Scaling Trust with Millions of Containers: Microsegmentation Strategies for Authorization
About Me

- Drupal
  - Security Team
  - Database Maintainer
- Service Mgmt for RHEL/Ubuntu
  - Committer
  - Scalable CGroups Management
- Pantheon
## Pros and Cons of Microservice Architecture

<table>
<thead>
<tr>
<th>Pros</th>
<th>Cons</th>
</tr>
</thead>
<tbody>
<tr>
<td>● Language/framework diversity</td>
<td>● Language/framework diversity</td>
</tr>
<tr>
<td>● Lower-risk experimentation</td>
<td>● Change management</td>
</tr>
<tr>
<td>● Compartmentalized testability</td>
<td>● New attack surfaces</td>
</tr>
<tr>
<td></td>
<td>● Single point of failure (SPoF) risks for authorization</td>
</tr>
</tbody>
</table>
Traditional, Monolithic Web Architecture

Internet → Firewall → Network → Runtime
- AuthZ
- AuthN
- Business Logic 1
- Business Logic 2
- Business Logic 3
→ Network → Storage
Monoliths Have Important Security Upsides

- Runtimes can enforce caller/callee relationships
- Limited serialization and parsing
- Easy, frequent, fine-grained authorization checks

- A singular “edge” to clients
- Centralized logging

We lose these with microservices... ...unless we’re careful.
A Naive Microservice Deployment

Internet

Firewall

Network

Runtime 1
- AuthZ
- AuthN

Network

Storage 1

Runtime 2
- Business Logic 1

Network

Storage 2

Runtime 3
- Business Logic 2

Network

Storage 3

...
This is worse than before.

Microservices do not guarantee microsegmentation.
The Attack Surface Has Expanded

Inherent Challenges

● Lots of serialization and parsing
● Every stack has unique vulnerabilities
● Anything can (try to) send a packet to anything

Common Anti-Patterns = Risk

● **All in the Family:** Trusting everything behind the firewall
● **God Proxies:** Trusting frontend systems to behave
The Confused Deputy Problem

- Attacker
- Tricks
- Service
- Request
- Attack Target
An Old Problem

Attacker → Tricks → passwd → Alters → Someone Else’s Password
A Current Problem

Attacker -> Tricks -> Web Server -> Injected SQL -> Database
Back with a Vengeance in Microservices

Attacker

Tricks

Web Frontend or Proxy

Payment Data

Authentication Data

...
The Big Question

“A Foot in the Door”

If an attacker gains the ability to make network requests behind the firewall what happens?
Getting Behind the Firewall

Many Methods

- Tricking a Proxy
- VPN Access
- Arbitrary Exec on Any Service
- Employee Device Compromise

Many Successful Hacks

- Sony
- Panama Papers
- Stratfor
- DNC Emails
- Equifax

...all relied on using a vulnerability in one system to compromise another
Locking Down Access
Phase One: mTLS and Firewalls

**Goal:** Only allow each service to interact with the **services** they have a need to access.

<table>
<thead>
<tr>
<th>From</th>
<th>AuthN/IdP Service</th>
<th>LDAP</th>
<th>Billing Service</th>
<th>Cardholder Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Web Frontend</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>AuthN/IdP Service</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>LDAP</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>Billing Service</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>Cardholder DB</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
</tbody>
</table>
## Complements: Firewalls and mTLS

<table>
<thead>
<tr>
<th></th>
<th>Firewalls and SDNs</th>
<th>Mutual TLS (mTLS)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Enforced</strong></td>
<td>Beneath the application</td>
<td>By the application or proxy</td>
</tr>
<tr>
<td><strong>Complexity</strong></td>
<td>Lower</td>
<td>High</td>
</tr>
<tr>
<td><strong>Scaling</strong></td>
<td>Easy (if Static)</td>
<td>Easy</td>
</tr>
<tr>
<td></td>
<td>Tough (if Dynamic)</td>
<td></td>
</tr>
<tr>
<td><strong>Ideal Use</strong></td>
<td>Coarse Segmentation</td>
<td>Fine or Micro Segmentation</td>
</tr>
</tbody>
</table>
We’re back to the **same** surface area (or less) than a monolithic design.
But, what about attacks that traverse established, allowed paths?
**Problem:** The web frontend can access anyone’s data if tricked into doing so.

<table>
<thead>
<tr>
<th>From</th>
<th>To</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice → Web Frontend</td>
<td>Alice’s Billing Records</td>
</tr>
<tr>
<td></td>
<td>Bob’s Billing Records</td>
</tr>
<tr>
<td>Bob → Web Frontend</td>
<td></td>
</tr>
<tr>
<td>Eve <em>Attacks</em> → Web Frontend</td>
<td></td>
</tr>
</tbody>
</table>

*Eve’s Billing Records:*
- Allowed
- Disallowed

With Just Firewalls and mTLS

Pantheon.io
Phase Two: Capabilities

**Goal:** Only allow each service to interact with the *data* they have an *immediate* need to access.

- **Allowed**
- **Disallowed**

<table>
<thead>
<tr>
<th>From</th>
<th>Alice’s Billing Records</th>
<th>Bob’s Billing Records</th>
<th>Eve’s Billing Records</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Alice ➔ Web Frontend</strong></td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td><strong>Bob ➔ Web Frontend</strong></td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td><strong>Eve <em>Attacks</em> ➔ Web Frontend</strong></td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
</tbody>
</table>
# Ambient Authority vs. Capabilities

<table>
<thead>
<tr>
<th>Locus</th>
<th>Ambient Authority</th>
<th>Capabilities</th>
</tr>
</thead>
<tbody>
<tr>
<td>Authorization</td>
<td>Lookups for Actor+Verb+Object</td>
<td>Implied by possession of token for Verb+Object</td>
</tr>
<tr>
<td>Scaling Impact</td>
<td>Lookups on every request</td>
<td>Renewals at time intervals</td>
</tr>
<tr>
<td>Each Microservice Needs...</td>
<td>Client for session and authorization lookups (or even business logic)</td>
<td>Public key and parser</td>
</tr>
<tr>
<td>Biggest Risk</td>
<td>Actors can be tricked</td>
<td>Revocation</td>
</tr>
</tbody>
</table>
Capabilities in Action: A Familiar Scenario

Person
  Obtains Ticket
  Gives Ticket

Box Office
  Uses

Usher
  Allows

Ticket-Checking

Payment
Systems

Purchase
Records

Theater
Access

Records
Applied to Web Systems

Person
- Obtains User Token
- Gives User Token + Project ID

AuthZ
- Uses
- Receives Project Token
- Attempts to Access
- Forwards Project Token

Web Frontend
- Attempts to Access
- Receives Project Token

User Credential Store
- Generates user token and uses a private key to sign it

RBAC Service
- Validates user token with a public key
- Checks project access rights
- Generates project token and uses a private key to sign it

Billing System
- Verifies project token with public key and matches with project being accessed

Pantheon.io
Capabilities in the Wild: S3 Signed URLs

https://s3.amazonaws.com
/#{bucket}
/#{path}

AWSAccessKeyId=#{key-id}
Expires=#{expiration}
Signature=#{signature}

Properties:

- **Key ID** allows audit trail of use
- Possibly to **forward** without the recipient having authority
- Authorizations map to specific **objects** and **actions**
Authorization: Bearer
eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiYWRtaW4iOnRydWV9.EkN-DOSnsuRjRO6BxXemmJDm3HbxrbRzXglbN2S4sOkopdU4IsDxTI8j019W_A4K8ZPJijNLis4EZsHeY559a4DF0d50_0qgHGuERTqYZyuhtF39yxJPAjUESwxk2J5k_4zM3O-vtd1Ghyo4IboKKSy6J9mTniYJPenn5-HIirE
JWT Decoded

Header:
{
  "alg": "RS256",
  "typ": "JWT"
}

Payload:
{
  "sub": "project-39839",
  "access": "ro",
  "name": "Jane Doe",
  "email": "jane@doe.com"
}

Signature: (not human readable)

Properties:

- All the benefits of signed URLs
- Fully validatable en route
- Strong hashing (HMAC SHA-2)
- Widespread library support
Revisiting the Equifax Breach

How the Attack Worked

1. **Exploit** of Apache Struts
2. Arbitrary **code execution** from the web frontend
3. Used the **ambient authority** of the frontend to obtain bulk records from deeper systems

If They Had Capability Security

- **Web frontend vulnerability:** Would only affect current visitors
- **Capability token system vulnerability:** Possible to throttle in the web frontend
Scaling Security to Millions of Containers

Firewalls and mTLS

- Firewalls for coarse segments
- Service-to-service mTLS
- Short-lived certs (no revocation)
- Internal, automated certificate authority

Capability Tokens

- Self-evident authorization (no lookups)
- Forwardable with queues and sub-requests
- Contain actor data only for logging and audit purposes
Questions?

@DavidStrauss
Bonus Capabilities in the Wild: FT Cookies

Financial Times + Paywall AuthN in CDN

1. **User** authenticates.

2. Authorization system maps **user** to **subscription level**.

3. User receives a **capability token** with the signed authorization information in a **JWT-format cookie**.

4. The **CDN can validate** the user’s subscription as they load paywalled articles and still **hit the cache**.

CDNs create familiar challenges: Capability tokens allow proxies to **validate** without being **fully trusted**.
Complements: Firewalls and mTLS

**Firewalls and SDNs**
- Imposed beneath the application layer
- Sometimes imposed beneath the virtual machine or even physical machine
- **Simpler** initial setup
- Scaling up requires dynamic SDN lookups and is especially hard in cloud environments

**Mutual TLS (mTLS)**
- Each side (client and server) has certificates signed by a mutually trusted authority (usually internal)
- **Complex** initial setup
- Application connections use a certificate on each side
- **Scales** effortlessly

More useful for **coarse** segmentations

More useful for **fine** segmentations
Ambient Authority vs. Capabilities

Ambient Authority

- Shallow services have full reign of data in the deeper systems they use
- Shallow services apply access rules to requests made to deeper ones
- Access decisions require constantly looking up and applying RBAC rules, which scales poorly
- Shallow services are vulnerable to becoming a confused deputy

Capabilities

- Clients accessing the shallower services must supply access tokens
- Shallow services forward the tokens as necessary when invoking deeper services
- Access is self-evident from the tokens, which scales well
- Even when tricked, shallow services have limited ability to access or manipulate deeper services