Modern Service Networking with Consul Connect, Envoy, and Kubernetes
Nic Jackson
Developer Advocate
HashiCorp
Why?
Easy to bypass the perimeter by attacking code

Service vulnerable to remote code execution

perimeter firewall
“One of the key phases in most targeted attacks is what’s known as lateral movement. Attackers rarely luck out and manage to immediately compromise the computer.”

Symantec Internet Threat Report 2018
Example attack on vulnerable service

1. Remote code execution on random service
2. Scan for open ports and other services
3. Sniff HTTP traffic
4. Create fake request with stolen token
5. Transfer to hostile account

Order processing
Favorites service

HTTP with JWT token

Transaction processing
HTTPS

Banking provider
Hostile actor
Perimeter firewall
Two factors which allowed lateral movement

• Open network access

• Traffic between services are not encrypted
Problem 1:
Open network access
Traditional approach to security was a perimeter firewall
We need internal network isolation
Network segmentation

- network segment - frontend
- network segment - backend
Service Segmentation

- Service segment - A -> B
- Service segment - A -> C
- Network segment - frontend

A
A
A

B
B
B

C
C

D
D
Problem: Dynamic environments result in constantly changing IP addresses and ingress ports.
Definition of a dynamic environment

- Applications and infrastructure subject to frequent changes
- Subject to auto scaling or automated instance replacement
- Applications running in a scheduler like Kubernetes
Problems with network and service segmentation in dynamic environments

• Application deployment is disconnected from the network configuration

• Applications are scheduled in a modern scheduler
Application deployment is disconnected from the network configuration

Routing rules:
- 10.12.32.10 -> 10.13.12.1
- 10.12.32.12 -> 10.13.12.1

Firewall:
- B allow port 9912

Network segment - frontend:
- A 10.12.32.12
- A 10.12.32.10

Network segment - backend:
- B 10.13.12.1
Application deployment is disconnected from the network configuration

Routing rules:
- 10.12.32.10 -> 10.13.12.1
- 10.12.32.12 -> 10.13.12.1

Firewall:
- B allow port 9912

Applications are redeployed, routing rules must also change.
Applications are scheduled in a modern scheduler

desired rules:
192.168.1.1 -> 192.168.10.1
192.168.1.2 -> 192.168.10.1
192.168.1.4 -> 192.168.10.1

reality:
192.168.10.0/24 -> 192.168.10.1
Network / Service segmentation with intention based security

Routing rules:
- 10.12.32.10 -> 10.13.12.1
- 10.12.32.12 -> 10.13.12.1
- 10.12.32.12 -> 10.13.12.1
- 10.12.32.10 -> 10.13.12.23
- 10.12.32.12 -> 10.13.12.23
- 10.12.32.12 -> 10.13.12.23

Intentions:
- service a -> service b
Problem 2:
Traffic between services is not encrypted
Example attack on vulnerable service

1. Remote code execution on random service
   - perimeter firewall

Order processing
- HTTP with JWT token

2. Scan for open ports and other services
3. Sniff HTTP traffic

Transaction processing
- HTTPS

4. Create fake request with stolen token

Hostile actor
- Transfer to hostile account

Banking provider
- HTTPS

Favorites service

Perimeter firewall
"it takes on average 200 days, to detect a breach"
Implementing and managing certificates is hard
Reality, a service mesh can do all this for you

• You need to manage a Certificate Authority (CA)

• Have to distribute, and rotate certificates and keys

• Application code needs to be changed to handle TLS termination
What is a Service mesh
Service Mesh Architecture

Control Plane

Data Plane

pod

pod
Service Mesh Architecture

Control Plane

- Service to service communication policy
- Observability
- Service catalog
- CA and x509 certificate generation
- Configuration and proxy management
Service Mesh Architecture

Data Plane

Features:
- Authorization
- Request tracing
- Traffic shaping
- Load balancing
- Service discovery
- Circuit breaking
- Retry logic
Sidecar pattern

- Service binds to localhost (http)
- Sidecar accepts inbound (https)
- Service and sidecar run in isolated environment e.g. k8s pod
- 0 code changes
Consul Architecture
Consul Architecture
Service -> Consul Communication
Consul Architecture

Health Checks

Server (follower) ← LEADER FORWARDING → Server (leader) ← LEADER FORWARDING → Server (follower)

RPC TCP/8300

Client (healthy) ↖ HEALTH CHECK ↙ HEALTH CHECK Client (healthy) ↖ HEALTH CHECK ↙ HEALTH CHECK Client (healthy) ↖ HEALTH CHECK ↙ HEALTH CHECK Client (unhealthy)

C (healthy) ↖ HEALTH CHECK ↙ HEALTH CHECK C (healthy) ↖ HEALTH CHECK ↙ C (healthy) ↖ HEALTH CHECK ↙ C (healthy) ↖ HEALTH CHECK ↙ C (healthy)

C (unhealthy) ↖ HEALTH CHECK ↙ C (healthy) ↖ HEALTH CHECK ↙ C (healthy) ↖ C (healthy) ↖ C (healthy) ↖ C (healthy)
Consul Architecture
Certificate Authority - Built In

CA
Server (follower)

CA
Server (leader)

CA
Server (follower)
Consul Architecture
Certificate Authority - Vault
Consul Architecture
Certificate Generation & Rotation

Generate Key Pair -> Certificate Signing Request -> Sign Certificate
Consul Architecture
Certificate Format

X.509 Certificate

SPIFFE Compatible
Consul Architecture
Certificate Authority Rotation

Root

Intermediary

Leaf
Consul Architecture
Certificate Authority Rotation

Root

Intermediary

Leaf

Intermediary

Leaf

Root
Consul Architecture
Certificate Authority Rotation

Root

Intermediary

Leaf

Intermediary

Leaf

Intermediary

Leaf
Intentions

$ consul intention create \
  -deny \
  -meta description='Hello there' \
  web db \
...

$ consul intention get web db
Source: web
Destination: db
Action: deny
ID: 31449e02-c787-f7f4-aa92-72b5d9b0d9ec
Meta[description]: Hello there
CreatedAt: Friday, 25-May-18 02:07:51 CEST
Data plane
How Connect Secures Traffic
How Connect Secures Traffic

1. Envoy contacts the local Consul client to obtain certificates and keys required to enable Mutual TLS, this is through a bi-directional gRPC connection using Envoys xDS admin API.
2. The Consul client creates the x509 certificate containing the SPIFFE id for the proxy.
3. It requests this certificate to be signed by the Consul server.
Data plane

How Connect Secures Traffic

4. The certificate and key bundle is sent back to the proxy which it uses to secure the inbound connection and identify itself for outbound connections.
Data plane

How Connect Secures Traffic

5. The client starts a blocking connection to the server and waits for any updates to intentions or certificates.

6. Envoy listens for changes using the xDS API implemented in Consul.
Data plane
How Connect Secures Traffic

7. Client initiates request and performs TLS Handshake
8. Upstream service requests client certificate as part of mTLS request, validating cert is signed by valid source
Data plane

How Connect Secures Traffic

9. Envoy validates that the connection is allowed by calling the `ext_authz` filters api (once per new connection)
10. If allowed the request is passed to the upstream service
11. Send the response to the caller
How do we segment the network?

desired rules:
192.168.1.1 -> 192.168.10.1
192.168.1.2 -> 192.168.10.1
192.168.1.4 -> 192.168.10.1

reality:
192.168.10.0/24 -> 192.168.10.1
How do we segment the network?

network rules:
192.168.10.0/24 -> 192.168.10.1

intentions:
service a -> service b

All proxy ports are secured with mTLS and Intentions enabling secure and simple segmentation.
Consul on Kubernetes

$ helm install -n consul ./helm_charts/consul-helm
NAME:   consul
LAST DEPLOYED: Thu Nov 1 21:04:15 2018
NAMESPACE: default
STATUS: DEPLOYED

RESOURCES:

===> v1/ConfigMap
NAME                  AGE
consul-client-config  2s
consul-server-config  2s

===> v1/Service
consul-connect-injector-svc  2s
consul-dns                   2s
consul-server                2s
consul-ui                    2s
consul-ui                    2s

#...
Automatic sidecar injection

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: emojify-facebox
labels:
  app: emojify-facebox
spec: #...
template:
  metadata:
    labels:
      app: emojify-facebox
    annotations:
      "consul.hashicorp.com/connect-inject": "true"
      "consul.hashicorp.com/connect-service-port": "8080"
spec:
  containers:
  - name: emojify-facebox
    image: "machinebox/facebox"
    env:
      - name: MB_ADDRESS
        value: "127.0.0.1"
```
Consul Connect
Not Just Kubernetes

Features:

- Integrate service discovery between K8s and legacy applications
- Easily integrate non cloud native workloads
- Capable of running in Windows through Consul Connect Native
Consul Connect
Connect is L4 capable

Features:
- Able to route traffic to cloud managed datastore
- Bridge Cloud and On-Prem
Consul Connect
Connect has a native SDK

```go
// initialize the connect SDK and return a gRPC Resolver for Connect services
r, dialer, _ := resolver.NewConnectServiceQueryResolver(consulAddress, serviceName)

// create a RoundRobin load balancer
lb := grpc.RoundRobin(r)

// create a new gRPC client connection
c, err := grpc.Dial(
    cacheService,
    grpc.WithInsecure(),
    grpc.WithBalancer(lb),
    grpc.WithBlock(),
    dialer,
)
if err != nil {
    return nil, err
}

// create the gRPC service
cc := cache.NewCacheClient(c)
rc := &RemoteCache{cc}
```
DEMO
Demo
Architecture Overview

Kubernetes Cluster

Node a
Node b
Node c

Nginx Ingress

Envoy

Web Front End (ReactJS)

Public API

Web Front End (ReactJS)

Public API

Public API

Consul Client

Consul Server

Consul Server

Consul Server

Consul Client

Payment Service (Java)

Envoy

Blackbox Machine Learning (FaceBox.io)

Blackbox Machine Learning (FaceBox.io)

Blackbox Machine Learning (FaceBox.io)

Envoy

Envoy

Envoy

Envoy

Envoy

Envoy

Envoy

Linux VM

Consul Client

Legacy

Envoy
Implementing and managing certificates is hard
Reality, Consul Connect can do all this for you

• You need to manage a Certificate Authority (CA)

• Have to distribute, and rotate certificates and keys

• Application code needs to be change to handle TLS termination

• Consul Connect has a built in CA, capable of leveraging HashiCorp Vault

• Consul Connect manages key and certificate rotation

• Sidecar proxy terminates TLS and implements Authz, no code changes
Service and Network segmentation in Dynamic environments is hard

Reality, Consul Connect can do all this for you

• Managing routing and firewall rules when application locations keep changing

• Maintaining millions of rules in a large environment

• Open networks due to complexity

• Manage service policy through intentions

• Leverage mTLS to service to service authorization
Thank you.

twitter: @sheriffjackson
email: nic@hashicorp.com