RSAConference2019

San Francisco | March 4 – 8 | Moscone Center

SESSION ID: ASD-R11

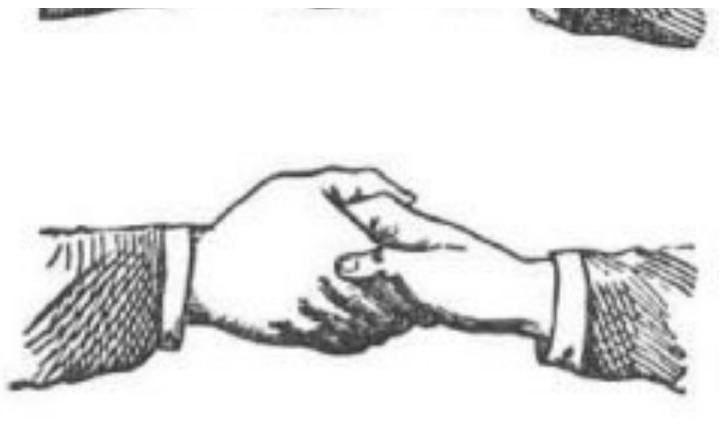
API Security: Assume Possible Interference

Julie Tsai

Cybersecurity Leader & Advisor

#RSAC

APIs - Application Programming Interfaces: How the 2019 World Wide Web Communicates



Apprentice to the pass grip of a fellow craft

Pass-grip of a mark master mason

Real grip of a fellow craft

Real grip of a mark master mason

A Protocol?

A FRAMEWORK FOR INTERACTION, EXCHANGING DATA AND SERVICES

Be a Journalist to Secure APIs

WHO

WHAT

WHERE

WHEN

WHY

HOW

Like Hilde

Specifically…

WHO => aka AuthN (Authentication) via **AuthID**

WHAT => aka AuthZ (Authorization) via **OAuth**

WHERE => Are Sessions, Keys, Tokens **Stored**?

WHEN => Does Access and **Data** Expire?

WHY => Our **Purpose** for Accessing This Data

HOW ***Least-Privilege, Explicit Trust, Implicit Deny***

360-View on Exploitability

Do the services interact in a way that creates the least astonishment?

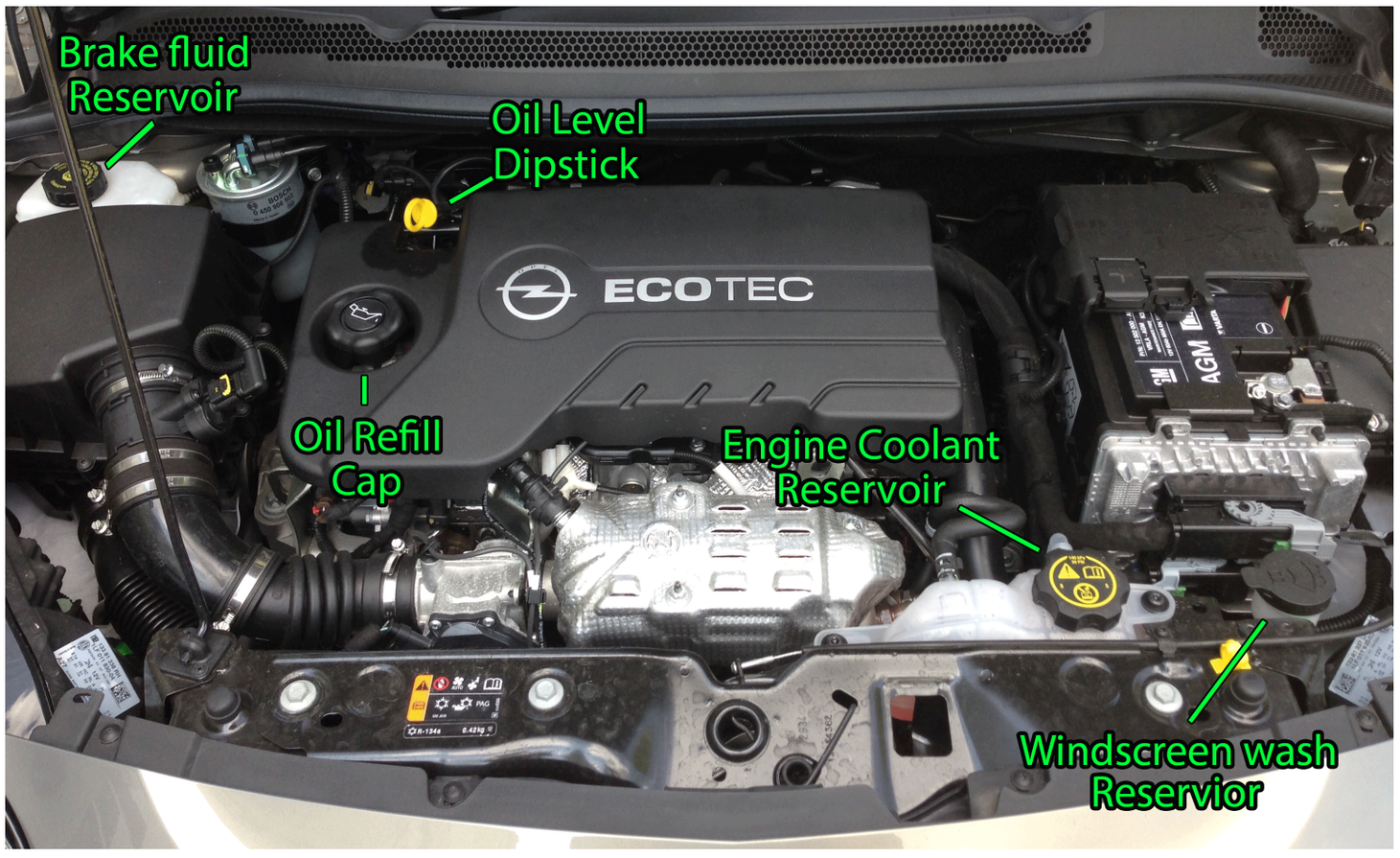
Is the lifecycle and travel path of the data known?

Is the data forgotten after use? Can this be proven?

What if authorization models for different services get mixed? To different users, with different privileges?

Wait, didn't containers and service isolation solve this?

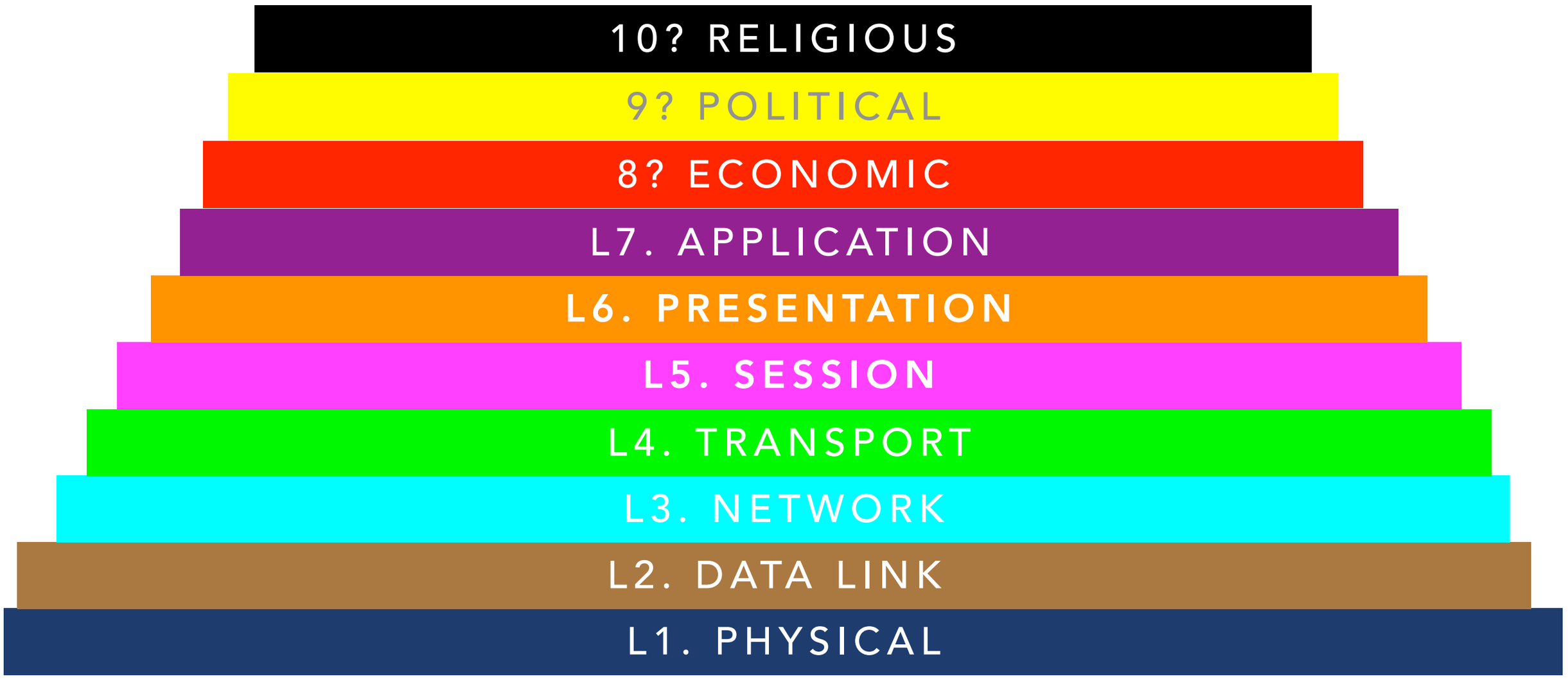
BUT API IS JUST THE BEGINNING…



There's still some stuff beneath the hood

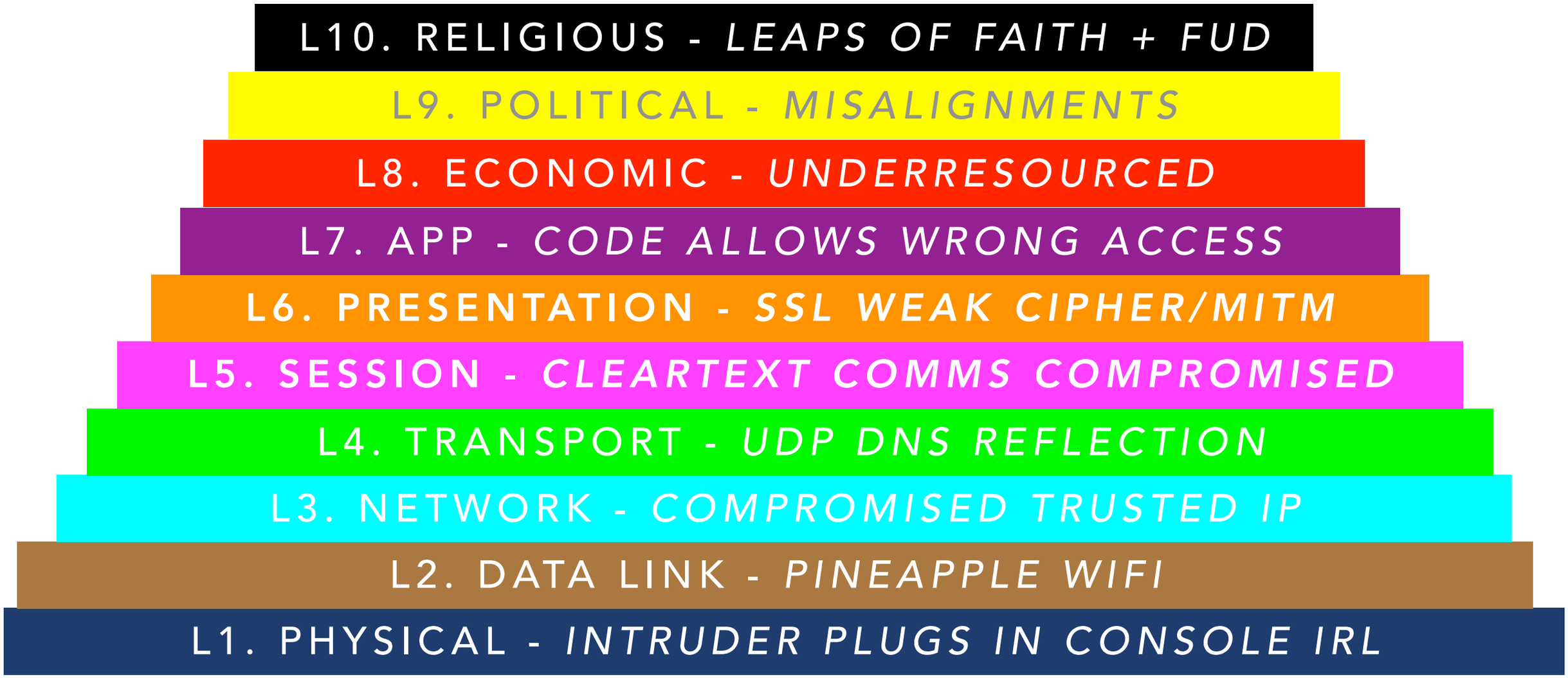
Defense-in-Depth Matters

SECURITY IS FULL-STACK OSI PROBLEM - LAYERS1-7,8,9...10



Defense-in-Depth Matters

What could happen?!



RSAConference2019

SHIFT LEFT & SHIFT RIGHT

Upstream Secure Containers & End-State Monitoring

Containers and Chroot Jails: Classic Tune, New Words

Dedicated namespaces and resources. Process isolation.
Ephemerality.

But shared kernel.

*Example threat model - **https://www.twistlock.com/2016/12/06/protecting-containerized-apps/**: Malicious container attacks underlying OS or other containers*

Container SRC as Workhorses, Not Just Cattle

(Skipping the pets - you know why you shouldn't make pets)

Container Lifecycle: Cattle v. Workhorse

1. *Born*

Instance created

2. *Eat/Sleep*

Powered but no new changes to instance, or updates to image

3. *Expire*

When deprecated or compromised/ vulnerable, instance terminated with no feedback

1. *Born*

Instance created

2. *Trained*

Secured configs/ images

3. *Updated*

Config & Repos continuously updated

4. *Corral*

Use cgroups to authorize. System calls whitelists.

5. *Maintain or Expire*

Maintain by chef/ puppet/cfengine/ ansible/salt etc. Or expire.

Container SRC as Workhorses, not Just Cattle #RSAC

(Skipping the pets - you know why you shouldn't make pets)

Container images must be maintained, not just fire and forget or expiraton.

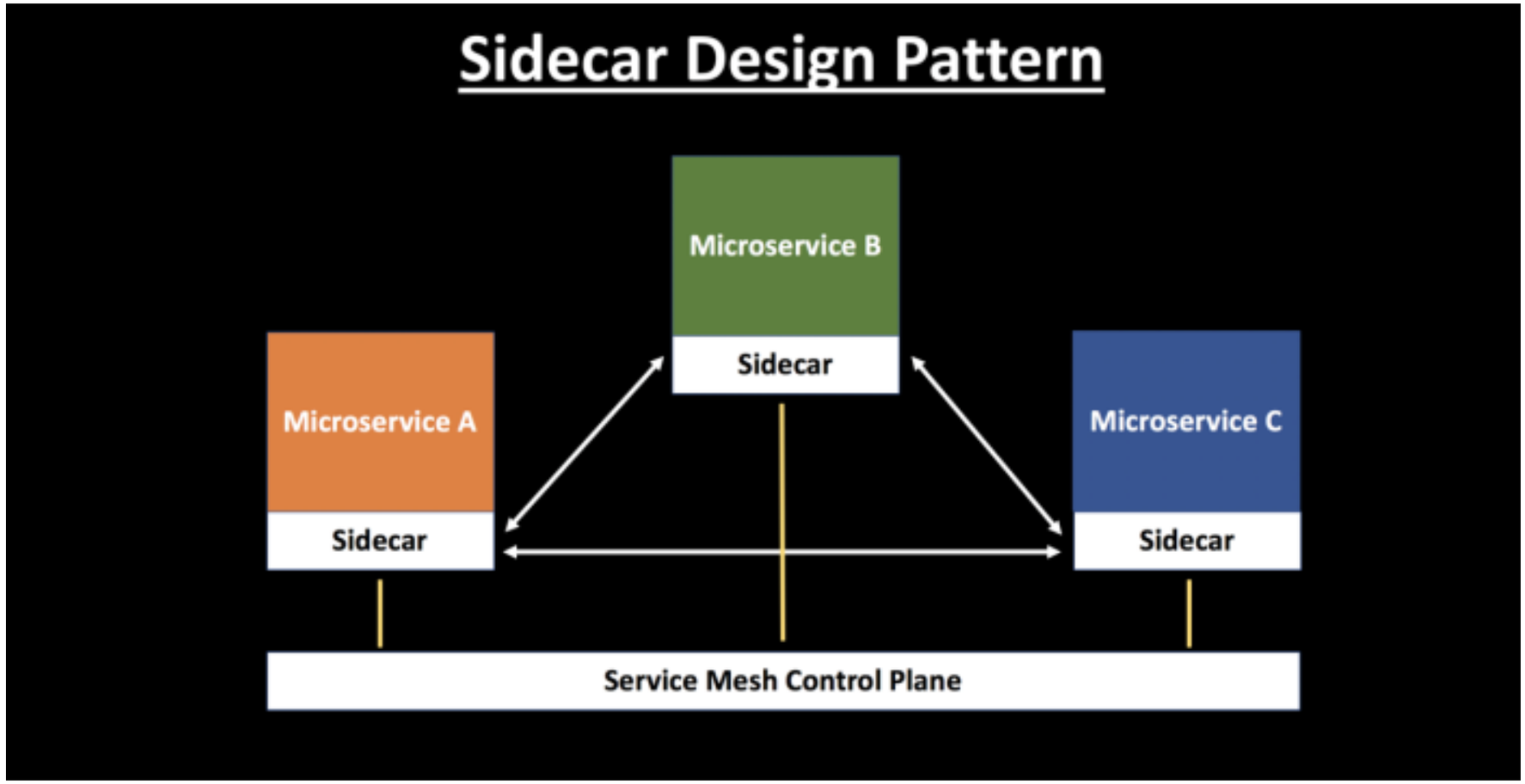
Expiring the compromised container or service (i.e. shooting the cattle) only solves once. But the herd needs to be inoculated – to evolve – beyond issues to prevent recurrence. Or, think of baking a cake vs. stir-frying*.

*CREDIT ~DALVES.

—> So what? Well, upstream *learned* remediation drives down Rate of Defect Recurrence! In practical terms, you note and track the vuln, and incorporate the fix in the Dockerfile (or source code or secure patch repo) ***in source-controlled, change-managed config***. ***So it scales beyond that one image.***

Example ***https://engineering.salesforce.com/open-sourcing-dockerfile-image-update-6400121c1a75***

Workhorse Communication: Service Mesh



From *https://dotnetvibes.com/2018/07/23/sidecar-design-pattern-in-your-microservices-ecosystem/*

Less Is More in Containers

WHAT'S A HARDENED CONTAINER?

LET'S WALK THROUGH THIS:

1. strip down packages, especially easily exploitable ones useful to hackers *(telnet, ftp, make, gdb, nmap, strace,* legacy daemons*)*
2. minimize host/network services and daemons
3. ssh and account enforcement
4. logs
5. key file/config alerting
6. vulns + patching scanning
7. and… ***scrap anything in there you don't need.***

Monitoring the End-State Abuse, Anomalies, and Access

What's Normal? Anomalies in conjunction tell the story.

We don't know - and can't comprehensively predict - what we don't know. Root cause and security incidents don't present themselves as such upfront.

What to do?

Flag + Correlate for simple indicators of what shouldn't be happening. i.e. Resources volume or ways that shouldn't be used. Things that shouldn't be accessed. Non-cyclical transaction trends.

Post-Flight Gut-Check DevSecOps

AKA ARE WE SEC-ING AND OPS-ING WITH OUR DEV-ING?

1. Security, Compliance, Privacy

When are your S-C-P pros leveraged in the process for the product? Customer development, inception or design, dev or testing, alpha – or when an "event" happens?

2. Uptime and Performance

Do we truly know it better than our end-users?

3. **Scale Complexity and Change**

Is there the necessary speed in deployments/change across all of your teams?

Can you accommodate (or tolerate) diverse deployment & build methods across teams?

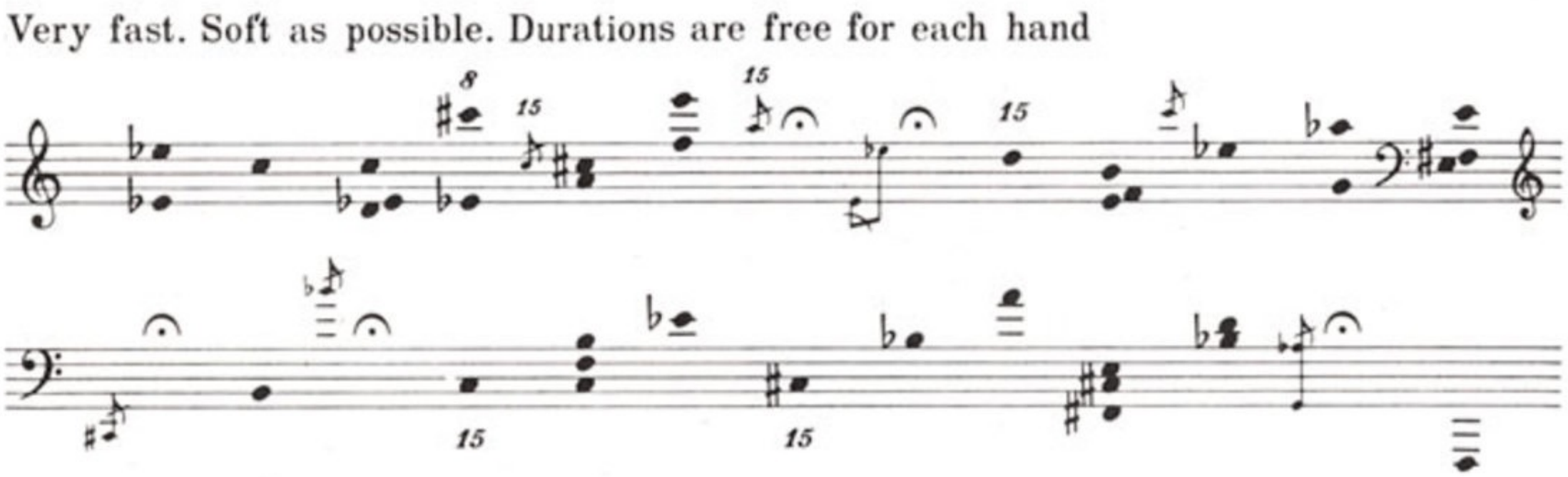
Do you get the same level of visibility, health, and security from each of them?

RSAConference2019

Guidelines & Guardrails

THE RIGHT AMOUNT OF STRUCTURE

The Right Amount of Structure - a Score



Last Pieces *for solo piano, by Morton Feldman; beginning of the fourth and final section. [© 1963 C. F. Peters Corp. Used by permission.] The directions shown here are the only performance notes provided (each section has similar instructions). The notes were composed rather than being determined by chance operations, and the performer is free to improvise durations.*

Where To Use Guardrails 1

1. AuthN/AuthZ

(AutheNtication/AuthoriZation)

Where To Use Guardrails 2

1. AuthN/AuthZ
2. Lifecycle of the Data

Where To Use Guardrails 3

1. AuthN/AuthZ
2. Lifecycle of the Data
3. **Build Integration**

Where To Use Guardrails 4

1. AuthN/AuthZ
2. Lifecycle of the Data
3. Build Integration
4. Production Deployment

Where To Use Guardrails 5

1. AuthN/AuthZ
2. Lifecycle of the Data
3. Build Integration
4. Production Deployment
5. **Change - Code, Configs, Dependencies**

Where To Use Guardrails 6

1. AuthN/AuthZ
2. Lifecycle of the Data
3. Build Integration
4. Production Deployment
5. Change - Code, Configs, Dependencies
6. **Regression Test - Catch the Ripple**

Where To Use Guardrails 7

1. AuthN/AuthZ
2. Lifecycle of the Data
3. Build Integration
4. Production Deployment
5. Change - Code, Configs, Dependencies
6. Regression Test - Catch the Ripple
7. **Security & Compliance Requirements**

Where To Use Guidelines 1

...WITH INTERNAL CONSISTENCY IN RELEVANT SCOPE

1. Develop/Build in the OS/Language You Will

Where To Use Guidelines 2

...WITH INTERNAL CONSISTENCY IN RELEVANT SCOPE

1. Develop/Build in the OS/Language You Will
2. **Use the Tools That Work for You, without Significantly Hampering Others**

Where To Use Guidelines 3

...WITH INTERNAL CONSISTENCY IN RELEVANT SCOPE

1. Develop/Build in the OS/Language You Will
2. Use the Tools That Work for You, without Significantly Hampering Others
3. Deployment Methods. CI/CD Innovation. Private or Public Cloud. Bare Metal. Multi-cloud or Single.

Where To Use Guidelines 4

...WITH INTERNAL CONSISTENCY IN RELEVANT SCOPE

1. Develop/Build in the OS/Language You Will
2. Use the Tools That Work for You, without Significantly Hampering Others
3. Deployment Methods. CI/CD Innovation. Private or Public Cloud. Bare Metal. Multi-cloud or Single.
4. **How DevSecOps Manifests in Your Org**

Where To Use Guidelines 5

...WITH INTERNAL CONSISTENCY IN RELEVANT SCOPE

1. Develop/Build in the OS/Language You Will
2. Use the Tools That Work for You, without Significantly Hampering Others
3. Deployment Methods. CI/CD Innovation. Private or Public Cloud. Bare Metal. Multi-cloud or Single.
4. How DevSecOps Manifests in Your Org
5. **Culture — Congruence of the Walk with the Talk**

Are We There Yet?

From monolith to microservices…

Decoupling dependencies…

Agile, independent teams…

Continuous Deployment of epheremal containers…

Open, networked world…

Are We There Yet?

…with Great Power Comes Great Responsibility.

References & Image Credits