Secure OOP with Java

Lecture - Unit 13

Claudia Maderthaner <claudia.maderthaner@fh-hagenberg.at>

Secure Software Development

Security is a concern, not a feature

Secure by Design
— Johnsson, Deogun, Swano

Security

- Protecting your assets from
  - attackers
  - natural disasters
  - vandalism
  - loss
  - misuse

Information Security

Protecting information and information systems from unauthorized access, use, disclosure, disruption, modification, or destruction.

— Federal Information Security Modernization Act of 2002

Software Security

Software security is at once a logical practice and an art, one based on intuitive decision making. It requires an understanding of modern digital systems, but also a sensitivity to the humans interacting with, and affected by, those systems.

Designing Secure Software - A Guide for Developers
— Loren Kohnfelder

Foundations

- Trust

Violations of Trust

- Malice
- Incompetence

Principles

- "CIA"
  - Confidentiality
  - Integrity
  - Availability
- "**Gold Standard**"
  - **Au**thentication
  - **Au**thorization
  - **Au**diting

Confidentiality

- Disclosing private information in only an authorized manner
- Keeping things secret that should not be made known to the public
- Attacked by
  - Interception

Integrity

- Authenticity of data
- Accuracy of data
- Attacked by
  - Interruption
  - Modification
  - Fabrication

⇒ Protecting against unauthorized tampering or removal

Availability

- Data is at hand in a timely manner
- Attacked by
  - Interruption
  - Modification
  - Fabrication

Authentication

- Determination of the identity of a principal
- Claim is tested by credentials
  - something you know
  - something you have
  - something you are
  - somewhere you are

Principal

- Person
- Business
- Organization
- Government entity
- Application
- System
- Device
- ...

Authorization

- Allowing an action by an authenticated principal
  - Role-based access control (RBAC)
  - Attribute-based access control (ABAC)
  - Policy-based access control (PBAC)

Auditing

- Reliable records of actions by principals
  - authentication and authorization events
  - system startup and shutdown
  - software updates
  - administrative access
  - ...

⇒ Non-repudiability, traceability

Privacy

Data protection	restricted access
Right to know	data subject access requests
Right to be forgotten	retention/deletion
Judicial investigations	consent decrees

Data privacy may be defined as the authorized, fair, and legitimate processing of personal information. — The Privacy Engineer's Manifesto

Secure Design

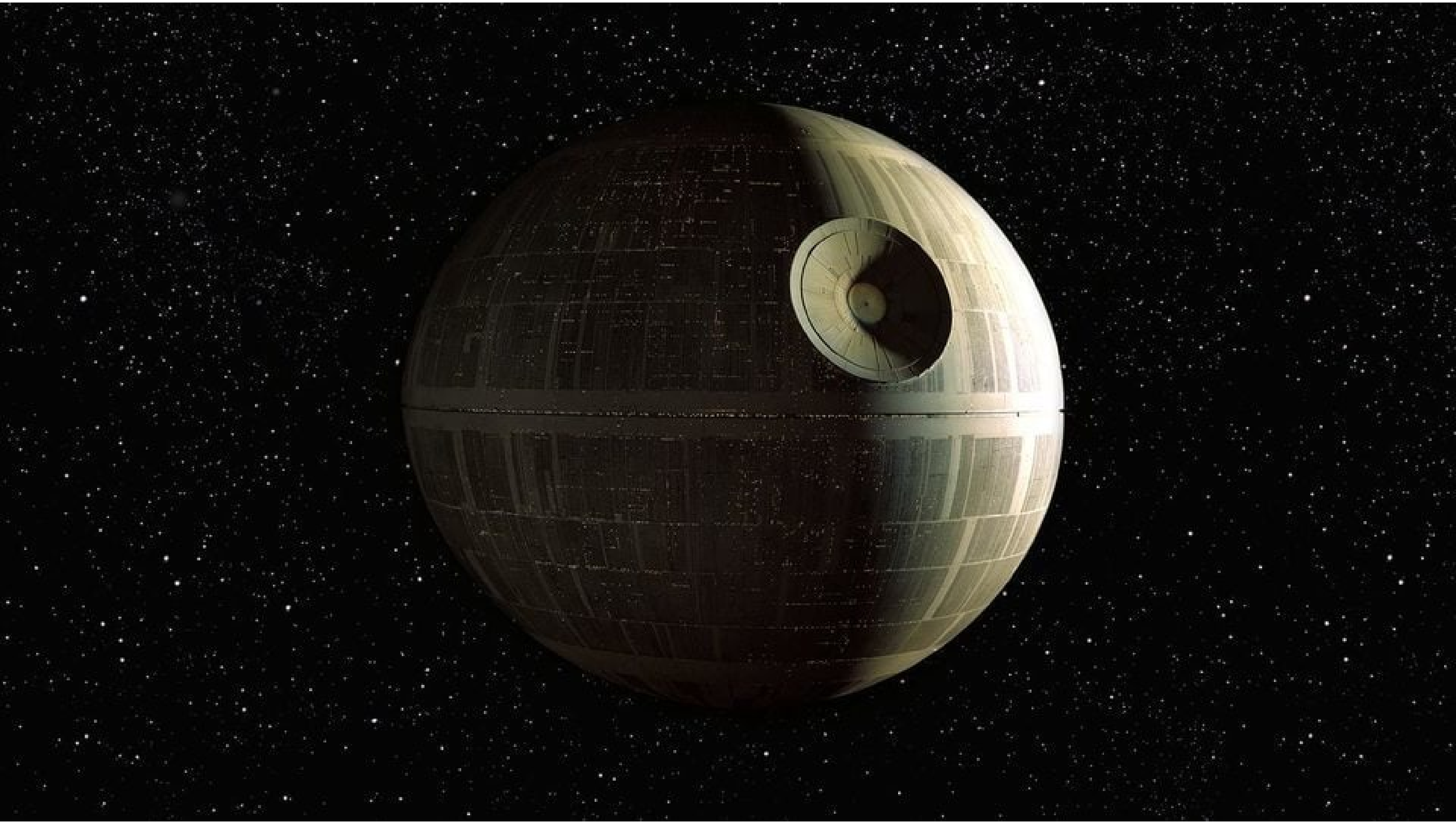
Design Attributes

- Economy of Design

  → Designs should be as simple as possible

- Transparent Design

  → Strong protection should never rely on secrecy



Exposure Minimization

- Least Privilege
- Least Information
- Secure by Default
- Allow-lists over Block-lists
- Avoid predictability
- Fail securely

Strong Enforcement

- Complete mediation - checking all accesses consistently
- Least common mechanism

Redundancy

- Defense in Depth
- Separation of Privilege

Trust and Responsibility

- Reluctance to trust
- Accept security responsibility

Anti-Patterns

- Confused deputy - higher-privilege code is invoked to do things on behalf of the lower-privilege caller
- Backflow of trust
- Third-party hooks
- Unpatchable components

Domain Driven Design (DDD)

- Understanding the complexity of the domain
- Solving business logic problems
- Models are simplifications/abstractions
- Models are strict
- Models capture (common) understanding

Domain Primitives

- Smallest building blocks
- Use instead of language primitives

```
public class Address {

    private final String postCode;
    private final String city;

    public Address(String postCode, String city) {
        this.postCode = postCode;
        this.city = city;
    }

    public String getPostCode() {
        return postCode;
    }

    public String getCity() {
        return city;
    }
}
```

```
Address address = new Address("Franz", "Ferdinand");
```

```java
public class Address {

    private final int postCode;
    private final String city;

    public Address(int postCode, String city) {
        this.postCode = postCode;
        this.city = city;
    }

    public int getPostCode() {
        return postCode;
    }

    public String getCity() {
        return city;
    }
}
```

```java
Address address = new Address(-200, "Linz");
```

```java
import java.util.regex.Pattern;

public class PostCode {
    private static final Pattern POST_CODE_PATTERN = Pattern.compile("\\d{4}");

    private final String value;

    public PostCode(final String value) {
        check(value);
        this.value = value;
    }

    private void check(final String value) {
        if (!POST_CODE_PATTERN.matcher(value).matches()) {
            throw new IllegalArgumentException("Invalid post code format: " + value);
        }
    }

    public String getValue() {
        return value;
    }
}
```

```java
public class Address {

    private final PostCode postCode;
    private final String city;

    public Address(PostCode postCode, String city) {
        this.postCode = postCode;
        this.city = city;
    }

    public PostCode getPostCode() {
        return postCode;
    }

    public String getCity() {
        return city;
    }
}
```

Promoting Security

- Immutability
- Failing fast
  - Check preconditions for method arguments
  - Check invariants in constructors
- Failing for bad state

Software Design Reviews

SDR Process

1. Study the design
2. Inquire about the design
3. Identify the most security-critical parts
4. Collaborate with the designer(s)
5. Write a summary report
6. Follow up on subsequent design changes

Four Questions

1. What are we working on?
2. What can go wrong?
3. What are we going to do about it?
4. Did we do a good job?

Secure Implementations

Challenges

- Malicious influence of attackers
- Vulnerabilities are bugs
- Vulnerability chains
- Bugs and entropy
- Vigilance

Code Quality

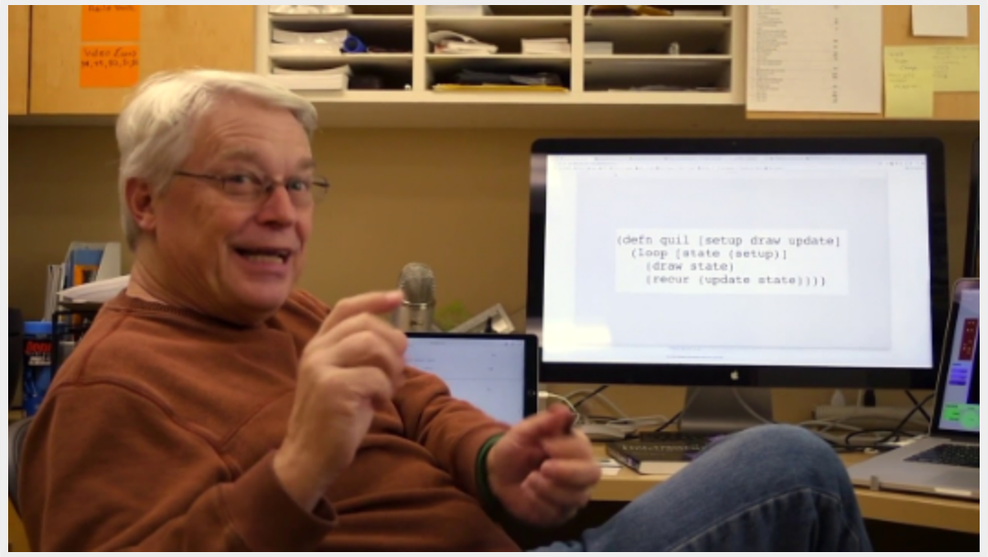
- Code hygiene/clean code
- Exception Handling
- (Security) Documentation
- Security Code Reviews

Clean Code

Any fool can write code that a computer can understand. Good programmers write code that humans can understand.

— Martin Fowler

Uncle Bob


(defn quil [setup draw update]
(loop [state (setup)]
(draw state)
(recur (update state))))

Clean Code

Clean code is code that is

- easy to understand and
- easy to change

⇒ Clean code can be read and enhanced by a developer other than its original author.

⇒ With understandability comes readability, changeability, extensibility and maintainability.

Secure Development Environment

- Strictly separate development from production
- Secure development tools
- Formal release process

Coding Vulnerabilities

- Atomicity
- Timing attacks
- Serialization

The Usual Suspects

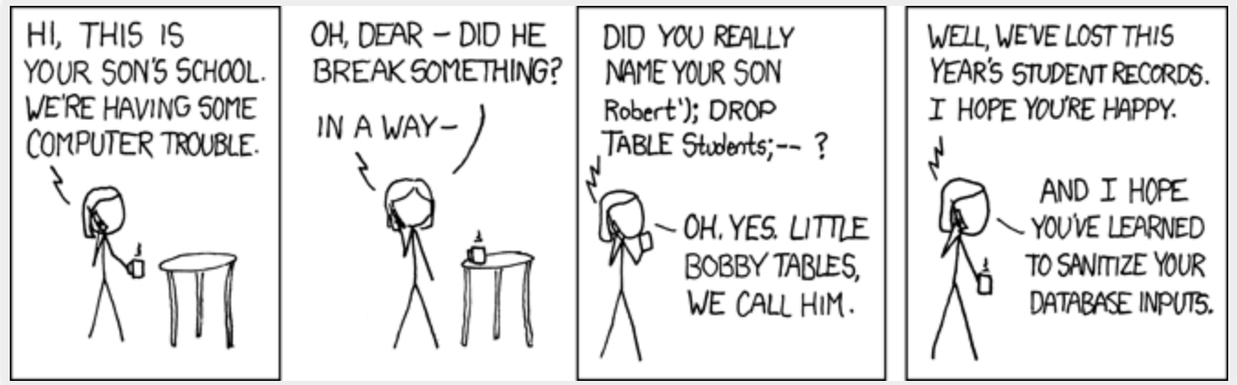
- Fixed-width integer vulnerabilities
- Floating-point precision vulnerabilities
- Buffer overflow and other memory management issues
- Input validation
- Character string mishandling
- Injection attacks
- Web security

Untrusted Input

Inputs that are out of your control and might be manipulated

Injection Vulnerabilities

- SQL statements
- Path traversals
- Regular expressions
- XML (XXE declarations)
- Shell commands
- Interpreting strings as code
- HTML and HTTP headers



https://xkcd.com/327

Prepared Statement

```
CREATE TABLE staff(id NUMBER(10), name VARCHAR(50));
```

```
String sql="insert into staff values(?,?)";
```

```
try (Connection connection =
    DriverManager.getConnection("jdbc:postgresql://localhost:5432/example",
        "postgres", "postgres")) {
    PreparedStatement stmt=con.prepareStatement("insert into Emp values(?,?)");
    stmt.setInt(1,101);
    stmt.setString(2,"John Doe");

    int i=stmt.executeUpdate();
    System.out.println(i+" records inserted");
}
```

Input Validation

aka input sanitization

- Defensive coding imposing restrictions on inputs
- Forcing conformity to prescribed rules

1. Determining what is valid
2. Define validation criteria
3. Reject invalid input
4. Correcting invalid input

```java
import java.util.regex.Pattern;

public class PostCode {
    private static final Pattern POST_CODE_PATTERN = Pattern.compile("\\d{4}");

    private final String value;

    public PostCode(final String value) {
        check(value);
        this.value = value;
    }

    private void check(final String value) {
        if (!POST_CODE_PATTERN.matcher(value).matches()) {
            throw new IllegalArgumentException("Invalid post code format: " + value);
        }
    }

    public String getValue() {
        return value;
    }
}
```

Input Validation Checklist

Origin	Is the data from a legitimate sender?
Length/size	Is it reasonably big?
Lexical content	Does it contain the right characters and encoding?
Syntax	Is the format right?
Semantics	Does the data make sense?

Character String Vulnerabilities

- Length issues
- Unicode issues
- Encodings and glyphs

```
wikipedia.org
```

```
xn--wikipedi-86g.org
```

```
wikipedia.org
```

Secure Failure Handling

- Technical exceptions
- Business failures
  - Handling without exceptions
  - Handling with domain specific exceptions

    WARNING

    Do not leak sensitive information through exception payloads!

Dependencies

- External components
- 3rd-party libraries
- Application runtimes
- Host systems

Web Security

OWASP Top 10 2021

1. Broken Access Control
2. Cryptographic Failures
3. Injection
4. Insecure Design
5. Security Misconfiguration
6. Vulnerable and Outdated Components
7. Identification and Authentication Failures
8. Software and Data Integrity Failures
9. Security Logging and Monitoring Failures
10. Server-Side Request Forgery

Security Testing

Types of Security Testing

- Input validation tests
- Fuzz testing
- Security regression tests
- Availability testing

Automated Security Testing

- Automate your security tests as far as you can.
- Integrate your security tests - like any other tests - in your build pipeline.

Limits of Security Tests

- More important for code than for security
- Often check for actions or failing rather than success
- Ensure that key steps are working correctly

Contact

Moodle Discussion Board

claudia.maderthaner@fh-hagenberg.at