Spring Boot

Validation

Rules and Regulations

- Three Types of Activities
  - Presentation
  - Live-Coding
  - *Little* assignments
- Questions welcome!
- Regular Breaks
  - After **Presentations**: 5 min

Validation

What is validation and why do we need this?

amazon.de

Konto erstellen

Ihr Name

Andreas Steffens

Mobiltelefonnummer oder E-Mail-Adresse

ast@interactive-pioneers

Passwort

•••••

ℹ Passwörter müssen mindestens 6 Zeichen lang sein.

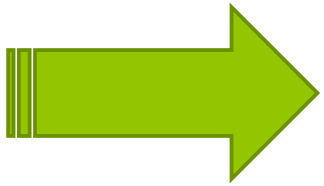
Passwort nochmals eingeben

••••••

Weiter

Mit Ihrer Anmeldung erklären Sie sich mit unseren Allgemeinen Geschäftsbedingungen einverstanden. Bitte lesen Sie unsere Datenschutzerklärung, unsere Hinweise zu Cookies und unsere Hinweise zu interessenbasierter Werbung.

Sie haben bereits ein Konto? Anmelden ›
Kaufen Sie für Ihr Unternehmen ein? Erstellen Sie ein kostenloses Unternehmenskonto



amazon.de

Konto erstellen

Ihr Name

Andreas Steffens

Mobiltelefonnummer oder E-Mail-Adresse

ast@interactive-pioneers

! Ungültige E-Mail-Adresse oder Mobiltelefonnummer Korrigieren Sie das Problem, und wiederholen Sie den Vorgang.

Passwort

•••••

! Mindestens 6 Zeichen erforderlich

Passwort nochmals eingeben

••••••

! Die Passwörter stimmen nicht überein

E-Mail-Adresse bestätigen

Mit Ihrer Anmeldung erklären Sie sich mit unseren Allgemeinen Geschäftsbedingungen einverstanden. Bitte lesen Sie unsere Datenschutzerklärung, unsere Hinweise zu Cookies und unsere Hinweise zu interessenbasierter Werbung.

Sie haben bereits ein Konto? Anmelden ›
Kaufen Sie für Ihr Unternehmen ein? Erstellen Sie ein kostenloses Unternehmenskonto

Top 10:2021 List

A01 Broken Access Control

A02 Cryptographic Failures

A03 Injection

A04 Insecure Design

A05 Security Misconfiguration

A06 Vulnerable and Outdated Components

A07 Identification and Authentication Failures

A08 Software and Data Integrity Failures

A09 Security Logging and Monitoring Failures

A10 Server Side Request Forgery (SSRF)

Next Steps

A03:2021 – Injection

Factors

CWEs Mapped	Max Incidence Rate	Avg Incidence Rate	Avg Weighted Exploit	Avg Weighted Impact	Max Coverage	Avg Coverage
33	19.09%	3.37%	7.25	7.15	94.04%	47.90%

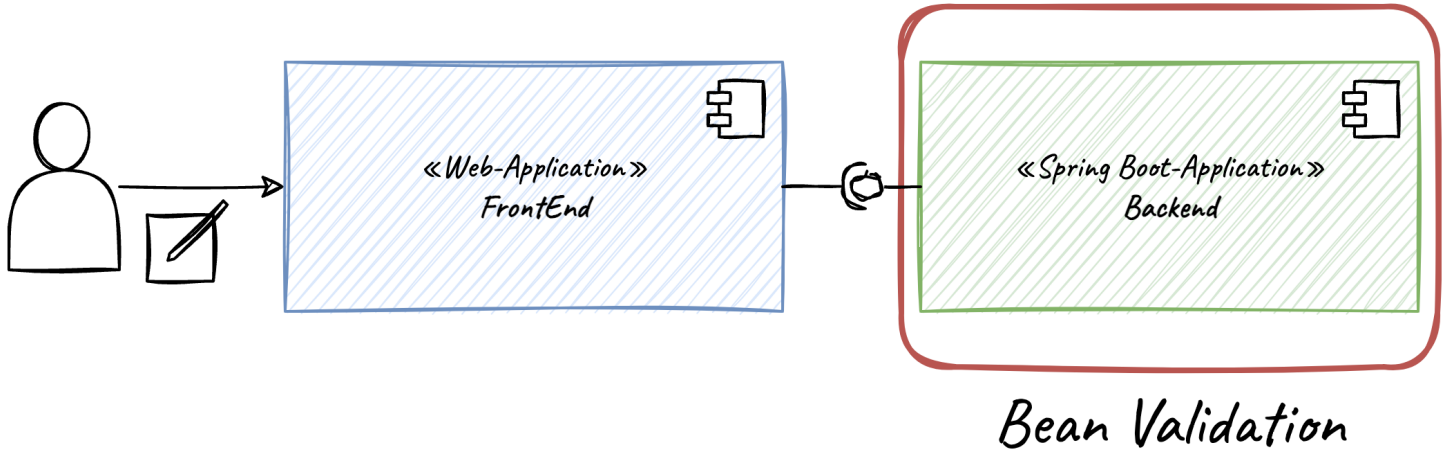
Overview

Injection slides down to the third position. 94% of the applications were tested for some form of injection with a max incidence rate of 19%, an average incidence rate of 3%, and 274k occurrences. Notable Common Weakness Enumerations (CWEs) included are *CWE-79: Cross-site Scripting*, *CWE-89: SQL Injection*, and *CWE-73: External Control of File Name or Path*.

Description

An application is vulnerable to attack when:

- User-supplied data is not validated, filtered, or sanitized by the application.
- Dynamic queries or non-parameterized calls without context-aware escaping are used directly in the interpreter.
- Hostile data is used within object-relational mapping (ORM) search parameters to extract additional, sensitive records.
- Hostile data is directly used or concatenated. The SQL or command contains the structure and malicious data in dynamic queries, commands, or stored procedures.


«Web-Application»
FrontEnd
«Spring Boot-Application»
Backend
Bean Validation

Jakarta Bean Validation

Bean Validation API

- Bean Validation API 2.0/3.0 is a Jakarta EE Standard
  - https://beanvalidation.org/
- Standardized API for validating any kind of data
- Reference Implementation: Hibernate Validator

Bean Validation

- Define constraints with annotations
- Generic validation API via validator class
- Violations reporting with ContraintViolation

Contraints

```
public class Person {

   @NotNull
   @Size(max=256)
   String firstName;

   @Size(max=256)
   String middleName;

   @NotNull
   @Size(max=256)
   String lastName;

   @Past
   LocalDate birthday;

   @Valid
   @NotNull
   Address address;

   @NotNull
   @Valid
   ContactInformation contact;

   @PastOrPresent
   LocalDate creationDate;

}
```

```
public class Address {

   @NotNull
   String street;

   @NotNull
   @Positive
   Integer housenumber;

   @NotNull
   @Min(10000)
   @Max(99999)
   Integer postalCode;

   @NotNull
   String city;

   @NotNull
   String state;

   @NotNull
   String country;


}
```

Build-In Constraints

- ***@NotNull***
- ***@NotBlank***
- ***@NotEmpty***
- ***@Size***
- ***@Min***
- ***@Max***
- ***@Email***
- ***@AssertTrue***
  @AssertFalse
- ***@Positive***
  @PositiveOrZero
- ***@Negative***
  @NegativeOrZero
- ***@Past***
  @PastOrPresent
- ***@Future***
  @FutureOrPresent

Contraint properties

- Values

```
@Size(max=256)
String firstName;
```

- Message

```
@Min(10000, message = "This is not valid
german postal code")
Integer postalCode;
```

- Groups

```
@Past(groups = ContractContext.class)
LocalDate birthday;
```

Contraint application

- Field

```
@Size(max=256)
String firstName;
```

- Property/Bean

```
@Min(10000, message = "This is not valid german postal code")
public Integer getPostalCode() {
   return postalCode;
}
```

- Class-level

```
@AddressExists
public class Address { … }
```

Inheritance & Composition

- Constraints defined in super-class are inherited by child-classes
- Object graphs: constraint validation can be delegated: @Valid

```
public class Person {
…

   @Valid
   @NotNull
   Address address;
…
}
```

Validating Constraint

- Central class: Validator
- Factory: ValidatorFactory

```
void setUp() {
    ValidatorFactory factory = Validation.buildDefaultValidatorFactory();
    validator = factory.getValidator();
}
```

Validation with Validator

```
Person p1 = new Person();
p1.setFirstName("Andreas");
p1.setLastName("Steffens");
p1.setCreationDate(LocalDate.now());

Set<ConstraintViolation<Person>> violations = validator.validate(p1);
```

ContraintViolation

- Validator returns a ContraintViolation for each failed constraint during validation

- Important properties
  - getMessage(): returns message defined in Constraint
  - getPropertyPath(): returns failed property
  - getInvalidValue(): return the given invalid value

Assignment 1: Bean Validation

Insurance

- An insurance company wants to provide an API, where customers can register and calculate the rates for different insurance policies

- Create classes for representing a customer with peronal, address and contact information

Insurance

- Create a Spring Boot project with **Spring Initializer**
  - **https://start.spring.io**
  - **group: de.neuefische.java**
  - **artifact: beanvalidation**
- Include the necessary dependencies
  - Testing
  - Validation
  - REST/Web API
  - **No database**

- Personal Information
  - FirstName, LastName, MiddleName, DataOfBirth, Nationality, Languages, Gender
- Adress Information
  - Street,Housenumber, ZipCode, Country, State
- Contact Information
  - PhoneNumber, Email, Website/SocialProfile

1. Define a appropiate class structure
2. Define Constraints for:
    1. Valid Person: valid FirstName, valid LastName, valid **German** PhoneNumber, valid DateOfBirth, valid CreationDate, valid german accountNumber
    2. If Email is provided it needs to be a valid email
    3. If Address is provided, the address data needs to be complete (all data set and valid)
3. Define additonal constraints for as many properties as you like.

4. Provide JUNIT5 Tests for each class and defined constraints. Define a positve and negative test.
5. log violation messages into console with detailed information which constraint failed
6. Adapt Constraint messages to provide better messages

Bean Validation in Spring

Bean Validation in Spring

- Bean Validation API 2.0 is deeply integrated into Spring, Spring MVC and Spring Boot

1. Annotate Domain Objects/DTO with Constraints
2. Activate Validation on important incoming APIs

Example for RESTful API

```
@RestController("/insurance")
@Validated
public class InsuranceController {
    @PostMapping
    public Person createContact(@RequestBody @Valid Person person){
        // ... store Person in database
    }
}
```

@Validated & @Valid

- @Validated: class-level annotation to activate validation for controller methods
- @Valid: method-parameter annotation to activate validation of specific input objects

Contraints on parameters

```
@GetMapping(path = "/{lastname}")
public String getContact(@PathVariable(name = "lastname")
@Size(max = 25) String name){
    // ... fetch Person in database
    return name;
}
```

Assignment 2: Insurance API

1. Create a RestController in you application which provides CRUD-functions for insurance customers
2. **HINT: no database needed, methods can be empty …**
3. Validate the given input data
4. Create Tests (e.g. with @WebMvcTest and MockMvC) to prove your validation is applied

Custom Constraints in Spring

Custom Constraints

- Complex validations

- Needed Components:
  - Custom annotation
  - Implementation of ConstraintValidator<Annotation,ObjectClass>

```
@Target({FIELD})
@Retention(RUNTIME)
@Constraint(validatedBy = IBANValidator.class)
@Documented
public @interface GermanIBAN {

    String message() default "is no valid IBAN";

    Class<?>[] groups() default {};

    Class<? extends Payload>[] payload() default {};


}
```

```
public class IBANValidator implements
ConstraintValidator<GermanIBAN,String> {
    @Override
    public boolean isValid(String s, ConstraintValidatorContext
constraintValidatorContext) {
        return s.startsWith("DE") && s.length() == 22;
    }
}
```

Advanced Validation

Bean Validation

- Possible Validation will be exported to OpenAPI Schema


```
"components": {
    "schemas": {
        "Image": {
            "required": [
                "location",
                "name"
            ],
            "type": "object",
            "properties": {
                "id": {
                    "type": "integer",
                    "format": "int64"
                },
                "name": {
                    "maxLength": 64,
                    "minLength": 4,
                    "type": "string"
                },
                "location": {
                    "type": "string",
                    "format": "url"
                },
                "publishedAt": {
                    "type": "string",
                    "format": "date"
                },
                "lastModified": {
                    "type": "string",
                    "format": "date"
                }
            }
```

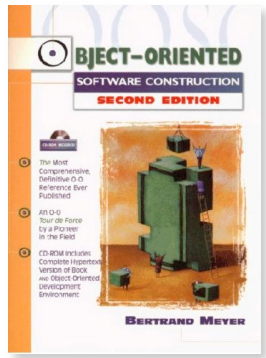
Property-based validation

```
public @Positive Integer getAge(){
   return  Period.between(birthday,LocalDate.now()).getYears();
}
```

Design-by-Contract in Spring

Design by Contract

- The same idea, on a per-method / per-class basis
- Pre-conditions
  - Input requirements
  - Caller's promise to the method
- Post-conditions
  - Output requirements
  - Method's promise to the caller

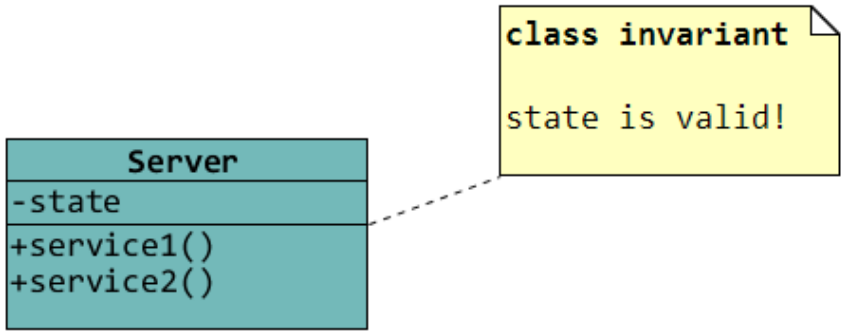


Bertrand Meyer, Object-Oriented Software Construction, Prentice Hall, 1997.

Pre- and Post-conditions

- The pre-condition obligates the client!
  - Defines what is required for a call to the operation to be legitimate
  - Typically involves
    - initial state - before execution
    - Parameters
- The post-condition obligates the server!
  - Defines the conditions that the server ensures on return
  - Typically involves
    - initial state
    - final state - after execution-
    - parameters
    - result

Class Invariant



- Invariant
  - A predicate that must hold at certain points in the execution of a program
- Class invariant
  - An invariant the characterizes the valid states of its objects
  - It must hold
    - after construction of each object (ensured by each constructor)
    - before and after the execution of every public method

Design by Contract

- When you design a class, each public method M must specify a clear contract!
  - Pre-condition + class invariant
  - Post-condition + class invariant

> *"If you promise to call M with the precondition satisfied, then M promises to deliver a final state in which the post-condition is satisfied."*

- Implication
  - If the pre-condition does not hold, the server is not required to provide anything!
  - Typically, an exception is raised!

Benefits and Obligations

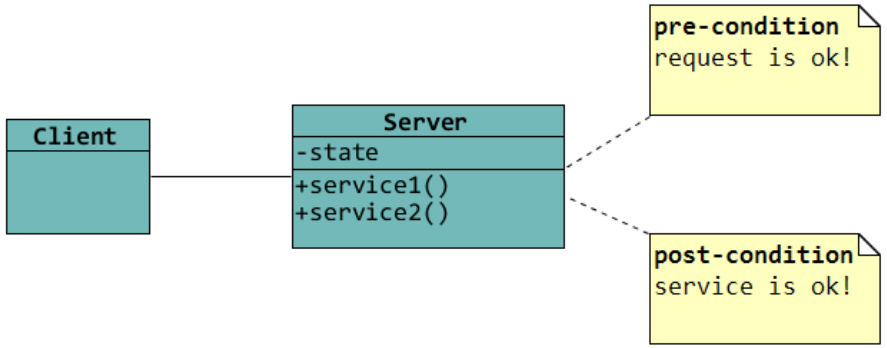
- A contract provides benefits and obligations for both clients and servers!

	Obligations	Benefits
Client	Only call a method if its pre-condition is met!	Can rely on the promised post-condition.
Server	Ensure that the promised function is implemented.	No need to validate the pre-condition!

Violating the Contract

Contracts and Violations

- A contract obligates the client to pose valid requests and the server to correctly provide the service.



- If either the client or the server violates the contract, an exception is raised.
  - The server does not need to implement any "defensive code" to handle these errors!

Exceptions

- Exception (ISO/IEC/IEEE 24765:2010)
  - An event that causes suspension of normal program execution.
- Reasons for exceptions
  - Programming errors
    - e.g., NullPointerException
    - The client code usually cannot do anything about programming errors.
  - Resource failures
    - e.g., program runs out of memory
    - Client can retry the operation after some time or just log the resource failure and bring the application to a halt.
  - Client code errors
    - Client attempts something not allowed by the API (violates the contract)
    - Client can take some alternative course of action, if there is useful information provided in the exception.

Reaction to Exceptions

- There are two reasonable ways to react to an exception:
  - Clean up the environment and report failure to the client ("organized panic")
  - Attempt to change the conditions that led to failure and retry

Reaction to Exceptions

- A failed pre-condition often indicates that the server is not prepared to perform the service.
- Deliver information, so that the client can change the conditions!

- A failed post-condition always indicates presence of a software defect, so “organized panic” is usually the best policy!

Pre-Conditions with Bean Validation

```
@Service
@Validated
public class ImageService {

    public Image findById(@Min(1) int nr) {
        return repository.findById((long) nr).orElse(null);
    }
…
}
```

Pre-Conditions with Bean Validation

```
javax.validation.ConstraintViolationException:
        findById.nr: must be greater than or equal to 1
```

```
@Test
public void testContractForFindById(){
 assertThatThrownBy(() -> { imageService.findById(0);})
        .isInstanceOf(ConstraintViolationException.class);
}
```

Vielen Dank!

Andreas Steffens

Interactive Pioneers Solutions GmbH
Belvedereallee 5
D-52070 Aachen

ast@interactive-pioneers.de
+49 0241 – 90880 730