ISACA®
Canberra Chapter
May 2021

Using Technology to Help With GRC in Public Cloud and Modern Application Environments

Shain Singh
Cloud/5G Security Architect [APCJ]
shsingh@ieee.org

Who am I?

Shain Singh

Cloud/5G Security Architect @F5

Social

https://linkedin.com/in/shsingh

shsingh@ieee.org

https://twitter.com/shainsingh

https://github.com/shsingh

https://shain.io

Professional Memberships

Why this talk?

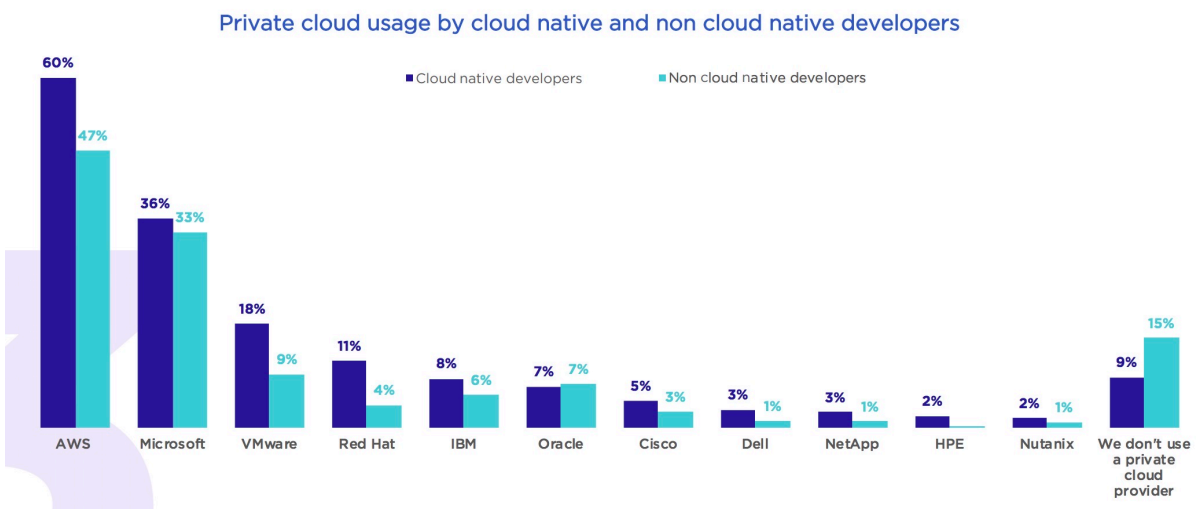
Make Security Great Again™

- Blue Teaming should be as fun as Red Teaming
- Create cultural shift in organizations by embracing ***DevOps principles***
  - Security should move from a “NO by default” to a “YES with caveats”
  - Meeting developers halfway encourages them to do the same
- Leverage toolsets and methodologies that are becoming common-place for application and infrastructure deployment

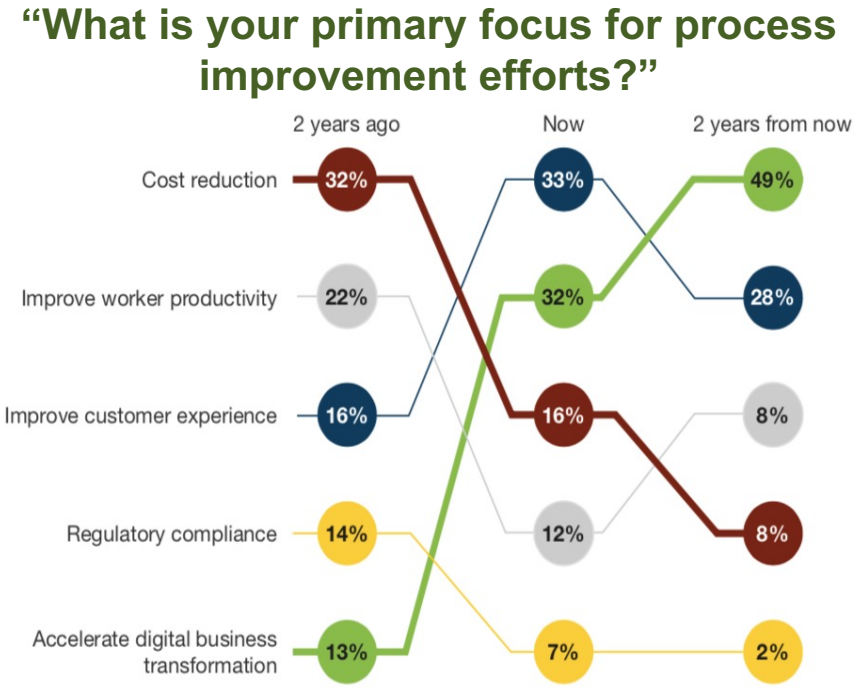
Disclaimer

- I am not an expert, I am a curious security practitioner learning how these new technologies can help with raising the bar

Cloud adoption



Software Architecture and Design InfoQ Trends Report—April 2020



Forrester [2018] - The Growing Importance Of Process To Digital Transformation

Container adoption

Compliance can assist to set guardrails

CJIS
Criminal Justice Information Services

DoD SRG
DoD Data Processing

FedRAMP
Government Data Standards

FERPA
Educational Privacy Act

FFIEC
Financial Institutions Regulation

FIPS
Government Security Standards

FISMA
Federal Information Security Management

GxP
Quality Guidelines and Regulations

HIPAA
Protected Health Information

HITRUST CSF
Health Information Trust Alliance Common Security Framework

CSA
Cloud Security Alliance Controls

ISO 9001
Global Quality Standard

ISO 27001
Security Management Controls

ISO 27017
Cloud Specific Controls

ISO 27018
Personal Data Protection

PCI DSS Level 1
Payment Card Standards

SOC 1
Audit Controls Report

SOC 2
Security, Availability, & Confidentiality Report

SOC 3
General Controls Report

FISC [Japan]
Financial Industry Information Systems

IRAP [Australia]
Australian Security Standards

K-ISMS [Korea]
Korean Information Security

MTCS Tier 3 [Singapore]
Multi-Tier Cloud Security Standard

OSPAR [Singapore]
Outsourcing Guidelines

Industry standards define deployment patterns

Cloud Controls Matrix
Security Guidance For Critical Areas of Focus in Cloud Computing

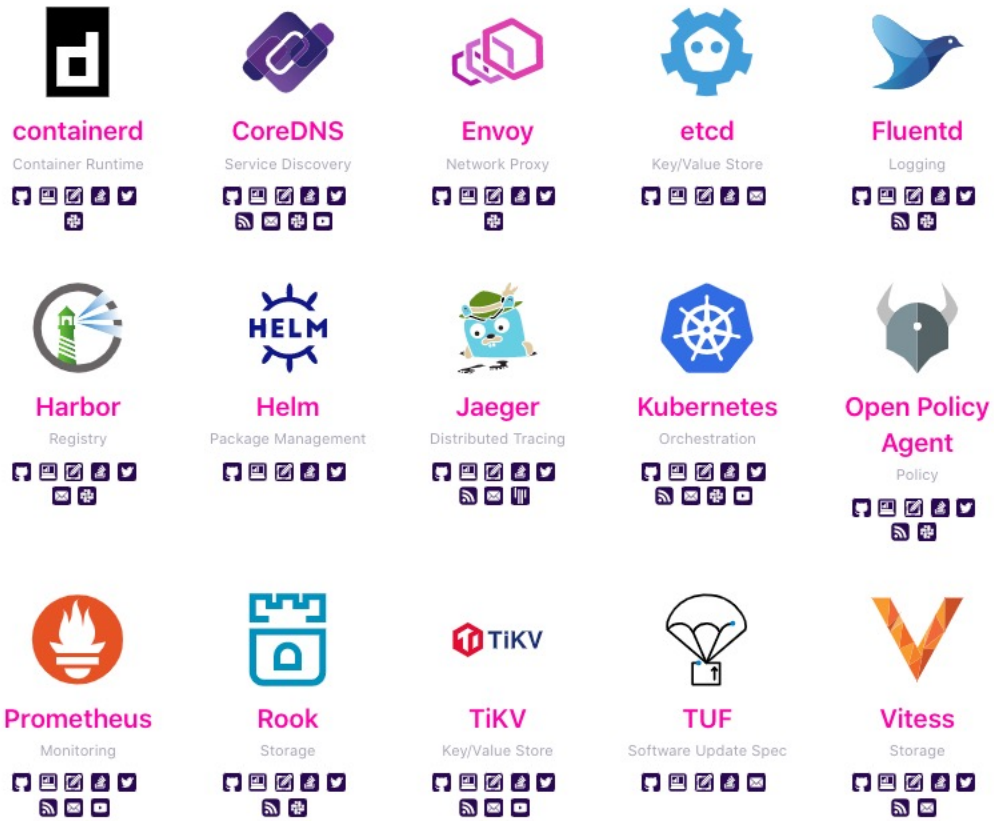
Benefits, Risks and Recommendations For Information Security

Cybersecurity Framework

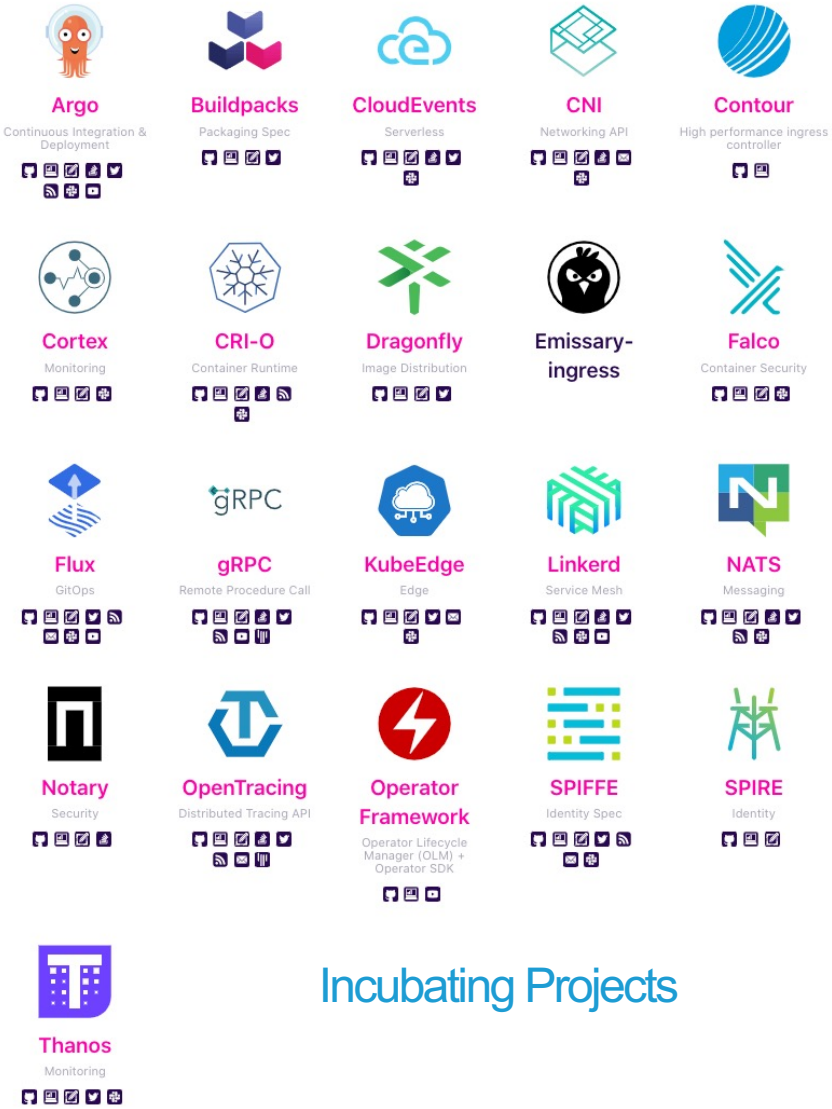
Secure Cloud Computing Architecture

CIS Benchmarks

Cloud native technologies



Graduated Projects



Incubating Projects

https://www.cncf.io

Policy and Controls

Why use a service mesh?

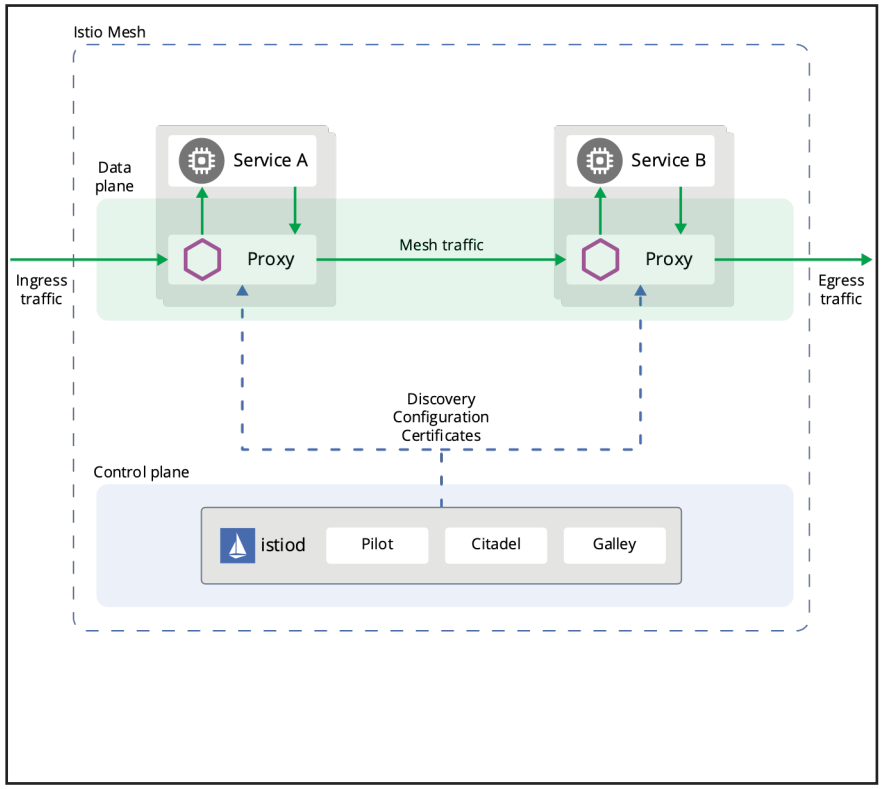
- The distributed cross-domain nature of microservices needs secure token service (STS), key management and encryption services for authentication and authorization, and secure communication protocols.
- The ephemeral nature of clustered containers (by which microservices are implemented) calls for secure service discovery.
- The availability requirement calls for:
  - (a) resiliency techniques, such as load balancing, circuit breaking, and throttling
  - (b) continuous monitoring (for the health of the service).
- The *service mesh* is the best-known approach that can facilitate specification of these requirements at a level of abstraction such that it can be uniformly and consistently defined while also being effectively implemented without making changes to individual microservice code.çç

NIST SP 800-204A - Building Secure Microservices-based Applications Using Service-Mesh Architecture

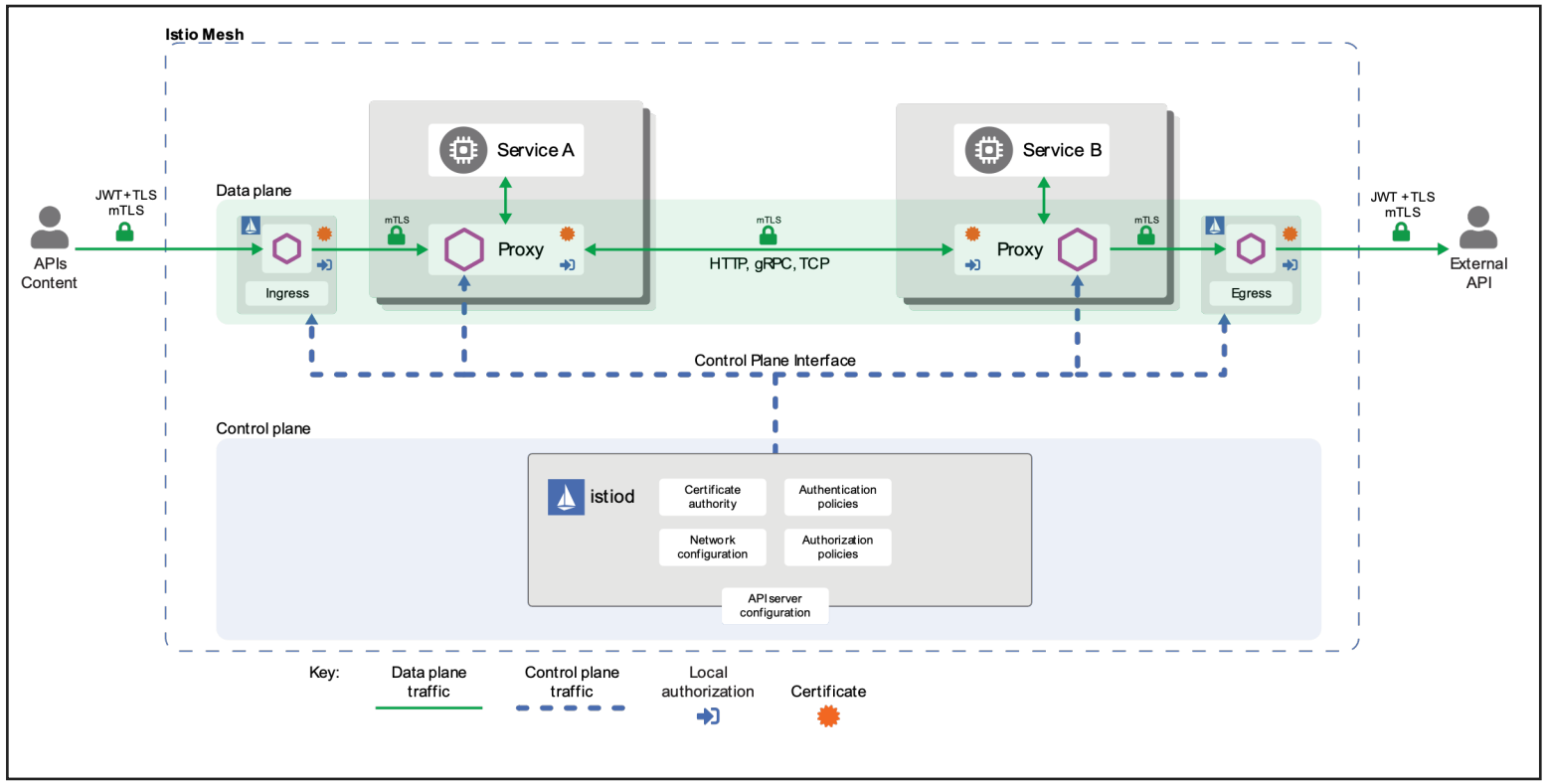
- Deployment architecture in cloud-native applications now consists of loosely coupled components (microservices), with all application services provided through a dedicated infrastructure (service mesh) independent of the application code.
- Two critical security requirements in this architecture are
  - (a) to build the concept of zero trust by enabling mutual authentication in communication between any pair of services
  - (b) a robust access control mechanism based on an access control model such as Attribute-based Access Control (ABAC) that can be used to express a wide set of policies and is scalable in terms of user base, objects (resources), and deployment environment.

NIST SP 800-204B - Attribute-based Access Control for Microservices-based Applications using a Service Mesh

What is a service mesh?



Components



Security Architecture

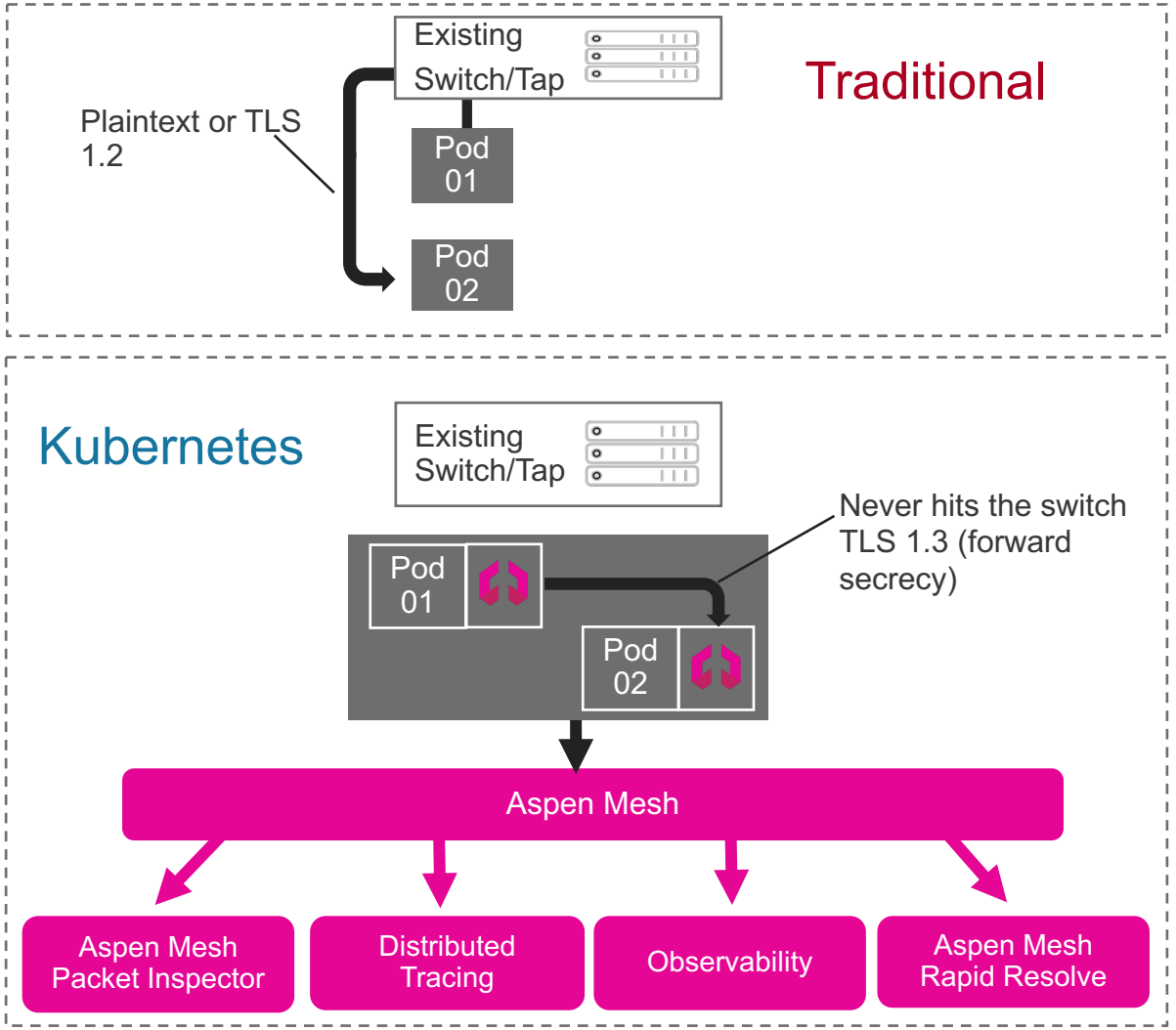
Example – vendor implementation of service mesh

Challenges

- Packet-level inspection of flows in container environment
- Key management and mTLS 1.3 PFS challenges
- Lawful intercept and compliance requirements
- Leverage existing packet broker investment
- Operations troubleshooting, knowledge and training

Solution: Aspen Mesh Packet Inspector

- Inter-service capture at sidecar
- Pre-encryption tapping
- Compatible with TLS 1.3 Forward Secrecy
- Integrates into existing infrastructure & automation
- Scalable and extensible



What are SPIFFE/SPIRE?

https://spiffe.io/docs/latest/spiffe-about/overview/

- A set of specifications that cover how a workload should retrieve and use its identity
  - SPIFFE ID
  - SPIFFE Verifiable Identity Documents (SVIDs)
  - The SPIFFE Workload API

https://spiffe.io/docs/latest/spire-about/

- The SPIFFE Runtime Environment
- Open-source Reference Implementation that applies the SPIFFE Workload API for a variety of platforms and environments
- Highly extensible through plug-ins

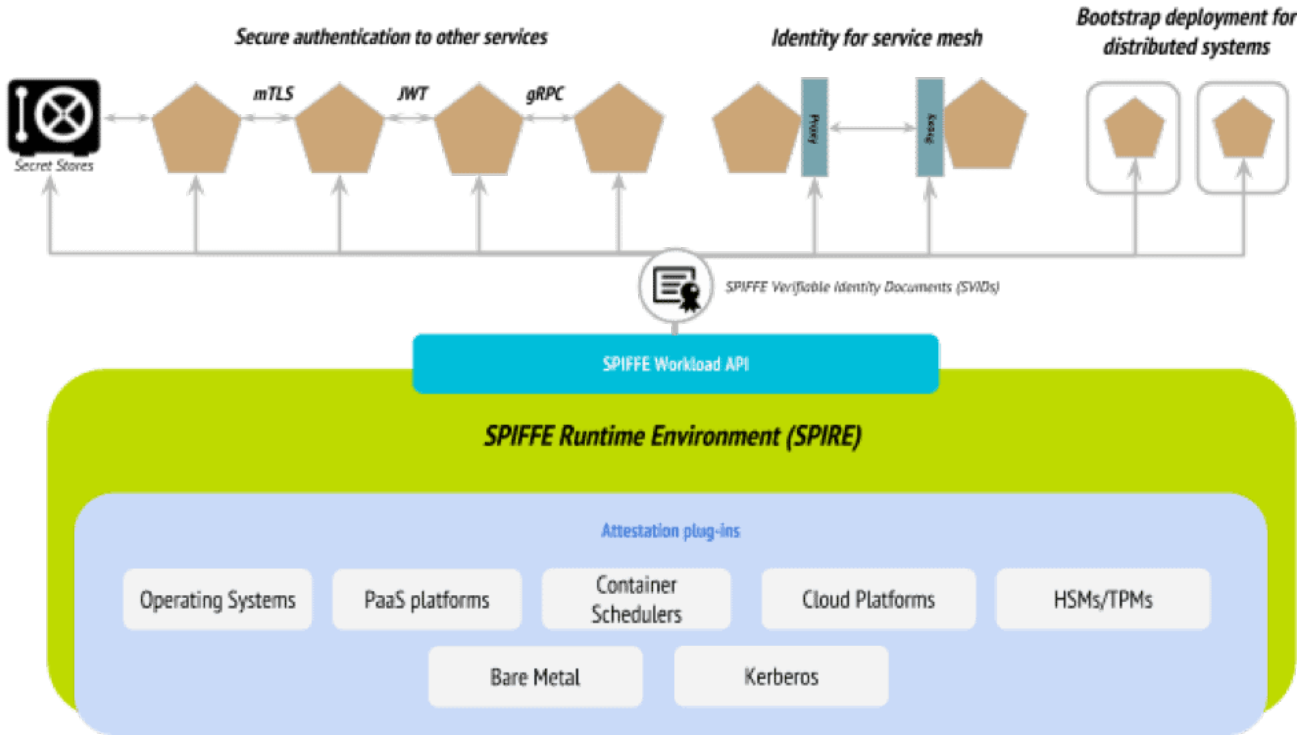
SPIFFE Overview

SPIFFE Verifiable Identity Document

spiffe://acme.com/billing/payments

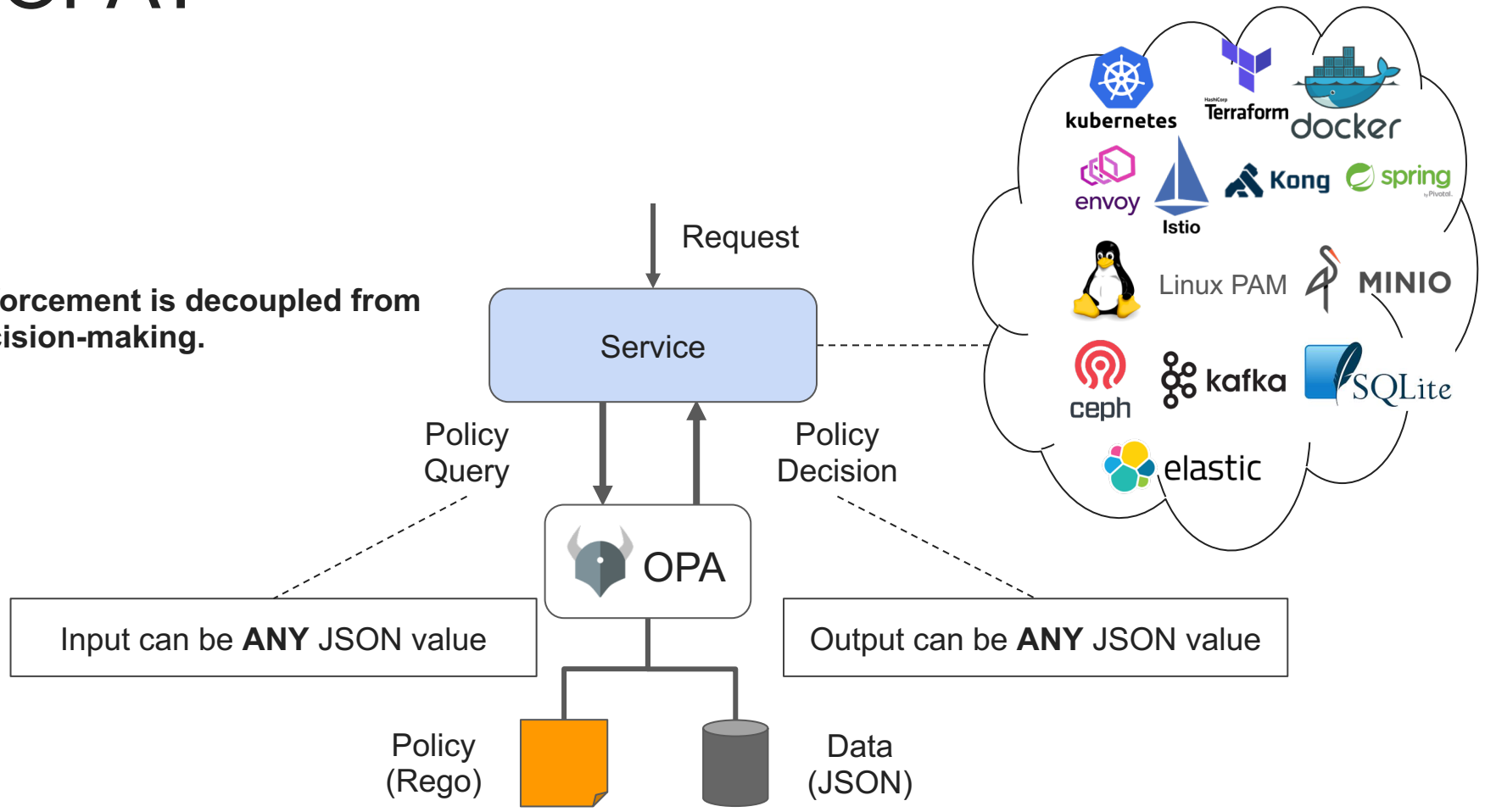
Typically short-lived

Today only one form of SVID (X509-SVID).
Other document types under consideration
(including JWT-SVID)



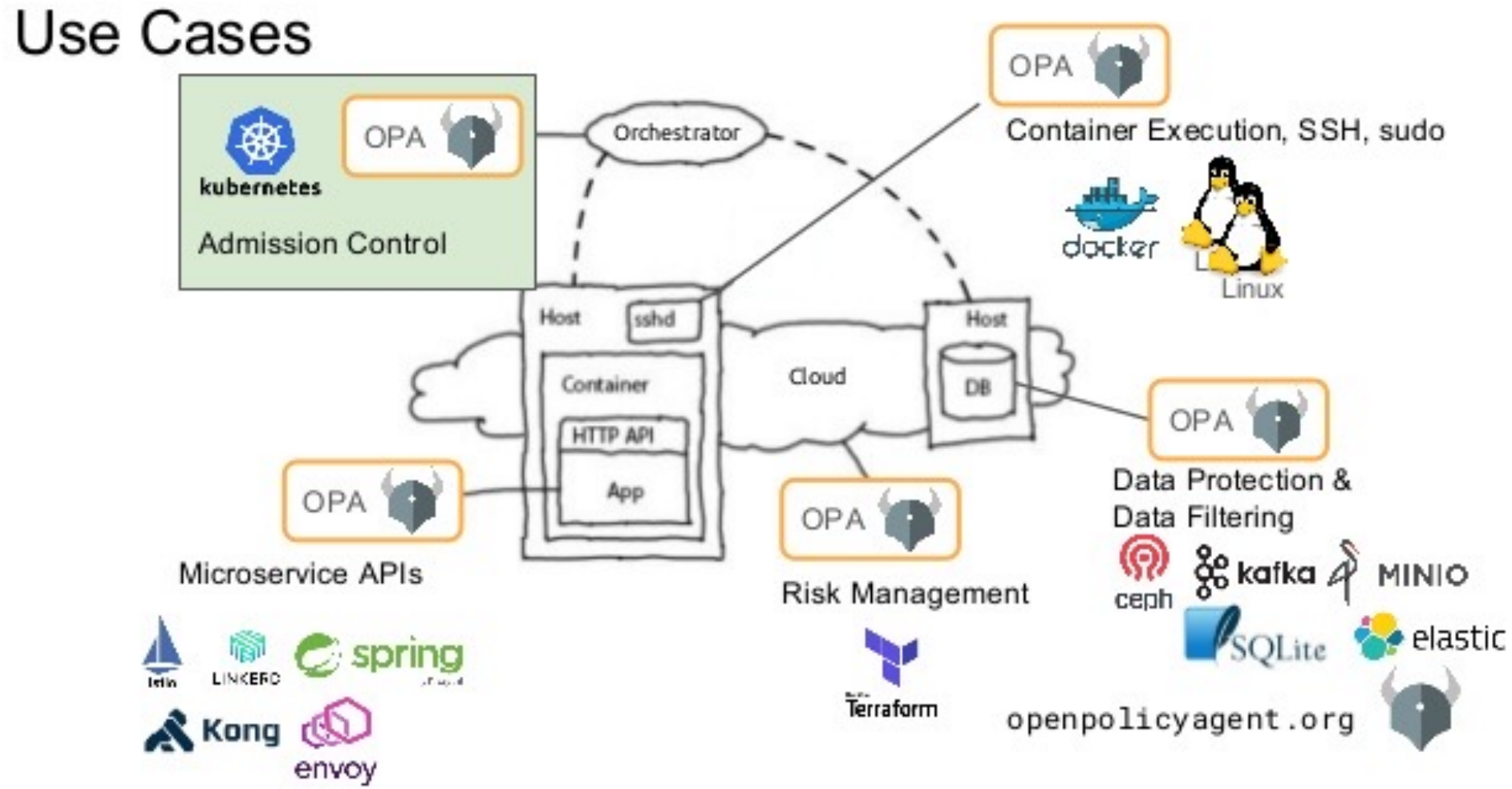
What is OPA?

Enforcement is decoupled from decision-making.

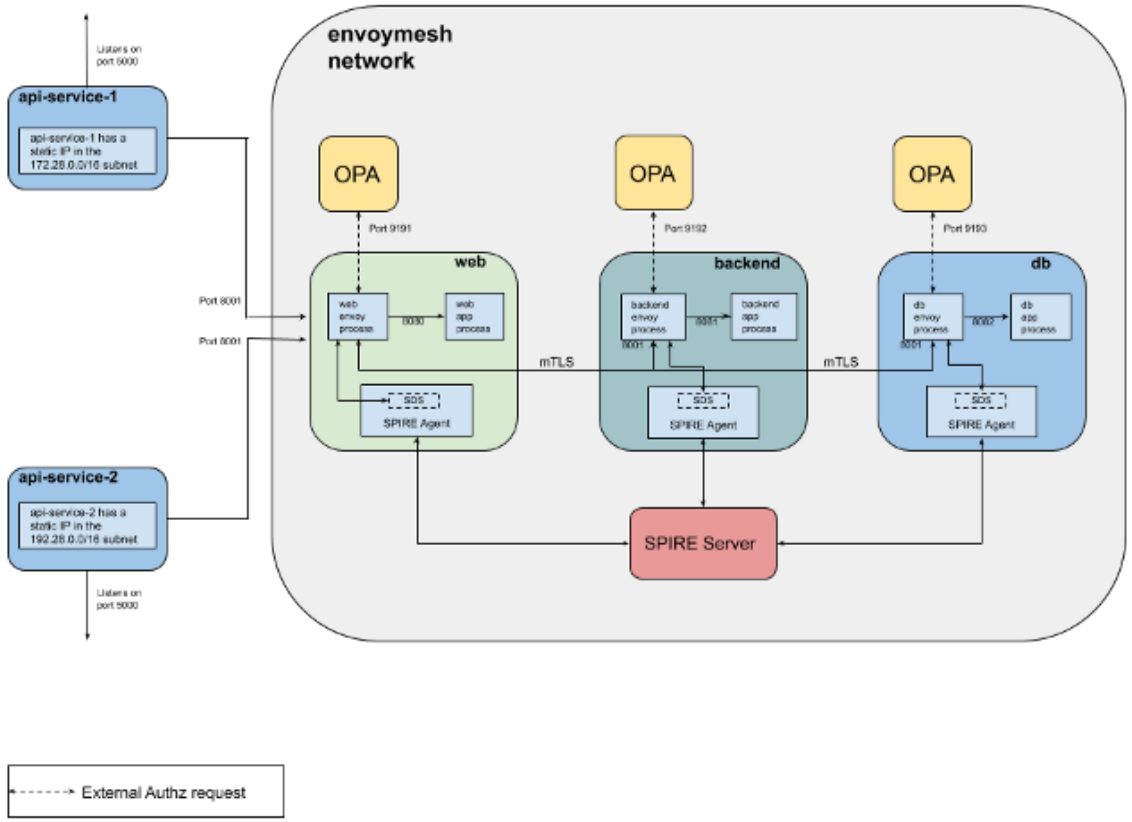


https://www.openpolicyagent.org/

OPA Overview



Example – Istio + SPIFFE + OPA




```
package envoy.authz

import input.attributes.request.http as http_request
import input.attributes.source.address as source_address

default allow = false

# allow Backend service to access DB service
allow {
    http_request.path == "/good/db"
    http_request.method == "GET"
    svc_spiffe_id == "spiffe://domain.test/backend-server"
}

svc_spiffe_id = client_id {
    [_, _, uri_type_san] := split(http_request.headers["x-forwarded-client-cert"], ";")
    [_, client_id] := split(uri_type_san, "=")
}
```

Compliance as Code

Compliance as Code

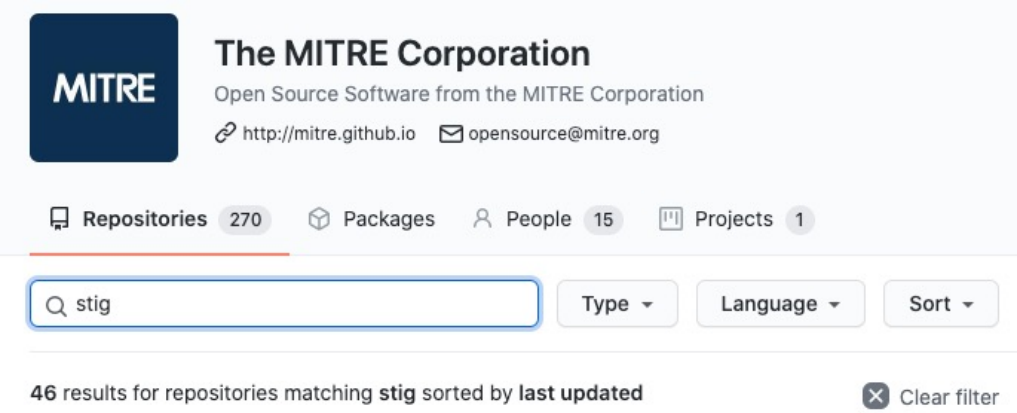
https://www.open-scap.org/

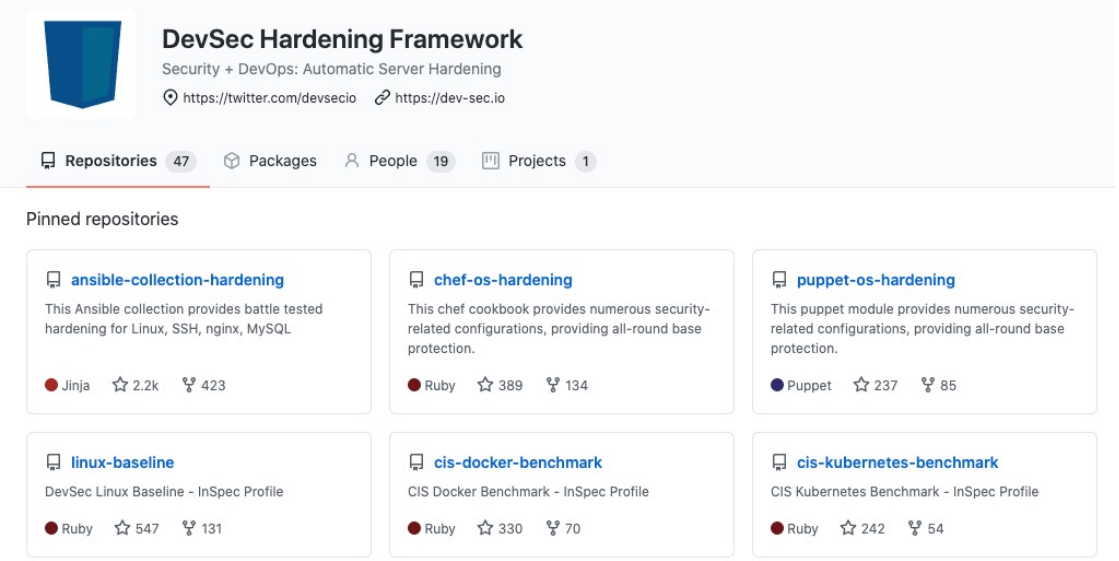
- Set of open-source tools for security compliance and vulnerability assessment
  - Security Content Automation Protocol (SCAP) is a framework that supports automated configuration, vulnerability and patch checking, technical control compliance activities, and security measurement
  - SCAP standard includes:
    - Extensible Configuration Checklist Description Format (XCCDF)
    - Open Vulnerability and Assessment Language (OVAL)
    - DataStream
    - Asset Reporting Format (ARF)
    - Common Platform Enumeration (CPE)
    - Common Vulnerabilities and Exposures (CVE)
    - Common Weakness Enumeration (CWE)

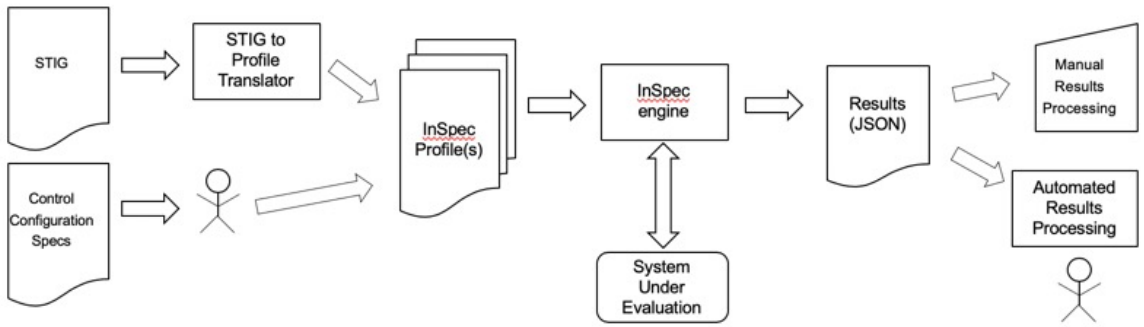
https://inspec.io/

- Open-source testing framework with human- and machine-readable language for specifying compliance, security and policy requirements
- Uses Infrastructure as Code principles to keep compliance in Source Code Management (SCM)
- Tests can be run locally, remotely or as part of CI/CD pipelines for continuous compliance
- Highly extensible and support for large ecosystem of software

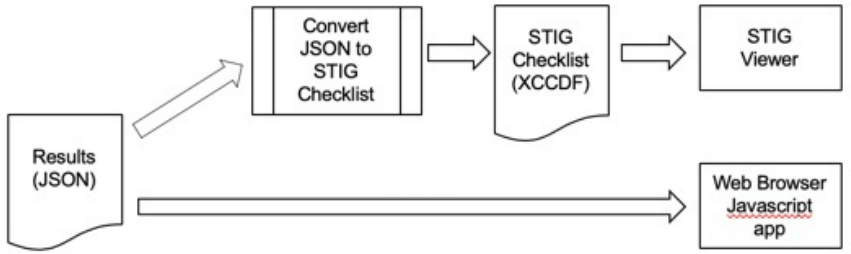
Inspec





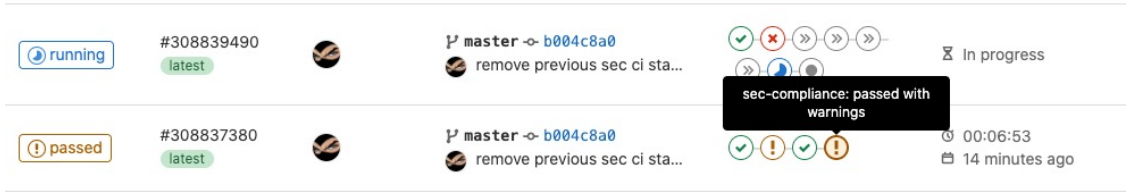


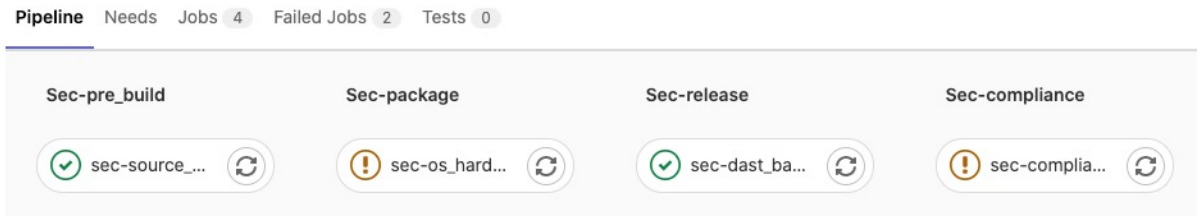
Automating Security Validation Using InSpec



Processing InSpec Results

Example – DevSecOps + Inspec





sec-package.gitlab-ci.yml 760 Bytes — Edit | Web IDE

```yaml
services:
  - docker:dind

sec-os_hardening:
  stage: sec-package
  image: ansible/galaxy
  before_script:
    - mkdir -p ~/.ssh
    - echo "$DEPLOYMENT_SERVER_SSH_PRIVKEY" | tr -d '\r' > ~/.ssh/id_rsa
    - chmod 600 ~/.ssh/id_rsa
    - eval "$(ssh-agent -s)"
    - ssh-add ~/.ssh/id_rsa
    - echo -e "Host *\n\tStrictHostKeyChecking no\n\n" > ~/.ssh/config
  script:
    - echo "[prod]" >> inventory.ini
    - echo "$DEPLOYMENT_SERVER" >> inventory.ini
    - export ANSIBLE_STDOUT_CALLBACK=json
    - ansible-galaxy install dev-sec.os-hardening
    - ansible-playbook -i inventory.ini ansible-hardening.yml > sec-os_hardening-results.json
  artifacts:
    paths: [sec-os_hardening-results.json]
    when: always
    expire_in: one week
  allow_failure: true
```

sec-compliance.gitlab-ci.yml 694 Bytes — Edit | Web IDE | Lock | Replace | Delete

```yaml
services:
  - docker:dind

sec-compliance:
  stage: sec-compliance
  image:
    name: chef/inspec
  only:
    - "master"
  environment: production
  before_script:
    - mkdir -p ~/.ssh
    - echo "$DEPLOYMENT_SERVER_SSH_PRIVKEY" | tr -d '\r' > ~/.ssh/id_rsa
    - chmod 600 ~/.ssh/id_rsa
    - eval "$(ssh-agent -s)"
    - ssh-add ~/.ssh/id_rsa
    - echo -e "Host *\n\tStrictHostKeyChecking no\n\n" > ~/.ssh/config
  script:
    - inspec exec https://github.com/dev-sec/linux-baseline -t ssh://root@$DEPLOYMENT_SERVER -i /id_rsa --chef-license accept --reporter json:/opt/sec-
  artifacts:
    paths: [sec-compliance-results.json]
    when: always
  allow_failure: true
```

Challenges

Challenges

Handling exceptions

- Use of third-party software
  - How should vendor software be handled when security and compliance issues are found (e.g. break the build process?)
- What mechanisms to use for alert management from security pipeline processes?
  - Multiple integration scenarios such as Jira for bug/defect tracking and DefectDojo for security violation tracking

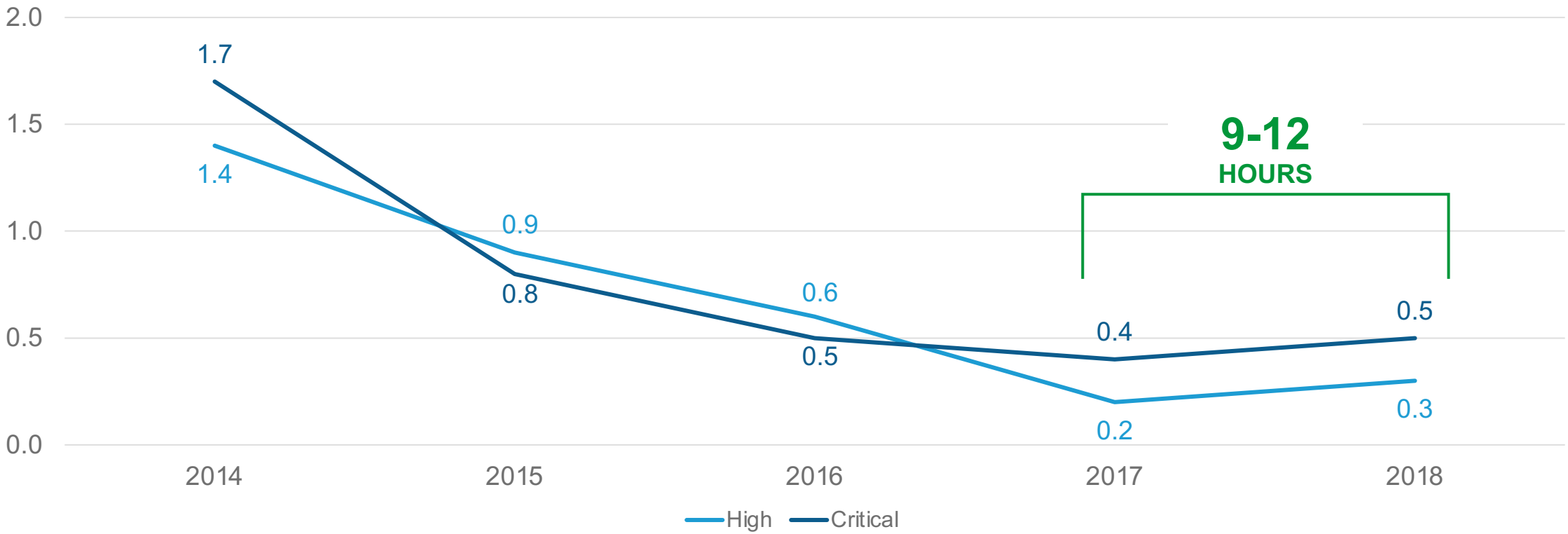
Organisational Culture

- Moving to a DevSecOps way of working requires significant work
  - People are almost always the hardest to change (DevSecOps involves People, Process and Technology)

Why now?

Protecting against Abuse of Functionality

Average days between “HIGH” AND “CRITICAL” CVEs released



Protecting against Abuse of Intent

The Automated Threat Handbook

Web Applications

The Automated Threat Handbook provides actionable information and resources to help defend against automated threats to web applications.

OAT-020 Account Aggregation
OAT-019 Account Creation
OAT-003 Ad Fraud
OAT-009 CAPTCHA Defeat
OAT-010 Card Cracking
OAT-001 Carding
OAT-012 Cashing Out
OAT-007 Credential Cracking
OAT-008 Credential Stuffing
OAT-021 Denial of Inventory
OAT-015 Denial of Service
OAT-006 Expediting
OAT-004 Fingerprinting
OAT-018 Footprinting
OAT-005 Scalping
OAT-011 Scraping
OAT-016 Skewing
OAT-013 Sniping
OAT-017 Spamming
OAT-002 Token Cracking
OAT-014 Vulnerability Scanning

Further Information

References

- NIST DevSecOps
- NIST 800-24A Building Secure Microservices-based Applications Using Service-Mesh Architecture
- NIST 800-24B Attribute-based Access Control for Microservices-based Applications using a Service Mesh
- OWASP DevSecOps Maturity Model

Technical

- DoD Enterprise DevSecOps Initiative
- Security Hardening and Baseline profiles
- MITRE STIG Inspec profiles

Slides available at

https://oi.shain.io/presentations