How Do We Microsegment Container Environments?
Overview

1. Container Networking 101
2. Dynamic Environments vs. Policy
3. Lateral Movement vs. Policy Complexity
4. Rethink: App-Centric Policy
Container Networking 101

- Pluggable Architecture
  - NATs
  - Bridging
  - Overlay Networks
- Highly Customizable
- Wide Variability Across Environments
Container Networking 101

NIC #1

10.0.1.5

loopback

net

eth0

pod

update.sh

nginx

loopback

net

sshda
Container Networking 101 – Docker Classic

10.x.y.z to 10.0.1.5
port 8080

10.0.1.5

loopback

docker0

et0

172.17.0.2

net

10.x.y.z to
172.17.0.2
port 80

DNAT Happens Here

pod

update.sh

nginx

loopback

Cross Net Namespace Bridging
Applying Firewall Policy Is Hard!

- Context Dependent
- Complex, Variable Topology
- Heavy Dependence On Configuration
- Many Containers → Rules Explosion
Dynamic Environments vs. Policy
Dynamic Environments vs. Policy

- IT Isn’t Static Anymore
  - Dynamic Container Placement
  - Dynamically Scaling Infrastructure
- Environments Change Rapidly
  - Infrastructure-as-Code
  - Change-by-API-Call vs. Change-by-Walk-to-Rack
Dynamic Environments vs. Policy

ALLOW FROM 10.0.1.5 – 10.0.1.90 TO 10.1.8.8:3306
Dynamic Environments vs. Policy

- Policy Often Falls Behind
  - Infrastructure Changes: Milliseconds
  - Policy Changes: Hours
- We have Infrastructure-by-Algorithm
- We need Policy-by-Algorithm
Dynamic Environments vs. Policy

- Is Policy-By-Algorithm Sufficient?
Lateral Movement vs. Policy Complexity
From Our Last Example:

- What Did The Firewall Rule Permit?
Lateral Movement vs. Policy Complexity

- Issue: Over-Permissiveness
- Containers Make This Worse

Uh Oh! Our Policy Was Supposed To Allow Java!

ALLOW FROM 10.0.1.5 – 10.0.1.10 TO 10.1.8.8:3306
Lateral Movement vs. Policy Complexity

- Can We Improve Specificity?
  - Overlay Networks Can, But…
  - Policies Become Unmanageable!
- Policy-By-Algorithm Is Not Sufficient
Rethink: App-Centric Policy
We *Think* In Terms Of Apps

We *Translate* To IPs/Ports

NIC #1

prod-app1
10.0.1.5

sshd

java

NIC #1

prod-db1
10.1.8.8

mysqld

“Allow Java On prod-app* To Talk To Mysql on prod-db**”

“ALLOW 10.0.1.5:* TO 10.1.8.8:3306 ...”
Is Translation Necessary?
No!

- Kernel Mandatory Access Control (MAC)
- Prove End-End Connectivity
Mandatory Access Control?

- E.g., Linux Security Modules, eBPF:

  ![Diagram showing sshd and java processes with kernel and NIC #1]
Rethink: App-Centric Policy

- Why MAC vs. Host Firewall?
  - Closer To Intent
  - Less Mental “Accounting”
  - Increased Specificity
Why MAC vs. Host Firewall?

- Fewer, Simpler, Specific Rules

ALLOW FROM 
{ my_app1, my_app2, my_app3 } 
ON HOSTS { ... } TO 
{ mysqld, postgres } 
ON HOSTS { ... }

vs.

ALLOW FROM 10.0.1.12 TO 10.0.5.8:3306
ALLOW FROM 10.0.1.12 TO 10.0.5.8:5432
ALLOW FROM 10.0.1.12 TO 10.0.5.14:3306
ALLOW FROM 10.0.1.12 TO 10.0.5.14:5432
ALLOW FROM 10.0.1.12 TO 10.0.5.16:3306
ALLOW FROM 10.0.1.12 TO 10.0.5.16:5432
ALLOW FROM 10.0.1.12 TO 10.0.5.24:3306
ALLOW FROM 10.0.1.12 TO 10.0.5.24:5432
ALLOW FROM 10.0.1.12 TO 10.0.5.88:3306
ALLOW FROM 10.0.1.12 TO 10.0.5.88:5432
Why MAC vs. Host Firewall?

- Very Hard to Spoof
- Can Cryptographically Fingerprint Apps
- Authorize Per-Connection (vs. Per-Packet)
Rethink: App-Centric Policy

• How To Prove End-To-End Connectivity?

  1. Build Accurate Topological Maps (Hard)

  2. Wrap All Traffic in TLS/DTLS (Expensive)
Strategies Are Complementary

- Different Failure Modes
- CPU Utilization vs. Security Posture

Ultimately, Both Are Needed
Closing Thoughts

● Containers Exacerbate Firewall Over-Permissiveness
● Dynamic Infrastructure → Real-time Firewall Updates
  ○ Realistically, Overly-Permissive Rules
● We Can Do Better!
  ○ App-Centric Policies
  ○ Distributed Enforcement
    ■ Preferably In-Kernel Using MAC
Thanks!

Follow-up Questions?
Please Feel Free To Contact Me:

tom@edgewise.net