Informationssicherheit und IT-Forensik

3-02 – Grundlagen sicherer Softwareentwicklung

Grundlagen

Lernziele dieser Einheit

- Verstehen, dass Informationssicherheit bereits zu Beginn der Entwicklung eines neuen Systems (z.B. einer Anwendungssoftware) fester Bestandteil des Entwicklungsprojektes sein muss
- Verstehen, dass spätere Änderungen teurer/aufwändiger sind, als die frühestmögliche Berücksichtigung in der Architektur
- Die Grundkonzepte des Security Engineering verstehen, erklären und anwenden können

Grundlagen

- *„I conclude that there are two ways of constructing a software design: One way is to make it so simple that there are obviously no deficiencies and the other way is to make it so complicated that there are no obvious deficiencies. The first method is far more difficult.”*
  Tony Hoare, Dankesrede für den Turingpreis, 1980

Security Engineering

Definition

- *„Security engineering is about building systems to remain dependable in the face of malice, error, or mischance. As a discipline, it focuses on the tools, processes, and method needed to design, implement, and test complete systems, and to adapt existing systems as their environment evolves.“*
  Ross Anderson, Security Engineering, Second Edition, S. 3

- Grundsätzliche Gefahr in der Softwareentwicklung liegt also darin, die falschen Dinge zu schützen, oder die richtigen Dinge mit falschen Mitteln

Security Engineering

Framework von Ross Anderson (Security Engineering, S. 5)

- Nach Anderson müssen für gutes Security Engineering vier Bereiche zusammenspielen, bzw. diese interagieren
  - Policy
    - was muss/soll erreicht werden
  - Mechanism
    - Technik/Algorithmen/Modelle, die zur Verfügung stehen
  - Assurance
    - Grad an Vertrauen in die jeweilige Technik
  - Incentives
    - Angreifer
    - Betreiber/Betreuer

Top-Down-Vorgehen

1. Threat Model
2. Security Policy
   - Statement oder Dokument mit Aussagen, die die Schutzstrategie eines Systems definieren
   - Beispiel (schon fast zu spezifisch): *„Zugriff auf den zentralen Datenbankhost darf nur von zwei Administratoren gleichzeitig erfolgen. Jeder darf nur eine Hälfte des Passwortes kennen und muss diese beim Login so eingeben, dass der andere keine Kenntnis erlangen kann“*
   - Beispiel: *„Das System muss fail-safe sein, es darf zu keiner Zeit für irgendwen (außer dem Administrator X) Zugriff auf die Entwicklungsdaten im Klartext geben“*
3. Security Mechanisms

Security Engineering

Threat Modeling

- *„A vulnerability is a property of a system or its environment which, in conjunction with an internal or external **threat**, can lead to a security failure, which is a breach of the system‘s security policy.“*
  Ross Anderson, Security Engineering, 2008, S. 15
- Threat Modeling: Prozess der Bedrohungsanalyse, der bereits in der Design-Phase eines Software-/Entwicklungs-Projektes stattfindet

Threat Modeling

- Aktivitäten z.B.:
  - Datenflussdiagramm
    - Alle Datenflüsse erfassen und strukturieren
  - Zu schützende Ressourcen identifizieren und strukturieren
  - Fault Tree Analysis
    - Wurzel: ein unerwünschtes Verhalten/Ergebnis/Zustand
    - Knoten: Ursachen/Abstrahierte Gruppen (z.B. „Insiderangriff“)
    - Blätter: Ursachen
    - Jedem Element anschließend Wahrscheinlichkeit/Bedingungen zuweisen
  - Mitigation
- Threat Tree:

Threat Modeling

- Gute Quelle für Praxis-Know-How:
  - https://www.owasp.org/index.php/Application_Threat_Modeling

Security Engineering

Requirements Engineering, Security Requirements Engineering

- Security Requirements Engineering nimmt bewusst auch die Angreiferperspektive ein
- Liefert als Ergebnis auch eine Grundlage für spätere Evaluation (SOLL/IST)
- Muss Teil des allgemeinen Requirements Engineering sein, oder zumindest eng aufeinander abgestimmt
- Dazu gibt es verschiedene Methoden, etwa „CLASP“ und „SQUARE“
- Quelle z.B.: https://www.us-cert.gov/bsi/articles/best-practices/requirements-engineering/security-requirements-engineering

Change Management

- In der Regel unterliegen jegliche Anpassungen der Systeme einem umfassenden Veränderungs-Management (Change Management)
- Ziel ist es, die Anpassungen zu dokumentieren und zu kontrollieren um problematische und fehlerhafte Auswirkungen einer Anpassung möglichst auszuschließen bzw. rückgängig zu machen

Allgemeine Kontrollen: Anwendungskontrollen

- Anwendungskontrollen sind spezifische Kontrollen für betriebswirtschaftliche Anwendungen
- Beinhalten manuelle und automatische Prozeduren
- Klassifikation:
  - Eingabekontrollen
    - Maßnahmen zur Überprüfung von Daten bei der Eingabe in das IT-System hinsichtlich Genauigkeit und Vollständigkeit.
  - Verarbeitungskontrollen
    - Routinen, die sicherstellen, dass die Daten bei der Verarbeitung vollständig und genau sind.
  - Ausgabekontrollen
    - Maßnahmen, die sicherstellen, dass die Ergebnisse der durch den Computer durchgeführten Verarbeitung genau und vollständig sind sowie richtig verteilt werden.

Security Engineering

Allgemeine Kontrollen: Anwendungskontrollen

Name der Kontrolle	Art der Kontrolle	Beschreibung
Kontrollsummen	Eingabe, Verarbeitung	Diese Summen für Eingabe- und Verarbeitungstransaktionen können von einem einfachen Dokumentzähler bis hin zu Summen für Mengenfelder gehen, wie beispielsweise Gesamtverkaufspreis oder Gesamtzinssumme (für einen Transaktionsstapel).
Bearbeitungsprüfung	Eingabe	Programmroutinen, die ausgeführt werden, um Eingabedaten vor der Verarbeitung auf Fehler zu überprüfen. Transaktionen, die den Bearbeitungskriterien nicht entsprechen, werden abgewiesen. Beispielsweise könnte überprüft werden, ob das richtige Format vorliegt (z. B. soll eine zehnstellige Reisepassnummer keine Buchstaben enthalten). Mit dieser Kontrolle lassen sich beispielsweise spezifische Angriffe wie Code-Injection abwehren. Bei Code-Injection versucht der Angreifer Programmcode über Sicherheitslücken bei der Eingabe in ein System einzuschleusen.
Computervergleich / Konsistenzchecks	Eingabe, Verarbeitung	Vergleicht Eingabedaten mit Informationen in Master- oder Transaktionsdateien. Beispielsweise könnte ein Vergleichsprogramm die Anwesenheitszeiten von Angestellten mit einer Master-Abrechnungsdatei vergleichen und auf fehlende oder doppelt vorhandene Zeiten hinweisen.
Ausführung von Kontrollsummen	Verarbeitung, Ausgabe	Vergleich der Gesamtzahl an verarbeiteten Transaktionen mit der Gesamtzahl der ein- oder ausgegebenen Transaktionen.
Bericht über die Verteilung	Ausgabe	Dokumentation, die auflistet, dass die autorisierten Empfänger ihre Berichte und andere kritische Dokumente erhalten haben.

Quelle: Laudon/Laudon/Schoder (2015), Tabelle 15.10

Security Engineering

Definition: Fehlertolerante Computersysteme

- IT-Systeme, die zusätzliche, zumeist redundante Hardware-, Software und Stromversorgungs-komponenten verwenden, um eine unterbrechungsfreie Verfügbarkeit der Systeme zu gewährleisten.

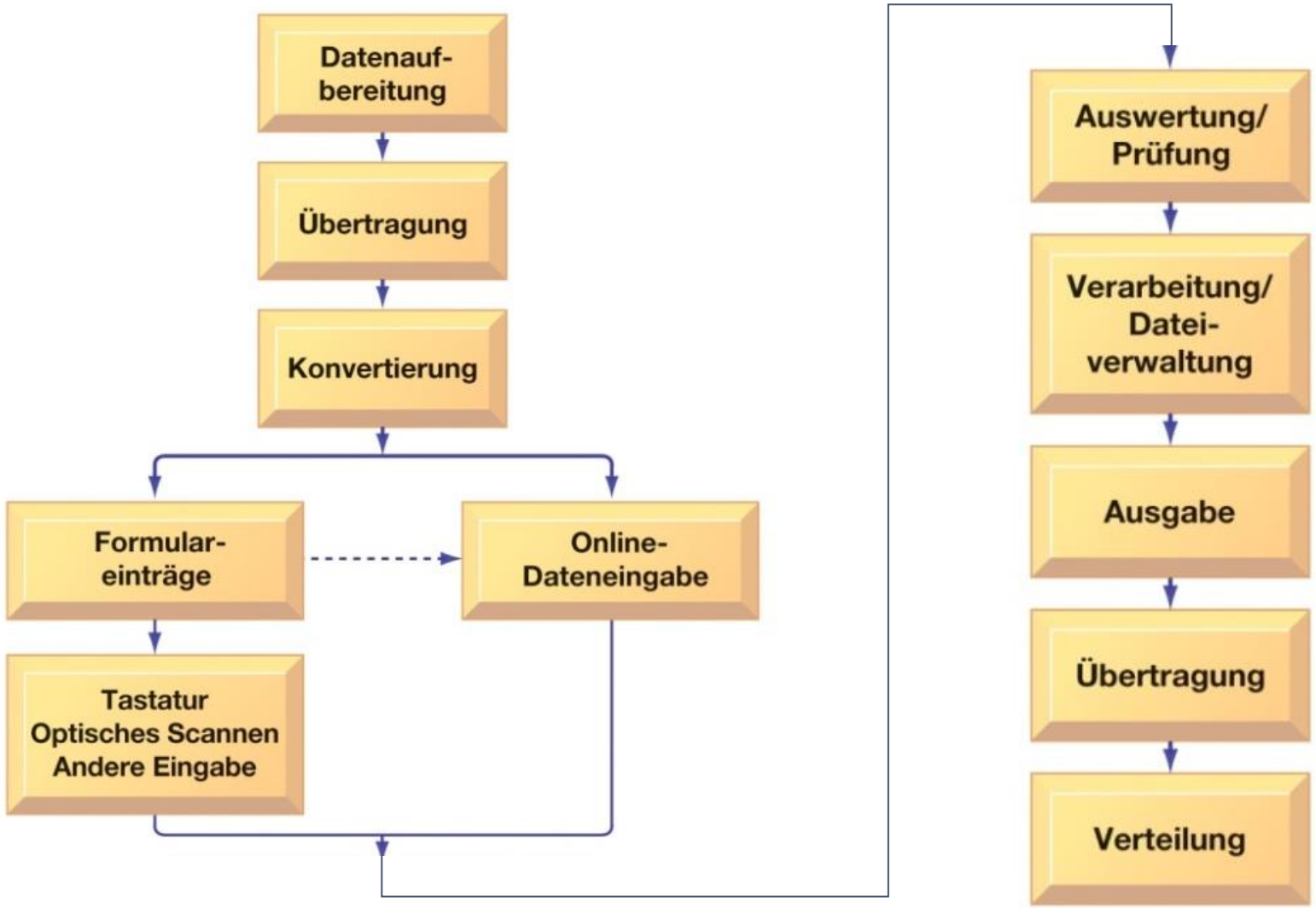
Security Engineering

IEC 61508

- Functional Safety
- *„Functional safety is a concept applicable across all industry sectors. It is fundamental to the enabling of complex technology used for safety-related systems. It provides the assurance that the safety-related systems will offer the necessary risk reduction required to achieve safety for the equipment.”*
- Quelle: https://www.iec.ch/functionalsafety/
- Adressiert insbesondere bereits die Entwicklungsphase sicherheitskritischer Anwendungen
- Bezieht sich u.a. auf den aktuellen Stand der Technik
- Lebenszyklus-Modell

Security Engineering

Probleme der Systemqualität: Software und Daten



Quelle: Laudon/Laudon/Schoder (2010), Abbildung 15.7

Programmfehler

- Es ist fast unmöglich, alle Fehler in großen Programmen zu beseitigen und deren Ernsthaftigkeit ist unbekannt
- Hauptfehlerursache ist die Komplexität des Codes
- **Validierung**: Wurde das richtige System gebaut?
- **Verifikation**: Wurde das System richtig gebaut?
- Aufgabe des Testens erfordert Experten
- Andere Fehlerarten: provozierte Fehler (buffer overflow), falsche Werkzeuge

Security Engineering

Typische Fehler bei der Einführung und dem Betrieb von Software

Fehler	Beschreibung / Beispiel
Software wird vor der Einführung nicht gründlich getestet.	Bei der Softwareentwicklung wird zumeist von der Gutwilligkeit der Benutzer ausgegangen, statt bewusste Falscheingaben zu berücksichtigen. Insbesondere eine fehlende Validierung von Benutzereingaben führt oftmals zu Sicherheitsproblemen (z. B. Möglichkeit von Code-Injection). Darüber hinaus bedarf es im Hinblick auf die Sicherheit des Gesamtsystems umfassender Integrationstests, denn das System ist nur so sicher, wie das schwächste Glied. Ferner ergeben sich oftmals durch nicht entfernten Test- oder Debug-Code Zugänge für unberechtigte Zugriffe.
Software erfüllt nur scheinbar die Sicherheitsanforderungen aus dem Prospekt bzw. Pflichtenheft.	Auch wenn Anbieter zahlreiche effektive Sicherheitsmaßnahmen in ihren Produkten anpreisen, sollten diese vor der Einführung überprüft werden. So untersuchte das Marktforschungsunternehmen Gartner Ende 2007 Anbieter von Webmailsystemen. Eines der als führend bewerteten Produkte enthielt in einem Release den folgenden Fehler: Die Software übertrug den zum Entschlüsseln der E-Mail verwendeten Schlüssel in jedem Fall an den Browser – auch wenn dieser ein falsches Passwort eingegeben hatte. So war es für Angreifer ohne Weiteres möglich, verschlüsselte E-Mails zu lesen. Bei der Softwareauswahl sollten speziell die Fähigkeiten des Anbieters untersucht werden, ob dieser potenzielle Sicherheitslücken konsequent publiziert und entsprechende Patches umgehend zur Verfügung stellt. Es sollte auch analysiert werden, welche Prozesse und Verfahren der Anbieter verwendet, um seine Systeme sicher zu gestalten, und welchen Stellenwert die IT-Sicherheit bei der Entwicklung der Produkte besitzt.
Sicherheitsmaßnahmen wurden für die Einführungsphase implementiert. Für den Betrieb des Systems werden allerdings keine effektiven Mechanismen zum Sicherheitsmanagement angeboten.	Die SOA-Middleware eines führenden Herstellers nutzte Mitte 2008 State-of-the-Art-Mechanismen der WS-Security (z. B. asymmetrische Schlüssel und PKI für Verschlüsselung und Authentifizierung). Leider enthielt das System keine Mechanismen für das Schlüssel-Management. Es gab keine Prozeduren, um ablaufende Zertifikate zu verlängern oder zu erneuern, und es gab keine Möglichkeit, Sperrlisten abzurufen.

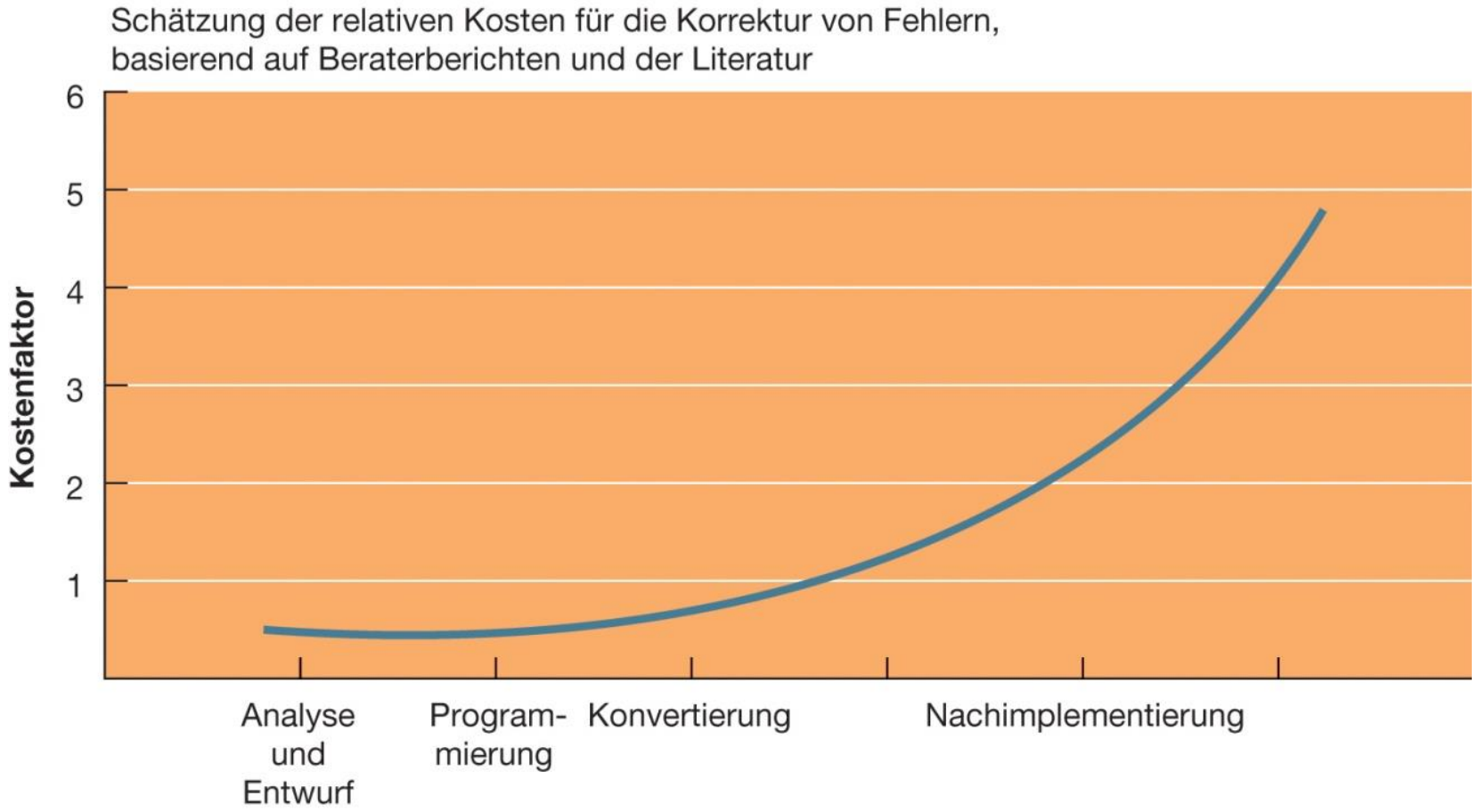
Quelle: Laudon/Laudon/Schoder (2010), Abbildung 15.7

Wartungsalbtraum

- In den meisten Unternehmen wird fast die Hälfte der Zeit des für Informationssysteme zuständigen Personals für die Wartung und den Betrieb vorhandener Systeme aufgewendet (Run-the-IT-costs versus Change-the-IT-costs)
- Gründe:
  - Anpassung an Änderungen im Unternehmen
  - Komplexität der Software
  - Fehlerhafte Systemanalyse und Entwurf
  - Vielfalt verschiedener Systeme

Security Engineering

Kostenaufwand für Fehler im Systementwicklungszyklus



Quelle: Laudon/Laudon/Schoder (2015), Abbildung 15.2

Einsatz von Kryptografie im Unternehmen

- Unternehmensspezifische Entscheidungen
  - Wie sensibel sind die Daten?
  - Wie lange ist der Inhalt geheim zu halten?
  - Geht es um gespeicherte (data at rest) oder übertragene (data in transit) Daten, deren Vertraulichkeit zu schützen ist?
  - Sind die Daten aufgrund von firmeneigenen oder regulatorischen Vorgaben zu schützen?
  - Wie sicher ist die Umgebung, in der die Verschlüsselung stattfindet?
  - Soll Hardware- oder Softwareverschlüsselung angewandt werden?
  - Müssen die Implementierungen zertifiziert sein?
  - Muss die Verschlüsselung Ende-zu-Ende oder „nur“ von Tür-zu-Tür erfolgen?
- Unternehmen müssen verstehen, wie man Kryptografie implementiert und betreibt

Security Engineering

Allgemeine Konstruktionsprinzipien (nach Claudia Eckert, S. 184f, 9. Auflage)

- Erlaubnisprinzip (fail-safe defaults, default deny)
- Vollständigkeitsprinzip (complete mediation)
  - Jeder Zugriff muss auf dessen Zulässigkeit hin geprüft werden
- Prinzip der minimalen Rechte (need-to-know)
- Prinzip der Benutzerakzeptanz (economy of mechanism)
- Prinzip des offenen Entwurfs (open design)
  - Kein Security by Obscurity

- In der Praxis denkbar: Zonen-Modell
  - Z.B. besonders sicherer Kern
  - Weniger sichere, weniger privilegierte Bereiche „drum herum“

Security Engineering

BSI-Risikoanalyse

- BSI-Standard 200-3: Risikoanalyse auf der Basis von IT-Grundschutz
- Nach BSI dann anzuwenden, wenn Sicherheits-anforderungen über ein normales Maß hinausgehen
- Quelle: https://www.bsi.bund.de/DE/Themen/Unternehmen-und-Organisationen/Standards-und-Zertifizierung/IT-Grundschutz/BSI-Standards/BSI-Standard-200-3-Risikomanagement/bsi-standard-200-3-risikomanagement_node.html

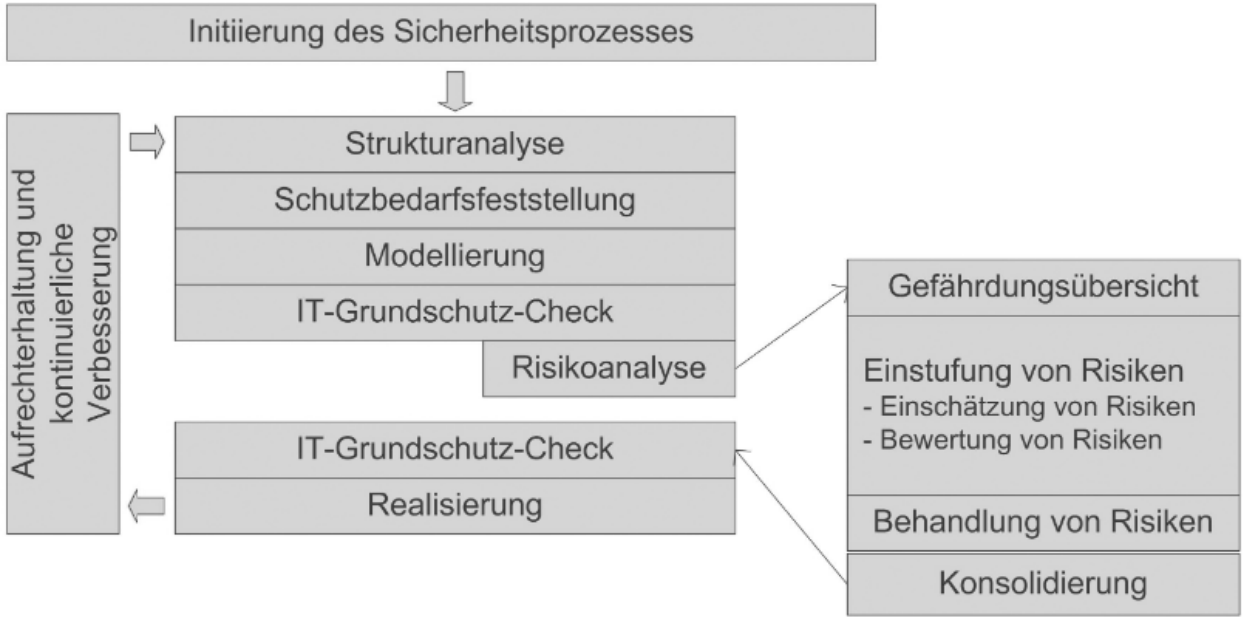
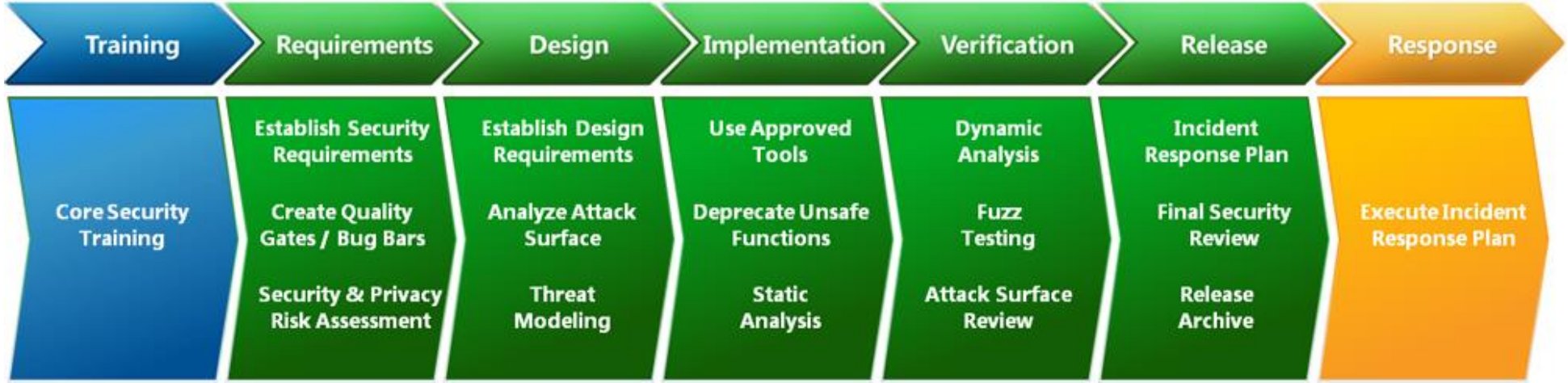


Abbildung 1: Integration der Risikoanalyse in den Sicherheitsprozess

Security Engineering

Security Development Lifecycle (SDL)

- Von Microsoft (auch) für eigene Verwendung entwickelt
- Drei Paradigmen:
  - Secure by Design
  - Secure by Default
  - Secure in Deployment
- Quelle: https://www.microsoft.com/en-us/securityengineering/sdl/ und Quelle Grafik https://social.technet.microsoft.com/wiki/contents/articles/7100.the-security-development-lifecycle.aspx



Exkurs: Sicherheit in Programmiersprachen

Typsicherheit

- „[B]y far the most popular and best established lightweight formal methods are t*ype systems […]* A type system is a tractable syntactic method for proving the absence of certain program behaviors […].“ - Benjamin C. Pierce, *„Types and Programming Languages“*
- Typisierung ist ein eigenes Thema für sich (schwach, stark, statisch, dynamisch, …)
- Von Typsicherheit spricht man, wenn keine Operation in der Programmiersprache es erlaubt, die Regeln des Typsystems zu verletzen.
- Beispiel: eine typsichere Sprache verhindert, dass außerhalb der Grenzen eines Arrays gelesen oder geschrieben wird, denn dieser Speicherbereich ist undefiniert und entspricht nicht dem definierten Typ des Arrays

  -> in C ist dies möglich, da C nicht typsicher ist

Exkurs: Sicherheit in Programmiersprachen

Garbage Collection: Wie kann ein Computerprogramm Speicher verwalten?

- Variante 1: manuelles Speichermanagement (klassischer Vertreter: C)
- Variante 2: automatisches Speichermanagement durch einen Garbage Collector (Python, Perl, Java, C#, …)
- Neben diversen anderen Punkten hat Garbage Collection auch einen Sicherheitsaspekt:
- Bestimmte Fehlerklassen wie Use-after-Free oder Double-free-Bugs werden systematisch verhindert
  - Durch steigende Sicherheitsmechanismen gegen „klassische“ Programmierfehler wie Buffer Overflows gehören diese Fehlerklassen mit zu den wichtigsten Quellen für moderne Exploits, etwa in Webbrowsern
- „Sichere“ Programmiersprachen können auch ohne Garbage Collection arbeiten (vgl. Rust)

Exkurs: Sicherheit in Programmiersprachen

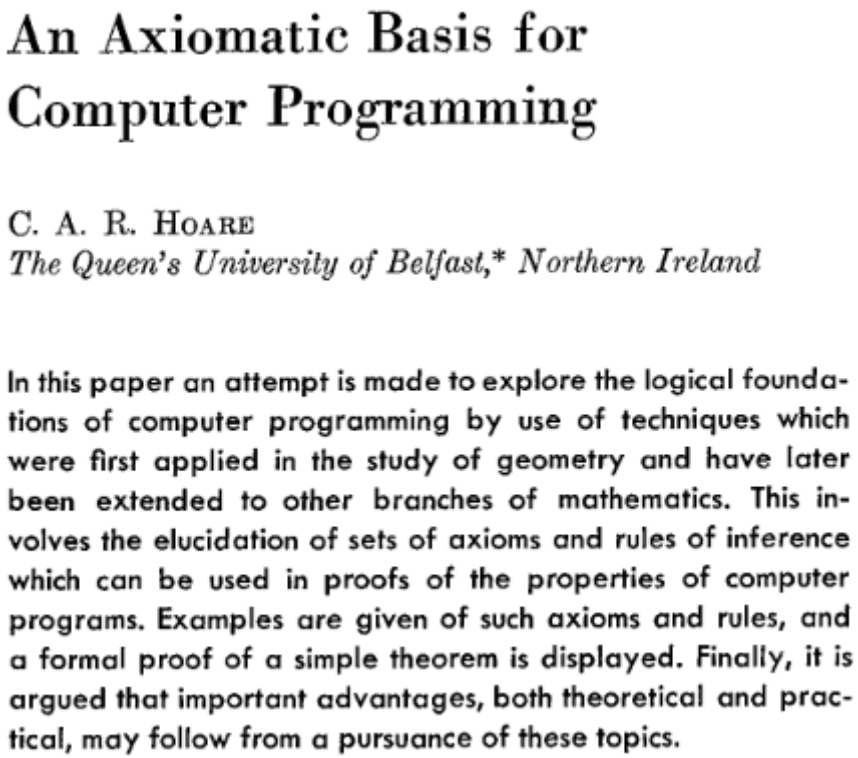
Fazit: Sind typ- und speichersichere Programmiersprachen die Lösung aller Probleme?

- Für alle Konzepte ist noch keine Lösung gefunden (z.B. können in Rust Bereiche explizit als „unsafe“ markiert werden)
- Zahlreiche Schnittstellen existieren (z.B. können in Android-Apps [Java-basiert] Teile in „native code“ [C-basiert] programmiert werden)
- Keine sichere Programmiersprache verhindert, dass der Programmierer Fehler in der Businesslogik macht:
  - Wenn ein Softwareprodukt für Automobilhändler 1€ als Verkaufspreis für einen neuen BMW akzeptiert, hilft auch die sicherste Programmiersprache nicht dabei, diesen wirtschaftlichen Schaden zu verhindern.

Exkurs: Formale Korrektheitsbeweise

Hoare-Kalkül

- Ein sehr kluges System, um die Korrektheit von Software beweisen zu können
- Ist jedoch nicht für jegliche Software beliebigen Umfangs geeignet
- Quelle: Charles Antony Richard Hoare: An Axiomatic Basis for Computer Programming, In: Communications of the ACM. Bd. 12, Nr. 10, 1969, S. 576–585
- Was, wenn später der Compiler einen Fehler macht?
- Was, wenn die Annahmen fehlerhaft sind?



An Axiomatic Basis for Computer Programming

C. A. R. Hoare
The Queen's University of Belfast, Northern Ireland*

In this paper an attempt is made to explore the logical foundations of computer programming by use of techniques which were first applied in the study of geometry and have later been extended to other branches of mathematics. This involves the elucidation of sets of axioms and rules of inference which can be used in proofs of the properties of computer programs. Examples are given of such axioms and rules, and a formal proof of a simple theorem is displayed. Finally, it is argued that important advantages, both theoretical and practical, may follow from a pursuance of these topics.

Ein Auftrag an Sie: unser Musterfall

Wiederkehrende, vorlesungs- und übungsbegleitende Übungsaufgabe

- Überarbeiten Sie Ihre Planung für die Errichtung einer „smarten“ Einbruchmeldeanlage 2.0 für das Handwerksunternehmen Ihrer Eltern
- Nehmen Sie sich dafür jetzt 5 Minuten Zeit
- Beantworten und begründen Sie unter eigenen Annahmen z.B. folgende Fragen:
  - Ändern Sie Ihre bisherige Planung?
  - Denken Sie kurz darüber nach, wie Sie den Entwicklungsprozess für die Software der Einbruchmeldeanlage gestalten wollen. Welche Programmiersprache erscheint Ihnen warum geeignet? Welche Bedrohungen wollen Sie bereits im Design des Systems adressieren?
  - Skizzieren Sie mögliche Problemstellen. Wo könnten Probleme lauern?