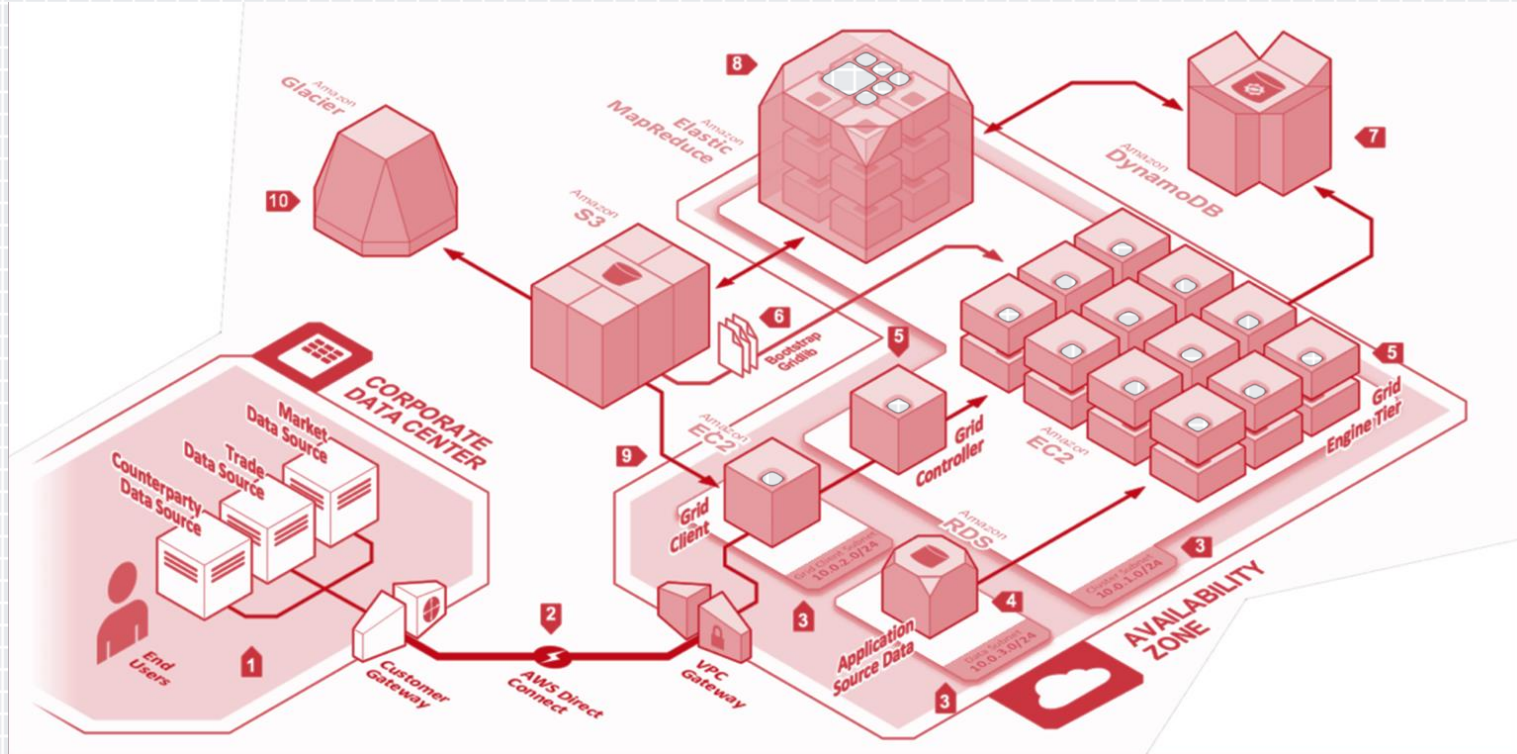


The diagram illustrates a multi-availability zone AWS architecture for a corporate data center. The components and their interactions are as follows:

- End Users (1):** Represented by a person icon, they connect to the Corporate Data Center.
- CORPORATE DATA CENTER:** Contains data sources (Counterparty, Trade, Market) and a Customer Gateway.
- AWS Direct Connect (2):** A dedicated network link connecting the Corporate Data Center to the AWS cloud.
- VPC Gateway:** Connects the VPC to the Corporate Data Center.
- Application Source Data (3):** Data ingested from the Corporate Data Center into the AWS cloud.
- Grid Client (3):** An Amazon EC2 instance that receives data from the Corporate Data Center.
- Grid Controller (5):** An Amazon EC2 instance that manages the Grid Engine Tier.
- Grid Engine Tier (5):** A cluster of Amazon EC2 instances that process data.
- Amazon S3 (6):** Used for storing data, including Brokered Grids.
- Amazon RDS (3):** A relational database instance for Application Source Data.
- Amazon DynamoDB (7):** A NoSQL database instance for Application Source Data.
- Amazon ElastiCache (3):** A cache instance for Application Source Data.
- Availability Zones:** The architecture is spread across three Availability Zones for high availability.

Red arrows indicate the path for third-party data ingestion, starting from the Corporate Data Center, through the AWS Direct Connect link, and into the Amazon S3 and Amazon EC2 instances.

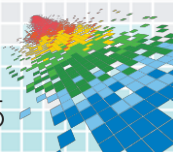
# Majority comes from the Cloud Provider



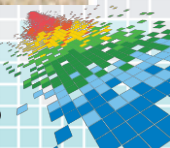


# You may be tempted to Ignore all that and...

- ◆ Treat your cloud provider as the digital equivalent of infinite rack space with a nice web interface
- ◆ Rely on your existing security controls
- ◆ Assume your cloud providers mile long list of compliance certifications protects you



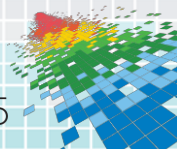
...after a few months this might be you





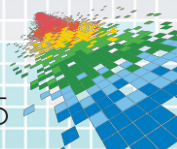
# Forklifting is Dangerous

- ◆ Forklifting is the process of taking legacy data center applications and loading them into a cloud provider with little or no changes
  - ◆ Often this is the most expensive way to go Cloud
- ◆ It is also dangerous
  - ◆ IDS and network access controls are unaware of AWS API activity
  - ◆ Previously ignored or de-prioritized vulnerabilities can become critical
  - ◆ The meaning of availability changes



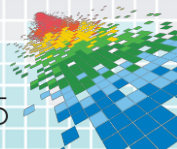
# Ok, just embrace “DevOps” to fix it!

- ◆ DevOps is a powerful approach to building and maintaining complex, fast moving systems but...
  - ◆ It requires a culture change throughout your engineering org
  - ◆ It does not mean go fire all your Ops people and let development figure it out now
  - ◆ What about Security....



# DevOps Culture tends to Fail Open

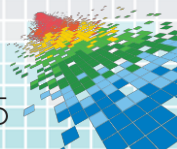
- ◆ Developers building systems who are new to the Cloud might now be the ones configuring the firewall rules and API Access Policies as well
- ◆ Developers are often pre-disposed (and compensated) to “Just make it work”
- ◆ Good DevOps should mean your engineering teams **care** about operations **a lot**, not that they are now responsible for it all by themselves



# EMERGENT INSECURITY

The individual components of your system may be secure but when deployed into the Cloud, the system becomes insecure

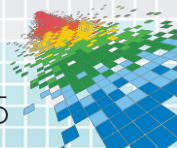
AND IT GETS WORSE AT SCALE





# 4 HORSEMEN OF EMERGENT INSECURITY

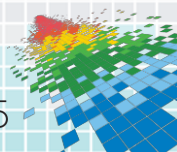
- ◆ **“Internet Weather”** – Cloud systems and API calls are subject to unpredictable, non-persistent, network latency, system performance and connection interruption
- ◆ **Guaranteed Failure** - System availability is a factor of redundancy and automation, not the stability and performance of monolithic systems
- ◆ **Software defined everything (SDe)** - Virtualized networks, network interfaces, file systems, computing power and more can change independent of the underlying system
- ◆ **Out-of-band management** – Cloud API’s operate outside of traditional security controls, can make all existing controls irrelevant



# What's the Real Attack Surface?

- ◆ “If your security sucks now, you’ll be pleasantly surprised by the lack of change when you move to Cloud.”\*
- ◆ **In reality is it gets worse**
  - ◆ AWS API endpoints are open by design, trumps all existing controls
  - ◆ Unexpected disclosures from provisioning systems and metadata
  - ◆ Private might be public, IP’s, data leaks through 3rd party services
  - ◆ Only thing standing between total compromise of your entire datacenter is the secrecy of your API keys

\*Chris Hoff, @beaker



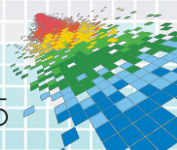


#RSA



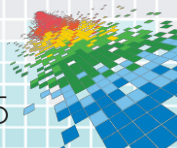
IN THE CLOUD

# API ACCESS == PHYSICAL ACCESS



# THE API BYPASSES TRADITIONAL CONTROLS

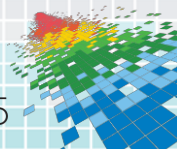
- ◆ **For example, start with an basic system**
  - ◆ Block all network traffic using host FW, throw away SSH keys/passwords, install a network, log all network traffic to/from system
  - ◆ In a traditional data center, system is now inaccessible, any attempt to access would be impossible and also detected
- ◆ **In the cloud however**
  - ◆ Use APIs to snapshot the disk, mount snapshots on different system, and extract everything without touching the network or system
  - ◆ Zero indication from traditional controls that any access has taken place
  - ◆ Same is true of cloud databases, I don't need your passwords or even SQL Injection, I just need to snapshot your DB





# API Credential Exposure Impact

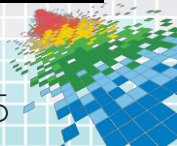
- ◆ The attacker can:
  - ◆ Sets up bitcoin mining operations within your cloud environment
  - ◆ Alter your applications to spread malware
  - ◆ Use your environment as a means to launch additional attacks
  - ◆ Download or manipulate all of your customers data





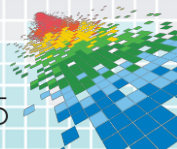
Or nuke your entire cloud environment

(from orbit, it's the only way to be sure)

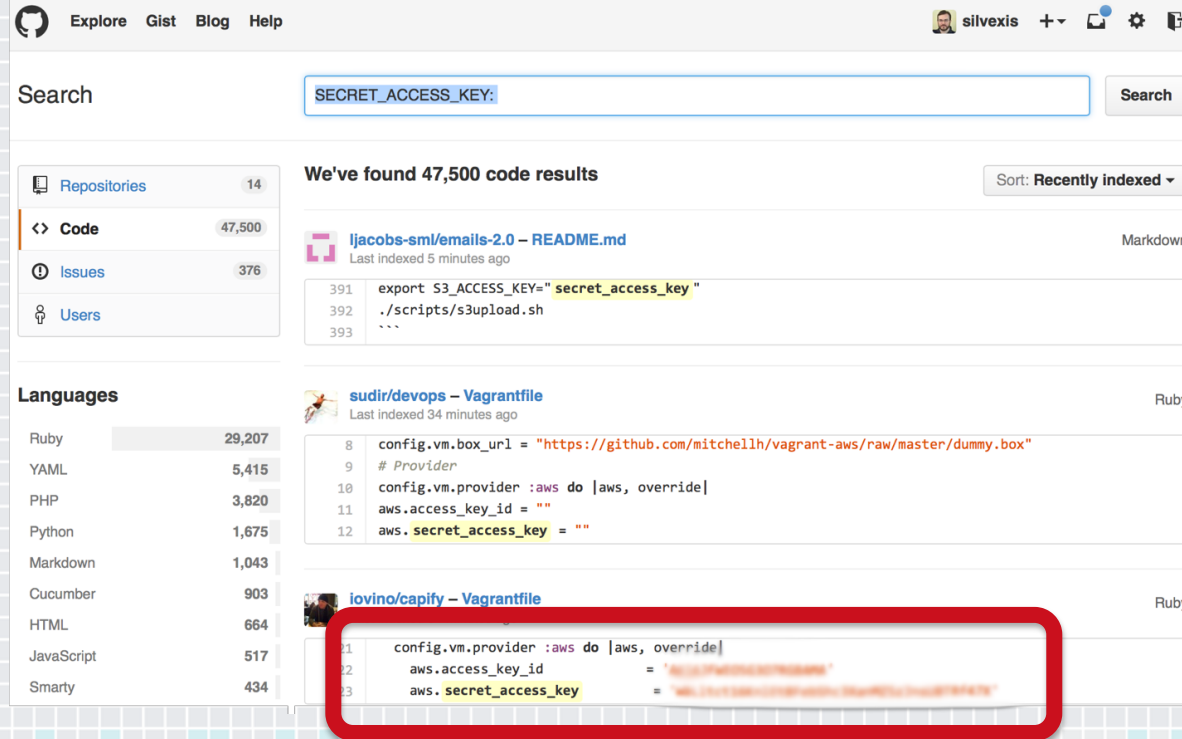


# Getting API keys

- ◆ **API keys checked into source control**
  - ◆ Many cloud providers are scanning public repos for API keys to protect you from yourself
  - ◆ If you using API keys in production apps however you are likely
- ◆ **Exploiting vulnerabilities to extract keys from Metadata or Bootstrap systems**
  - ◆ Harder to deal with, requires you to think differently when testing your apps for security issues



# GitHub Example



The screenshot shows the GitHub search interface. The search bar contains the text "SECRET\_ACCESS\_KEY:". The search results show 47,500 code results. The left sidebar shows the "Code" tab selected. The "Languages" section lists various programming languages with their respective counts. The main content area displays two search results. The first result is from "ljacobs-sml/emails-2.0 - README.md" and shows a code snippet with "secret\_access\_key" highlighted. The second result is from "sudir/devops - Vagrantfile" and shows a code snippet with "secret\_access\_key" highlighted. A red box highlights the second result, "iovinio/capify - Vagrantfile", which also shows a code snippet with "secret\_access\_key" highlighted.

Explore Gist Blog Help

silvexis

Search

SECRET\_ACCESS\_KEY:

Search

We've found 47,500 code results

Sort: Recently indexed

Repositories 14

Code 47,500

Issues 376

Users

Languages

Language	Count
Ruby	29,207
YAML	5,415
PHP	3,820
Python	1,675
Markdown	1,043
Cucumber	903
HTML	664
JavaScript	517
Smarty	434

**ljacobs-sml/emails-2.0 - README.md** (Markdown)

Last indexed 5 minutes ago

```
391 export S3_ACCESS_KEY="secret_access_key"
392 ./scripts/s3upload.sh
393 ...
```

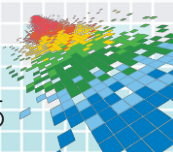
**sudir/devops - Vagrantfile** (Ruby)

Last indexed 34 minutes ago

```
8 config.vm.box_url = "https://github.com/mitchellh/vagrant-aws/raw/master/dummy.box"
9 # Provider
10 config.vm.provider :aws do |aws, override|
11   aws.access_key_id = ""
12   aws.secret_access_key = ""
```

**iovinio/capify - Vagrantfile** (Ruby)

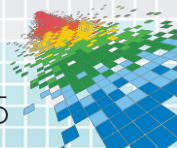
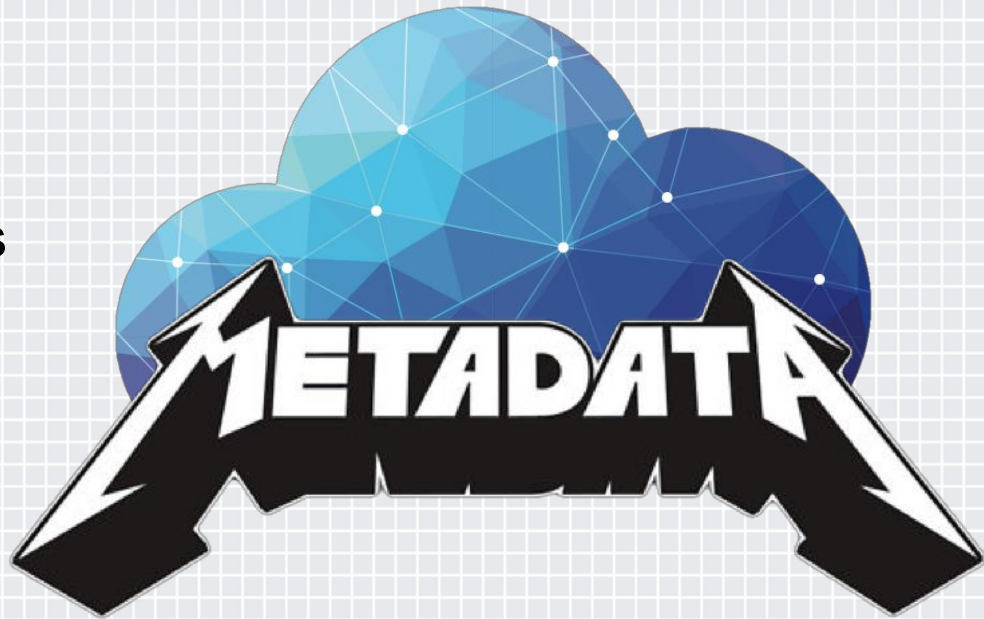
```
1 config.vm.provider :aws do |aws, override|
2   aws.access_key_id = ""
3   aws.secret_access_key = ""
```





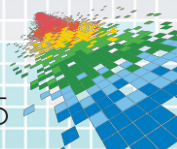
# Cloud Metadata

- ◆ Fundamentally a critical part of cloud infrastructure
- ◆ Simplifies provisioning of cloud systems, in many cases makes things more secure
- ◆ Something you need to be aware of and protect however



# What is Cloud Metadata?

- ◆ Based on RFC 3927 - Dynamic Configuration of IPv4 Link-Local Addresses
- ◆ Metadata can contain all kinds of awesome things, like startup scripts and your API access credentials
- ◆ All cloud providers have it
  - ◆ AWS: <http://169.254.169.254/latest/user-data>
  - ◆ Google: <http://169.254.169.254/computeMetadata/v1/>
  - ◆ DigitalOcean: <http://169.254.169.254/metadata/v1/>
  - ◆ OpenStack (includes RackSpace & IBM Bluemix): <http://169.254.169.254/openstack>
  - ◆ HP Helion: <http://169.254.169.254/2009-04-04/meta-data/>
- ◆ On Azure, metadata is not dynamic but is copied to `/var/lib/waagent` (linux) or `%SYSTEMDRIVE%\AzureData\CustomData.bin` (Windows)



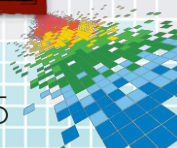
# Example: Accessing AWS Metadata

```
$ wget -q -O - http://169.254.169.254/latest/meta-data/
ami-id
ami-launch-index
ami-manifest-path
block-device-mapping/
hostname
iam/
instance-action
instance-id
instance-type
kernel-id
local-hostname
local-ipv4
mac
metrics/
network/
placement/
profile
public-hostname
public-ipv4
public-keys/
reservation-id
security-groups
```

Each one of these represents meta data you can access

- AWS command line tool ec2-metadata will extract some (but not all!) of the metadata
- To get everything use wget or curl

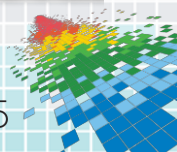
```
$ wget -q -O - http://169.254.169.254/latest/meta-data/instance-id
i-0496132e
```



# Metadata Credential Extraction (AWS)

```
[ec2-user@ip-10-0-1-125 ~]$ wget -q -O - http://169.254.169.254/latest/meta-data/iam/security-credentials/<YOUR PROFILE HERE>
```

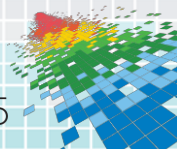
```
{
  "Code" : "Success",
  "LastUpdated" : "2014-08-13T16:43:54Z",
  "Type" : "AWS-HMAC",
  "AccessKeyId" : "ASIAJ4YSTDWONSUFPHIA",
  "SecretAccessKey" : "iqFuJWcj9AUaBXe0tbuc6MC70oQW2wehWufZ9cQV",
  "Token" : "AQoDYXdzEGlaOAMskSUj1Ing9DHLTlQmd
+vdImxTCnAbrNGcGPbv9jEPY05LDMMLBAjVdklFo7vS8HnEdrH3ea0T7f8aXW9BGMSdc/iF94PTi8+kO5sxgboy4XPB
+Bh44xHSKFV4WIrMKfMUwAftcieER7z6CakegOoe6Q/H0PsK9GpS1p06g+iyZLw8mT5ADz9zGUQTf
+P3anQ3da132SWYEiJR0fTQCuKqE8/dpLbnmdhOn3WyW8eF3TJFPd8/L0MQak3EMgolpAxm+eWAMj1B5Crewy4sbvBzf
+GcemFYiMClS9YgFxCxOexV09j9nPos/d9VRpFakmltWAS+sqHKz1zxLidWJewUfuhyLSxcR5xOeZYJ6/
Pt6bQitf21ep6FJExEGE3Ho0A10z4tv9Yo5c2tPafEhWsACBOia
+kpQExftmuIulmkRK9NugNuKcd0OzDkoftkpIFAj09oP2tgsDuImc0R3LSciJbmgLZsG1UuEpNpTnr
+7DmbvmrJWihfTD2hJzFAzhvsN8ytt
+mWkSGAeBhA6UosCgj2VjBIkHG6kkTcL9FnEU7XPoKqVsIusRqwZY00eITsL4dE38pcfHhmbPaCI6H5Tbj04FuRQc
+iFQaFBvCD3q66fBQ==",
  "Expiration" : "2014-08-13T23:05:19Z"
}
```





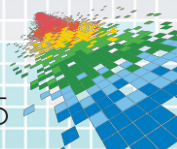
# Old Vulnerabilities, New Life

- ◆ These vulnerabilities can result in a total Cloud environment compromise
  - ◆ CWE-918: SSRF
  - ◆ CWE-611: XXE
  - ◆ CWE-441: Unintended Proxy or Intermediary
  - ◆ CWE-77: Command Injection
  - ◆ CWE-200: Information Exposure
  - ◆ CWE-268: Privilege Chaining
- ◆ Why?
  - ◆ All of these can lead to unintended exposure of metadata



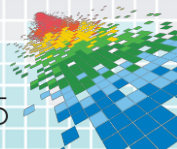
# Real world Examples

- ◆ Prezi experienced this first hand
  - ◆ <http://engineering.prezi.com/blog/2014/03/24/prezi-got-pwned-a-tale-of-responsible-disclosure/>
- ◆ Andres Riancho demonstrated this in his BlackHat talk “Pivoting in Amazon Clouds”
  - ◆ Exploited SSRF vuln to inject malicious AWS SQS messages to then exploit a celery/python vuln and take control



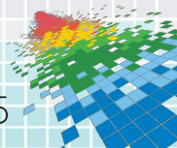
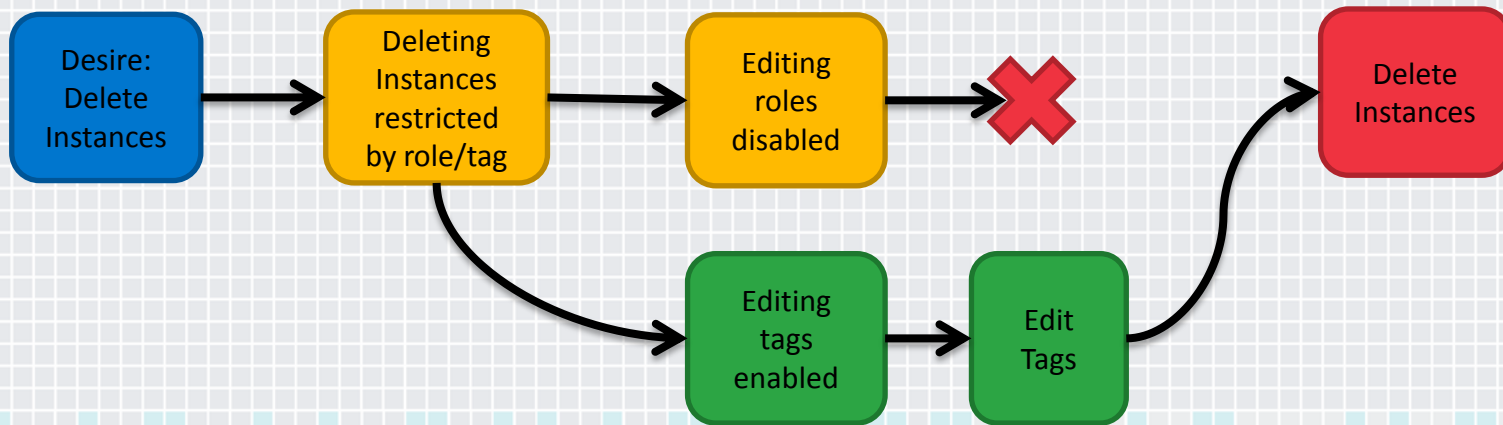
# Excuse me, your Cloud is leaking

- ◆ Tagging of Cloud assets is critical to keeping track of things
- ◆ Tags however often can contain private data that does not have strong access protections
- ◆ If you use any 3rd party Cloud management system your tag data is replicated to those systems
- ◆ I've seen customer data, API keys and passwords in tags!!!



# Chained Privilege Escalation

- ◆ Cloud API permissions are extremely rich
  - ◆ For example there are almost 1000 policy/service permissions in AWS
  - ◆ Locking down only user admin permissions is not good enough



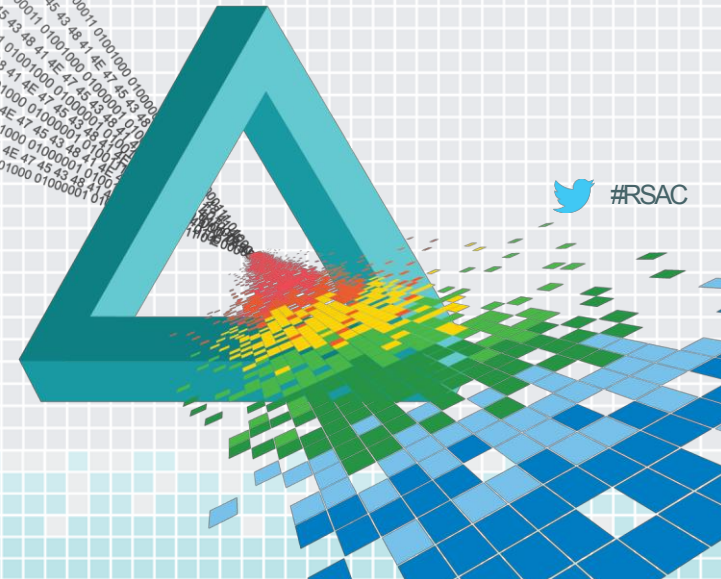




# RSA<sup>®</sup>Conference2015

San Francisco | April 20-24 | Moscone Center

## Protecting Yourself

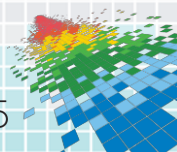


## FOR NEW PROJECTS:

### Consider an eventually consistent security model

Instead of trying to enforce change control which creates brittle systems that are insecure and not survivable, design your systems to be eventually consistent with your security and operational goals

If your system requires strict change control to maintain order, in the Cloud, you will eventually have chaos.



# Eventually consistent security Is a big shift

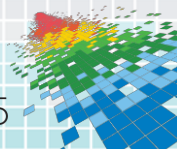
## Instead of this:

- ◆ Control change
- ◆ Create Stable Environments
- ◆ Disaster Recovery Plan

## Do this:

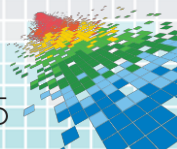
- ◆ React to Change
- ◆ Create Hostile Environments
- ◆ Disaster Survivability

Warning: If your application requires that you never drop a single transaction or can account for every single change, this approach might not be for you



# FOR ALL PROJECTS: Re-think your Threat Model

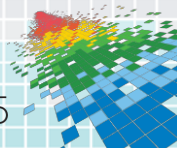
- ◆ Low priority or “mostly harmless” software vulnerabilities might now be deadly
- ◆ API Access circumvents traditional controls
- ◆ Any one of these could mean game over
  - ◆ CWE-918: SSRF
  - ◆ CWE-611: XXE
  - ◆ CWE-441: Unintended Proxy or Intermediary
  - ◆ CWE-77: Command Injection
  - ◆ CWE-200: Information Exposure
  - ◆ CWE-268: Privilege Chaining



# Control and Audit API Access

- ◆ IP Restrictions are not just for firewalls
  - ◆ Keys locked down to specific IP's have less chance of damage
- ◆ Require 2 Factor Auth on Risky permissions
  - ◆ Highly unlikely your app needs to be able to create or edit users
- ◆ Regularly audit your API permissions

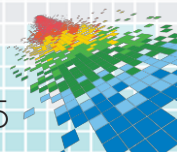
```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Deny",
    "Action": "*",
    "Resource": "*",
    "Condition": {
      "NotIpAddress": {
        "aws:SourceIp": ["192.168.1.1/32"]
      }
    }
  }]
}
```





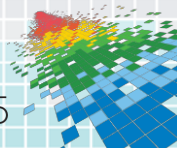
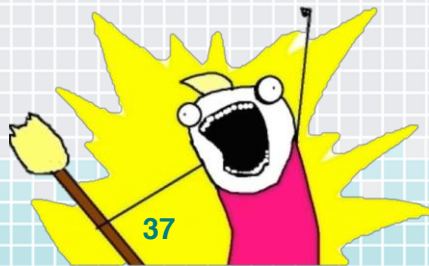
# Don't use your bill as an IDS

- ◆ Many cloud customers use their bill as their cloud IDS
- ◆ So far, most of these people have been very, very lucky
- ◆ Instead:
  - ◆ Turn on billing alerts to get near real time notifications
  - ◆ Watch your API logs for suspicious activity
  - ◆ Don't have API logging turned on you say?!



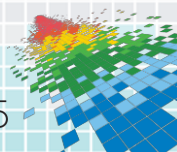
# Log all the things, especially API activity

- ◆ If you do nothing else, do this now!
- ◆ Logging is off by default in most cloud providers and is the most important thing you can log
- ◆ In AWS, turn on CloudTrail for all regions
- ◆ Use LogStash, develop your own, or buy a commercial solution to make API activity logs accessible **and** monitored



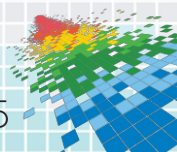
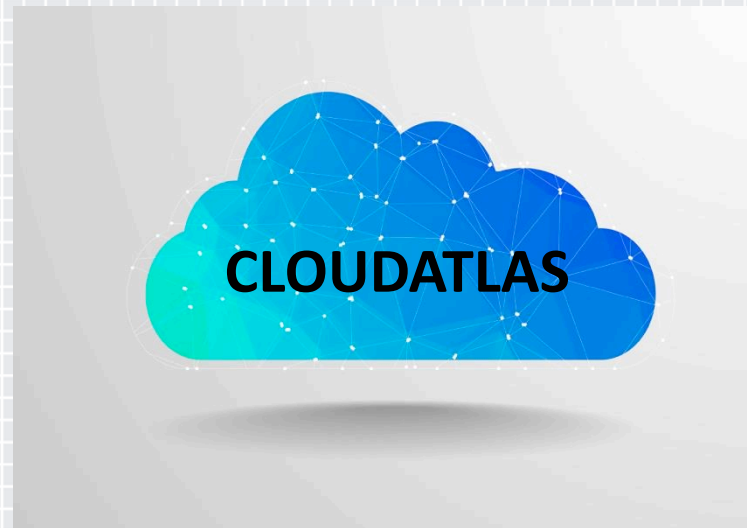
# Free Cloud Security Analysis Tools

- ◆ Amazon (AWS)
  - ◆ Netflix Security Monkey
    - ◆ <http://techblog.netflix.com/2014/06/announcing-security-monkey-aws-security.html>
  - ◆ Trusted Advisor
    - ◆ <https://aws.amazon.com/premiumsupport/trustedadvisor/>
- ◆ Microsoft Azure
  - ◆ Azure Operational Insights (in preview)
- ◆ Google
  - ◆ Google Cloud Security Scanner (for AppEngine)
    - ◆ <https://cloud.google.com/tools/security-scanner/>



# Application Focused Cloud Security Research

- ◆ Veracode R&D project  
“Cloudatlas”
  - ◆ I am looking for beta participants, please contact me if interested
  - ◆ Investigating how cloud changes application security
  - ◆ Currently uncharted waters



# Apply What You Have Learned Today

- ◆ Next week you should:
  - ◆ Turn on API Logging, do it NOW
- ◆ In the first three months following this presentation you should:
  - ◆ Analyze your applications for vulnerabilities that can expose metadata
- ◆ Within six months you should:
  - ◆ Rethink your threat model, consider an eventual consistency security model that doesn't rely on strict controls to be effective

