Security Considerations for Microservice Architectures

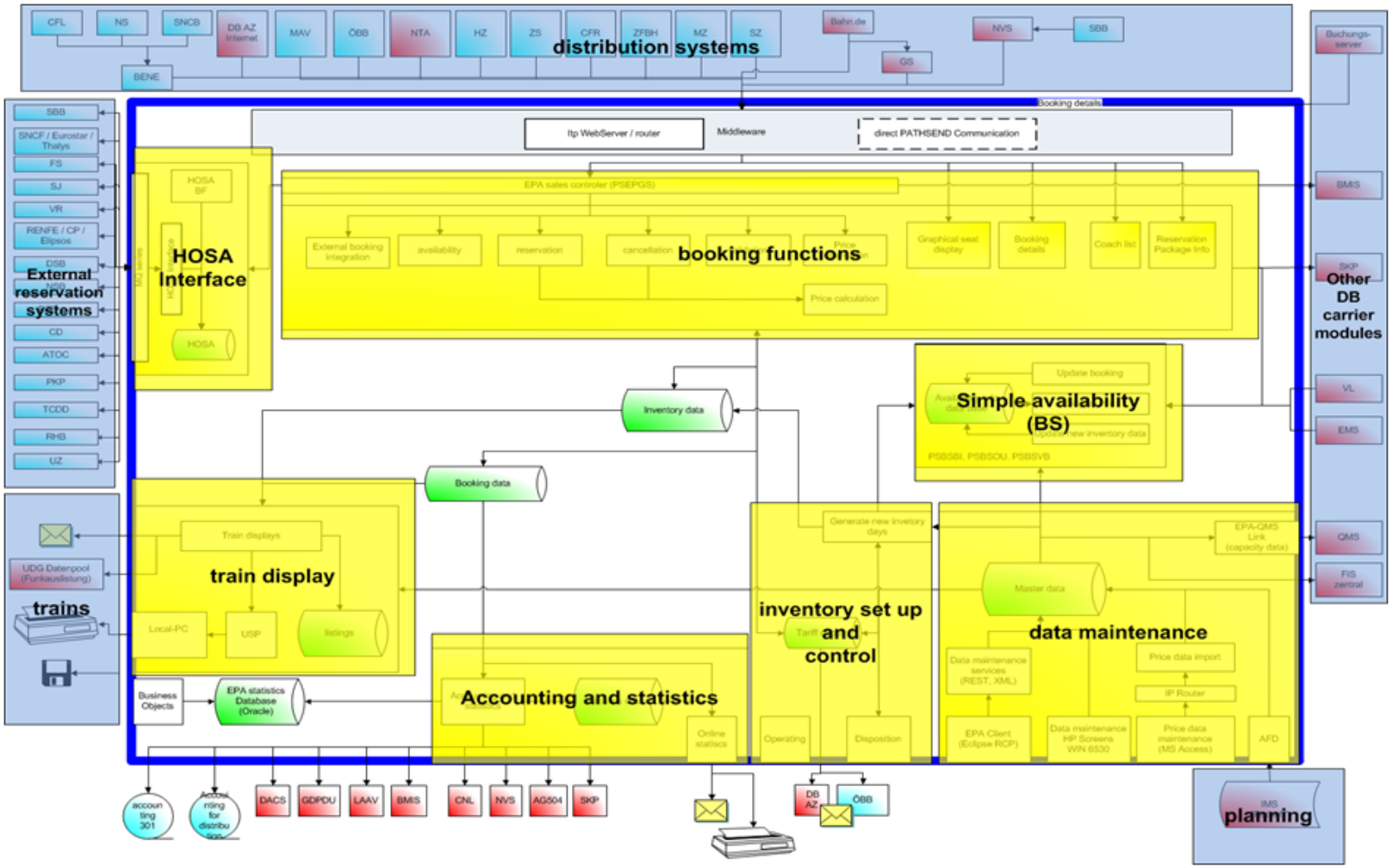
Daniel Richter, Tim Neumann, and Andreas Polze

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


distribution systems
External reservation systems
HOSA Interface
booking functions
Other DB carrier modules
Simple availability (BS)
train display
trains
inventory set up and control
data maintenance
Accounting and statistics
planning

Motivation

Microservices to the Rescue!

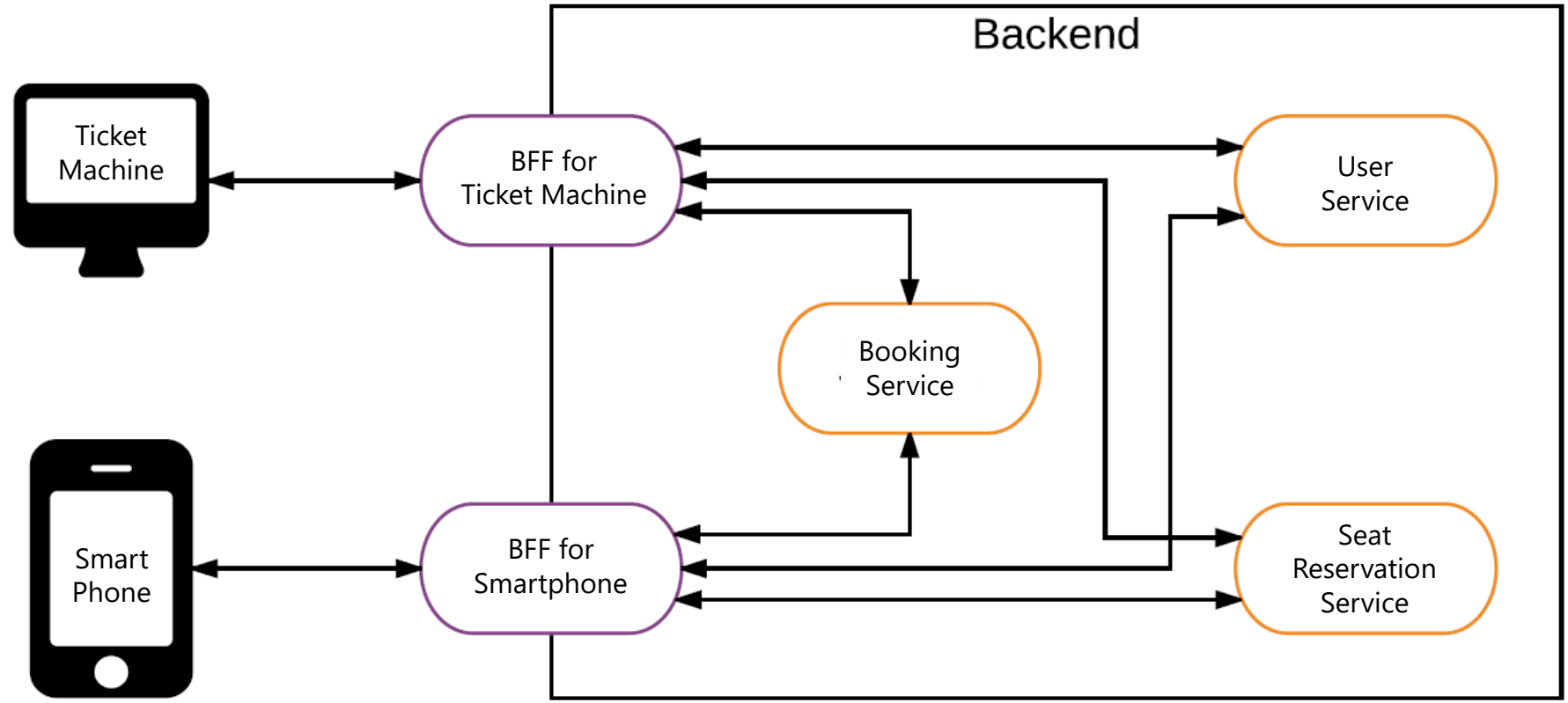
- small, independent, autonomous services
- small, specific range of features
- encapsulates all its functions *and* data
- cooperation with other microservices (usually ReST & message queues)
- DevOps

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


Backend
Ticket Machine
BFF for Ticket Machine
User Service
Booking Service
Smart Phone
BFF for Smartphone
Seat Reservation Service

Our Testbed

- application layer


Three **base layer groups**:

- compute provider
- encapsulation technology
- deployment

EC2, EBS, VPC

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

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

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

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

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 Groups associated with i-01158862e30a7a323

Ports	Protocol	Source	HPI-Kube	HPI-Kube-Ingress	HPI-Internal-SSH
6783-6784	udp	sg-0234f469	✔		
6781	tcp	sg-0234f469	✔		
6443	tcp	sg-0234f469, sg-61408f0a	✔		
10255	tcp	sg-0234f469	✔		
6783	tcp	sg-0234f469	✔		
10250	tcp	sg-0234f469	✔		
9898	tcp	sg-0234f469	✔		
6782	tcp	sg-0234f469	✔		
4194	tcp	sg-0234f469	✔		
80	tcp	sg-63995708, sg-ac06c6c7		✔	
1194	udp	sg-ac06c6c7		✔	
53	udp	sg-63995708		✔	
443	tcp	sg-63995708, sg-ac06c6c7		✔	
22	tcp	sg-61408f0a, sg-ac06c6c7			✔

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

Security Evaluation

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

Authentication & Authorization

Methods

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

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

Authentication & Authorization

Method	Fine-grained access control	Secret-based	Session-based	Network-based	Stack Level
Trust	No	No	No	No	N/A
Network policy	No	No	No	Yes	Network
IP-based	Yes	No	No	Yes	Network/ Application
Key/token-based	Yes	Yes, pre-shared	No	No	Application
MAC-based	Yes	Yes, pre-shared	No	No	Application
Signing-based	Yes	Yes, asymmetric	No	No	Application
Certificate-based	Yes	Yes, asymmetric	No	No	Transport
Session-based	Yes, within a session	Yes, after session start	Yes	No	Application
Password-based	Yes	Yes, pre-shared and after session start	Yes	No	Application

Evaluation of Authn & Authz in our Testbed

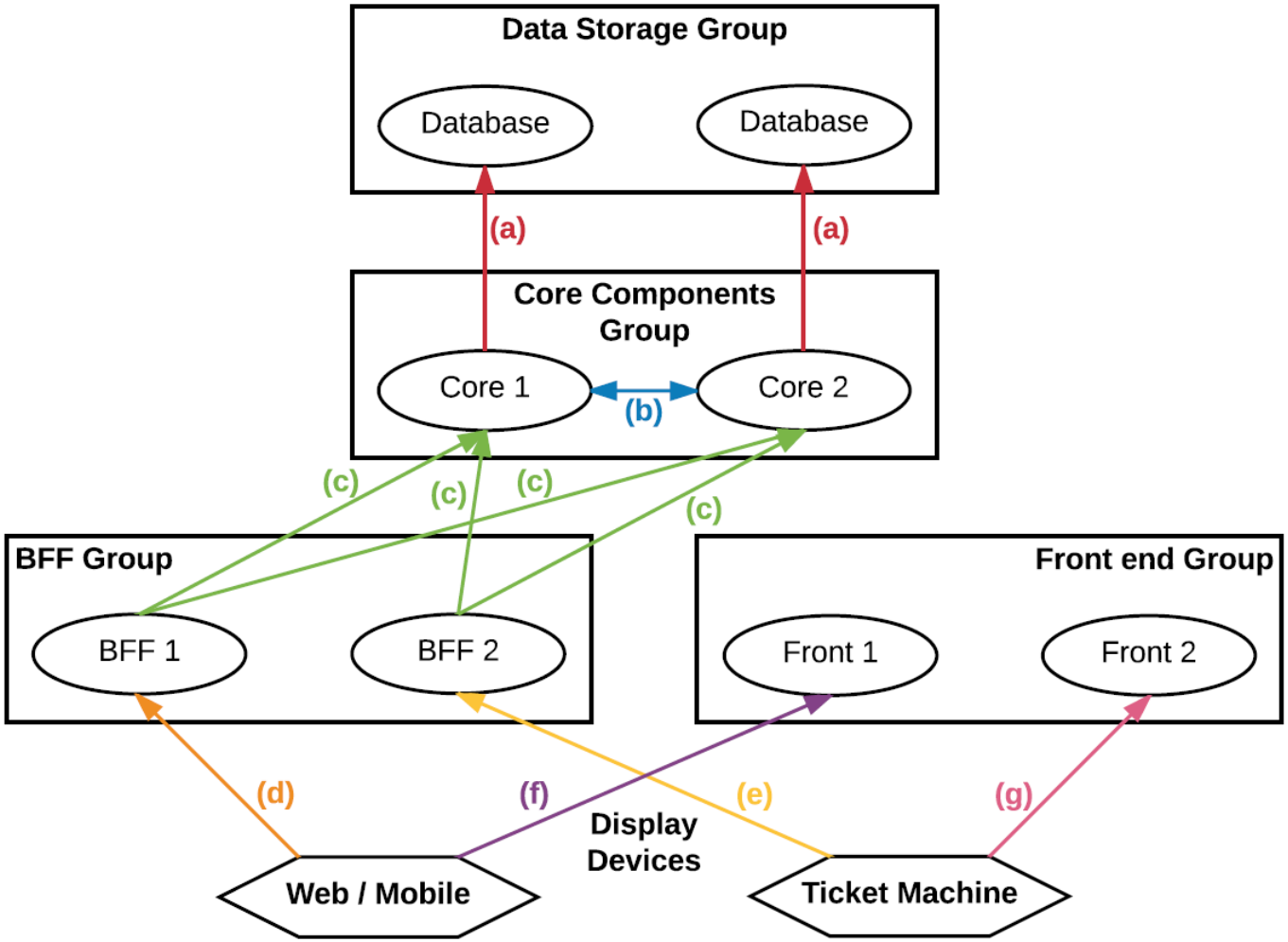
Testbed Components

simplified reimplementation of the *Elektronische Platzbuchungsanlage* (EPA, "electronic seat reservation and booking system") of Deutsche Bahn

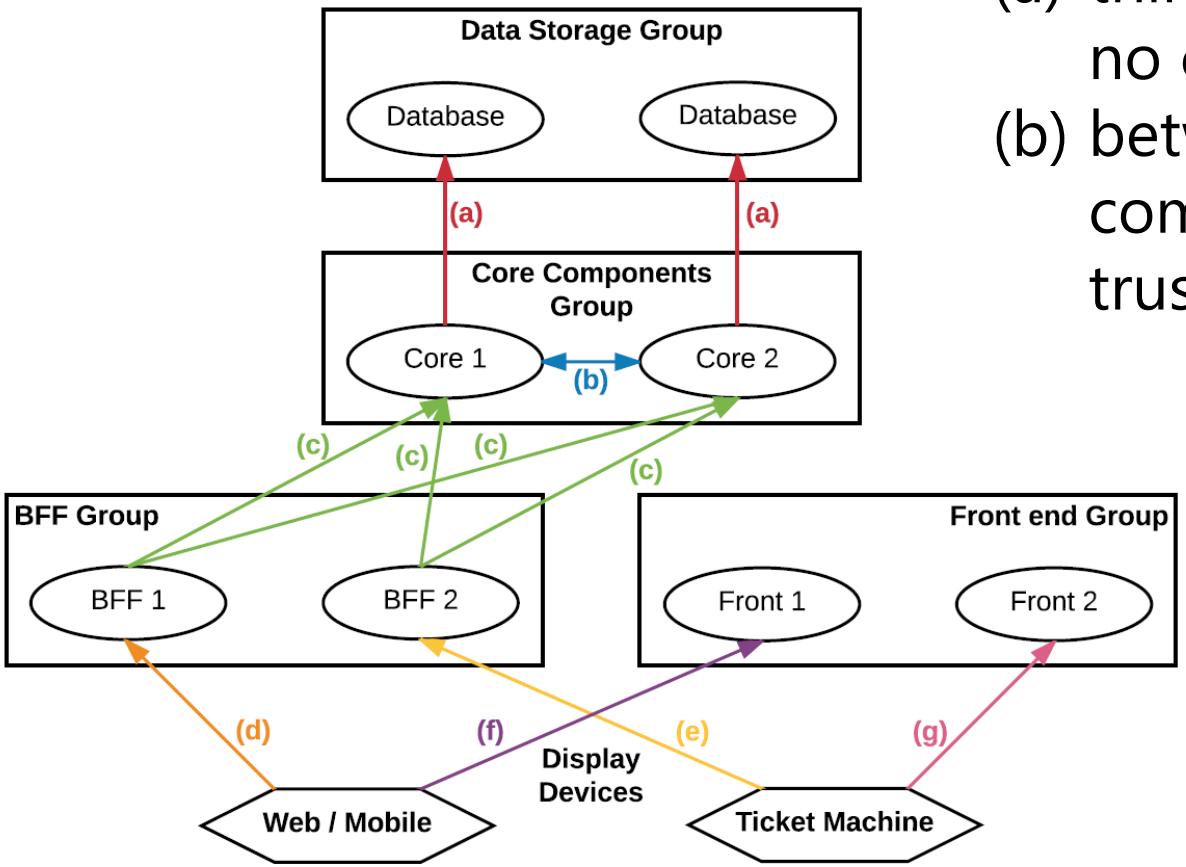
- Customer component (manage login data
- Seat & schedule component
- Booking component

- each component backed by a separate database
- two front-ends

Communication Groups



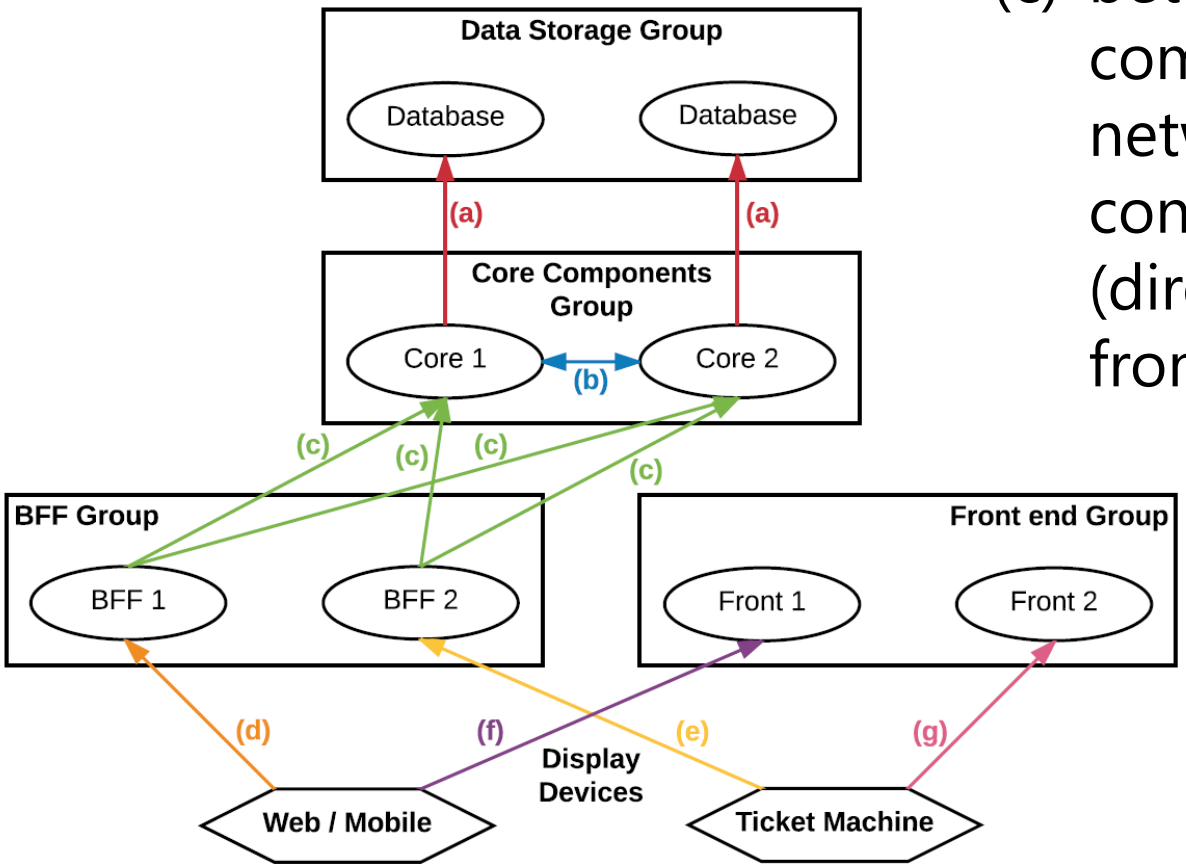
Communication Groups



(a) third-party software: no control

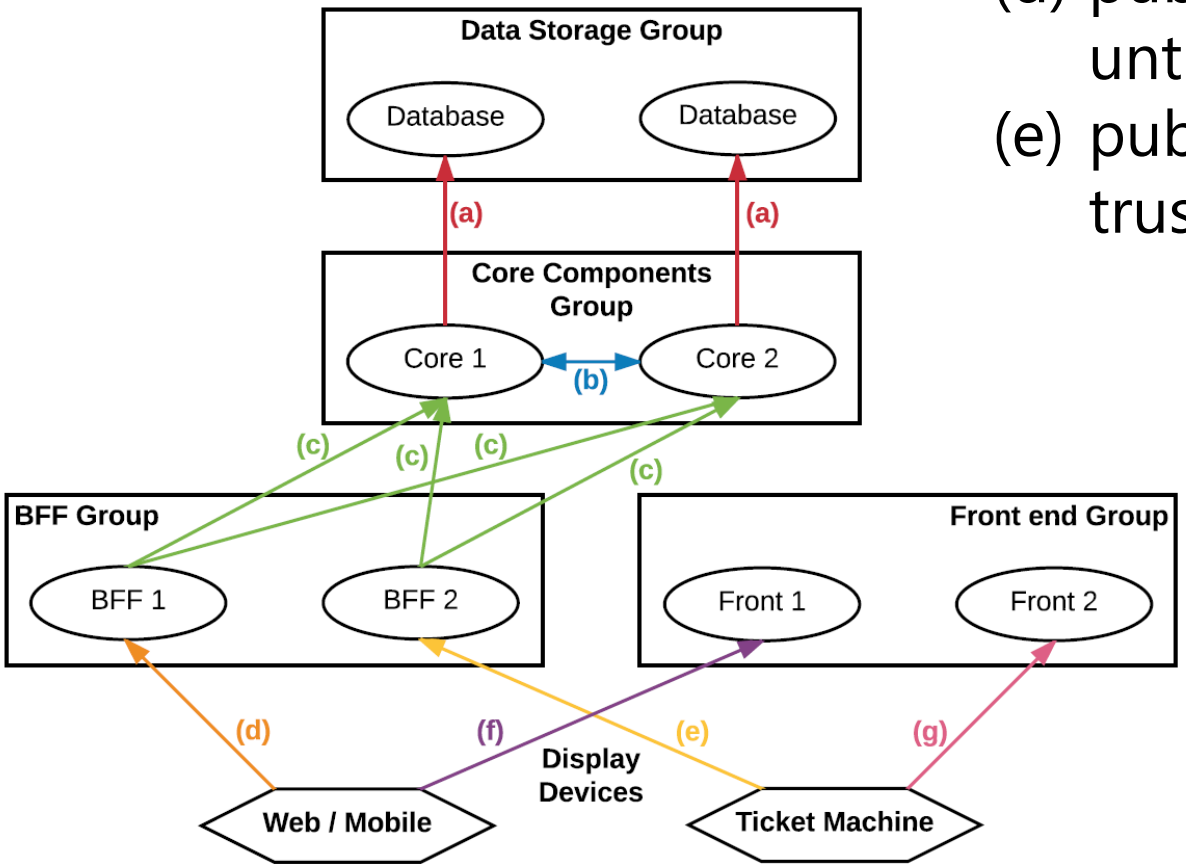
(b) between different core components: assumed trusted network

Communication Groups



(c) between BFFs and core components: separate networks & BFFs may considered untrusted (directly accessible from public network)

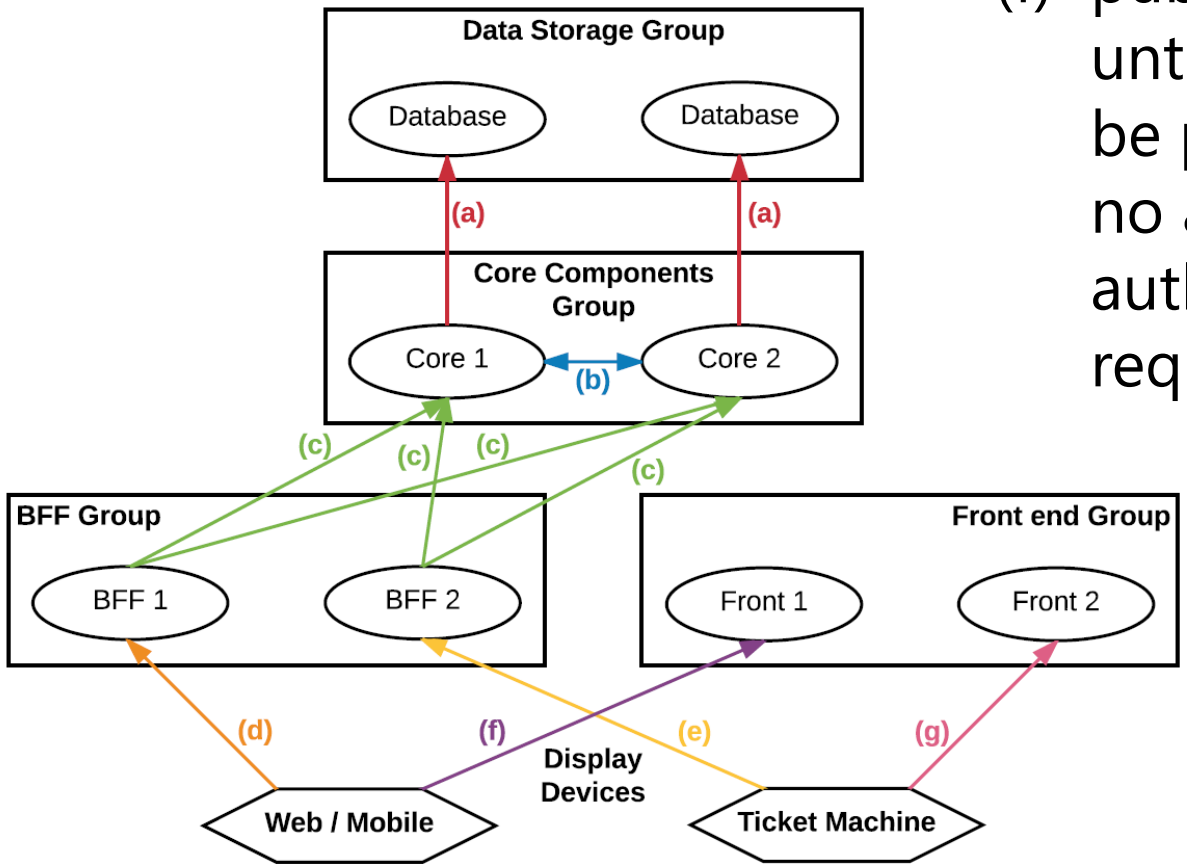
Communication Groups



(d) public network from untrusted device

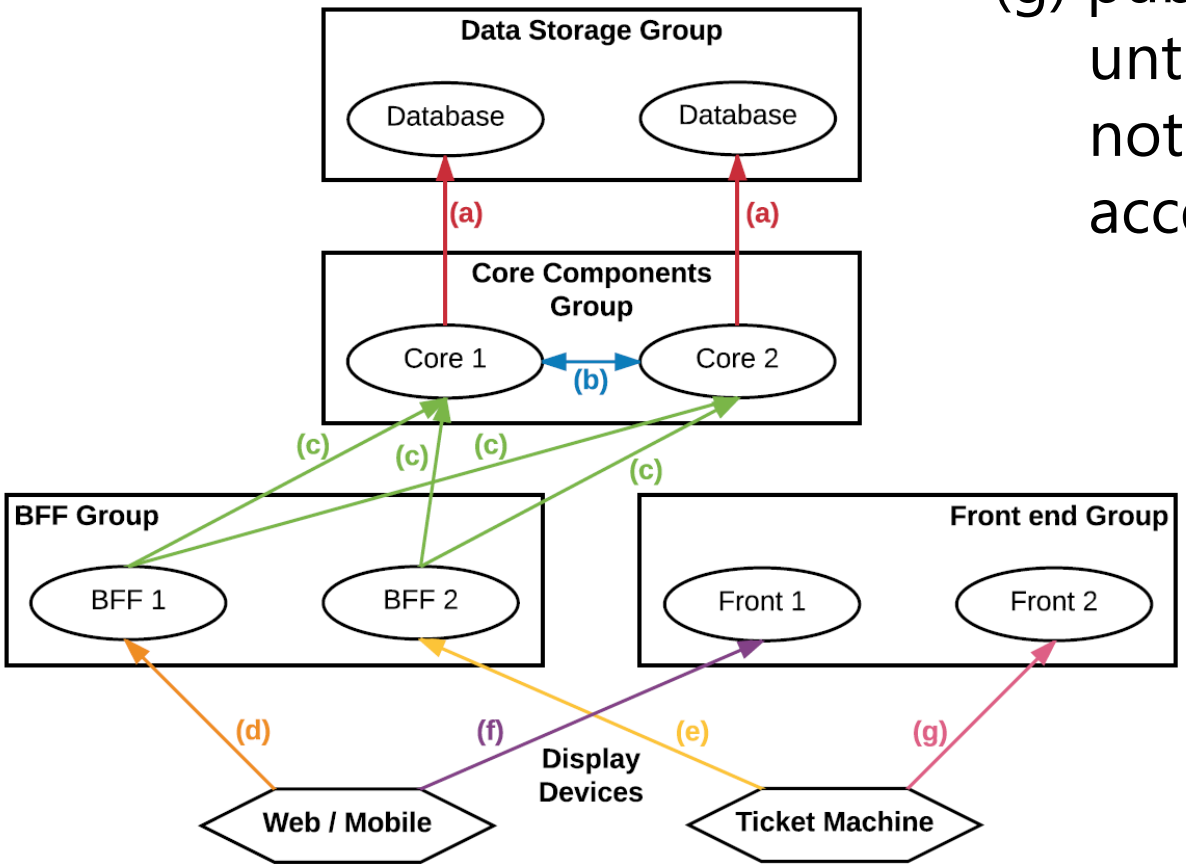
(e) public network from trusted device

Communication Groups



(f) public network from untrusted device; must be publicly accessible, no authorization or authentication required or possible

Communication Groups



(g) public network from untrusted device; does not have to be publicly accessible

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.