# ARCHITECTING SECURE APPLICATIONS IN AZURE ASSUMING BREACH

Mike Douglas – VP Digital Consulting - Engineering

LUNAVI

# GOALS

- Understand Zero Trust Security and Advantages over Traditional Perimeter security
- Answer why do we want to Assume Breach?
- Learn how to Architect Applications using Assume Breach mentality

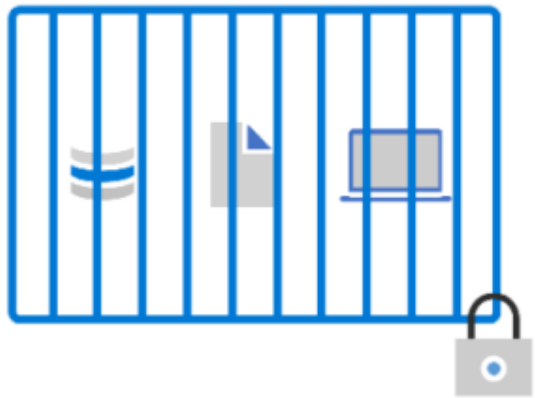
# ABOUT ME

- Mike Douglas
- Solution Consultant and VP Digital Consulting – Engineering at Lunavi
- Microsoft MVP – Developer Technologies – DevOps
- Organizer of Omaha DevOps Meetup
- Competitive Robotics Club Coordinator for 7th – 8th Graders
- @mikedouglasdev on twitter

# ZERO TRUST

# TRADITIONAL APPROACH

- Restrict access to corporate owned network assets
- Few network security perimeter and the open, flat networks
- Minimal threat protection and static traffic filtering
- Unencrypted internal traffic
- Role of creating secure applications was the network team’s role



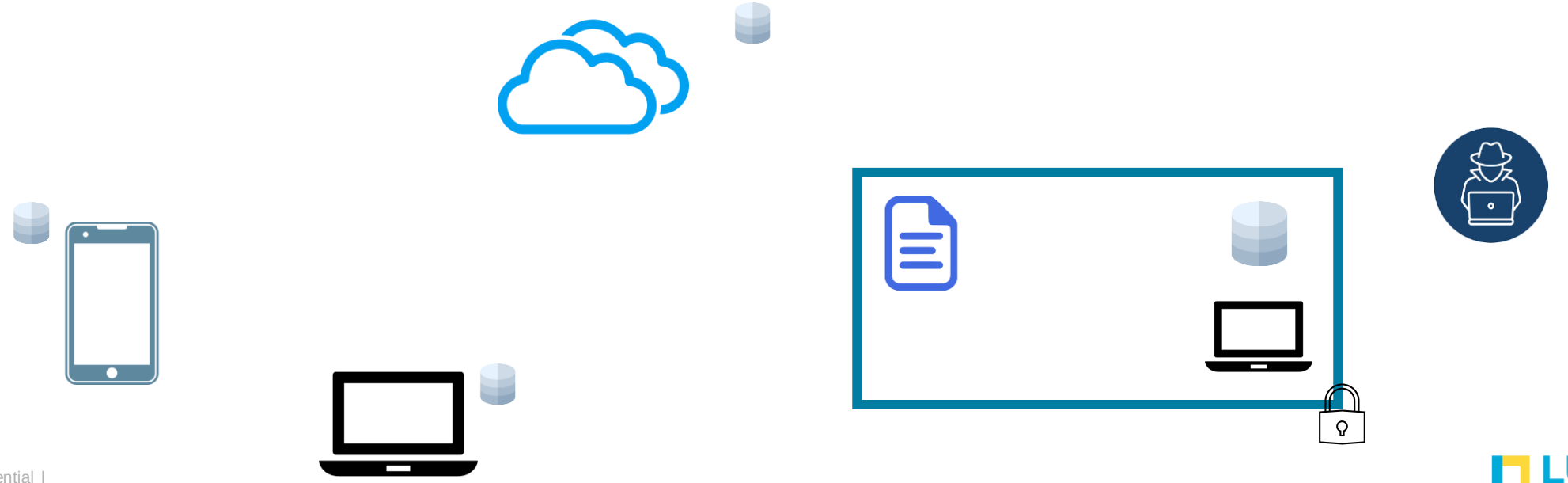
**Classic Approach**
Restrict everything to a ‘secure’ network



LUNAVI

# TRADITIONAL APPROACH

- Challenges
  - Today’s environments are made up of distributed assets and remote workers
  - Environments are hybrid across cloud and on-premises
  - Public Sites and Endpoints are not contained within a DMZ anymore
  - Once within the network, there is little security





LUNAVI

# ZERO TRUST

- Secure assets where they are at
- Prefer identity as primary control over network security technology
- Network technology and the security perimeter tactic are still present, but not dominant approach
- Security is both a security team and development team responsibility
- 3 Aspect
  - Verify Explicitly
  - Use Least Privilege Access
  - Assume Breach

**Zero Trust**
Protect assets anywhere with central policy



LUNAVI

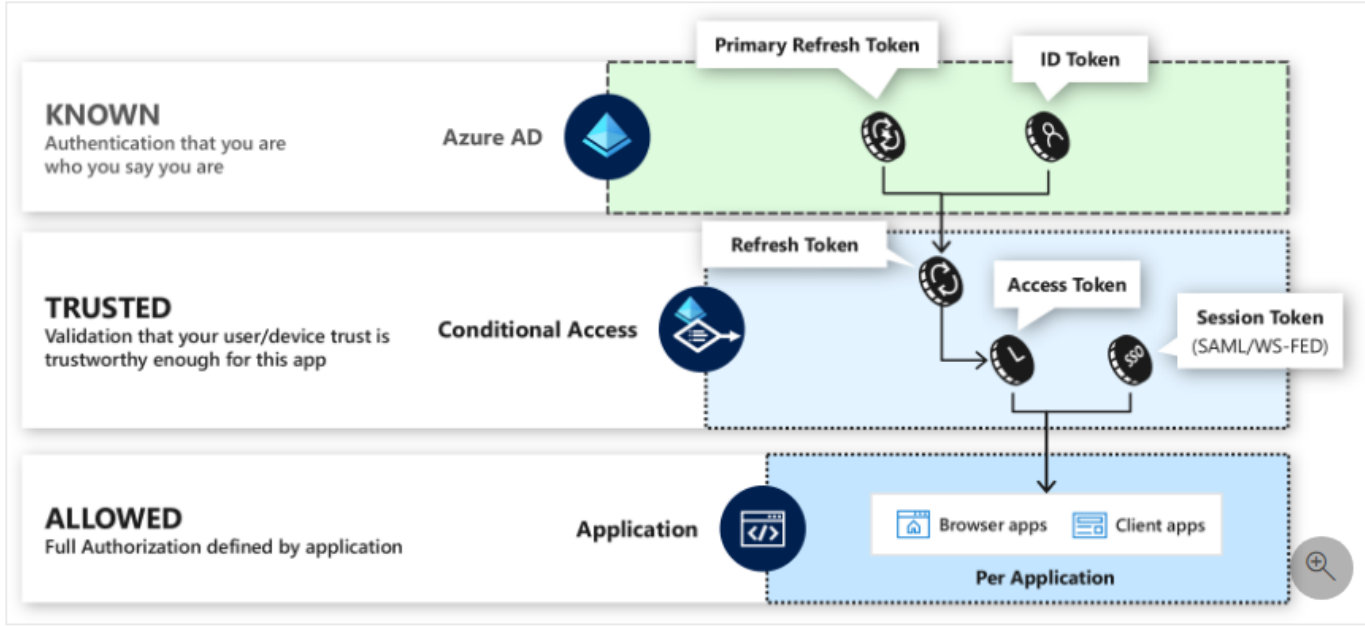
# ZERO TRUST AND APPLICATION ARCHITECTURE

# VERIFY EXPLICITLY

- Always Authenticate and Authorizes Based on All Available Data Points
  - User Identity
  - Location
  - Device health
  - Service or Workload
  - Data Classification
  - Anomalies

# VERIFY EXPLICITLY

- Shift from static two-step process of authentication/authorization, to dynamic 3 step process
  - **Known** - Auth ensures you are who you say you are
  - **Trusted** - Validation that the user or device is trustworthy enough to access the resource
  - **Allowed** - Granting of specific rights and privileges for the application, service, or data

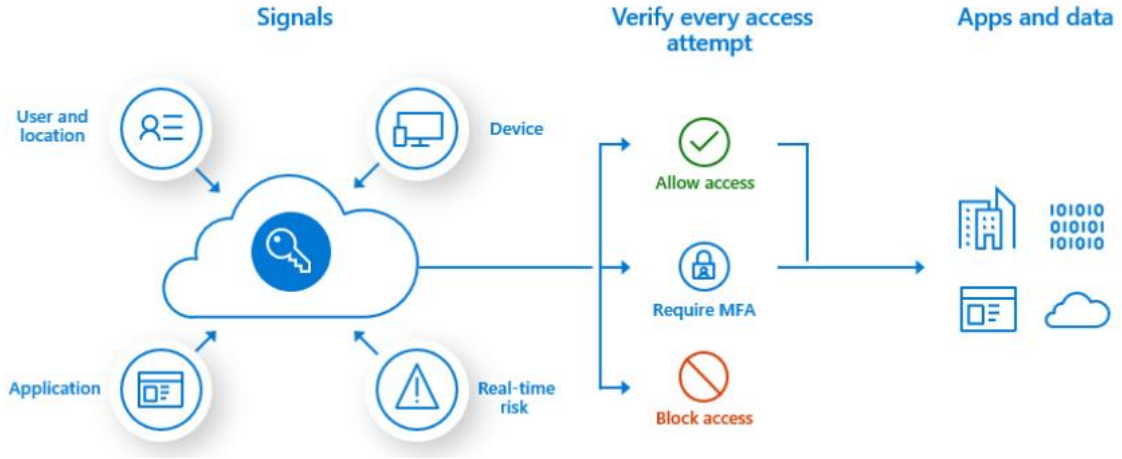





LUNAVI

# VERIFY EXPLICITLY

- Best Practices
  - MSAL – OpenID Connect Client library on various platforms to authenticate and connect to secured resources
  - Continuous Access Evaluation (CAE)
    - Client-Side claim challenge - can reject a token even if it hasn't expired
    - B2C
      - Conditional Access Policies
      - Identity Proofing
      - MFA

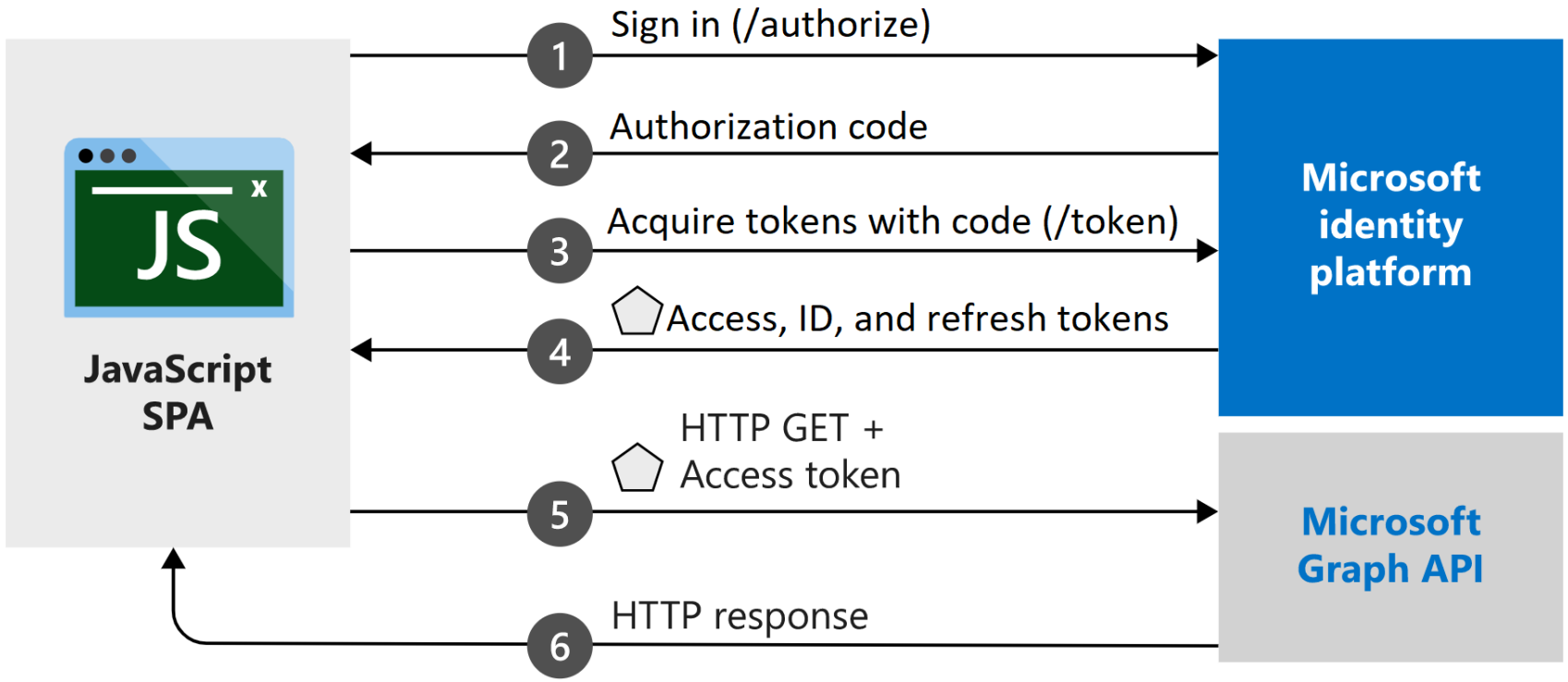





LUNAVI

# VERIFY EXPLICITLY

- Use the correct authentication flow
  - Web (server) - confidential client flow
  - SPA/Mobile – Authorize Code Flow with PKCE (use MSAL v2)
    - ~~Implicit Grant Flow~~






LUNAVI

# USE LEAST PRIVILEGE ACCESS

- Limit user access – JEA / JIT
- Require MFA for admin access

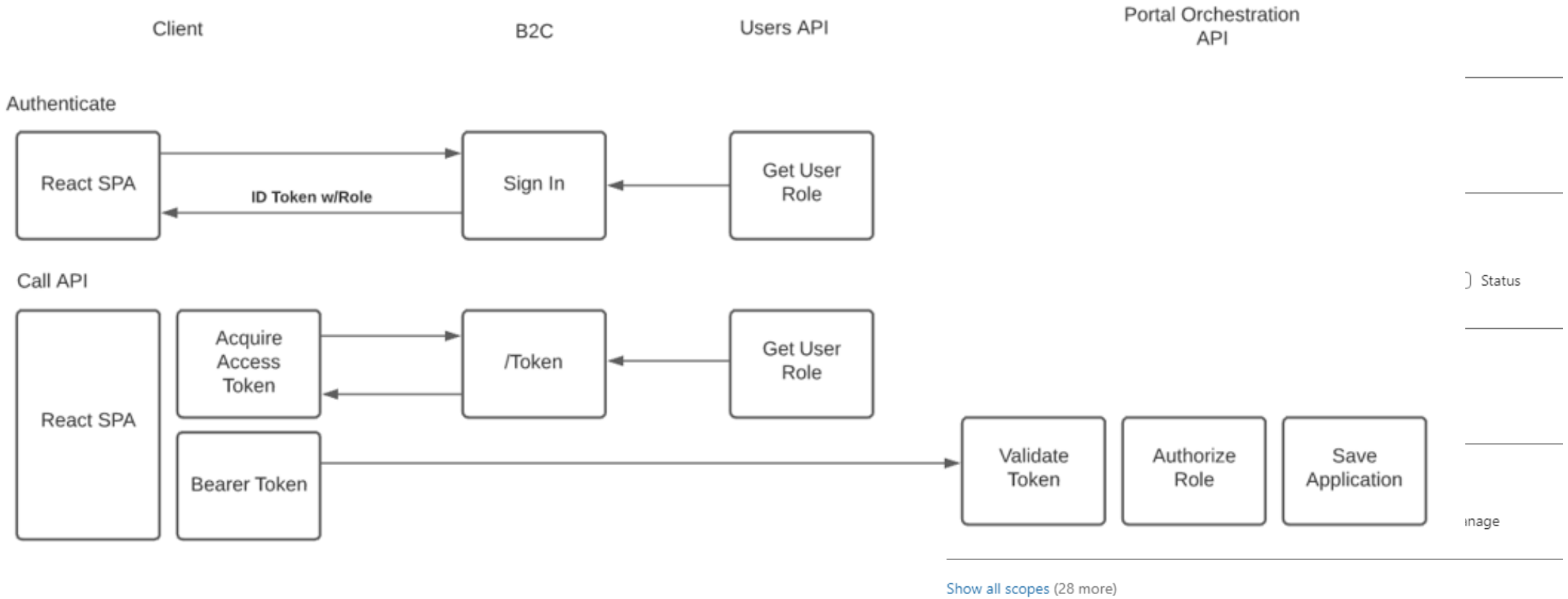
Azure Privilege Identity Management (PIM)




LUNAVI

# USE LEAST PRIVILEGE ACCESS

- Securing APIs




Scopes

Status

Show all scopes (28 more)



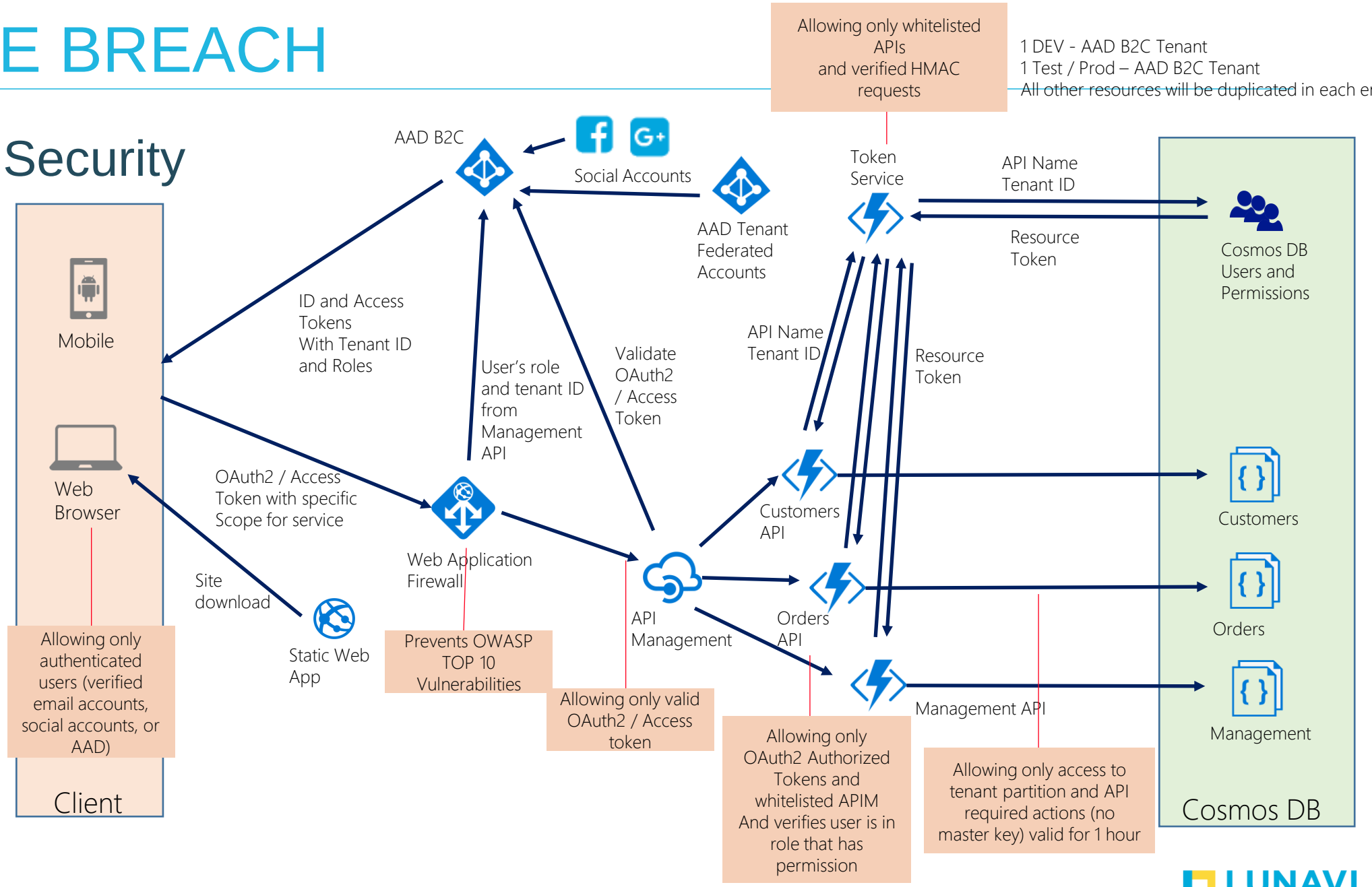
LUNAVI

# ASSUME BREACH

- Minimize blast radius for breaches prevent lateral movement by segmenting access by network, user, devices and application awareness.
- Verify all sessions are encrypted end to end
- Use analytics to get visibility, drive threat detection, and improve defenses

# ASSUME BREACH

- Layered Security

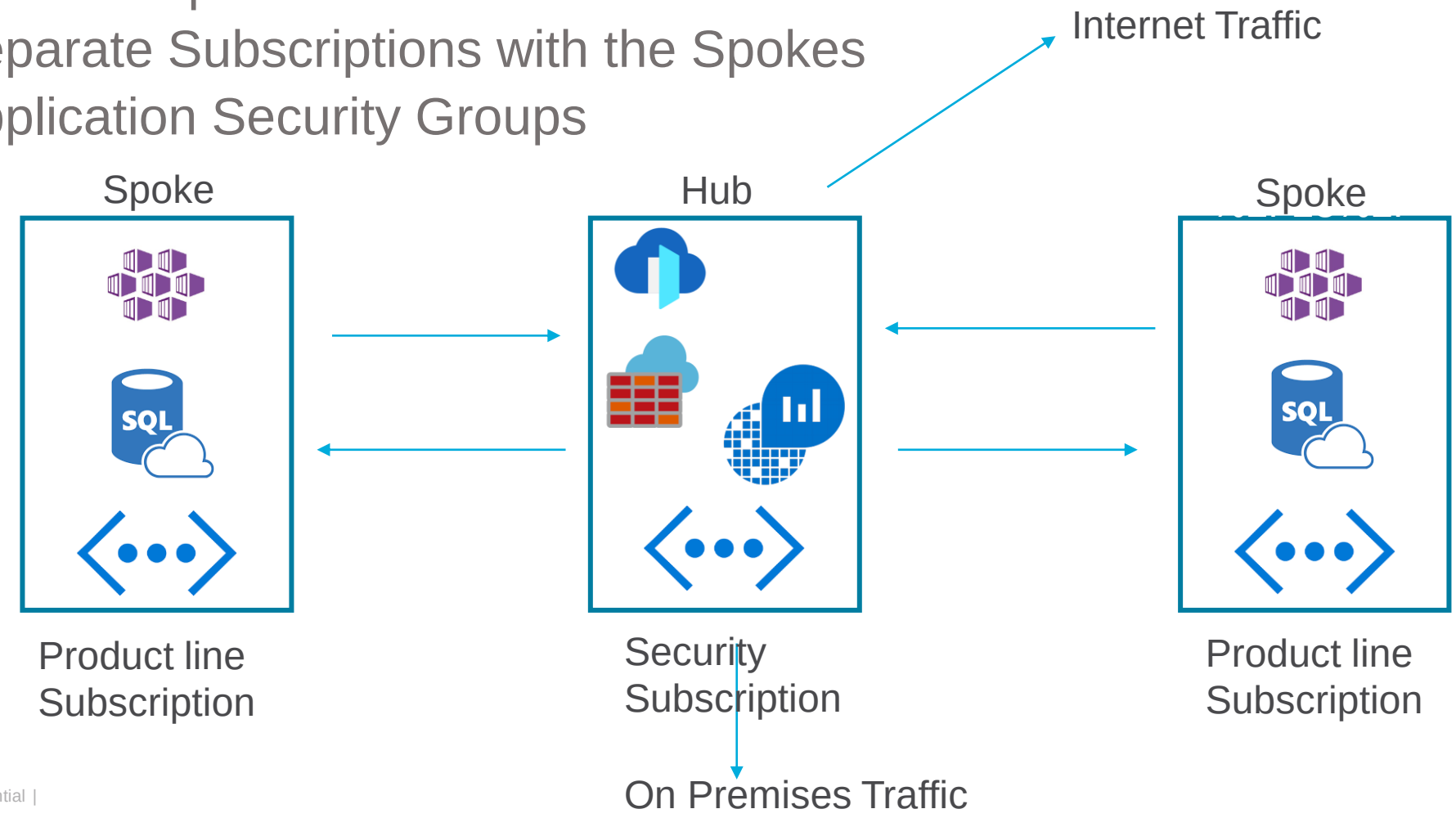





LUNAVI

# ASSUME BREACH

- Segmentation
  - Hub and Spoke
  - Separate Subscriptions with the Spokes
  - Application Security Groups

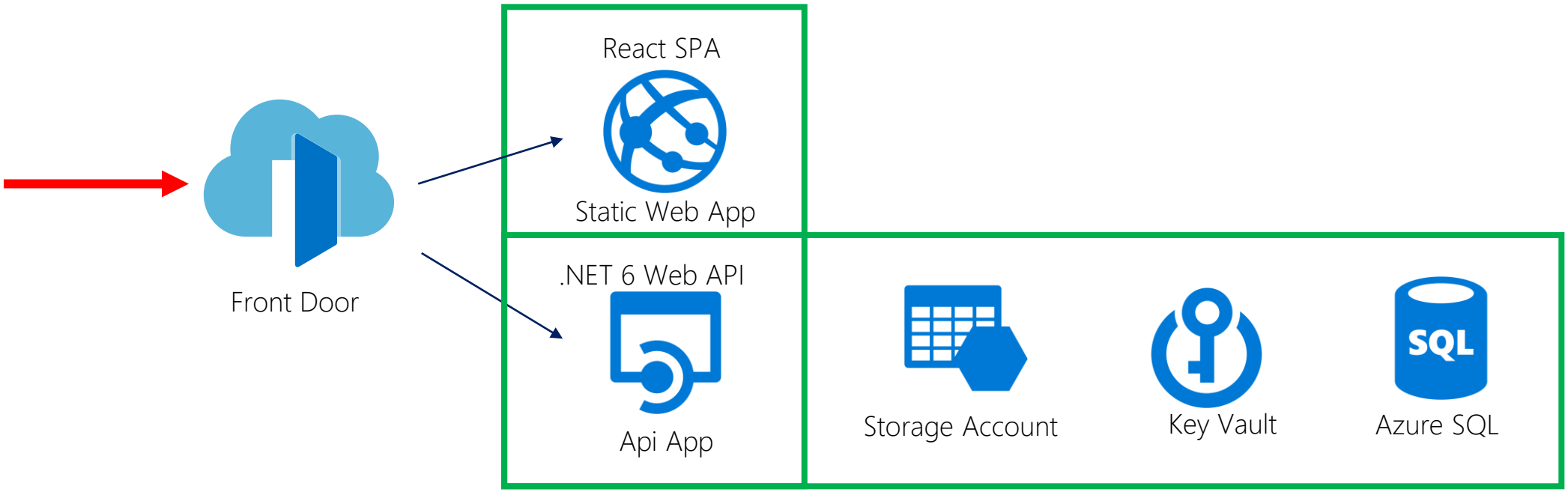





LUNAVI

# ASSUME BREACH

- Infrastructure Architecture




Protect all resources through access restrictions and proper auth using Managed Identity when possible



LUNAVI

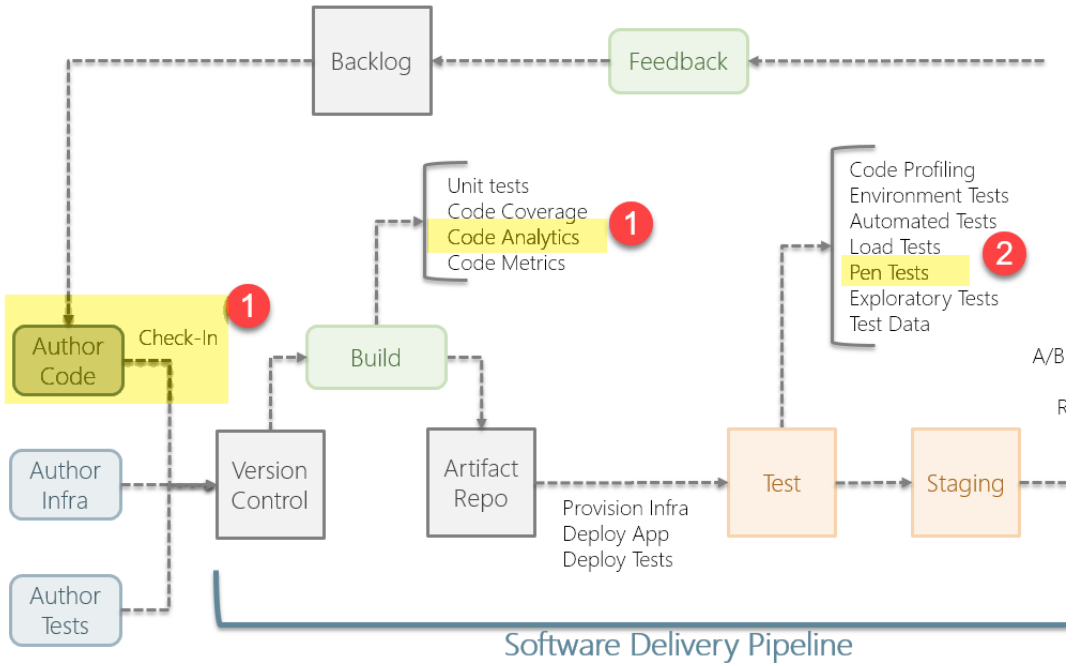
# ASSUME BREACH - PIPELINE

## 1. Code Review and Static Code Analysis

- Use the code review in pull request to require other developers review code being merged to central repository
- Run against source code
- On developers machines and on build server at check in
- VS Static Code Analysis include security checks and there are other tools like Checkmarx and MS Devskim

## 2. Penetration Testing

- Dynamic scanning of the application using OWASP ZAP will be done as part of the deployment to test. This will be a "smoke test" like security scan. A full security scan will be run each night.




Nightly Dynamic security scan of website



LUNAVI

# ASSUME BREACH - MONITORING