Securing Apps in the Open-By-Default Cloud

Winston Howes and Michael Wozniak
Who are we?

Michael Wozniak
Infrastructure Security

Winston Howes
Application Security
Welcome to the Cloud
Welcome to the Cloud

EC2  EKS  GCE  GKE  App Engine
"After deploying the application, you need to expose it to the Internet so that users can access it."
- GKE Quickstart
Constraints

● Networking
  ○ Not possible to have one large internal only network
  ○ Limited enforcement options provided by AWS/GCP
  ○ Services like App Engine must be exposed directly to the Internet

● Central Management
  ○ Lack of central CI/CD Pipeline
  ○ Wide variety of technologies
Development Lifecycle

- It's unclear when security should review an app.
Development Lifecycle

- It’s unclear when security should review an app.

New app created

You’re on the internet 🎉

Hello 🌍
Development Lifecycle

- It’s unclear when security should review an app.

**New app created**
- You’re on the internet 🌟
- Hello 🌍

**Ready to launch**
- You’re encouraged to file a security review.
Development Lifecycle

- It’s unclear when security should review an app.

**New app created**
- You’re on the internet 🎉
- Hello 🌍

**Ready to launch**
- You’re encouraged to file a security review.

**Going Steady**
- The app has a bunch of new features.
It’s unclear when security should review an app.

- **New app created**
  - You’re on the internet 🌐
  - Hello 🌍

- **Ready to launch**
  - You’re encouraged to file a security review.

- **Going Steady**
  - The app has a bunch of new features.

---

**Pre-Launch**

**Post-Launch**
Considered Gating Approaches

1. Enabling Billing Post-Review
2. Implement AuthN & AuthZ controls on individual services
3. Firewalls
4. Google’s Identity Aware Proxy
Considered Gating Approaches

1. Enabling Billing Post-Review
2. Implement AuthN & AuthZ controls on individual services
3. Firewalls
4. Google’s Identity Aware Proxy

*Denied*
Considered Gating Approaches

1. Enabling Billing Post-Review
2. Implement AuthN & AuthZ controls on individual services
3. Firewalls
4. Google’s Identity Aware Proxy

Limited Scalability
Considered Gating Approaches

1. Enabling Billing Post-Review
2. Implement AuthN & AuthZ controls on individual services
3. Firewalls
4. Google’s Identity Aware Proxy

Limited Granularity
Considered Gating Approaches

1. Enabling Billing Post Review
2. Implement AuthN & AuthZ controls on individual services
3. Firewalls
4. Google’s Identity Aware Proxy

Not Automatable
Goals

- **Flexibility**: Minimum opinions about development environments and cloud feature use*
- **Scalability**: No need for developer instrumentation
- **Granularity**: By default all services are gated with granular authN and authZ
- **Automatability**: Reduce operational costs

*if developers want high QPS or to receive user traffic, there will be necessary changes
Laying the Groundwork: Primitives

1. Network Control
2. Service Inventory
Laying the Groundwork: Primitives

Solution: Central service that enables billing and gives the security team network management access and inventories services
Development Lifecycle

- It’s unclear when security should review an app.

**Pre-Launch**
- New app created
  - You’re on the internet 🎉
  - Hello 🌍

**Post-Launch**
- Ready to launch
  - You’re encouraged to file a security review.

- Going Steady
  - The app has a bunch of new features.
Development Lifecycle

- It's unclear when security should review an app.

**New app created**
- You're on the internet 🎉
- Hello 🌍

**Ready to launch**
- You're required to file a security review.

**Going Steady**
- The app has a bunch of new features.

*UnManaged* | *Managed*
UnManaged Services

1. New Services in Development
2. Internal Tools

Treated identically by Security
UnManaged Services: Primitives

1. Firewall Manager
2. Stateless AuthN/Z Proxy
Firewall Manager

1. Import every service from our central inventory
2. Set base level firewall rules on every service
   a. App Engine: Only allow requests from our stateless proxy
   b. Other: Only allow requests from our SSH proxy
3. Revert non-Security approved modifications to the firewall rules
Firewall Manager Architecture

Service Inventory \(\xrightarrow{\text{New Service}}\) Firewall Manager (Source of Truth) \(\xrightarrow{\text{Synchronize Firewall Rules}}\) Fleet

Update Rules
Stateless AuthN/Z Proxy

- Support multiple forms of AuthN
  - Service-to-service
  - User-to-service
- Easy integration
  - App Engine: zero setup
  - Other: config change to stateless proxy
- Easily offboard users
  - Periodic syncs with ACL source of truth
- Reliable
Stateless AuthN/Z Proxy Architecture

1. Configuration
2. Authentication and Authorization
3. Proxying Requests
Stateless AuthN/Z Proxy Architecture: Configuration

- Source Control
  - Update Configuration
  - Poll every 10 minutes
  - Upload Configuration to GCS

- GCS
  - Poll every 10 minutes
  - Upload ACLs to GCS

- Proxy

- ACL Service
  - Poll every 10 minutes

BlackHat 2019
Stateless AuthN/Z Proxy Architecture:

User tries to access service behind proxy
Stateless AuthN/Z Proxy Architecture: AuthN/Z

Proxy can’t authenticate the user. Redirects to Jump Point

Browser

Proxy

User reaches Google’s Identity Aware Proxy (IAP) and signs in

IAP

Jump Point
Stateless AuthN/Z Proxy Architecture: AuthN/Z

The Jump Point creates a ticket with the user’s Identity and redirects the user to the Proxy.
Stateless AuthN/Z Proxy Architecture:

User forwards the ticket to the proxy, which compares the identity against its ACLs and proxies the request.
Stateless AuthN/Z Proxy Architecture: AuthN/Z

User’s request reaches service

Browser

Proxy

IAP

Jump Point
Stateless AuthN/Z Proxy Architecture:
AuthN/Z

IAP + Jump Point can be generalized as a SSO provider
Stateless AuthN/Z Proxy Architecture: Proxying

Central Proxy

App Engine Service

Leaf Proxy

Service A

Service B

VPC Peering

Inbound Request
Stateless AuthN/Z Proxy Challenges

1. Higher latency, particularly for App Engine
2. Double Billing - twice the egress
Managed Services
Managed Services: Goals

1. Low Latency
2. Cheap
3. Granular Auth N/Z
4. Visibility
Managed Services: Components

1. API Gateway
2. Service Mesh
3. Configuration Controller
4. Service Sidecar
Managed Services: API Gateway

1. Envoy as a front-proxy
2. Single entry point for external traffic
3. Set of audited AuthN filters
4. Centrally managed
Managed Services: Service Mesh

1. Centrally managed and visible routing
2. Envoy provides
   a. Authentication
   b. Encryption
   c. Metrics
3. Not routable from Internet except via API Gateway
Managed Services: Configuration Controller

1. Central component to manage routes
2. Routes need to be approved by owners
3. Authentication included automatically based on configuration state
Managed Services: Service Sidecar

1. Envoy as a sidecar
2. Connects to CA to establish identity
3. Fetches config from central configuration service
4. Authenticates all incoming traffic
5. Exposes a port locally for service egress
Managed Services: Challenges

1. Onboarding: configuration changes require approval
2. Noisy Neighbors: single account/VPC means that cloud quotas are shared by all services
3. Central Point of Failure
What about the non-migrated services?
Introspection
Introspection Library

- Easy to integrate
  - Single line of code
  - Supports all service frameworks
- Gathers security-critical information
  - Routes
  - Auth Controls (Filters, decorators, annotations, etc.)
  - Packages
  - Service Metadata
- Runs on instance startup
- Triggers high signal alerts
Introspection Architecture

Provision Bucket for Service A

Periodically aggregate bucket data

GCS/S3

Write data to bucket on instance startup

Service A

Alert Platform

Trigger any alerts

Billing service pings Introspection service about new Service A

Billing Enabler

Introspection backend
Core Infrastructure

- **Firewall Manager**: Gate services by default
- **Stateless Proxy**: Allow authenticated access to services
- **API Gateway & Service Mesh**: Production environment to run services with controls
- **Introspection**: Understand service state
Revisiting Goals

- **Flexibility**: Minimum opinions about development environments and cloud feature use*
- **Scalability**: No need for developer instrumentation
- **Granularity**: By default all services are gated with granular authN and authZ
- **Automatability**: Reduce operational costs

*if developers want high QPS or to receive user traffic, there will be necessary changes
Order of Operations
Step 1: Lay the Foundation

- Create a central hook that provides ways to make future changes
- Inventory all new services
Step 2: Start Simple

- Gate services in development to just corporate IPs
- Build Firewall Manager
Step 3: Add Granularity

- Transition from IP-based auth to service identities
- Build Stateless AuthN/Z Proxy
- As things transition to production perform manual review
Step 4: Understand Production

- Learn how your services change over time
- Build out an Introspection library
Step 5: Provide Robust Controls in Production

● Build out a central gateway and service mesh
● Migrate existing services
Lessons Learned

Knowledge is POWER
Security is Engineering
Gain a central hook into your fleet early
Visibility before enforcement
Make your security posture something you can reason about - no black boxes
Offer other engineering teams a carrot
Thank you