IT-Security

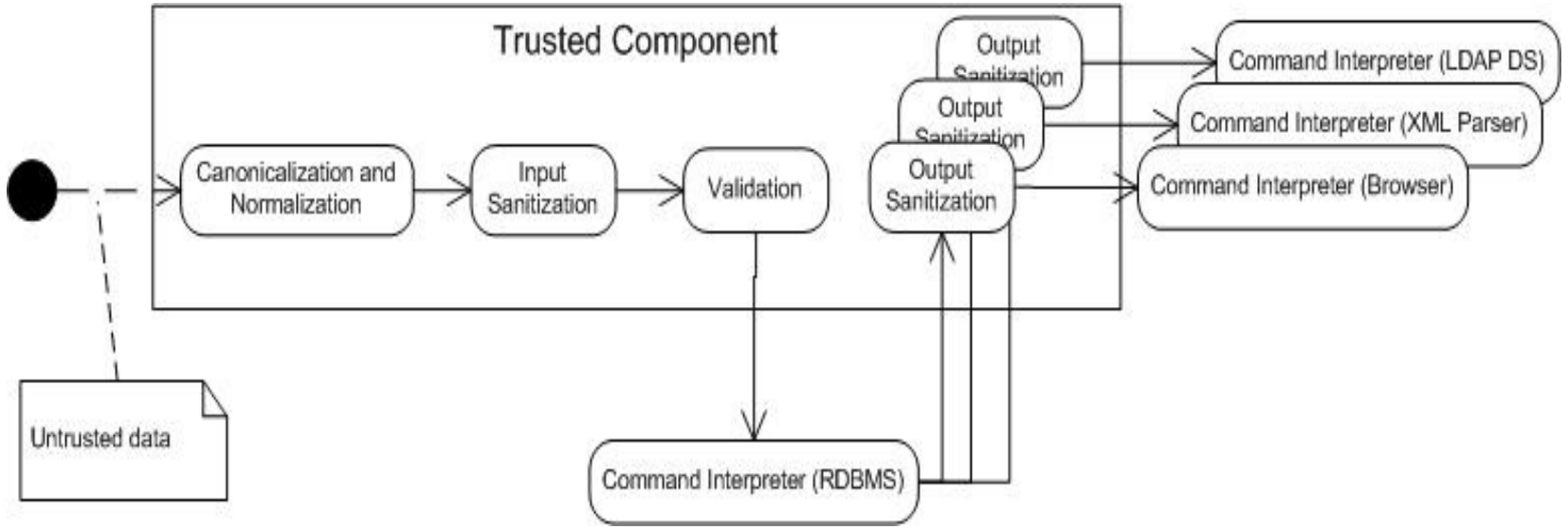
Chapter 5: Application Security Part 2

- Solutions for application security
- Safe Components, Validation
- Error handling, security testing
- Logging, Safe Operation
- Cloud Security
- IoT Security

Prof. Dr. Reiner Hüttl, FH Rosenheim, © 2023, 15.03.2023

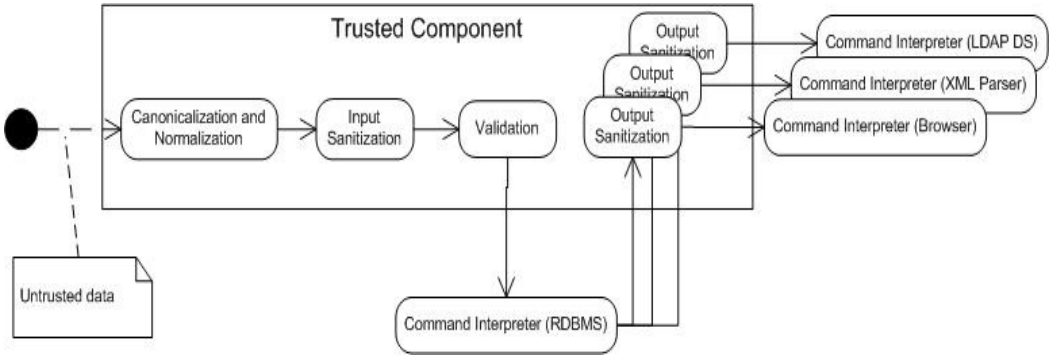
Secure programming with components

- Secure programming helps against the vulnerabilities of OWASP and all the other vulnerabilities of applications
- A smart decomposition into components with security in mind facilitates secure programming
- Secure components have trust boundaries.



Trusted Components

- Canonicalization & Normalization
  - reduction to the simplest, standardized form of representation



- Sanitization
  - ensure that transferred data meets the requirements of the (third party) component
- Validation
  - verify that input matches the expected pattern
  - check for expected types
  - validate requirements for numeric inputs

Canonicalization

- Problem:
  - there are often multiple options when assigning a name
  - attackers can exploit code that makes decisions based on filenames or URLs
- Examples of problems with URLs:
  - ROSE == R%6fse ???
  - http://www%2emicrosoft%2ecom%2ftechnet%2fsecurity
  - http://172.43.122.12 == http://2888530444
    Dotless-IP: a.b.c.d = (a * 256^3) + (b* 256^2) + (c * 256^1) + (d * 256^0)
- Examples of potential file name problems:
  - old file name formats (Fiscal04Budget.xls == FISCAL~1.xls)
  - dot at the end of filename allowed, but will be removed automatically
  - absolute and relative file names
  - upper and lower case
- Measure
  - canonicalize filenames (long filenames only, remove dot at end, absolute path) and URLs

Validation of user input

- Motivation: "Wrong" inputs can cause programs to behave in malicious manner
- Identify all sources of input to a web application
  - all URL parameters
  - data passed with POST from text inputs, checkboxes, radio buttons, drop-down lists, buttons, hidden fields
  - WARNING: even predefined values can be manipulated
  - data from http headers and cookies
  - Input from other sources (DB tables, files)
- Goal of validation
  - ensure that data has the expected format

Validation (1)

- Create domain types (business data types)
  (e.g. email, account, date, customer ID)
- Make sure that all inputs are identified and validated
  - validate before any other task
  - perform authorization together with validation
- Write functions that perform validation
- Check length, range (e.g. numeric and non-negative) , format and range
- Use whitelisting instead of blacklisting for filtering
  - **Whitelisting** only allows data that is believed to be harmless
  - **Blacklisting** allows everything that is not explicitly forbidden
- Never use client-side validation as the only basis for decision-making

Validation (2)

- Use data indirection
  - keep important data at the server side as much as possible
  - incoming data is not the target data but only suitable for searching the target data on the server
  - examples: reference account number, price of goods via customer no. or article no.
- Check also server generated input to subsystems
  - identify characters that are metacharacters in a subsystem (see SQL-injection, shell-command-injection)
  - handle metacharacters before passing data to subsystems
  - protect input to subsystems with cryptographic hash functions (MAC) or encryption

Measures for meta character problems

- Meta characters
  - Are characters that do not stand for themselves within a certain context but have a special meaning.
  - when passing data to a subsystem they change from a text character to a control character
- Measures
  - insert escape character like /, if meta character also makes sense as normal character
  - otherwise: remove metacharacters
  - use subsystems to interpret metacharacters (e.g. prepared statements, DOM)
  - encapsulate the communication with other systems
  - keep permissions in subsystems minimal
  - validate input
  - **Multi layer defense**: if one security mechanism fails, another should handle the problem

    Example: restrictive administration of a DB, in case that SQL input validation fails

Use regular expressions for validation

- A regular expression is a string for describing a set of strings
- This is useful in IT security for the validation of input and output
- Example: regular expression for file names:
  - ^[cd] (\\ \w+) + \\ \w {1,32} \. (txt | jpg | gif)$
  - matching file: d:\mydir\a\myfile.jpg
- These are many implementations available in different programming languages
  - Java: classes Pattern, Matcher
  - PHP: Functions ereg, eregi
  - .Net: classes Regex, Match
- Regular expressions are useful but also error prone
  → if available it is better to use proven APIs

HTML encoding

- Certain HTML metacharacters are mapped to equivalent characters
  - any & to &amp
  - any „ to &quot
  - any < to &lt
  - any > to &gt
  - single quotes to &#39
- There are implementations in various web programming languages
  - htmlspecialcharacter in PHP
  - HttpServerUtility.HTMLEncode in ASP.Net
- HTML encoding causes the browser to display data as it was written, rather than interpreting it as HTML tag

Error Handling

- Clean and stable error handling is important for secure software
  - secure software can slow down,
    but it should not crash or give incorrect results.
  - avoid that errors a user gets more rights by an error
  - after an error, an application must be brought to a stable and clean state.
- Do not correct an invalid input to make it a valid one
- Logging of errors
  - the system logs are usually not sufficient (web server, database, firewall, …)
  - additional logging on application level is necesary to find out where the cause of the error lies

Rules for Exception Handling

- If you find a security related error
  - correct the bug and look for similar problems in other code parts
  - correct it as close to the vulnerability as possible
  - treat the cause not the symptoms
  - extend the test drivers to include the bug
- Do not send detailed error messages to the client
  - no database errors, access violations, file names, table names
  - because otherwise the attacker gets information about internal structures of the application
  - catch all errors and show only a general error message page
  - write detailed error description in the log files
  - generate log files on application level

Information concealment is the first line of defense

Anti Pattern: error output to the user

Information disclosure and leakage

HTTP Status 500 - An exception occurred processing JSP page /basket.jsp at line 244

type Exception report

message An exception occurred processing JSP page /basket.jsp at line 244

description The server encountered an internal error that prevented it from fulfilling this request.

exception

```
org.apache.jasper.JasperException: An exception occurred processing JSP page /basket.jsp at line 244

241:                                    stmt.execute();
242:                                    stmt.close();
243:                            } else {
244:                                    stmt = conn.prepareStatement("UPDATE BasketContents SET quantity = " + Integer.parseInt(value) + " WHERE basketid=" + basketId +
245:                                                    " AND productid = " + prodId);
246:                                    stmt.execute();
247:                                    if (Integer.parseInt(value) < 0) {


Stacktrace:
        org.apache.jasper.servlet.JspServletWrapper.handleJspException(JspServletWrapper.java:568)
        org.apache.jasper.servlet.JspServletWrapper.service(JspServletWrapper.java:455)
        org.apache.jasper.servlet.JspServlet.serviceJspFile(JspServlet.java:395)
        org.apache.jasper.servlet.JspServlet.service(JspServlet.java:339)
        javax.servlet.http.HttpServlet.service(HttpServlet.java:727)
        org.apache.tomcat.websocket.server.WsFilter.doFilter(WsFilter.java:52)
```

root cause

```
javax.servlet.ServletException: java.sql.SQLException: Unexpected token: % in statement [UPDATE BasketContents SET quantity = 4 WHERE basketid=%27 AND productid = 20]
        org.apache.jasper.runtime.PageContextImpl.doHandlePageException(PageContextImpl.java:916)
        org.apache.jasper.runtime.PageContextImpl.handlePageException(PageContextImpl.java:845)
        org.apache.jsp.basket_jsp._jspService(basket_jsp.java:390)
        org.apache.jasper.runtime.HttpJspBase.service(HttpJspBase.java:70)
        javax.servlet.http.HttpServlet.service(HttpServlet.java:727)
        org.apache.jasper.servlet.JspServletWrapper.service(JspServletWrapper.java:432)
```

note The full stack trace of the root cause is available in the Apache Tomcat/7.0.2 logs.

Apache Tomcat/7.0.2


```
        org.apache.tomcat.websocket.server.WsFilter.doFilter(WsFilter.java:52)
```

note The full stack trace of the root cause is available in the Apache Tomcat/7.0.2 logs.

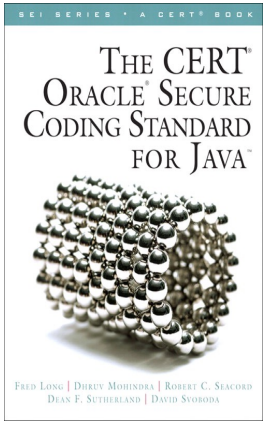
Apache Tomcat/7.0.2

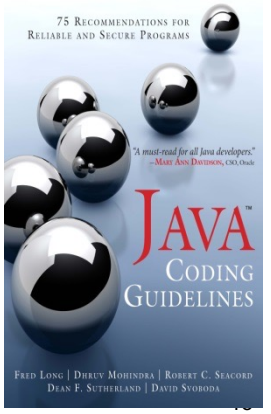
Is this a problem?

- Step 1: start Google search for "apache tomcat 7.0.2 vulnerabilities".
- Step 2: select second hit
- Step 3: and you find 42 vulnerabilities in total!
- Among them, for example:
  - CVE-2013-4444 „Unrestricted file upload vulnerability in Apache Tomcat 7.x before 7.0.40, in certain situations involving outdated java.io.File code and a custom JMX configuration, allows remote attackers to execute arbitrary code by uploading and accessing a JSP file.“
  - CVE-2011-3190 „Certain AJP protocol connector implementations in Apache Tomcat 7.0.0 through 7.0.20, 6.0.0 through 6.0.33, 5.5.0 through 5.5.33, and possibly other versions allow remote attackers to spoof AJP requests, bypass authentication, and obtain sensitive information by causing the connector to interpret a request body as a new request.“
  - CVE-2011-1419 “Apache Tomcat 7.x before 7.0.11, when web.xml has no security constraints, does not follow ServletSecurity annotations, which allows remote attackers to bypass intended access restrictions via HTTP requests to a web application. NOTE: this vulnerability exists because of an incomplete fix for CVE-2011-1088.“

Secure Programming

- Almost all security vulnerabilities are due to bad, careless and insecure programming!
- That's why SW developers and architects need to read literature on secure programming
- Examples for Java
  - **Secure Coding Guidelines for Java SE**
    Updated for Java SE, Version: 9.0, Last updated: Januar 2022
    https://www.oracle.com/java/technologies/javase/seccodeguide.html
  - ***The CERT™ Oracle™ Secure Coding Standard for Java***
    Fred Long, Dhruv Mohindra, Robert C. Seacord, Dean F. Sutherland, David Svoboda
    Rules available online at www.securecoding.cert.org
  - ***Java Coding Guidelines***
    Fred Long, Dhruv Mohindra, Robert C. Seacord, Dean F. Sutherland, David Svoboda





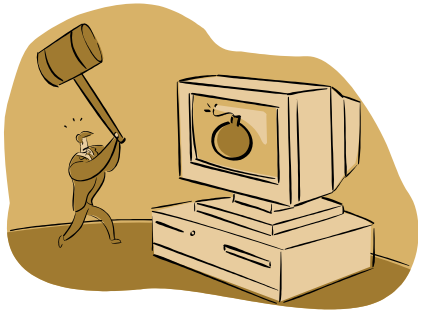
Software Development Rules

- Best practices for good quality code
  - avoid duplicate code
  - isolate functionality
  - limit the length of code blocks
  - limit the functionality of functions
  - limit the need for comments
  - limit the scope of variables
  - perform unit tests
- Enemies of secure code
  - ignorance
  - disorder
  - deadlines, time pressure

Measure for secure software: Test all software!!!

- Permanent manual and automatic tests are necessary to detect and eliminate errors in the software in time

- The types of testing procedures is diverse, here are a few examples:
  - unit test, component test, integration test, system test
  - acceptance test, Test Driven Development
  - usability test
  - regression test
  - load test, robustness test



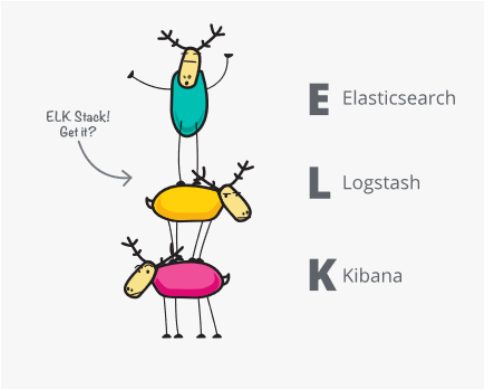
Risks always remains, therefore logging is necessary
OWASP Nr. 9: Security Logging and Monitoring Failures

- A residual risk always remains: For this reason, attacks must not only be defended against, but also detected.
- The most important source of information for detecting attacks and intrusions are logs.

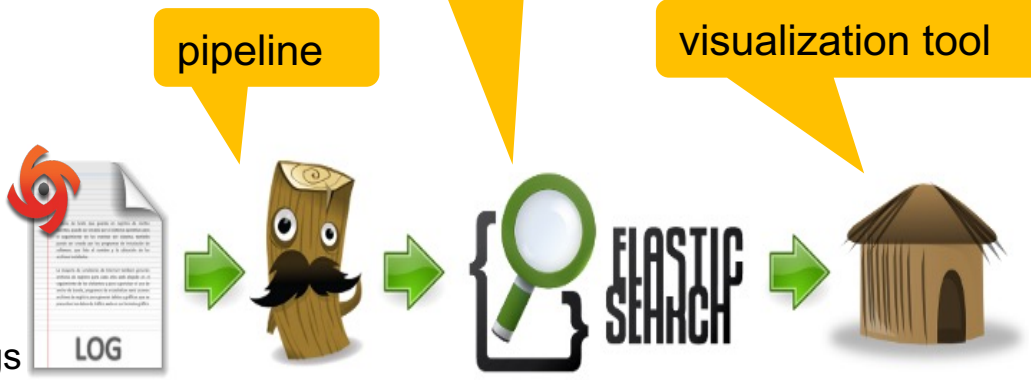
Good logging is the basis for forensic

- Central Questions:
  - What will be logged? (https://www.owasp.org/index.php/Logging_Cheat_Sheet)
  - What will be not logged?
  - How are the logs collected? (e.g. logstash, ...)
  - How are the logs analyzed and visualized? (e.g. Kibana, ...)
- There are ready-made tool chains for log analysis (ELK Stack)
  - OSSEC (https://www.ossec.net), elastic (https://www.elastic.co)

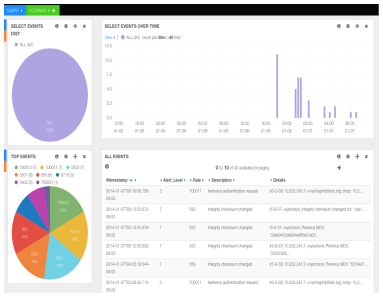
The ELK stack



- server-logs
- application-logs
- interface-Logs
- operating system-logs



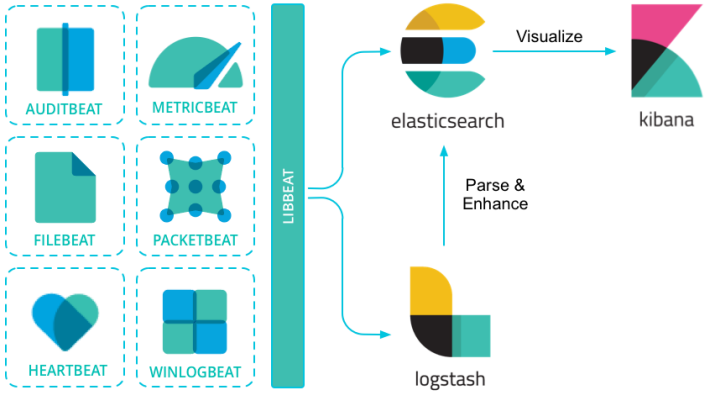
Logstash **Kibana**



- abnormal trends
- attack pattern
- alerts

inform monitoring

data shipper



Elastic stack

Quelle: https://www.elastic.co/de/what-is/elk-stack

Rules for auditing and logging and monitoring

- **Auditing**
  - Recording of important **user activities** (**traceability**)
- **Logging**
  - Log all **security relevant events**
- **Monitoring**
  - Collect log data to **observe the performance** of an application, detect abnormal behaviour and **trigger alerts**
- Rules
  - Identify malicious behavior
  - Know the normal behavior of the application (good traffic)
  - Audit and log activities on all application layers
  - Restrict access to log files and protect log integrity
  - Do not include private or sensitive data (e.g. passwords) in log files
  - Secure and analyze log files regularly
  - Integrate Logging with Monitoring

Secure administration and operation

- In addition to secure programming, secure operation is also required to reduce the attack vectors
  - establish a secure infrastructure (FW, WAF, SSL gateway, access gateway, security zones, ...)
  - separate the production systems from test and development systems (especially the credentials)
  - secure the remote access for administration especially strong
- Establish an **Incidence Response and Emergency Management**
- **Change Management**
  - Many security vulnerabilities are introduced into the systems by bug fixing and adding new features
  - Track and test all changes

Secure administration and operation

- **System Hardening**: reduce the attack surface of all deployed systems by restrictive configuration
  - change insecure default settings
  - adjust access rights
  - disable unnecessary services
  - perform vulnerability scans
- **Decomission**
  - remove authentication and authorization settings (accounts, key, certificates, ...)
  - reset configuration settings (FW settings, open ports, DNS entries, ...)
  - delete or save the data

Security in Cloud Plattforms

- In cloud platforms there is the possibility of **Continuous Integration CI** and **Continuous Deployment CD**
- Agility increases, but so do security risks
- The areas of development and operation are growing together more and more in modern cloud platforms based on micro services
- Developers must also deal with administration and operation
  → there is a new role in software engineering **DevOps**
- Automate as much as possible in cloud platforms
  - this ensures traceability of what is happening in the platform
  - it reduces errors and human interaction

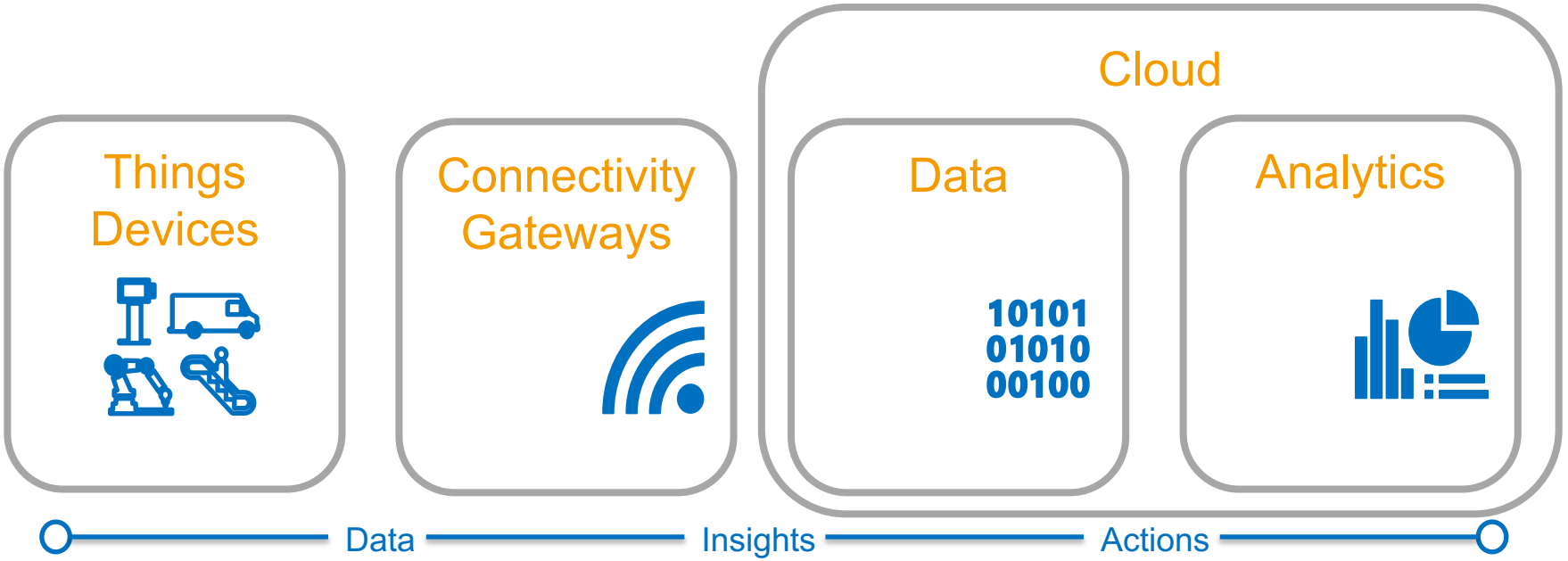
DevOps/Cloud Security Issues

- Securing the CI/CD pipeline
  - Goal: Deployment from trusted sources
  - Risk: Failures due to untested or insecure code.
  - Measures: Version control in GitHub, 2FA, 4-eye-principle, automatic tests, security checks, artifact repository, image registry, rollback options
- Image and container security
  - Risk: unauthorized changes on the prod environment
  - Measures:
    - use secure base images/containers
    - vulnerability checks for used libraries, external services, APIs, frameworks, configurations
- Network security
  - Protection against other applications, clients, attackers on the platform
  - Isolation
  - Ingress and Egress control

DevOps/Cloud Security Issues

- Access and Account Management
  - Control the rights and roles on different levels
  - Technical accounts (administration of the cloud platform and infrastructure, GitOps Cockpit based on cluster security policies)
  - Functional accounts (application administration, user management, data services, reporting, billing, hotline)
  - DevOps/Maintenance Accounts (Error Analysis and Recovery, Logging, Monitoring, Alerting)
- Secret Management
  - Goal: passwords, API keys, access tokens must be protected.
  - Measures: Sealed Secrets, Key Vault, Secret Manager
- Cluster Backup and Recovery
  - Prevention of data loss
  - Fast failover

IoT-Security (Internet of Things)



- IoT means: many devices, lots of communication, lots of data
- Examples for IoT applications
  - Smart Home, Smart Mobility, Smart Factory, Smart Health

IoT brings new challenges for IT security

- Securing devices that do not have physical access protection
- Updating (old) devices for long-term operation
- The generated data enables control of movement and behavior
- Autonomous systems can make independent decisions
- Massive communication and sharing of data
- Many standards without security features
- Large amount of cheap consumer products

Source: Wendzel S., IT –Sicherheit für TCP/IP- und IoT-Netzwerke, Springer-Vieweg, 2018, eBook in Bibliothek

IoT has special risks

- Threats
  - Hijacked Devices (IoT Botnets)
  - information leakage
  - disruption of services, functionality
- Vulnerabilities
  - exposed device in WWW (Shodan tool shows exposed devices https://www.shodan.io/)
  - insecure web interfaces (bad session management, default credentials)
  - insecure frameworks and protocols
  - generic security vulnerabilities (no access control, no logging, insecure storage, hardcoded passwords
  - outdated devices
- IoT requires more skills than pure software engineering: additionally, systems engineering is required

Security measures for IoT

- All rules for secure application development apply to IoT.
- Special focus must be placed on the following measures
  - encrypt all communication
  - secure key management
  - (mutual) authentication mechanism
  - no execution of unauthorized code
  - secure updates
  - code signing mechanism
  - implement devices as clients (do not host services, do not act as servers)
  - least privilege
  - intrusion detection

OWASP for IoT

I1 Weak Guessable, or Hardcoded Passwords

I2 Insecure Network Services

I3 Insecure Ecosystem Interfaces

I4 Lack of Secure Update Mechanism

I5 Use of Insecure or Outdated Components

I6 Insufficient Privacy Protection

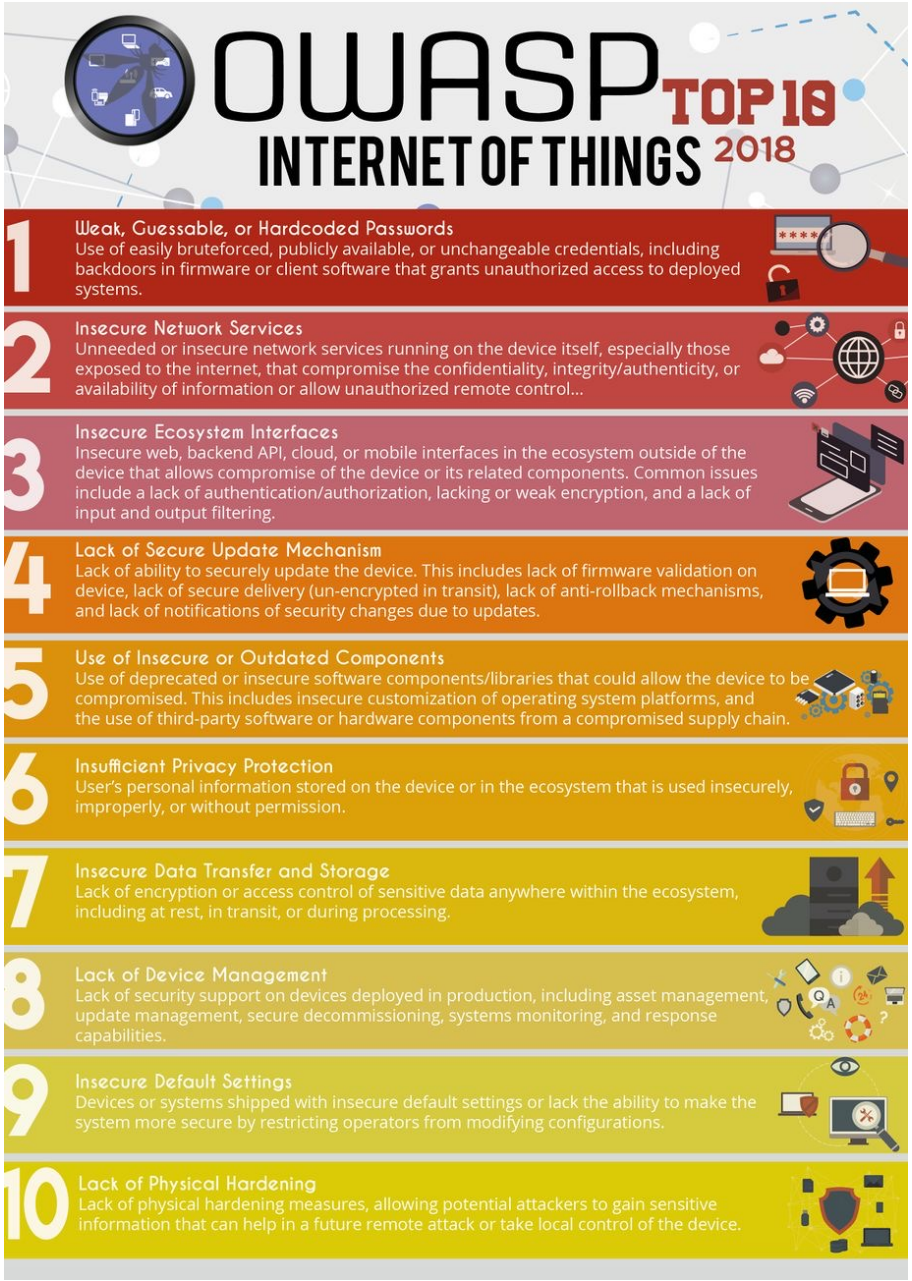
I7 Insecure Data Transfer and Storage

I8 Lack of Device Management

I9 Insecure Default Settings

I10 Lack of Physical Hardening

https://owasp.org/www-project-internet-of-things/



Summary application security

- Every software engineer must know the most important threats and protect his applications against them.
- Security must be considered from the very beginning of the SW engineering process.
- All input to an application must be validated, canonicalized, filtered, sanitized and escaped.
- In addition, measures such as secure programming, logging, secure error handling, security testing and secure operation must be implemented.
- In certain application domains (e.g., cloud, IoT), additional measures must be considered