Security Considerations for Microservice Architectures

Daniel Richter, Tim Neumann, and Andreas Polze
Operating Systems & Middleware Group
Hasso Plattner Institute at University of Potsdam, Germany
Motivation

- EPA – the legacy system
  - reserve and book train seats operated by Deutsche Bahn (German railway)
  - 1 mio seat requests & 300,000 bookings
  - first version: 1980s
  - set of *Pathway Services* as part of *HP NonStop* system
  - especially fault-tolerant and highly-available
Motivation
Motivation

Microservices to the Rescue!

▪ small, independent, autonomous services
▪ small, specific range of features
▪ encapsulates all its functions and data
▪ cooperation with other microservices (usually ReST & message queues)
▪ DevOps
Motivation

Microservices, but...

▪ introduces additional complexity through dependencies to supporting technology
  ▪ e.g. for deployment, scaling and management of containerized applications.

▪ use of additional technologies increases the surface attack area
  ▪ different technology stacks
  ▪ different vendors, teams, products...
  ▪ frequent new versions
Our Testbed
Our Testbed

- application layer

Three **base layer groups:**
- compute provider
- encapsulation technology
- deployment

Example: secure the communication between individual application components *(authentication and authorization)*
Base Layer Groups
Base Layer Groups

Technologies

- **Compute Provider** group
  - all AWS related layers
  - provides some kind of computing infrastructure (physical or virtual machines, some networking solution, and some file storage system)
  - start a new machine (based on template) & connect to network
    - physical machines, virtual machines
    - own data center, 3rd party data center, cloud provider
  - e.g. AWS, Google Cloud Platform, Microsoft Azure, OpenStack
Base Layer Groups

Technologies

▪ Encapsulation Technology group
  ▪ Docker layer and Weave layer
  ▪ provide a distributed runtime environment for containers, responsible for isolating services from each other so they cannot interfere with each other (except by predefined communication)

▪ running multiple (lightweight) services on one machine
  ▪ VM-based encapsulation vs. container-based encapsulation ➔ isolation vs. overhead, technology independence, tools
  ▪ multiple network addresses ➔ overlay network
Base Layer Groups

Technologies

- **Deployment group**
  - Kubernetes layers
  - distribute containers among multiple nodes automatically
  - take software in source or binary format and ensure its execution and configuration
  - avoid doing “by hand”
  - e.g. Docker Swarm, Kubernetes
Base Layer Groups

Security Evaluation

**Compute provider group**

- managed by Amazon, security cannot be influenced by customers
- data centers comply with various commercial and governmental security guidelines
  - such as PCI DSS Level 1
- allows detailed rules for communication between EC2 instances
- Amazon VPC acts as a firewall
Security Evaluation

Encapsulation technology group

- Docker allowed certain users full access to the computer on which it is installed (as required by Kubernetes)
- Weave Net is configured and managed by Kubernetes
- Weave Net default configuration can be improved by specifying password to encrypt communication between the Weave Net instances running on each node
Deployment group

- Kubernetes and Weave Net provide one network to all applications running in Kubernetes, allowing communication without restrictions (by default)
- employ Network Policies to limit communication to specific applications
- Kubernetes 1.5
  - very coarse-grained access control (essentially either full or no access to cluster)
  - API server: unauthenticated & unencrypted endpoint
- Kubernetes 1.6: Role-Based Access Control
Application Layer
Application Layer

Authentication & Authorization

Methods
- trust
- network policy
- IP-based
- key/token-based
- MAC-based (Message Authentication Code)
- signing-based & Certificate-based
- session-based & Password-based
Application Layer

Authentication & Authorization

Criteria

- support of fine-grained access control
- secret-based
  - pre-shared, asymmetric, after session start
- session-based
- network-based
- stack level
  - network, Application, transport
## Application Layer

### Authentication & Authorization

<table>
<thead>
<tr>
<th>Method</th>
<th>Fine-grained access control</th>
<th>Secret-based</th>
<th>Session-based</th>
<th>Network-based</th>
<th>Stack Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trust</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>N/A</td>
</tr>
<tr>
<td>Network policy</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Network</td>
</tr>
<tr>
<td>IP-based</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Network/Application</td>
</tr>
<tr>
<td>Key/token-based</td>
<td>Yes</td>
<td>Yes, pre-shared</td>
<td>No</td>
<td>No</td>
<td>Application</td>
</tr>
<tr>
<td>MAC-based</td>
<td>Yes</td>
<td>Yes, pre-shared</td>
<td>No</td>
<td>No</td>
<td>Application</td>
</tr>
<tr>
<td>Signing-based</td>
<td>Yes</td>
<td>Yes, asymmetric</td>
<td>No</td>
<td>No</td>
<td>Application</td>
</tr>
<tr>
<td>Certificate-based</td>
<td>Yes</td>
<td>Yes, asymmetric</td>
<td>No</td>
<td>No</td>
<td>Transport</td>
</tr>
<tr>
<td>Session-based</td>
<td>Yes, within a session</td>
<td>Yes, after session start</td>
<td>Yes</td>
<td>No</td>
<td>Application</td>
</tr>
<tr>
<td>Password-based</td>
<td>Yes</td>
<td>Yes, pre-shared and after session start</td>
<td>Yes</td>
<td>No</td>
<td>Application</td>
</tr>
</tbody>
</table>
Evaluation of Authn & Authz in our Testbed
simplified reimplementation of the *Elektronische Platzbuchungsanlage* (EPA, “electronic seat reservation and booking system”) of Deutsche Bahn

- Customer component (manage login data)
- Seat & schedule component
- Booking component

- each component backed by a separate database
- two front-ends
Evaluation of Authentication and Authorization in our Testbed

Communication Groups
Evaluation of Authentication and Authorization in our Testbed

Communication Groups

(a) third-party software: no control
(b) between different core components: assumed trusted network
Evaluation of Authentication and Authorization in our Testbed

Communication Groups

(c) between BFFs and core components: separate networks & BFFs may considered untrusted (directly accessible from public network)
Evaluation of Authentication and Authorization in our Testbed

Communication Groups

(d) public network from untrusted device
(e) public network from trusted device
Evaluation of Authentication and Authorization in our Testbed

Communication Groups

(f) public network from untrusted device; must be publicly accessible, no authorization or authentication required or possible
Evaluation of Authentication and Authorization in our Testbed

Communication Groups

(g) public network from untrusted device; does not have to be publicly accessible
Evaluation of Authentication and Authorization in our Testbed

Communication Channels

Two authentication and authorization methods were used:

- Token-based authentication and authorization (connect to the database servers)
- session-based authentication and authorization (connections between display devices and BFFs)
Conclusion & Future Work

▪ In comparison to monolithic applications, the use of cloud-infrastructure (compute provider layer) introduces additional complexity as well as additional attack vectors.

▪ Compared to classic VM-based cloud applications, technologies introduced in the encapsulation technology layer lead to the fact that more safety requirements have to be met.

▪ Choice between complexity and practicality especially in Microservice architectures.
Conclusion & Future Work

Additional security concerns (OWASP Top 10 Security Risks 2017)

- authorization and authentication (A2:2017)
- security misconfiguration (A6:2017)
- vulnerable components (A9:2017)
- insufficient logging and monitoring (A10:2017)
- Dev-ops (culture unifying development and operation) nonproduction environment exposure
Conclusion & Future Work

Security should be a consideration from the very beginning of planning a system, to be able to implement effective and comprehensive security measures throughout the project – especially if monolithic applications are to be realized based on microservice applications.

We would like to thank Lena Feinbube, Leonard Marschke, Cornelius Pohl, Robert Beilich, Tim Basel, Timo Traulsen, Henry Hübler, Dr. Stephan Gerberding, Wolfgang Schwab, and Ingo Schwarzer for their support and assistance with this project.