Fraunhofer SIT

FRAUNHOFER-INSTITUT FÜR SICHERE INFORMATIONSTECHNOLOGIE

SIT TECHNICAL REPORTS

Trend- und Strategiebericht

Entwicklung sicherer Software durch Security by Design

Michael Waidner, Michael Backes, Jörn Müller-Quade

FRAUNHOFER VERLAG

Fraunhofer
SIT

Entwicklung sicherer Software durch Security by Design

Michael Waidner (Hrsg.), Michael Backes (Hrsg.), Jörn Müller-Quade (Hrsg.), Eric Bodden, Markus Schneider, Michael Kreutzer, Mira Mezini, Christian Hammer, Andreas Zeller, Dirk Achenbach, Matthias Huber, Daniel Kraschewski

SIT Technical Reports
SIT-TR-2013-01

Mai 2013

Fraunhofer-Institut für Sichere
Informationstechnologie SIT
Rheinstraße 75
64295 Darmstadt

GEFÖRDERT VOM

Dieser Trend- und Strategiebericht wurde gefördert vom Bundesministerium für Bildung und Forschung.

Bundesministerium
für Bildung
und Forschung

FRAUNHOFER VERLAG

IMPRESSUM

Kontaktadresse:
Fraunhofer-Institut für
Sichere Informationstechnologie SIT
Rheinstraße 75
64295 Darmstadt
Telefon 06151 869-213
Telefax 06151 869-224
E-Mail info@sit.fraunhofer.de
URL www.sit.fraunhofer.de

Bibliografische Information der Deutschen Nationalbibliothek
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.d-nb.de abrufbar.

Hrsg. Michael Waidner
SIT Technical Reports
Entwicklung sicherer Software durch Security by Design (SIT-TR-2013-01)
Michael Waidner (Hrsg.), Michael Backes (Hrsg.), Jörn Müller-Quade (Hrsg.), Eric Bodden, Markus Schneider, Michael Kreutzer, Mira Mezini, Christian Hammer, Andreas Zeller, Dirk Achenbach, Matthias Huber, Daniel Kraschewski
ISBN 978-3-8396-0567-7
ISSN 2192-8169

Druck und Weiterverarbeitung:
IRB Mediendienstleistungen
Fraunhofer-Informationszentrum Raum und Bau IRB, Stuttgart

Für den Druck des Buches wurde chlor- und säurefreies Papier verwendet.

Fraunhofer-Informationszentrum Raum und Bau IRB
Postfach 800469, 70504 Stuttgart
Nobelstraße 12, 70569 Stuttgart
Telefon 0711 970-2500
Telefax 0711 970-2508
E-Mail verlag@fraunhofer.de
URL http://verlag.fraunhofer.de

Entwicklung sicherer Software durch Security by Design

Michael Waidner (Hrsg.), Michael Backes (Hrsg.), Jörn Müller-Quade (Hrsg.), Eric Bodden, Markus Schneider, Michael Kreutzer, Mira Mezini, Christian Hammer, Andreas Zeller, Dirk Achenbach, Matthias Huber, Daniel Kraschewski

---

Dieser Trend- und Strategiebericht vertritt die These, dass die Entwicklung und Integration sicherer Software nach dem Prinzip *Security by Design* ausgestaltet werden muss und benennt entsprechende Herausforderungen für eine praxisorientierte Forschungsagenda. Software ist heute wie auch zukünftig der wichtigste Treiber von Innovationen in vielen Anwendungsbereichen und Branchen. Viele Schwachstellen und Angriffe lassen sich auf Sicherheitslücken in Anwendungssoftware zurückführen. Sicherheitsfragen werden bei der heutigen Entwicklung oder Integration von Anwendungssoftware entweder überhaupt nicht oder nur unzureichend betrachtet, so dass durch Anwendungssoftware immer wieder neue Ansatzpunkte für Angriffe entstehen. So wird die Sicherheit von Software neben der Funktionalität für Anwender und Hersteller immer wichtiger. Die Anwendung neuer praktischer Methoden und das systematische Befolgen von Sicherheitsprozessen sollen Hersteller und Integratoren von Software bei der Vermeidung von Sicherheitslücken unterstützen. Die Verbesserung von Entwicklungs- und Sicherheitsprozessen bietet Herstellern auch die Möglichkeit, bei verbesserten Sicherheitseigenschaften Kosten und Entwicklungszeiten von Software zu reduzieren. Für Unternehmen hat dieser Schritt eine große strategische Bedeutung mit großer Relevanz für deren mittel- bis langfristige Wettbewerbsfähigkeit. Da Softwareprodukte und Softwareentwicklungsprozesse heute sehr komplex sein können, ist es für Hersteller nicht klar, wie *Security by Design* und die hierfür erforderlichen Sicherheitsprozesse nutzbringend und wirtschaftlich umgesetzt werden können. Es ist die Aufgabe der angewandten Forschung, die Herausforderungen in diesem Zusammenhang anzugehen, zu bewältigen und verwertbare Lösungen in die Praxis zu transferieren.

Key Words: Security by Design, Secure Engineering, Software Engineering, Security Development Lifecycle, Application Security, Supply Chain, Software Development

---

Michael Waidner (Hrsg.)
EC SPRIDE, TU Darmstadt,
Fraunhofer-Institut für Sichere Informationstechnologie (SIT)
Fraunhofer SIT, Rheinstraße 75, 64295 Darmstadt
www.sit.fraunhofer.de, www.ec-spride.de, www.informatik.tu-darmstadt.de

Michael Backes (Hrsg.)
CISPA, Saarland University
Universität des Saarlandes, Postfach 151150, 66041 Saarbrücken
www.cs.uni-saarland.de, www.cispa-security.de

Jörn Müller-Quade (Hrsg.)
KASTEL, Karlsruher Institut für Technologie (KIT)
Karlsruher Institut für Technologie, Kaiserstraße 12, 76131 Karlsruhe
www.kit.edu, www.kastel.kit.edu

Eric Bodden, Markus Schneider
EC SPRIDE, Fraunhofer SIT
Fraunhofer SIT, Rheinstraße 75, 64295 Darmstadt
www.ec-spride.de, www.sit.fraunhofer.de

Michael Kreutzer, Mira Mezini
EC SPRIDE, TU Darmstadt
EC SPRIDE, Mornewegstraße 30, 64293 Darmstadt
www.ec-spride.de, www.informatik.tu-darmstadt.de

Christian Hammer, Andreas Zeller
CISPA, Universität des Saarlandes
Universität des Saarlandes, Postfach 151150, 66041 Saarbrücken
www.cispa-security.de, www.cs.uni-saarland.de

Dirk Achenbach, Matthias Huber, Daniel Kraschewski
KASTEL, Karlsruher Institut für Technologie (KIT)
Karlsruher Institut für Technologie, Kaiserstraße 12, 76131 Karlsruhe
www.kastel.kit.edu, www.kit.edu

INHALTSVERZEICHNIS

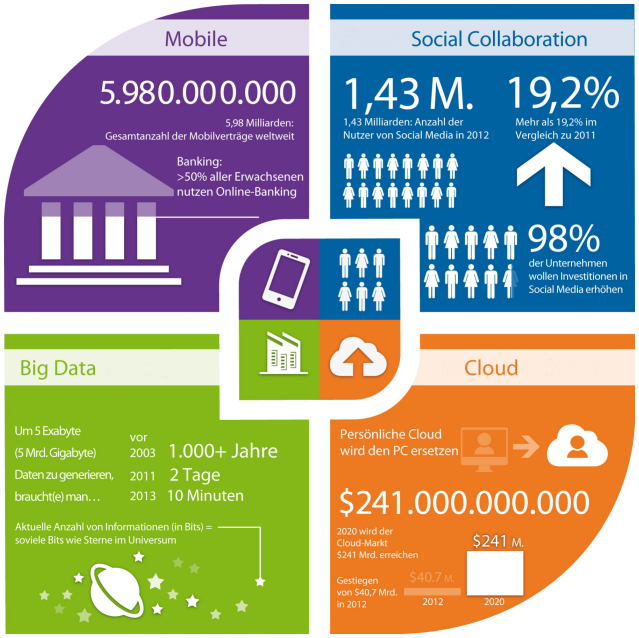
Grußwort von Herrn Karl-Heinz Streibich

Vorstandsvorsitzender der Software AG

Sehr geehrte Damen und Herren,
werte Kollegen aus Wissenschaft und Wirtschaft,

alle zwei Tage erschafft die digitale Welt heute so viele Daten, wie in der Zeit von Anbeginn der menschlichen Zivilisation bis zum Jahr 2003. Milliarden mobiler Endgeräte sind im Gebrauch und aus unserem Alltag nicht mehr wegzudenken – Nutzer dokumentieren, wo sie sind, mit wem sie sprechen, was sie bewegt. Aus dem klassischen Mobiltelefon ist eine Datenquelle geworden.

Erstmals haben wir die technischen Möglichkeiten, unsere Umwelt, unseren Alltag und unser Leben in Echtzeit zu vermessen. Wir haben es hier in der globalen Softwareindustrie mit einer einmaligen Konstellation zu tun, da gleichzeitig vier technologische Megatrends aufeinander treffen:



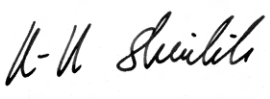
- Mobile – die zunehmende mobile Kommunikation und die mobile Nutzung des Internets.
- Cloud Computing – die Verlagerung von Daten und Anwendungen ins Internet.
- Social Collaboration – die verstärkte Nutzung sozialer Netzwerke.
- Big Data – die Bearbeitung und Analyse riesiger Datenmengen in Echtzeit.

Software ist zum fundamentalen Werkstoff und Innovationstreiber in nahezu allen Industrien geworden. Prozesse, Produkte und Produktionsverfahren werden mit dem Internet verbunden und können dadurch auf völlig neue Art und Weise mit digitalen Informationen angereichert und vernetzt werden. Mit dieser steigenden Vernetzung

wächst auch der Kundenbedarf an sicheren, digitalen Lösungen über die gesamte Wertschöpfungskette. Heute ist die Software AG mit den Produktfamilien Adabas und Natural, webMethods, ARIS und Terracotta führend in 15 Marktsektoren. Wir bieten unseren Kunden die qualitativ besten Lösungen zur Digitalisierung ihres Unternehmens an. Unsere führende Marktposition ist das Ergebnis jahrzehntelanger Forschungs- und Entwicklungsarbeit – auch über Unternehmensgrenzen hinweg – und die Grundlage der strategischen Partnerschaft mit dem European Center for Security and Privacy by Design (EC SPRIDE). Die Software AG kann durch diese Partnerschaft auf die Kompetenzen einer wissenschaftlichen Einrichtung der Spitzenforschung im Bereich der IT-Sicherheit zurückgreifen und die Erkenntnisse in ihren Software-Entwicklungsprozess einfließen lassen. Schwerpunkt der gemeinsamen Aktivitäten ist das Labor für Secure Engineering. Dieses Secure Engineering Lab bildet den organisatorischen Rahmen für die gemeinsamen Forschungsaktivitäten, den Ausbau unserer Entwicklungsmannschaft sowie die kontinuierliche Optimierung unserer Entwicklungsprozesse auf Basis der neuesten Forschungsergebnisse. Die Methoden der Software-Produktion müssen sich den neuen Ansprüchen und Gegebenheiten anpassen, die zunehmend gekennzeichnet sind durch Dezentralisierung und Verteilung von Entwicklungsarbeiten (weltweit verteilte Entwicklungsteams, Integration von Dritt- und Open-Source-Komponenten, unternehmensübergreifende Prozesse). Sicherheit muss von Anfang an im Entwicklungsprozess berücksichtigt werden (Security by Design), dazu sind auch Änderungen und Erweiterungen von IT-Tools unabdingbar. In diesen Bereichen arbeiten EC SPRIDE und die Software AG gemeinsam daran, neueste Forschungsergebnisse unter spezifischen Gegebenheiten in die Praxis umzusetzen.

Ziel ist es, eine enge Verzahnung von Wirtschaft und Wissenschaft herzustellen, denn innovative Produkte und Dienstleistungen sind ohne sichere Software in Zukunft nicht mehr denkbar. Die Wettbewerbsfähigkeit der deutschen Wirtschaft wird entscheidend von der Fähigkeit abhängen, Software-basierte Produkte und Dienstleistungen mit höchster Qualität zu erstellen. Die Softwarekompetenz wird die Voraussetzung dafür sein, dass Deutschland seine führende Stellung im Ingenieurbereich halten und seine Position als eine der führenden Exportnationen ausbauen kann. Von einer dynamischen und erfolgreichen deutschen Softwareindustrie gehen wichtige Impulse für sämtliche Wirtschaftszweige und damit für die Wettbewerbsfähigkeit der deutschen Volkswirtschaft aus. Deshalb ist uns die Kooperation mit einer aktiven und engagierten Forschergemeinde, wie dem EC SPRIDE, ein wichtiges Anliegen.

Ihr,



Karl-Heinz Streibich - Vorstandsvorsitzender der Software AG

1. SOFTWARESICHERHEIT UND SOFTWAREENTWICKLUNG IM WANDEL

Die meisten Innovationen basieren heute auf Informationstechnologie. Das gilt für die Innovationen der IT-Branche selbst und darüber hinaus für andere Branchen wie etwa Energie, Finanzen, Gesundheit, Handel, Logistik, Medien, Produktion, Umwelt und Verkehr. Überall dort spielt Informationstechnologie, die häufig als Software implementiert ist, eine herausragende Rolle.

Heute setzen Unternehmen und Organisationen Anwendungssoftware in wichtigen Geschäftsprozessen ein, die oft kritisch für den Geschäftserfolg sind. Diese Anwendungssoftware zeichnet sich durch spezielle Funktionen aus, die für die verschiedensten Zwecke benötigt werden. Bei der Entwicklung von Anwendungssoftware werden heute fast ausschließlich diese gewünschten Funktionen betrachtet. Die Entwickler sind Experten in den jeweiligen Anwendungsdomänen. Sicherheit wird im Entwicklungsprozess entweder gar nicht oder nur am Rande betrachtet. Dadurch entstehen zwangsläufig Sicherheitslücken in der Anwendungssoftware. Entsprechend versuchen Hacker immer wieder, sich durch diese Sicherheitslücken erfolgreich Zugang zu Daten und Systemen zu verschaffen und sich auf diesem Weg zu bereichern [BKA12; BKA11]. Somit wird neben der Funktionalität von Software, zu deren Zweck sie implementiert wurde, die Sicherheit von Software für Anwender und für Hersteller immer wichtiger. Sicherheitslücken in Anwendungssoftware stellen große Risiken für Organisation und Unternehmen dar und sie werden mittlerweile als gefährlichste Quelle von Bedrohungen verstanden (siehe hierzu bspw. Abbildung 1). Die begründete Sorge vor finanziellen Verlusten rückt bei Anwendern immer mehr die Frage nach der Sicherheit von Anwendungssoftware in den Mittelpunkt. Entsprechend sind die Hersteller von Anwendungssoftware aufgefordert zu reagieren und die Sicherheit ihrer Produkte zu verbessern.

Bei der bisherigen Vorgehensweise haben Hersteller versucht, Aufgaben zur Sicherheit zu externalisieren. Dies geschah mit Firewalls, Wrappern, Intrusion Detection oder Malware-Schutz. Hat eine Anwendungssoftware Sicherheitslücken, dann lassen sich diese mittels extern hinzugefügten Sicherheitskomponenten nicht immer ohne Funktionalitätsverlust schließen. Die derzeit stark verbreitete Praxis zur Softwareentwicklung führt dazu, dass ständig Sicherheitslücken gefunden werden, die dann möglichst schnell in aufwändigen und teuren Patchzyklen zu schließen sind.

Nachdem die Ursachen für Sicherheitslücken in der Praxis besser verstanden werden als dies noch vor ein paar Jahren der Fall war reift die Erkenntnis immer mehr, dass die Sicherheit von Software bei der Entwicklung und Integration viel stärker berücksichtigt werden muss. Ohne Verbesserung der Sicherheit in Anwendungssoftware wird sich das Lagebild hinsichtlich der Bedrohungen und Risiken nicht substanziell verbessern lassen.

Zur Verbesserung der Sicherheit von Anwendungssoftware ist es dringend erforderlich, dass Sicherheit von Beginn an bei der Entwicklung, also bereits in der De-

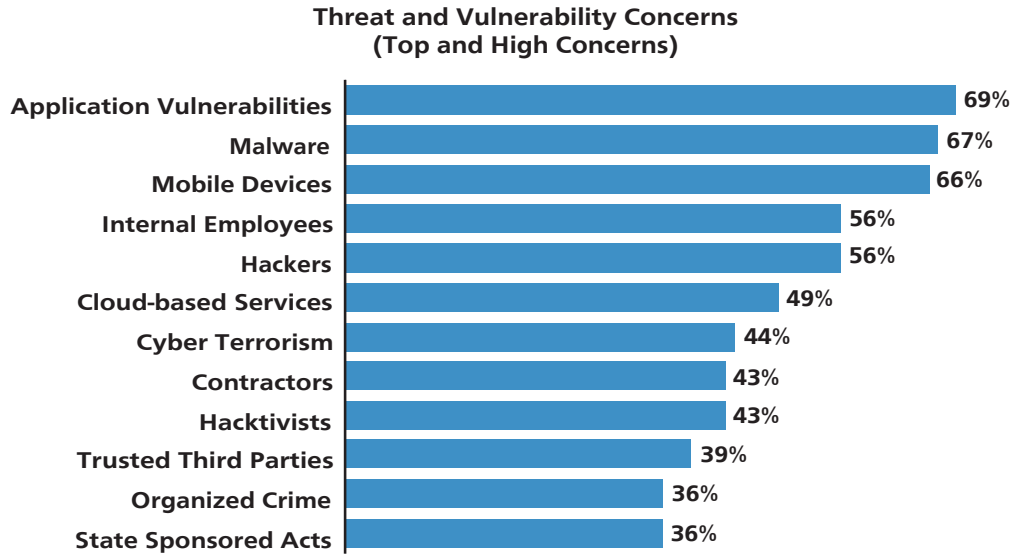


Abbildung 1: Gemäß einer Untersuchung von Frost & Sullivan, $(ISC)^2$ und Booz, Allen, Hamilton geht von Sicherheitslücken in Anwendungssoftware die stärkste Bedrohung aus (Quelle: [FIB13])

signphase, berücksichtigt wird und dann über dem kompletten Lebenszyklus der Softwareentwicklung betrachtet werden muss (siehe zum Beispiel den *Security Development Lifecycle* (SDL) von Microsoft [Mic10]). Von diesem Ansatz versprechen sich Hersteller nicht nur Produkte mit besseren Sicherheitseigenschaften, sondern auch niedrigere Kosten für die Herstellung von sicherer Software [For11a; Abe10]. Je früher Sicherheitslücken bei der Entwicklung durch einen solchen Sicherheitsprozess erkannt werden, desto niedriger sind die Kosten zu deren Behebung: „Eine nachträgliche Implementierung von Sicherheitsmaßnahmen ist bedeutend teurer und bietet im Allgemeinen weniger Schutz als Sicherheit, die von Beginn an in den Systementwicklungsprozess oder in den Auswahlprozess für ein Produkt integriert wurde. Sicherheit sollte daher integrierter Bestandteil des gesamten Lebenszyklus eines IT-Systems bzw. eines Produktes sein." [BSI06]

Damit wird die strategische Dimension von Sicherheitsprozessen deutlich. Wenn Unternehmen als Hersteller von Software ihre Entwicklungs- und Sicherheitsprozesse entsprechend anpassen und weiter entwickeln, können sie die Sicherheit ihrer Produkte wie auch ihre Wettbewerbsfähigkeit verbessern. Hierzu braucht es einen Paradigmenwechsel, so dass sich Sicherheitsprozesse wirtschaftlich in die Praxis umsetzen lassen und einzelne Unternehmen bereit sind, Startinvestitionen für diesen Wandel aufzubringen. Die Einführung von Sicherheitsprozessen ist für Softwarehersteller ein wichtiger Aspekt, um sich im Wettbewerb zu behaupten.

Software und Softwareentwicklungsprozesse können insbesondere bei größeren Projekten sehr komplex sein. So können in einem einzigen Softwareendprodukt heute Softwarekomponenten vieler verschiedener Hersteller integriert sein, wofür die heutigen Sicherheitsprozesse unzureichend sind. Aus Gründen der Zeit und Wirtschaftlichkeit können auch Komponenten integriert werden, die noch unter anderen

Rahmenbedingungen entwickelt wurden (Legacy). Die Komplexität der Softwareentwicklung und der Faktor *Mensch* bei der Entwicklung führen immer wieder zu Fehlern und somit zu Sicherheitslücken. Diese Problematik ist durch Verwendung von unterstützenden Werkzeugen zu entschärfen.

Die Sicherheitsprozesse bei der Softwareentwicklung müssen sich in Anbetracht des Bedarfs an sicherer Software auf der einen Seite und der Verletzlichkeit von Wirtschaft und Gesellschaft auf der anderen Seite stark verändern. Dennoch können Transformationsprozesse bei der Herstellung und Integration von Software nur gelingen, wenn sich diese evolutionär gestalten lassen. Es muss berücksichtigt werden, dass Hersteller nicht *ad hoc* auf andere Entwicklerressourcen zurückgreifen können. Deshalb wird es bei der industriellen Softwareentwicklung sehr wichtig sein, neue Werkzeuge zu erstellen, die in vorhandene Entwicklungsumgebungen zu integrieren sind und die gegenwärtigen Entwickler mit weniger stark ausgeprägter Sicherheitsexpertise darin unterstützen, Sicherheitslücken zu vermeiden. Es ist davon auszugehen, dass sich die industrielle Softwareentwicklung und die damit einhergehenden Sicherheitsprozesse in den kommenden Jahren stark weiterentwickeln werden. Am Ziel steht die Erwartung, dass Sicherheit von Software bereits von der Designphase an berücksichtigt wird und über dem Lebenszyklus von Software systematisch und methodisch verbessert wird. Diese Erwartung ist gekennzeichnet durch verschiedene Visionen, die aus unterschiedlichen Perspektiven Idealbilder der Entwicklung sicherer Software darstellen. Damit diese Visionen Wirklichkeit werden können, muss die Forschung eine Reihe von Herausforderungen angehen und bewältigen. Diese müssen danach in einem weiteren Schritt in die reale Softwareentwicklung transferiert werden.

Dieser Trend- und Strategiebericht beschreibt die Idealbilder der zukünftigen Entwicklung sicherer Software als Visionen und stellt die Herausforderungen dar, welche die praxisorientierte Forschungsagenda in den kommenden Jahren bestimmen werden.

2. DIE BEDEUTUNG VON SECURITY BY DESIGN

2.1 Begriff Security by Design

Der Begriff *Security by Design* kann in unterschiedlicher Weise verstanden werden. Im engeren Sinn bedeutet *Security by Design* die Berücksichtigung von Sicherheit bereits in der Entwurfsphase des Softwareentwicklungsprozesses. In einem weiter gefassten Sinn kann man unter *Security by Design* den systematisch organisierten und methodisch ausgestatteten Rahmen verstehen, der im Lebenszyklus von sicherer Software Anwendung findet. Dieser Rahmen umfasst dann beispielsweise die Verankerung sicherer Softwareentwicklung auf der Governance-Ebene, einzelne Sicherheitsprozesse für die Phasen im Lebenszyklus der Software und Sicherheitsanalysen von zu integrierenden Softwarekomponenten anderer Hersteller. In diesem Dokument verstehen wir *Security by Design* in der weiter gefassten Bedeutung.

2.2 Bedeutung für die Gesellschaft

Software und insbesondere sichere Software sind für die Gesellschaft sowie für das Funktionieren und die Aufrechterhaltung unseres Gesellschaftssystems sehr wichtig. Informationstechnologie bzw. Software haben mittlerweile Einzug in fast alle Bereiche des täglichen Lebens gehalten, in staatlichen Institutionen, Unternehmen oder bei Privatanwendern. Die gesellschaftliche Bedeutung von *Security by Design* wird durch die folgenden Punkte verdeutlicht:

— Wohlstand: Informationstechnologie trägt heute in vielerlei Hinsicht zum Wohlergehen von Bürgerinnen und Bürgern bei. Als wesentlicher Innovations- und Produktivitätstreiber sichert Informationstechnologie Arbeitsplätze und somit die Basis des Wohlstands von vielen Menschen. Die digitale Wirtschaft hat in Deutschland mit ihrer Wertschöpfung bereits deutsche Traditionsbranchen wie Automobilindustrie und Maschinenbau überholt [BMW12b; BMW12a]. Informationstechnologie und das Internet sind zum Rückgrat und Nervensystem unserer Gesellschaft geworden. Auch die Weise, wie Bürgerinnen und Bürger als soziale Wesen interagieren, ist mittlerweile stark durch Informationstechnologie, und somit auch durch Software, geprägt. Bei Kommunikation und anderen Informationsprozessen des Alltags, wie z.B. bei Informationsrecherchen oder Einkäufen, spielt Software heute häufig eine wichtige Rolle. In all diesen Anwendungen und Kontexten ist es für Bürgerinnen und Bürger wichtig, dass sie geschützt sind. Auch wenn es die hierfür verwendeten Technologien im Prinzip bereits seit mehr als 10 Jahren gibt, treten immer wieder Sicherheitslücken zutage, die für viele Bürgerinnen und Bürger ein erhebliches Risiko darstellen, wie z.B. bei der im Jahr 2013 gefundenen Lücke bei Amazon [hei13] oder bei der Playstation-Sicherheitslücke von Sony, bei der Daten von mehr als 70 Millionen Kunden gestohlen werden konnten [hei11]. Immer mehr Bürgerinnen und

Bürger haben Angst vor Sicherheitslücken und Angriffen [hei12b]. *Security by Design* und insbesondere verbesserte Sicherheitsprozesse bei der Herstellung von Anwendungssoftware können die Risiken für die Gesellschaft reduzieren.

— Wirtschaft: Der Nutzen, den die deutsche Wirtschaft aus sicherer Software und *Security by Design* ziehen kann, hat eine gesellschaftliche Dimension. Deutschland ist als Hochlohnland auf die Umsetzung von innovativen Ideen, die Qualität seiner Produkte wie auch effiziente und wirtschaftlich gestaltbare Produktionsprozesse angewiesen. Darüber hinaus sind Unternehmen in einer offenen, vernetzten und digitalisierten Welt darauf angewiesen, ihr Wissen, welches die Basis ihres Wettbewerbsvorteils darstellt, gegen Wettbewerber und potenzielle Angreifer zu schützen. *Security by Design* verschafft Akteuren der Wirtschaft für den Schutz der eigenen Interessen eine verbesserte Ausgangsposition. Damit dies gelingen kann, muss insbesondere die Position des Mittelstands in Deutschland verbessert werden. Mittelständische Hersteller von Software sind heute nicht in der Lage aus eigener Kraft ihre Entwicklungsprozesse zu verbessern. Hierfür sind Vorarbeiten und Unterstützung durch die angewandte Forschung erforderlich.

— eGovernment: Software ist auch aus den staatlichen Institutionen nicht mehr wegzudenken. Das gilt sowohl für die internen Prozesse als auch für die Abwicklung von Vorgängen mit Bürgerinnen und Bürgern. Hierunter gibt es viele Prozesse, bei denen der Bedarf an sicherer Software offensichtlich ist, z.B. bei der Einreichung der elektronischen Steuererklärung beim Finanzamt. Bzgl. der Sicherheit von Behördensoftware existieren offensichtlich erhebliche Risiken [WAZ12]. *Security by Design* hilft, die Sicherheit der Software für das eGoverment zu verbessern.

— Öffentliche Sicherheit: Die öffentliche Sicherheit umfasst die innere und äußere Sicherheit eines Staates. Die in diesem Zusammenhang aktiv werdenden Organe, z.B. Polizei, sind bei der Organisation und Ausführung ihrer Arbeiten oftmals auf moderne Informationstechnologie angewiesen. Da sich die Bedrohungslage wie etwa durch organisierte Kriminalität und internationalen Terrorismus stark verändert hat (z.B. durch den Einsatz von moderner Informationstechnologie), müssen sich die staatlichen Vertreter neuen Aufgaben stellen, um die Risiken für die Gesellschaft reduzieren zu können [RGWS08]. Für die Reduktion von Risiken und Angriffsflächen ist es wichtig, die Sicherheitslücken in der von staatlichen Organen verwendeten Software zu reduzieren.

— Kritische Infrastrukturen: In kritischen Infrastrukturen, wie Stromversorgung, Kommunikationsnetze, Wasserversorgung oder Transport, wird heute in einem erheblichen Umfang Informationstechnologie eingesetzt. In Anbetracht der großen Bedeutung dieser Infrastrukturen für die Gesellschaft ist es sehr wichtig, dass die in den Infrastrukturen verwendete Software sicher gegen Angriffe ist, z.B. bei Manipulationen oder Sabotageakten. Um die Verletzlichkeit dieser Infrastrukturen zu reduzieren, sollte die dort eingesetzte Software sicher sein und deshalb nach dem Paradigma *Security by Design* entwickelt werden. Der Plan

der Bundesregierung, die Betreiber von kritischen Infrastrukturen mittels eines IT-Sicherheitsgesetzes zu mehr IT-Sicherheit zu verpflichten, ist ein Schritt in diese Richtung.

— Demokratie: Dass Informationstechnologie zu Demokratisierungsprozessen beitragen kann, ist spätestens seit dem Arabischen Frühling bekannt (siehe z.B. [Nü12]). Informationstechnologie ist jedoch auch wichtig für die Demokratien in Europa: Sie hilft Prozesse zu organisieren, die in einer Demokratie unerlässlich sind. Mit ihr können beispielsweise Informationen, die für eine informierte Meinungsbildung von Bürgerinnen und Bürgern erforderlich sind, schnell und praktisch ohne Kosten beschafft werden. Weitere wichtige Prozesse wie Debatten und Austausch mit Anderen werden durch Überwindung von Hindernissen wie Zeit und Raum einfach möglich. Informationstechnologie und Vernetzung können Transparenz schaffen und dienen der Evaluation von Politik und staatlichen Organen durch den Souverän. Diese Prozesse verlangen in einer Demokratie Selbstbestimmung und Freiheit der Bürgerinnen und Bürger. In diesem Zusammenhang spielen der Datenschutz und die Sicherheit von Software eine wichtige Rolle. Hierbei hilft *Security by Design*.

2.3 Bedeutung für Anwender von Software

Anwender brauchen Software mit ausgezeichneten Sicherheitseigenschaften. Das gilt sowohl für die professionelle wie auch die private Anwendung. Sicherheitslücken in Software können für Anwender ein hohes Risiko darstellen, insbesondere wenn die Software in Bereichen eingesetzt wird, die kritisch für den Geschäftserfolg sind, mit realen finanziellen Verlusten in Zusammenhang stehen oder die Existenzgrundlage bedrohen können. Um die unerfreulichen Folgen von Sicherheitslücken zu belegen, seien folgende Beispiele genannt:

— Das Technologieunternehmen Nortel wurde unter der Ausnutzung von Sicherheitslücken über Jahre durch eingeschleusten Schadcode ausspioniert und ausgeplündert [Spi12]. Das Problem wurde jahrelang nicht ernst genommen. Die Angreifer hätten „Zugang zu allem gehabt", sagte Brian Shields, der Manager, der seinerzeit die Prüfung bei Nortel geleitet hatte [hei12a]. Wenn es Angreifern gelingt, einen Schadcode zu installieren, dann gibt es vielfältige Möglichkeiten für Angriffe. Ist ein Angreifer so weit gekommen, kann man zur Abwehr mit *Security by Design* oft nur noch sehr wenig ausrichten. *Security by Design* kann jedoch dabei helfen, dass die Installation von Schadcode für Angreifer sehr viel schwieriger wird.

— Die New York Times wurde ebenfalls ausspioniert, indem wahrscheinlich über E-Mails Schadcode auf die Computer von Mitarbeitern verteilt wurde [Spi13]. Man nimmt an, dass die Angriffe darauf abgezielt haben, die Identität von solchen Informanten in Erfahrung zu bringen, die mit Journalisten der Zeitung zusammengearbeitet haben.

– Mit der auf Online Banking ausgelegten Schadsoftware Eurograbber haben Hacker im Jahr 2012 bei mehr als 30.000 Bankkunden insgesamt mehr als 36 Millionen Euro erbeutet [DMN12].

Verwendet man das Paradigma *Security by Design* bei der Softwareentwicklung, können viele Sicherheitslücken vermieden werden, wodurch sich die Risiken für Anwender reduzieren. Neben den direkten Verlusten können für Anwender weitere Probleme aus Sicherheitslücken resultieren. Hier sind beispielsweise Reputationsverluste zu nennen. In Unternehmen stellt sich darüber hinaus die Frage der Haftung, z.B. gegenüber Kunden oder Partnern, die durch Sicherheitslücken beim Anwender einen Nachteil erleiden. Es ist ebenfalls möglich, dass hochrangige Entscheider persönlich haften müssen, wenn die Anwendung von Software mit Sicherheitslücken als fahrlässig eingeschätzt wird.

Werden durch *Security by Design* Sicherheitslücken reduziert, dann können auf Anwenderseite Aufwände für Wartungsprozesse reduziert werden, da deutlich seltener Sicherheitspatches organisiert, getestet sowie ggf. verteilt und installiert werden müssen. Dadurch vermindern sich die Kosten einer Software im Betrieb (*Cost of Ownership*). Darüber hinaus ist nicht in jedem Fall davon auszugehen, dass jeder Anwender über das Fachwissen verfügt, um sein Risiko durch bestimmte Sicherheitslücken angemessen einschätzen zu können. Eine Verbesserung der Ausgangssituation für Anwender mittels *Security by Design* hat auch eine psychologische Komponente, da sich bestehende Ängste gegenüber der Technik abbauen bzw. reduzieren lassen und ein vertrauensvoller Umgang mit Technik gefördert wird.

Insbesondere Anwender, für die Software einen hohen Anteil ihres Budgets ausmacht, beginnen zunehmend damit, bei Herstellern die angewendeten Sicherheitsprozesse im Rahmen von *Security by Design* zu hinterfragen und von diesen zu verlangen, ihre Maßnahmen für Software mit verbesserten Sicherheitseigenschaften darzulegen. Die bloße Existenz von solchen Sicherheitsprozessen kann für Anwender ein wichtiges Kriterium bei der Entscheidung zum Erwerb einer Software sein. Jedoch auch für Anwender mit wenig Marktmacht bis hin zu Privatpersonen kann die Information, dass Herstellungsprozesse von Produkten dem Paradigma *Security by Design* folgen, interessant sein. Insbesondere für solche Anwender, die weniger mit Fragestellungen der IT-Sicherheit vertraut sind, ist eine solche Information hilfreich. Die Umstellung von Produktionsprozessen war auch in anderen Bereichen ein Markterfolg, wie etwa bei Bio-Lebensmitteln.

2.4 Bedeutung für Hersteller von Software

Die Einführung von *Security by Design* kann für Unternehmen eine existenzielle Tragweite haben. Es gibt eine Reihe von Gründen, die für eine Einführung dieses Paradigmas in heutige Produktionsprozesse sprechen. Zu diesen gehören:

– Reduktion der Entwicklungskosten von sicherer Software: Dies lässt sich verdeutlichen, wenn man die bestehende Softwareentwicklung und Sicherheitsprozesse

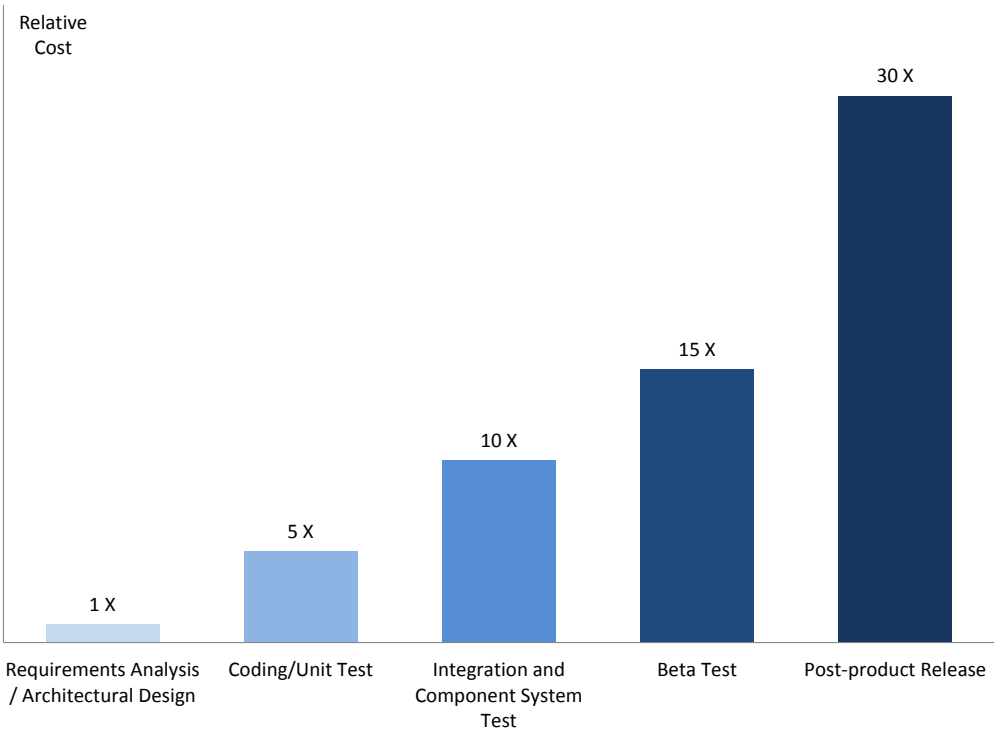


Abbildung 2: Die Entwicklung der Kosten zur Behebung von Fehlern in verschiedenen Phasen im Softwarelebenszyklus relativ dargestellt gemäß einer Untersuchung des NIST (Quelle: [Tas02]).

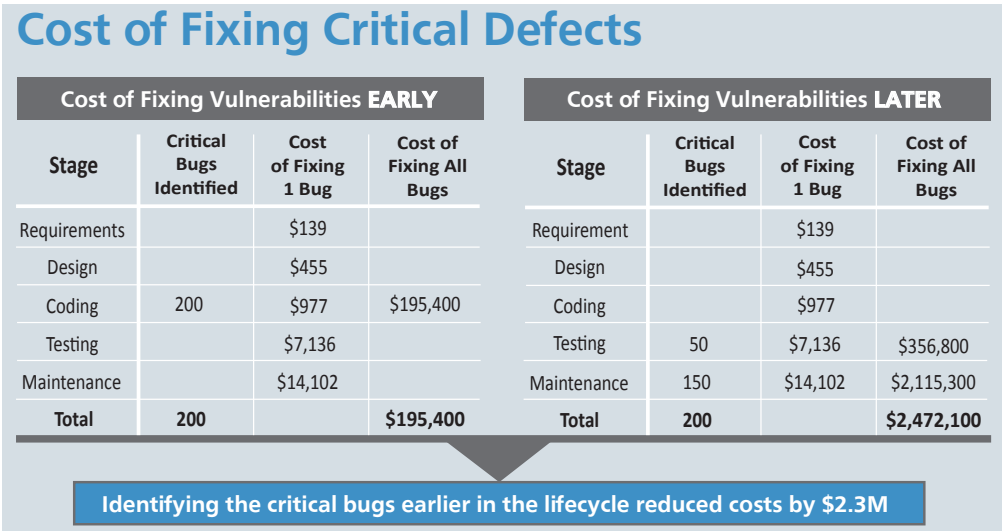


Abbildung 3: Die unterschiedlichen Kosten bei der Behebung von kritischen Fehlern in verschiedenen Phasen (Quelle: [VK11]).

betrachtet. Sicherheit hat in der Vergangenheit oftmals keine oder nur eine geringe Rolle gespielt. Sicherheitsexperten wurden oftmals erst dann eingebunden, wenn ein Produkt schon ziemlich weit entwickelt war. Haben die Experten dann eine Lücke entdeckt, war es aufgrund von gewählten Architektur- und Entwurfs-

entscheidungen nicht immer möglich, diese in einfacher Weise zu schließen. Zur Beseitigung von solchen Lücken mussten, sofern dies überhaupt möglich war, dann teilweise größere Veränderungen an der jeweiligen Software vorgenommen werden. Dadurch wurden Arbeitsergebnisse, für welche im ersten Anlauf Investitionen aufgebracht wurden, gerade wieder vernichtet. Solche Situationen können vermieden werden, wenn bereits ab der Designphase einer Software Sicherheitsanforderungen berücksichtigt werden. Je früher Korrekturen vorgenommen werden können, desto größer sind die Einsparungsmöglichkeiten im Vergleich zur traditionellen Vorgehensweise. Diese Erkenntnis ist keineswegs neu. Bereits vor mehr als 10 Jahren hat das NIST die Kosten bei der Beseitigung von Fehlern in verschiedenen Phasen miteinander verglichen [Tas02]. Ein Ergebnis aus dieser Untersuchung wird in Abbildung 2 dargestellt. Dort verändern sich die durchschnittlichen Kosten zwischen einer frühen und späten Beseitigung von Fehlern um den Faktor 30. Es ist anzunehmen, dass dieses Missverhältnis bei der ausschließlichen Betrachtung von Sicherheitslücken bei einem höheren Faktor liegt. Diese Einschätzung wird bestätigt durch die Daten in [VK11] (siehe auch Abbildung 3): Dort belaufen sich die mittleren Kosten zur Beseitigung kritischer Fehler zwischen den Phasen *Requirements* und *Maintainance* auf einen Faktor, der größer als 100 ist.

— Verbesserung der Sicherheit von Software: Durch die systematische Anwendung von Sicherheitsprozessen bekommt Sicherheit im Entwicklungsprozess im Vergleich zur Vergangenheit eine größere Bedeutung. Sicherheitsfragen werden dadurch über dem kompletten Lebenszyklus berücksichtigt und analysiert. Dies führt dazu, dass die Sicherheit von Software verbessert wird. Ein Beispiel hierfür ist Microsoft mit dem *SDL* [Mic13b]. Abbildung 4 zeigt am Beispiel von zwei Microsoft-Produkten die Verbesserung von deren Sicherheitseigenschaften nach der Einführung von *SDL*. Ein weiteres Beispiel ist die Umsetzung des *Adobe Secure Product Lifecycle* (SPLC) [Ado13]: Sie führte zu einer erheblich besseren Qualität und höheren Resistenz gegen Angriffe bei den Produkten *Adobe Reader* und *Adobe Flash*.

— Reduktion der Kosten für Bereitstellung von Patches: Mit der Verbesserung von Sicherheitseigenschaften reduziert sich die Anzahl der Sicherheitslücken. In unmittelbarer Konsequenz verringert sich ebenfalls die Häufigkeit von Sicherheitsupdates oder Patches. In einer weiteren Folge reduzieren sich dadurch für die Hersteller die Kosten, welche in der Vergangenheit durch die Entwicklung, Testen, Bereitstellung und Support im Zusammenhang mit Patches entstanden sind.

— Pflege der Herstellerreputation: Durch die Verbesserung der Sicherheitseigenschaften der eigenen Produkte erhält ein Hersteller seltener negative Schlagzeilen in den Medien wegen Sicherheitslücken. Die Umsetzung des Paradigmas *Security by Design* lässt sich von Herstellern im positiven Sinn nutzen. Investitionen zur

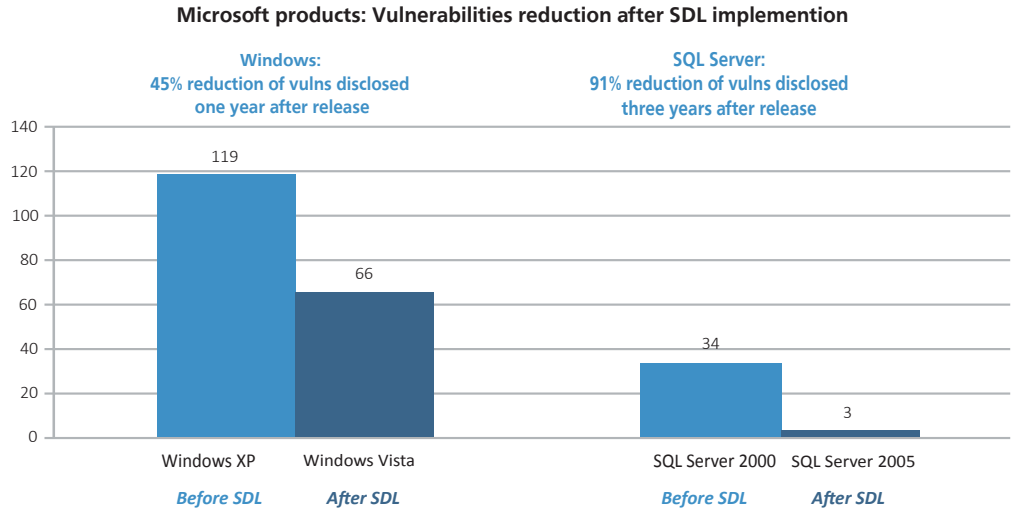


Abbildung 4: Die Auswirkungen von SDL auf die Sicherheit von Anwendungssoftware (Quelle: [Mic13b]).

Verbesserung der Produktionsprozesse zum Wohle der Verbraucher werden von den Kunden sehr geschätzt.

— Keine Beschränkung von Absatzmärkten: Produktionsprozesse, die sich nicht am Stand der Kunst orientieren, können für Kunden ein Ausschlusskriterium bei der Entscheidung für einen Hersteller oder für ein Produkt sein. Vor diesem Hintergrund ist es für Hersteller wichtig *Security by Design* umzusetzen, um dadurch die eigenen Absatzmärkte nicht zu beschränken.

— Verbesserung der Wettbewerbsfähigkeit: Die Entscheidung zur Umsetzung von *Security by Design* in den eigenen Produktionsprozessen zum richtigen Zeitpunkt verbessert die Wettbewerbsfähigkeit. Eine solche Verbesserung kann jedoch nur dann erfolgen, wenn die Umsetzung im Vergleich zu den wichtigsten Wettbewerbern nicht zu spät erfolgt, da dann Marktanteile verloren gehen können. Das Wiedererlangen dieser Marktanteile kann sehr schwierig sein, da Kunden nicht sofort zurückgewonnen werden können, wenn sie sich erst einmal für ein Produkt eines Wettbewerbers entschieden haben.

Für einen Hersteller bedeutet die Umstellung der bisherigen Entwicklungsprozesse auf *Security by Design* eine strategische Entscheidung mit weitreichenden mittel- bis langfristigen Konsequenzen. Diese Entscheidung muss unternehmensweit umgesetzt werden und benötigt in der Umsetzungsphase gewisse Investitionen, die sich nach einem Einspielen der Prozesse mehr als amortisieren werden.

Viele Hersteller sind mittelständisch geprägt und können die Umstellung ihrer Softwareentwicklungsprozesse auf *Security by Design* nicht aus eigener Kraft bewältigen. Lediglich Weltkonzerne von einer Größe wie z.B. Microsoft oder IBM können solche Transformationsprozesse in ihrer Produktion alleine bewältigen. Für weniger große Hersteller ist es wichtig, dass sie bei der Einführung von Ansätzen für *Securi-*

ty by Design unterstützt werden. Dadurch können auch kleinere Hersteller in ihren jeweiligen Nischen im Vergleich zu den großen Herstellern konkurrenzfähig bleiben.

Für die praktische Umsetzbarkeit von *Security by Design* ist es unbedingt erforderlich, dass die Forschung die heute etablierten Eigenheiten und Besonderheiten von Softwareproduktionsprozessen berücksichtigt. Produktionsprozesse können sehr komplex sein und sind durch viele Nebenbedingungen geprägt, wie etwa:

- Zeitdruck
- Wirtschaftlichkeit
- Innovationsdruck
- Compliance-Vorgaben für bestimmten Branchen oder Länder
- Produktlinien
- Integration von Zulieferer-Code
- Integration von Open-Source-Komponenten
- Verwendung von Legacy-Code
- Reduktion menschlicher Fehlereinflüsse
- Mess- und Steuerbarkeit der Maßnahmen im Rahmen von *Security by Design*

Die Einführung von neuen Methoden und Sicherheitsprozessen bei der Softwareherstellung und Integration muss kontrollierbar und steuerbar sein. So müssen die Effekte einzelner Maßnahmen bei der Transformation der Herstellungsprozesse möglichst objektiv messbar sein, um bewerten zu können, welche Maßnahmen nutzenbringend und auch wirtschaftlich umsetzbar sind und bei welchen weiterer Modifikationsbedarf besteht. Deshalb verlangt praktisch jede Neuerung im Rahmen von *Security by Design* auf der Produktionsebene eine korrespondierende Lösung auf der Managementebene, welche eine Kontrollierbarkeit und Steuerbarkeit ermöglicht. Die Lösung auf der Managementebene muss die relevanten Aspekte der Sicherheit mit Informationen, die mit den oben genannten Nebenbedingungen in Zusammenhang stehen, zusammen führen, auswerten und zur Entscheidungsunterstützung darstellen.

Für Hersteller bestehen neben dem Ansatz auf Basis von *Security by Design* auch andere Möglichkeiten, die Sicherheit ihrer Produktionsprozesse und Produkte zu verbessern, wie etwa durch Zertifizierungen z.B. mit *Common Criteria*. Auch wenn diese Möglichkeit seit vielen Jahren besteht, wird sie von Herstellern meistens aus verschiedenen Gründen gemieden. Zertifizierung ist teuer, zeitaufwändig und muss für jede noch so geringe Modifikation und Weiterentwicklung eines Produktes neu durchlaufen werden. Zertifizierung wird heute meist nur bei Nischenprodukten mit besonderen Sicherheitsanforderungen angewendet.

3. SOFTWARESICHERHEIT DURCH AUTOMATISIERUNG UND REDUKTION MENSCHLICHER FEHLEREINFLÜSSE

Die IBM-X-Force-Berichte [IBM12], die BSI-Lageberichte [BSI13], die jährlichen Coverity Scan Reports [Cov13] und die von SANS als „gefährlich" eingestuften häufigen Softwarefehler [Chr11] zeigen in ihrer Analyse und Bewertung übereinstimmend, dass über Jahre hinweg überwiegend immer wieder dieselben Typen von Softwareschwachstellen auftreten. Die Fehler bzw. die daraus resultierenden Schwachstellen wären also vermeidbar gewesen. Beispielsweise stellt Gary McGraw in seinem Standardwerk über sichere Softwareentwicklung eine komplette Taxonomie für solche bekannten, potenziell sicherheitskritischen Fehler für die Phase der Programmierung (*Coding Errors*) auf (vergleiche Kapitel 12 in [McG06]). Es handelt sich in der Mehrzahl um Fehler durch den Faktor *Mensch*. Um zu verstehen, wie diese Fehler durch den Faktor *Mensch* entstehen, ist ein Blick auf die Bedingungen hilfreich, wie Software, insbesondere Anwendungssoftware, heute entwickelt wird. Die Entwicklung von Software wird heute in vielen Fällen noch fast ausschließlich durch die Funktionalität der Software getrieben. Sicherheit spielt nur eine untergeordnete Rolle, wenn überhaupt. Die Entwickler sind Experten in den jeweiligen Anwendungsdomänen der Software; Fragen der Sicherheit sind für Entwickler nicht hoch priorisiert. Der sich auf die Entwicklung neuer Funktionen auswirkende Innovationsdruck gibt Entwicklern auch wenig Freiräume, sich mit zusätzlichen Fragen der Sicherheit zu beschäftigen. Wenn tatsächlich Sicherheitsrichtlinien für Softwareentwicklung existieren wie etwa Programmierrichtlinien und -leitfäden, dann wurden diese oft nur unzureichend umgesetzt. Stattdessen wurden Freiheitsgrade der Programmiersprachen oft gedankenlos genutzt, wenn damit die gewünschte Funktion erzielt werden kann. Wenn Sicherheitsaspekte systematisch berücksichtigt wurden, dann wurde Sicherheit oft eher externalisiert, z.B. indem Sicherheitsexperten spezielle Sicherheitskomponenten entwickelt haben wie etwa Wrapper, Firewalls oder Virenscanner. Bereits existierende Hilfsmittel zum Aufspüren von Schwachstellen in Software wurden von Entwicklern oftmals nicht verwendet.

Sicherheitslücken, die bisher durch den Faktor *Mensch* entstanden sind, wird man in der Praxis wahrscheinlich leider nicht effizient und wirksam dadurch ändern können, indem man die Ursachen bekämpft, z.B. durch Einwirken auf Entwickler, ihre Arbeitsmethoden zu ändern. Es ist davon auszugehen, dass menschliche Fehler, die auf Unwissenheit, Leichtsinn oder Flüchtigkeit zurückgehen, weiter in fast gleichem Ausmaß gemacht werden. Der Gedanke, dass ein Hersteller die große Menge von Entwicklern innerhalb kurzer Zeit ändern kann, ist nicht realistisch. Eine Möglichkeit zur Verbesserung der Lage besteht darin, den Entwicklern technische Lösungen an die Seite zu stellen, die sie davor bewahren, entsprechende Fehler zu begehen.

Diese von Menschen verursachten und mittlerweile gut bekannten Sicherheitsfehler könnten durch Assistenzsysteme bei der Entwicklung [Zel07; BBMM10] und durch sicherheitsorientierte Rahmenbedingungen größtenteils vermieden werden. Diese As-

sistenzsysteme könnten, wenn in Entwicklungsumgebungen integriert, automatisiert die Fehler erkennen, die zu Sicherheitsproblemen führen und Alternativen zur Lösung vorschlagen oder durch technologische Weiterentwicklung dahin führen, dass bestimmte Fehler gar nicht mehr gemacht werden können. Die dann noch verbliebenen Schwachstellen könnten in ihrer Mehrzahl durch halb- bis vollautomatische Unterstützung vor dem Ausrollen der Software entdeckt werden. Diese Punkte werden zur Vision zusammengefasst:

> *Der Softwareentwicklungsprozess der Zukunft wird durch Programmiersprachen und Tools geprägt sein, die konsequent sicherheitsorientiert sind und nahtlos integriert werden können. Hierdurch werden sicherheitsrelevante Fehler entsprechend dem jeweils aktuellen Stand der Forschung verhindert und Schwachstellen systematisch und weitestgehend automatisiert gefunden.*

Diese Evolution des Softwareentwicklungsprozesses verbessert zudem die Wirtschaftlichkeit der Softwareentwicklung.

3.1 Herausforderung: Sicherheitsorientierte Programmiersprachen und -konstrukte sowie Managed Code

Pufferüberläufe (*Buffer Overflows*) zeigen eindrucksvoll das Problem der unzureichenden Sicherheitsorientierung von Programmiersprachen. Pufferüberläufe werden als Sicherheitslücken seit über zwei Jahrzehnten ausgenutzt und sie gehören seitdem ununterbrochen zu den 25 gefährlichsten Schwachstellen [Chr11]. Davon betroffen ist grundsätzlich jeder Code, der in Programmiersprachen geschrieben wird, die die Zugriffe auf Speicherbereiche nicht automatisch überwachen — prominente Beispiele für diese Programmiersprachen sind C und C++.

Bei korrekter Implementierung der *Java Virtual Machine* (JVM) können Pufferüberläufe bei Java nicht auftreten, da die JVM die Einhaltung der Speicherbereiche kontrolliert. Der Aufruf von nativem Code ist allerdings weiterhin aus mehreren Java-Technologien heraus möglich, so dass Pufferüberläufe „durch die Hintertür" auch bei Java Programmen möglich sind.

Die verwaltete Maschinensprache (*Managed Code*) des .NET-Rahmenwerks von Microsoft wurde wie die JVM in Hinblick auf Sicherheit entworfen: Bytecode, der in der *Common Language Runtime* (CLR) ausgeführt wird, verhindert Schwachstellen wie Pufferüberlauf und Rechteausweitung (*Privilege Escalation*). Die Programmiersprache C# des .NET-Rahmenwerks vermeidet die schwachstelleninduzierende Zeigerarithmetik leider ebenfalls nicht konsequent: Mit dem Schlüsselwort *unsafe* ist Zeigerarithmetik weiterhin möglich.

Aktuelle Exploits der *Java API* haben die Aufmerksamkeit verstärkt auf das Java-Sicherheitsmodell gelenkt: Am 17. April 2013 wurde mit dem Update 21 von Java 7

ein Patch Release veröffentlicht, das 42 Patches gegen Sicherheitsfehler liefert, von denen mehrere den Höchstwert 10 im *Common Vulnerability Scoring System* erreichen. Dies wiegt umso schwerer, da diese Verwundbarkeiten für verschiedene Betriebssysteme existieren – durch die Plattformunabhängigkeit von Java. Diese Attacken nutzen Lücken in der Sicherung kritischer Ressourcen in der *Java API* wie z.B. *Class Loading* oder *Reflection*, die unwissend von Entwicklern bei der Erweiterung der Plattform eingeführt wurden. Da das Java-Sicherheitsmodell die aktive Einschränkung von Berechtigungen vorsieht, gehen diese Lücken unbemerkt in neue Releases von Java ein. Durch ein anderes Modell, das Sicherheit von Anbeginn vorsieht, werden diese Lücken unmöglich oder zumindest erkennbar.

Typsysteme könnten viel breiter als heute eingesetzt werden: Typsysteme prüfen und schützen die Semantik und stellen somit einen Ansatz dar, der IT-Sicherheit durch *Safety* erreicht. Die Sicherheitsmodelle von Managed-Code-Sprachen wie Java sind ohne ein Typsystem nicht denkbar. So stellt beispielsweise das Java-Typsystem sicher, dass Zeigerarithmetik selbst durch Typkonvertierungen nicht stattfinden kann. Andere Teile der Sicherheitsarchitektur verlassen sich auf diese Invarianten, die das Typsystem garantiert. Typsysteme lassen sich beliebig mächtig gestalten und es wurden einige Ansätze entwickelt, die teilweise weit über Typsysteme wie das von Java hinausgehen: Ein komplettes Sicherheitstypsystem wurde für *Bali*, einer Variante von Java, vorgestellt [ON98]. Mit [Loc12] liegt ein typsicheres Modell für nebenläufige Java-Programme vor. Ein erster Ansatz für typsichere Produktlinien wurde in [AKGL10] vorgeschlagen. Für Webanwendungen gibt es eine WSDL-Erweiterung in Richtung Typsysteme [LPT06]. Für die WSDL-Komposition wird in [HHH12] ein Ansatz mit Kontrakten vorgestellt. Eine inhärente Limitierung von Typsystemen ist, dass sie in der Regel kontext-insensitiv gestaltet werden müssen. Sicherheitstypsysteme assoziieren Informationen wie *secret* oder *public* in der Regel fest mit Programmteilen wie einzelnen Anweisungen oder Variablen. Während der Ausführung eines Programms können diese Teile jedoch mehrere Werte verarbeiten, die abhängig vom Ausführungskontext sowohl *secret* als auch *public* sein können. Komplexere Typsysteme sind daher oft zu grobgranular, um realistisches Programmverhalten abbilden zu können.

Programmiersprachen in Richtung IT-Sicherheitsorientiertung umzubauen scheint letztlich der konsequenteste Weg. Ein erster Ansatz liegt mit JOE-E [MWC10] für Java vor.

Als Forschungsherausforderung wird aufzuzeigen sein, wie ein Migrationspfad in Richtung sicherheitsorientierte Programmiersprachen aussieht und wie konsequent er beschritten werden kann [BHLM13], insbesondere so, dass er verträglich mit der großen Menge existierender Software ist.

3.2 Herausforderung: Risiko-, Bedrohungs- und Reifegradmodelle

Durch Risiko-, Bedrohungs- und Reifegradmodellierung werden Risiken überhaupt erst erfass-, beschreib- und handhabbar. Leider gibt es keine allgemein anerkannte Herangehensweise und kein allgemein akzeptiertes Tool für die Risiko-, Bedrohungs- und Reifegradmodellierung zur Entwicklung sicherer Softwareprodukte, die nicht für den Hochsicherheitsbereich bestimmt sind.

Die nachfolgende Auflistung von Tools zur Risiko- und Bedrohungsmodellierung zeigt, dass die Hersteller von unterschiedlichen Grundannahmen und Ausgangspunkten ausgehen:

- *TRIKE Threat Modeling Methodology* [SLE05]: TRIKE ist eine Heuristik zur Bedrohungsmodellierung und kann für Systeme und Software eingesetzt werden. TRIKE bindet alle Parteien in die Einschätzung und Zustimmung von Risiken ein.
- *CORAS Model-based Method for Security Risk Analysis* [LSS11]: CORAS fokussiert sich auf die Risikoanalyse und ist allgemeiner anwendbar als auf Software(entwicklung). Das Rahmenwerk bietet eine toolgestützte Methodik zur modellbasierten Risikoanalyse von sicherheitskritischen Systemen.
- *Operationally Critical Threat, Asset, and Vulnerability Evaluation for operational risk, not technical risk (OCTAVE)*: OCTAVE behandelt nur operative Risiken, keine technischen.
- *CCTA Risk Analysis and Management Method (CRAMM)*: Die von der *Central Computing and Telecommunications Agency* (CCTA) entwickelte Methodik ist eng an die Verwendung eines kommerziellen Tools gebunden und führt eine Bedrohungs- und Schwachstellenanalyse sowie eine Risikobewertung durch, um daraus entsprechende Maßnahmen abzuleiten. Da die Durchführung von CRAMM mit signifikantem Aufwand verbunden ist, wird sie als Methode der Wahl eher für kritische Systeme angesehen.
- *AZ/NZS 4360*: Mit AZ/NZS 4360 liegt ein generischer Standard zum Dokumentieren und Managen von Risiken vor. AZ/NZS 4360 hat sieben Schritte: Risiko-Strategie, Risiko-Identifikation, Risiko-Analyse, Risiko-Gewichtung, Risiko-Handhabung, Risiko-Dokumentation und -Kommunikation, Risiko-Kontrolle und -Überwachung.

Die folgenden drei Rahmenwerke für Aussagen über das erreichte Sicherheitsniveau starten ebenfalls an verschiedenen Punkten:

- Die vom *Software Engineering Institute* (SEI) der *Carnegie Mellon University* (CMU) vorgeschlagene Methodik *Integrated Measurement and Analysis Framework for Software Security* [AAS10] kann auf die Phasen des Softwareentwicklungsprozesses angewendet werden.
- Die Publikation [AAS12] gibt einen Überblick über verschiedene Möglichkeiten zur Messung des Sicherheitsniveaus.

— *CVSS Common Vulnerability Scoring System* bestimmt *ex post* den Schweregrad einer Schwachstelle als Wert zwischen 0 und 10 mittels mehrerer Kategorien.

Zur Bestimmung des Reifegrades der Governance bei der Entwicklung sicherer Software gibt es mindestens ein Analysetool: Das *Open Software Assurance Maturity Model* (OpenSAMM [Ope13]) ist ein Modell für die Bestimmung des Reifegrades einer Organisation in Bezug auf die Prozesse für sichere Software, bezieht sich also auf organisatorische Kenndaten.

Hersteller bieten zwar verschiedene Tools für die Risiko-, Bedrohungs- und Reifegradmodellierung an, es gibt allerdings mehrere Aspekte mit Klärungsbedarf:

— Wie kann man Risiko-, Bedrohungs- und Reifegradmodellierung durchführen, so dass sie intersubjektiv nachvollziehbare Ergebnisse liefern?
— Wie kann erreicht werden, dass objektive Ansätze zur Risiko-, Bedrohungs- und Reifegradmodellierung verstärkt eingesetzt werden?
— Wie interagieren die Modelle dieser Herausforderung mit den Entwicklungsmodellen der nächsten Herausforderung? Wie kann eine nahtlose Integration von Risiko-, Bedrohungs- und Reifegradmodellen mit Entwicklungsmodellen für den sicheren Softwarelebenszyklus erreicht werden?

3.3 Herausforderung: Entwicklungsmodelle für sicheren Softwarelebenszyklus

Entwicklungsmodelle erhöhen, wenn sie rigoros angewendet werden, das Sicherheitsniveau von Software von Anfang an und über die gesamte Lebenszeit von Software [Mic13b]. Zur Umsetzung dieser Rahmenwerke ist es essenziell, dass sie ohne Verzögerung der Entwicklungszeiten schrittweise eingeführt werden und so ineinander greifen, dass sie für die Akteuere wie aus einem Guss integriert erscheinen und nicht — wie bisher — siloartig nebeneinander stehen. Leider weist kein Rahmenwerk vollständig und nahtlos integrierte Assistenzsysteme auf und die korrekte und nachhaltige Anwendung von sicherheitsorientierten Tools ist weder belegbar noch überprüfbar. Hier genannt sind Rahmenwerke mit hohem Reifegrad:

— *Microsoft Security Development Lifecycle (SDL)* [HL06]: SDL hat nach Angaben von Microsoft zu einer messbaren Reduktion der sicherheitsrelevanten Verwundbarkeiten geführt [LSP+11]. Für jeden SDL-Schritt gibt es unterstützende Tools [Mic13a]. Soweit bekannt muss jedoch kein Tool verpflichtend für einen Schritt angewendet werden, die Toolanwendung kann nicht halb- oder vollautomatisch überprüft werden und nur ein Teil der Tools sind in Entwicklungsumgebungen integriert.
— *Software Assurance Forum for Excellence in Code (SAFECode):* Das Konsortium SAFECode [SAF07] startete mit dem Ziel Prozesse zur Entwicklung sicherer Software industrieweit zu verbreiten. Mitglieder sind beispielsweise Adobe, CA Technologies, EMC Corporation, Intel Corporation, Microsoft Corp., SAP

AG, Siemens AG und Symantec. Die Empfehlungen sind durchweg zu begrüßen. Offen bleibt, wie die Detaillierung, Durchsetzung und der Nachweis der Durchführung der Empfehlungen erfolgt und wie die Automatisierung der Softwaresicherheit mittels Tools angegangen wird.

Die Integration folgender Forschungsansätze als Tools würde signifikante Lücken bei der Herstellung sicherer Software schließen. Diese Ansätze stellen attraktive Ausgangspunkte für Assistenztools entsprechend der obigen Beschreibung dar:

— *Programmverstehen:* Die an den Universitäten Stuttgart und Bremen laufenden Arbeiten zum Programmverstehen können gerade auch im Kontext sicherer Softwareentwicklung einen vielversprechenden Ansatz bieten. Programmverhaltens- und Architekturanalysen sollten Bestandteil eines sicheren Entwicklungsprozesses sein, eine mögliche technische Lösung hierfür bildet das Projekt Bauhaus [Bau13]. Die auf diese Weise möglichen sicherheitstechnische Analysen auf Architekturebene werden von Bunke und Sohr in [BS11] beschrieben.
— *Safety im Softwareentwicklungsprozess:* [RBG12]: SAFE bietet ein hierarchisches Programmiermodell, das zur sicheren Erweiterbarkeit (bis hin zu sicherem personalisiertem Code einzelner Anwender/innen) von Webanwendungen beiträgt.

Die genannten Entwicklungsmodelle und Forschungsansätze sind zweifellos nützlich zur Erhöhung des Sicherheitsniveaus von Software von Anfang an. Für deren Weiterentwicklung müssen folgende Fragen beantwortet werden:

— Wie können Assistenzsysteme zur Schwachstellenvermeidung im Softwareerstellungsprozess rigoros und nahtlos in Entwicklungsumgebungen eingebettet werden, so dass bestehende Lücken in Lebenszyklusansätzen geschlossen werden? Solche Werkzeuge zur vollautomatischen Schwachstellenerkennung bei der Softwareerstellung könnten einen Großteil bekannter Schwachstellen verhindern, indem sie beispielsweise bei einer automatisch erkannten Schwachstelle die Übertragung einer Version in das Repository eines Versionenkontrollsystems erst dann zulassen, nachdem die Schwachstelle eliminiert wurde.
— Wie können Übergänge zwischen Phasen im Entwicklungszyklus gestaltet werden, so dass zugesichert werden kann, dass dezidiert aufgelistete Schwachstellen nicht (mehr) vorhanden sind. Solche Zusicherungen müssten idealerweise vollautomatisch oder hilfsweise halbautomatisch überprüft werden können.

3.4 Herausforderung: Verifikation und Testen

Bei jeder Software muss letztendlich geprüft werden, ob sie ihre Anforderungen erfüllt – in unserem Fall, ob sie *sicher* ist, also gegebenen Sicherheitsanforderungen gerecht wird. Angesichts der Komplexität der Software (und der zu prüfenden Anforderungen!) gilt es auch hier, die Prüfung weitestgehend zu automatisieren.

Zur Prüfung stehen im Wesentlichen drei Verfahren zur Wahl, die jeweils Stärken und Schwächen haben. *Statische Codeanalyse* inspiziert den Programmcode, um

alle möglichen Ausführungen eines Programms zu betrachten. Das gewünschte Ergebnis ist, dass sämtliche möglichen Ausführungen die (Sicherheits-)Anforderungen erfüllen; das Programm entspricht dann beweisbar den Anforderungen. Ein solcher Beweis ist offensichtlich außerordentlich wertvoll. Interessanterweise wandelt sich im Bereich der IT-Sicherheit ein viel zitierter Nachteil statischer Analysen zum Vorteil. Statische Codeanalysen abstrahieren von den Benutzereingaben eines Programms. In anderen Anwendungsbereichen führt dieser Mangel an Information über realistische Benutzereingaben oft zu ungenauen Analyseergebnissen. In der IT-Sicherheit muss man jedoch von einem böswilligen Nutzer (dem Angreifer) ausgehen, für den somit sämtliche möglichen Eingaben realistisch sind. Statische Codeanalysen berücksichtigen automatisch solche Eingaben ebenso wie alle anderen.

Leider hat die statische Analyse sowohl theoretische Schranken als auch praktische Probleme. Das sogenannte *Halteproblem* besagt, dass es kein allgemeines Verfahren geben kann, das für ein beliebiges Programm dessen Verhalten vorhersagen kann. Daher müssen statische Codeanalysen mit *Annäherungen* arbeiten. Je nach Design der Analyse können diese entweder zu Fehlalarmen führen oder dazu, dass tatsächlich existierende Probleme übersehen werden. Eine Analyse zu konstruieren, die für beliebige Programme Schwachstellen hundertprozentig trennscharf erkennt, ist leider nicht möglich.

Ein weiteres Problem in der Praxis ist, dass die statische Codeanalyse den gesamten Programmtext kennen und analysieren können muss, um gesicherte Aussagen treffen zu können. Der Einsatz verschiedener Programmiersprachen, verteilter oder nicht zugänglicher Programmcode stellen die statische Codeanalyse vor große Herausforderungen. Ein Technologie-Stack wie etwa Web-Anwendungen (z.B. JavaScript im Browser, PHP-SQL-C-Assembler im Server) verschließt sich in der Praxis aktuellen Analysetechnologien. Statische Codeanalyse ist daher heute in der Praxis meist auf einzelne Teilsysteme beschränkt, deren sicheres Funktionieren aber eine wichtige Grundlage für die Sicherheit des Gesamtsystems bildet. Für solche Systeme haben Codeanalysen jedoch mittlerweile einen hohen Reifegrad erreicht. So wurden unlängst Systeme zur statischen Codeanalyse, präziser zur *Information Flow Control*, verfügbar gemacht und erfolgreich auf mittelgroßen bis großen Programmen durchgeführt, allen voran die Werkzeuge JOANA [HS09] und FlowDroid [FAR+13].

Die zweite Technik, das *Testen*, kommt mit anderen Anforderungen daher. Zum Testen benötigt man die Möglichkeit, das Programm auszuführen, um das Ergebnis mit den Anforderungen zu vergleichen. Bei vielen Testansätzen ist es hierbei wenig relevant, welche Programmiersprachen für die zu testende Software verwendet wurden. Unter der Annahme, dass das Erkennen von Fehlern zuverlässig möglich ist, verursacht auch Testen keine Fehlalarme (erfüllt das Ergebnis die Anforderungen nicht, hat man ein Problem). Das Problem des Testens ist, dass nur eine *endliche* Menge von Ausführungen geprüft werden kann, die Menge der möglichen Ausführungen aber *unendlich* groß ist, und somit trotz bestem Testens die nächste neue Ausführung ein Problem aufwirft.

In der Praxis kommt es daher darauf an, möglichst viele Verhaltensweisen des Programms abzutesten; hierfür kommen zunehmend *Testgeneratoren* zum Einsatz, die Eingabedaten für den Test erzeugen. Solche Generatoren können zufällige Eingaben erzeugen (*Fuzzing*), aber auch spezifisch nach Sicherheitslücken suchen. Moderne Testgeneratoren suchen gezielt nach Schwachstellen, die durch statische Codeanalyse als möglich bestimmt wurden (*DART* / Microsoft), oder rekombinieren fehlerverursachende Eingaben (*LangFuzz* / Mozilla), um automatisch Hunderte von Sicherheitslücken zu bestimmen. Eine Garantie für zukünftige Ausführungen kann jedoch keines dieser Systeme bieten.

Die dritte Alternative besteht darin, den Test in die tatsächliche Ausführung zu verlagern und so das Ergebnis bei jeder Ausführung – also auch in der Produktion! – zu prüfen. Hiermit können Fehlergebnisse per Konstruktion ausgeschlossen werden. Die Nachteile dieser *Laufzeit-Verifikation* sind der erhöhte Rechenaufwand zur Laufzeit und die Tatsache, dass Fehlersituationen erst zur Ausführungszeit erkannt und abgehandelt werden können. Zu dieser Zeit ist oft nur wenig Kontextinformation vorhanden, was es schwer macht, eine sinnvolle Fehlerbehandlung zu betreiben. In der Praxis können solche Laufzeitprüfungen mit vertretbarem Aufwand umgesetzt werden [Bod10], jedoch bleibt die statische Codeanalyse die einzige Technik, die die Abwesenheit von Fehlern vorab garantieren kann.

Ob statische Codeanalyse, Testen, oder Laufzeitprüfung: Jede Programmanalyse muss wissen, wonach sie suchen muss – und benötigt somit eine Spezifikation des erwünschten Verhaltens (und kann dann nach möglichen Verletzungen suchen) oder des unerwünschten Verhaltens (und kann dann nach Möglichkeiten suchen, dieses zu erreichen). Es gibt eine Reihe von Programmverhalten, die gewöhnlich zum undefinierten Verhalten oder Programmabbruch führen und somit immer unerwünscht sind; so kann man etwa gezielt auf Pufferüberläufe verifizieren oder testen. Darüber hinaus muss aber das erwünschte oder unerwünschte Programmverhalten exakt spezifiziert werden – etwa in Form eines Sicherheitsmodells, das die genauen Rechte eines jeden Nutzers und Subsystems beschreibt und einschränkt. Solche Modelle können – wie auch andere Spezifikationen – sehr schnell sehr komplex werden. Das führt zu der absurden Situation, dass – hinreichende Fortschritte in Verifikation und Testen vorausgesetzt – wir zwar immer besser prüfen können, ob eine Software der Spezifikation entspricht; wir aber nicht wissen, ob die Spezifikation das umfasst, was man will oder braucht.

Angesichts der Vielzahl der Herausforderungen ist klar, dass kein Ansatz für sich allein genommen ausreichen kann. Die verschiedenen Verfahren der Programmanalyse (statische Codeanalyse, Testen, Laufzeitprüfung) müssen *Hand in Hand* arbeiten, um ihre jeweiligen Stärken auszuspielen – etwa durch statische Codeanalyse kleiner Subsysteme, deren Zusammenspiel im Kontext dann durch umfassende Tests geprüft wird. Die größte Herausforderung jedoch ist das Formulieren geeigneter Spezifikationen – und zwar auf eine Weise, die jedem Programmierer zugänglich ist. Ohne

Spezifikation gibt es keine Fehler, aber auch keine Korrektheit – sondern „nur“ Überraschungen.

Chancen eröffnen hier Verfahren zum *Extrahieren von Spezifikationen* aus bestehenden Systemen – derzeit in Form von axiomatischen Vor- und Nachbedingungen [ECGN01], endlichen Automaten [DKM+12] oder Prozessmodellen [Sch11]. Die Grundidee ist, solche Verfahren auf bestehende Systeme anzuwenden, und daraus *Standardmodelle* für deren Verhalten (auch im Hinblick auf Sicherheit!) zu extrahieren, um dann (mit Hilfe von Verifikation und Testen) zu prüfen, inwiefern andere Systeme diese (impliziten) Standards erfüllen. Das Ergebnis wäre dann nicht mehr eine *Verletzung* eines explizit spezifizierten Sicherheitsmodells, sondern vielmehr eine *Anomalie* im Vergleich zu anderen (ähnlichen) Systemen, was die Sicherheit angeht. Die Extraktion solch detaillierter Spezifikationen ist eine offene Forschungsfrage; die in Milliarden von Programmzeilen codierte Erfahrung aber ist ein Schatz, den es zu heben gilt.

3.5 Herausforderung: Nachhaltig sichere Integration von kryptographischen Primitiven und Protokollen

Der Entwurf komplexer Systeme erfolgt in der Regel komponentenweise; die gewaltige Komplexität großer Softwareprojekte, wie beispielsweise moderner Mehrbenutzer-Betriebssysteme, ist ohne Modularisierung nicht beherrschbar. Anders als im Fall von fehlender Funktionalität, welche meist durch Hinzunahme eines weiteren Moduls leicht nachgerüstet werden kann, ist jedoch eine Nachrüstung von Sicherheitseigenschaften normalerweise nicht ohne Weiteres möglich. Die mit der Modularisierung oft einhergehende isolierte Sicht auf einzelne Teilsysteme birgt daher hohe Sicherheitsrisiken. Auch wenn jede einzelne Komponente „lokal sicher“ scheint, ist damit längst nicht garantiert, dass das Gesamtsystem „global sicher“ ist.

Dieses Kompositionsproblem besteht in zwei Dimensionen: In der vertikalen Dimension kompromittiert ein Angreifer einen Teil des Softwarestacks, um Zugriff auf andere Schichten zu erhalten. Beispielsweise wird in das Betriebssystem eines Rechners eingebrochen, um auf dem Rechner betriebene Anwendungen zu manipulieren. Das Problem ist in der horizontalen Dimension subtiler, aber nicht geringer. Sicherheitslücken in unwichtigen Komponenten können die Sicherheit hochkritischer Komponenten (und damit die des Gesamtsystems) beeinträchtigen. So konnte die Malware Stuxnet beispielsweise eine Sicherheitslücke im Drucksystem von Windows nutzen, um den ganzen Rechner zu kompromittieren und sich schlussendlich in der Aufbereitungsanlage in Bushehr auszubreiten.

Wie lokale Sicherheitsgarantien konkret durch ungeeignete Komposition global ausgehebelt werden können, demonstriert ein Angriff auf das Chip-and-PIN-Verfahren [MDAB10]. Beim Chip-and-PIN-Verfahren handelt es sich um ein Chipkartengestütztes Bezahlsystem; der Kunde führt seine Karte in das Händler-Terminal ein und autorisiert die Zahlung mittels Eingabe einer PIN oder per Unterschrift auf ei-

ner Rechnung. Jede einzelne der zur Auswahl stehenden Autorisierungsformen kann dabei für sich genommen als hinreichend sicher angesehen werden. Der Mechanismus zur Auswahl zwischen beiden Modi ist jedoch so implementiert, dass die Karte bei Autorisierung per Unterschrift jede PIN akzeptiert. Bei einem Man-in-the-Middle-Angriff kann man nun dem Terminal vorgaukeln, die Autorisierung erfolge per PIN, während die Karte im Modus für Autorisierung per Unterschrift arbeitet. Das heißt, ein Angreifer kann eine gestohlene Karte zum Bezahlen verwenden, ohne die gültige PIN zu kennen oder eine Unterschrift fälschen zu müssen. Er muss lediglich die Kommunikation zwischen der gestohlenen Karte und dem Terminal kontrollieren können. Das kann zum Beispiel dadurch geschehen, indem beim Bezahlvorgang am Terminal eine selbsterstellte Dummy-Karte verwendet wird, welche über Funk oder ein verstecktes Kabel mit der gestohlenen Karte verbunden ist.

Besonders deutlich wird das Kompositionsproblem beim TLS-Key-Renegotiation-Angriff [RRDO10]. Das TLS-Protokoll selbst dient zum Aufbau und Betrieb einer verschlüsselten und authentifizierten Kommunikationsverbindung. Dabei ist es auch möglich, während einer laufenden Sitzung den aktuellen Schlüssel zu verwerfen und einen neuen Schlüssel für die weitere Kommunikation auszuhandeln. Bei einem klassischen Key-Renegotiation-Angriff unterbricht ein Angreifer den TLS-gesicherten Kommunikationsaufbau seines Opfers und startet stattdessen eine eigene TLS-gesicherte Sitzung. Er stößt dann eine Key-Renegotiation an. Nun lässt er den bislang blockierten Kommunikationsaufbau des Opfers weiterlaufen. Die so entstehende Verbindung ist zwar wirksam verschlüsselt und authentifiziert. Allerdings ist seitens des Servers der Authentifikationsvorgang abgeschlossen, der Client befindet sich durch die Unterbrechung jedoch noch mitten im Anmeldevorgang. In der Folge sendet er Anmeldeinformationen. Das kann zum Beispiel dazu führen, dass vertrauliche Login-Information als öffentliche Kurznachricht in einem Social-Media-Portal sichtbar wird.

Die theoretische Kryptographie bietet mit Universal-Composability- bzw. Reactive-Simulatability-Modellen [Can01; BPW07] einen Ansatz zur Lösung des Dilemmas an: Gelingt in einem dieser Modelle ein formaler Sicherheitsbeweis für eine Komponente, so ist damit der sichere Einsatz dieser Komponente in beliebigen Kontexten garantiert. Beweisbare Sicherheit in den genannten Modellen bringt jedoch eine Fülle von Nachteilen mit sich, die dem praktischen Nutzen entgegenstehen. Zunächst sind die Sicherheitsbeweise selbst ausgesprochen aufwändig zu führen und entsprechend fehleranfällig. Da tatsächlich alle formal denkbaren Angriffe ausgeschlossen werden, sind die Modelle entsprechend streng; es ist oft immenser Aufwand nötig, um Systeme beweisbar sicher zu konzipieren, und das Ergebnis bleibt in Sachen Effizienz um Größenordnungen hinter praktisch motivierten, aber theoretisch unsicheren, Ad-hoc-Lösungen zurück. Ist auch nur eine einzige Sicherheitsannahme verletzt, kann in der Regel keinerlei Restgarantie mehr gegeben werden. Aus all diesen Gründen sind die genannten Modelle *de facto* praxisuntauglich.

Ein pragmatischerer Lösungsansatz aus der Softwaretechnik sieht „Verträge“ zwischen einzelnen Systemkomponenten vor. Jede Komponente eines komplexen Systems steht mit anderen Komponenten in Wechselwirkung und nutzt oder erbringt Dienste. Die Sicherheitseigenschaften der erbrachten Dienste werden vertraglich geregelt. Dadurch wird zumindest sichergestellt, dass keine Komponente fälschlich bestimmte Sicherheitseigenschaften einer anderen Komponente voraussetzt. Wie sich jedoch lokale Verträge zwischen Komponenten aus globalen Sicherheitsanforderungen ableiten lassen, ist weiterhin eine offene Frage. Das Vertragsmodell zwischen Komponenten macht außerdem die Wiederverwendung dieser Komponenten in anderen Kontexten umständlicher. Dies schränkt den Nutzen der Modularität stark ein und insbesondere das Problem der sicheren Einbindung von Legacy-Systemen bleibt ungelöst. Ein prominentes Beispiel der potentiellen Problematik, wenn nur ein einziges Modul ausgetauscht wird, stellt der CAN-Bus für die elektronische Kommunikation zwischen Steuergeräten in Kraftfahrzeugen dar. Ursprünglich mit dem Ziel konzipiert, Kabelbäume und damit das Fahrzeuggewicht zu reduzieren, stand Sicherheit gegen Manipulation durch externe Angreifer nicht im Fokus der Entwicklung. Ein Zugriff auf den Bus (z.B. zu Wartungszwecken) war ohnehin nur kabelgebunden über einen Steckkontakt im Fahrzeuginneren vorgesehen. Umso kritischer gestaltete sich der mit dem allgemeinen Aufkommen von WLAN- und Bluetooth-Schnittstellen einhergehende Wunsch nach der Möglichkeit eines drahtlosen Wartungszugriffs ohne umständliche Verkabelung. Durch die Integration eines Funkmoduls war ein universeller Kommunikationsbus, der auch kritische Komponenten wie die Motorsteuerung oder Bremsen steuert, ohne ein geeignetes Sicherheitskonzept drahtlos von außerhalb des Fahrzeuges zu erreichen.

Zusammenfassend stellen sich hinsichtlich des Themas „sichere Integration“ verschiedene offene Fragen. Zum einen ist bislang unzureichend geklärt, inwieweit sich lokale Sicherheitsanforderungen auf Komponentenebene aus den globalen Sicherheitsanforderungen des Gesamtsystems ableiten lassen. Selbiges gilt auch für den umgekehrten Weg, bei dem aus den Sicherheitseigenschaften der einzelnen Komponenten auf möglichst maximale Sicherheitsgarantien des Gesamtsystems geschlossen wird. Der gangbarste Ansatz scheint hier, Werkzeuge zu entwickeln, die es erlauben, eine Architektur beginnend mit einem abstrakten Gesamtsystem schrittweise so auf konkrete Module zu verfeinern, dass dabei gleichzeitig der Rückweg für einen Sicherheitsbeweis des Gesamtsystems basierend auf den Eigenschaften der einzelnen Module geebnet wird. Selbst bei einem rein intuitiven Systementwurf wird dieser Ansatz zwar bereits oft „händisch“ verfolgt, es besteht aber zur Zeit nur unzureichende Unterstützung durch durchgängige formale Werkzeuge. Zwei Fragen bleiben hiervon jedoch unberührt: Wie kann man überhaupt systematisch die erforderlichen globalen Sicherheitsanforderungen für ein Gesamtsystem identifizieren? Wie kann man die gewährleisteten formalen Sicherheitsgarantien im Fall von Legacy-Systemen zuverlässig rückgewinnen?

3.6 Herausforderung: Aufspüren absichtlich eingetragener Schwachstellen und Provenance Tracking

Um die Sicherheit von Software zu erhöhen wird heutzutage ein Zertifikat verlangt, das sicherstellt, dass ein bestimmtes Softwareprodukt von einem vertrauenswürdigem Hersteller stammt. Ganz abgesehen von der Problematik mit gefälschten Zertifikaten, die in letzter Zeit gehäuft aufgetreten sind, enthält solch ein Verfahren jedoch immer noch einige Angriffspunkte: Der Nutzer müsste einerseits alle Anbieter kennen um ihnen wirklich Vertrauen entgegenbringen zu können. Andererseits kann auch ein grundsätzlich vertrauenswürdiger und bekannter Softwarehersteller andere Interessen haben als der Nutzer. So ist in der Vergangenheit schon des Öfteren Software bekannt geworden, die den Nutzer zu einem gewissen Grad ausspioniert. So haben z.B. mobile Apps wie Facebook oder Twitter das gesamte Adressbuch eines Handys ohne explizite Zustimmung des Nutzers auf ihre Server transferiert, um dieses nach bekannten Kontakten zu durchforsten. Aber auch Innentäter oder Hacker können unbemerkt Code in ein Programm einschleusen und damit dessen Sicherheit kompromittieren.

Besser als auf die Gutartigkeit eines Herstellers zu vertrauen wäre es allerdings, wenn man die Funktionsweise eines Programms analysieren könnte. Programmanalysen können zwar aufgrund des sogenannten Halteproblems nie die volle Funktionalität eines Programms verstehen, allerdings können bestimmte Sicherheitsaussagen wenigstens so approximiert werden, dass ein Programm, das als sicher eingestuft wird auf jeden Fall sicher ist, während ein als unsicher eingestuftes Programm wirklich ein Sicherheitsproblem aufweisen kann oder aber nicht genau genug analysierbar war. Die entsprechenden Techniken werden unter dem Stichwort *Sprachbasierte Sicherheit* eingeordnet. Insbesondere das Teilgebiet der Informationsflusskontrolle bietet die Möglichkeit Programme auf Schwachstellen zu untersuchen: Informationsflusskontrolle überprüft, ob sensitive Daten, wie z.B. ein Adressbuch, in öffentlichen Kanälen wie dem Internet landen können. Somit lassen sich also spionierende Programme aufspüren. Weiterhin kann Informationsflusskontrolle überprüfen, ob nicht vertrauenswürdige Eingaben eines Benutzers wichtige Berechnungen des Programms beeinflussen können. Solche Injektionsattacken tauchen leider immer wieder auf und erlauben dem Angreifer beliebigen Code auszuführen, wodurch ganze Server im Internet gekapert und z.B. Nutzerdaten wie Kreditkartennummern gestohlen werden können.

Um Informationsflusskontrolle effektiv durchführen zu können, muss man die Herkunft (*Provenance*) von Daten kennen. Die Herkunft wird dann an alle Ergebnisse von Berechnungen geheftet, die von diesen Daten abhängen. Nur so kann gewährleistet werden, dass am Ende einer Berechnung noch bekannt ist, ob diese von geheimen Eingaben abhängt oder ob die berechneten Daten öffentlich einsehbar sein können.

Im Endeffekt möchte man eine sogenannte Ende-zu-Ende-Sicherheit gewährleisten, die sensible Nutzerdaten auf ihrem ganzen Lebensweg schützt. Dies beginnt mit der

verschlüsselten Speicherung auf einem Server, der Zugangskontrolle zu den Daten, Informationsflusskontrolle während der Verarbeitung von Daten und hört mit der verschlüsselten Übertragung oder Speicherung der Ergebnisse auf. Ziel muss es sein, ein Zertifikat nicht nur über die Herkunft des Programms zu erhalten, sondern auch eine Garantie, dass ein Programm sicher mit seinen Daten umgeht.

3.7 Herausforderung: Gemeinsame Sprache

Security by Design, also das Berücksichtigen der Sicherheit von Anfang an, bedingt, dass der gesamte Entwicklungsprozess von Dokumenten begleitet werden muss, in denen Sicherheitsanforderungen und schon erreichte Sicherheitsgarantien festgehalten sind. Diese Dokumente dienen der Kommunikation über verschiedene Entwicklungsstadien hinweg und darüber hinaus der Kommunikation zwischen verschiedenen Fachdisziplinen.

Bisher ist aber nicht sichergestellt, dass die unterschiedlichen Sichten der beteiligten Einzeldisziplinen konsistent sind. Dies wird insbesondere dadurch behindert, dass die Fachsprachen der beteiligten Einzeldisziplinen nicht kompatibel sind. Umgangssprachliche Formulierungen, auf die häufig als gemeinsame Sprache ausgewichen wird, sind nicht präzise genug und führen zu Missverständnissen. Die einzelnen Garantien, die von den beteiligten Fachdisziplinen gegeben werden, ergänzen sich somit häufig nicht zu einer lückenlosen Gesamtgarantie. Wirklich verlässliche Sicherheitsaussagen gibt es damit häufig nur „lokal“, also beispielsweise für einzelne sichere Kommunikationsverbindungen, für die Verfügbarkeit von Backups oder für die korrekte Implementierung einer bestimmten funktionalen Anforderung. Welche Sicherheitsgarantie aber für das ganze System gilt, wenn die einzelnen Sichten der Disziplinen inkonsistent sind, ist nicht klar.

Ein anschauliches Beispiel für die dabei entstehenden Probleme gibt eine im Jahr 2004 mit Quantenkryptographie gesicherte Banküberweisung. Physiker hatten ein Verfahren umgesetzt, bei dem ein Angreifer garantiert keine Information über den Schlüssel erhält. Es war für einen Angreifer aber möglich, Nachrichten gezielt zu verändern, ohne dabei den Inhalt zu erfahren oder kennen zu müssen. Das auf das quantenkryptographische Verfahren aufgesetzte Protokoll für die Banküberweisung setzte aber einen anderen Sicherheitsbegriff voraus. Durch die Fehlannahme, dass durch die geheime Übertragung der Schlüssel automatisch eine sichere Überweisung entsteht, wurde das Gesamtprotokoll angreifbar und zu überweisende Beträge konnten gezielt verändert werden [BMQS05].

In der Kryptographie wird vorausgesetzt, dass Implementierungen korrekt sind. Die Kryptographie untersucht nur prinzipielle Schwächen, die von Implementierungsfehlern unabhängig sind. Die Verifikation von Programmcode überprüft die Korrektheit einer Implementierung. In den meisten Fällen sind diese beiden Begriffe von Korrektheit aber nicht deckungsgleich, da die häufig rein funktionale Spezifikation, die bei der Verifikation überprüft wird, beispielsweise nicht sicherstellt, dass etwa

das beim Verschlüsseln verwendete Schlüsselmaterial gut ist. Bei der Verwendung schlechten Schlüsselmaterials kann ein Angreifer unter Umständen Informationen über den verschlüsselten Klartext erhalten [hei08].

Programmierfehler können auch zu einem verbotenen Informationsfluss führen. Spezielle Tools der Code-Analyse (Information Flow Control) finden unerwünschte Informationsflüsse. Um solche unerwünschten Informationsflüsse aber finden zu können, muss spezifiziert sein, welche Informationsflüsse erlaubt sind. Es ist allerdings nicht sichergestellt, dass eine solche Spezifikation konsistent mit der kryptographischen Spezifikation ist.

In der Softwareentwicklung gibt es bereits vielversprechende Ansätze, die es ermöglichen, schon während des Entwurfszeitpunkts Sicherheitsaspekte zu modellieren [Jür02; BDL06; LBD02] und deren Implementierung zu überprüfen [JYB08; DPP12]. Hierbei handelt es sich aber meist um Lösungen mit einem fokussierten Anwendungsfeld. Es ist eine große Herausforderung, übergreifende Lösungen zu finden, die sicherstellen, dass während des gesamten Entwicklungszeitraums und über alle beteiligten Disziplinen hinweg ein konsistentes Bild vorhanden ist.

Die Softwareverifikation untersucht das Verhältnis von Eingaben in einen Prozess zu dessen Ausgabe, also die funktionalen Eigenschaften von Prozessen. Sicherheitseigenschaften sind aber nichtfunktional. Beispielsweise ist eine Verschlüsselung funktional über das Gelingen der Entschlüsselung definiert. Die Sicherheit einer Verschlüsselung rührt jedoch vielmehr aus Verteilungen von Ausgaben her, nicht von deren Verhältnis zu Eingaben. Gelingt es, diese Lücke zu schließen, sind die Methoden der Softwareverifikation auch im Bereich der IT-Sicherheit anwendbar.

Sicherheitsanforderungen an Gesamtsysteme werden meist ganzheitlich formuliert. Welche Ansprüche an Teilsysteme diese Anforderungen implizieren, ist oft unklar. Im Gegenzug ist es im Allgemeinen schwer zu bestimmen, welche Garantien für Gesamtsysteme aus Eigenschaften einzelner Komponenten ableitbar sind. Es ist eine Herausforderung, Anforderungen und Garantien gleichermaßen zwischen den einzelnen Stufen eines Entwicklungsprozesses zu propagieren.

Im Bereich der *Information Flow Control* muss ein Weg gefunden werden, erlaubte Informationsflüsse auf der Grundlage von kryptographischen Anforderungen und Architekturmodellen zu spezifizieren.

Die Forderung nach einer gemeinsamen Sprache für verschiedene Disziplinen wirft neue Fragen auf, beispielsweise nach den richtigen Abstraktionsgraden. Ein hoher Detailgrad ist für manche Anwendungen, wie zum Beispiel die Verifikation von kryptographischen Protokollen, notwendig. Für andere Anwendungen kann er sich aber aufgrund der Komplexität des Gesamtsystems negativ auswirken.

Es ist offen, wie von abstrakten, umgangssprachlichen Sicherheitsaussagen systematisch auf Fragestellungen von Einzeldisziplinen geschlossen werden kann. Eine Methodik der schrittweisen Verfeinerung im Sinne eines Angriffsbaums ist denkbar.

Durch die zunehmende Verrechtlichung von Anforderungen an die IT-Sicherheit spielt im Rahmen von *Security by Design* aber auch der Gesetzgeber zunehmend eine

Rolle bei der Formulierung von funktionalen und nichtfunktionalen Anforderungen an die Systeme. Die Besonderheit ist, dass er in Teilen ein eigenes Sprachsystem, die Rechtsterminologie, mit zwingendem Geltungsanspruch erzeugt. Die sinnerhaltende Transformation dieser Rechtssprache in Allgemeinbegriffe ist die klassische Tätigkeit des Juristen. Im Rahmen von *Security by Design* kommt nun noch die nur interdisziplinär zu bewältigende Aufgabe hinzu, auch die sinnerhaltende Transformation in die Sprachdomänen der Informatik-Fachdisziplinen zu gewährleisten und die Prozesse dieser Übertragung nachvollziehbar zu dokumentieren.

Die Gesamtheit der Betrachtungsweisen von Einzeldisziplinen soll helfen, die Sicherheit von Gesamtsystemen zu evaluieren. Inwiefern die Sichtweisen der Einzeldisziplinen aber alle Sicherheitsrisiken beleuchten, ist nicht bekannt.

4. SECURITY BY DESIGN BEI VERTEILTER ENTWICKLUNG UND INTEGRATION

Heutige und zukünftige Softwareprodukte oder IT-Lösungen entstammen nur in den seltensten Fällen einem einzigen Entwicklerteam, wie Abbildung 5 zeigt. So liefern fremde Hersteller im Rahmen von Entwicklungsaufträgen oder über die Bereitstellung von Open-Source-Lizenzen Software als Komponenten, Bibliotheken bis hin zu Diensten, die mit eigenem Komponenten zu größeren Produkten kombiniert werden. In einem weiteren Aggregationsschritt werden verschiedene Produkte häufig zu komplexen IT-Lösungen integriert. Für Anwender ist es wichtig, dass die von ihnen eingesetzte Software die erwarteten Sicherheitseigenschaften hat, wobei die Sicherheitsbedürfnisse und Erwartungen verschiedener Anwender unterschiedlich sein können [FPP12]. Entsprechend hinterfragen mittlerweile viele Anwender mit höherem Sicherheitsbedürfnis, was Integratoren oder Hersteller unternehmen, um die Sicherheit von IT-Lösungen oder Produkten zu verbessern [Bai12]. Verwenden Integratoren oder Hersteller wiederum Produkte anderer Hersteller, dann sollten entlang der kompletten Wertschöpfungskette geeignete Methoden angewendet werden, die zur Sicherheit des Endprodukts beitragen. Eine Berücksichtigung der kompletten Wertschöpfungskette ist insbesondere deshalb wichtig, damit Hersteller Risiken durch sogenannte *Advanced Persistent Threats* (APT) für Anwender reduzieren können, bei denen individualisierte und spezialisierte Angriffe auf ausgewählte Ziele durchgeführt werden. In der Vergangenheit wurden für solche Angriffe oftmals gerade solche Sicherheitslücken ausgenutzt, die dadurch entstanden sind, dass bei der verteilten Entwicklung und Integration keine adäquaten Sicherheitsprozesse angewendet wurden [Bai12]. Selbst die Sicherheit der Einzelteile stellt keine hinreichende Bedingung für die Sicherheit des durch verteilte Entwicklung oder Integration entstehenden Gesamtproduktes dar. So treten Sicherheitslücken bei der Integration oftmals an den Schnittstellen der integrierten Komponenten bzw. Produkte auf. Eine weitere Problematik ergibt sich durch die Integration von Open-Source-Software, Commercial-of-the-Shelf-Software (COTS) oder Legacy-Code, was den typischen Marktanforderungen heutiger Softwareentwicklung hinsichtlich Zeit und Kosten geschuldet ist.

Um die Sicherheit von in verteilter Entwicklung entstandenen Produkten und integrierten Lösungen zu verbessern, braucht es geeignete Vorgehensweisen und Methoden, bei welchen die teilweise äußerst komplexen Wertschöpfungsketten der Softwareentwicklung berücksichtigt werden. Die Verantwortung zur Anwendung solcher Vorgehensweisen und Methoden liegt typischerweise im letzten Glied der Wertschöpfungskette. Zur Entwicklung von sicherer Software müssen jedoch auch deren Lieferanten in die Sicherheitsprozesse einbezogen werden.

Die große Bedeutung von wertschöpfungskettenumfassenden Sicherheitsprozessen zur Entwicklung sicherer Software und IT-Lösungen wurde mittlerweile von der Softwareindustrie erkannt. So gibt es hier Aktivitäten wie etwa von dem *Open Group*

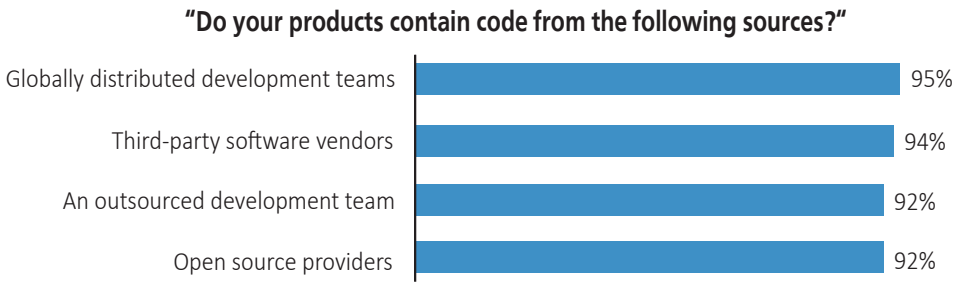


Abbildung 5: Die Verwendung von extern entwickeltem Code (Quelle: [For11b]): Die Werte basieren auf einer Befragung von 336 IT-Spezialisten mit Bezug zur Softwareentwicklung in ihren jeweiligen Unternehmen. Die Unternehmen haben ihren Sitz in den USA, Kanada, Großbritannien, Frankreich und Deutschland.

Trusted Technology Forum [OTT11], welche die Sicherheit von Software unter Berücksichtigung verteilter Herstellungsprozesse betrachtet.

Auch wenn heute für Anwender und Hersteller Sicherheit als Eigenschaft und Qualitätsmerkmal ihrer IT-Produkte und -Lösungen immer wichtiger wird, so ist festzustellen, dass Hersteller hinsichtlich der systematischen und methodisch verankerten Erreichung von Sicherheit bei extern entwickelten Softwarekomponenten deutlich weniger Aufwand betreiben, als sie dies für eigene Softwarekomponenten tun. Einen Beleg hierfür liefern die Ergebnisse einer Untersuchung zum Test von Sicherheitseigenschaften von extern entwickeltem Code, die in Abbildung 6 dargestellt werden. Die Betrachtung der in Abbildung 6 zugrunde liegenden Untersuchung bezieht sich nur auf Phasen, die im Softwarelebenszyklus hinter der Designphase liegen. Es ist jedoch anzunehmen, dass sich bei der Mehrheit von Herstellern und Integratoren die aktuelle Situation hinsichtlich der Designphase von der Kernaussage in Abbildung 6 nicht wesentlich unterscheidet. Ein wichtiger Grund für diese Defizite mag darin bestehen, dass Herstellern und Integratoren keine einheitlichen Standards mit Vorgehensweisen und Methoden zur Verfügung stehen, mit denen wertschöpfungskettenumfassende Sicherheitsprozesse umgesetzt werden können. Existierende Sicherheitsentwicklungsprozesse wie etwa der Microsoft SDL wurden nicht explizit für verteilte Entwicklung über komplexen Wertschöpfungsketten oder für Integration entwickelt [WOUK12].

Da heute in den meisten praktisch relevanten Softwareprodukten und IT-Lösungen Komponenten verschiedener Hersteller bzw. Komponenten, die nach verschiedenen Sicherheitsprozessen entwickelt wurden, integriert werden, verlangt die Entwicklung sicherer Softwareprodukte und IT-Lösungen nach einheitlichen und wertschöpfungskettenumfassenden Lösungen für sichere Softwareentwicklungsprozesse. Ansätze, die sich nur auf die eigene Softwareentwicklung beziehen, reichen nicht aus, um die Erfolgsaussichten für Hacker zu reduzieren und die Softwaresicherheit für Anwender signifikant zu verbessern [CA11]. Hier besteht für die praktische Anwendung ein

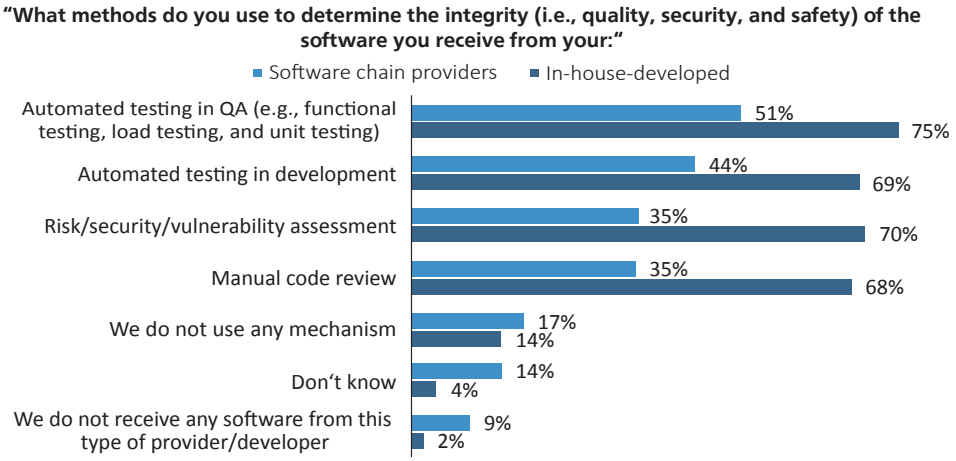


Abbildung 6: Die Unterschiede in der Qualitätssicherung von intern und extern entwickeltem Code (Quelle: [For11b]): Grundlage ist hier dieselbe Befragung wie in Abbildung 5.

enormer Forschungsbedarf. Die Vorstellung der zukünftigen, sicheren Softwareentwicklung wird bestimmt von folgender Vision:

> *Die verteilte Entwicklung von sicherer Software und Integration von sicheren IT-Lösungen wird durch vereinheitlichte, organisationsübergreifende und wertschöpfungskettenumfassende Sicherheitsprozesse gekennzeichnet sein, bei denen Sicherheit zum jeweils frühest möglichen Zeitpunkt und durchgängig im Lebenszyklus berücksichtigt wird.*

Der Schritt zur Umsetzung dieser Vision stellt für Hersteller von Software eine wichtige strategische Entscheidung dar. Auf der einen Seite bedeutet diese Entscheidung für Hersteller, dass sie zur Verbesserung von Sicherheit kooperieren müssen und darauf angewiesen sind, dass ihre Partner entsprechend zur Kooperation beitragen. Kooperation verlangt ebenfalls, dass vorhandene Formen der Interaktion weiterentwickelt und verändert werden müssen. Auf der anderen Seite bietet eine solche strategische Entscheidung Softwareherstellern das Potenzial zur Verbesserung der Sicherheit ihrer Produkte verbunden mit günstigeren Entwicklungskosten. Die Umsetzung wertschöpfungskettenumfassender Sicherheitsprozesse stellt für Hersteller einen wichtigen Wettbewerbsfaktor dar. Mit wachsender Bedeutung der Softwaresicherheit für Anwender wie etwa aufgrund immer weiterer Compliancevorgaben zur Reduktion von Risiken sind solche Sicherheitsprozesse ein wichtiges Kriterium bei der Vermarktung.

Damit diese Vision Wirklichkeit werden kann, sind eine Reihe von Herausforderungen zu bewältigen, die im Folgenden beschrieben werden.

4.1 Herausforderung: Standardisierung von wertschöpfungskettenumfassenden Sicherheitsprozessen

Damit Sicherheitsprozesse wertschöpfungskettenumfassend angewendet werden können, ist ein aufeinander abgestimmtes und vereinheitlichtes Vorgehen zwischen den verschiedenen Akteuren der Wertschöpfungskette erforderlich. Hierfür benötigt man Standards, die entwickelt werden müssen und die alle relevanten Aspekte der verteilten Entwicklung abdecken. Hierbei sind zu berücksichtigen:

(1) vereinheitlichte Methoden und Werkzeuge, die in den Sicherheitsprozessen angewendet werden
(2) standardisierte Beschreibung der bei der Entwicklung von Komponenten angewendeten Sicherheitsprozesse
(3) standardisierte Beschreibung der von den Komponenten geforderten und angebotenen Sicherheitseigenschaften
(4) Möglichkeiten zur Überprüfung der korrekten Anwendung von Sicherheitsprozessen

Standards müssen in diesem Zusammenhang das komplette Spektrum der heutigen verteilten Entwicklung abdecken. Dieses reicht von der verteilten Entwicklung, bei der im Rahmen dedizierter Entwicklungsaufträge neue Softwarekomponenten entwickelt werden, wobei sich Design und Entwicklung der Softwarekomponenten an den spezifischen Anforderungen des Auftrags orientieren kann, bis hin zur Integration vorgefertigter Komponenten wie etwa Open-Source-Produkte oder COTS-Produkte. Die Entwicklung solcher Lösungen bis hin zu Standards stellt einerseits eine große Herausforderung dar, die es zu bewältigen gilt. Andererseits bieten solche Lösungen und Standards auch eine große Chance für Hersteller und Integratoren zur Verbesserung der Sicherheit von Software, da damit Vorgehensweisen und Interaktionsformen festgelegt sind und diese nicht wiederholt in Einzelfällen festgelegt werden müssen. Durch einen einheitlichen Standard werden unter den Beteiligten ein gemeinsames Verständnis und kongruente Sichtweisen geschaffen.

Die Welt der Softwareentwicklung ist heute durch eine sehr große Komplexität gekennzeichnet. Auch wenn sich die Softwareindustrie stark globalisiert und hinsichtlich bestimmter Aspekte etwas vereinheitlicht hat, so wird diese Komplexität bestimmt von Dingen wie unterschiedlichen Unternehmenskulturen, Eigenheiten der Anwenderbranche, nationale und internationale Regulierung, unterschiedliche Methoden des Software Engineerings (z.B. agile Entwicklung) bis hin zu unterschiedlich ausgeprägten Sicherheitsprozessen in der Softwareentwicklung [Bai12]. Diese Komplexität stellt eine große Hürde dar, die es bei der Standardisierung der wertschöpfungskettenumfassenden Behandlung von Sicherheit zu überwinden gilt.

Momentan stehen viele Unternehmen der Softwareindustrie noch vor dem Schritt, Sicherheitsprozesse für die eigenen Entwicklungsarbeiten zu verbessern. Eine darüber hinausgehende Behandlung der gesamten Wertschöpfungskette liegt für die meisten Unternehmen noch in der ferneren Zukunft. Dabei haben einige Vertreter

der Softwareindustrie und der Anwender längst verstanden, dass Maßnahmen zur Sicherheit von Softwareprodukten die Lieferkette bei der Softwareentwicklung mit einschließen müssen. So wurde bereits vorgeschlagen, dass das Risikomanagement von Unternehmen die Risiken durch Lieferketten zu berücksichtigen hat. Arbeiten in diesem Zusammenhang liefern bisher hauptsächlich Antworten, was man gegen Angriffe auf Lieferketten unternehmen kann, wie etwa in dem Standard [ISO11] oder in [MM08; WLL08; SRM+09]. Diese Vorschläge zur Lieferkettensicherheit sind jedoch nicht spezifisch für Softwareprodukte. Man kann verstärkt den Trend feststellen, dass insbesondere staatliche Organisationen als Bezieher von Software in Form von Endprodukten, Komponenten oder Integrationslösungen die Sicherheitsprozesse der Hersteller stärker hinterfragen. Für diese stellt die Existenz von geeigneten Sicherheitsprozessen ein wichtiges Kriterium bei der Entscheidung für bestimmte Produkte oder Hersteller dar [NIS10].

Für Unternehmen und Organisationen als Anwender ist die Betrachtung der Sicherheit von eingesetzter Software ein wesentlicher Bestandteil der eigenen Sicherheitsarchitektur [The11]. Bei der Integration von Softwareprodukten verschiedener Hersteller in Unternehmensinfrastrukturen ist es ebenfalls von Vorteil, wenn Integratoren Aussagen oder Zusicherungen der Hersteller über Sicherheitseigenschaften ihrer Software verwerten können. Aus Gründen von Effektivität und Effizienz ist eine Vereinheitlichung dieses Informationsflusses auf der Basis eines Standards wichtig.

Konkretere Vorschläge und Best Practices hinsichtlich Lieferkettensicherheit für Software und Entscheidungshilfen zur Bewertung von Produkten und Herstellern vor dem Hintergrund ihrer Sicherheitsprozesse wurden von dem *Open Group Technology Forum* in [OTT11] gegeben. Jedoch sind die Vorschläge in der Praxis schwierig umzusetzen, da auf Seiten der Hersteller einheitliche Vorgehensweisen fehlen wie etwa herstellerübergreifende konsistente Begriffe oder einheitliche und wertschöpfungskettenumfassende Sicherheitsprozesse.

Damit Sicherheitsprozesse wertschöpfungskettenumfassend funktionieren können, müssen im Rahmen von Standards unter anderem die folgenden Fragen beantwortet werden:

- Wie lassen sich aus den Sicherheitsanforderungen der Anwendung die Sicherheitsanforderungen an die Komponenten ableiten?
- Wie können die Sicherheitseigenschaften an zu entwickelnde Komponenten und an einzubindende Komponenten einfach und effizient beschrieben werden?
- Wie können solche Beschreibungen so gestaltet werden, dass sie maschinenprüfbar und zugleich für Entwickler lesbar sind?
- Wie können Sicherheitseigenschaften und Sicherheitsgarantien von Komponenten beschrieben werden, die für eine klar beschriebene Anwendung und eine dedizierte Umgebung entwickelt wurden?
- Wie können die Sicherheitseigenschaften und Sicherheitsgarantien von Komponenten beschrieben werden, bei denen konkrete Anwendung und Umgebung zum

Zeitpunkt von deren Entwicklung und Bereitstellung noch gar nicht bekannt sind?

- Wie kann man sicherstellen, dass alle relevanten Sicherheitsanforderungen an Komponenten bereits zur Designphase erfasst werden?
- Wie lassen sich Sicherheitsprozesse anwendungsbranchenübergreifend vereinheitlichen?
- Wie kann man die Wirtschaftlichkeit von wertschöpfungskettenumfassenden Sicherheitsprozessen messen?
- Wie können Aspekte von Produktlinien in den Standards berücksichtigt werden?
- Wie kann man die Einhaltung der standardisierten Sicherheitsprozesse durch Hersteller oder Lieferanten überprüfen?
- Wie kann man Verletzungen der standardisierten Sicherheitsprozesse durch Hersteller oder Lieferanten nachweisbar machen?
- Wie können Hersteller den Integratoren relevante Informationen zu Sicherheitseigenschaften ihrer Produkte für eine sichere Integration zur Verfügung stellen?
- Wie können Integratoren die ihnen von den Produktherstellern gegebenen Informationen zu Sicherheitseigenschaften zusammenführen und nutzenbringend kombinieren?

4.2 Herausforderung: Governance-Rahmenwerk bei verteilter Entwicklung und Integration

Governance spielt bei der Umstrukturierung von Softewareentwicklungsprozessen eine herausragende Rolle [CA11]. Da Softwareprodukte und Integrationslösungen in der Regel Softwarekomponenten enthalten, die von Dritten entwickelt und bezogen wurden, muss der Umgang mittels eines Governance-Rahmenwerks geregelt werden. Dies umfasst (1) eine unternehmensweite und transparente Regelung aller wesentlichen Aspekte im Umgang mit Software von anderen Herstellern, (2) die in diesem Zusammenhang bestehenden Verantwortlichkeiten und (3) die Rechenschaftspflichten. Damit Hersteller von Software unternehmensweit wertschöpfungskettenübergreifende Sicherheitsprozesse einführen können, ist ein Governance-Rahmenwerk erforderlich; dieses sollte in einer Organisation vereinheitlicht und verpflichtend umgesetzt werden. Ein solches Rahmenwerk existiert noch nicht und muss deshalb entwickelt werden. In diesem Rahmenwerk muss beschrieben sein, wie Sicherheitsprozesse organisatorisch umgesetzt werden. Das Rahmenwerk muss hierbei die Verpflichtungen und Verantwortlichkeiten aller einbezogenen Akteure durch klare und transparente Regelungsstrukturen beschreiben.

Es ist aus verschiendenen Gründen unbedingt erforderlich, in einem solchen Rahmenwerk die Steuerung und Kontrolle sowie die Verantwortung in der Führung eines Unternehmens vorzusehen:

- Die Einführung von neuen Sicherheitsprozessen, ob ausschließlich unternehmensintern oder wertschöpfungskettenumfassend, hat für Softwarehersteller eine strategische Dimension. Solche Sicherheitsprozesse haben für Hersteller das Potenzial, die finanziellen Aufwände über dem Softwarelebenszyklus bei Verbesserung des Sicherheitsniveaus zu reduzieren. Vor diesem Hintergrund hat eine solche Entscheidung eine große Relevanz im Wettbewerb mit anderen Herstellern.
- Für bestimmte Kategorien von Kunden ist die Existenz von Sicherheitsprozessen ein immer wichtiger werdender Aspekt bei der Kaufentscheidung. Insbesondere für Hersteller von Software, die in regulierten Branchen eingesetzt wird, ist die Bedeutung von Sicherheitsprozessen besonders groß. Insofern besteht hier ein strategischer Aspekt für Softwarehersteller, der von der Unternehmensführung berücksichtigt werden muss.
- Es ist bekannt, dass Sicherheitsmängel von Software Auswirkungen auf die Börsennotierungen der Hersteller haben können [TW07; Wri11]. Der Schutz von Unternehmenswerten ist eine der wesentlichen Aufgaben des oberen Managements.
- Die Risiken für ein Unternehmen werden bei der Vergabe von Krediten berücksichtigt, gemäß den EU-Richtlinien EG/2006/48 und EG/2006/49 [EU 06a; EU 06b], die aus Basel II hervorgegangen sind. Die Entwicklung von Software mit Sicherheitsmängeln kann für Softwarehersteller insofern riskant sein [Cre11].
- Die organisationsweite Umstellung von Softwareentwicklungsprozessen braucht ein Budget, das von dem oberen Management verantwortet und zur Verfügung gestellt werden muss.
- Die Verbesserung der Anwendungssicherheit durch Sicherheitsprozesse verlangt, dass diese von Softwarearchitekten und Entwicklern team- und abteilungsübergreifend angewendet und umgesetzt werden. Die organisationsweite Einführung von wertschöpfungskettenumfassenden Sicherheitsprozessen impliziert, dass alle an den Softwareentwicklungsprozessen Beteiligten die entsprechenden Vorgaben in abgestimmter Weise implementieren müssen. Hierfür ist eine Führung durch das obere Management erforderlich.
- Durch die Einführung von neuen Sicherheitsprozessen bei der Softwareentwicklung wird sich die herkömmliche Arbeit der Entwickler verändern. Vergleichbar umfassende Veränderungsprozesse sind in der Praxis oftmals durch Widerstände gekennzeichnet, die auf die Bewahrung des *status quo* ausgerichtet sind. Vor diesem Hintergrund sollte die Kontrolle und Steuerung zur Einführung neuer wertschöpfungskettenumfassender Sicherheitsprozesse in der Unternehmensführung verankert sein.

— Zu welchem Zeitpunkt welcher Standard (siehe Abschnitt 4.1) zur Implementierung von wertschöpfungskettenumfassenden Sicherheitsprozessen in einer Organisation ausgewählt wird, kann nur von der obersten Managementebene verantwortet werden.
— Die Einführung von wertschöpfungskettenumfassenden Sicherheitsprozessen muss organisationsweit gesteuert und kontrolliert werden.
— Durch die Verankerung eines Frameworks auf der obersten Managementebene wird die Bedeutung und Ernsthaftigkeit der Umstellung von Sicherheitsprozessen in der Organisation unterstrichen.

Die Ziele des Governance-Rahmenwerks bestehen darin, Unternehmen ein Vorgehensmodell zu liefern, mit dem die bisherige Softwareentwicklung durch die Erweiterung um wertschöpfungskettenumfassende Sicherheitsprozesse verbessert und betrieben werden kann. Dies umfasst die Definition von neuen Rollen mit ihren Zuständigkeiten und Verantwortlichkeiten in der Organisation. Um diese Vorgehensmodelle umsetzen zu können, müssen Hindernisse in der Organisation erkannt und beseitigt werden. Aufgrund der Tatsache, dass bisherige Vorgehensweisen und Gewohnheiten bei der Softwareentwicklung hinterfragt, auf den Prüfstand gestellt und verändert werden müssen, sind Widerstände und Reibungsverluste realistisch. Vor diesem Hintergrund hat Transparenz bei der Führung eine herausragende Bedeutung, so dass alle einbezogenen Akteure die Gründe zur Weiterentwicklung und Umstrukturierung der Softwareentwicklungsprozesse verstehen können. Dies stellt auch Anforderungen an die Metriken, die man zum Management der Weiterentwicklung und Umstrukturierung benötigt.

Zur Steuerung der Einführung neuer wertschöpfungskettenumfassender Sicherheitsprozesse braucht man Metriken, um Fortschritte oder Probleme erkennen zu können. Hierzu müssen zunächst geeignete Metriken entwickelt werden, mit welchen die wesentlichen Aspekte möglichst effektiv, effizient und objektiv gemessen werden können. Sie dienen dem Management und den ausführenden Akteuren dazu, erkennen zu können, ob bzw. wann die angestrebten Ziele erreicht sind. Darüber hinaus sollte das Kontrollinstrumentarium hinreichend differenziert sein, um eine Feinjustierung hinsichtlich einzelner Eigenschaften vornehmen zu können. Das Instrumentarium zur Kontrolle und Steuerung soll auf möglichst viele Abteilungen der Organisation wiederholt angewendet werden können.

Das Governance-Rahmenwerk muss alle für einen Softwarehersteller relevanten Bezugsquellen von Software und Wertschöpfungsketten umfassen. Insbesondere muss das Governance-Rahmenwerk auch Vorschläge enthalten, wie Lieferanten und Bezieher sich auf Zusicherungen hinsichtlich ineinandergreifender Sicherheitsmechanismen abstimmen, wie diesbezügliche Zusicherungen gegeben werden und wie solche Zusicherungen überprüft werden können.

Damit die eigenen Investitionen in die Umstrukturierung von Softwareentwicklungsprozessen zu bestmöglichen Resultaten führen können, ist es erforderlich, dass auch die eigenen Zulieferer ihre Prozesse entsprechend weiterentwickeln und die noch

zu entwickelnden Industriestandards übernehmen (siehe Abschnitt 4.1). Durch die Einbeziehung des obersten Managements in diese Umstrukturierung ergibt sich eine gute Ausgangssituation, andere Softwarehersteller zur Übernahme von Standards zu beeinflussen.

Bei der Entwicklung eines Governance-Rahmenwerks müssen folgende Fragen beantwortet werden:

- Welche Rollen sind in einem solchen Governance-Rahmenwerk erforderlich?
- Welche Prozesse verlangt das Governance-Rahmenwerk?
- Welche spezifischen Prozesse verlangt das Governance-Rahmenwerk für welchen Typ von extern bezogener Komponente?
- Welche Metriken sind für das Governance-Rahmenwerk sinnvoll?
- Wie steigert man die Transparenz bei der Umsetzung des Governance-Rahmenwerks?
- Wie sind die Prozesse des Governance-Rahmenwerks zu dokumentieren?
- Wie soll das Governance-Rahmenwerk ausgestaltet sein, so dass Softwareentwicklungsprozesse möglichst wirtschaftlich umstrukturiert werden können?
- Wie müssen auf der Governance-Ebene Sicherheitsprozesse bei Zuliefererbeziehungen geregelt werden?
- Wie kann die Einhaltung von Zusicherungen der Zulieferer objektiv überprüft werden?

4.3 Herausforderung: Sicherheitsprozesse für Softwareproduktlinien

Die Softwareindustrie steht unter einem massiven Wettbewerbsdruck. Steigerung der Produktivität und Reduktion von Entwicklungszeiten (*Time to Market*) und Entwicklungskosten sind für das langfristige Überleben sehr wichtig. In diesem Zusammenhang hat die Wiederverwendung von bereits entwickelten Softwarekomponenten eine große Bedeutung.

Eine besonderer Rahmen, innerhalb dessen die Wiederverwendung von Softwarekomponenten systematisch geplant und organisiert wird, ist bei Produktlinien gegeben. Produktlinien umfassen verschiedene Ausprägungen eines Softwareprodukts, die auf Basis einer für diese Ausprägungen gemeinsamen Plattform bzw. eines gemeinsamen Kerns entwickelt werden. Plattform bzw. Kern sind dann in allen verschiedenen Produktausprägungen enthalten. Die verschiedenen Produkte einer Produktlinie entstehen dadurch, dass Plattform bzw. Kern an jeweiligen Variationspunkten um verschiedene sogenannte Features erweitert werden. Bei der Planung einer Produktlinie müssen geeignete Variationspunkte erkannt werden, an denen später potenzielle Weiterentwicklungen ansetzen. Gegenstand für solche Variabilitäten in Produktlinien sind hauptsächlich Anforderungen hinsichtlich Funktionalität oder Kompatibilität mit der Umgebung. Nichtfunktionale Anforderungen wie die Sicherheit liegen in der Regel orthogonal zu den Weiterentwicklungsachsen und finden

daher in der semantischen Modellierung von Produktlinien keine natürliche Entsprechung.

Für Hersteller von komplexeren Softwareprodukten spielen sowohl Wiederverwendung und Produktlinien als auch verteilte Entwicklung und Integration eine Rolle. Die Komplexität für *Security by Design* wird gesteigert, wenn Aspekte von Produktlinien und verteilter Entwicklung über Wertschöpfungsketten zu kombinieren sind.

Für die Berücksichtigung von Produktlinienaspekten und Wertschöpfungsketten sind verschiedene Perspektiven relevant.

(1) Für Hersteller von Softwarekomponenten als Lieferanten innerhalb von Wertschöpfungsketten: Bei den von einem Lieferanten entwickelten Softwarekomponenten kann es sich um ein Produkt innerhalb einer Produktlinie handeln. Die Entwicklung von Plattform bzw. Kern sowie Produktausprägungen ist von dem Hersteller so zu planen und durchzuführen, dass die Anforderungen bzgl. Sicherheitsprozesse und Sicherheitseigenschaften der jeweiligen Abnehmer der Softwarekomponenten erfüllt werden. Eine Schwiergkeit besteht hierbei darin, dass die konkreten Anforderungen der potenziellen Abnehmer zum Zeitpunkt des Produktliniendesigns noch nicht vollständig bekannt sind.

(2) Für Hersteller von Softwareendprodukten bzw. Integratoren, die in ihren Produkten Softwarekomponenten verschiedener Hersteller integrieren: Bei einem durch Integration von Komponenten verschiedener Hersteller entstandenen Softwareendprodukt kann es sich ebenfalls um ein Produkt handeln, das im Rahmen einer Produktlinie entstanden ist. Auch hier müssen Sicherheitsprozesse und Sicherheitseigenschaften beim Produktliniendesign so berücksichtigt werden, dass möglichst viele relevante Sicherheitsanforderungen an Produktausprägungen erfüllt werden können. Auch hier besteht das Problem, dass bestimmte Sicherheitsanforderungen von Anwendern zum Zeitpunkt des Produktliniendesigns noch unbekannt sind.

Bei dem Design von Produktlinien und beim Sicherheitsdesign der Plattform muss man von Beginn an mit einer Vielzahl von Sicherheitsanforderungen umgehen können. Diese können sich zwischen verschiedenen Produktausprägungen voneinander unterscheiden. Zur systematischen Behandlung und Verwaltung dieser Sicherheitsanforderungen wurden bereits erste Managementsysteme für Sicherheitsanforderungen in Produktlinien entwickelt [MFMP09; MFMP08a; MFMP08b; MRFMP09]. Eine weitere Schwierigkeit im Zusammenhang mit Produktlinien besteht insbesondere darin, dass für jeweilige Anwendungsfälle die Bedrohungsanalysen und das konkrete Requirements Engineering hinsichtlich Sicherheit erst dann erfolgen können, wenn die Plattform, auf dem die Produktlinie aufsetzt, bereits implementiert ist. Insofern ist es möglich, dass spezielle Sicherheitsanforderungen bei dem Sicherheitsdesign der Plattform nicht berücksichtigt wurden. Es ist dann nicht auszuschließen, dass bestimmte Sicherheitsanforderungen auf Basis der getroffenen Designentscheidungen bzgl. der Plattform nicht einfach umgesetzt werden können. In Einzelfällen kann es

sogar möglich sein, dass getroffene Sicherheitsdesignentscheidungen bei der Plattform und Sicherheitsanforderungen des Produkts in direktem Widerspruch zueinander stehen. Um Sicherheitslücken in Produkten zu vermeiden, ist es insofern immer erforderlich, dass die Sicherheitsanforderungen der Anwendung gegen die Sicherheitseigenschaften der Plattform geprüft werden. Deshalb ist es bei der Behandlung von Produktlinien auch wichtig, dass die typischen Sicherheitsprozesse in der Softwareentwicklung auf die Besonderheiten von Produktlinien angepasst werden. Bei der Umsetzung dieser Prozesse wird eine Unterstützung durch geeignete Werkzeuge äußerst hilfreich sein (siehe Kapitel 3).

Bei dem Design einer Produktlinie muss es unter anderem darum gehen, eine gute Balance zwischen potenziell zu erfüllenden Sicherheitsanforderungen von zukünftigen Ausprägungen und Fragen der Effizienz und Wirtschaftlichkeit zu finden. Bei zu starker Berücksichtigung potenzieller Sicherheitsanforderungen besteht die Gefahr des Overengineerings, so dass die Entwicklungskosten der Produktlinie zu hoch werden und das Einsparpotenzial des Produktlinienansatzes nicht ausgenutzt werden kann.

Produktlinien zeichnen sich dadurch aus, dass sich beim Vorhandensein von vielen Variationspunkten ein sehr großer Raum von möglichen Softwareprodukten ergeben kann. Das bedeutet, dass für *Security by Design* viele verschiedene Ausprägungen behandelt und analysisert werden müssen. Hierzu gibt es bereits Ergebnisse [BRT+13], welche die Sicherheit von solchen Produktausprägungen behandeln, die über Variation von Präprozessoroptionen erreicht werden können. Damit wurde bereits ein erster wichtiger Schritt für *Security by Desgin* bei Produktlinien erzielt, jedoch müssen dieser Arbeit weitere folgen, die nicht auf die Variation von Präprozessoroptionen beschränkt sind und die darüber hinaus auch noch die Probleme der verteilten Entwicklung von Software berücksichtigen.

Zur Berücksichtigung von Sicherheitsprozessen und Sicherheitseigenschaften von Produktlinien bei verteilter Entwicklung muss die Forschung unter anderem die folgenden Fragen beantworten:

- Wie sind die wertschöpfungskettenumfassenden Sicherheitsprozesse bei der Softwareentwicklung unter Berücksichtigung von Produktlinien auszugestalten?
- Wie sind Produktlinien zu designen, damit möglichst alle relevanten Sicherheitsanforderungen mit vertretbarem Aufwand erreicht werden können?
- Wie ist zum Zeitpunkt des Sicherheitsdesigns mit noch unbekannten Sicherheitsanforderungen für Produktausprägungen umzugehen?
- Wie können spezielle Produktausprägungen mit besonderen Sicherheitsanforderungen bei der Produktliniengestaltung identifiziert werden?
- Wie können Sicherheitsanalysewerkzeuge so gestaltet werden, dass sie Gemeinsamkeiten in verschiedenen Produkten effizient ausnutzen, jedoch gleichzeitig auch solche Klassen von Schwachstellen erkennen, die durch Variabilität hervorgerufen werden?

— Wie kann man im Sicherheitsdesign der Produktlinienplattform effektiv und effizient Widersprüche zu später gegebenen Sicherheitsanforderungen von Produktausprägungen identifizieren?
— Wie kann ein Integrator die Sicherheitsanforderungen der Produktlinienplattform in Sicherheitsanforderungen für Komponenten übertragen, die von Zulieferern hergestellt werden?
— Welche Dokumentationsformate braucht man für wertschöpfungskettenumfassende Sicherheitsprozesse unter Berücksichtigung von Produktlinien?

4.4 Herausforderung: Sicherheit bei der Integration großer Systeme

In modernen Unternehmen kommen Softwaresysteme in vielen betrieblichen Arbeitsabläufen zum Einsatz. Sie unterstützen Geschäftsprozesse und machen diese effektiver, produktiver und akkurater. Ohne entsprechende Softwareunterstützung sind heutige Unternehmen nicht mehr wettbewerbsfähig. Ein entscheidender Vorteil von Softwaresystemen ist dann gegeben, wenn sich unterschiedliche Geschäftsprozesse bestimmte Daten teilen können und innerhalb dieser Geschäftsprozesse auf die selben Daten und Funktionen zugegriffen werden kann. Dies wird ermöglicht durch die Integration verschiedener Anwendungen, was auch mit *Enterprise Application Integration* (EAI) bezeichnet wird. Mittels EAI wird es möglich, agil und flexibel auf neue Bedarfe reagieren zu können, indem die vorhandenen Softwaresysteme erweitert oder modifiziert werden. So bietet EAI Unternehmen darüber hinaus auch die Grundlage zur technischen Integration von Geschäftsprozessen über Unternehmensgrenzen hinweg. Die Potenziale von EAI für Unternehmen sind seit längerer Zeit bekannt [Gle05]. Das gilt sowohl für Unternehmen im Bereich der Produktion als auch dem Dienstleistungssektor [Xu11]. Alle Personen, die für die Organisation von informationstechnischen Infrastrukturen in Unternehmen verantwortlich sind, müssen sich mit den Fragen und Problemen der EAI auseinander setzen. Diese Fragen und Probleme entstehen durch den immer größer werdenden Integrationsgrad im Vergleich zu früheren Informationssystemen, die sich auf bestimmte ausgewählte Funktionen und partielle Integration beschränkt haben.

Durch den hohen Integrationsgrad von EAI entstehen typischerweise sehr große und komplexe Systeme, die sehr spezifisch auf die Anforderungen der jeweiligen Anwender zugeschnitten sind. So werden Geschäftsprozesse integriert, welche in ihren jeweiligen Schritten und Ausprägungen die besonderen Anforderungen des jeweiligen Unternehmens erfüllen. Mittels EAI werden auf einer technischen Ebene unterschiedliche Komponenten wie Systeme, Anwendungen, Schnittstellen (z.B. Benutzerschnittstellen) oder Daten, die sehr heterogen sein können, zu komplexen Prozessen integriert. Die Integration ist meistens schwierig und aufwändig, da die Komponenten beispielsweise mit verschiedenen Methoden für verschiedene Systeme entwickelt wurden, sie keine gemeinsamen Schnittstellen unterstützen, oder auf unterschiedlichen Datenmodellen basieren. Komponenten und Subsysteme zu integrieren, die

in sich sehr heterogen sind, verlangt hohen manuellen Aufwand von Entwicklern und Integratoren, der wegen der Heterogenität selten vereinheitlichten, systematischen und strukturierten Vorgehensweisen folgt. Schätzungen zufolge erfordert die Integration heute mehr als 30% der Investitionen, die von Anwendern für ihre IT-Infrastruktur aufgebracht werden [ROB11]. Im Vordergrund steht bei EAI immer die Funktionalität, aus der sich Vorteile und Nutzeneffekte für die anwendende Organisation ergeben.

EAI wird heute intensiv für sogenannte *Enterprise Resource Planning Systeme* (ERP) verwendet, die wichtige Geschäftsprozesse für Unternehmen abdecken [NTD12]. Über ERP-Systeme hinaus findet je nach Bedarf noch Software für das *Customer Relationship Mananagement* (CRM), das *Supply Chain Management* (SCM) oder für unternehmensübergreifende Geschäftsprozesse (B2B) Anwendung, die im Rahmen von EAI miteinander integriert werden. Als Ausgangspunkt für große Softwaresysteme werden in vielen Unternehmen ERP-Universalsoftwareprodukte eingesetzt, die für ein breites Spektrum von Anwendern entwickelt worden sind und über Funktionen wie beispielsweise eine integrierte Datenhaltung, Standardanwendungen (z.B. für Personalangelegenheiten, Verkauf, Buchhaltung, Produktion) und allgemeine Geschäftsprozessimplementierungen verfügen. Darüber hinaus gibt es auch industrie- und branchenbezogene Ausführungen von ERP-Systemen [WXH09]. Alle diese ERP-Systeme bieten für typische wiederkehrende Fragestellungen bzgl. Geschäftsprozessen Lösungen in Form von *Best Practices* oder etablierten Standards und erlauben bedarfsgerechte Spezialisierungen für das jeweilige Unternehmen (*Customization*). Der von den ERP-Universalsoftwareprodukten angebotene Funktionsumfang deckt jedoch in vielen Fällen die Anforderungen und Wünsche der Anwender nicht vollständig ab, so dass zusätzliche Softwareprodukte integriert werden [SS05].

Mit der Bereitsstellung von Diensten im Rahmen von serviceorientierten Architekturen besteht auch die Möglichkeit, Funktionalität zu nutzen, die über das Internet im Rahmen von Diensten zur Verfügung gestellt wird [WL11]. Die Vorschläge zur Integration von Diensten gehen sogar so weit, dass Dienste dynamisch und adaptiv von unterschiedlichen Anbietern eingebunden werden [MRFU11]. Durch die unterschiedlichen Bedarfe, die Dynamik, Flexilibität und die unterschiedlichen technischen Implementierungen der anwenderspezifischen Integration zusätzlicher Komponenten entstehen komplexe Informationssysteme, die sich im integrierten Zustand selbst bei Verwendung der gleichen ERP-Produkte zwischen verschiedenen Anwendern stark voneinander unterscheiden.

Mit der breiten Einführung von EAI steigen jedoch auch die Risiken durch Ausnutzung von Sicherheitslücken für die Anwender erheblich an. Komponenten oder Subsysteme der durch EAI entstandenen Systeme bieten Zugang zu kritischen Informationen. Die durch Integration entstehenden großen Systeme sind für die anwendenden Unternehmen vergleichbar mit einer digitalen Schatzkammer, da sie praktisch sämtliche Informationen der relevanten Geschäftsprozesse umfassen. Die entstehenden Systeme sind sehr komplex, so dass sämtliche Implikationen für die Sicher-

heit nur schwierig zu überschauen sind. Es ist nicht auszuschließen, dass Angreifer über Komponenten oder Subsysteme Zugriff auf Daten bekommen können, was den Sicherheitsregeln eines Unternehmens widerspricht. Die Ansatzpunkte für Angriffe können insbesondere an den Schnittstellen zwischen den integrierten Komponenten entstehen. Sowohl für die initiale Integration als auch für den kompletten Lebenszyklus existieren keine expliziten systematischen Vorgehensweisen und Methoden im Sinne von *Security by Design*. In der Praxis spielen Fragen der IT-Sicherheit bei der Integration keine wesentliche Rolle [KT09]. Untersuchungen zeigen, dass bei der Integration in der Praxis Sicherheitslücken immer wieder durch sehr einfache und vermeidbare Fehler entstehen [Kal12].

Vorhandene Systematiken zur Integration beziehen sich auf den Architekturlevel und beschreiben, wie Komponenten in die Gesamtumgebung einzubetten sind und wie diese interagieren; andere Systematiken beschreiben Koordinierungsmodelle und die Anwendung von Werkzeugen für die Integration von Daten und komplexen Prozessen [ROB11; Gle05; HN08]. Die vorliegenden Arbeiten beinhalten jedoch keine umfassenden Sicherheitsprozesse für die Integration. Wenn Sicherheit betrachtet wird, dann beschränkt sich dies meist auf die Berücksichtigung von Sicherheitsstandards wie z.B. die Standards zu Web Service Security [OAS12] als wichtige technische Grundbausteine zur sicheren Komposition von netzbasierten Diensten. Darüber hinausgehende Vorschläge zur Verbesserung der Sicherheit auf Basis kompositionaler Beschreibungen von Anforderungen und Zusicherungen der zu integrierenden Komponenten, wie z.B. mittels sogenannter Compositional Security Contracts in [KT09], bieten vielversprechende Ansätze, sie sind jedoch weder hinreichend ausgearbeitet noch in die Praxis transferiert.

Die Rahmenbedingungen für *Security by Design* bei der Integration großer Systeme hängen stark von den Integrationsmodellen ab. Das Spektrum des Möglichen ist hier sehr groß: Es reicht von der Integration von lokal vorhandenen Softwarekomponenten, an deren Entwicklung das anwendende Unternehmen selbst beteiligt war, über die Integration von lokal installierter Fremdsoftware bis hin zur Einbindung von Softwarekomponenten in Form von Diensten, die von anderen zum Zugriff über das Internet angeboten werden, z.B. als Cloud-Dienste. Bei der Verwendung von Diensten fremder Anbieter steigt das Risiko für den Anwender, da Daten zu Dienstanbietern gelangen, deren Plattformen zusätzlich noch von vielen anderen Kunden, z.B. potenziellen Angreifern genutzt werden, die ggf. Schwachstellen zum Zugriff auf die eigenen Daten ausnutzen könnten. Je nach Integrationsmodell unterscheiden sich die Möglichkeiten für *Security by Design* stark voneinander. Unabhängig von dem verwendeten Integrationsmodell sollten bei der Entwicklung großer Systeme Vorgehensweisen und Methoden zur Anwendung kommen, so dass die Sicherheit der entstehenden Systeme über den kompletten Lebenszyklus hin verbessert und aufrechterhalten wird. Hierbei müssen auch Agilität, Flexibilität und wirtschaftliche Umsetzbarkeit für zukünftige Erweiterungen und Anpassungen bei der Integration berücksichtigt werden. Um dies zu bewerkstelligen müssen zunächst die für die ver-

schiedenen Integrationsmodelle passenden Verfahren entwickelt werden. In diesem Zusammenhang müssen unter anderem die folgenden Herausforderungen bewältigt werden:

- Wie können Sicherheitsanforderungen von Komponenten erfasst, verständlich und verwertbar ausgedrückt werden?
- Wie sind Zusicherungen hinsichtlich Sicherheit zu erfassen sowie verständlich und verwertbar auszudrücken?
- In welcher Tiefe müssen Sicherheitsanforderungen und Zusicherungen behandelt werden, dass sie in einem wirtschaftlichen Rahmen für zukünftige Änderungen und Modifikationen der großen Systeme angewendet werden können?
- Wie kann man aus den Sicherheitsanforderungen der zu implementierenden Gesamtprozesse und der jeweiligen Komponenten systematisch Entscheidungen bzgl. Architektur und Design der bei der Integration zu entwickelnden verbindenden Technik ableiten und diese umsetzen?
- Wie sind die Prozesse zu etablieren, damit bei einer Integration von Funktionalität als Dienst aus technischen Modifikationen auf einer Seite Sicherheitsimplikationen für die restlichen Komponenten erkannt werden und dies möglichst bevor die technischen Modifikationen implementiert werden?
- Wie müssen bestehende Vorgehensweisen zur Planung und Koordinierung von Integrationsarbeiten für *Security by Design* ergänzt werden?
- Wie können für die dynamische Integration von Diensten Sicherheitsaspekte berücksichtigt werden?
- Wie sind bestehende Dienstbeschreibungen für die dynamische Integration anzupassen, damit keine Dienste ausgewählt werden, die gegen Sicherheitsanforderungen des restlichen Systems verstoßen?

4.5 Herausforderung: Zusicherungen mittels Sicherheitsprozessen

Für Anwender wird Sicherheit von Software ein wichtiger werdendes Kriterium bei der Kaufentscheidung. Das gilt insbesondere für Anwender mit großer Marktmacht, wie z.B. Behörden oder andere staatliche Institutionen, sowie für Anwender aus bestimmten Branchen, für die strengere Regeln gelten und für deren Einhaltung Organisationen oder das Management haften müssen.

Ein Anwender interessiert sich in diesem Zusammenhang immer für die Sicherheit des kompletten Endproduktes, auch wenn das Endprodukt Komponenten verschiedener Hersteller und Zulieferer enthält. Aus der Perspektive des Anwenders ist immer der Hersteller des End- oder Gesamtproduktes für dessen Eigenschaften verantwortlich, denn schließlich ist er derjenige, der die Komponenten von dritten Anbietern ausgewählt hat. Entsprechend müssen Hersteller bzw. Integratoren auch Fragen der Sicherheit bei der Entscheidung bzgl. Zulieferern bzw. der zu integrierenden Softwarekomponenten berücksichtigen. Fragen zu Sicherheit sind für Integratoren oder

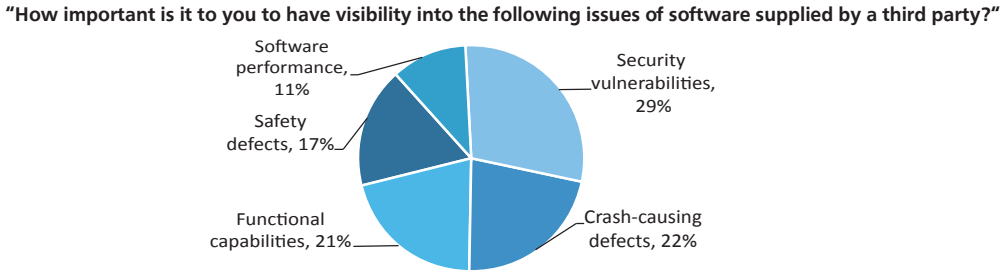


Abbildung 7: Die Bedeutung von Sicherheit bei verwendeten Softwarekomponenten, die von anderen Herstellern entwickelt wurden (Quelle: [For11b]): Grundlage ist hier dieselbe Befragung wie in Abbildungen 5 und 6.

Hersteller von Softwareendprodukten bei dieser Entscheidung sehr wichtig; dies belegen die in Abbildung 7 gezeigten Ergebnisse von Umfragen in der Softwareindustrie [For11b].

Bezieher von Softwarekomponenten brauchen von ihren Zulieferern Aussagen, anhand derer sie das Sicherheitsniveau der Komponenten einschätzen können. Diese Aussagen sollten über einen angemessenen Detaillierungsgrad verfügen und eine Verbindlichkeit haben. Aussagen zum absoluten Sicherheitsniveau von Softwareprodukten sind jedoch in der Praxis schwierig zu treffen, insbesondere wenn Softwareprodukte durch Komposition von Teilen verschiedener Hersteller entstehen. Aussagen zu Sicherheitsprozessen bei der Herstellung bieten eine Alternative, um Herstellern, Integratoren oder Anwendern Zusicherungen zu geben, dass Sicherheitsaspekte bei der Herstellung von Software berücksichtigt wurden. Mittels solcher Zusicherungen sollten Hersteller Aussagen darüber treffen, in welchem Umfang und mit welcher Genauigkeit und Sorgfalt sie bestimmte Systematiken anwenden, um Sicherheit zu gewährleisten. Solche Zusicherungen sind insbesondere dann hilfreich, wenn sie im Zweifel im Rahmen von Audits möglichst eindeutig überprüft werden können und wenn aus einem nachweisbaren Verstoß gegen Zusicherungen negative Konsequenzen für denjenigen drohen, der gegen seine Zusicherungen verstoßen hat.

Zusicherungen auf Basis von Sicherheitsprozessen treffen also eine indirekte Aussage zur Sicherheit von Software. Die Zusicherung, dass bei der Herstellung bestimmte Sicherheitsprozesse eingehalten werden, lässt auf ein höheres Sicherheitsniveau schließen. Solche Zusicherungen auf der Basis von Sicherheitsprozessen, beispielsweise durch Zertifizierung von Sicherheitsmaßnahmen in der Herstellung, stehen der Zertifizierung von Produkten gegenüber, z.B. auf der Basis von *Common Criteria*. Bei dieser Zertifizierung wird eine direkte Aussage über die Sicherheit von Softwareprodukten für verschiedene Zusicherungsniveaus – bei Common Criteria sind das die sogenannten Evaluation Assurance Level – getroffen. Auch wenn eine direkte Aussage zur Sicherheit von Softwareprodukten wie z.B. mittels *Common Criteria* zunächst geeigneter erscheint als indirekte Zusicherungen auf der Basis von Herstellungsprozessen, so liefert die praktische Erfahrung dennoch einige Argumente, die für den indirekten Ansatz bzw. gegen den direkten Ansatz sprechen. Nach [Jac06] sind die Zertifizierungen nach *Common Criteria* zu schwerfällig, langwierig und sehr teu-

er. Zertifizierungen nach *Common Criteria* werden deshalb nur in Nischenbereichen angewendet, insbesondere in Fällen, bei denen es besonders hohe Sicherheitsanforderungen gibt, z.B. auf Grund von Auflagen durch Regulierung. Gemäß den Angaben des BSI in [BSI12] wird die Zertifizierung nach *Common Criteria* für Softwareprodukte wie Betriebssysteme, Datenbanken, Firewalls, PC-Sicherheitsprodukte, VPN-Produkte, E-Mail-Server und Signaturanwendungskomponenten verwendet. Für Anwendungssoftware vermeiden Hersteller Aufwand und Kosten durch *Common Criteria*. Hierbei spielen eine Reihe von grundlegenden Problemen eine Rolle, die sich durch *Common Criteria* ergeben und die im Widerspruch zu den Anforderungen von softwareherstellenden Unternehmen stehen. Softwarehersteller stehen meist unter einem hohen Zeitdruck, ihre Produkte auf den Markt zu bringen. Dieser Anforderung stehen die erheblichen Verzögerungen durch *Common Criteria* gegenüber. Hinzu kommt, dass Softwareprodukte, wenn sie auf dem Markt sind, meistens kontinuierlich in kleinen Schritten weiter entwickelt werden, die dann im Rahmen von Updates den Benutzern zur Verfügung gestellt werden. Direkte Zertifizierungen wie durch *Common Criteria* implizieren jedoch, dass die Zusicherung nicht mehr gültig ist, wenn es ein Update oder eine neue Softwareversion eines Produktes gibt. Hersteller müssen für jedes Update und jede neue Softwareversion immer wieder den langwierigen und teuren Zertifizierungsprozess durchlaufen. Eine weitere wichtige Eigenschaft von *Common Criteria*, die im Widerspruch zu den Anforderungen der Hersteller von Anwendungssoftware steht, liegt darin begründet, dass *Common Criteria* keine flexiblen Kompositionen unterstützt, wie sie sich z.B. durch das Zusammensetzen eines Softwareprodukts aus Komponenten verschiedener Hersteller ergeben. Zusicherungen bzw. Aussagen hinsichtlich Sicherheitseigenschaften sind jedoch gerade für solche Produkte in der Praxis eine sehr wichtige Anforderung, da ein sehr großer Anteil von realen Softwareprodukten Komponenten verschiedener Hersteller integriert.

Somit ist die Welt der Softwareprodukte hinsichtlich Zusicherungen von IT-Sicherheitseigenschaften heute zweigeteilt: Für Spezialprodukte mit hohen Sicherheitsanforderungen gibt es Zertifikate, die in direkter Weise Aussagen über Sicherheitseigenschaften von Produkten treffen. Für typische Anwendungssoftware ohne spezielle Anforderungen gibt es solche Aussagen nicht, so dass es für Hersteller von Endprodukten, Integratoren und Anwender keine Zusicherungen gibt, auf die sie sich beziehen können.

Mit Zusicherungen hinsichtlich der bei der Herstellung verwendeten Sicherheitsprozesse würde sich diese Situation verbessern lassen. Es wäre möglich, dass eine solche Zusicherung auch dann noch gültig ist, wenn ein Produkt mit den entspechenden Sicherheitsprozessen weiter entwickelt wird. Ebenfalls wäre es möglich, dass Zusicherungen, die sich auf die Herstellungsprozesse beziehen, auch bei der Kompositionen zu komplexen Produkten unter entsprechenden Bedingungen noch gültig bleiben können. Somit könnte genau in den Fällen ein Gewinn erzielt werden, an

denen andere Zertifizierungsmethoden wie z.B. mittels *Common Criteria* die Anforderungen der Praxis nicht erfüllen können.

Auch wenn sich mit den Zusicherungen auf der Basis von angewendeten Sicherheitsprozessen keine direkten Aussagen zu Sicherheitseigenschaften erzielen lassen, können indirekte Ansätze sehr wertvoll sein, da sie bei vielen Produkten Zusicherungen und Aussagen geben können, bei denen es heute keine verwertbaren Aussagen zur Sicherheit gibt. Darüber hinaus haben, wie in Abschnitt 2.4 beschrieben wurde, Untersuchungen belegt, dass sich durch die systematische Anwendung von Sicherheitsprozessen die Sicherheit von Softwareprodukten deutlich verbessert hat.

Betrachtet man dies nun vor dem Hintergrund von verteilten Entwicklungsprozessen, so sind für Hersteller von Softwarekomponenten Zusicherungen auf der Basis von Sicherheitsprozessen gegenüber Herstellern von Endprodukten möglich. Hierfür muss ein entsprechender Rahmen entwickelt werden, in dem man für verschiedene Produkte sinnvolle und klar beschreibbare Schritte (z.B. Methoden für Requirements Engineering, Designmethoden, Sicherheitstests) sowie besondere Kriterien für das jeweilige Vorgehen (z.B. Berücksichtigung von bestimmten relevanten Schwachstellensammlungen wie etwa OWASP `http://www.owasp.org` im Zusammenhang mit Web-Anwendungen, die bei den Tests berücksichtigt werden; Häufigkeiten von Tests; Anwendung von anerkannten Tools, die Entwickler bei der Programmierung dahingehend unterstützen, indem sie bestimmte Programmierfehler vermeiden) vorschreibt. Darüber hinaus ist bei dem zu entwickelnden Rahmen wichtig, dass wesentliche Teile des Sicherheitsprozesses auditierbar sein sollten. Durch die Auditierbarkeit der Einhaltung von Zusicherungen wird ermöglicht, den Zusicherungen eine notwendige Verbindlichkeit zu verleihen. Zulieferer könnten sonst einfach behaupten, dass sie bestimmte Prozesse durchführen, ohne dies tatsächlich zu tun.

Es sollte für die Verbindlichkeit genügen, wenn der Aufwand zur Umgehung der auditierbaren Sicherheitsprozesse ungefähr so hoch wäre wie die Umsetzung der Sicherheitsprozesse. Andererseits sollte die Lösung für auditierbare Sicherheitsprozesse für den Zulieferer auch dahingehend sicher sein, dass keine Verstöße gegen Zusicherungen konstruiert werden können, wenn alle Zusicherungen korrekt umgesetzt wurden.

Für den Rahmen hinsichtlich Zusicherungen und Auditierbarkeit müssen unter anderem folgende Probleme bewältigt werden:

- Wie können für verschiedene Softwarekomponenten und verschiedene Anwendungsbereiche in effizienter Weise die relevanten Zusicherungen ermittelt werden?
- Wie kann man überprüfen, dass für den Anwendungsbereich die relevanten Zusicherungen identifiziert wurden?
- Wie können Zusicherungen präzise ausgedrückt werden?
- Wie kann man sicherstellen, dass Zulieferer und Integratoren bei den Zusicherungen die gleiche Sprache sprechen?

- Welche Rückwirkungen haben Zusicherungen auf die Ausgestaltung von Sicherheitsprozessen? (Bei verschiedenen denkbaren Varianten von Sicherheitsprozessen ist möglicherweise ein solcher vorzuziehen, bei dem die Erfüllung von Zusicherungen am wirtschaftlichsten ist.)
- Wie können Zusicherungen gegeben werden, so dass Einhaltungen bzw. Verstöße überprüfbar sind?
- Wie können Verletzungen von Zusicherungen zweifelsfrei erkannt werden?
- Wie lassen sich Verletzungen zweifelsfrei ihrem Verursacher zuordnen?
- Wie lassen sich Verletzungen und Einhaltungen von Zusicherungen effizient überprüfen?
- Wie kann man auditierbare Zusicherungen gegen Betrug absichern?
- Wie bringt man Zusicherungen und Auditierbarkeit von Zusicherungen in Einklang mit den anderen Lösungen für wertschöpfungskettenumfassende Sicherheitsprozesse?
- Wie können Sicherheitsprozesse eines Zulieferers über dem gesamten Lebenszyklus auch nach Lieferung von Softwarekomponenten überprüft werden?
- Wie kann man durch Toolunterstützung Zusicherungen gewinnen?
- Wie müssen Zusicherungen auf Basis von angewendeten Sicherheitsprozessen erneuert werden, wenn sich Methodik und Werkzeuge weiterentwickeln?

5. SECURITY BY DESIGN FÜR LEGACY-SOFTWARE

Robert C. Seacord, Daniel Plakosh und Grace A. Lewis verwenden in ihrem Buch über Legacy-Software [SPL03] den Begriff *Legacy Krise*, um die zunehmenden Herausforderungen bezüglich Legacy-Software eindringlich darzustellen. Die Entscheidung, ob es ökonomisch ist, eine Bestandssoftware wieder bzw. weiter zu verwenden oder ob die geforderte Funktionalität im Extremfall komplett neu programmiert werden muss, hat viele Dimensionen; genannt seien hier nur Vollständigkeit und Qualität der Dokumentation, Plattformabhängigkeit, Programmiersprachenabhängigkeit und der Vergleich zwischen Soll und Ist des erreichten Sicherheitsniveaus. Ein Mindestsicherheitsniveau ist eine notwendige Voraussetzung für die Wieder- bzw. Weiterverwendung von Software.

Die Vision dieses Kapitels fokussiert die Sicherheitsrevision von Legacy-Software:

> *Notwendige Bedingung für die Wieder- oder Weiterverwendung von Legacy-Software ist ihre IT-Sicherheitsrevision: Nur bei adäquat hohem Sicherheitsniveau für ihr Einsatzgebiet darf Legacy-Software zum Einsatz kommen. Als Entscheidungsgrundlage müssen plausible Aussagen über das vorhandene IT-Sicherheitsniveau getroffen werden können. Bei Wieder- oder Weiterverwendung muss die Software in den Sicherheitslebenszyklus eingeführt werden. Für die Weiterverwendung existierender Software wird es wesentlich einfacher als heute möglich sein, diese auf ein höheres Sicherheitsniveau zu bringen.*

5.1 Herausforderung: Aussagen zur Sicherheit von Legacy-Software

Aussagen zur Sicherheit von Legacy-Software werden angesichts des zunehmenden Bedarfs der Integration von Legacy-Software (vergleiche [SPL03], Kapitel 4 und 5) dringend benötigt. Ob das Sicherheitsniveau von Legacy-Software tatsächlich ermittelt werden kann, ist offen: Ein einziger unentdeckter Programmierfehler kann sich Jahre später als sicherheitsrelevant herausstellen. Dies bedeutet nicht nur, dass Software grundsätzlich unter Unsicherheit betrieben wird, sondern es wird sogar argumentiert, dass es grundsätzlich unmöglich ist, die Sicherheit von Software zu ermitteln [Bel06].

Auch wenn das Sicherheitsniveau von Software nicht intersubjektiv und bis ins Detail bestimmbar ist, dann muss es mindestens plausibel abschätzbar sein, um eine Risikoabwägung durchführen zu können. Nur so kann für Legacy-Software entschieden werden, ob sie weiter oder in neuem Kontext bei gegebenem Sicherheitsmindestniveau eingesetzt werden darf.

Bestehende Ansätze zur Ermittlung des Sicherheitsniveaus gehen in unterschiedliche Richtungen und es gibt kein Messverfahren, das als Stand der Technik und Forschung akzeptiert ist.

Auf der Quellcodeebene seien drei Ansätze genannt, die sich methodisch unterscheiden:

- BogoSec (*source code security quality metrics*) [KS06] verwendet für die Quellcodeanalyse instrumentierte Testtools, die in Kombination angewendet werden und aus denen ein aggregiertes Sicherheitsniveau errechnet wird.
- Die Strukturanalyse des Quellcodes nach [CCZ08] erzeugt Aussagen auf Basis der durchgehenden Einhaltung von Programmierprinzipien.
- Michael A. Howard schlägt wiederum eine gänzlich andere Methodik vor, nämlich ein vergleichendes *Code Review* [How06]: Durch ein experimentelles Setting mit zwei Entwicklungsteams schätzt er je nach Überdeckungsgrad der gefundenen Schwachstellen die Anzahl der noch unentdeckten Schwachstellen ab.

Liegt die Software nicht als Quellcode vor, dann ist die Abschätzung des Sicherheitsniveaus offensichtlich eine noch härtere Herausforderung [PC10; Sav10]. Zu prüfen wäre beispielsweise, ob das – entsprechend angepasste – experimentelle Setting [How06] hier ebenfalls ein Kandidat für ein Messverfahren ist. Kann *Software Penetration Testing* [ASM05] so modifiziert werden, dass es auch für Legacy-Software angewendet werden kann und Aussagen zum Sicherheitsniveau hiermit ermöglicht werden? Können einschlägige Assessment Tools [Boo09] so angepasst werden, dass sie Aussagen über das erreichte Sicherheitsniveau erlauben?

Angesichts der genannten – wenn auch vielversprechenden – und zugleich verschiedenartigen ersten Ideen steht die Forschung bei der Frage der Messbarkeit des Sicherheitsniveaus von Legacy-Software ganz am Anfang. Es besteht erheblicher Forschungsbedarf.

Mehrere wesentliche Fragen sind offen, wie beispielsweise:

- Welche Messverfahren sind plausible Kandidaten für Aussagen über das Sicherheitsniveaus von Legacy-Software?
- Sind die gefundenen Aussagen über das IT-Sicherheitsniveau leicht kommunizierbar und eine echte Entscheidungshilfe bezüglich Wieder- und Weiterverwendung von Legacy-Software im Hinblick auf Sicherheit?
- Wie hoch ist der Aufwand zur Messung (Zeit, Ressourceneinsatz)?
- Wann ist die Messung praktikabel durchführbar?
- Ist die Messung robust, valide und intersubjektiv wiederholbar?

5.2 Herausforderung: Legacy-Software in Sicherheitslifecycle überführen

Legacy-Software, die wieder- oder weiterverwendet werden soll und sich noch nicht im Sicherheitslifecycle befindet, muss dort eingeführt werden. Besonders wichtig ist

die vollständige Integration von Legacy-Software in den Prozess des systematischen Nachverfolgens, Überwachens und Überprüfens von bekannten Schwachstellen (z. B. der systematischen *Common Weakness Enumeration* (CWE) [MIT13]).

Ein nicht zu unterschätzendes Problem ist die Frage, wie im jeweils verwendeten Lifecycle Einstiegspunkte für Legacy-Software identifiziert werden können, so dass Sicherheitsbetrachtungen und -maßnahmen nach einer Einführungsphase integriert für die Gesamtsoftware möglich werden. Ein erster Ansatz für solche Einstiegspunkte ist durch die *Legacy Roadmap* von CLASP [Gra06] gegeben.

IBM hat mit der *IBM Internet Security Systems Product Lifecycle Policy* [IBM06] ein Regelwerk für Sicherheitsaspekte eigener Software vorgelegt, die den Vorteil hat, dass Legacy-Sicherheitsaspekte bereits bei der Erstellung der Software berücksichtigt sind.

Um Legacy-Software systematisch in den Sicherheitslifecycle einführen zu können, müssen zumindest folgende Fragen herstellerunabhängig geklärt werden:

- Wie kann Software in Hinblick auf sichere Wieder- und Weiterverwendung bereits bei ihrer Entwicklung vorbereitet werden?
- Wie können Policies für die Wieder- und Weiterverwendung von Altsoftware von Herstellern formuliert werden, die es den weiteren Glieder der Lieferkette erleichtern die Altsoftware zu integrieren?

5.3 Herausforderung: Erhöhung der Sicherheit von Legacy-Software

Software, die nur wenig oder überhaupt nicht unter Berücksichtigung von Sicherheit erstellt wurde und trotzdem weiterverwendet werden soll, muss häufig auf ein (höheres) Sicherheitsniveau gebracht werden. Um das Sicherheitsniveau von Legacy-Software zu erhöhen gibt es diverse Vorschläge. Welche davon effektiv und effizient sind, ist derzeit offen. Eine systematische Analyse und Vergleich ist hier dringend notwendig, ggf. auch Verbesserungen.

Selbstredend stehen bei verfügbarem Quellcode die meisten Optionen zur Härtung zur Verfügung, insbesondere wenn dieser sehr gut dokumentiert ist. Das Spektrum reicht von aufwändiger Analyse und anschließender Sicherheitshärtung durch Menschen (*Source Code Review*) bis zu vollautomatischer Härtung durch Quellcodeersetzungen. Aus ökonomischer Sicht sind letztere besonders interessant. Exemplarisch seien Maßnahmen auf verschiedenen Ebenen genannt:

- Inkrementelle Typsicherheit: Die Typsicherheit bestehender Programme zu erhöhen ist ein erster sinnvoller Schritt. *Gradual Typing* fängt beim unsicherem Programm an und fügt inkrementell Typsysteme hinzu [ST07].
- Programmiersprachenbezogene Härtung: Quellcode-bezogene Maßnahmen seien hier an zwei Beispielen gezeigt, eine für C, eine für Java. CCURED [NCH+05] erhöht durch Code-Ersetzungen sicherheitskritischer Programmteile die Speicher-

und Typsicherheit von C-Quellcode. Zur Härtung von Java-Quellcode fokussiert [MLD08] einen aspektorientierten Ansatz mittels *Hardening Patterns*.

- Erweiterungen zur Durchsetzung von Security Policies können für Legacy-Software durch spezifische Programmanalysetools [GJJ06] unterstützt werden. Ein Beispiel ist die automatische Code-Ersetzung [Ham06], die auf *Managed Code* des .NET-Frameworks angewendet wird. Ein weiteres Beispiel ist die Härtung von Sicherheitspolicies bei *Web Services* [MOA11] mittels automatischer Erzeugung von BPEL-Aspekten (*Business Process Execution Language*).
- Runtime Monitoring vermag Legacy-Komponenten zu kapseln und kann somit sicher stellen, dass sie gewisse Policies erfüllen [Bod12].

Übliche Techniken zur sicheren Integration von Black-Box-Legacy-Software [SM99] wie die Analyse und Verhinderung von Systemaufrufen (wie z. B. [LRB+05] und [RHJS05]), Wrapper, Sandboxes, Firewalls und Instrumentierung (z.B. durch Monitore) erfahren derzeit eine vielversprechende Ergänzung durch das Tool *SecondWrite* [OAK+11], welches Code für *Black Box Executables* an sicherheitskritischen Stellen auf der unteren Systemebene mit sicherem Code überschreibt.

Das Sicherheitsniveau von Legacy-Software zu erhöhen ist durchaus möglich; wie gezeigt stehen eine ganze Reihe von Maßnahmen zur Verfügung. Folgende Fragen bedürfen der Klärung:

- Wie kann mit wenig Aufwand entschieden werden, ob eine Härtung sich rentieren wird, oder ob beispielsweise die komplette Neuprogrammierung zielführender wäre?
- Wie kann Legacy-Software kategorisiert werden, so dass den resultierenden Kategorien passende Maßnahmen zur Härtung zugeordnet werden können? Kategorien könnten beispielsweise Programmiersprachen, verwendete Softwaretechnik, Alter der Software aber auch Reifegrad und Vollständigkeit der Dokumentation sein.

6. DIE ZUKUNFT MIT SECURITY BY DESIGN

Wollen wir tatsächlich täglich Nachrichten zu neuen Sicherheitslücken und Angriffen lesen? Sollen wir weiterhin Software einsetzen, bei der Sicherheit in der Herstellung keine wesentliche Rolle gespielt hat, obwohl Computer und Software immer relevanter für viele Bereiche unseres Alltags werden? Wie lange wollen wir dieses Hase-und-Igel-Spiel zwischen Hackern und Herstellern noch erdulden, dessen Leidtragende eigentlich immer die Anwender sind? Den *status quo* zu ändern, liegt in den Händen der Hersteller, Anwender, der Gesellschaft und auch der Politik.

Ein vielversprechender Ausweg aus dieser Situation ist *Security by Design*. Die Geschichte zeigt, dass Produktionsprozesse auch schon an anderen Stellen erfolgreich verändert werden konnten: Die Chemieindustrie leitet ihre Abwässer nicht mehr ungeklärt in Flüsse und alle PKWs verursachen durch reduzierte Schadstoffemissionen mittlerweile geringere Belastungen für die Umwelt. Vergleichbare Änderungen sollten auch für die Produktionsprozesse sicherer Software möglich sein.

Security by Design zeichnet sich darüber hinaus dadurch aus, dass es für alle involvierten Akteure Vorteile bietet: Software wird sicherer, die Risiken werden geringer, Kosten der Herstellung und Wartung werden reduziert und die herstellenden Unternehmen gewinnen Wettbewerbsvorteile. Sicherheit kann ein wichtiger Mehrwert im Softwareherstellungsprozess werden.

In der Zukunft wird es darum gehen, die entscheidenden Fragen rund um *Security by Design* zu erforschen und verwertbare Lösungen zu entwickeln. Hier sind Wirtschaft, Forschung und Politik gefragt. Konzerne sollten die Vorreiterrolle übernehmen, da die oftmals mittelständisch geprägten Hersteller von Software nicht aus eigener Kraft in der Lage sind, ihre Produktionsprozesse umzustellen.

Der Trend- und Strategiebericht gibt mit seinen Visionen und Idealbildern Richtungen vor, in die sich *Security by Design* entwickeln kann bzw. muss. Zusätzlich beschreibt der Bericht Herausforderungen, mit denen man sich auf dem Weg dorthin auseinander zu setzen hat, und Probleme, die gelöst werden müssen. Diese Visionen und Herausforderungen werden die Forschungsagenda der Cybersicherheit in den kommenden Jahren prägen.

Es braucht einen engen Schulterschluss zwischen Softwareindustrie, Forschung und Politik, um zielgerichtet und anwendungsorientiert verwertbare Ergebnisse produzieren zu können und diese in die praktische Softwareherstellung zu transferieren.

7. ANHANG: LITERATURVERZEICHNIS

LITERATUR

[AAS10] Alberts, C.; Allen, J. ; Stoddard, R.: *Integrated measurement and analysis framework for software security.* White Paper, SEI CERT,http://www.cert.org/archive/pdf/10tn025.pdf, 2010

[AAS12] Allen, J.; Alberts, C. ; Stoddard, R.: *Deriving Software Security Measures from Information Security Standards of Practice.* White Paper, SEI CERT,http://www.sei.cmu.edu/library/assets/whitepapers/derivingsecuritymeasures.pdf, 2012

[Abe10] Aberdeen Group: *Security and the Software Development Lifecycle: Secure at the Source.* http://www.microsoft.com/en-us/download/confirmation.aspx?id=6968, 2010

[Ado13] Adobe Systems Incorporated: *Secure Product Lifecycle.* http://www.adobe.com/de/security/splc/. Version: 2013

[AKGL10] Apel, Sven; Kästner, Christian; Größlinger, Armin ; Lengauer, Christian: Type safety for feature-oriented product lines. In: *Automated Software Engineering* 17 (2010), September, Nr. 3, S. 251–300

[ASM05] Arkin, Brad; Stender, Scott ; McGraw, Gary: Software penetration testing. In: *IEEE Security & Privacy* 3 (2005), Nr. 1, S. 84–87

[Bai12] Baize, Eric: Developing Secure Products in the Age of Advanced Persistent Threats. In: *IEEE Security & Privacy* 10 (2012), Nr. 3, S. 88–92

[Bau13] Bauhaus-Projekt: *Software-Architektur, Software-Reengineering und Programmverstehen.* http://www.iste.uni-stuttgart.de/ps/projekt-bauhaus.html. Version: 2013

[BBMM10] Bruch, Marcel; Bodden, Eric; Monperrus, Martin ; Mezini, Mira: IDE 2.0: collective intelligence in software development. In: *Proceedings of the FSE/SDP workshop on Future of software engineering research (FoSER '10)*, 2010

[BDL06] Basin, David; Doser, Jürgen ; Lodderstedt, Torsten: Model driven security: From UML models to access control infrastructures. In: *ACM Trans. Softw. Eng. Methodol.* 15 (2006), Januar, Nr. 1, S. 39–91

[Bel06] Bellovin, S.M.: On the Brittleness of Software and the Infeasibility of Security Metrics. In: *IEEE Security & Privacy* 4 (2006), Nr. 4, S. 96

[BHLM13] Bodden, Eric; Hermann, Ben; Lerch, Johnannes ; Mezini, Mira: *to appear: Reducing Human Factors in Software Security Architectures.* http://www.future-security2013.de/. Version: 2013

[BKA11] BKA (Bundeskriminalamt): *Wirtschaftskriminalität — Bundeslagebild 2010.* http://www.bka.de/nn_193360/DE/Publikationen/JahresberichteUndLagebilder/Wirtschaftskriminalitaet/wirtschaftskriminalitaet__node.html?__nnn=true, 2011

[BKA12] BKA (Bundeskriminalamt): *Cybercrime — Bundeslagebild 2011.* http://www.bka.de/DE/Publikationen/JahresberichteUndLagebilder/Cybercrime/cybercrime__node.html?__nnn=true, 2012

[BMQS05] Beth, Thomas; Müller-Quade, Jörn ; Steinwandt, Rainer: Cryptanalysis of a practical quantum key distribution with polarization-entangled photons. In: *Quantum Information & Computation* 5 (2005), Nr. 3, S. 181–186

[BMW12a] BMWi (Bundesministerium für Wirtschaft und Technologie): *Monitoring-Report Digitale Wirtschaft 2012 — MehrWert für Deutschland.* http://www.bmwi.de/BMWi/Redaktion/PDF/Publikationen/it-gipfel-2012-monitoring-report-digitale-wirtschaft-2012-langfassung,property=pdf,bereich=bmwi2012,sprache=de,rwb=true.pdf, 2012

[BMW12b] BMWi (Bundesministerium für Wirtschaft und Technologie): *Nationaler IT-Gipfel 2012: digitalisieren_vernetzen_gründen (Essener Erklärung).* http://www.bmwi.de/BMWi/Redaktion/PDF/Publikationen/it-gipfel-2012-essener-erklaerung,property=pdf,bereich=bmwi2012,sprache=de,rwb=true.pdf, 2012

[Bod10] Bodden, Eric: Efficient Hybrid Typestate Analysis by Determining Continuation-Equivalent States. In: *ICSE '10: International Conference on Software Engineering*, 2010, 5–14

[Bod12] Bodden, Eric: *Project RUNSECURE.* http://www.ec-spride.tu-darmstadt.de/csf/sse/projects_sse/emmy_noether/emmy_noether.en.jsp, 2012

[Boo09] Booz Allen Hamilton: *Software Security Assessment Tools Review*, März 2009

[BPW07] Backes, Michael; Pfitzmann, Birgit ; Waidner, Michael: The reactive simulatability (RSIM) framework for asynchronous systems. In: *Inf. Comput.* 205 (2007), Nr. 12, S. 1685–1720

[BRT+13] Bodden, Eric; Ribeiro, Márcio; Tolêdo, Társis; Brabrand, Claus; Borba, Paulo ; Mezini, Mira: SPLLIFT—Statically Analyzing Software Product Lines in Minutes Instead of Years. In: *ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2013

[BS11] Bunke, Michaela; Sohr, Karsten: An architecture-centric approach to detecting security patterns in software. In: *Engineering Secure Software and Systems.* Springer, 2011, S. 156–166

[BSI06] BSI (Bundesamt für Sicherheit in der Informationstechnik): *M 2.378 System-Entwicklung.* https://www.bsi.bund.de/DE/Themen/ITGrundschutz/ITGrundschutzKataloge/Inhalt/_content/m/m02/m02378.html. Version: 2006

[BSI12] BSI (Bundesamt für Sicherheit in der Informationstechnik): *Zertifizierte IT-Sicherheit.* https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/Broschueren/ZertIT/zertifizierte-IT.pdf?__blob=publicationFile, Oktober 2012

[BSI13] BSI (Bundesamt für Sicherheit in der Informationstechnik): *Lageberichte des Bundesamts für Sicherheit in der Informationstechnik (BSI).* `https://www.bsi.bund.de/DE/Publikationen/Lageberichte/lageberichte_node.html`, Januar 2013

[CA11] Chess, B.; Arkin, B.: Software Security in Practice. In: *IEEE Security & Privacy* 9 (2011), March-April, Nr. 2, S. 89–92

[Can01] Canetti, Ran: Universally Composable Security: A New Paradigm for Cryptographic Protocols. In: *Proceedings of FOCS 2001*, 2001, S. 136–145. – Revised version online available at `http://eprint.iacr.org/2000/067`

[CCZ08] Chowdhury, Istehad; Chan, Brian ; Zulkernine, Mohammad: Security metrics for source code structures. In: *Proceedings of the fourth international workshop on Software engineering for secure systems (SESS '08)*, 2008

[Chr11] Christley, Steve: *CWE//SANS Top 25 Most Dangerous Software Errors.* `http://cwe.mitre.org/top25/`, 2011

[Cov13] Coverity: *Annual Coverity Scan Report.* `http://softwareintegrity.coverity.com/register-for-the-coverity-2012-scan-report.html`. Version: 2013

[Cre11] Creative Intellect Consulting: *Failure to invest in secure software delivery puts businesses at risk.* businesswire, `http://www.businesswire.com/news/home/20110223006536/en/Failure-Invest-Secure-Software-Delivery-Puts-Businesses`, Februar 2011

[DKM+12] Dallmeier, Valentin; Knopp, Nikolai; Mallon, Christoph; Fraser, Gordon; Hack, Sebastian ; Zeller, Andreas: Automatically Generating Test Cases for Specification Mining. In: *IEEE Trans. Softw. Eng.* 38 (2012), März, Nr. 2, S. 243–257

[DMN12] DMN (Deutsche Mittelstands Nachrichten): *Angriff auf Online-Banking: Hacker stehlen 36 Millionen Euro von Privatkunden.* `http://www.deutsche-mittelstands-nachrichten.de/2012/12/48673/`, 2012

[DPP12] Denney, Ewen; Pai, Ganesh ; Pohl, Josef: Heterogeneous Aviation Safety Cases: Integrating the Formal and the Non-formal. In: *Proceedings of the 2012 IEEE 17th International Conference on Engineering of Complex Computer Systems (ICECCS '12)*, IEEE Computer Society, 2012, S. 199–208

[ECGN01] Ernst, Michael D.; Cockrell, Jake; Griswold, William G. ; Notkin, David: Dynamically discovering likely program invariants to support program evolution. In: *IEEE Transactions on Software Engineering* 27 (2001), Februar, Nr. 2, S. 99–123

[EU 06a] EU (Europäische Union): *RICHTLINIE 2006/48/EG DES EUROPÄISCHEN PARLAMENTS UND DES RATES vom 14. Juni 2006 über die Aufnahme und Ausübung der Tätigkeit der Kreditinstitute.* Amtsblatt der Europäischen Union L 177/1, 2006

[EU 06b] EU (Europäische Union): *RICHTLINIE 2006/49/EG DES EUROPÄISCHEN PARLAMENTS UND DES RATES vom 14. Juni 2006 über die an-*

gemessene Eigenkapitalausstattung von Wertpapierfirmen und Kreditinstituten. Amtsblatt der Europäischen Union L 177/201, 2006

[FAR+13] Fritz, Christian; Arzt, Steven; Rasthofer, Siegfried; Bodden, Eric; Bartel, Alexandre; Klein, Jacques; le Traon, Yves; Octeau, Damien ; McDaniel, Patrick: *Highly Precise Taint Analysis for Android Applications. Technical Report.* http://www.bodden.de/pubs/TUD-CS-2013-0113.pdf, Mai 2013

[FIB13] Frost & Sullivan; (ISC)^2 ; Booz Allen Hamilton: *The 2013 (ISC)^2 Global Information Security Workforce Study.* https://www.isc2.org/workforcestudy/Default.aspx, 2013

[For11a] Forrester Consulting: *State of Application Security.* http://www.microsoft.com/en-us/download/confirmation.aspx?id=2629, 2011

[For11b] Forrester Research: *Software Integrity Risk Report — The Critical Link Between Business Risk And Development Risk.* http://www.coverity.com/library/pdf/Software_Integrity_Risk_Report.pdf, April 2011

[FPP12] Fichtinger, Barbara; Paulisch, Frances ; Panholzer, Peter: Driving Secure Software Development Experience in a Diverse Product Environment. In: *IEEE Security & Privacy* 10 (2012), Nr. 2, S. 97–101

[GJJ06] Ganapathy, V.; Jaeger, T. ; Jha, S.: Retrofitting legacy code for authorization policy enforcement. In: *2006 IEEE Symposium on Security and Privacy*, 2006

[Gle05] Gleghorn, Rodney: Enterprise Application Integration: A Manager's Perspective. In: *IT Professional* 7 (2005), November, Nr. 6, S. 17–23

[Gra06] Graham, Dan: *The CLASP Application Security Process.* https://buildsecurityin.us-cert.gov/bsi/100/version/1/part/4/data/CLASP_ApplicationSecurityProcess.pdf?branch=main&language=default, 2006

[Ham06] Hamlen, Kevin: *Security policy enforcement by automated program-rewriting.* Ithaca, NY, USA, Diss., 2006

[hei08] heise Online: *Schwache Krypto-Schlüssel unter Debian, Ubuntu und Co.* http://www.heise.de/security/meldung/Schwache-Krypto-Schluessel-unter-Debian-Ubuntu-und-Co-207332.html. Version: Mai 2008

[hei11] heise Security: *Angriff auf Playstation Network: Persönliche Daten von Millionen Kunden gestohlen.* http://www.heise.de/security/meldung/Angriff-auf-Playstation-Network-Persoenliche-Daten-von-Millionen-Kunden-gestohlen-1233136.html, April 2011

[hei12a] heise Security: *Chinesische Hacker gingen bei Nortel ein und aus.* http://www.heise.de/security/meldung/Chinesische-Hacker-gingen-bei-Nortel-ein-und-aus-1433741.html. Version: 2012

[hei12b] heise Security: *Immer mehr EU-Bürger haben Angst vor Cyber-Kriminalität.* http://www.heise.de/security/meldung/Immer-mehr-EU-Buerger-haben-Angst-vor-Cyber-Kriminalitaet-1635864.html, Juli 2012

[hei13] heise Security: *Schwerwiegende Sicherheitslücke bei Amazon.* http://www.heise.de/security/meldung/Schwerwiegende-Sicherheitsluecke-

bei-Amazon-1786722.html, Januar 2013

[HHH12] Hollunder, B.; Herrmann, M. ; Hülzenbecher, A.: Design by Contract for Web Services: Architecture, Guidelines, and Mappings. In: *International Journal On Advances in Software* 5 (2012), Nr. 1 and 2, S. 53–64

[HL06] Howard, Michael; Lipner, Steve: *The Security Development Lifecycle*. Redmond, WA, USA : Microsoft Press, 2006

[HN08] Haase, Thomas; Nagl, Manfred: Service-Oriented Architectures and Application Integration. In: *Collaborative and Distributed Chemical Engineering. From Understanding to Substantial Design Process Support - Results of the IMPROVE Project* Bd. 4970. Springer, 2008, S. 727–740

[How06] Howard, Michael: A Process for Performing Security Code Reviews. In: *IEEE Security & Privacy* 4 (2006), Juli, Nr. 4, S. 74–79

[HS09] Hammer, Christian; Snelting, Gregor: Flow-Sensitive, Context-Sensitive, and Object-sensitive Information Flow Control Based on Program Dependence Graphs. In: *International Journal of Information Security* 8 (2009), Dezember, Nr. 6, S. 399–422

[IBM06] IBM: *IBM Internet Security Systems Product Lifecycle Policy*. http://www-935.ibm.com/services/us/iss/pdf/support_product_lifecycle_policy.pdf. Version: June 2006

[IBM12] IBM: *IBM X-Force 2012 Mid-year Trend and Risk Report*. http://www-935.ibm.com/services/us/iss/xforce/trendreports/, September 2012

[ISO11] ISO (International Standardization Organisation): *Security management systems for the supply chain – Development of resilience in the supply chain – Requirements with guidance for use*. http://www.iso.org/iso/home/store/catalogue_ics/catalogue_detail_ics.htm?csnumber=56087, 2011

[Jac06] Jackson, Joab: *Symantec: Common Criteria is bad for you*. http://gcn.com/Articles/2007/05/04/Symantec-Common-Criteria-is-bad-for-you.aspx?p=1, 2006

[Jür02] Jürjens, Jan: UMLsec: Extending UML for Secure Systems Development. In: *Proceedings of the 5th International Conference on The Unified Modeling Language (UML '02)*, 2002

[JYB08] Jürjens, Jan; Yu, Yijun ; Bauer, Andreas: Tools for traceable security verification. In: *Proceedings of the 2008 international conference on Visions of Computer Science: BCS International Academic Conference (VoCS'08)*, 2008, 367–378

[Kal12] Kallus, Michael: *5 Sicherheitsschwachstellen in SAPSystemen*. CIO Magazin http://www.cio.de/2889344, August 2012

[KS06] Kirkland, Dustin; Salem, Loulwa: *BogoSec: Source Code Security Quality Calculator*. http://sourceforge.net/projects/bogosec/, März 2006

[KT09] Khan, Khaled M.; Tan, Calvin: SecCom: A Prototype for Integrating Security-Aware Components. In: *Information Systems: Modeling, Development, and Integration, Third International United Information System Conference,*

UNISCON 2009, Sydney, Australia, April 21-24, 2009. Proceedings Bd. 20, Springer, 2009 (Lecture Notes in Business Information Processing), S. 393–403

[LBD02] Lodderstedt, Torsten; Basin, David A. ; Doser, Jürgen: SecureUML: A UML-Based Modeling Language for Model-Driven Security. In: *Proceedings of the 5th International Conference on The Unified Modeling Language (UML '02)*, 2002

[Loc12] Lochbihler, Andreas: *A Machine-Checked, Type-Safe Model of Java Concurrency : Language, Virtual Machine, Memory Model, and Verified Compiler*, Karlsruher Institut für Technologie, Fakultät für Informatik, Diss., Juli 2012

[LPT06] Lapadula, A.; Pugliese, R. ; Tiezzi, F.: A WSDL-based type system for WS-BPEL. In: *Coordination Models and Languages* Springer, 2006, S. 145–163

[LRB+05] Linn, C. M.; Rajagopalan, M.; Baker, S.; Collberg, C.; Deinsty, S. K. ; Hartman, J. H.: Protecting against unexpected system calls. In: *In Proceedings of the 14th USENIX Security Symposium*, 2005, S. 239–254

[LSP+11] Ladd, David; Simorjay, Frank; Pulikkathara, Georgeo; Jones, Jeff; Miller, Matt; Lipner, Steve ; Rains, Tim: *The SDL Progress Report.* `http://www.microsoft.com/en-us/download/details.aspx?id=14107`, 2011

[LSS11] Lund, Mass S.; Solhaug, Bjørnar ; Stølen, Ketil: *Model-Driven Risk Analysis - The CORAS Approach.* Springer, 2011

[McG06] McGraw, Gary: *Building Secure Software.* Addison Wesley Professional Computing, 2006

[MDAB10] Murdoch, Steven J.; Drimer, Saar; Anderson, Ross J. ; Bond, Mike: Chip and PIN is Broken. In: *IEEE Symposium on Security and Privacy (S&P 2010)*, 2010

[MFMP08a] Mellado, D.; Fernández-Medina, E. ; Piattini, M.: Security Requirements Variability for Software Product Lines. In: *Third International Conference on Availability, Reliability and Security(ARES '08)*, 2008, S. 1413–1420

[MFMP08b] Mellado, Daniel; Fernández-Medina, Eduardo ; Piattini, Mario: Towards security requirements management for software product lines: A security domain requirements engineering process. In: *Computer Standards & Interfaces* 30 (2008), Nr. 6, S. 361–371

[MFMP09] Mellado, Daniel; Fernández-Medina, Eduardo ; Piattini, Mario: Security Requirements Management in Software Product Line Engineering. In: *e-Business and Telecommunications, International Conference, ICETE 2008, Porto, Portugal, July 26-29, 2008, Revised Selected Papers*, 2009

[Mic10] Microsoft: *Secure Development Lifecycle — Simplified Implementation of the Microsoft SDL.* `http://download.microsoft.com/download/F/7/D/F7D6B14F-0149-4FE8-A00F-0B9858404D85/Simplified%20Implementation%20of%20the%20SDL.doc`, 2010

[Mic13a] Microsoft: *Microsoft Security Development Lifecycle Tools.* `http://www.microsoft.com/security/sdl/adopt/tools.aspx`, Januar 2013

[Mic13b] Microsoft: *SDL Helps Build More Secure Software.* http://www.microsoft.com/security/sdl/learn/measurable.aspx, 2013

[MIT13] MITRE: *Common Weakness Enumeration.* http://sourceforge.net/projects/bogosec/, Februar 2013

[MLD08] Mourad, Azzam; Laverdière, Marc-André ; Debbabi, Mourad: An aspect-oriented approach for the systematic security hardening of code. In: *Computers and Security* 27 (2008), Nr. 3-4, S. 101 – 114

[MM08] Manuj, Ila; Mentzer, John T.: Global Supply Chain Risk Management. In: *Journal of Business Logistics* 29 (2008), Nr. 1, S. 133–155

[MOA11] Mourad, A.; Otrok, H. ; Ayoubi, S.: Toward Systematic Integration of Security Policies into Web Services. In: *2011 European Intelligence and Security Informatics Conference (EISIC)*, 2011, S. 220 –223

[MRFMP09] Mellado, Daniel; Rodriguez, J.; Fernández-Medina, E. ; Piattini, M.: Automated Support for Security Requirements Engineering in Software Product Line Domain Engineering. In: *International Conference on Availability, Reliability and Security,(ARES '09)*, 2009, S. 224–231

[MRFU11] Mukhija, Arun; Rosenblum, David S.; Foster, Howard ; Uchitel, Sebastián: Runtime Support for Dynamic and Adaptive Service Composition. In: *Rigorous Software Engineering for Service-Oriented Systems - Results of the SENSORIA Project on Software Engineering for Service-Oriented Computing* Bd. 6582. Springer, 2011, S. 585–603

[MWC10] Mettler, Adrian; Wagner, David ; Close, Tyler: *Joe-E: A Security-Oriented Subset of Java.* http://joe-e.org/, 2010

[NCH+05] Necula, George C.; Condit, Jeremy; Harren, Matthew; McPeak, Scott ; Weimer, Westley: CCured: type-safe retrofitting of legacy software. In: *ACM Trans. Program. Lang. Syst.* 27 (2005), Mai, Nr. 3, S. 477–526

[NIS10] NIST (National Institute for Standards): *Guide for Applying the Risk Management Framework to Federal Information Systems — A Security Life Cycle Approach.* NIST Special Publication 800-37 Rev. 1, http://csrc.nist.gov/publications/nistpubs/800-37-rev1/sp800-37-rev1-final.pdf, Februar 2010

[NTD12] Nazemi, Eslam; Tarokh, Mohammad J. ; Djavanshir, G.Reza: ERP: A Literature Survey. In: *The International Journal of Advanced Manufacturing Technology* 61 (2012), S. 999–1018

[Nü12] Nüsse, Andrea: *Revolution per Kurznachricht.* http://www.zeit.de/politik/ausland/2012-01/aegypten-revolution-jahrestag, Januar 2012

[OAK+11] O'Sullivan, Pádraig; Anand, Kapil; Kotha, Aparna; Smithson, Matthew; Barua, Rajeev ; Keromytis, AngelosD: Retrofitting Security in COTS Software with Binary Rewriting. In: *Future Challenges in Security and Privacy for Academia and Industry* Bd. 354. Springer Berlin Heidelberg, 2011

[OAS12] OASIS Web Services Security Maintenance TC: *Web Services Security v1.1.1.* OASIS Standards, https://www.oasis-open.org/standards#wssv1.

1.1, Mai 2012

[ON98] Oheimb, David von; Nipkow, Tobias: Machine-checking the Java Specification: Proving Type-Safety. In: *Formal Syntax and Semantics of JAVA*, Springer, 1998, S. 119–156

[Ope13] OpenSAMM: *Open Software Assurance Maturity Model.* http://www.opensamm.org/. Version: 2013

[OTT11] OTTF (The Open Group Trusted Technology Forum): *Open Trusted Technology Provider Framework (O-TTPF) — Industry Best Practices for Manufacturing Technology Products that Facilitate Customer Technology Acquisition Risk Management Practices and Options for Promoting Industry Adoption.* http://www.opengroup.org/ottf, Februar 2011

[PC10] Pfleeger, S.L.; Cunningham, R.K.: Why Measuring Security Is Hard. In: *IEEE Security & Privacy,* 8 (2010), Nr. 4, S. 46–54

[RBG12] Reischuk, Raphael M.; Backes, Michael ; Gehrke, Johannes: SAFE Extensibility for Data-Driven Web Applications. In: *WWW'12: Proceedings of the 21st International Conference on World Wide Web.* Lyon, France, 2012

[RGWS08] Reichenbach, Gerold; Göbel, Ralf; Wolff, Hartfrid ; Stokar von Neuforn, Silke: *Risiken und Herausforderungen für die öffentliche Sicherheit in Deutschland — Grünbuch des Zukunftsforums Öffentliche Sicherheit — Szenarien und Leitfragen.* http://www.zukunftsforum-oeffentliche-sicherheit.de/downloads/Gruenbuch_Zukunftsforum.pdf, 2008

[RHJS05] Rajagopalan, Mohan; Hiltunen, Matti; Jim, Trevor ; Schlichting, Richard: Authenticated System Calls. In: *In Proc. IEEE International Conference on Dependable Systems and Networks (DSN2005)*, 2005

[ROB11] Rodrigues, Nuno; Oliveira, Nuno ; Barbosa, Luís S.: The role of coordination analysis in software integration projects. In: *On the Move to Meaningful Internet Systems (OTM 2011)* Bd. LNCS 7046, Springer-Verlag, October 2011, S. 83–92

[RRDO10] Rescorla, E.; Ray, M.; Dispensa, S. ; Oskov, N.: *Transport Layer Security (TLS) Renegotiation Indication Extension.* RFC 5746 (Proposed Standard). http://www.ietf.org/rfc/rfc5746.txt. Version: Februar 2010

[SAF07] SAFECode (Software Assurance Forum for Excellence in Code): *SAFECode.* http://www.safecode.org/index.php, 2007

[Sav10] Savola, Reijo: On the Feasibility of Utilizing Security Metrics in Software-Intensive Systems. In: *IJCSNS International Journal of Computer Science and Network Security, VOL.10 No.1* (2010)

[Sch11] Schur, Matthias: Experimental specification mining for enterprise applications. In: *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering,(ESEC/FSE '11)*, 2011

[SLE05] Saitta, P.; Larcom, B. ; Eddington, M.: Trike v. 1 methodology document. (2005). http://www.octotrike.org/papers/Trike_v1_Methodology_

Document-draft.pdf

[SM99] Souder, T.; Mancoridis, S.: A tool for securely integrating legacy systems into a distributed environment. In: *Proceedings. Sixth Working Conference on Reverse Engineering*, 1999, S. 47 –55

[Spi12] Spiegel Online: *Industriespionage bei Nortel — Chinesische Hacker sollen Tech-Konzern ausgeplündert haben.* http://www.spiegel.de/netzwelt/web/industriespionage-bei-nortel-chinesischehacker-sollen-tech-konzern-ausgepluendert-haben-a-815102.html, 2012

[Spi13] Spiegel Online: *Monatelanger Angriff — Chinesische Hacker spähten „New York Times" aus.* http://www.spiegel.de/netzwelt/netzpolitik/new-york-times-monatelange-angriffechinesischer-hacker-a-880654.html, 2013

[SPL03] Seacord, R.C.; Plakosh, D. ; Lewis, G.A.: *Modernizing legacy systems: software technologies, engineering processes, and business practices.* Addison-Wesley Professional, 2003

[SRM+09] Simpson, Stacy; Reddy, Dan; Minnis, Brad; Fagan, Chris; McGuire, Cheri; Nicholas, Paul; Baldini, Diego; Uusilehto, Janne; Bitz, Gunter; Karabulut, Yuecel ; Phillips, Gary: *Software Supply Chain Integrity Framework — Defining Risks and Responsibilities for Securing Software in the Global Supply Chain.* SAFECode Publication, http://www.safecode.org/publications/SAFECode_Supply_Chain0709.pdf, 2009

[SS05] Schelp, Joachim; Schwinn, Alexander: Extending the business engineering framework for application integration purposes. In: *Proceedings of the 2005 ACM Symposium on Applied Computing (SAC)*, 2005, S. 1333–1337

[ST07] Siek, J.; Taha, W.: Gradual typing for objects. In: *ECOOP 2007–Object-Oriented Programming* (2007), S. 2–27

[Tas02] Tassey, Gregory: *The economic impacts of inadequate infrastructure for software testing.* NIST (National Institute of Standards and Technology), Planning Report 02-3, 2002

[The11] The Open Group TOGAF-SABSA Integration Working Group: *TOGAF and SABSA Integration — How SABSA and TOGAF complement each other to create better architectures.* White Paper, Reference W117, https://www2.opengroup.org/ogsys/catalog/w117, Oktober 2011

[TW07] Telang, Rahul; Wattal, Sunil: Impact of Software Vulnerability Announcements on the Market Value of Software Vendors — An Empirical Investigation. In: *Workshop on the Economics of Information Security (WEIS'07)*, 2007

[VK11] Vorgang, Blair R.; Karry, Alec: *Addressing Software Security in the Federal Acquisition Process.* Cigital White Paper, https://www.cigital.com, 2011

[WAZ12] WAZ: *Hacker nutzen immer öfter Sicherheitslücken bei Behörden.* http://www.derwesten.de/wirtschaft/digital/hacker-nutzen-immer-oefter-sicherheitsluecken-bei-behoerdenid6408800.html, Februar 2012

[WL11] Wu, Zhuang; Li, Yan: Research on enterprise application integration based on Web. In: *2011 International Conference on Mechatronic Science, Electric Engineering and Computer (MEC)*, 2011, S. 2221 –2224

[WLL08] Williams, Zachary; Lueg, Jason E. ; LeMay, Stephen A.: Supply chain security: an overview and research agenda. In: *International Journal of Logistics Management* 19 (2008), August, Nr. 2, S. 254–282

[WOUK12] Wataguchi, Yoshiro; Okubo, Takao; Unno, Yukie ; Kanaya, Nobuyuki: Cooperative Secure Integration Process for Secure System Development. In: *15th International Conference on Network-Based Information Systems (NBiS 2012)*, 2012, S. 782–786

[Wri11] Wright, Craig S.: Software, Vendors and Reputation: An Analysis of the Dilemma in Creating Secure Software. In: *Trusted Systems - Second International Conference, INTRUST 2010, Revised Selected Papers* Bd. LNCS 6802, Springer, 2011, S. 346–360

[WXH09] Wu, Shi L.; Xu, Lida ; He, Wu: Industry-oriented enterprise resource planning. In: *Enterprise Information Systems* 3 (2009), Nr. 4, S. 409–424

[Xu11] Xu, Li D.: Enterprise Systems: State-of-the-Art and Future Trends. In: *IEEE Transactions on Industrial Informatics* 7 (2011), Nr. 4, S. 630–640

[Zel07] Zeller, Andreas: The Future of Programming Environments: Integration, Synergy, and Assistance. In: *2007 Future of Software Engineering(FOSE '07)*, 2007

DANKSAGUNG

Die Entstehung dieses Trend- und Strategieberichts ist vom Bundesministerium für Bildung und Forschung (BMBF) im Rahmen der Förderung der Kompetenzzentren zur Cybersicherheit

— European Center for Security and Privacy by Design (EC SPRIDE, `http://www.ec-spride.de`),
— Center for IT-Security, Privacy and Accountability (CISPA, `http://www.cispa-security.de`) und
— Kompetenzzentrum für angewandte Sicherheitstechnologie (KASTEL, `http://www.kastel.kit.edu`)

unterstützt worden. Die Autoren danken dem Bundesministerium für Bildung und Forschung für diese Unterstützung vielmals.

Darüber hinaus möchten wir uns bei Herrn Thomas Caspers vom Bundesamt für Sicherheit in der Informationstechnik (BSI) für seine hilfreichen Hinweise bedanken. Weiterer Dank geht an Anne Grauenhorst (CASED und EC SPRIDE), Alex Wöhnl (EC SPRIDE), Viktoriia Kunetska (EC SPRIDE) und Sarah Ahmed (CASED) für deren Unterstützung.

Die Kompetenzzentren für Cybersicherheit

Damit sich Deutschland den großen Zukunftsfragen der Cybersicherheit langfristig stellen kann, hat das Bundesministerium für Bildung und Forschung (BMBF) mit CISPA, EC SPRIDE und KASTEL drei Kompetenzzentren ausgewählt. Sie bündeln herausragende Fähigkeiten der besten Hochschulen und außeruniversitären Forschungseinrichtungen auf dem Gebiet der Cybersicherheitsforschung thematisch und organisatorisch. Die Zentren werden seit dem Jahr 2011 von dem Bundesministerium für Bildung und Forschung (BMBF) gefördert. Wenngleich die Zentren für leicht unterschiedliche Schwerpunkte stehen, arbeiten sie inhaltlich eng zusammen.

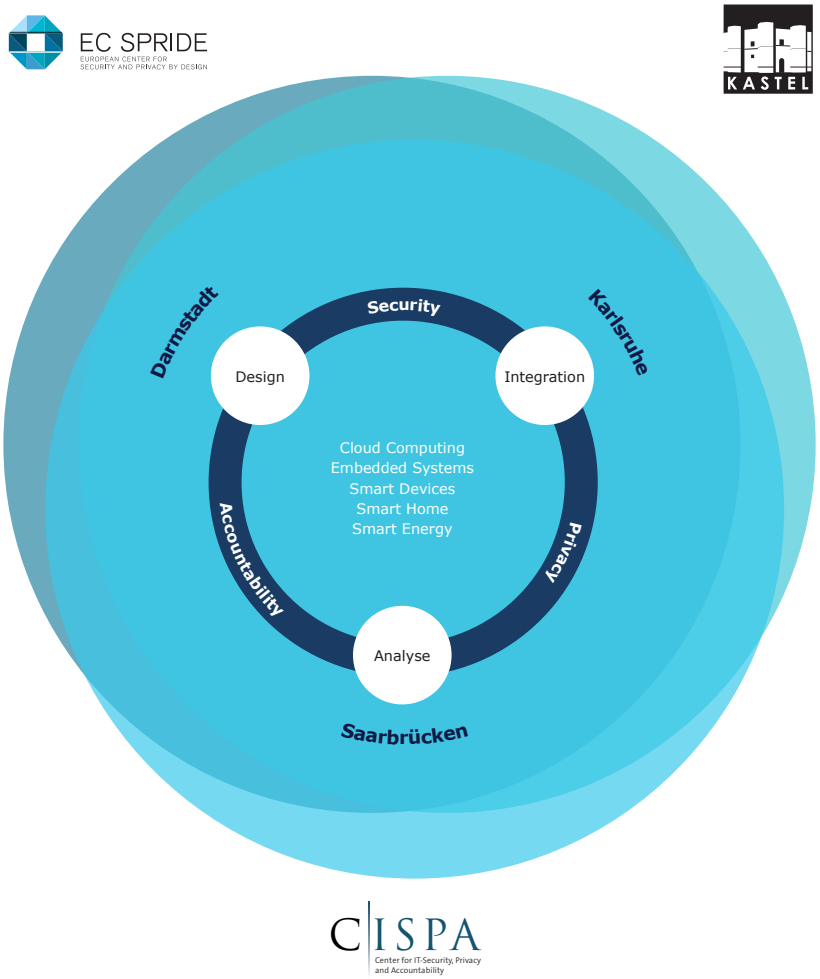


Abbildung 8: Die Kompetenzzentren für Cybersicherheit

CISPA (Saarbrücken)

Das Ziel des Center for IT-Security, Privacy and Accountability (CISPA) ist es, anhand eines ganzheitlichen Ansatzes Lösungen für die Kernprobleme der IT-Sicherheit in der digitalen Gesellschaft zu entwickeln. Das Zentrum kombiniert dazu eine breite Grundlagenforschung zur Analyse bestehender und Entdeckung neuer Lösungsansätze mit deren systematischer Weiterentwicklung zu einem universellen Werkzeugkasten von praktisch einsetzbaren Sicherheitstechnologien in komplexen Gesamtsystemen. Die Kernthemen sind: Verlässliche Sicherheit, Verantwortlichkeit und Schutz der Privatsphäre.

EC SPRIDE (Darmstadt)

Das European Center for Security and Privacy by Design (EC SPRIDE) erforscht, auf welche Weise IT-Entwickler/innen Software und IT-Systeme vom Entwurf an – also „by Design“ – und über den gesamten Lebenszyklus hinweg optimal absichern können. In den Forschungsbereichen *Engineering*, *Building Blocks* und *Blueprint* erarbeiten die Forscher/innen Grundlagenwissen sowie neue Entwicklungs- und Testverfahren für optimale Softwaresicherheit. Dabei berücksichtigen sie auch aktuelle technische und gesellschaftliche Entwicklungen als praxisrelevante Parameter.

KASTEL (Karlsruhe)

Das Kompetenzzentrum für angewandte Sicherheitstechnologie (KASTEL) untersucht, wie sichere Anwendungen in einem durchgängigen Prozess entwickelt werden können. Demonstriert wird dies an drei gesellschaftlich hoch relevanten Prototypen zu Cloud Computing, Smart Energy und privatsphärenrespektierender Kameraüberwachung. Dazu kooperieren elf Gruppen aus den Fachbereichen Informatik, Wirtschafts- und Rechtswissenschaften. Ziel ist die Abkehr von isolierten Teillösungen und die Entwicklung eines ganzheitlichen Ansatzes, der die Kompetenzen und Methoden verschiedener Disziplinen integriert.

CISPA
Center for IT-Security, Privacy and Accountability

EC SPRIDE
EUROPEAN CENTER FOR SECURITY AND PRIVACY BY DESIGN

KASTEL

Kompetenzzentren für IT-Sicherheit

ISBN 978-3-8396-0567-7