

# Software Quality

## Warum brauche ich Code Level Scurity?



Marc André Hahn  
Kay Grosskop

# Agenda

---

- Einleitung Security
- Vorstellungsrunde – Was sind meine Erfahrungen u. Fragen
- Vorstellung Modell Security
- Ablauf einer Analyse
- Einblick Beispielergebnisse
- Diskussion

# Software ist unternehmenskritisch!

- **Relevanz und Menge** von Software in Unternehmen steigt
- Softwareprojekte werden **immer komplexer**
- Auswirkung von **Fehlern** in der Software werden **immer grösser**
- Es fließen immer **mehr Ressourcen** in Software



# Wartbarkeit als Katalysator der Softwareentwicklung

## Softwarequalität ist messbar.

- Der Qualitätsstandard ISO 25010 definiert acht Eigenschaften

## Wartbarkeit ist die Schlüsseleigenschaft

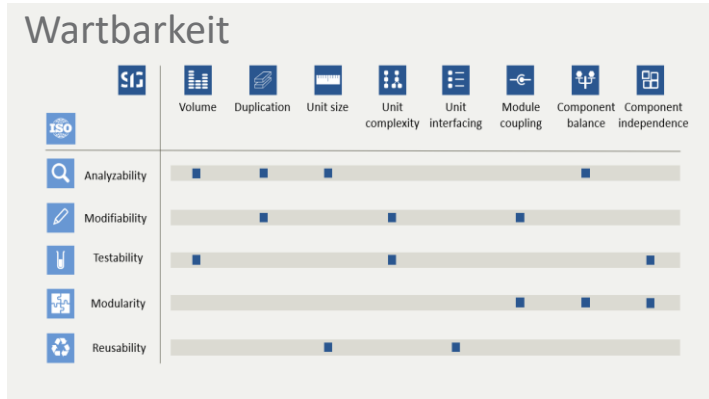
- Einfluss auf **Effektivität** und **Effizienz** von Änderungen am System
- Einfluss auf **Behebung** von **Fehlern**
- Einfluss auf **funktionale Erweiterungen**
- Einfluss auf die **Veränderbarkeit** der anderen **Qualitätseigenschaften**
- Wartbarkeit ist der **Kostentreiber**

## Security

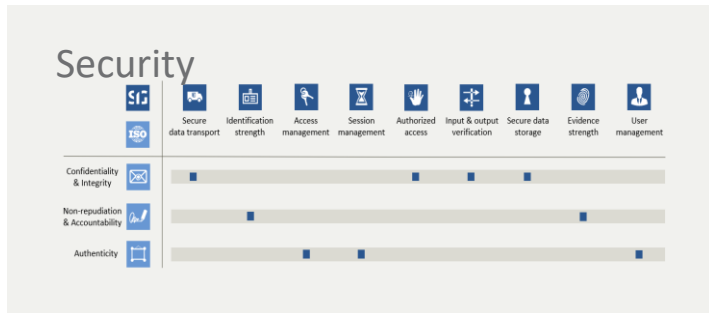
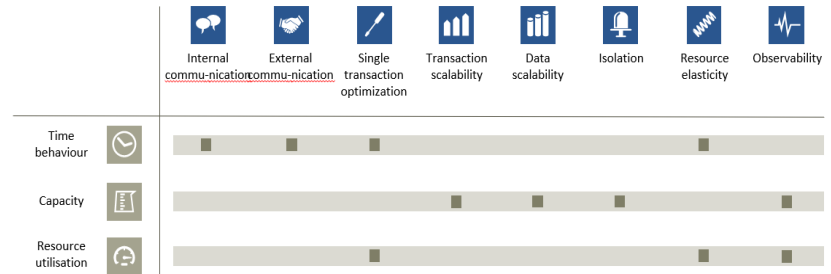
- Wird auf Code Level oft vernachlässigt.



# Modelle für jeden Teilaspekt



## Performance

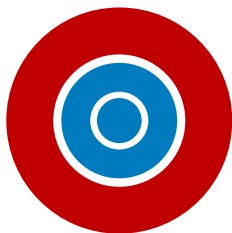


...Modelle für alle Teilaspekte

# Ihre Herausforderung

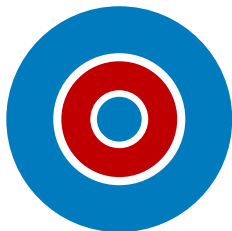
- Die **Sicherheit** der Unternehmung und ihrer Kunden **soll bestmöglich sichergestellt sein.**
- Die Risiko-Manager und Sicherheits-Verantwortlichen wenden viel Zeit auf, um **geeignete Instrumente** und Hilfsmittel zur **Einhaltung des Schutzes** bereitzustellen und brauchen ein **verlässliches Bild über** Transparenz und **Zuverlässigkeit des Unternehmens- und Informationsschutzes.**
- Das **Vertrauen der Kunden und Anwender** in die Sicherheit und **und die Zufriedenheit** mit der Sicherheit ist eine wichtige **Zielgrösse** für die Angebotsqualität und den **Absatz der Services am Markt**
- Der **Nachweis des angemessenen Schutzes** aller Services und Daten muss **objektiv erbracht** sein
- Um **Optimierungspotential** zu **erkennen** und **Massnahmen zu formulieren** muss ein Status Quo erhoben und mit den Anforderungen verglichen werden.





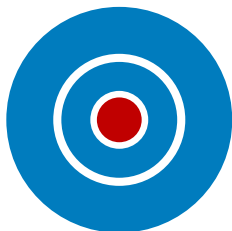
## Information Security Management

- ISO 27001



## IT Infrastructure Security

- Infrastructure configuration audits
- Penetration Testing



## Application Software Security

- Source Code Analyse
- Penetration Testing

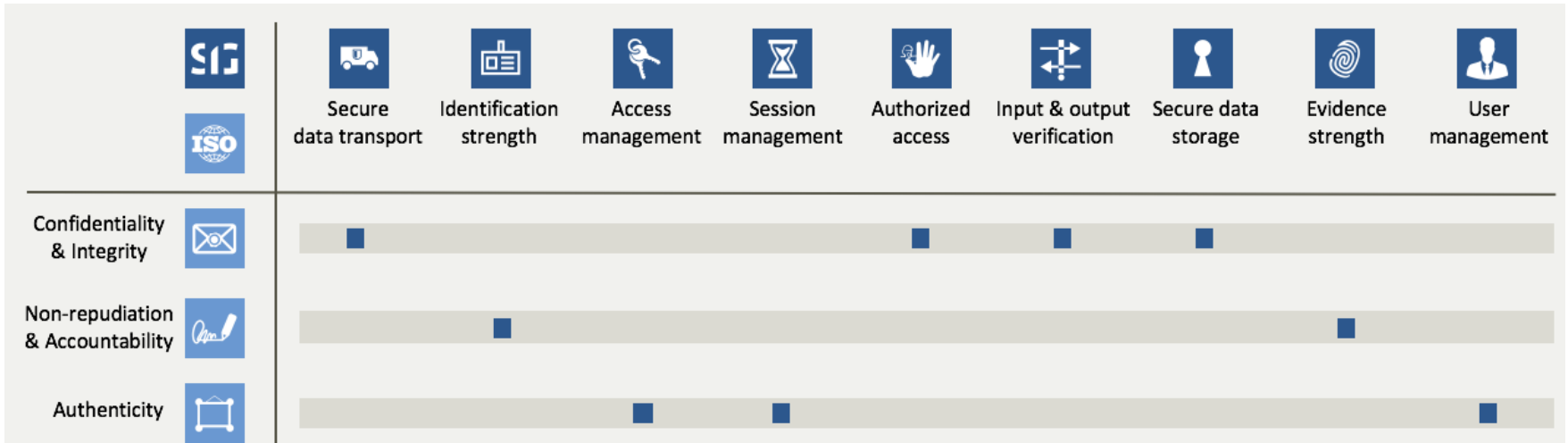
# Security Evaluationsmodell

---



# ISO 25010 Security

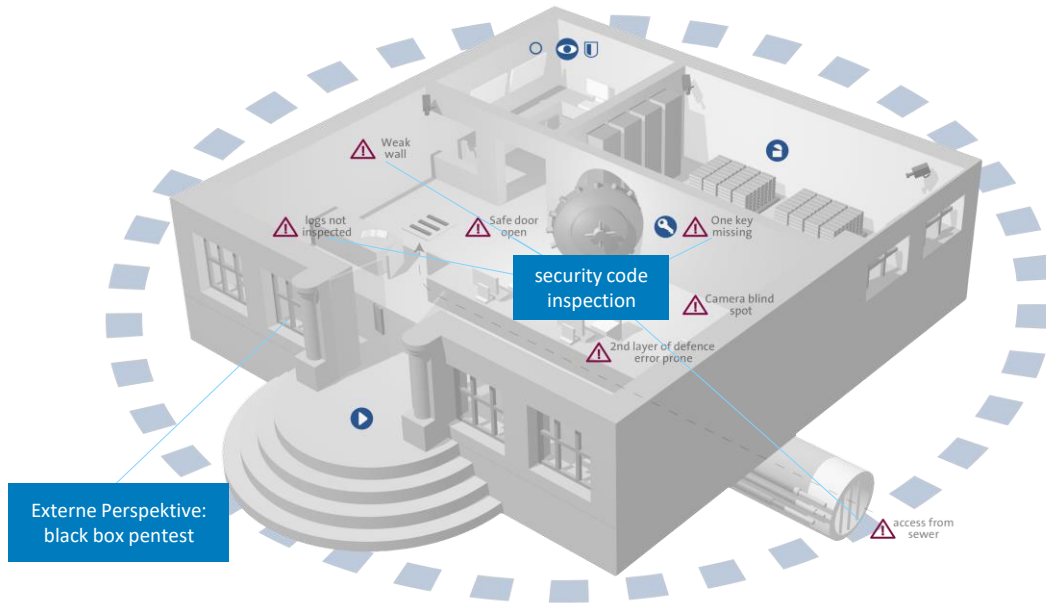
Das SIG Auswertungsmodell bestimmt ob für 9 Systemeigenschaften gängige best-practices angewand werden:



→ Ziel ist es festzustellen ob die Anwendung sicher entworfen ist und somit, die **Wahrscheinlichkeit auf zukünftige Schwachstellen** verringert wird.

# Security code inspection

How security is built in by design



- > Eine *software security inspection* beurteilt ob *security best practices*, umgesetzt sind und die Anwendungssicherheit auf *security by design* beruht.
- > Fehlen von *best practices* in der Entwicklung bedeuten nicht immer eine Schwachstelle, aber es erhöht die *Warscheinlichkeit* das das System anfällig wird.

Eine Codeanalyse kann Ursachen von möglichen Schwachstellen besser sehen und ist damit nachhaltiger

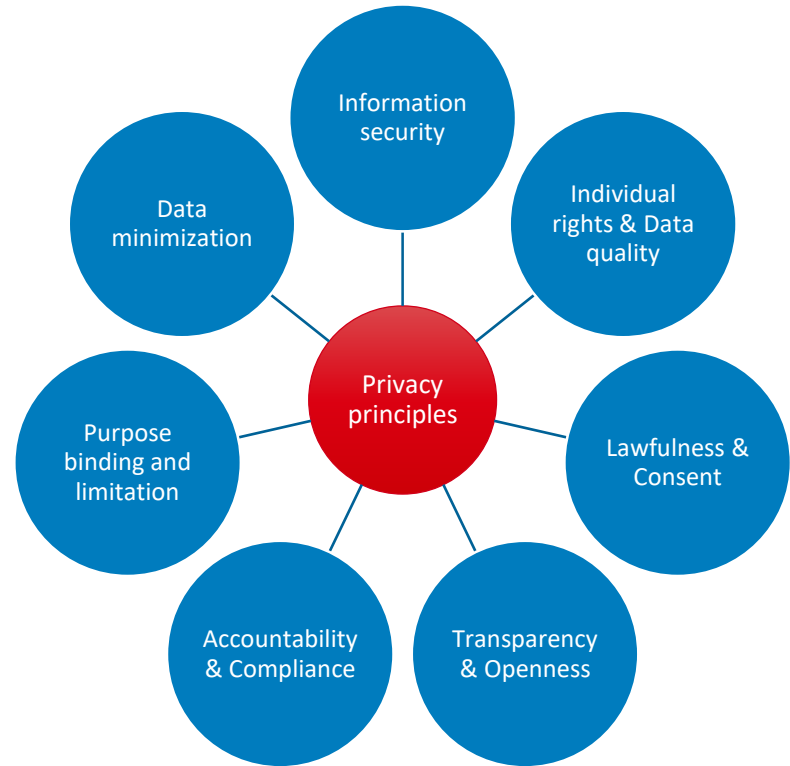


# Privacy Evaluationsmodell

---

# Privacy principles are derived from a set of standards and guidelines

- ISO/IEC 29100 Privacy framework
- GDPR/General Data Protection Regulation 2017
- OECD Privacy Principles
- Data protection directive (DPD) 1995 and 2012
- ENISA Data protection by design and by default
- Privacy by design 7 foundational principles (from Ann Cavoukian)



# Privacy principles

<b>Data minimization</b>	Only necessary data must be collected, processed and stored
<b>Purpose binding and limitation</b>	PII obtained must not be processed for other purposes that are not compatible with the original purpose
<b>Individual rights and Data quality</b>	PII is up-to-date and data subjects can add, change, request or delete associated PII
<b>Lawfulness and Consent</b>	Data collection is required by law, legal obligation, or the data subject gives explicit consent with respect to the processing of their PII
<b>Transparency and Openness</b>	Data subjects are informed about data collection, processing and storage of their PII
<b>Accountability and Compliance</b>	Ensure and demonstrate compliance with privacy and data protection principles (or legal requirements)
<b>Information Security</b>	Enforce confidentiality, integrity and availability of personal identifiable information (PII)

# SIG Auswertungsmodell Privacy

System properties SIG Privacy Characteristics	Collection & dissemination	De-identification	Profiling & Analytics	Access control	Privacy Settings	Logging	Retention & Erasure
Data Minimization	■	■					■
Purpose binding and limitation			■	■		■	
Individual rights and Data quality			■	■	■		■
Lawfulness and Consent	■		■		■		
Transparency and Openness	■		■		■	■	■
Accountability and compliance				■		■	■
Information Security	■	■					

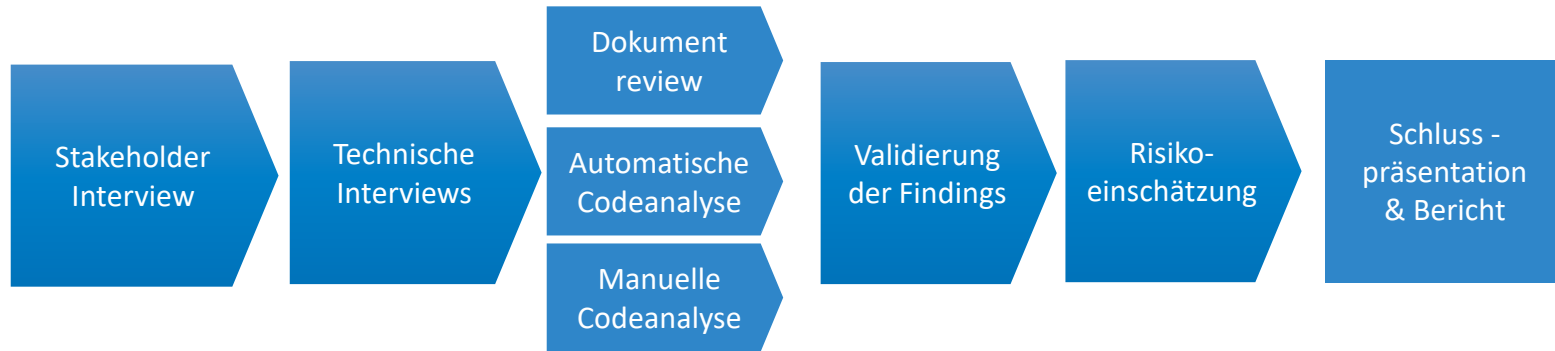
PAUSE

# What a typical pentest covers

Security question	Covered in a pentest?
Wird vertrauliche Information in Logfiles geschrieben?	no
Sind die Passwörter in der Datenbank verschlüsselt?	no
Werden bei Fehlermeldungen ungewollt technische Details angezeigt?	yes
Welche verwendeten Programmbibliotheken haben bekannte Sicherheitslücken?	no
Hat die Anwendung ein Session timeout?	yes
Werden Sicherheitsschlüssel im Coderepository aufbewahrt?	no
Ist die Anwendung strukturell gegen SQL injection gesichert?	no
Ist die Anwendung anfällig für cross-site scripting?	yes
Wird in der Testumgebung mit vertraulichen Produktionsdaten gearbeitet?	no



# Ablauf eines Security Assessments



## Beispiele aus der Praxis

---

# Einige Beispiele

## Payment processing für eine Bank (Produktsoftware, 100 Personenjahre gross)

- Schlechte Wartbarkeit
- Veraltete Libraries mit Schwachstellen
- Unsicherer Hash Algorithmus, fehlende Security Headers, unsichere Datenbankanfragen

## Sicherheitsrelevante Anwendung in öffentlicher Verwaltung (Produktsoftware, 18 PJ)

- Veraltete Libraries mit Schwachstellen
- Ungenutzter & duplizierter Code in Sicherheitsfunktionen

## Infrastruktursoftware (Individualentwicklung, 20 Jahre alt, 13 PJ)

- Veraltete Technologie aber angebunden ans Internet.
- Verwendung unverschlüsselter Kommunikationsprotokolle.
- Hardcoded Admin Credentials im Code.

# Fatal code smells – Toyota acceleration lawsuit



Source: <http://www.autoblog.com/2013/01/29/2013-toyota-camry-hybrid-review/>

## Background details

- Fatal crash of a Toyota Camry after sudden acceleration
- Toyota settled in 2013 for \$3 million
- Toyota loosely followed MISRA-C coding guidelines
- Toyota ETCS code:
  - 67 function with McCabe > 50
  - Throttle angle function: McCabe = 146

18 Q Then you mentioned the code quality metrics. what do  
19 you mean about that?  
20 A So the code complexity and the McCabe Code Complexity  
21 is one of the measures of that. And the code complexity for  
22 Toyota's code is very high. There are a large number of  
23 functions that are overly complex. By the standard industry  
24 metrics some of them are untestable, meaning that it is so  
25 complicated a recipe that there is no way to develop a

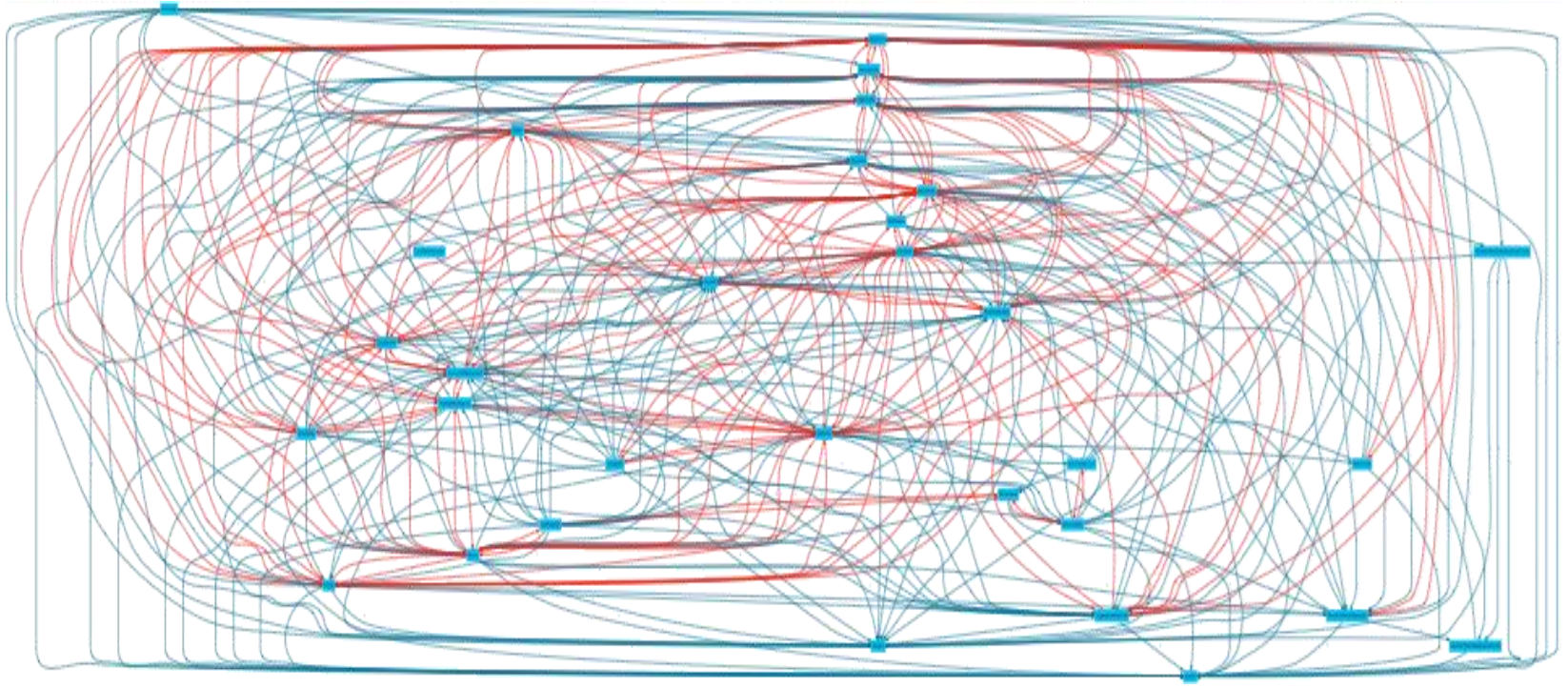
\*\*\*\*\* THIS TRANSCRIPT HAS NOT BEEN PROOFREAD \*\*\*\*\*

6 Q. Based on a lack of systematic processes you  
7 described, have you reached an opinion on whether this  
8 software is defective?  
9 A. Yes.  
10 Q. What's your opinion?  
11 A. In my opinion is that this code is an unreasonable  
12 quality and defective.

Read more: <http://www.latimes.com/business/autos/la-fi-hy-toyota-damages-20131026-story.html>

# Code Komplexität als Problem für die Sicherheit

- How did this happen? (one line of code at a time)





# Example Findings - Known vulnerabilities of 3<sup>rd</sup> party dependencies in practice

After discovering an outdated library is in use, finding vulnerabilities is easy

Fehlermeldung im System:

```
org.glassfish.jersey.servlet.WebComponent.serviceImpl(WebComponent)
org.glassfish.jersey.servlet.WebComponent.service(WebComponent)
org.glassfish.jersey.servlet.ServletContainer.service(ServletContainer)
org.glassfish.jersey.servlet.ServletContainer.service(ServletContainer)
org.apache.tomcat.websocket.server.WsFilter.doFilter(WsFilter)

Note The full stack trace of the root cause is available in the server logs.
```

**Apache Tomcat/8.5.24**

Tomcat 8.5.24 was released in November 2017, and outdated since January 2018



## Schwachstellendatenbank

**Apache » Tomcat » 8.5.24 : Security Vulnerabilities**

Cpe Name: [cpe:/a:apache:tomcat:8.5.24](#)  
CVSS Scores Greater Than: 0 1 2 3 4 5 6 7 8 9  
Sort Results By: [CVE Number Descending](#) [CVE Number Ascending](#) [CVSS Score Descending](#) [Number Of Exploits Descending](#)  
[Copy Results](#) [Download Results](#)

#	CVE ID	CWE ID	# of Exploits	Vulnerability Type(s)	Publish Date	Update Date	Score
1	<a href="#">CVE-2018-11784</a>	<a href="#">601</a>			2018-10-04	2019-01-23	4.3
When the default servlet in Apache Tomcat versions 9.0.0.M1 to 9.0.11, 8.5.0 to 8.5.33 and 7.0.23 to 7.0.90 returned a redirect to a directory the redirect to be generated to any URI of the attackers choice.							
2	<a href="#">CVE-2018-8037</a>	<a href="#">362</a>			2018-08-02	2018-10-16	4.3
If an async request was completed by the application at the same time as the container triggered the async timeout, a race condition existed present in the NIO and NIO2 connectors that did not correctly track the closure of the connection when an async request was completed seeing a response intended for another user. Versions Affected: Apache Tomcat 9.0.0.M9 to 9.0.9 and 8.5.5 to 8.5.31.							
3	<a href="#">CVE-2018-8034</a>	<a href="#">295</a>			2018-08-01	2019-01-23	5.0
The host name verification when using TLS with the WebSocket client was missing. It is now enabled by default. Versions Affected: Apache Tomcat 9.0.0.M1 to 9.0.11, 8.5.0 to 8.5.33 and 7.0.23 to 7.0.90.							
4	<a href="#">CVE-2018-8014</a>	<a href="#">254</a>			2018-05-16	2018-12-05	7.5
The defaults settings for the CORS filter provided in Apache Tomcat 9.0.0.M1 to 9.0.8, 8.5.0 to 8.5.31, 8.0.0.RC1 to 8.0.52, 7.0.41 to 7.0.90 will have configured it appropriately for their environment rather than using it in the default configuration. Therefore, it is expected that the CORS filter will be disabled.							
5	<a href="#">CVE-2018-1336</a>	<a href="#">400</a>		DoS Overflow	2018-08-02	2018-12-05	5.0
An improper handling of overflow in the UTF-8 decoder with supplementary characters can lead to an infinite loop in the decoder causing a denial of service. Versions Affected: Apache Tomcat 9.0.0.M1 to 9.0.11, 8.5.0 to 8.5.33 and 7.0.23 to 7.0.90.							
6	<a href="#">CVE-2018-1305</a>	<a href="#">284</a>			2018-02-23	2018-10-18	4.0
Security constraints defined by annotations of Servlets in Apache Tomcat 9.0.0.M1 to 9.0.4, 8.5.0 to 8.5.27, 8.0.0.RC1 to 8.0.49 and 7.0.23 to 7.0.90 may not be applied to the URL pattern and any URLs below that point, it was possible - depending on the order Servlets were loaded - for an unauthorised user to access them.							
7	<a href="#">CVE-2018-1304</a>	<a href="#">254</a>			2018-02-28	2018-10-18	4.3
The URL pattern of "" (the empty string) which exactly maps to the context root was not correctly handled in Apache Tomcat 9.0.0.M1 to 9.0.11, 8.5.0 to 8.5.33 and 7.0.23 to 7.0.90. This caused the constraint to be ignored. It was, therefore, possible for unauthorised users to gain access to web application resources.							

## Example Findings - Input & output verification

Tapestry encodes output by default, but this protection is sometimes disabled

+

> Tapestry encodes output for HTML by default, this prevents Cross-site scripting

-

> This can be circumvented by using `<t:outputRaw>`, or `<t:output filter="false">`

> `<t:outputRaw>` is used **71 times**. In some instances an exception is documented, but not always

Good

```
<!--The outputraw variable is not used in web, so it should not present any danger-->
<t:OutputRaw value="errorMessage" />
```

Source: /XXXX/src/main/webapp/layouts/xxx/xxx/xxxx.tml

```
<div class="message-container">
  <div class="{css}">
    <button data-dismiss="alert" class="close form-submit" type="button">x</button>
    <t:if test="showHeader">
      <h4>{header}</h4>
    </t:if>
    <ul>
      <li t:type="loop" source="message.records" value="currentRecord"><t:outputraw value="currentRecord"/></li>
    </ul>
  </div>
</div>
```

Are you sure message records can never contain user input?  
(This is a generic component)

Error

You must provide a value for Username.

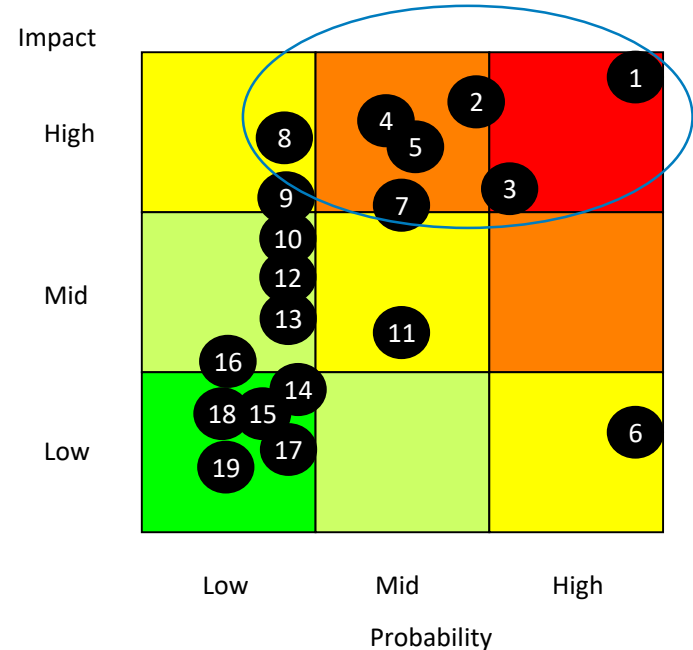
You must provide a value for Password.

Source: XXXX/src/main/resources/com/xxx/web/components/xxx/MessageTile.tml








# Beispiel: Overview of identified security issues

- 19 issues were identified, however most of them are low risk
- In total, 19 security issues were identified in the source code
- Many of these issues have a low risk, mainly due to their low probability (i.e.: It requires expert knowledge under certain circumstances to exploit the issues)
- Only issues #1 and #6\* require significant development effort\*
- **We recommend to address at least the first 8 on the short term**
  - These amount to 3-4 months of dev effort (1-2 months without #6)



\*estimated as more than 1-2 weeks

# Most important security issues

	 Risk	 Finding	 Risk	 Recommendation	 Dev. effort
#1	10	17 dependencies have known vulnerabilities	Various easy to exploit vulnerabilities	Update these dependencies	Months
#2	7,5	SQL queries without SQL injection prevention	Access to database (read, alter, delete)	Apply prepared statements	Weeks
#3	7	Frontend output encoding sometimes disabled	Code executed on user browser	Re-enable output encoding unless explicitly checked	Days
#4	6,5	API endpoints call Auth. service separately	Accidentally unprotected API's	Centralize authentication	Weeks
#5	6,5	No protection mechanism against CSRF implemented	Susceptible to phishing attacks	Add CSRF tokens	Days
#6	6	Core security functions have low maintainability	Implementation mistakes in security features	Improve maintainability	Months
#7	6	No account cleanup procedure/function	Ex-users can still access the system	Implement cleanup procedure	Weeks
#8	5,5	Passwords hashed using outdated algorithm	Susceptible to "brute-force" attacks (try all passwords)	Change algorithm	Weeks

Risk is a number between 1-10 which is a combination of **probability** and **impact** of the security issue

# Zusammenfassung

---

# Warum brauche ich **Code level security**?

- Software wird **wichtiger**. Beinahe alle Prozesse & Produkte haben eine kritische Softwarekomponente
- Software wird **grösser & komplexer**
- Durch **Vernetzung** sind Anwendungen mehr exponiert und sind schwieriger zu schützen.
- **Schnellere Verbreitung von Angriffstechniken & Tools.**
- Der **Reifegrad der Softwareindustrie** ist noch niedrig
- **Alle Anwendungen**, nicht nur öffentliche Websites sind betroffen.
- Viele Schwachstellen werden durch **Nachlässigkeiten der Entwickler** verursacht.
- Pentests finden vor allem aktuelle Schwachstellen und nicht **potentielle Schwachstellen**.
- Penetrationstests sind nützlich aber **spät im Entwicklungszyklus** und unvollständig. (“too little too late”)

# Was ist dabei wichtig?

## Security bei Design

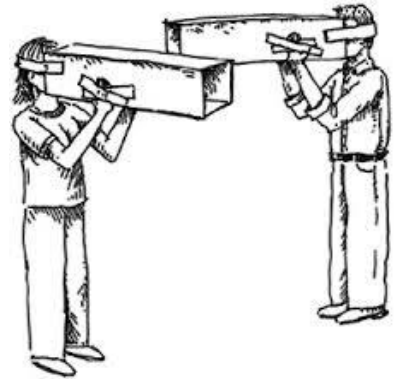
- Verankerung im **Entwicklungsprozess: Schulung** der Entwickler.
- In allen Bereichen: **Architektur > Development > Testing**

## Externer Audit

- Der eigenen Abteilung fehlt es oft an **Know-how und Ressourcen**
- **Bias** beim Selbstcheck. 'Blindheit' für eigenen Code ist normal.

## Gibt es dafür Tools?

- Ja, aber sie generieren viele 'false positives'. Ein Experte muss die **Ergebnisse interpretieren**.
- Typischerweise verweisen die Reports nach kurzer Zeit falls nicht Teil eines *secure development process*.



# Wann und wie oft?

---

- **Entwicklungsprojekte:** mindestens einmal jährlich
- **Wartungsprojekte:** Alle 1-2 Jahre
  
- Gute Zeitpunkte:
  - Major releases
  - Onboarding einer neuen Anwendung in die Wartung
  - Publikation des Sourcecodes

Fragen?

# Kunden & Kontaktangaben

## Kontakt



Marc André Hahn  
sieber&partners  
Schwanengasse 1, Bern  
Usterstrasse 19, Zürich  
+41 78 686 85 16  
marc.hahn@sieberpartners.com



Kay Grosskop  
sieber&partners, SIG  
Schwanengasse 1, Bern  
Usterstrasse 19, Zürich  
+41 31 566 93 00  
kay.grosskop@sieberpartners.com

## Kunden aus der Schweiz (Beispiele)



## Kunden weltweit (Beispiele)







**We lead from the emerging future!**

# Comparison of test approaches according to OWASP

## Penetration Testing

### Advantages:

- Can be fast (and therefore cheap)
- Requires a relatively lower skill-set than source code review
- Tests the code that is actually being exposed

### Disadvantages:

- Too late in the SDLC
- Front impact testing only.

## Source code review

### Advantages:

- Completeness and effectiveness
- Accuracy
- Fast (for competent reviewers)

### Disadvantages:

- Requires highly skilled security developers
- Can miss issues in compiled libraries
- Cannot detect run-time errors easily
- The source code actually deployed might differ from the one being analyzed

## **CNO Academy - Warum brauche ich Code Level Security?**

Die meisten IT-Leiter und Anwendungsverantwortliche haben erkannt, dass ein ISO27001 Zertifikat der Organisation und die Ausführung eines Pentests vor dem Release nicht genügen, um die Sicherheit ihrer Anwendungen zu garantieren. Die Zeit, in der Anwendungsverantwortliche die Sicherheit von Softwareanwendungen ignorieren konnten, oder die Verantwortung an Infrastrukturverantwortliche in der Organisation abschieben konnten, ist vorbei. Heute sollte sich jeder Eigentümer bewusst sein, dass Sicherheit (und Privacy) ein integraler Qualitätsaspekt von Software ist und proaktiv gesichert werden muss. Dies ist zum Beispiel relevant bei:

- Anwendungen in der Wartungsphase, die noch nie einen Security Check durchlaufen haben
- Übernahme einer Anwendung in eine neue Organisation oder in ein neues Entwicklungsteam
- Releases von neuen Versionen

Im Workshop diskutieren wir darüber, wie dieser Aspekt in der Praxis der Teilnehmer angegangen wird und zeigen, warum Kontrolle auf Sourcecode Level heute zum guten Ton gehört. Welche Arten von Sicherheitslücken werden typisch durch Entwickler verursacht und sind schwierig durch Pentests zu finden? Wann wird sinnvollerweise ein Code Audit durchgeführt und welche qualitätssichernden Massnahmen sind wichtig für den Entwicklungsprozess? Warum ist gute Wartbarkeit und Auditierbarkeit wichtig für die Sicherheit von Anwendungen?

Zielgruppe des Workshops sind Anwendungsverantwortliche oder andere Entscheidungsträger, die mehr Verständnis erlangen wollen, wie man den Aspekt Sicherheit in der Anwendungsentwicklung und -pflege verankert.

Der Workshop wird präsentiert von Kay Grosskop und Marc André Hahn von sieber&partners in Zusammenarbeit mit Security Experten der Software Improvement Group aus Amsterdam.

Herr	Frank	Bertisch	Coop Genossenschaft	Leiter Interne Revision
Herr	Mario	Kündig	Coop Genossenschaft	-
Herr	Miguel	Grutsch	Noser Engineering AG	Software Engineer
Herr	Jonas	Richner	Dr. Pascal Sieber & Partners AG	Transformation Consultant
Herr	Bernhard	Peter	Migros Genossenschafts Bund	Head of Development, SSC IT
Herr	André	Clerc	TEMET AG	-
Herr	Felix	Kägi	eXception handler Ltd.	-
Herr	Mark	Schieweck	LAUX LAWYERS AG	-
Herr	Alexander	Schubert	TCG Informatik AG	Professional Services Consultant at TCG Process
Herr	Kay	Grosskop	Dr. Pascal Sieber & Partners AG	Transformation Consultant
Herr	Marc André	Hahn	Dr. Pascal Sieber & Partners AG	CSO