qaware.de

QA|WARE

The Futureware Company

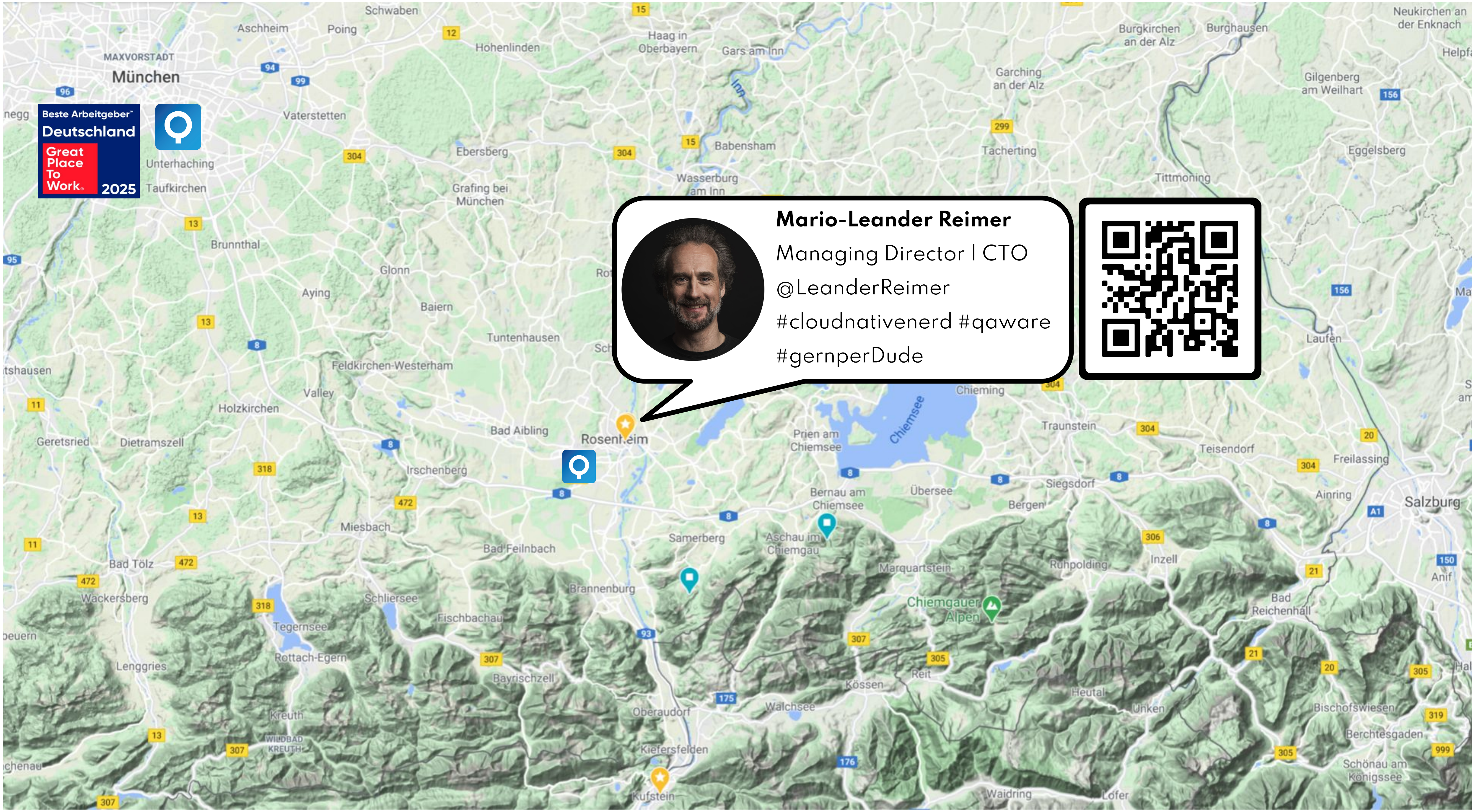
Mapping ISO 27001 to Software Engineering: **Secure by Design**

Mario-Leander Reimer

mario-leander.reimer@qaware.de

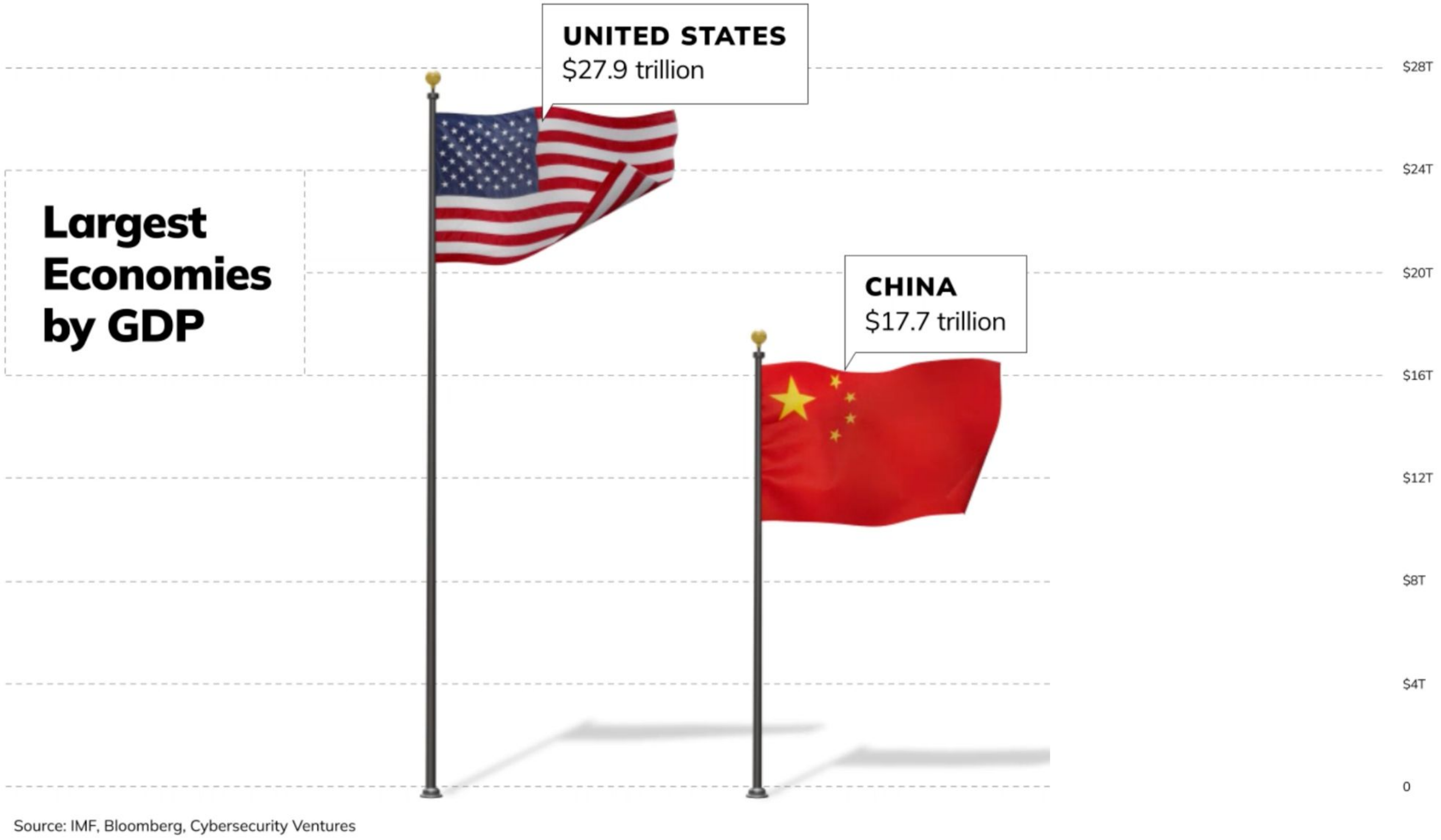
@LeanderReimer @qaware

#CloudNativeNerd #gerneperdude

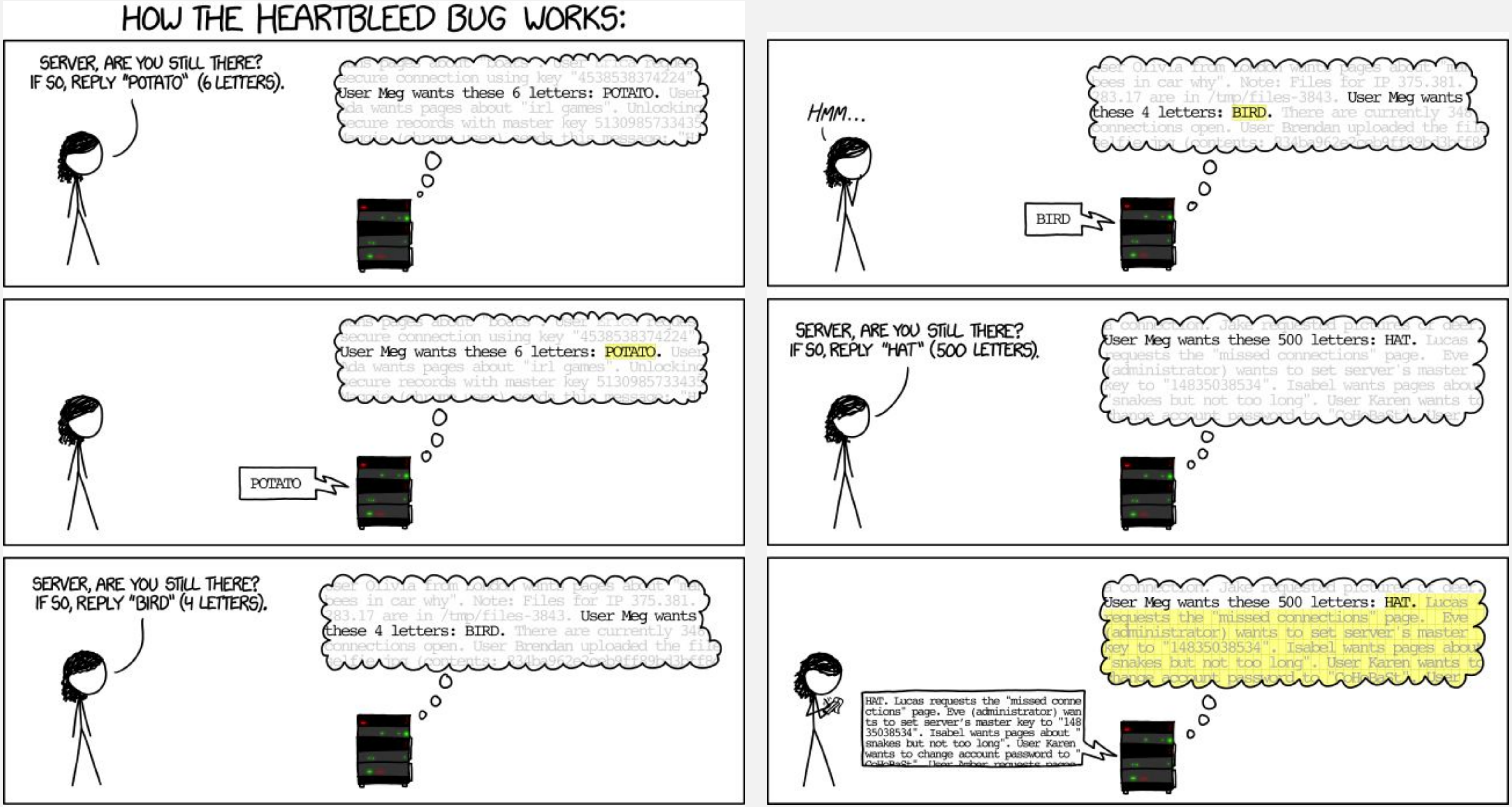

Beste Arbeitgeber
Deutschland
Great Place To Work
2025
Mario-Leander Reimer
Managing Director | CTO
@LeanderReimer
#cloudnativenerd #qaware
#gernperDude
München
Unterhaching
Taufkirchen
Rosenheim
Chiemsee
Salzburg
Kufstein
Bad Aibling
Traunstein
Wasserburg am Inn
Ebersberg
Holzkirchen
Miesbach
Bad Tölz
Tegernsee
Berchtesgaden

The World’s Third-Largest Economy Has Bad Intentions — and It’s Only Getting Bigger

QA|WARE



https://sponsored.bloomberg.com/quicksight/check-point/the-worlds-third-largest-economy-has-bad-intentions-and-its-only-getting-bigger


HOW THE HEARTBLEED BUG WORKS:
SERVER, ARE YOU STILL THERE? IF SO, REPLY "POTATO" (6 LETTERS).
secure connection using key "4538538374224".
User Meg wants these 6 letters: POTATO. User
da wants pages about "irl games". Unlocking
secure records with master key 5130985733435
secure connection using key "4538538374224".
User Meg wants these 6 letters: POTATO. User
da wants pages about "irl games". Unlocking
secure records with master key 5130985733435
POTATO
SERVER, ARE YOU STILL THERE? IF SO, REPLY "BIRD" (4 LETTERS).
bees in car why". Note: Files for IP 375.381.
283.17 are in /tmp/files-3843. User Meg wants
these 4 letters: BIRD. There are currently 346
connections open. User Brendan uploaded the file
HMM...
bees in car why". Note: Files for IP 375.381.
283.17 are in /tmp/files-3843. User Meg wants
these 4 letters: BIRD. There are currently 346
connections open. User Brendan uploaded the file
BIRD
SERVER, ARE YOU STILL THERE? IF SO, REPLY "HAT" (500 LETTERS).
User Meg wants these 500 letters: HAT. Lucas
requests the "missed connections" page. Eve
(administrator) wants to set server's master
key to "14835038534". Isabel wants pages about
"snakes but not too long". User Karen wants to
User Meg wants these 500 letters: HAT. Lucas
requests the "missed connections" page. Eve
(administrator) wants to set server's master
key to "14835038534". Isabel wants pages about
"snakes but not too long". User Karen wants to
HAT. Lucas requests the "missed connections" page. Eve (administrator) wants to set server's master key to "14835038534". Isabel wants pages about "snakes but not too long". User Karen wants to change account password to "

The Java exploit for Heartbleed only had 186 lines of code. The patch for Heartbleed only added 4 lines of code.

```
-       /* Read type and payload length first */
-       hbtype = *p++;
-       n2s(p, payload);
-       pl = p;
-
        if (s->msg_callback)
                s->msg_callback(0, s->version, TLS1_RT_HEARTBEAT,
                        &s->s3->rrec.data[0], s->s3->rrec.length,
                        s, s->msg_callback_arg);

+       /* Read type and payload length first */
+       if (1 + 2 + 16 > s->s3->rrec.length)
+               return 0; /* silently discard */
+       hbtype = *p++;
+       n2s(p, payload);
+       if (1 + 2 + payload + 16 > s->s3->rrec.length)
+               return 0; /* silently discard per RFC 6520 sec. 4 */
+       pl = p;
+
```

Bounds check for the correct record length

Apple's SSL bug: goto fail;

Always **goto fail;**

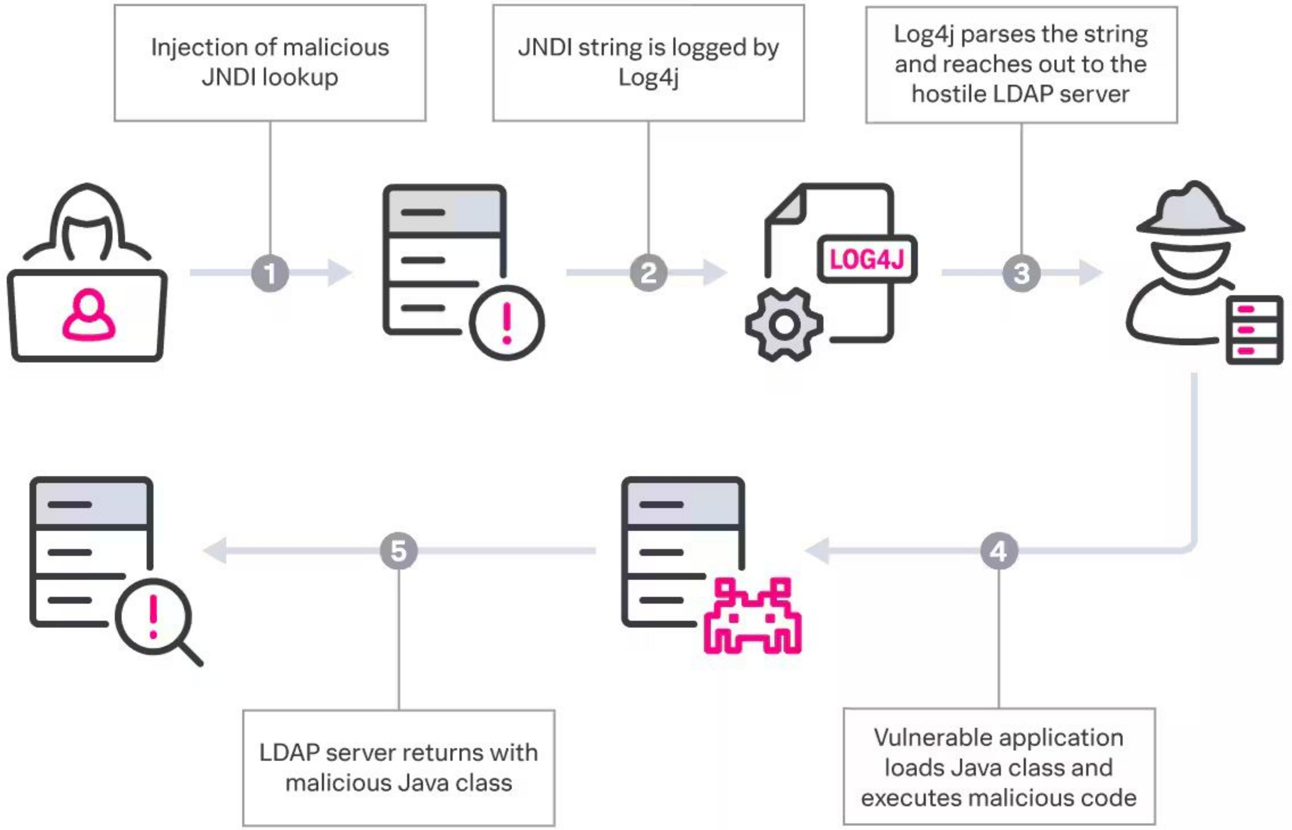
```c
    if ((err = ReadyHash(&SSLHashSHA1, &hashCtx)) != 0)
        goto fail;
    if ((err = SSLHashSHA1.update(&hashCtx, &clientRandom)) != 0)
        goto fail;
    if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
        goto fail;
    if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
        goto fail;
        goto fail;
    if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
        goto fail;

        err = sslRawVerify(ctx,
                           ctx->peerPubKey,
                           dataToSign,                /* plaintext */
                           dataToSignLen,             /* plaintext length */
                           signature,
                           signatureLen);
        if(err) {
                sslErrorLog("SSLDecodeSignedServerKeyExchange: sslRawVerify "
                    "returned %d\n", (int)err);
                goto fail;
        }

fail:
    SSLFreeBuffer(&signedHashes);
    SSLFreeBuffer(&hashCtx);
    return err;

}
```

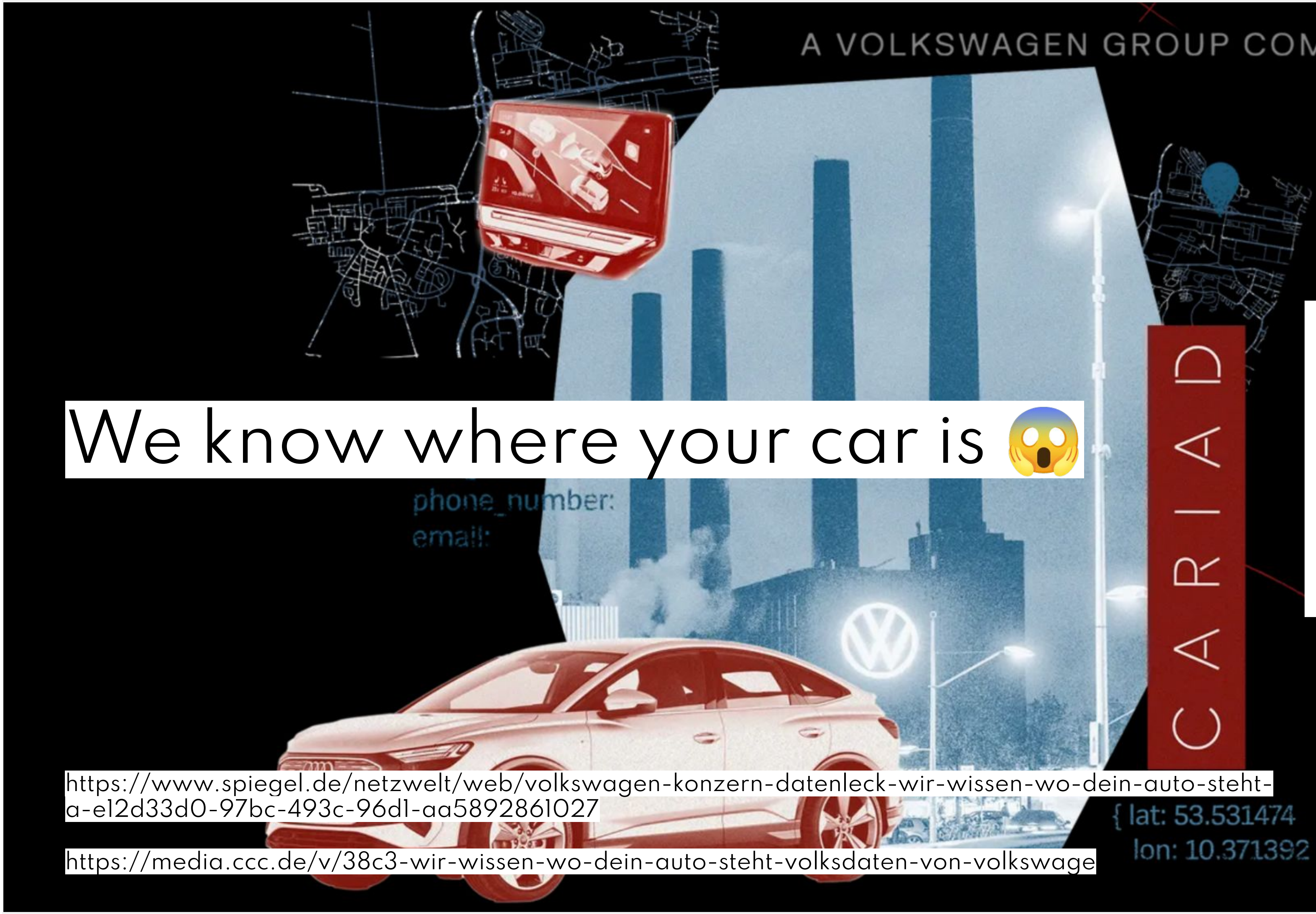
Never called!

Log4Shell

CVE-2021-44228

Date discovered: 24 Nov 2021
Date patched: 9 Dec 2021

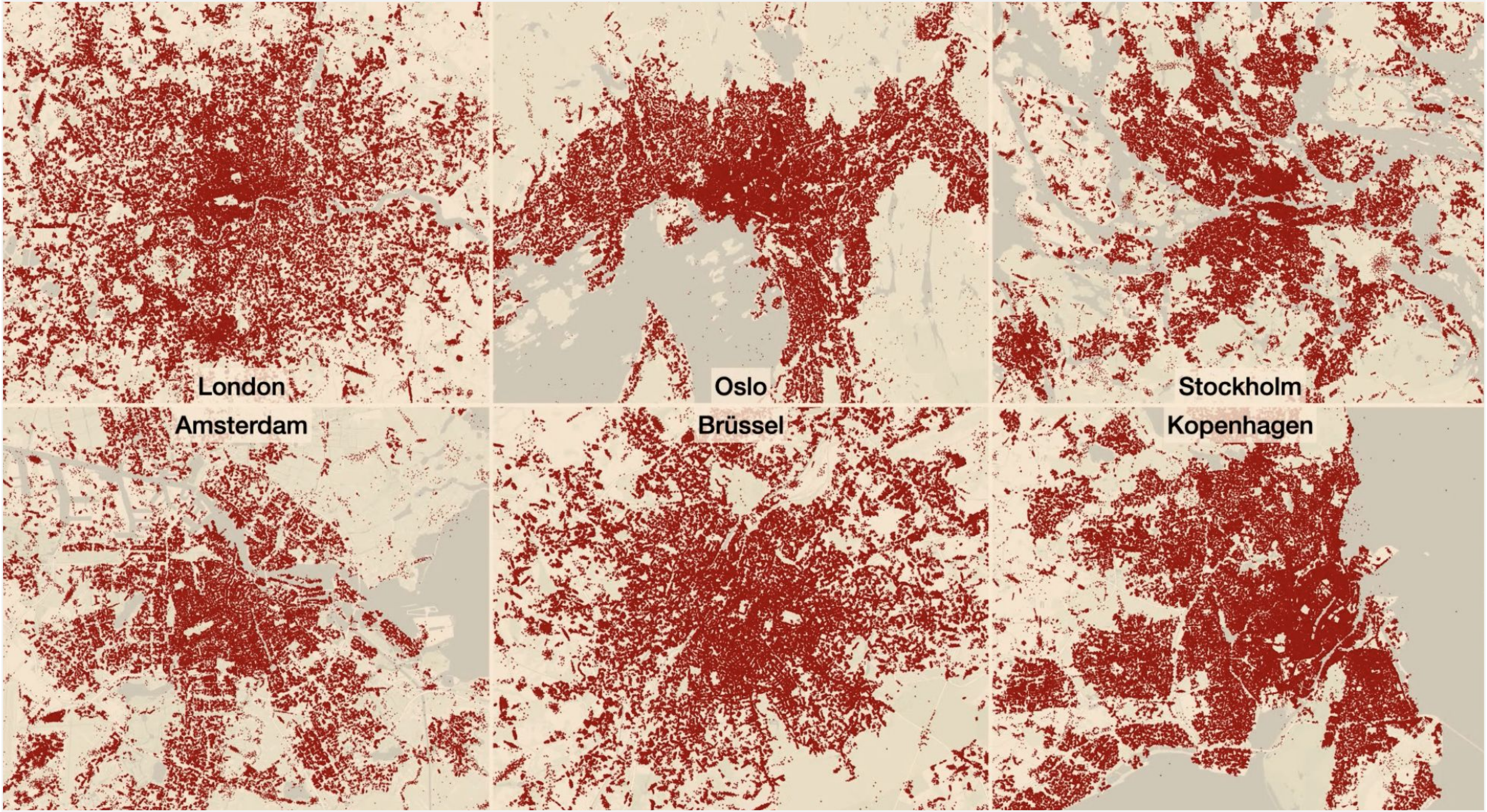




We know where your car is 😱

https://www.spiegel.de/netzwelt/web/volkswagen-konzern-datenleck-wir-wissen-wo-dein-auto-steht-a-e12d33d0-97bc-493c-96d1-aa5892861027

https://media.ccc.de/v/38c3-wir-wissen-wo-dein-auto-steht-volksdaten-von-volkswage


London
Oslo
Stockholm
Amsterdam
Brüssel
Kopenhagen

/auditevents lists security audit-related events such as user login/logout. Also, we filter by principal or type among other fields.

/beans returns all available beans in our *BeanFactory*. Unlike */auditevents*, it doesn't support filtering.

/conditions, formerly known as */autoconfig*, builds a report of conditions around autoconfiguration.

/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 summarizes the health status of our application.

/heapdump builds and returns a heap dump from the JVM used by our application.

/info returns general information. It might be custom data, build information or details about the latest commit.

/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. This might include generic metrics as well as custom ones.

/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.

QA|WARE

GET /actuator/heapdump

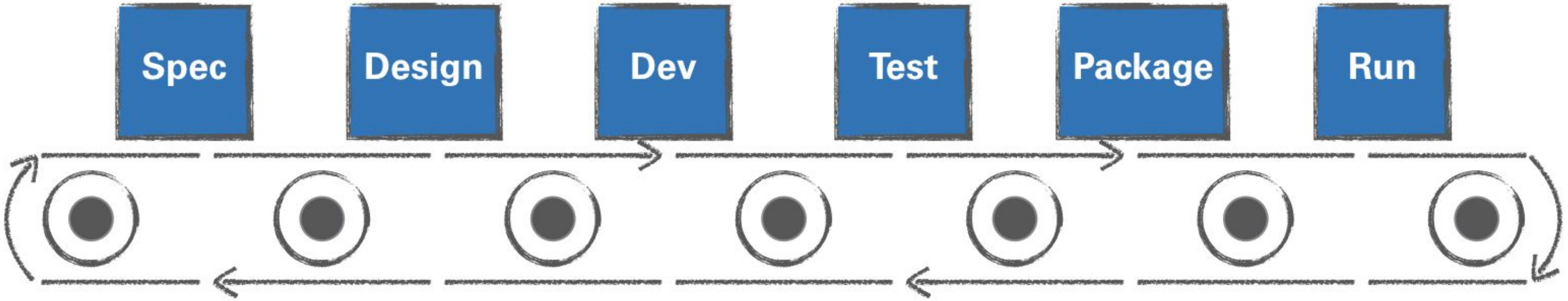
QA|WARE

Why Security Matters in Software Engineering

QA|WARE

- Cyber threats are **growing** – software is a primary attack vector.
- Security is **no longer optional** – breaches cost millions in damages and reputation.
- Regulations & compliance frameworks **demand accountability**.
- Customers & partners **expect secure software by default**.

Security must be built into **every stage** of software development.



What is ISO 27001? Why Should Software Engineers Care?

QA|WARE

- **ISO 27001** is an **international standard** for an information security management system (ISMS)
- Provides a systematic approach to managing information security risks
- Many organizations **require** and demand ISO 27001 for compliance.
- Helps to build **trust** with customers, regulators, and stakeholders.

📜 Key Components of ISO 27001

- Risk management & threat mitigation.
- Security policies & governance.
- Technical & operational controls to protect data.

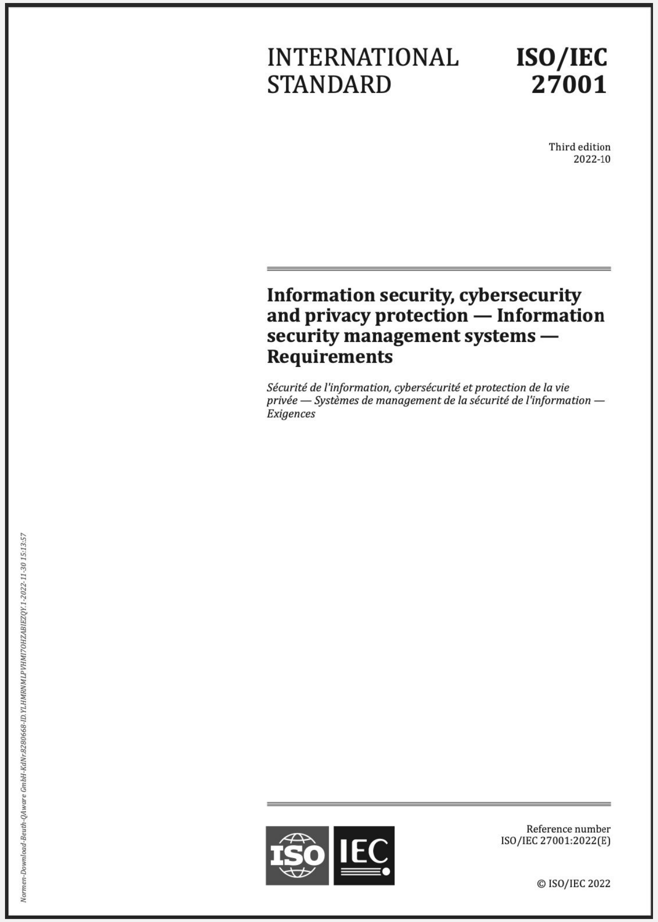
✅ Software Engineers' Role

- Implement secure development practices **(ISO 27001 Annex A.8).**
- Ensure **code, dependencies, deployment pipelines** and infrastructure are secure.
- Automate security controls **within CI/CD pipelines**.

ISO 27001:2022 Controls from Annex A.8

QA|WARE

- Access (1 - 5)
- Operations (6 - 9)
- Data Protection (10 - 13)
- Administration (14 - 19)
- Network (20 - 24)
- Application (25 - 29)
- Change (30 - 33)
- Audit testing (34)


INTERNATIONAL STANDARD

ISO/IEC 27001

Third edition
2022-10

Information security, cybersecurity and privacy protection — Information security management systems — Requirements

Sécurité de l'information, cybersécurité et protection de la vie privée — Systèmes de management de la sécurité de l'information — Exigences

ISO IEC

Reference number
ISO/IEC 27001:2022(E)

© ISO/IEC 2022

- Organizational (# = 37)
- People (# = 8)
- Physical (# = 14)
- **Technological (#=34)**

Annex A.8

Mapping of A.8 Technological Controls onto the SDLC

6, 9

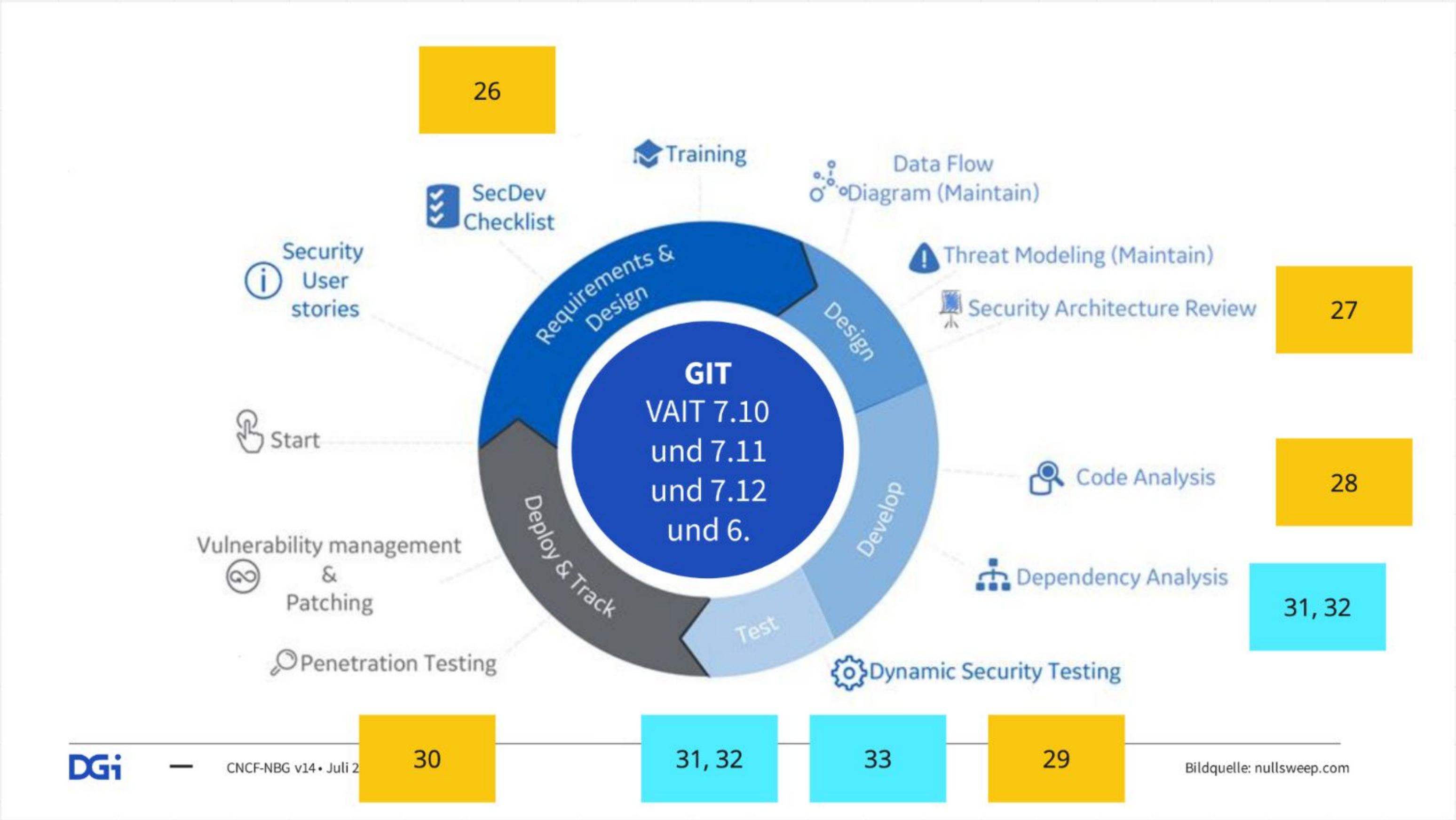
11, 13

2, 3, 5

14, 15

- Access
- Operations
- Data Protection
- Administration
- Network
- Application
- Change

10, 13



20, 21, 22, 24

2, 3, 4, 5

8

12

19

16, 18

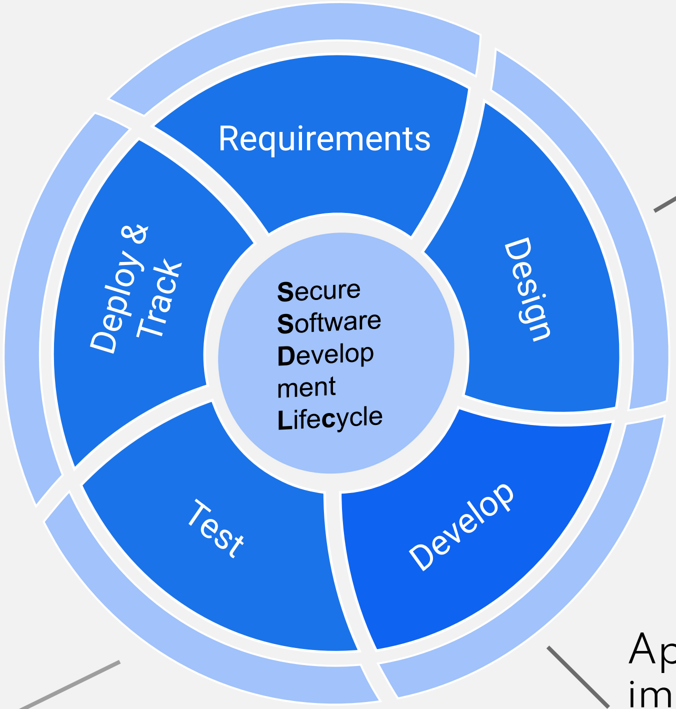
Our Secure Software Development Lifecycle (SSDLC)

Collect and analyse the system (security) requirements

Deployments only after sufficient test & review validation

Harmonize the design of software changes with the security architecture

Recognising and resolving potential security bugs and incidents during operation



Develop automated security tests
Guidelines (extract):

- Use of SAST mandatory (e.g. Sonarqube, Trivy, etc.)
- Use of DAST highly recommended

Apply secure coding practices during implementation
Guidelines (extract):

- Review of changes with security relevance
- We do not implement cryptographic algorithms or security mechanisms ourselves

When is the right time for a threat analysis?

QA|WARE

Sir, we‘ve analyzed
their attack pattern
and there is a danger.

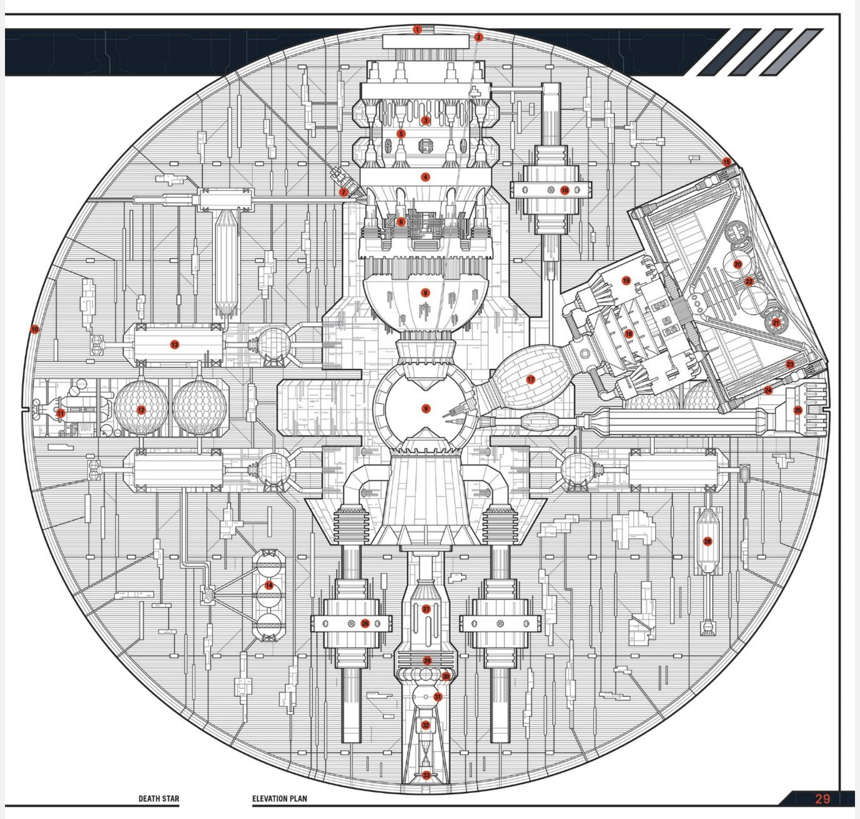
~ 50% of all threats are rooted in the specification and design

- With potentially serious consequences in implementation and operation
- Difficult to find automatically
- What is needed:

  An explorative method that can be used to identify threats at the design stage

  Threat analysis (threat modelling)

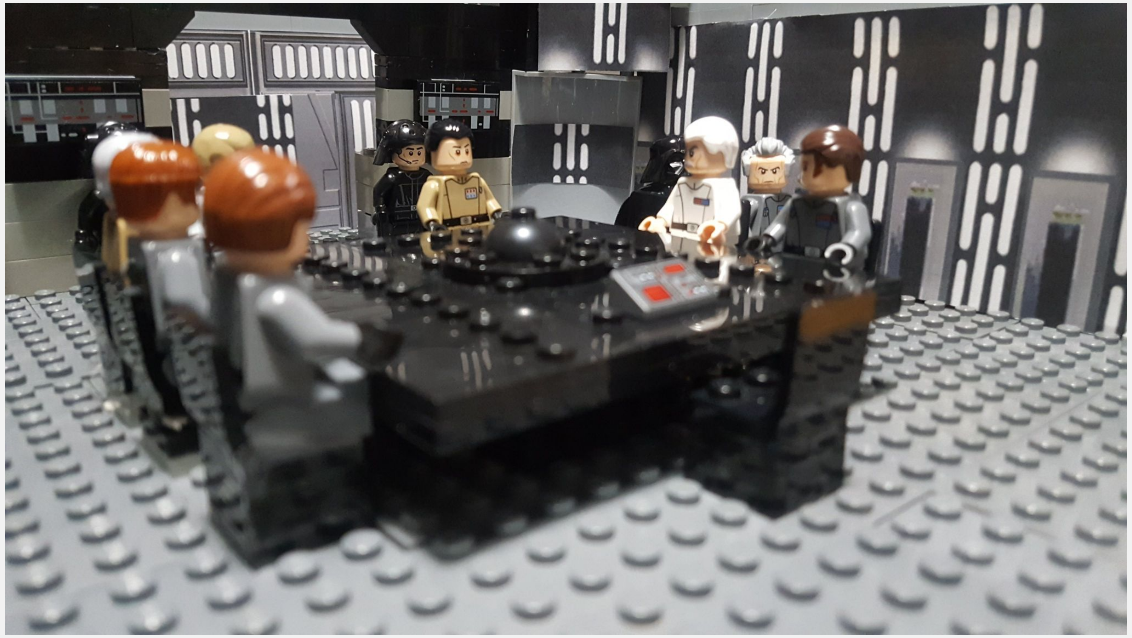
QA|WARE



Threat modeling is an analysis of representations of a system in a group of informed people to find concerns about security.

4 key 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 enough job?



Threat modeling manifesto :
We have come to value…

- **A culture of finding and fixing design issues** *over checkbox compliance.*
- **People and collaboration** *over processes, methodologies, and tools.*
- **A journey of understanding** *over a security or privacy snapshot.*
- **Doing threat modeling** *over talking about it.*
- **Continuous refinement** *over a single delivery.*

Who should threat model?

You. Everyone. Anyone who is concerned about the privacy, safety and security of the system.

https://www.threatmodelingmanifesto.org/

All possible attacks cannot be thought through!

- It is therefore advisable to analyse attack patterns
- **STRIDE** is frequently used - developed by *Loren Kohnfelder at Microsoft*
- Implementation in regular workshops with architects and product owners, among others
- "Whiteboard hacking"

- **STRIDE** structures threats into the following **6 attack patterns**:
  - **S**poofing
  - **T**ampering
  - **R**epudiation
  - **I**nformation Disclosure
  - **D**enial of Service
  - **E**levation of Privilege

STRIDE Attack Patterns – Spoofing

QA|WARE

Feigning a false identity

STRIDE Attack Pattern – Tampering

Manipulation of data and code

STRIDE Attack Pattern – Repudiation

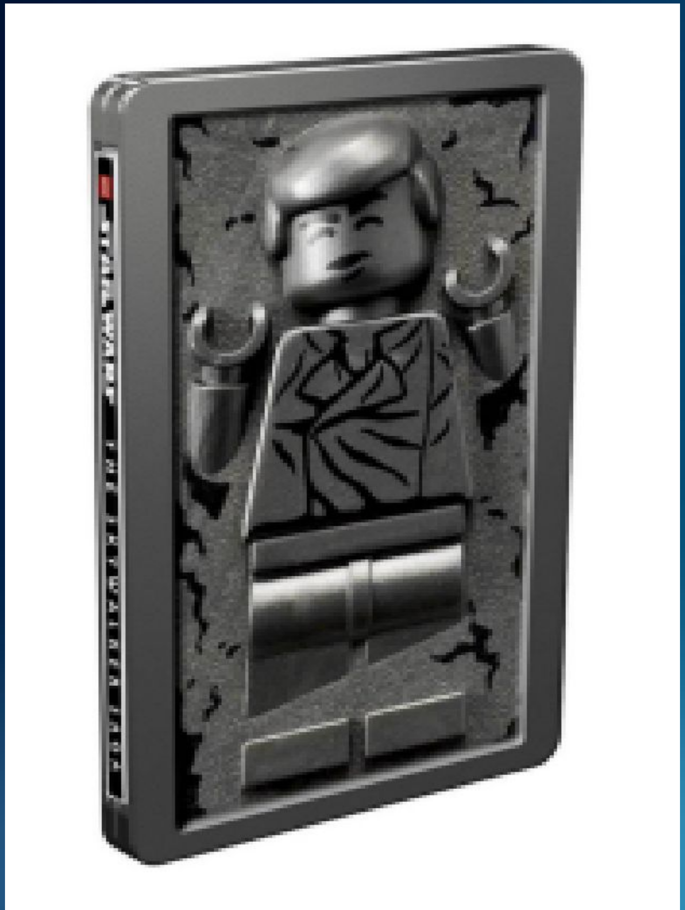
Denial of identity or information

STRIDE Attack Pattern – Information Disclosure

QA|WARE

Disclosure and dissemination of data

STRIDE Angriffsmuster – Denial of Service



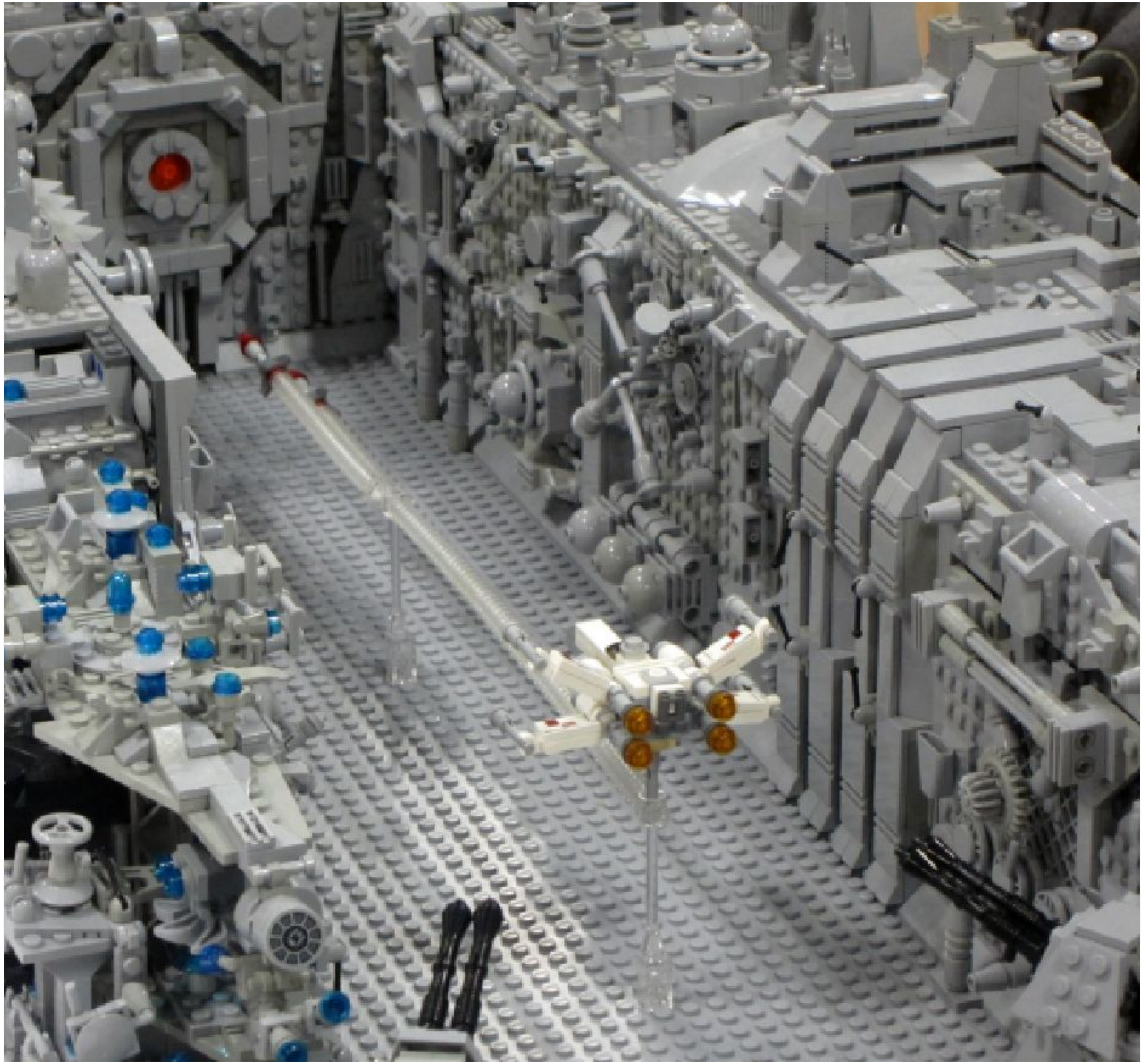
Disruption to the availability of functions or data

STRIDE Attack Pattern – Elevation of Privilege

QA|WARE

Appropriation of a role or authorisation

…and the consequences



STRIDE as a structured approach

- Document architecture (diagrams)
- Identify and evaluate threats
  - Some diagram elements (process, data store, data flow, external entity) only susceptible to certain attack patterns
  - Evaluate according to impact and probability, among other things
- Define countermeasures
  - Mitigation: Mitigate or complicate threat
  - Avoid: Delete feature or change architecture
  - Transfer: Transfer to someone else
  - Accept: mostly out of cost-benefit considerations
- Feedback and iteration

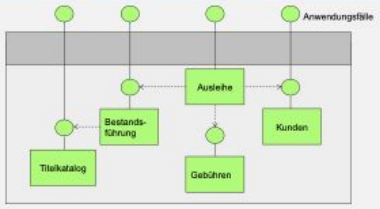
Element	Spoofing	Tampering	Repudiation	Information Disclosure	Denial of Service	Elevation of Privilege
Data Flows		X		X	X	
Data Stores		X		X	X	
Processes	X	X	X	X	X	X
Interactors	X		X			

Three views of software architecture define the fields of clean code, clean architecture, quality assurance and security.

QA|WARE

Conceptual

Goal: Structures the system from a business perspective

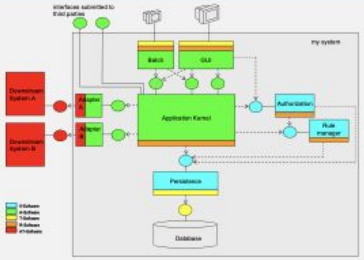


Typical Inputs:
- Data Models
- Use Cases
- Functional Requirements

Typical Outputs:
- Business Domain Components
- Business Domain Interfaces
- System Context (Embeddingt into the system landscape)

Technical

Goal: Defines how to map and run the conceptual view on the infrastructure

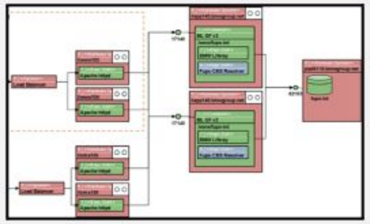


Typical Inputs:
- Non-functional Requirements
- Enterprise Standards
- Reference Architectures, Patterns and Blueprints

Typical Outputs:
- Layers, Onions, Hexagons, …
- Reusable common components (0-components) and technical components (T components)
- Selection of open source as building blocks for T/0-components (Software OEM)

Infrastructure

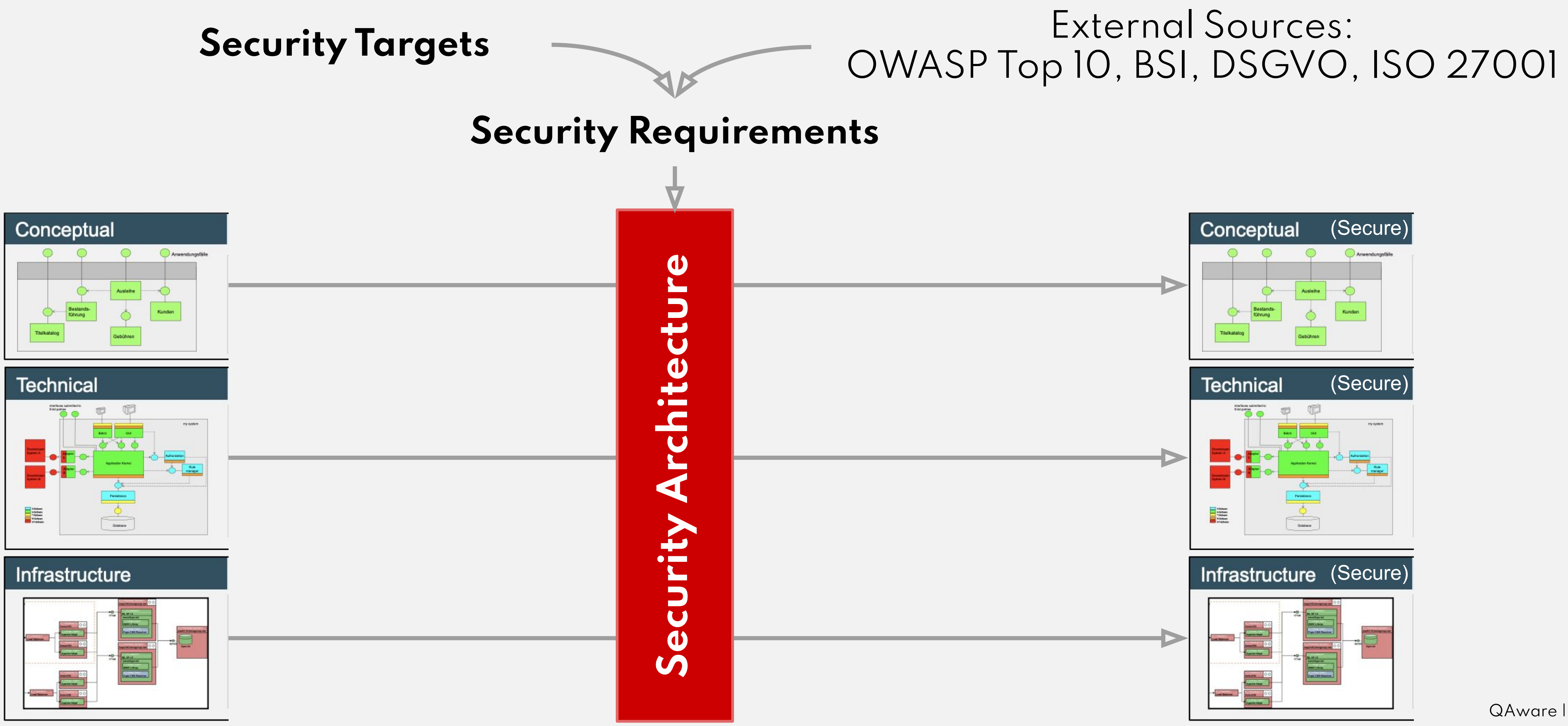
Gial: Defines the execution environment to run and operate our systems



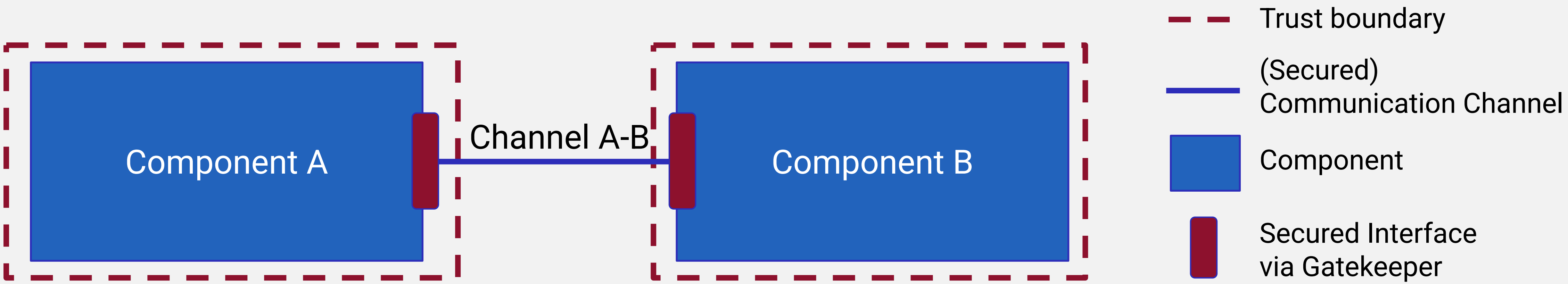
Typical Inputs:
- Non functional requirements
- Enterprise Standards
- Existing corporate infrastructure environments

Typical Outputs:
- Hardware Infrastructure (Compute, Network, Storage, …)
- Software Infrastructure (OS, DB, App-Servers, …) and protocols
- Ops components (individual executable pieces of the application)

The security architecture of a system defines how to secure the individual views of the overall architecture.

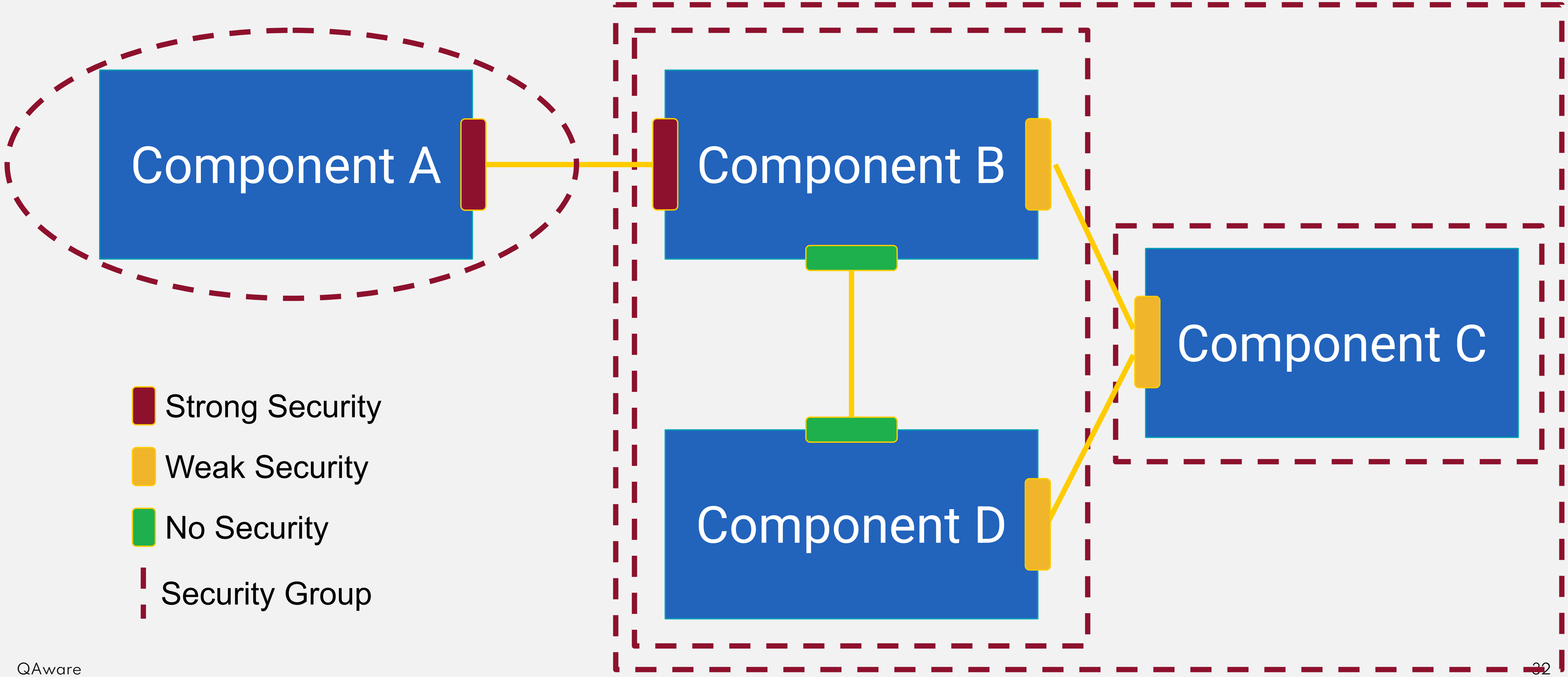


The security architecture consists of secure components and communication channels.

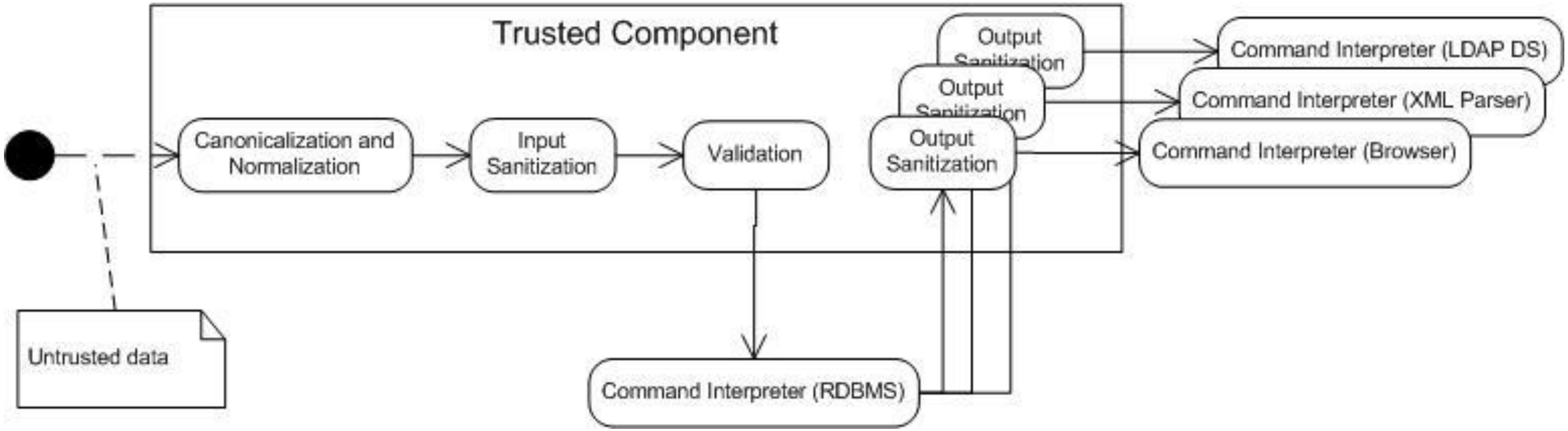


- **A system consists of components. These are connected by communication channels.**
  - Examples of components: Data centres, VMs, microservices, databases, software modules, …
  - Each component is provided by someone who is trusted or untrusted.
- **Each component has a defined level of security. From insecure to very secure.**
  - How thorough does the gatekeeper need to be: from everyone's right to a fortress
- **Each channel has a defined level of security. From very secure to insecure.**
  - How robust is the channel and the protocol used in it against typical attacks?

Security components can form security groups with tough border controls and lax internal security.



The internal design of a trusted component is influenced by the security requirements.

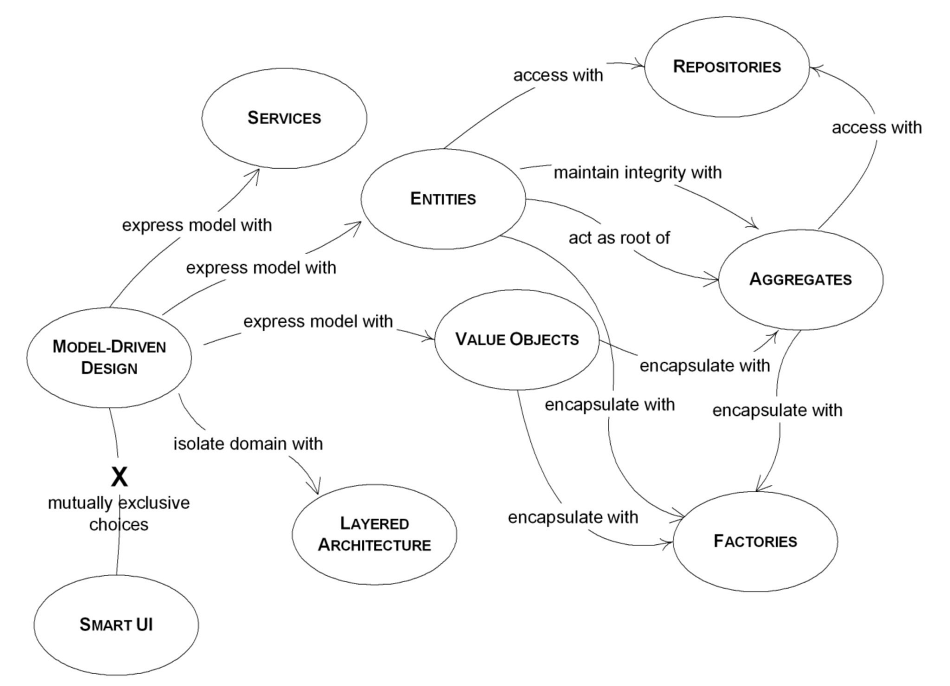


https://wiki.sei.cmu.edu/confluence/plugins/servlet/mobile?contentId=88487694#content/view/88487694

- Canonicalisation
  - Lossless simplification of the representation.
- Normalisation
  - Lossy simplification of the representation.
- Sanitization
  - Removal of nonsensical and harmful data values
- Validation
  - Type check and value range check

Some concepts of Domain Driven Design can ensure a robust and secure design.

- **Value Objects**
  - Are defined by value and are immutable.
  - Can contain other VOs.
  - Can be used as attributes in entities
  - Define and check important constraints.
- **Aggregates**
  - Controls access from outside
  - Ensures consistency within the boundary
  - Access via repositories



Clean Code & Architecture

General

1. **Follow standard conventions.**
2. Keep it simple stupid. Simpler is always better. Reduce complexity as much as possible.

Comments

1. **Always try to explain yourself in code.**
2. Don't be redundant.
3. **Don't add obvious noise.**
4. Don't comment out code. Just remove.
5. **Use as clarification of code.**
6. **Use as warning of consequences.**

Design

1. Keep configurable data at high levels.
2. **Prevent over-configurability.**
3. **Use dependency injection.**

Names

1. Don’t be funny.
2. **Choose descriptive and unambiguous names.**
3. Use pronounceable names.
4. Use searchable names.

Data Structures

1. **Prefer data structures.**
2. Hide internal structure.
3. Should be small.
4. **Small number of variables.**

Understandability

1. **Be consistent. If you do something a certain way, do all similar things in the same way.**
2. Use explanatory variables.
3. Avoid negative conditionals.

Source Code

1. Keep lines short.
2. **Use white space to associate related things and disassociate weakly related.**
3. Don't break indentation.
4. Dependent definitions should be close.
5. **Structure should clearly express modules, layers, components or conceptual architecture.**

Tests

1. **F**ast.
2. **I**ndependent.
3. **R**epeatable.
4. **S**elf-validating
5. **T**imely

https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html
Good overall summary can be found at https://gist.github.com/wojteklu/73c6914cc446146b8b533c0988cf8d29

lreimer/iso27001-secure-se

Google ErrorProne

Find common programming mistakes early during development as part of the Java compile phase.

```
plugins {
    id 'java'
    id "net.ltgt.errorprone" version "3.1.0"
}

dependencies {
    // dependency for the javac compiler plugin
    errorprone "com.google.errorprone:error_prone_core:2.19.1"
}

tasks.named("compileJava").configure {
    options.errorprone.enabled = true
    // and many other options
}
```

SonarCloud Security Analysis

QA|WARE

Sonar can detect 54 security vulnerabilities and 38 security hotspots using static code analysis.

```
plugins {
    id "jacoco"
    id "org.sonarqube" version "6.0.1.5171"
}

jacocoTestReport {
    reports { xml.required = true }
}

sonarqube {
  properties {
    property "sonar.projectKey", "lreimer_iso27001-secure-se"
    property "sonar.organization", "lreimer"
    property "sonar.host.url", "https://sonarcloud.io"
  }
}
```

Docker Image Vulnerability Scanning

QA|WARE

Several suitable tools can be used to scan your Docker images for vulnerable OS packages and other software components.

```
# Installation and usage instructions for Docker Lint
# https://github.com/projectatomic/dockerfile_lint
dockerfile_lint -f Dockerfile -r src/test/docker/basic_rules.yaml
dockerfile_lint -f Dockerfile -r src/test/docker/security_rules.yaml

# Installation and usage instructions for Trivy
# https://github.com/aquasecurity/trivy
trivy image -s HIGH,CRITICAL iso27001-service:1.0.0

# Installation and usage instructions for Snyk
# https://docs.snyk.io/snyk-cli/install-the-snyk-cli
snyk container test --file=Dockerfile iso27001-service:1.0.0
```

Kubernetes Security Scanning

QA|WARE

Many security misconfigurations are possible when deploying Kubernetes workloads. Most can be found easily via static code analysis using different tools.

```
# see https://github.com/zegl/kube-score
kubectl score src/main/k8s/base/microservice-deployment.yaml

# Checkov, see https://github.com/bridgecrewio/checkov
checkov --directory src/main/k8s/base
checkov --directory src/main/k8s/overlays/int

# Snyk, see https://docs.snyk.io/snyk-cli/install-the-snyk-cli
snyk iac test src/main/k8s/base
snyk iac test src/main/k8s/overlays/int

# Trivy, see https://github.com/aquasecurity/trivy
trivy src/main/k8s -n default --report summary all
trivy src/main/k8s -n default --report all all
```

Terraform Security Scanning

QA|WARE

Many security misconfigurations of your cloud infrastructure are possible when working with Terraform.
Most can be found easily via static code analysis using different tools.

```
# TFLint und Rule Sets
# see https://github.com/terraform-linters/tflint
# see https://github.com/terraform-linters/tflint-ruleset-aws
terraform init
terraform plan
tflint

# Checkov
# see https://github.com/bridgecrewio/checkov
checkov --directory src/main/terraform

# Snyk
# https://docs.snyk.io/snyk-cli/install-the-snyk-cli
snyk iac test src/main/terraform/
```

Continuous Developer Experience

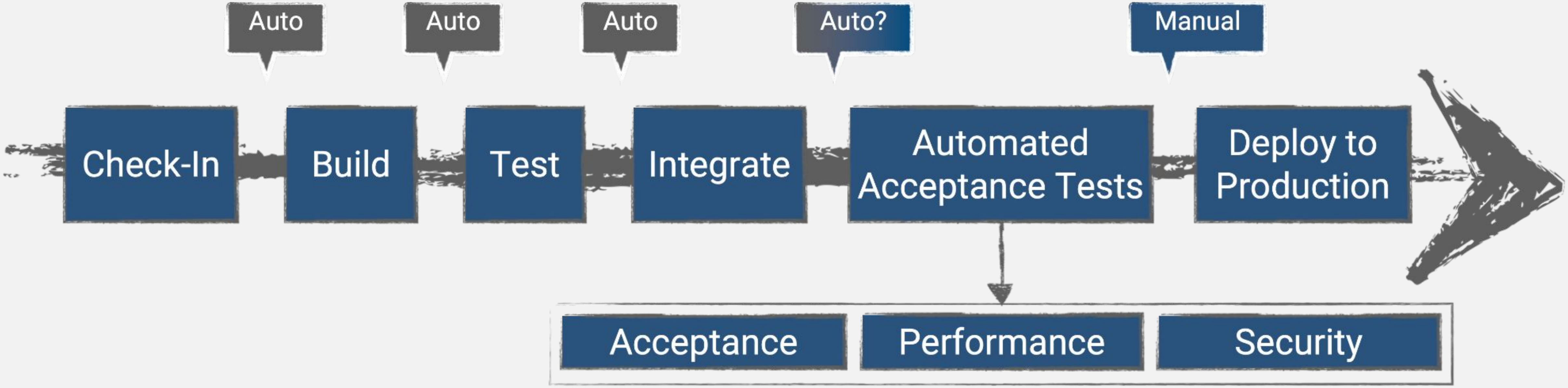
QA|WARE

The linters and static analysis tools are ideally run before and with every Git commit and push. Also GitHub and many other platforms provide CI and security integration functionality that can be used.

```
# see https://github.com/pre-commit/pre-commit
brew install pre-commit

# see https://pre-commit.com/hooks.html
# see https://github.com/gruntwork-io/pre-commit
# see https://github.com/antonbabenko/pre-commit-terraform

# install the Git hook scripts
pre-commit install
pre-commit run --all-files

# see https://github.com/lreimer/iso27001-secure-se/actions
# see https://github.com/lreimer/iso27001-secure-se/actions/new?category=security
```

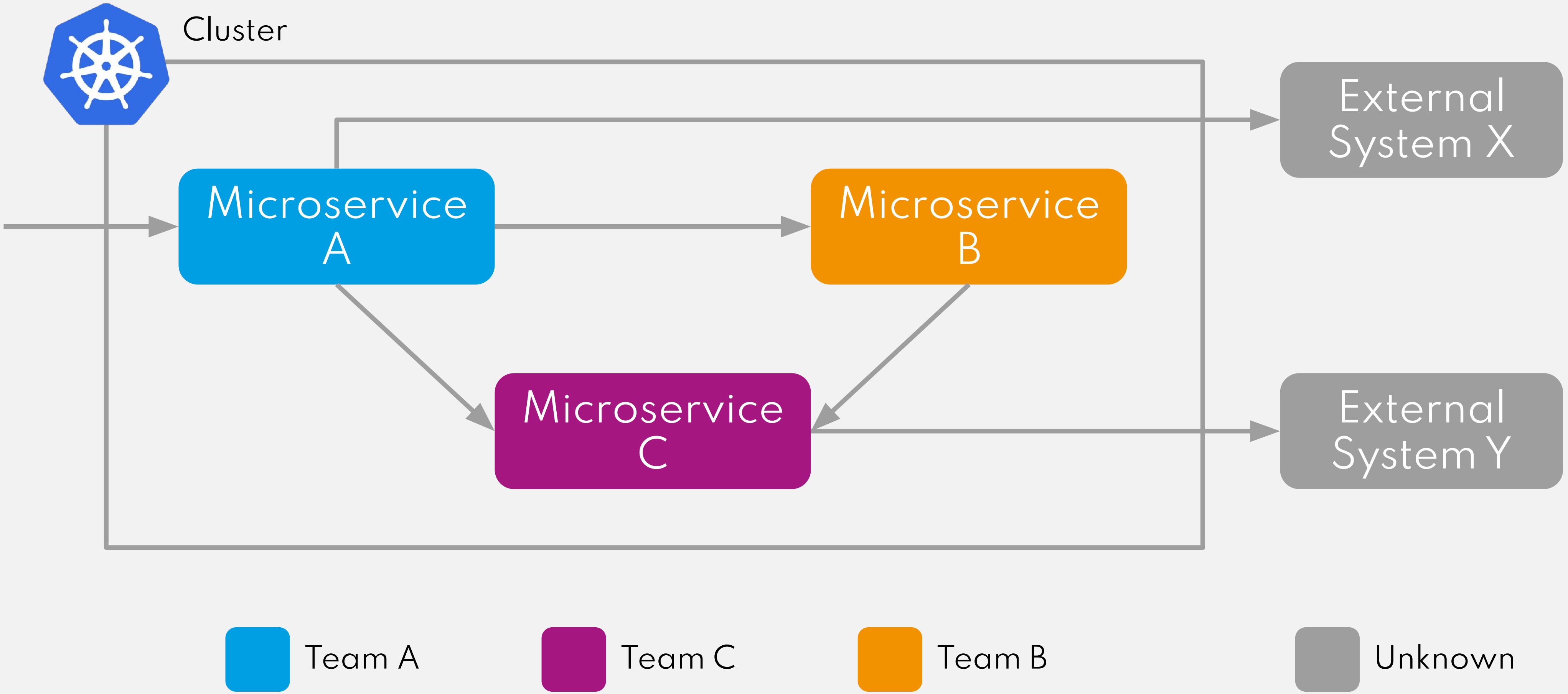
Monolithic, linear CI/CD pipelines are suboptimal and will result in delayed feedback and long release cycles.

QA|WARE



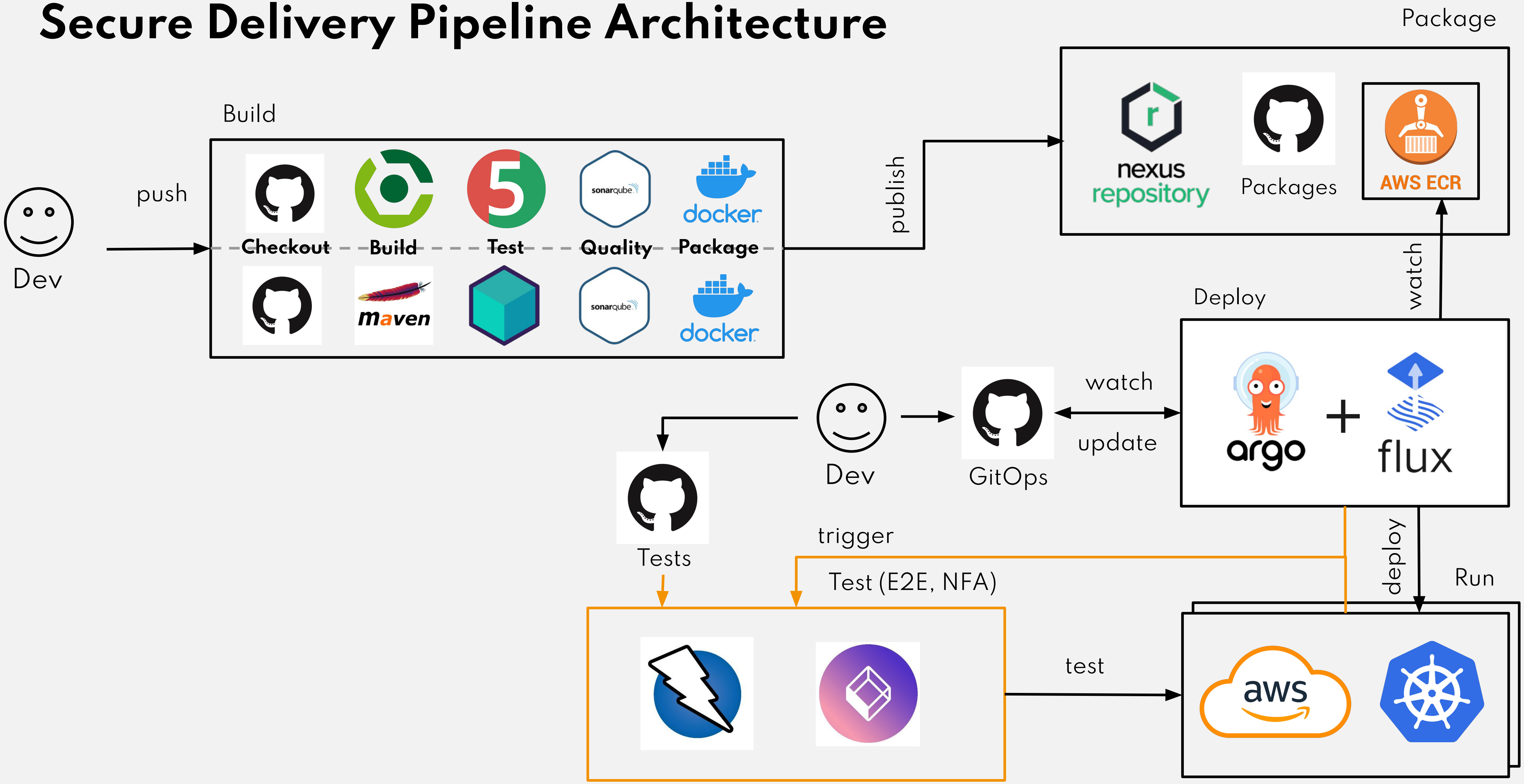
At the beginning often performed in parallel.
Later on, delayed until the end of sprint or the release.
Functionality vs. Performance vs. Security?
Which one first?

A microservice architecture with many downstream dependencies is complex and really hard to test.




DOING SO YOU MAY,
BUT GOING TO SUCK IT WILL.

Secure Delivery Pipeline Architecture

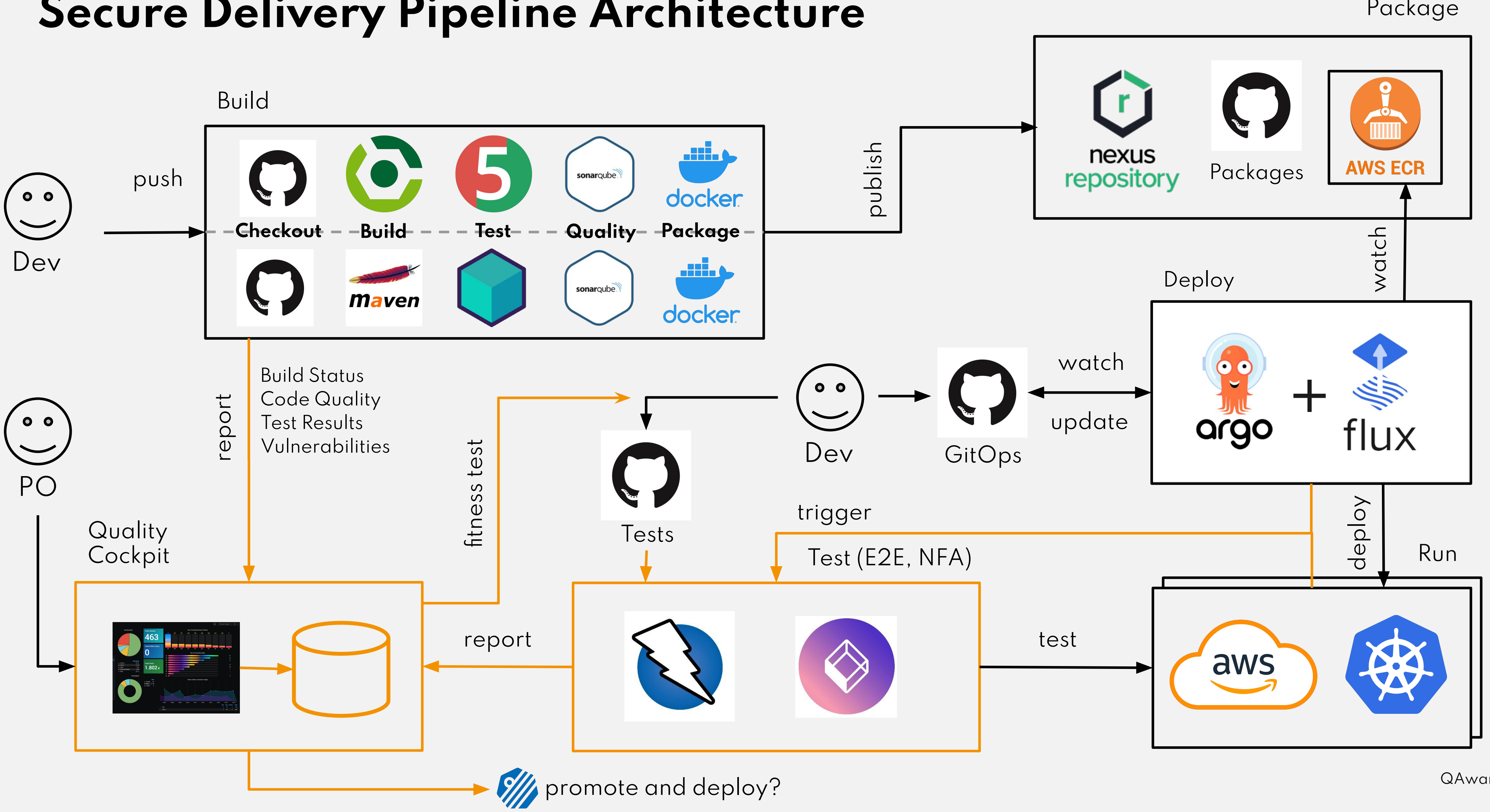


Zed Attack Proxy (ZAP)

QA|WARE

- Widespread and well-known open source web application vulnerability scanner
- Detailed documentation. International community.
- Several modes of operation: Intercepting Proxy, Active und Passive scanner, HTTP Spider, Brute Force Scanner, Port Scanner, OpenAPI v3, SOAP, GraphQL, Web Sockets
- ZAP provides a powerful API and tools for Security Scanning Automation
- The official ZAP Docker images provide an easy way to run ZAP, especially in CI/CD and container runtime environments such as Kubernetes
  - API Scan - a full scan of an API defined using OpenAPI / Swagger, or GraphQL
  - Baseline Scan - a time limited spider which reports issues found passively
  - Full Scan - a full spider, optional ajax scan and active scan which reports issues found
  - Webswing - run the ZAP Desktop UI in a browser
- https://www.zaproxy.org/docs/

Secure Delivery Pipeline Architecture



OWASP SAMM: Measuring & Improving Security Maturity

QA|WARE

Software Assurance Maturity Model (SAMM)

- Open-source framework by **OWASP** for improving software security.
- Helps organizations **assess, measure, and improve** security practices.
- **Aligns with ISO 27001** for secure development maturity.
- https://owaspsamm.org

📢 Why SAMM?

- ISO 27001 tells you WHAT to do (security controls).
- OWASP SAMM helps with HOW to do it (practical implementation).

OWASP SAMM: Core Structure and Model

QA|WARE

Governance	Design	Implementation	Verification	Operations
Strategy and Metrics	Threat Assessment	Secure Build	Architecture Assessment	Incident Management
Policy and Compliance	Security Requirements	Secure Deployment	Requirements-driven Testing	Environment Management
Education and Guidance	Secure Architecture	Defect Management	Security Testing	Operational Management

SAMM Maturity Levels

- Level 1: Basic security controls are in place
- Level 2: Security practices are documented and consistently applied
- Level 3: Security practices are fully integrated and optimized

- Mapping SAMM practices to ISO 27001 helps teams track progress and continuously improve security maturity.
- Use SAMM alongside ISO 27001 to drive continuous security improvements.

Steps to Adopt OWASP SAMM in Your Team & Organization

QA|WARE

Step 1: Assess Current Security Maturity

- Use the OWASP SAMM self-assessment tool.
- Identify strengths and gaps in your current security practices.

Step 2: Define Security Goals

- Set **realistic** security maturity targets.
- Align goals with ISO 27001 compliance requirements.

Step 3: Implement Enhancements

- Improve security policies, secure coding standards, testing automation.
- Automate security controls within CI/CD.

Step 4: Measure and Integrate

- Continuously track progress using SAMM maturity score.
- Conduct regular assessments and improve weak areas.

Security maturity is a journey - start small, track progress, and continuously improve.

Security is one of several software quality attributes. Don't treat it as 2nd class citizen! Secure by Design from Day 1!

QA|WARE

Software Product Quality (ISO 25010)

Security
- Confidentiality
- Integrity
- Non-repudiation
- Authenticity
- Accountability

Efficiency
- Time Behaviour
- Resource Utilization
- Capacity

Maintainability
- Modularity
- Reusability
- Analysability
- Modifiability
- Testability

Functional Suitability
- Completeness
- Correctness
- Appropriateness

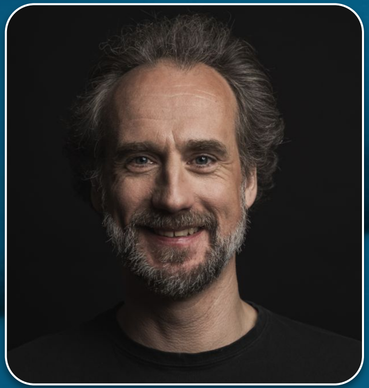
Portability
- Adaptability
- Installability
- Replaceability

Usability
- Operability
- Learnability
- UI Aesthetics
- Accessibility

Reliability
- Maturity
- Availability
- Fault Tolerance
- Recoverability

Compatibility
- Co-existence
- Interoperability

Thank you!
The next step? Let's talk.



Mario-Leander Reimer
Managing Director, CTO

mario-leander.reimer@qaware.de
+49 151 61314748

QA|WARE
SOFTWARE ENGINEERING