

Von Sicherheitslücken und Compliance – Wie man Security mit SAST erfolgreich umsetzt

JFS 2026

CQSE

Dr. Tobias Röhm



Dr. Tobias Röhm



Freelance-
Entwickler



2004 - 2010

2010 - 2015

2015 - Heute



Häufigkeit von Cyberangriffen

Cyberangriff auf Klinikum Esslingen gelang über Schwachstelle in Citrix-Zugang

Sicherheitslücke auch bei AOK im Saarland

„Wir erleben jede Minute mehr als 5.000 Cyberangriffe“

Interview mit dem Tagesspiegel



24.01.2026 | Joachim Nagel EN

...

Schaden durch Cyberangriffe

Unerwünschte Einblicke: Fataler Fehler bei Netatmo-Sicherheitskameras

Daten von Tausenden Bankkunden abgegriffen

Behörde für abgesicherte Ausweise geknackt – Millionen Franzosen betroffen

23.04.2026 23:01 Uhr 59 | heise Security

...

→ Hohes Risiko durch Cyberangriffe

ANHANG I

GRUNDLEGENDE CYBERSECURITYANFORDERUNGEN



Teil I Cybersicherheitsanforderungen in Bezug auf die Eigenschaften von Produkten mit digitalen Elementen

- (1) Produkte mit digitalen Elementen werden so konzipiert, entwickelt und hergestellt, dass sie angesichts der Risiken ein angemessenes Cybersicherheitsniveau gewährleisten.
- (2) Auf der Grundlage der Bewertung der Cybersicherheitsrisiken gemäß Artikel 13 Absatz 2 müssen Produkte mit digitalen Elementen, soweit zutreffend,
 - a) ohne bekannte ausnutzbare Schwachstellen auf dem Markt bereitgestellt werden,

Teil II Anforderungen an die Behandlung von Schwachstellen

- (3) die Sicherheit des Produkts mit digitalen Elementen regelmäßig und wirksam testen und überprüfen;

Cyber Resilience Act (CRA, Auszüge)

- (3) Im Rahmen des in Absatz 2 genannten Verfahrens sind Quellcodeprüfungen durchzuführen, die sowohl statische als auch dynamische Tests umfassen. Bei dies muss die Sicherheit internetexponierter Systeme und Anwendungen gemäß Artikel 8 Absatz 2 Buchstabe b Ziffern v, vi und vii getestet werden. Finanzunternehmen m
 - a) Schwachstellen und Anomalien im Quellcode ermitteln und analysieren,
 - b) einen Aktionsplan festlegen, um diese Schwachstellen und Anomalien zu beheben,
 - c) die Umsetzung dieses Aktionsplans überwachen.

Digital Operational Resilience Act (DORA, Auszüge)



Maschinenverordnung



Radio Equipment Directive



IEC 62443



NIS2

→ Viele regulatorische Vorgaben bezüglich Security



KRITIS

Fokus heute:
SAST (Static Application Security Testing) für Code-Security

Anforderungen an Software-Organisation

Sicherheitslücken vermeiden



Entwickler:in
IT-Security

Risikoorientiertes, zeitgemäßes Vorgehen nachweisen



Compliance-Verantwortliche
IT-Security

Agenda



Kurzübersicht »Static Application Security Testing (SAST)«



SAST für Entwickler:innen



SAST für Compliance-Verantwortliche

Agenda



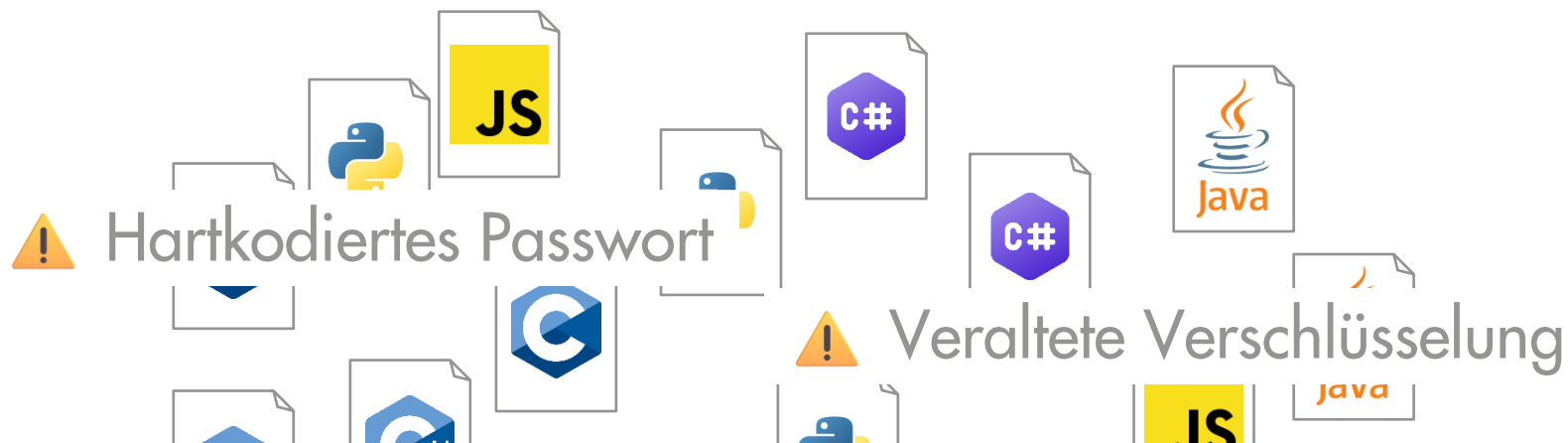
Kurzübersicht »Static Application Security Testing (SAST)«



SAST für Entwickler:innen



SAST für Compliance-Verantwortliche



Static Application Security Testing (SAST)



```

27
28  /**
29   * Provides the Basic Authorization header to a request.
30   */
31  public class BasicAuthorizationProvider implements AuthorizationProvider {
32      private static final String[] PREFIXES = {"log4j2.config.", "logging.auth."};
33      private static final String AUTH_USER_NAME = "username";
34      private static final String AUTH_PASSWORD = "password";
35      private static final String AUTH_PASSWORD_DECRYPTOR = "passwordDecryptor";
36      public static final String CONFIG_USER_NAME = "log4j2.configurationUserName";
37      public static final String CONFIG_PASSWORD = "log4j2.configurationPassword";
38      public static final String PASSWORD_DECRYPTOR = "log4j2.passwordDecryptor";
39
40      private static Logger LOGGER = StatusLogger.getLogger();
41
42      private String authString = null;
43
44      public BasicAuthorizationProvider(PropertiesUtil props) {

```

```

3452     when 'P' then
3453         [redacted]
3454         req := utl_http.begin_request('http://[redacted]');
3455         utl_http.set_authentication(r => req, username => '[redacted]', password => '[redacted]');
3456     end case;
3457

```

! Mögliches Path-Traversal

```
6  REPORT write_conf_to_fileserv.  
7  
8  PARAMETERS: p_vbeln  TYPE vbeln,  
9              p_posnr  TYPE posnr,  
10             p_server TYPE string DEFAULT '\\[REDACTED]\Configs\'.  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21  
22  
23  
24  DATA(fileserv) = |{ p_server }|{ p_vbeln }_|{ p_posnr }|.  
25  OPEN DATASET fileserv FOR OUTPUT IN BINARY MODE.  
26  CHECK sy-subrc = 0.  
27  TRANSFER xstring TO fileserv.  
28  CLOSE DATASET fileserv.
```



☐ ▼ Critical and Suspicious Statements	2450	☐ ▼ Insufficient Authority Checks	2	☐ ▼ Unsanitized User Input	2318
☐ Information Exposure Through An Error Message	9	☐ Empty database password	1	☐ HTTP cookie formed from untrusted input	10
☐ Overly permissive file permission	2	☐ Missing authentication for critical function (database) (Java)	1	☐ HTTP headers untrusted	432
☐ Potentially sensitive data in a cookie	2403	☐ ▼ Potential Injection	1758	☐ Potential HTTP Response Splitting	67
☐ Sensitive cookie in HTTPS session without 'Secure' attribute (Java)	36	☐ A prepared statement is generated from a nonconstant String	97	☐ Trust Boundary Violation	118
☐ ▼ Cross-Client Access	885	☐ Nonconstant string passed to execute or addBatch method on an SQL statement	178	☐ URLConnection Server-Side Request Forgery (SSRF) and File Disclosure	1
☐ Potential XSS in Servlet	884	☐ Potential Command Injection	237	☐ Untrusted Referer header	82
☐ Servlet reflected cross site scripting vulnerability	1	☐ Potential JDBC Injection	310	☐ Untrusted query string	295
☐ ▼ Directory Traversal	873	☐ Potential JDBC Injection (Spring JDBC)	153	☐ Untrusted servlet parameter	1313
☐ Absolute path traversal in servlet	1	☐ Potential LDAP Injection	54	☐ ▼ Weak Cryptography	1294
☐ Potential Path Traversal (file read)	584	☐ Potential SQL/HQL Injection (Hibernate)	2	☐ Cipher is susceptible to Padding Oracle	72
☐ Potential Path Traversal (file write)	38	☐ Potential XPath Injection	34	☐ Cipher with no integrity	97
☐ Relative path traversal in servlet	4	☐ Usage of an unchecked URL object can be insecure	1	☐ Cookie without the secure flag	36
☐ Improper limitation of a pathname to a restricted directory ('Path Traversal') (Java) (SpotbugsPathTraversalAbsolute)	246	☐ Improper neutralization of CRLF sequences in HTTP headers ('HTTP Response Splitting') (Java) (HttpServletResponseSplitting)	28	☐ DES is insecure	227
☐ ▼ External Entities	1	☐ Improper neutralization of special elements used in an OS command ('OS Command Injection') (Java)	234	☐ DESede is insecure	33
☐ XML parsing vulnerable to XXE (DocumentBuilder)	1	☐ Improper neutralization of special elements used in an LDAP query ('LDAP Injection') (Java)	55	☐ MD2, MD4 and MD5 are weak hash functions	28
☐ ▼ Hard-Coded Credentials	20	☐ Improper neutralization of input during web page generation ('Cross-site Scripting') (Java) (XSSReqParamToServletWriter)	375	☐ Predictable pseudorandom number generator	218
☐ Hard coded password	11			☐ SHA-1 is a weak hash function	85
☐ Hard-coded password	8			☐ Static IV	97
☐ A secure password should be used when connecting to a database (java:S2115)	1			☐ Use of a broken or risky cryptographic algorithm (SHA1/MD5) (Java)	113
				☐ Basic authentication should not be used (java:S2647)	1
				☐ Encryption algorithms should be used with secure mode and padding scheme (java:S5542)	157
				☐ Cipher algorithms should be robust (java:S5547)	130

Agenda



Kurzübersicht »Static Application Security Testing (SAST)«



SAST für Entwickler:innen



SAST für Compliance-Verantwortliche

Viele SAST-False-Positives

Wenig Security-Wissen

Wenig Zeit
im vollen Entwicklungsalltag

Qualitätssicherung für (viel) KI-Code



Entwickler:in
Emily

→ Alltägliche Vermeidung von Sicherheitslücken
statt Unsicherheit und Druck



Static Application Security Testing für Entwickler:innen

Einmalige Analyse



Kontinuierlicher Prozess



Static Application Security Testing für Entwickler:innen

Einmalig



Kritische Findings

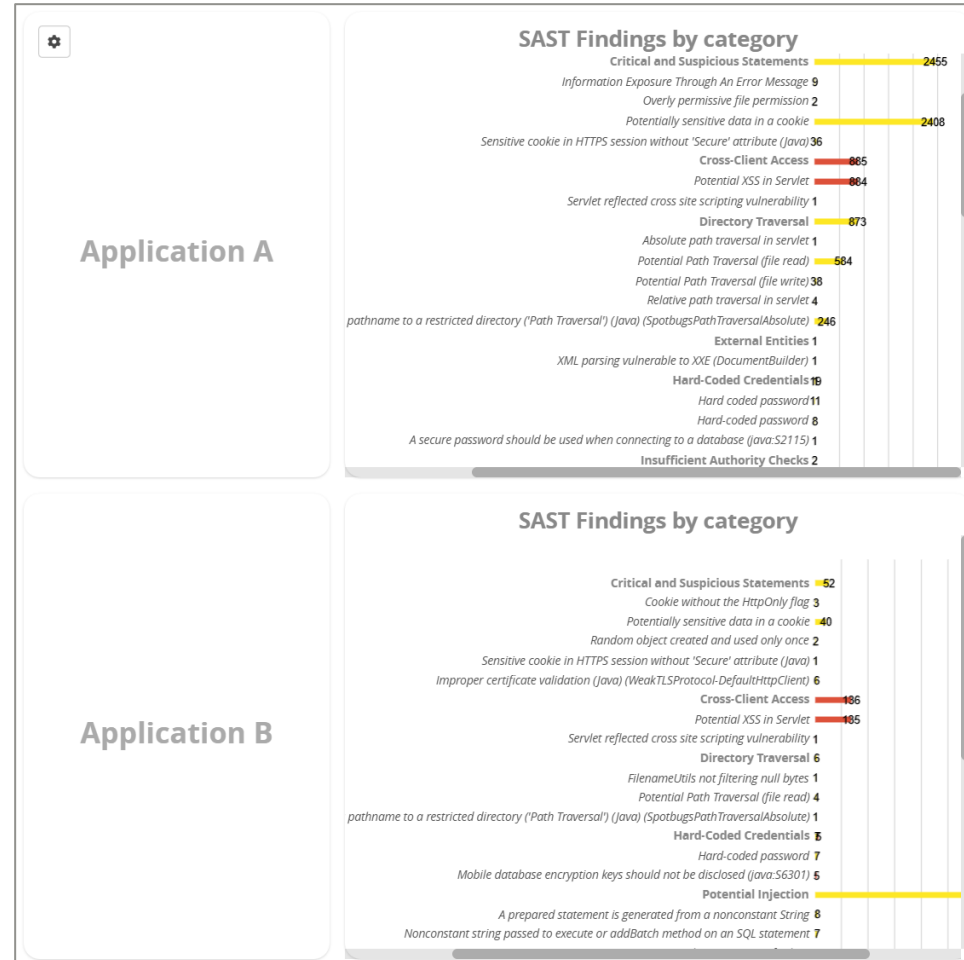
zuerst beheben



Kontinuierlich



Standortbestimmung



→ Standortbestimmung ermöglicht Behebung von kritischen Sicherheitslücken



Static Application Security Testing für Entwickler:innen

Einmalig



Kritische Findings

zuerst beheben



Security Hotspots

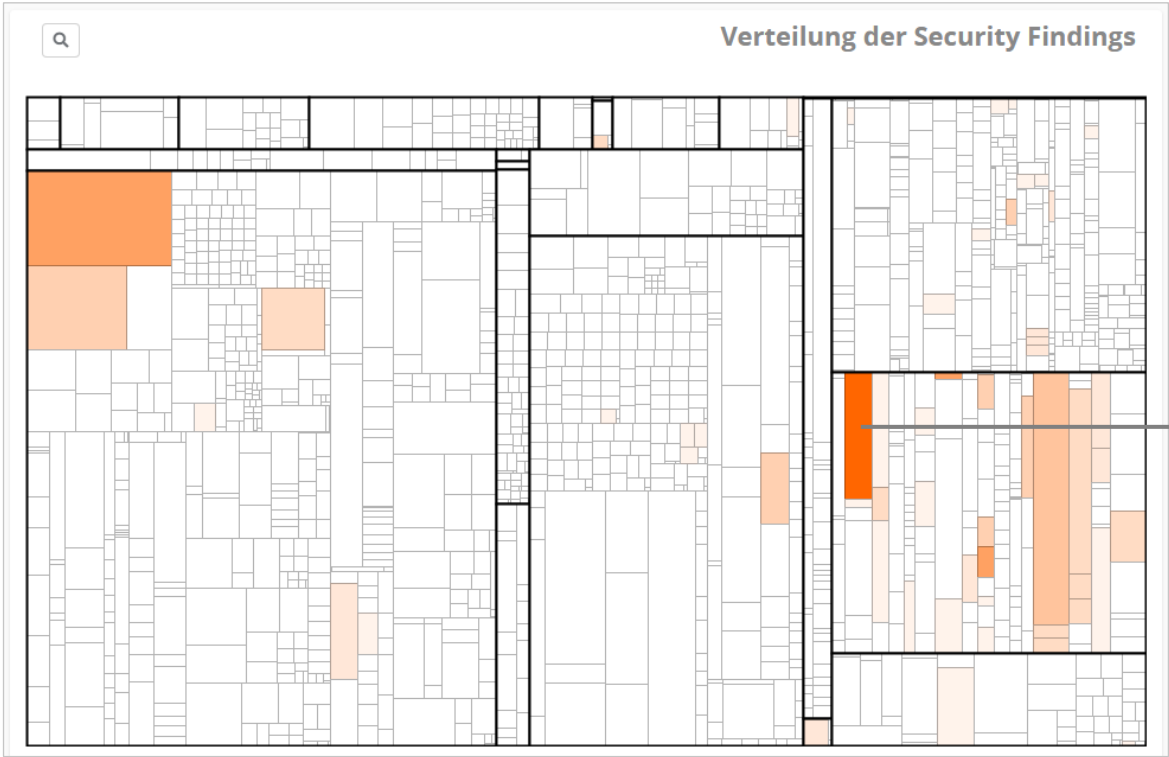
genauer betrachten



Kontinuierlich



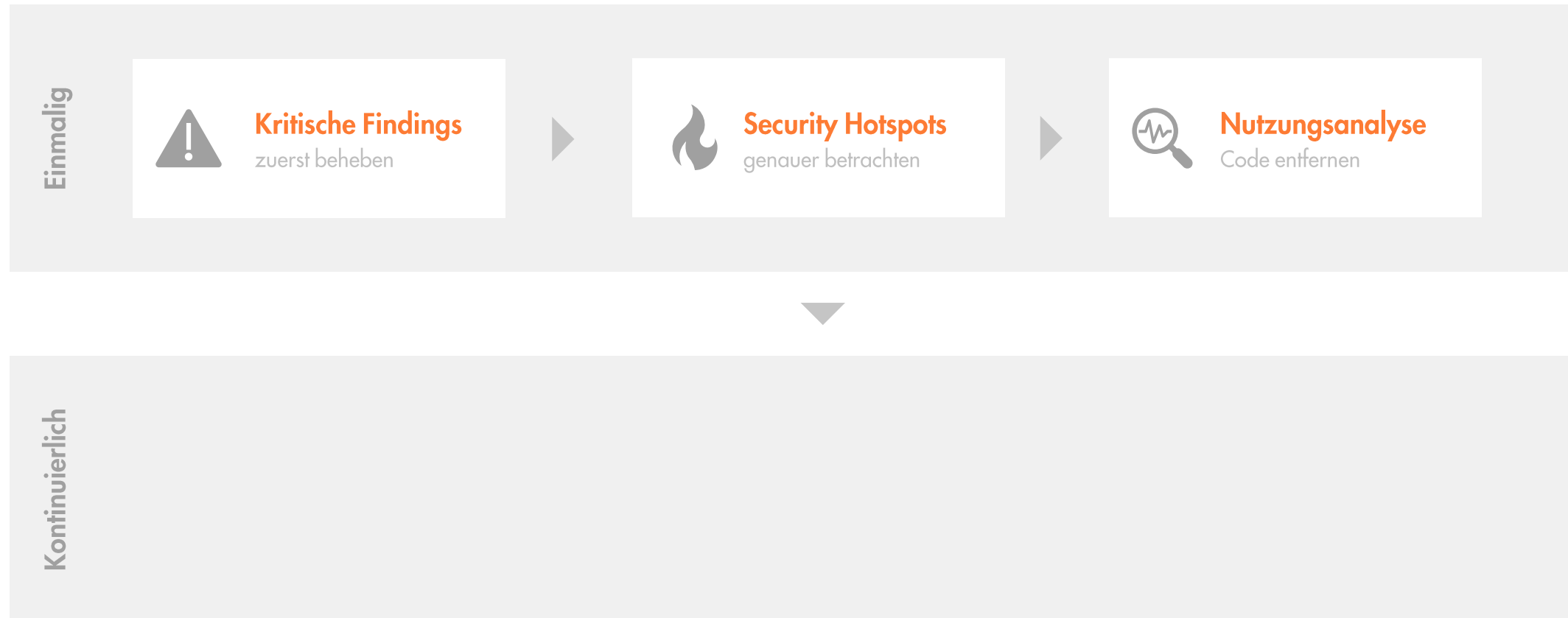
Security Hotspots



☒ Security	13
☒ Taint Propagation	13
☒ Taint Propagation - FILENAME	13
☐ Directory traversal with	35-36
☐ Directory traversal with	37
☐ Directory traversal with	37
☐ Directory traversal with	38
☐ Directory traversal with	38
☐ Directory traversal with	39
☐ Directory traversal with	



Static Application Security Testing für Entwickler:innen





i 37% ausgeführter Code innerhalb eines Jahres



```
1 report [REDACTED]
2 *-----
3 *& PROGRAMM : [REDACTED]
4 *& AUTOR : [REDACTED]
5 *& ERSTELLT AM: 22.04.2002
6 *-----
7 *& ÄNDERUNGEN :
8 *& USER&DATUM(JJJJMMTT) ÄNDERUNGSGRUND
9 *& [REDACTED] 020912 [REDACTED]
10 *& [REDACTED] 020429 [REDACTED]
11 *& [REDACTED] [REDACTED]
12 *& [REDACTED] 030114: [REDACTED]
13 *& [REDACTED] 030122: [REDACTED]
14 *& [REDACTED] [REDACTED]
15 *& [REDACTED] 030122: [REDACTED]
16 *& [REDACTED] 040419: [REDACTED]
17 *-----
18 *& BESCHREIBUNG:
19 *& [REDACTED]
20 *& [REDACTED]
21 *& Dynamische Schnittstelle für Tabelleninhalte [REDACTED]
```

☒ Security 28

☒ Critical and Suspicious Statements 1

- ☒ Call System Function: &1 (CRITICAL_STATEMENTS[0001] (W)) 1

☒ Cross-Client Access 1

- ☒ Cross-client database access 1

☒ Taint Propagation 15

- ☒ Taint Propagation - FILENAME 9
- ☒ Taint Propagation - SQL_INJECTION 6

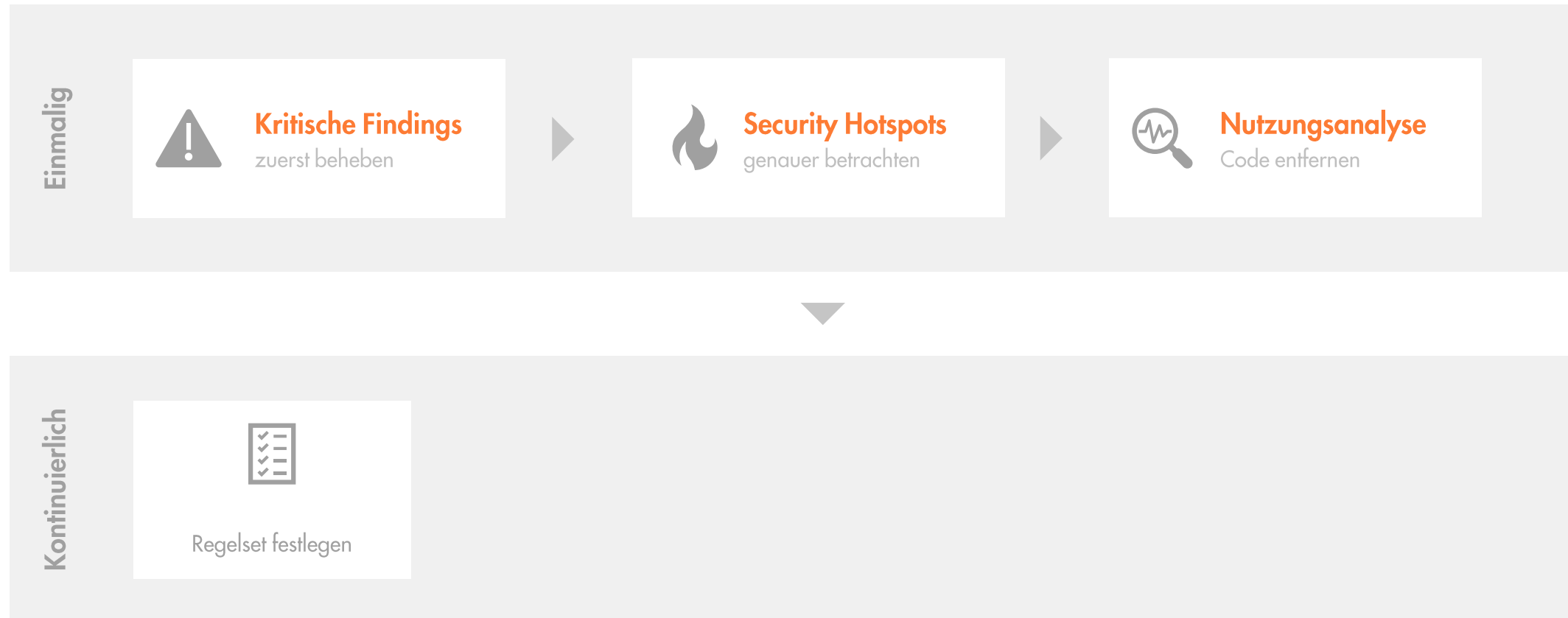
☒ Usage of System Fields 11

- ☒ Control flow depending on system variable (SY-...) 5
- ☒ No write access to system fields 6





Static Application Security Testing für Entwickler:innen





Analysekonfiguration festlegen

Relevante Prüfungen wählen und passend konfigurieren

☒ Security	1704
☒ ▼ Automated Code Manipulation	5
☒ No automated code manipulation	5
☒ ▶ Critical and Suspicious Statements	429
☒ ▼ Cross-Client Access	176
☒ Cross-client database access	176
☒ ▼ Hard-Coded Credentials	51
☒ Hard-coded user name	49
☒ Hard-coded password	2
☒ ▼ Potential Injection	1
☒ Suspicious usage of OPEN DATASET	1

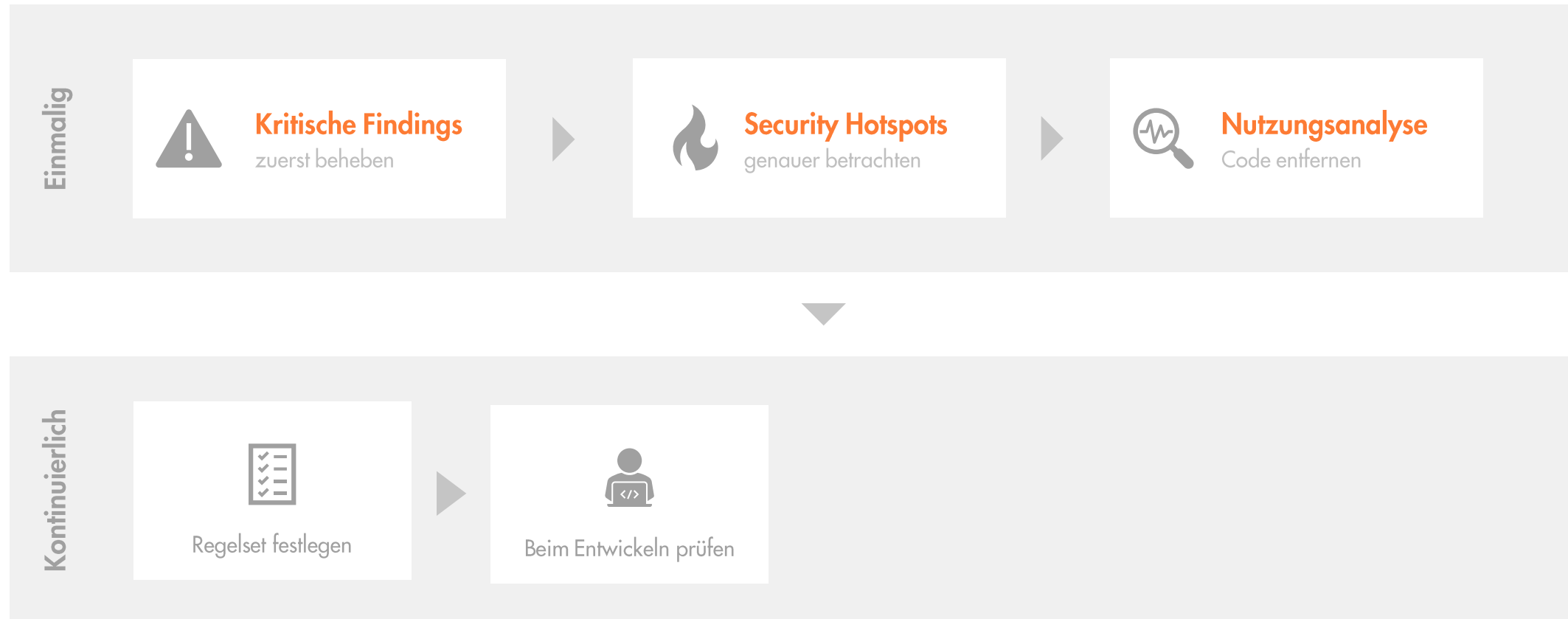
☒ ▼ Taint Propagation	313
☒ Taint Propagation - FILENAME	143
☒ Taint Propagation - GENERIC_MODULE_EXECUTION	1
☒ Taint Propagation - OS_COMMAND	12
☒ Taint Propagation - OTHER	66
☒ Taint Propagation - SQL_INJECTION	91
☒ ▼ Usage of System Fields	729
☒ Control flow depending on system variable (SY-...)	289
☒ No write access to system fields	440

Codediskriminierung durchführen

Anwendungscode
Testcode
Experimenteller Code
Entwicklungswerkzeug-Code



Static Application Security Testing für Entwickler:innen





Beim Entwickeln prüfen

```
123     private static Cipher getCipher() throws StorageException {
124         Cipher cipher = CIPHER_CACHE.get();
125         if (cipher == null) {
126             try {
127                 cipher = Cipher.getInstance(CIPHER_TRANSFORMATION);
128             } catch (NoSuchAlgorithmException | NoSuchPaddingException e) {
129                 throw new StorageException("Could not initialize crypto b
130             }
```

Findings

Findings	Category	Group	Lines
 Use another cipher mode or disable padding	Security	Weak Cryptography	127

 Open Ann-Sophie Kracker requested to merge [redacted] into [master](#)

Overview **3** Commits **3** Pipelines **2** Changes **11**

 Teamscale Findings **1**

 0  0 

 Pipeline **#1116773623** passed     

Pipeline passed for 185d669b on [redacted] 17 minutes ago



Beim Entwickeln prüfen

☒ Flag as False Positive

☐ Flag as Tolerated

 ~~Control flow depends on
system variable SY-UNAME~~

Security > Usage of System Fields >
Control flow depending on system variable (SY-...)

 Flagged as Tolerated

by RB  on Mar 30
2022 12:23:

Globales Sperrkonzept

Remove from Tolerated

Hard-coded password

Hard-coded credentials like passwords or auth tokens are a potential security risk. It can be resolved by storing the critical data separately from the code.

Why is This a Problem?

1. *Security risks*: If passwords are hard-coded into program code, they become not only part of the software's source code, but also they get compiled into the binary. Anyone who has access to the code or binary (developers, hackers, etc.). Attackers can exploit this vulnerability to steal passwords and compromise the system.
2. *Difficulty in changing passwords*: Hard-coded passwords are often difficult to change because they require the code to be recompiled and redeployed. This can be a time-consuming and costly process.
3. *Compliance issues*: Many industry and government regulations require the use of secure passwords and the regular rotation of passwords. Hard coding passwords makes it difficult to comply with these regulations.
4. *Lack of flexibility*: Hard-coded passwords limit the flexibility of the software and increase the difficulty to integrate with other systems or change the authentication logic.

How Can I Resolve This?

It is recommended to store passwords and secrets separately from the program code using secure and encrypted methods such as key management systems, password managers, or environment variables.

Examples

Non-Compliant

```
String authToken = "aH-3jxLE4q2vLtGrmywTUu4ZVsGFjwUR9G";
```

Compliant

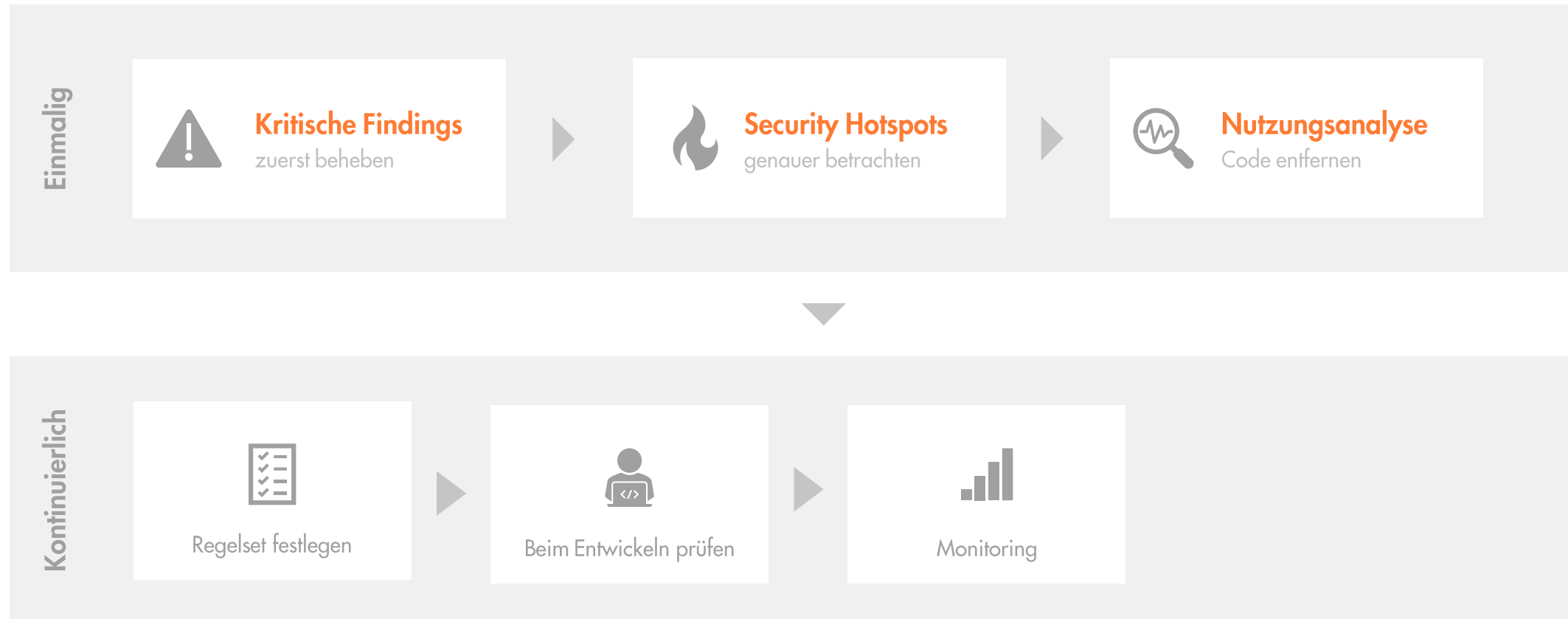
```
String authToken = System.getenv("AUTH_TOKEN");
```

Where Can I Learn More?

- [Security Guideline](#)
- [OWASP Credential Cheat Sheet](#)



Static Application Security Testing für Entwickler:innen





Monitoring aufsetzen



Security



Rote Security-Findings

735



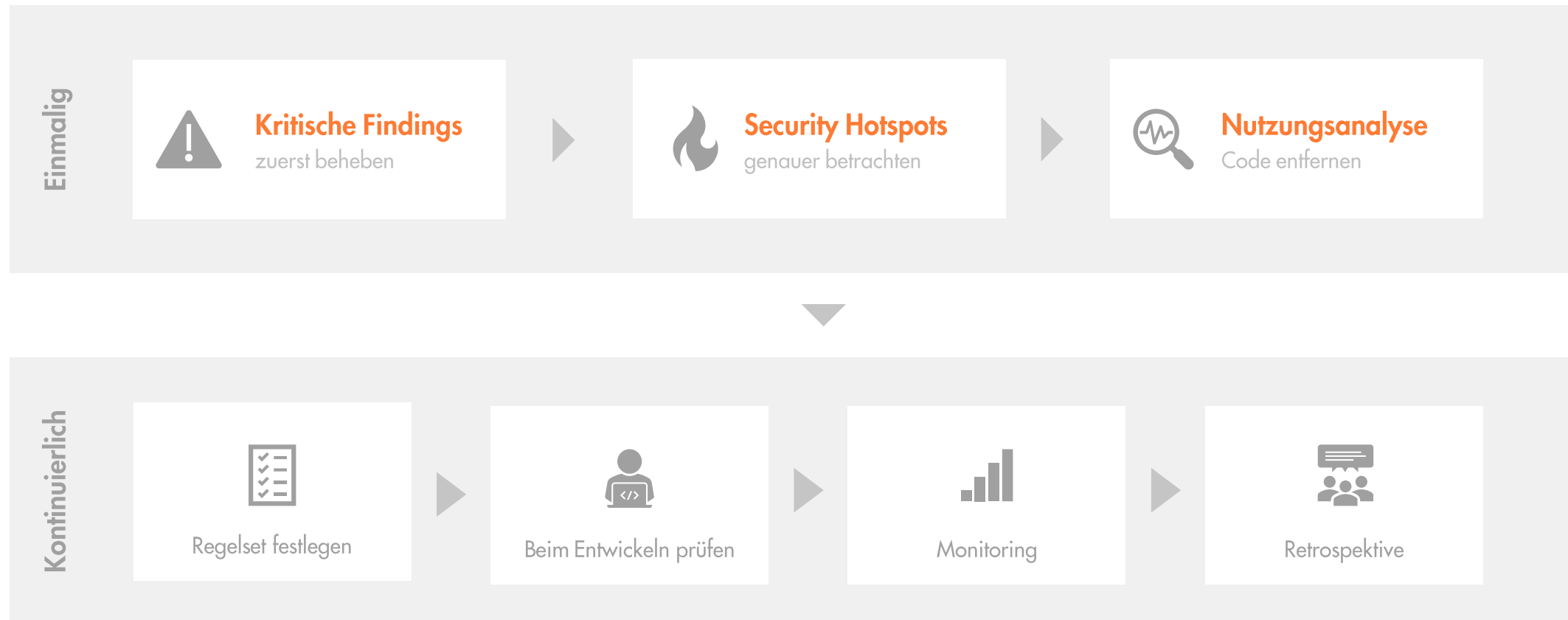
Security-Findings
je 1.000 SLOC

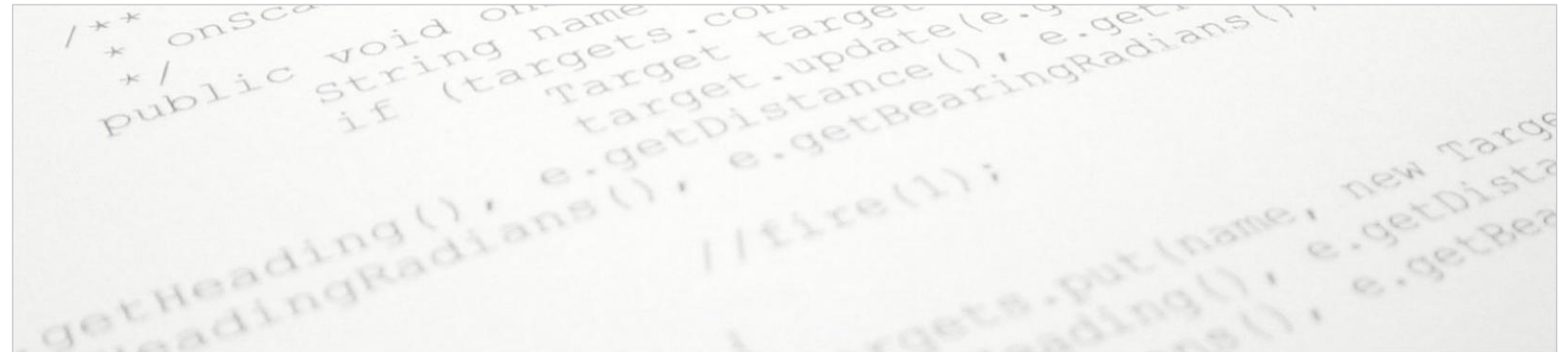
0.8





Static Application Security Testing für Entwickler:innen





Team-Retrospektive

November 2023



Details siehe [CQSE-Workshop »Qualität für alle und alle für Qualität«](#)



Überblick Codequalität

Quality Indicator (QI)	Value	Trend
Redundanz	7.2%	↗
Kritische Security-Findings	22	↗
Codeanomalien	32.9	↗
Prozedurlänge		↗
Schachtelungstiefe		↗



Sehr schöner Trend!



Kurze Schulung zu Verschlüsselungsalgorithmen

When selecting encryption algorithms, tools, or combinations, you should also consider two things:

1. No encryption is unbreakable.
2. The strength of an encryption algorithm is usually measured by the effort required to crack it within a reasonable time frame.

In today's cryptography, the length of the **key** directly affects the security level of cryptographic algorithms

Note that depending on the algorithm, the term **key** refers to a different mathematical property. For example:

- For RSA, the key is the product of two large prime numbers, also called the **modulus**.
- For AES and Elliptic Curve Cryptography (ECC), the key is only a sequence of randomly generated bytes.
 - In some cases, AES keys are derived from a master key or a passphrase using a Key Derivation Function (KDF) like PBKDF2 (Password-Based Key Derivation Function 2)

If an application uses a key that is considered short and **insecure**, the encrypted data is exposed to attacks aimed at getting at the plaintext.

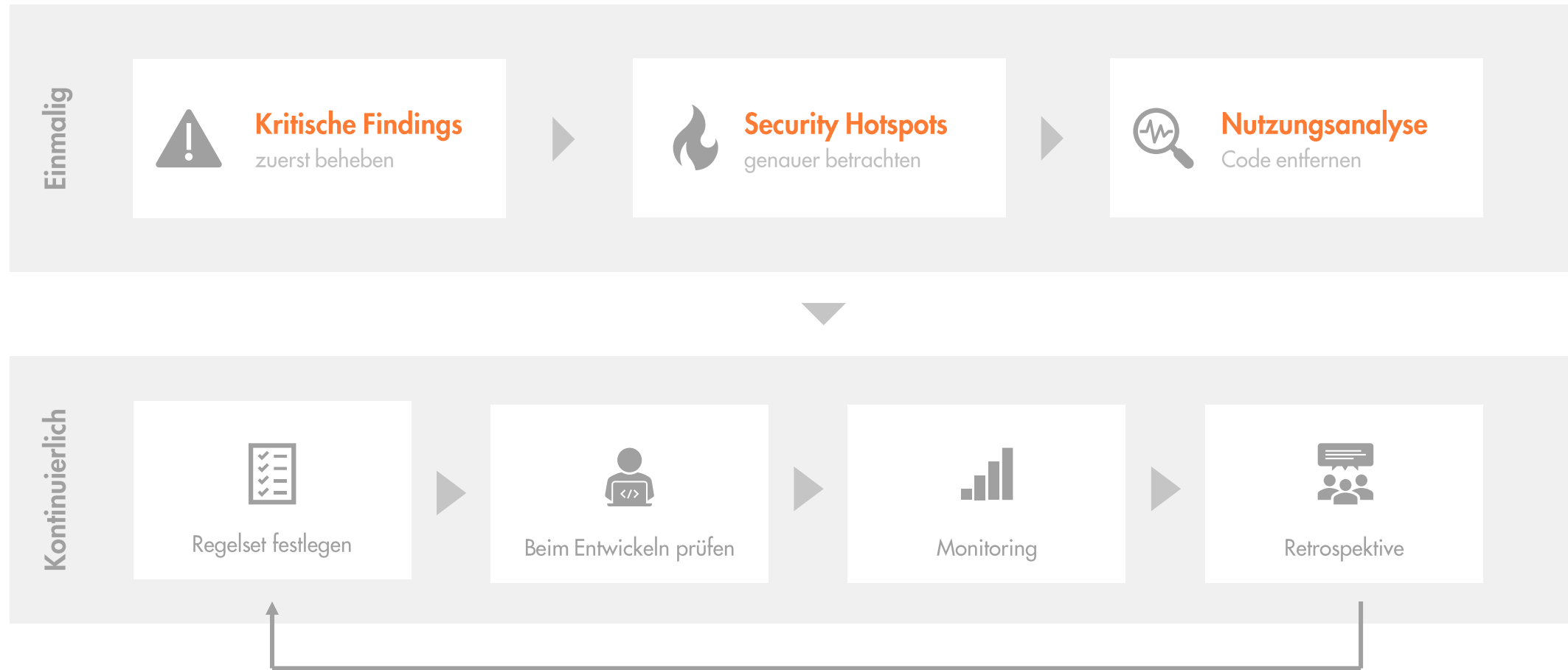
Compliant solution

```
using System;
using System.Security.Cryptography;

public void encrypt()
{
    var RsaCsp = new RSACryptoServiceProvider(2048);
}
```

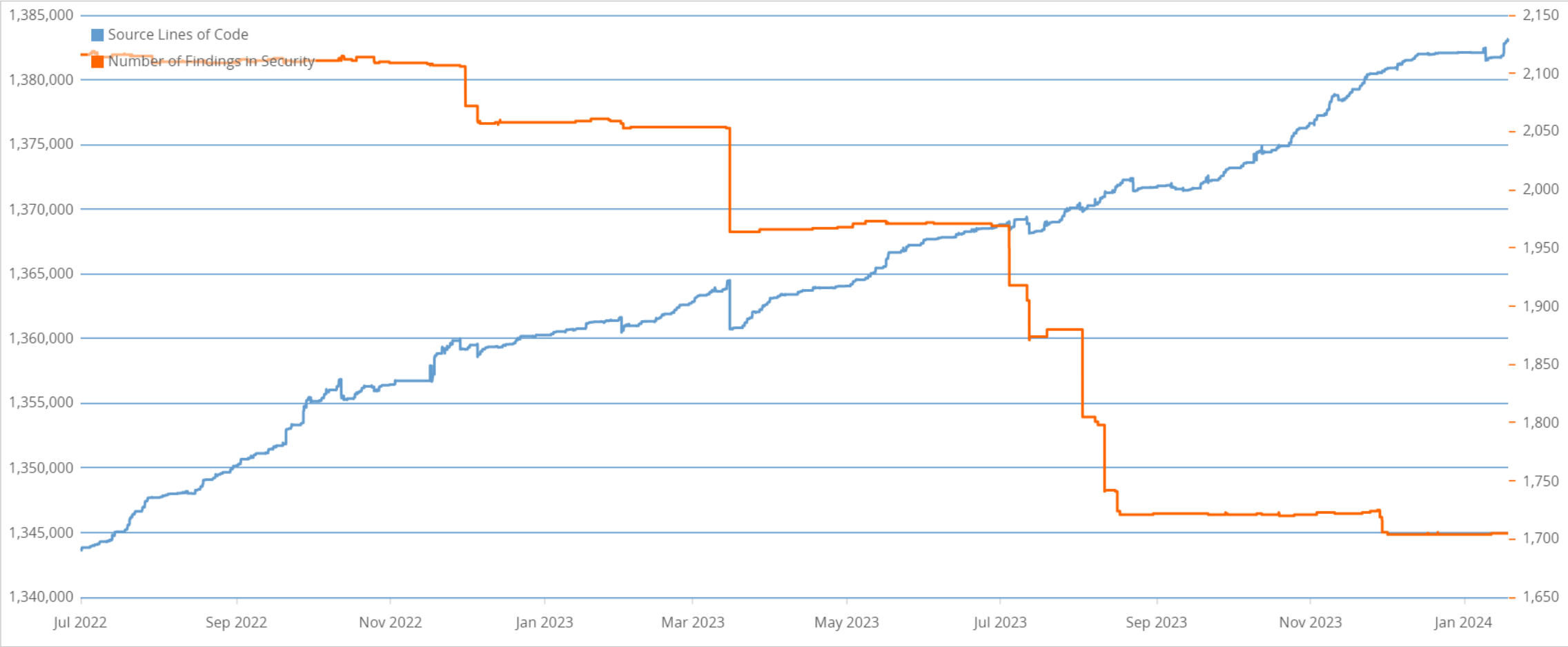


Static Application Security Testing für Entwickler:innen





Security-Trend



Viele SAST-False-Positives ✓
→ Kontextspezifische
SAST-Konfiguration

Wenig Security-Wissen ✓
→ Schulung durch SAST-Findings
und -Retrospektiven

Wenig Zeit
im vollen Entwicklungsalltag ✓
→ Automatische SAST-Analysen,
frühes Feedback

Qualitätssicherung für (viel) KI-Code ✓
→ Automatische SAST-Analysen



Entwickler:in
Emily

→ Alltägliche Vermeidung von Sicherheitslücken
durch Feedback und Wissensaufbau
statt Unsicherheit und Druck

Should We Evaluate LLM Based Security Analysis Approaches on Open Source Systems?

Kohei Dozono*, Jonas Engesser*, Benjamin Hummel[†], Tobias Roehm[†] and Alexander Pretschner*

*Technical University of Munich

Munich, Germany

{kohei.dozono, jonas.engesser, alexander.pretschner}@tum.de

[†]CQSE GmbH

Munich, Germany

{hummel, roehm}@cqse.eu

Abstract—Existing research has demonstrated promising results when applying large language models (LLMs) to detect security vulnerabilities in source code. However, these studies have been exclusively evaluated on benchmarks from Open Source systems, using publicly known vulnerabilities that are likely part of the LLMs' training data. This raises concerns that reported performance metrics may be inflated due to data contamination, providing a misleading view of the models' actual capabilities.

In this paper, we quantify this effect with a case study that evaluates five frontier LLMs on two carefully curated datasets: CWE-Bench-Java (Open Source dataset) and TS-Vuls (a closed source commercial codebase). To provide a second angle, we also split CWE-Bench-Java by the CVE record's date to explore temporal contamination based on LLM's knowledge cutoff dates.

Our results reveal that the average F1 score dropped by approximately 20 percentage points when comparing the Open Source to the closed-source dataset. Additionally, the precision drops from 56% to 34% on average, which is statistically significant ($p < 0.05$) for four of five models. This declining trend is consistent across all tested LLMs and metrics. In contrast, the results for temporal split on Open Source data are inconclusive, suggesting that using a knowledge cutoff may reduce but does not ensure the elimination of contamination effects.

Although our study is based on a single closed source system and thus not generalizable, these findings provide the first empirical evidence that evaluating LLM based vulnerability detection on Open Source benchmarks may lead to overly optimistic results. This motivates the extension of the closed source dataset in future LLM evaluations.

Index Terms—security, large language models, data contamination, static application security testing, security vulnerabilities

I. INTRODUCTION

Advances in language and code processing enabled by progress in large language models (LLMs) have significantly shaped the landscape of software engineering research and enabled new approaches that have not been possible before [1]. Still, while we understand LLMs in terms of data structures, model architectures, training data and algorithms, we currently have no deep understanding of the capabilities and limitations of those models. The sheer size, the stochastic evaluation strategy, and variations in model training lead to a situation where it is impossible to formally deduce if a model can address

a certain task. Instead, evaluation based on benchmarks has become the de-facto standard to determine the fitness of a model for a certain task.

A. Static Security Analysis

Static security analysis is one of the areas where LLMs have been applied recently [2]–[6]. With the constant rise of cyber crime [7] and a growing number of known software vulnerabilities, as indicated by the rise of CVE entries [8], static application security testing (SAST) becomes an increasingly important ingredient for maintaining and improving system security during software development. Additionally, many compliance frameworks, including NIS2 (Network and Information Security Directive 2 [9]) or DORA (Digital Operational Resilience Act [10]), effectively mandate the inclusion of SAST tools in the software development process.

The shift from traditional analysis based on program parsing and analysis, data-flow graph analysis, and similar techniques to LLM-based approaches is driven by the goal to cover a wider range of vulnerability patterns and improve precision and recall [3]. While existing work reports positive results, those approaches have been primarily evaluated on benchmark data distilled from Open Source systems (see Section VII).

B. Threats of Evaluation on Open Source Projects

Due to the availability of large amounts of data, empirical research and evaluations in software engineering are often based on Open Source software systems. This can be a potential threat to validity, as Open Source systems might be developed differently from commercial ones and hence exhibit different properties [11].

However, when evaluating LLM based approaches on Open Source projects, there is another effect that may impact the transferrability of research results, namely the training bias towards Open Source systems. Open Source projects are well-known to be a huge part of the training data used for the initial training of LLMs [12], while closed source projects (due to their nature of being closed) are not used for training of frontier LLMs. Additionally, meta information outside of the core project (such as third-party blog posts, vulnerability

ASE 2025
International Conference on
Automated Software
Engineering

Details siehe [Paper](#)

Open Source Security-
Benchmark

Code + 100 bekannte
Sicherheitsprobleme

Interner Benchmark auf
Basis von Teamscale

Code + 35 bekannte
(behobene!)
Sicherheitsprobleme



Das LLM soll bestimmen,
welche Methoden in den
Dateien ein
Sicherheitsproblem
enthalten



Precision

Wie viele der behaupteten
Sicherheitsprobleme existieren
wirklich?

Recall

Wie viele der enthaltenen
Sicherheitsprobleme wurden
gefunden?



LLM-basierte SAST-Analyse ist verrauscht, da im besten Fall $\sim 1/3$ der gemeldeten Sicherheitslücken False-Positives sind.



LLM-basierte SAST-Analyse ist unvollständig, da im besten Fall $\sim 1/4$ aller vorhandenen Sicherheitslücken NICHT gemeldet werden.



LLM-basierte SAST-Analyse funktioniert für Closed-Source-Code signifikant schlechter als für Open-Source-Code.



Alle Beispiele und Benchmarks sind Open-Source.
»Angriffsmethode« funktioniert nur für Open-Source.

INTERVIEW

C+ Sicherheitsforscher hält Anthropic-Geschichte „eher für Marketing“

von Jan Vollmer 10. April 2026 • 20:00 Uhr • 4 Min.

- Tatsächliche Leistungsfähigkeit und Übertragbarkeit auf Closed-Source-Code unklar.
- SAST bleibt »State of the Art«.

Agenda



Kurzübersicht »Static Application Security Testing (SAST)«



SAST für Entwickler:innen



SAST für Compliance-Verantwortliche

Technologie- & Werkzeugzoo

Viele, neue, schwammige
Compliance-Anforderungen

Umgang mit »immer mehr KI-Code« vs
»immer mehr Compliance-Vorgaben«



Compliance-
Verantwortlicher
Chris

→ Audit-Bereitschaft
statt Unsicherheit und Druck



Static Application Security Testing für Compliance-Verantwortliche

Einmalige Analyse



Kontinuierlicher Prozess



Static Application Security Testing für Compliance-Verantwortliche

Einmalig



Standort-
bestimmung



Kontinuierlich



→ Standortbestimmung ermöglicht initiale Risikoanalyse



Static Application Security Testing für Compliance-Verantwortliche

Einmalig



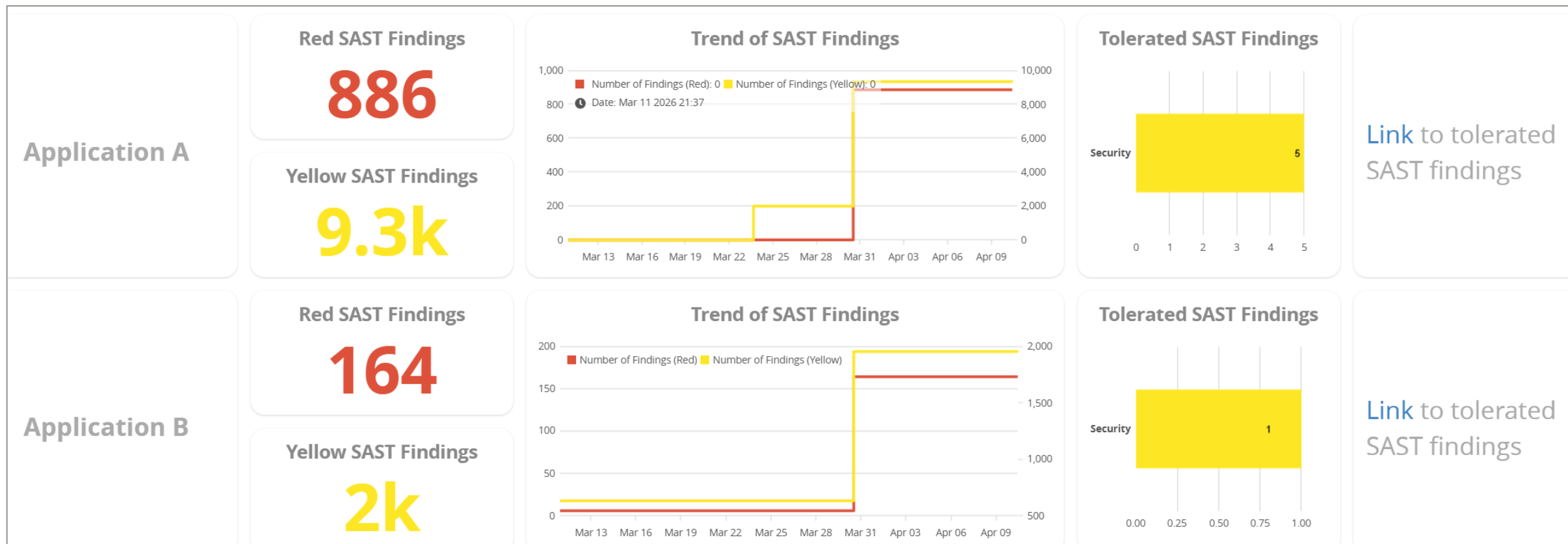
Standort-
bestimmung



Kontinuierlich



Monitoring



→ Monitoring ermöglicht kontinuierliche Risikoanalyse, Steuerung und Nachweis



Static Application Security Testing für Compliance-Verantwortliche

Einmalig



Standort-
bestimmung



Kontinuierlich



Monitoring



Vier-Augen-Tolerierungen



Genehmigungsworkflow für Finding-Tolerierungen



1. Emily schlägt SAST-Finding mit geringem Risiko und Begründung zur Tolerierung vor



2. Chris überprüft SAST-Finding und Begründung und genehmigt Tolerierung



Genehmigungsworkflow für Finding-Tolerierungen (Fortsetzung)

Findings10207

Tolerated12

False Positives1

⌚ Pending Tolerations2

⌚ Pending False Positives0

Choose folder...

Treemap

☐	Message	Location	Toleration Date	Toleration Rationale	Toleration Reporter	Introduction Date
☐	✖ ⌚ 🚫 Add password protection to this database	/HibernateUtil.java:74	Today 14:08	Code ist über Konfiguration abgesichert ...	Emily Demo-Entwicklerin	Mar 31 2025 14:22
☐	✖ ⌚ 🚫 Potential XSS in Servlet	/DatabaseHelper.java:365	Today 14:08		Emily Demo-Entwicklerin	Mar 30 2026 13:24

→ Genehmigungsworkflow ermöglicht pragmatischen, sorgfältigen Umgang mit unkritischen SAST-Findings und dessen Nachweis



Static Application Security Testing für Compliance-Verantwortliche

Einmalig



Standort-
bestimmung



Kontinuierlich



Monitoring



Vier-Augen-Tolerierungen



Guideline-Reporting



Guideline-basiertes Reporting



OWASP TOP10-based Report			CWE TOP25-based Report		
OWASP:A1: 2021-Broken Access Control	11 of 11 checks	254	CWE-20: Improper Input Validation		—
OWASP:A2: 2021-Cryptographic Failures	17 of 17 checks	288	CWE-22: Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')	246	
OWASP:A3: 2021-Injection	22 of 23 checks	692	CWE-77: Improper Neutralization of Special Elements used in a Command ('Command Injection')	0	
OWASP:A4: 2021-Insecure Design	5 of 5 checks	114	CWE-78: Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')	234	
OWASP:A5: 2021-Security Misconfiguration	15 of 15 checks	380	CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	375	
OWASP:A6: 2021-Vulnerable and Outdated Components	1 of 1 checks	0	CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	—	
OWASP:A7: 2021-Identification and Authentication Failures	9 of 9 checks	9 1	CWE-94: Improper Control of Generation of Code ('Code Injection')	0	
OWASP:A8: 2021-Software and Data Integrity Failures	1 of 1 checks	0			
OWASP:A9: 2021-Security Logging and Monitoring Failures					
OWASP:A10: 2021-Server-Side Request Forgery					
CERT-C++ for <u>webkit</u>			CERT-C for <u>webkit</u>		
CWE-119: Improper Restriction of Operations within the Bounds of a Memory Buffer			ARR30-C: Do not form or use out-of-bounds pointers or array subscripts		
CWE-128: Wrap-around Error			ARR32-C: Ensure size arguments for variable length arrays are in a valid range		
CWE-131: Incorrect Calculation of Buffer Size			ARR36-C: Do not subtract or compare two pointers that do not refer to the same array		
CWE-190: Integer Overflow or Wraparound			ARR37-C: Do not add or subtract an integer to a pointer to a non-array object		
CWE-195: Signed to Unsigned Conversion Error			ARR38-C: Guarantee that library functions do not form invalid pointers		
CWE-197: Numeric Truncation Error			ARR39-C: Do not add or subtract a scaled integer to a pointer		
CWE-252: Unchecked Return Value			CON30-C: Clean up thread-specific storage		
CWE-362: Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')			CON31-C: Do not destroy a mutex while it is locked		
CWE-369: Divide By Zero			CON32-C: Prevent data races when accessing bit-fields from multiple threads		
CWE-398: Indicator of Poor Code Quality		0 8	CON33-C: Avoid race conditions when using library functions		
CWE-401: Improper Release of Memory Before Removing Last Reference ('Memory Leak')			CON34-C: Declare objects shared between threads with appropriate storage durations		
CWE-404: Improper Resource Shutdown or Release			CON35-C: Avoid deadlock by locking in a predefined order		
CWE-416: Use After Free			CON36-C: Wrap functions that can spuriously wake up in a loop		

→ Guideline-basiertes Reporting ermöglicht Nachweis von zeitgemäßem Vorgehen



Static Application Security Testing für Compliance-Verantwortliche

Einmalig



Standort-
bestimmung



Kontinuierlich



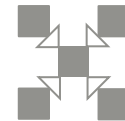
Monitoring



Vier-Augen-Tolerierungen



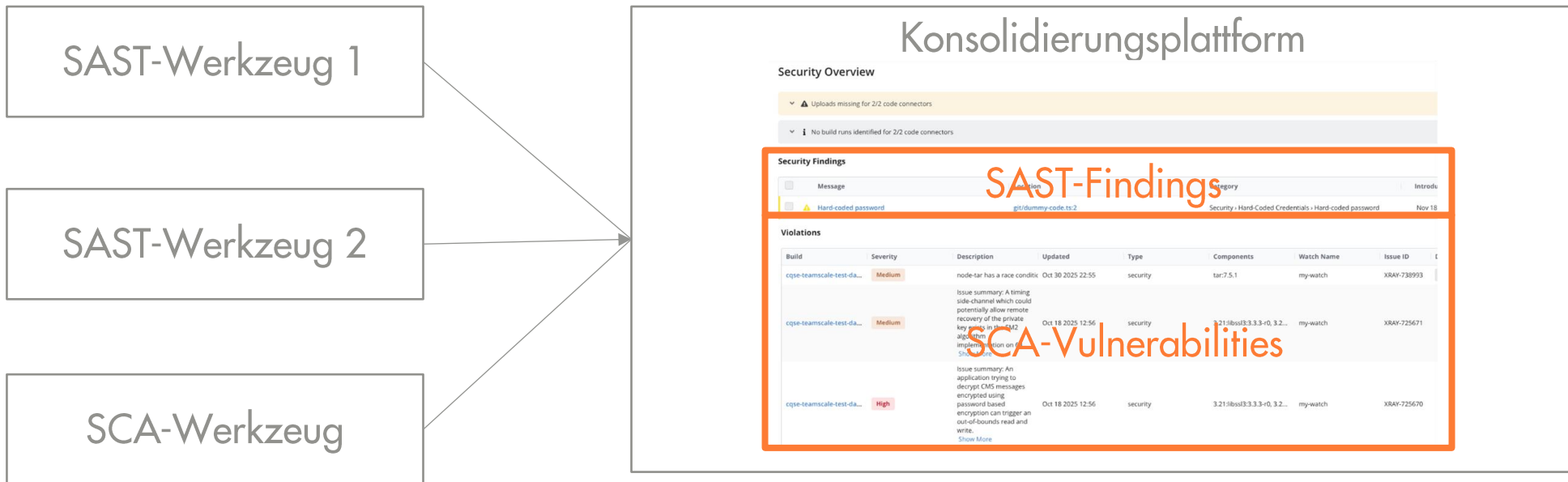
Guideline-Reporting



Finding-Konsolidierung



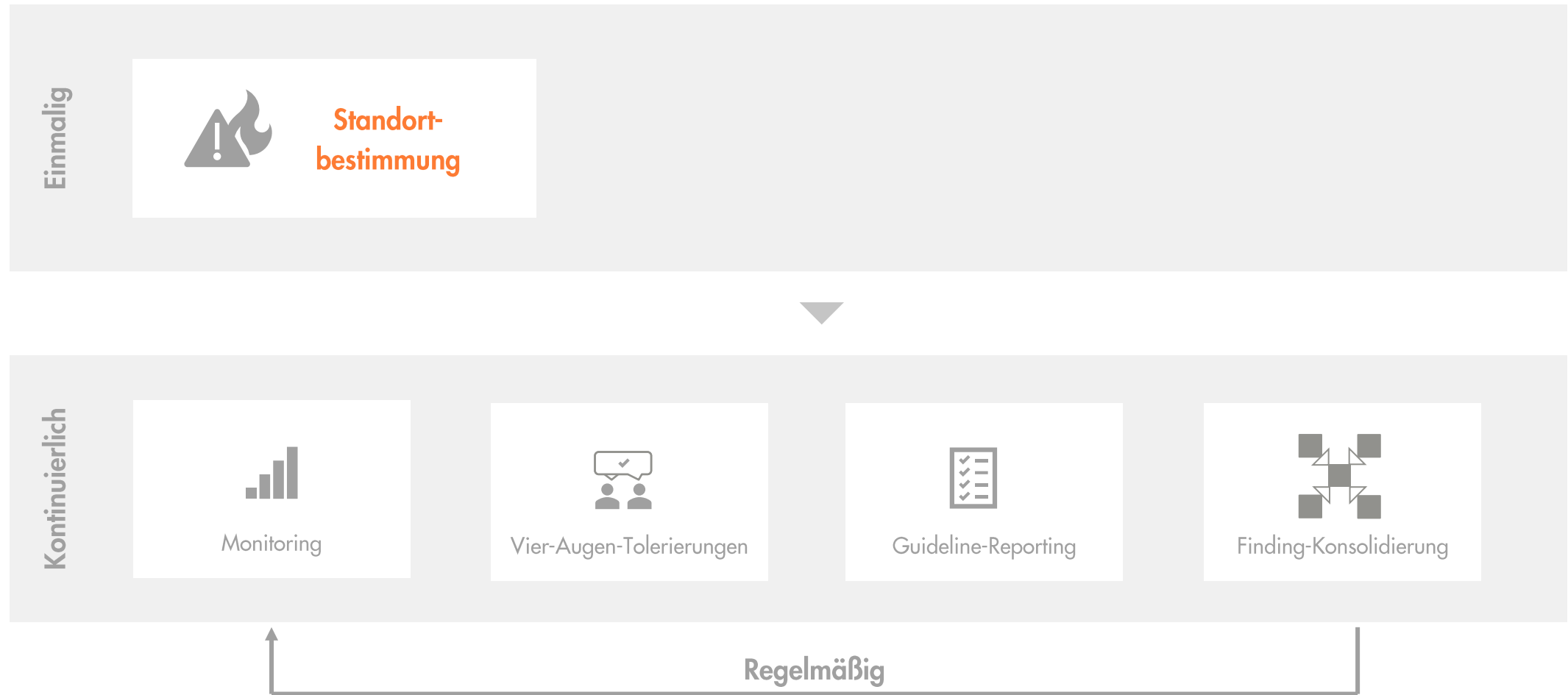
Finding-Konsolidierung (Fortsetzung)



→ Finding-Konsolidierung ermöglicht vollständige Sicht und Reporting



Static Application Security Testing für Compliance-Verantwortliche



Technologie- & Werkzeugzoo ✓
→ Integrierte Lösung, Finding-Konsolidierung

Viele, neue, schwammige
Compliance-Anforderungen ✓
→ Risikoorientierung,
zeitgemäßes Vorgehen,
Nachweismöglichkeiten

Umgang mit »immer mehr KI-Code« vs
»immer mehr Compliance-Vorgaben« ✓
→ Automatische SAST-Analysen



→ Audit-Bereitschaft
durch Risikoorientierung, Integration und Nachweismöglichkeiten
statt Unsicherheit und Druck

Agenda



Kurzübersicht »Static Application Security Testing (SAST)«

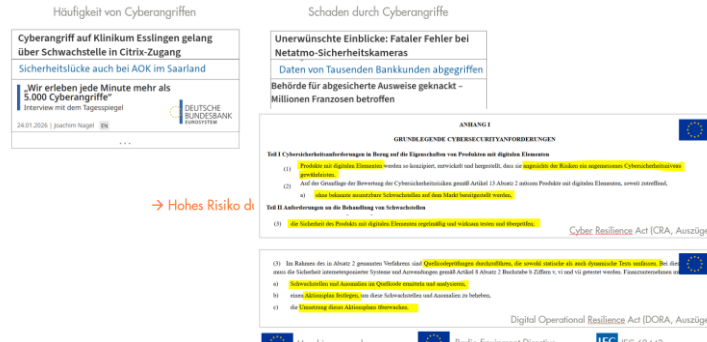


SAST für Entwickler:innen



SAST für Compliance-Verantwortliche

Zusammenfassung



→ Hohes Risiko d

→ Durch Cyberangriffe und Compliance-Vorgaben wird Security wichtiger



→ Alltägliche Vermeidung von Sicherheitslücken durch Feedback und Wissensaufbau statt Unsicherheit und Druck

Anforderungen an Software-Organisation

Sicherheitslücken vermeiden



Entwickler:in IT-Security

Risikoorientiertes, zeitgemäßes Vorgehen nachweisen



Compliance-Verantwortliche IT-Security

→ Adressierung von Security stellt Anforderungen an unsere Organisation



→ Audit-Bereitschaft durch Risikoorientierung, Integration und Nachweismöglichkeiten statt Unsicherheit und Druck

Danke fürs Zuhören

Hier gibt's die Folien und Updates

