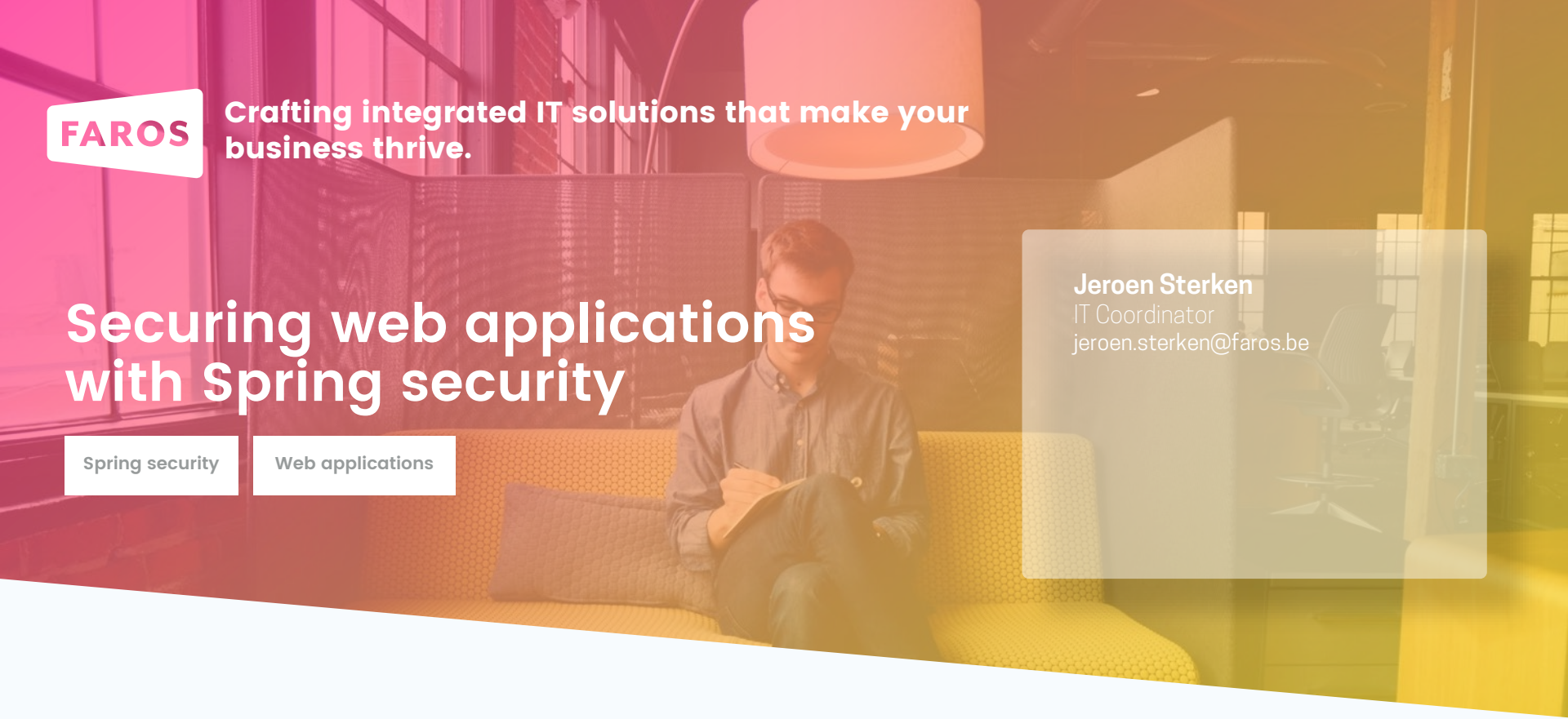


FAROS

Crafting integrated IT solutions that make your business thrive.

Securing web applications with Spring security

Spring security

Web applications

Jeroen Sterken
IT Coordinator
jeroen.sterken@faros.be

Who am i?

Jeroen Sterken

- IT Coordinator @ Faros
- Spring enthousiast
- Pivotal Spring teacher
- AFOL

Faros

we make IT fun!

IT Consultancy company - Leuven (Belgium)

Focus on Java & Spring Ecosystem

Keeping up to date is essential

No nonsense

spring ai serverless reactive

FAROS

farosbe
www.faros.be
@FarosBelgium

Agenda

Web applications security

Spring security: intro

Spring security: web

Spring security: method

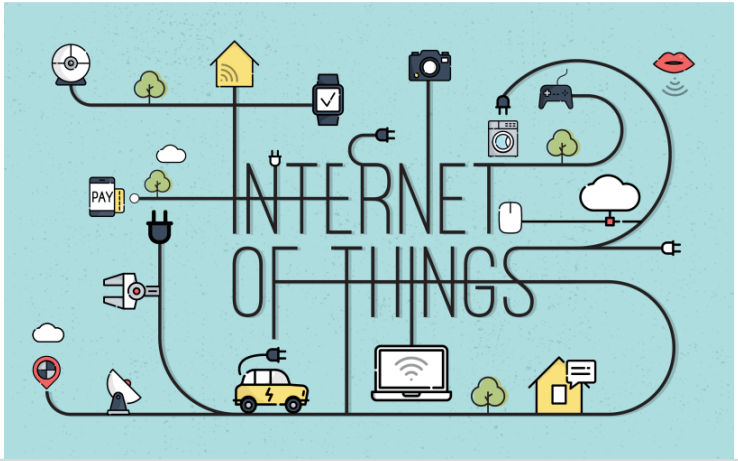
Web applications

Security

Web applications security

Sony Playstation security breach

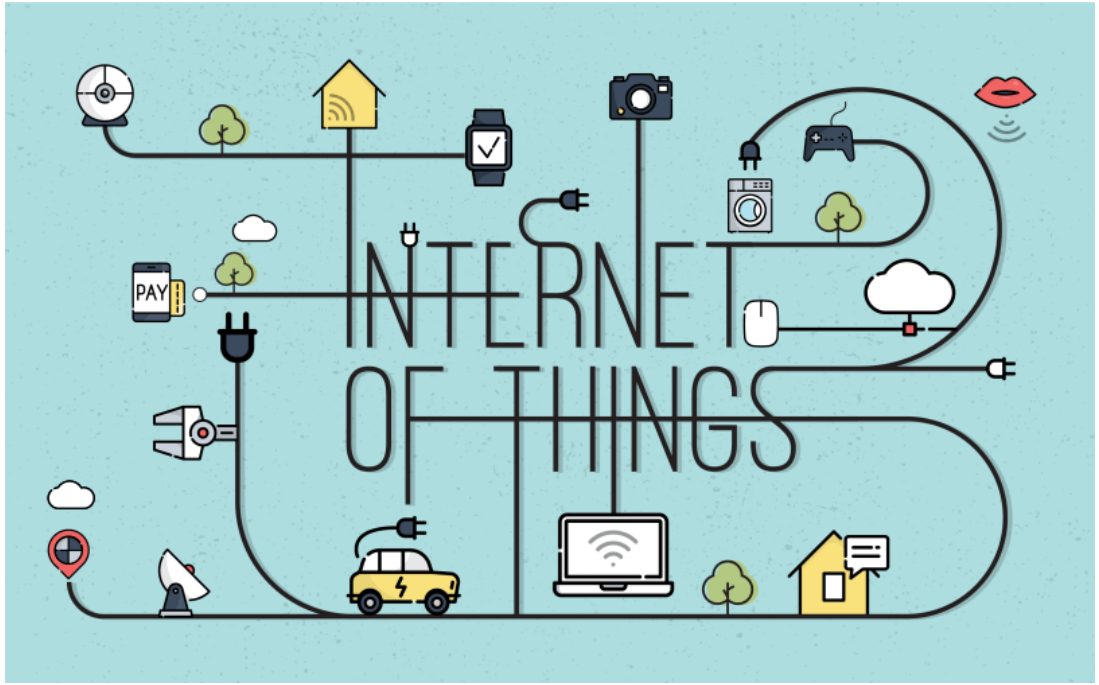
- 2011 PlayStation Network outage
- 77 million accounts were compromised (personal details/cc numbers)
- Network was out for 23 days



PlayStation Network

Web applications security

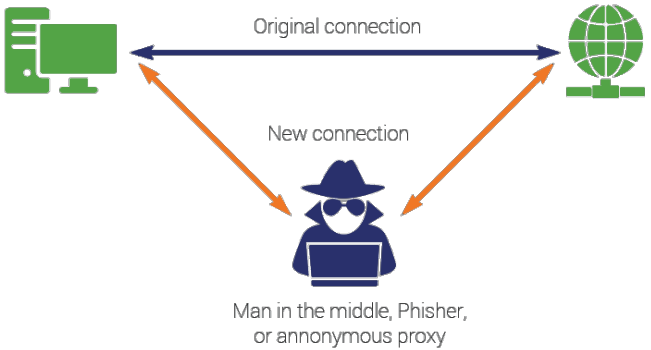
IoT



Web applications security

Security

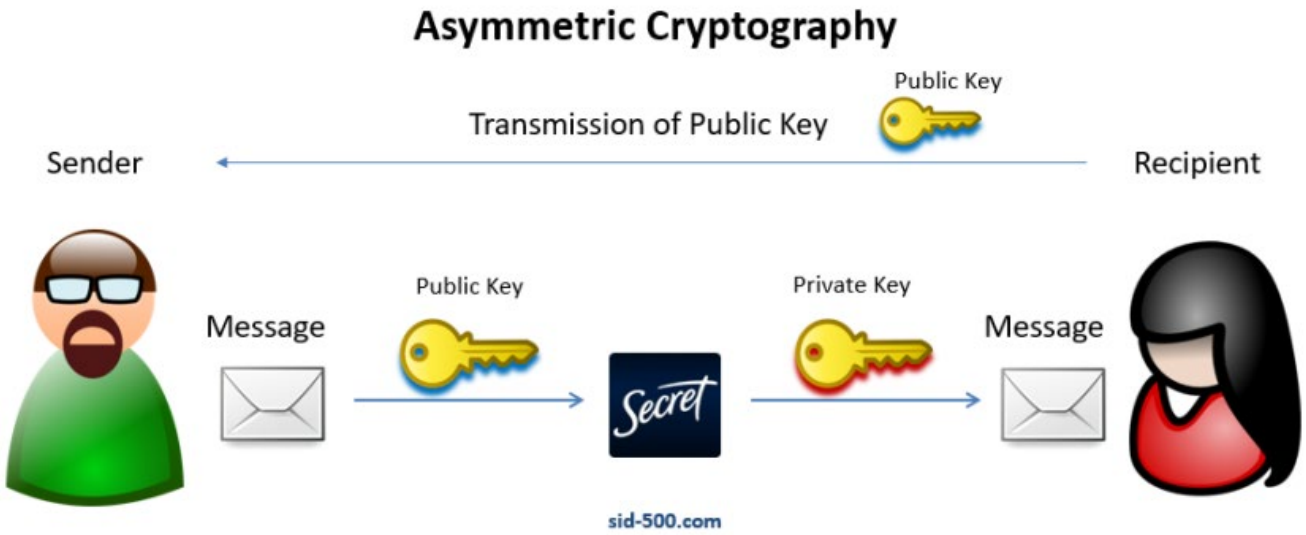
- Network
  - Ex. DDOS attacks
- Data
  - Ex. spoofing
  - HTTPS
- Other
  - Man in the middle attack



Web applications security

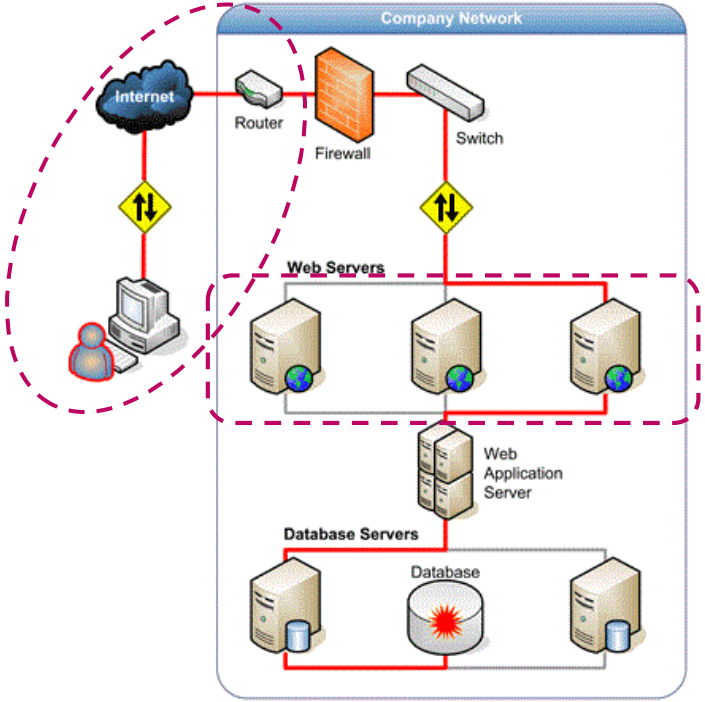
Security

- HTTP(S)



Web applications security

Security



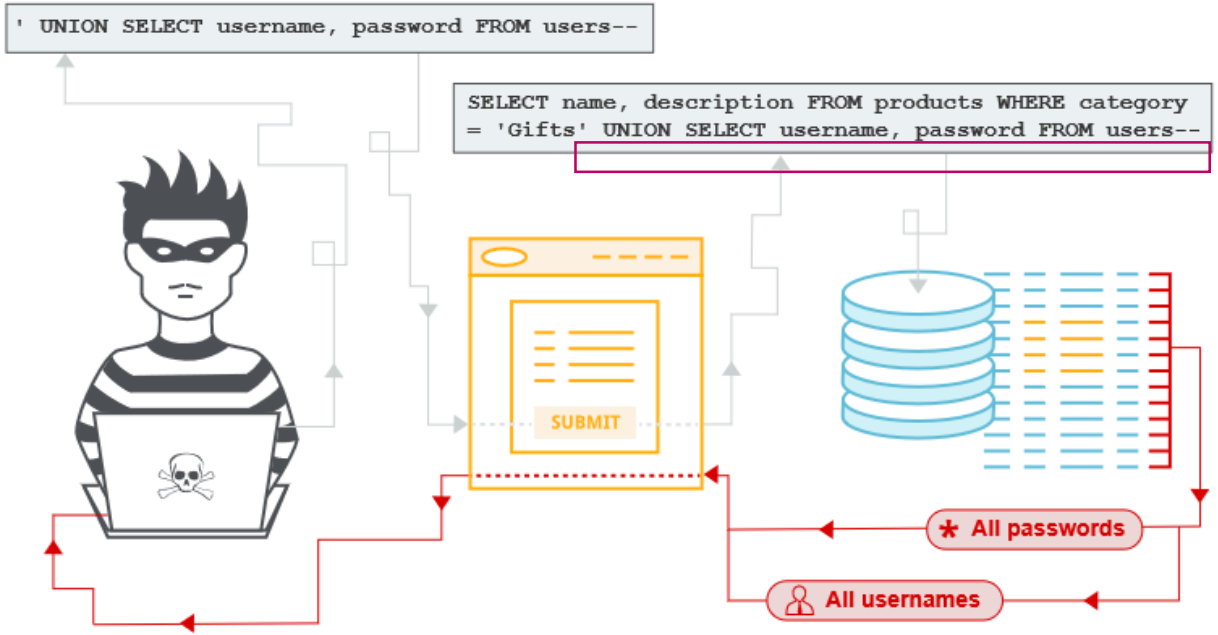
Web applications security

OWASP

- 10 Most Critical Web Application Security Risks
  https://www.owasp.org
  - A1 Injection
  - A2 Broken Authentication
  - A3 Sensitive Data Exposure
  - A4 XML External Entitites (XXE)
  - A5 Broken Access Control
  - A6 Security Misconfiguration
  - A7 Cross-Site Scripting (XSS)
  - A8 Insecure Deserialization
  - A9 Using Components with Known Vulnerabilities
  - A10 Insufficient Logging & Monitoring

Web applications security

A1 SQL Injection



Web applications security

A1 SQL Injection

```
Robert'); DROP TABLE students; --
```

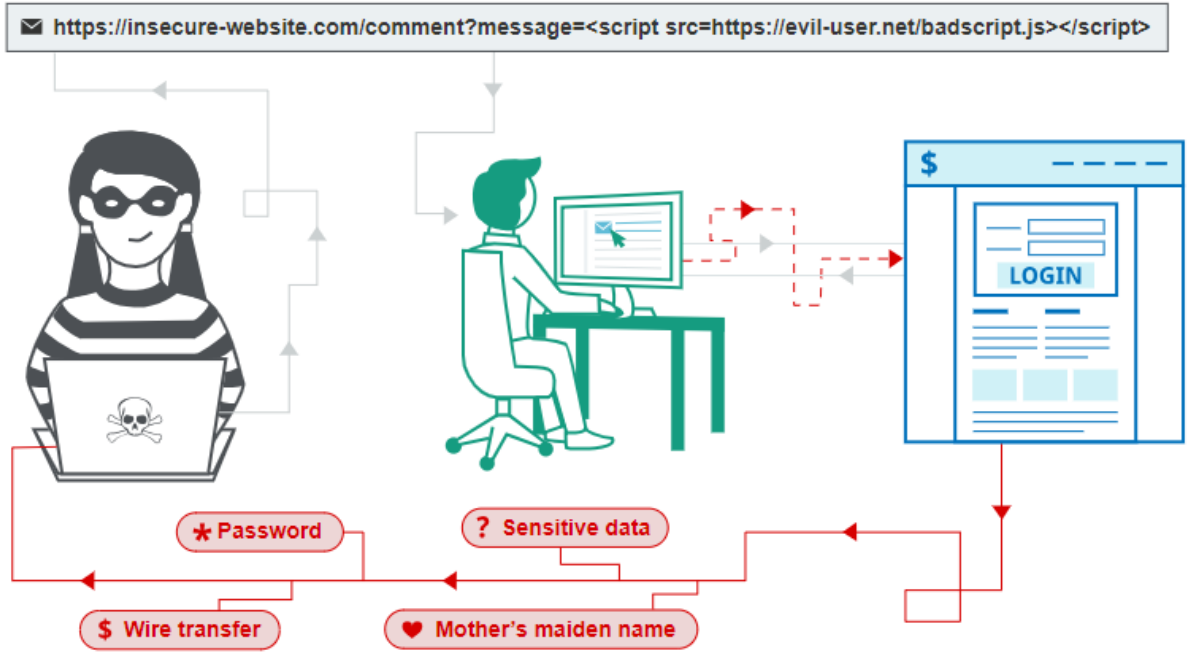
```
PreparedStatement stmt = conn.createStatement("INSERT INTO students VALUES('" + user + "')");
stmt.execute();
```

OR

```
PreparedStatement stmt = conn.prepareStatement("INSERT INTO student VALUES(?)");
stmt.setString(1, user);
stmt.execute();
```

Web applications security

A7 Cross-Site Scripting (XSS)



Web applications security

A7 Cross-Site Scripting (XSS)

- Same-origin policy
  - Browsers permit scripts contained in a first web page to access data in a second web page, but only if both web pages have the **same origin**
- Origin
  - combination of host name and port number
  - http://www.example.com/dir/page.htm
    - http://www.example.com/dir/page2.html = OK
    - http://www.example.com:81/dir/page2.html = NOT OK

Web applications security

Security frameworks

- Framework Security
  - Spring security
  - Apache Shiro

- Application Server Security
  - IBM Websphere, Oracle Weblogic, ...

Security

Overview

Security

Authentication

- Authentication mechanisms
  - Ex. Basic, Form, X.509, OAuth, …
- Storage options for credential and authority information
  - Ex. in-memory (dev), Database, LDAP, …

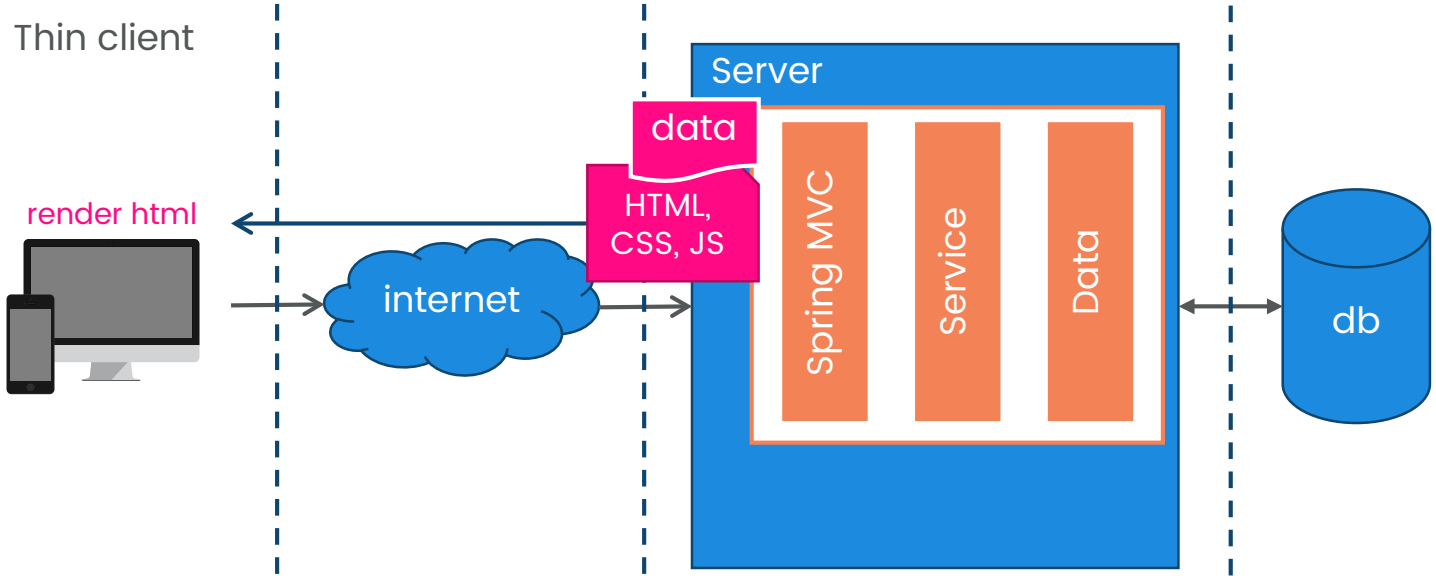
Security

Authorization

- Authorization depends on authentication
- Determines if you have the required Authority
- Often role based
  - ADMIN can cancel orders
  - MEMBER can place orders
  - GUEST can browse the catalog

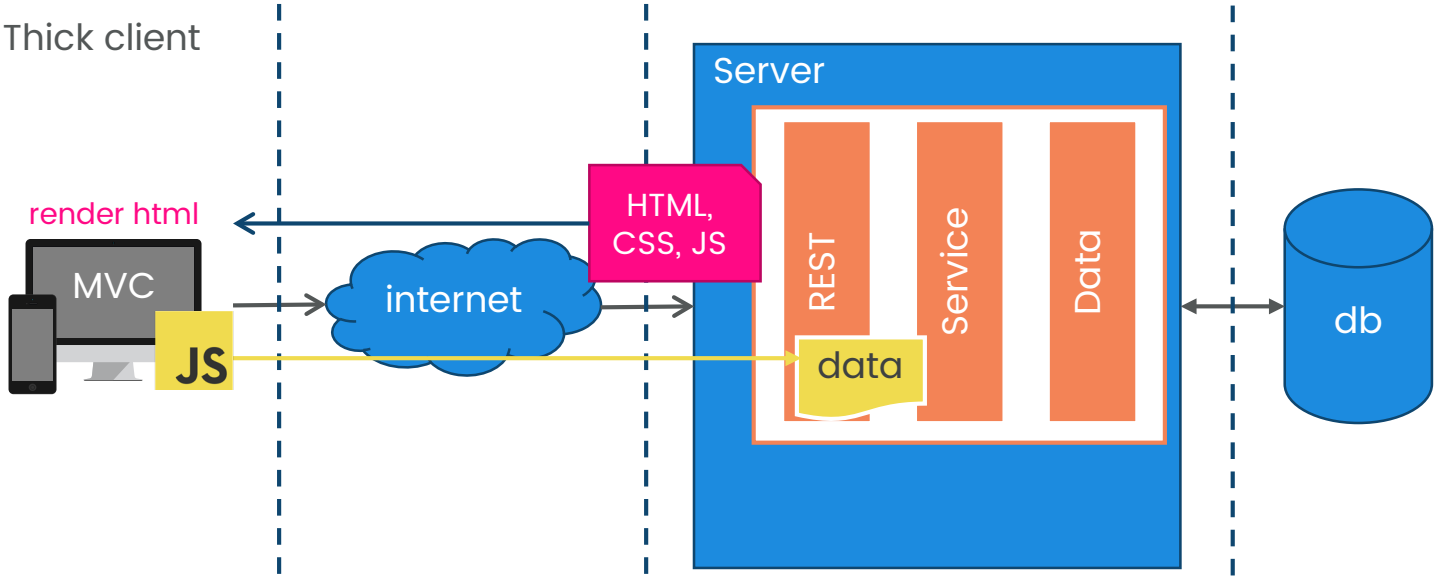
Security

Web – Thin vs Thick clients



Security

Web – Thin vs Thick clients



Spring security

Introduction

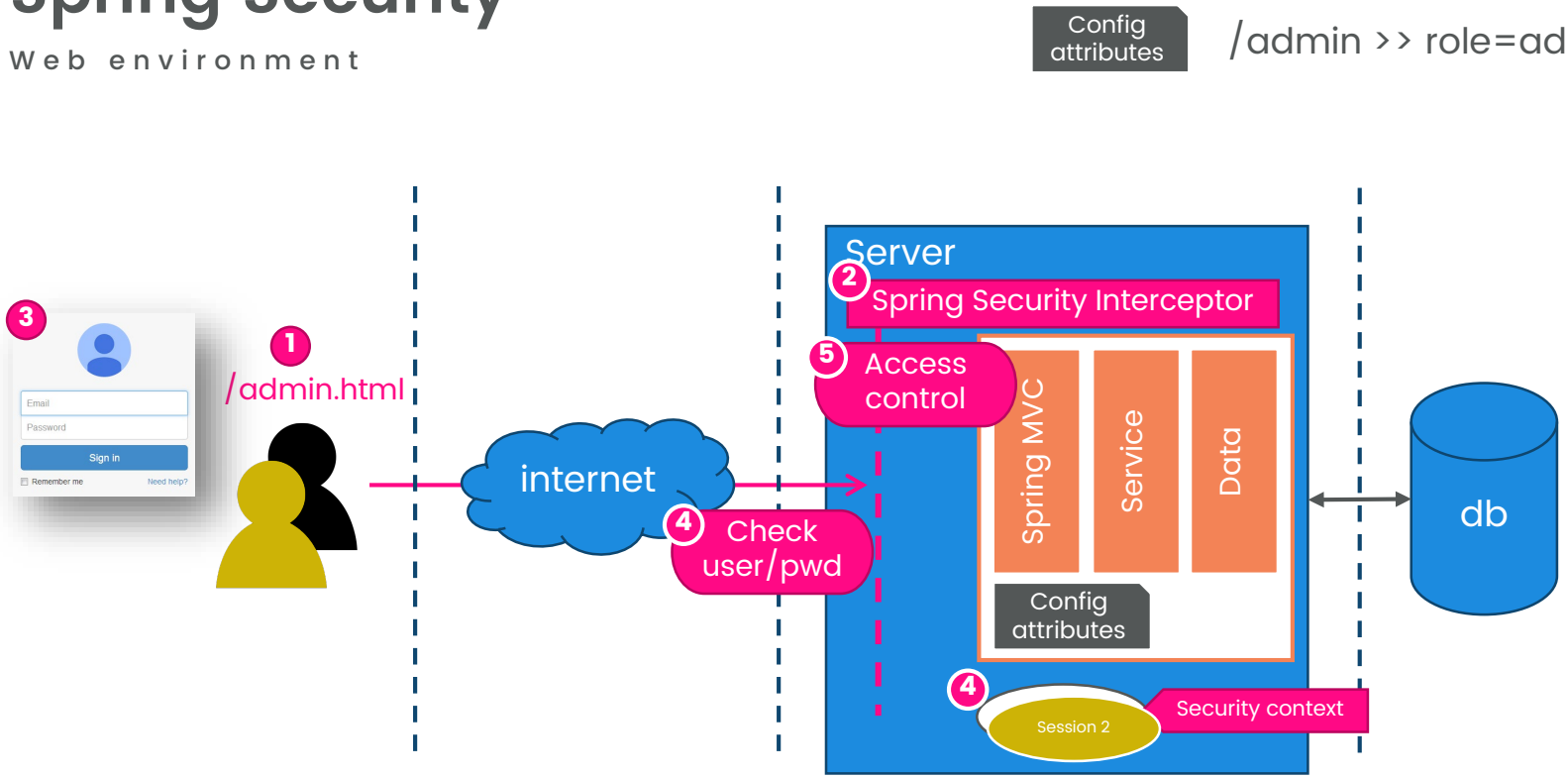
Spring security

Reasons

- Portable
  - Secured archive (JAR, WAR, EAR) can be deployed as-is
- Separation of Concerns
  - Business logic is decoupled from security concerns
  - Authentication and Authorization are decoupled
- Flexible & Extensible
  - Authentication: Basic, Form, X.509, OAuth, Cookies, Single-Sign-On, …
  - Storage: LDAP, RDBMS, Properties file, custom DAOs, …
  - Highly customizable

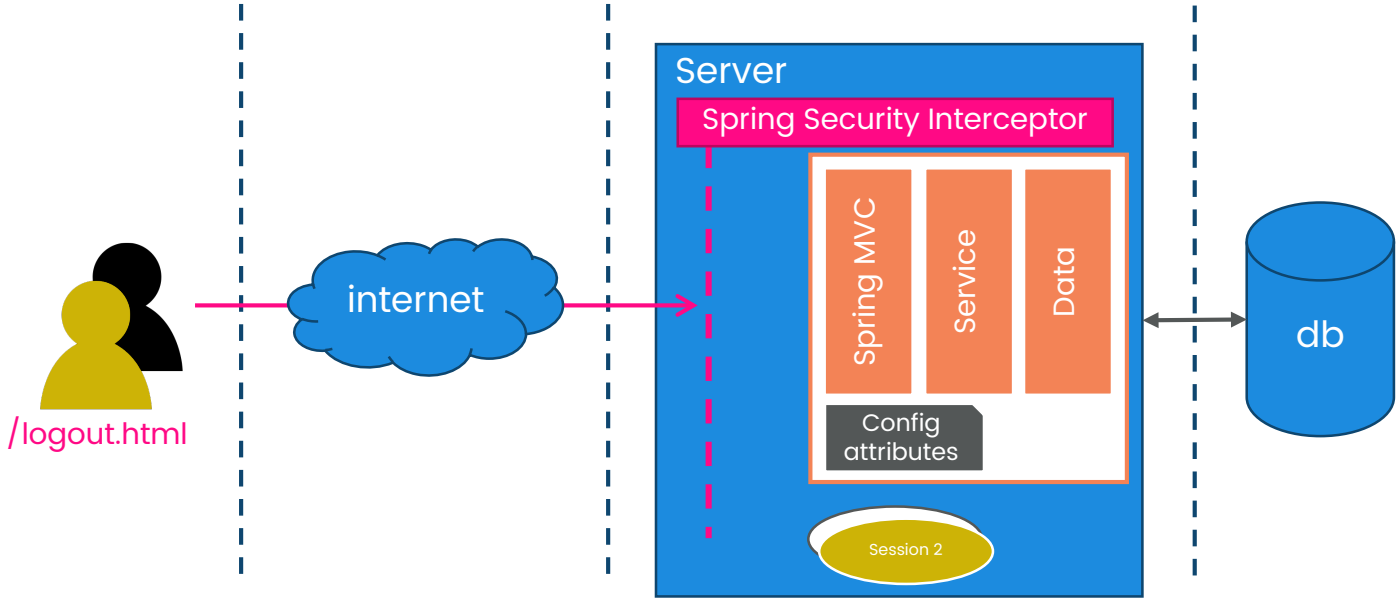
Spring Security

Web environment



Spring Security

Web environment



Spring security

Web application

Spring Security

Setup

- Three steps
  - Setup Filter chain (HTTP Servlet Filters) Spring Boot
  - Configure security (authorization) rules
  - Setup Web Authentication

Spring Security

Setup

- Spring Boot
  - Sets up a single in-memory user called “user”
  - Auto-generates a UUID password
  - All URLs require a logged-in user

Spring Security

Setup

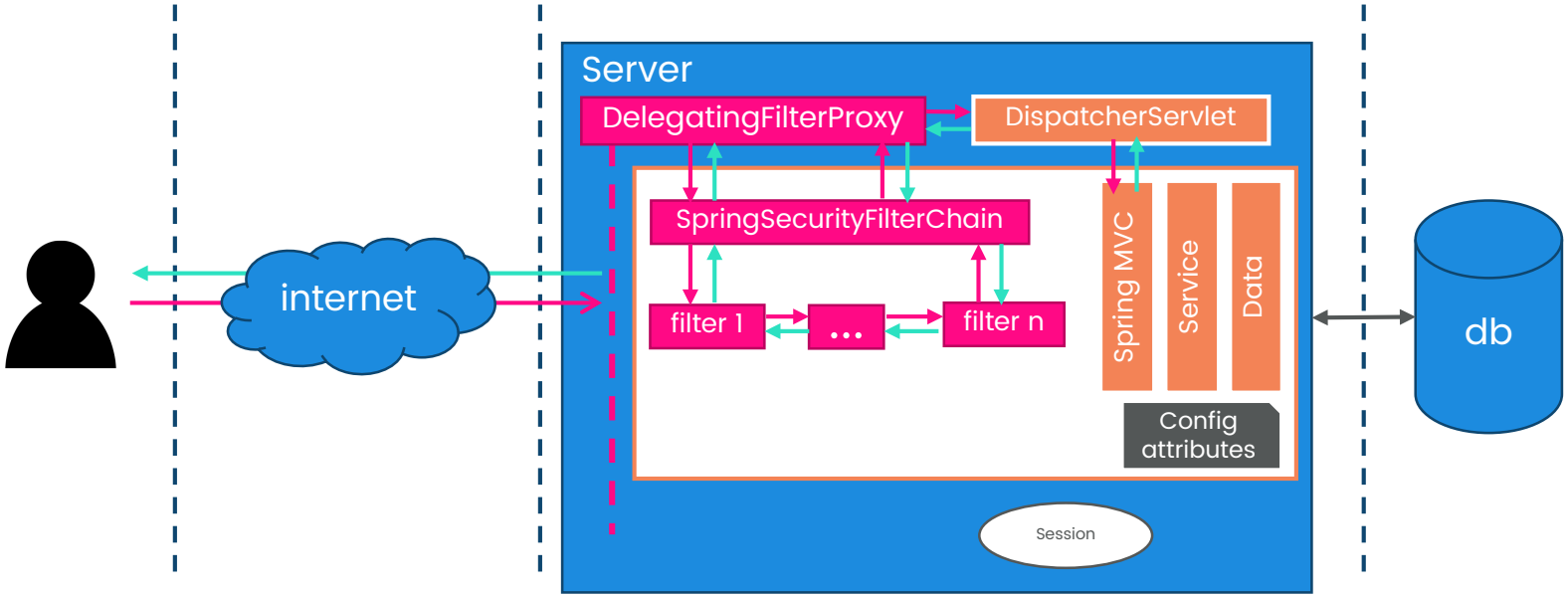
- Spring Boot
  - Maven dependency

```
<dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-security</artifactId>
</dependency>
```

  - Start application
    - Default ‘user’ & pwd via console logs

Spring Security

Setup



Spring Security

Configuration

```
@Configuration
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        // web security config
    }

    @Autowired
    public  void configureGlobal(AuthenticationManagerBuilder auth)
                throws Exception {
        // authentication manager config
    }
}
```

Spring Security

Configuration

- Define specific authorization restrictions for URLs

```
@Override
protected void configure(HttpSecurity http) throws Exception {
      http.authorizeRequests()
            .mvcMatchers("/admin/**").hasRole("ADMIN")
            …
}
```

Spring Security

Configuration

- Chain multiple restrictions
  - First match is used, put specific matches first

```
@Override
protected void configure(HttpSecurity http) throws Exception {
  http
    .authorizeRequests()
       .mvcMatchers("/signup", "/about").permitAll()
       .mvcMatchers("/persons/delete*").access("hasRole('ADMIN')
                                              && hasRole('USER')")
       .mvcMatchers("/persons/edit*").hasRole("ADMIN")
       .mvcMatchers("/persons/**").hasAnyRole("USER",”ADMIN”)
       .anyRequest().authenticated();
```

SpeL

Spring Security

Configuration

```
@Configuration
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        // web security config
    }

    @Autowired
    public  void configureGlobal(AuthenticationManagerBuilder auth)
                throws Exception {
        // authentication manager config
    }
}
```

Spring Security

Configuration

- Example of inMemory UserDetailsService

```
@Autowired
public void configureGlobal(Authenti                Builder auth) throws
Exception {
    auth
        .inMemoryAuthentication()
       .withUser(“jan").password(“{noop}pwd1").roles(“USER").and()
       .withUser(“erik").password("{noop}pwd2").roles(“ADMIN");
}
```

Do NOT use in production!

Spring Security

Configuration

- Database

```
private DataSource dataSource;

@Autowired
public void setDataSource(DataSource dataSource) throws Exception {
    this.dataSource = dataSource;
}

@Autowired
public void configureGlobal(AuthenticationManagerBuilder auth) throws
Exception {
    auth.jdbcAuthentication()
        .dataSource(dataSource);
}
```

Spring Security

Configuration

- Database
  - Queries DB for users and roles
  - Default queries

```
SELECT username, password, enabled FROM users WHERE username = ?
SELECT username, authority FROM authorities WHERE username = ?
```

  - Customize queries

```
auth.jdbcAuthentication()
    .usersByUsernameQuery(<custom-query>)
    .authoritiesByUsernameQuery(<custom-query>)
    .dataSource(dataSource);
```

Spring Security

Configuration

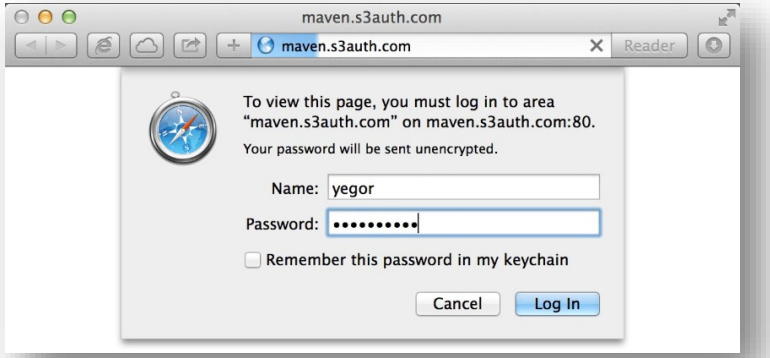
- Password encoding
  - BCrypt is recommended over SHA-256
    - Secure passwords further by specifying a “strength” (N)
    - Internally the hash is rehashed 2^N times, default is 2^{10}

```
auth.inMemoryAuthentication()
    .passwordEncoder(new BCryptPasswordEncoder(12));
    .withUser(“jan")
    .password("$2y$12$PZRRdjKywU..cTvobTIGU31aFNPEiC6MfPZtm")
    .roles("GENERAL")
```

Spring Security

Configuration

- Login
  - Example: HTTP Basic




```
protected void configure(HttpSecurity http) throws Exception {
  http
    .authorizeRequests()
        .mvcMatchers("/home/**").hasRole("USER")
        ...
    .and()
       .httpBasic(); // enable HTTP Basic
}
```

Spring Security

Configuration

- Login
  - Form-based

```
protected void configure(HttpSecurity http) throws Exception {
  http.authorizeRequests()
        .mvcMatchers("/home/**").hasRole("USER")...
        .and()
    .formLogin()                    //  setup form-based authentication
        .loginPage("/login")        //  URL to use when login is needed
        .permitAll()                //  any user can access
        .and()
    .logout()                       //  configure logout
        .logoutSuccessUrl("/home")  //  go here after successful logout
        .permitAll();               //  any user can access
}
```

Spring Security

Configuration

- Login
  - Example login page

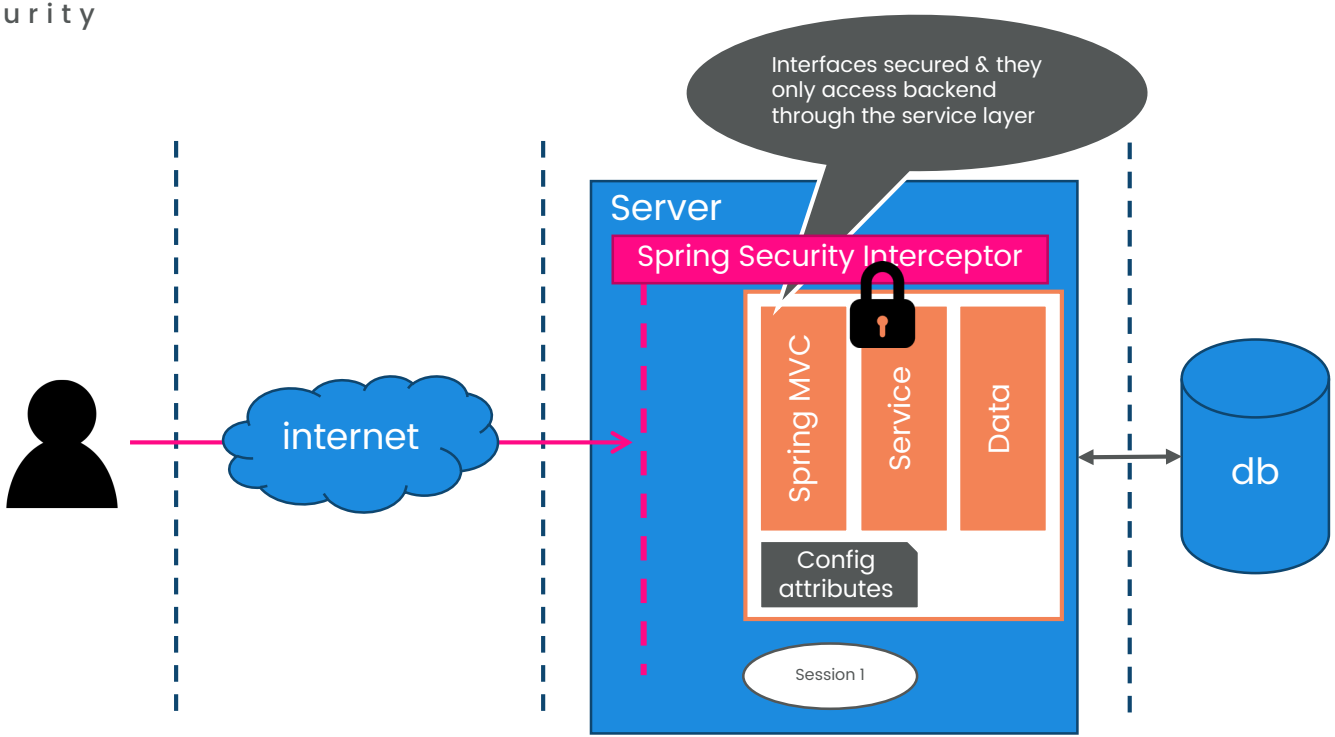
```
<form action="/login" method="POST">
   <input type="text" name="username"/>
   <input type="password" name="password"/>

   <input type="submit" name="submit" value="LOGIN"/>
</form>
```

Spring security

Method security

Spring Security

Method security



Spring Security

Method security

- JSR-250 standard
  - Only supports role-based security
  - Configuration

```
@EnableGlobalMethodSecurity(jsr250Enabled=true)
```

  - Code

```
import javax.annotation.security.RolesAllowed;

public class ItemManager {
   @RolesAllowed("ROLE_MEMBER")
   public Item findItem(long itemNumber) {
      ...
   }
}
```

Internally role authorities are stored with **ROLE_** prefix.
APIs seen previously hide this. Here you *must* use full name

Spring Security

Method security

- Spring's annotation (supports SpEL)
  - Configuration

```
@EnableGlobalMethodSecurity(prePostEnabled=true)
```

  - Code

```
import org.springframework.security.annotation.PreAuthorize;

public class ItemManager {
   // Members may only find their own order items
   @PreAuthorize("hasRole('MEMBER')   &&  " +
                 "#order.owner.id == principal.user.id")
   public Item findItem(Order order, long itemNumber) {
      ...
   }
}
```

Full role-names not required. ROLE_ prepended automatically

Spring boot actuator

Security

Spring Boot Actuator

Security

- Spring boot actuator
  - Monitoring app
  - Gathering metrics
  - Understanding traffic
  - State of our database
  - ...
- Exposed as HTTP endpoints or JMX beans

Spring Boot Actuator

Security

- Getting started
  - Maven dependency

```
<dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
```

  - Most endpoints (/actuato/…) disabled by default except
    - /actuator/health
    - /actuator/info

Spring Boot Actuator

Security

- HTTP endpoints

  /auditevents – lists security audit-related events such as user login/logout.
  /beans – *shows* all available beans in our *BeanFactory*
  /conditions – formerly known as */autoconfig*, builds a report of conditions around auto-configuration
  /configprops – allows us to fetch all *@ConfigurationProperties* beans
  /env – returns the current environment properties. Additionally, we can retrieve single properties
  /flyway – provides details about our Flyway database migrations
  /health – summarises the health status of our application
  /heapdump – builds and returns a heap dump from the JVM used by our application
  /info – returns general information.
  /liquibase – behaves like */flyway* but for Liquibase
  /logfile – returns ordinary application logs
  /loggers – enables us to query and modify the logging level of our application
  /metrics – details metrics of our application.
  /prometheus – returns metrics like the previous one, but formatted to work with a Prometheus server
  /scheduledtasks – provides details about every scheduled task within our application
  /sessions – lists HTTP sessions given we are using Spring Session
  /shutdown – performs a graceful shutdown of the application
  /threaddump – dumps the thread information of the underlying JVM

Spring Boot Actuator

Security

- Access
  - Enable and expose endpoints
  - Enable
    - All endpoints enabled by default except: /shutdown are enabled

      ```
      management.endpoint.shutdown.enabled=true
      ```
  - Expose
    - Only /health & /info endpoints are exposed by default

      ```
      management.endpoints.web.exposure.include=*
      ```
    - Expose all enabled endpoints except one (ex. /loggers)

      ```
      management.endpoints.web.exposure.include=*
      management.endpoints.web.exposure.exclude=loggers
      ```

Spring security

Resources

Spring security

Resources

- Securing a Web Application (Spring MVC / thymeleaf)
  https://spring.io/guides/gs/securing-web
- Spring Security Architecture
  https://spring.io/guides/topicals/spring-security-architecture/
- Spring Security and Angular
  https://spring.io/guides/tutorials/spring-security-and-angular-js/

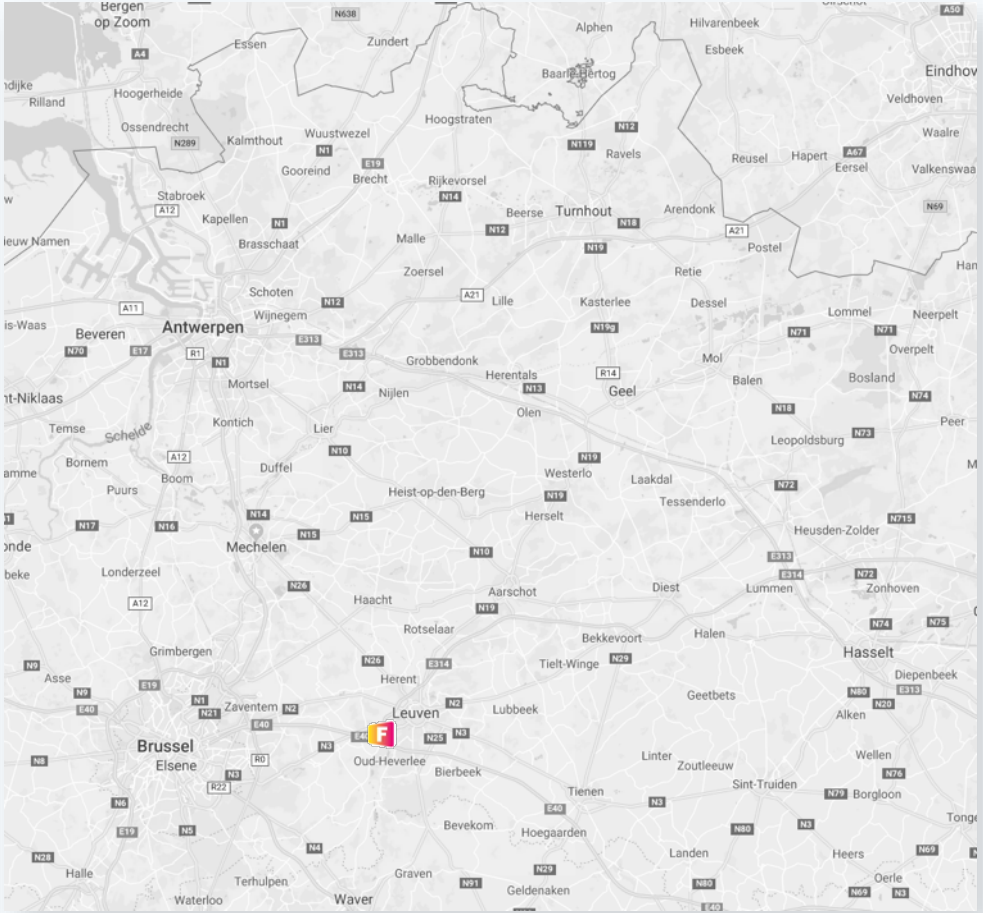
Contact

Gaston Geenslaan 11 B4
3000 Leuven

hello@faros.be

Jeroen Sterken
jeroen.sterken@faros.be

@FarosBelgium farosbe



FAROS

Adres gegevens
info@faros.be