

Design Flaws and how to spot them

Simon Schneider

Institute of Software Security – TU Hamburg – Germany

`simon.schneider@tuhh.de`

Lecture in Software Security course



DESIGN FLAWS

3 Types of Security Design Flaws

Omission weakness

A threat is not addressed
at all

Commission weakness

A threat is addressed with
the wrong solution

Realization weakness

A threat is addressed with
the right solution, but its
implementation is wrong

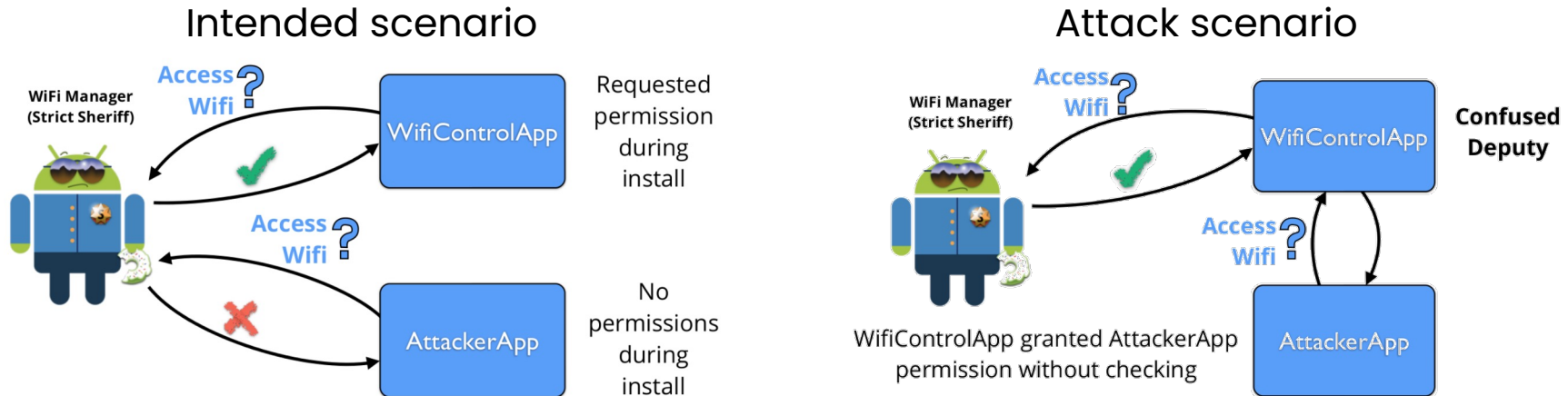
Examples

- Unprotected Storage of Credentials
- Using Weak Authentication (e.g., "API keys")
- Trust Boundary Violation (e.g., I/O validation)

Mehdi Mirakhorli: "Common Architecture Weakness Enumeration (CAWE)" <http://blog.ieeesoftware.org/2016/04/common-architecture-weakness.html> and Santos, Joanna C S, Katy Tarrit, and Mehdi Mirakhorli. "A Catalog of Security Architecture Weaknesses." In *International Conference of Software Architecture Workshops (ECSAW)*, 2017. <https://doi.org/10.1109/ICSASW.2017.25>.

Example of a Design Flaw

- CWE-926: Improper Export of Android Application Components ("confused deputy")



CWE: <https://cwe.mitre.org/data/definitions/926.html>

Image: https://owasp.org/www-pdf-archive/Danelon_OWASP_EU_Tour_2013.pdf

Example: Permission re-delegation in Android

The Android application **exports a component for use by other applications**, but **does not properly restrict** which applications can launch the component or access the data it contains.

Intent spoofing: attacker sends malicious intent to an intent processor (e.g., **permission re-delegation**, but **intent flooding** also an option)

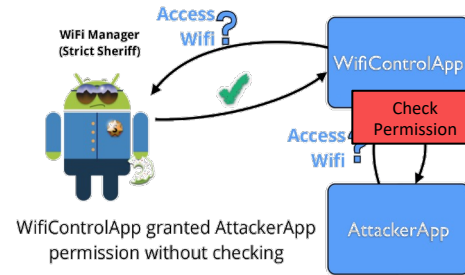


Obvious consideration:
design choices/fixes have an
effect on the code base

Possible Solution

Checking for permissions at the receiving side

```
public class Sender extends Service {  
    public int onStartCommand(Intent intent, int flags, int startId){  
        if (checkCallingPermission("android.permission.SEND_SMS") ==  
            PackageManager.PERMISSION_GRANTED) {  
            String phoneNumber = intent.getStringExtra("PHONE_NUMBER");  
            String msg = intent.getStringExtra("MSG_CONTENT");  
            SmsManager smsManager = SmsManager.getDefault();  
            smsManager.sendTextMessage(phoneNumber, null, msg, null, null);  
        }  
    }  
}
```





More coming up in course
“Secure Software Engineering”



ARCHITECTURAL SECURITY

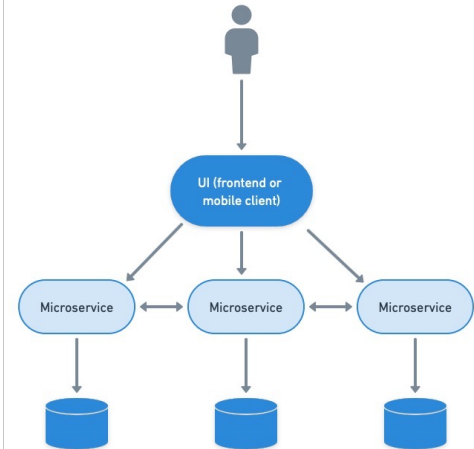
Modelling Software Systems

- Software models = **abstractions** of the implementation
- **Architectural** models: high-level overview of software **components** and their **interconnections**
- Useful to **reason about architectural design and security** of the system
- Some **design flaws can be identified** in architectural models

Microservice Architecture

“Microservices are small, autonomous services that work together” [1]

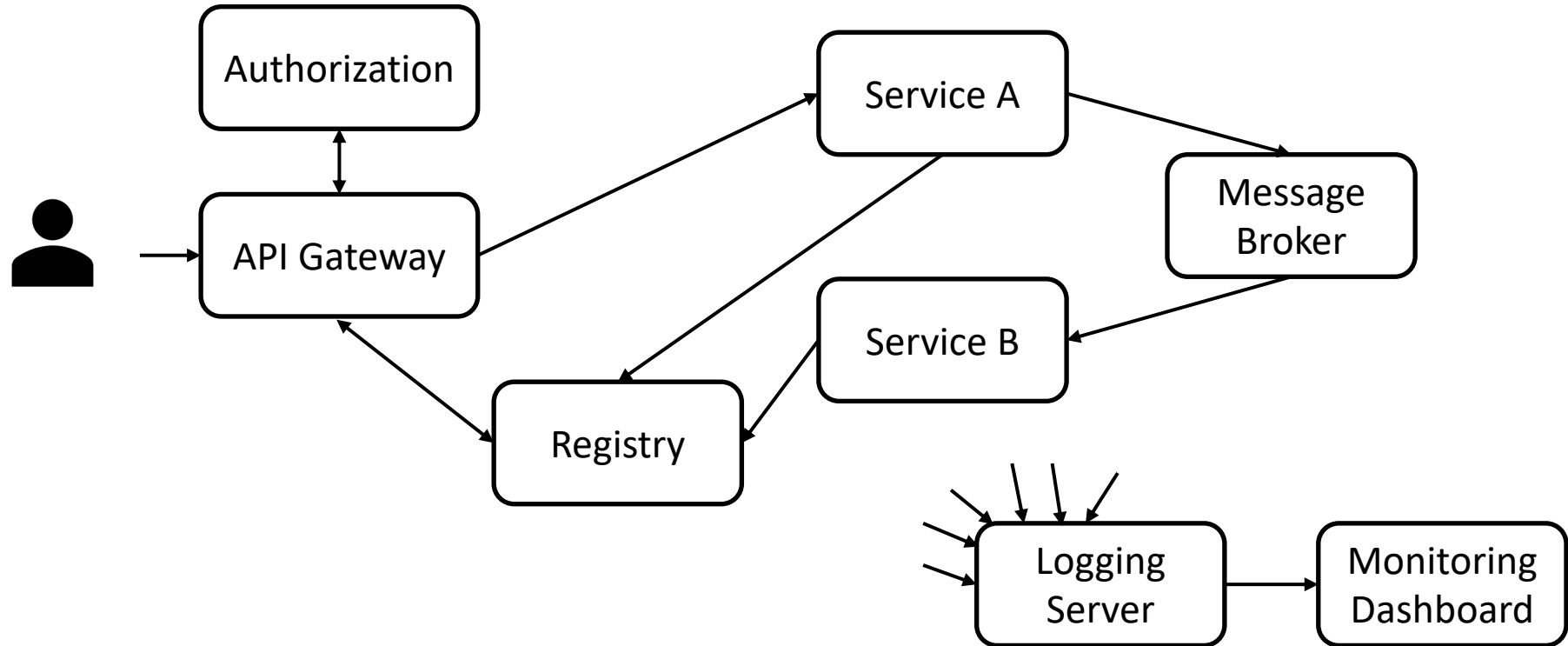
- Split application into multiple microservices, each with single functionality and dedicated resources
- Deploy them independently
- Let them communicate over the network to fulfill business logic



Typical Components of μ -services

- Some components present in most microservices:
 - **Service registry**: keeps track of deployed instances
 - **Authorization server**: central server issuing auth tokens
 - **API gateway**: single entry point to the system
 - **Monitoring dashboard**: visualization of the system
 - **Logging server**: central server storing logs of all others
 - **Message broker**: for asynchronous communication between services

Example of typical architecture



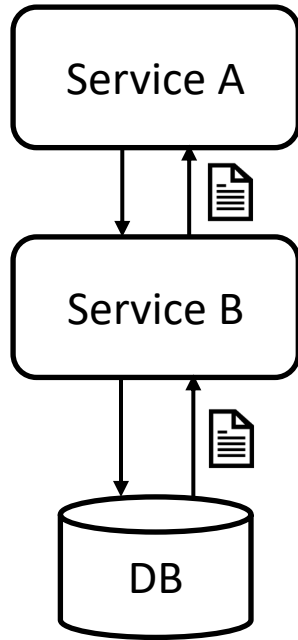
Security Challenges of Microservices

Most pressing:

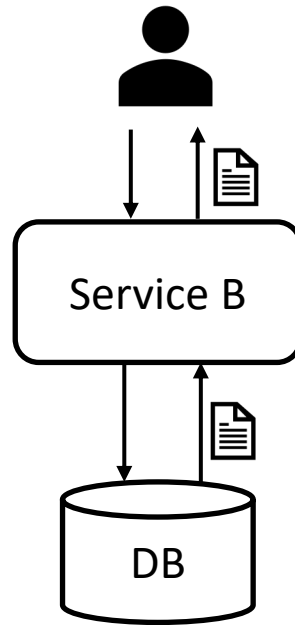
- Communication over **untrusted network**
- Increased **attack surface**
- **Establishing trust** between services

Example: auth

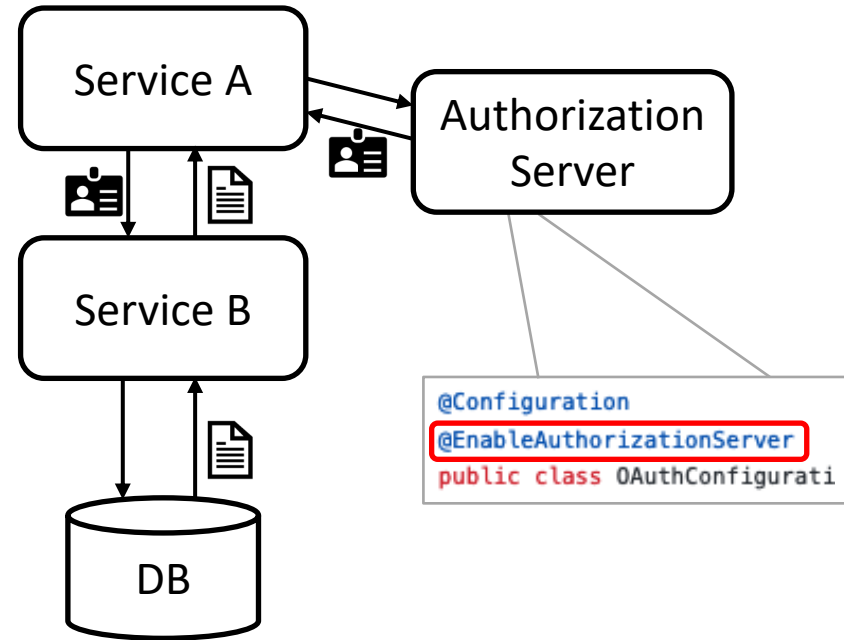
Design



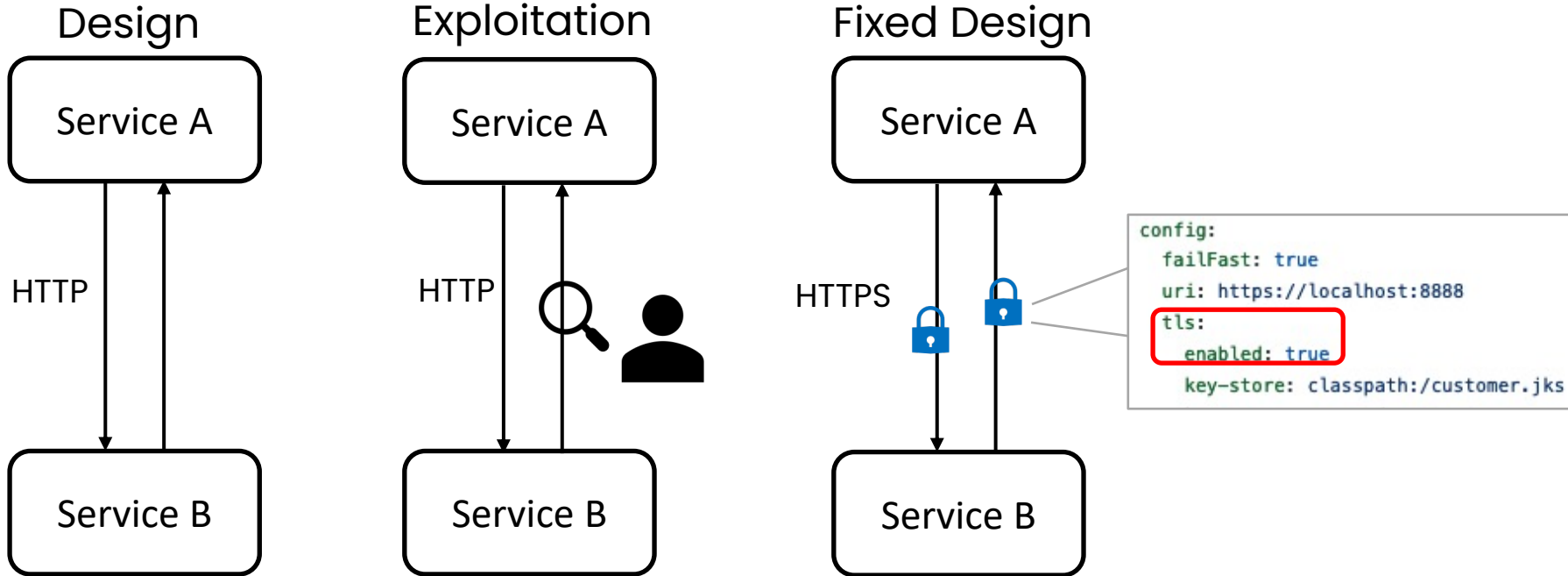
Exploitation



Fixed Design



Example: encryption



Best-practices to follow

- Best-practice **rules for secure architectural design**
 - E.g. by OWASP [1], NIST [2], or CSA [3]
- **Design and implementation guidelines** that should be followed by any microservice application
- E.g. how to set up logging in a distributed environment, how to handle authorization, etc.

[1] https://cheatsheetseries.owasp.org/cheatsheets/Microservices_security.html

[2] <https://csrc.nist.gov/publications/detail/sp/800-204/final>

[3] <https://cloudsecurityalliance.org/artifacts/best-practices-in-implementing-asecure-microservices-architecture>

Example rules

- “An API Gateway should exist as single entry point to the system and perform authorization and authentication.”
- “All communication between the services should be encrypted using secure communication protocols.”
- “A central logging subsystem which includes a monitoring dashboard should exist.”

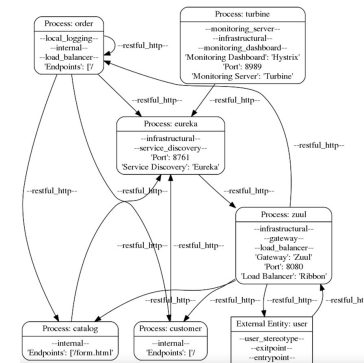
How to obtain models

- If we want to check rules on the models, we need **models correctly representing the application**
- Unambiguous **source: code** of the system

```

14  *
15  */
16
17  @EnableZuulProxy
18  @SpringBootApplication
19  public class ApiGatewayApplication {
20
21      public static void main(String[] args) {
22          SpringApplication.run(ApiGatewayApplication.class, args);
23      }
24
25      @Bean
26      public CorsFilter corsFilter() {
27          final UrlBasedCorsConfigurationSource source = new UrlBasedCorsConfigurationSource();
28          final CorsConfiguration config = new CorsConfiguration();
29          config.setAllowCredentials(true);
30          config.addAllowedOrigin("*");
31          config.addAllowedHeader("*");
32          config.addAllowedMethod("GET");
33          source.registerCorsConfiguration("/**", config);
34          return new CorsFilter(source);
35      }
36  }

```

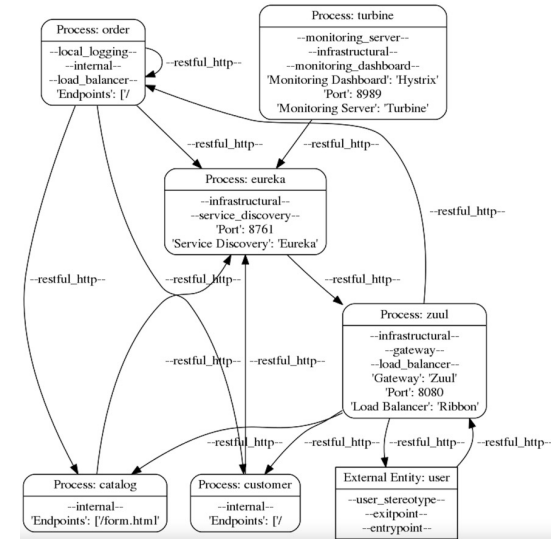




EXTRACTING DFDS FROM CODE

Dataflow Diagrams (DFDs)

- **Directed graph** (**nodes** and **edges**) depicting microservices' architecture
- Some **additional components** seen in literature, e.g. trust boundaries
- What we add:
annotations = additional information about security or other properties



Manual DFD extraction

- **Sifting through codebase** of an application to identify all relevant items
- E.g. microservices from **docker-compose file**, information flows from direct **API calls**, annotations from **config files**

```
services:
  zookeeper:
    image: wurstmeister/zookeeper:3.4.6
  kafka:
    image: wurstmeister/kafka:2.12-2.5.0
    links:
      - zookeeper
```

```
mongodb:
  host: auth-mongodb
  username: user
  password: ${MONGODB_PASSWORD}
  database: piggymetrics
  port: 27017
```

```
UserRating userRating = restTemplate.getForObject("http://ratings-data-service/ratingsdata/user/" + userId, UserRating.class);
```

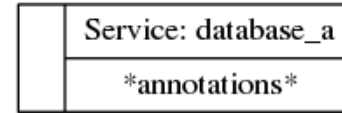
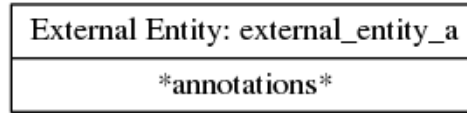
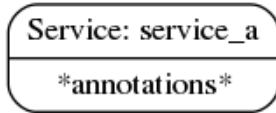
Automatic DFD extraction

- Tools can **mimic manual extraction**
- Detection of **keywords** such as specific commands in the code can reveal items in the DFD
- Our approach, based on this idea: Code2DFD [1], an automatic extractor of models for Java microservices

[1] Simon Schneider, Riccardo Scandariato, *Automatic Extraction of Security-Rich Dataflow Diagrams for Microservice Applications written in Java*. UNDER SUBMISSION: Journal of Systems and Software

Nodes in the DFDs (for microservices)

- Nodes are (micro)services, external entities, or databases
- Visualization:



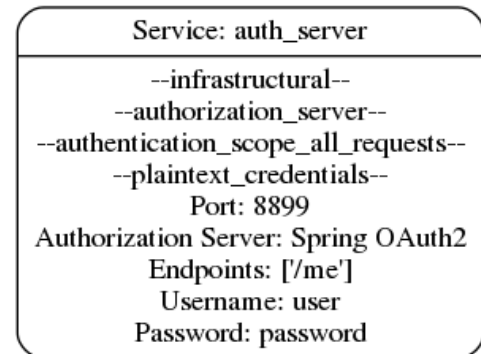
- Found mainly in build files (Maven, Gradle) and IaC files (Docker, Docker Compose)

Flows in the DFDs (for microservices)

- Information flows represent **connections between two nodes** over which some data is exchanged
- **Direct API calls** between nodes or communication from **infrastructural workings**
 - (e.g. authorization protocol, heartbeat messages, periodic logging...)
- Found in code via **direct invocations** or **implicitly** by knowing how infrastructural components operate

Additional Annotations in the DFDs

- What we do differently than others in Code2DFD
- Show additional properties of nodes and flows
- **Stereotypes** and **tagged values** (key/value-pairs)
- Stereotypes show, what a node's **functional purpose** is or what **properties** it has
- Found throughout the codebase, in code and config files





SECURITY ASSESSMENT

Technique 1: Examining Code

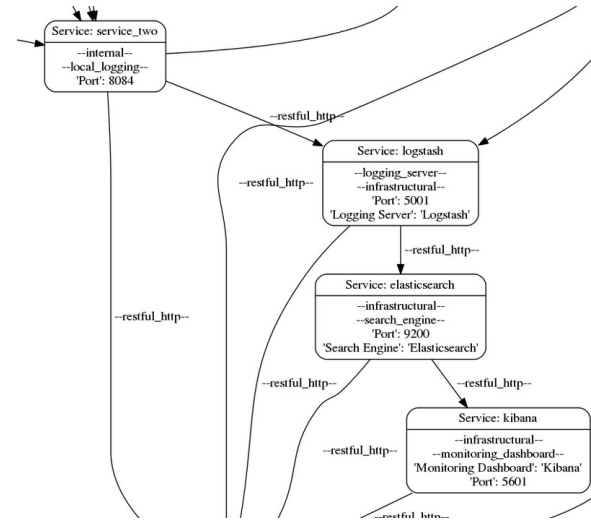
- Source code and configuration **determine the system behaviour**
- **Security properties can thus be checked directly in code**
- E.g.: “Is there hard-coded confidential data in the code?” → look for plaintext passwords

```
security.oauth2.client:  
  clientId: acme  
  clientSecret: acmesecret
```

```
ssl:  
  key-store: classpath:server.jks  
  key-store-password: password  
  key-password: password
```

Technique 2: Examining Model

- Models derived from source code depict system components and properties
- Some security properties can thus be checked in models
- E.g.: “Is logging data collected centrally?”



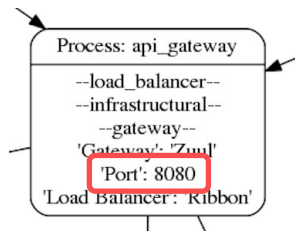
Technique 3: Examining Code and Model

- Some properties are hard to depict nuanced enough in models
- Looking at specific implementation of item found in the model might be needed to grasp details
- E.g.: “Are outgoing connections of a service guarded by a circuit breaker?”
 - Model shows a circuit breaker for the service, code reveals that not all outgoing flows are covered

Traceability Information for Technique 3

- Traceability information: **link from model item to place in code** where evidence for this item is found
- Identify relevant model item → follow traceability information → examine code for details

Model



Traceability file

```
{'item': 'api-gateway',
  'file': [...],
  'sub_items':
    {'0':
      {'item': 'Port',
        'file': *link to GitHub*,
        'line': 19,
        'span': '(8, 12)'}
```

[...]

Target of link

```
16
17  server:
18    contextPath: /
19    port: 8080
20
```



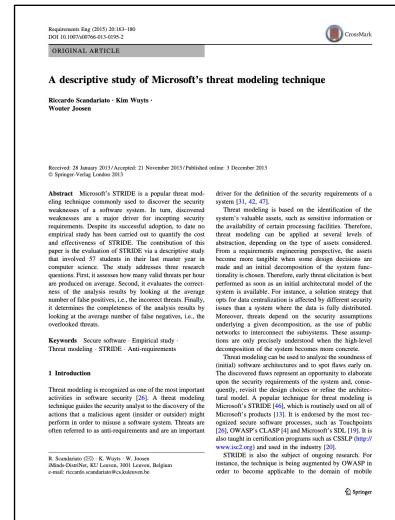
NEXT LAB SESSIONS

Research Experiment



Next two Lab Sessions

- Collect data for **scientific publication**
 - If you consent, we'll evaluate your answers **anonymously**
- Preparation for exam as usual
 - Material is **relevant for final exam**
- **Important:**
 - There are no negative or positive consequences if you participate or not ("only miss one lab"-rule still applies)



Informed Consent

- You have to **read and sign** the provided “informed consent” form
(available physically in lecture and labs plus digitally on Stud.IP)

Please do this at the end of this lecture!

(if you haven't already)

- If you don't sign, we won't collect your data. You still **have to perform the tasks** to get your bonus

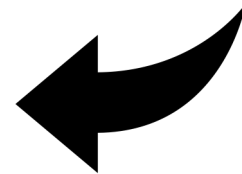
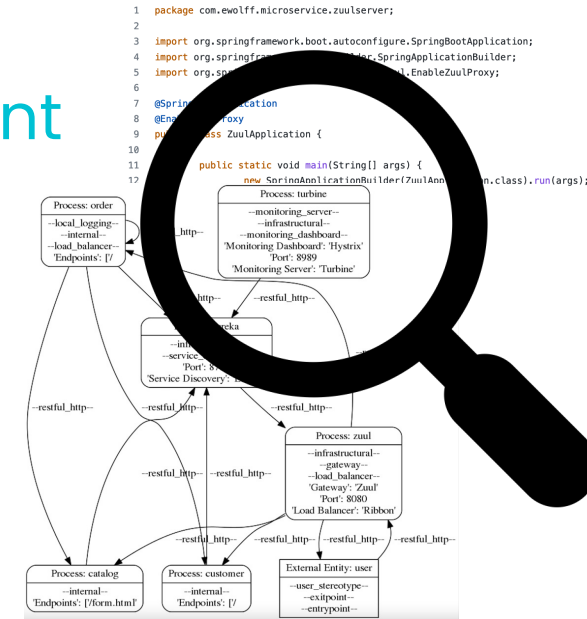
Tasks

- Perform tasks for **security assessment** of microservice applications
- Inspect code (and DFD in one week)
- Give **evidence** for your answers by referring to place in code

```

1 package com.ewolff.microservice.zuulserver;
2
3 import org.springframework.boot.autoconfigure.SpringBootApplication;
4 import org.springframework.boot.builder.SpringApplicationBuilder;
5 import org.springframework.cloud.netflix.zuul.EnableZuulProxy;
6
7 @SpringBootApplication
8 @EnableZuulProxy
9 public class ZuulApplication {
10
11     public static void main(String[] args) {
12         new SpringApplicationBuilder(ZuulApplication.class).run(args);
13     }
14
15 }

```



Material provided in the labs

- In the lab sessions, you'll receive:
 - Source code of a microservice application (on GitHub)
 - Short textual documentation of the app
 - Dataflow Diagram of the app (only in one week)
 - Traceability information corresponding to DFD (only in one week)
 - Task sheet

Process of next two labs

- You perform **similar tasks** in both weeks, but on two **different apps** and with **different material**
- One week: all material listed in previous slide
- Other week: same without the DFD
- We will tell you, what to use in each week (half of you will do the same in each week)

Important Notes

The tasks are **individual work**.

Please, **do not talk to your peers** about the tasks until the end of the study!

(which is AFTER the second week's session, on the 18th / 19th)

TODO for you

- For the next lab sessions, **you need to do** two things:
 - Read the **supplementary material**
 - Read and sign the **informed consent** sheet if not done yet
- (both available on Stud.IP, informed consent also physically here in the lecture)