Bachelorarbeit

Jakob Ledig

Evaluation und Auswahl geeigneter Sicherheitskonzepte in der agilen Softwareentwicklung am Beispiel einer Single Page Application

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Jakob Ledig

Evaluation und Auswahl geeigneter Sicherheitskonzepte in der agilen Softwareentwicklung am Beispiel einer Single Page Application

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

im Studiengang Bachelor of Science Angewandte Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Klaus Peter Kossakowksi
Zweitgutachter: Prof. Dr. Stefan Sarstedt

Eingereicht am: 19. September 2018

Jakob Ledig

Thema der Arbeit
Evaluation und Auswahl geeigneter Sicherheitskonzepte in der agilen Softwareentwicklung am Beispiel einer Single Page Application

Stichworte
Softwareentwicklung, IT-Sicherheit, Security Engineering, Softwaredesign, Securitykonzepte, Security Engineering

Kurzzusammenfassung
In der Softwarentwicklung haben sich agile Vorgehensweisen weitestgehend durchgesetzt. Nicht selten jedoch wird in solchen Projekten IT-Sicherheit konzeptuell vernachlässigt. Vorgehensmodelle wie Scrum müssen erst erweitert werden, um im alltäglichen Geschäft unter Zeitdruck Securityaspekten den nötigen Stellenwert einzuräumen, um essentielle Fehler zu vermeiden. Diese Arbeit evaluiert anhand des Anwendungsfalles einer Single Page Application verschiedene Leitfäden zur IT-Sicherheit und stellt einige Prozesserweiterungen für konzeptuell sicherere agile Softwareentwicklung zusammen.
Jakob Ledig

Title of the paper
Evaluation and selection security concepts in agile software engineering at the example of a single page application

Keywords
Software development, IT security, security engineering, software design, security concepts, security engineering

Abstract
In software development, agile methods have prevailed. Fequently though, security is being neglected conceptually. Frameworks like Scrum have to be extended to accomodate aspects of security even in a time critical daily routine to stay ahead of essential mistakes. This thesis evaluates different security compendiums at the example of a single page application and provides process enhancements for a security infused agile software development.

Danksagung

Eine Arbeit wie diese entsteht nicht im luftleeren Raum. Insbesondere bedurfte es einiger Ausdauer, mich überhaupt zur Aufnahme des Studiums der Angewandten Informatik zu überzeugen und während der vergangenen Jahre mit Rat und Tat an meiner Seite zu stehen. Daher möchte ich meiner Lebensgefährtin Fenia noch vor allen anderen Unterstützer*innen danken, denn ohne sie wäre keine einzige der folgenden Seiten entstanden.

Ebenfalls danken möchte ich meiner Familie, deren geistiger wie materieller Beistand ebenfalls wesentlich für mein gesamtes Studium und das Zustandekommen dieser Arbeit war. Der unbedingte Glaube aller Vorgenannten an mich und meine Fähigkeiten hat mich in den vergangenen Jahren erheblich getragen.

Darüber hinaus gebührt Dank meinem Arbeitgeber, der evodion IT GmbH. So konnte ich nicht nur unter ihrem Flügel diese Arbeit schreiben, sondern zog erheblichen Nutzen aus der Gelassenheit & Kompetenz bei meiner dortigen Unterstützung und Beratung zu Fachthemen. In diesem Atemzug möchte ich auch direkt Herrn Professor Dr. Klaus-Peter Kossakowski Dank sagen, dessen geduldige und engagierte Art der Betreuung mir manches mal aufkeimende Sorgen nehmen konnte.

Den Beitrag meines restlichen persönlichen Umfeldes möchte ich gleichfalls würdigen - die Stunden, die in das Glätten wenig eleganter Formulierungen und interessierte Nachfragen geflossen sind, kann ich nicht einmal beziffern. Allen Genannten möchte ich daher hiermit tief empfunden „Danke!“ sagen.

Inhaltsverzeichnis

1. Einleitung **1**
- 1.1. Beispiele bekannter, offensichtlicher Sicherheitsprobleme . . . 3
  - 1.1.1. Das besondere elektronische Anwaltspostfach . . . 3
  - 1.1.2. PC-Wahl . . . 5
- 1.2. Zielgruppe, Voraussetzungen und Anspruch . . . 6
- 1.3. Ziel: Bewertung etablierter IT-Sicherheitskonzepte für SPA-Entwicklung . . . 6
  - 1.3.1. Zielsetzung . . . 6
  - 1.3.2. Exkurs: Single Page Application . . . 7
- 1.4. Übersicht über die weitere Arbeit . . . 9
- 1.5. Technische Anmerkung zu Referenzen auf Bücher in EPUB-Form . . . 10

2. Bestandsaufnahme zur IT-Sicherheit in der agilen Softwareentwicklung **11**
- 2.1. Begriffseinführungen . . . 11
  - 2.1.1. Subjekt und Objekt . . . 11
  - 2.1.2. Risiko und Bedrohung/Gefährdung . . . 12
  - 2.1.3. Schutzgüter/Schutzziele . . . 12
  - 2.1.4. Abgrenzung IT-, Daten-, Cyber- und Informationssicherheit . . . 13
  - 2.1.5. Verfahren, Konzepte, Modelle, Techniken . . . 15
  - 2.1.6. Risikomanagement in der Softwareentwicklung . . . 16
- 2.2. Evaluierung dreier Leitfäden . . . 16
  - 2.2.1. BSI-Grundschutz-Kataloge . . . 17
  - 2.2.2. Security Software Development Lifecycle . . . 21
  - 2.2.3. OWASP . . . 25
- 2.3. Übersicht über Risikomanagementverfahren . . . 27
  - 2.3.1. STRIDE . . . 28
  - 2.3.2. DREAD . . . 29
  - 2.3.3. Risk Levels . . . 30

3. Erarbeitung der Vorgehensweise **31**
- 3.1. Zusammenstellung der Prozessschritte . . . 31
  - 3.1.1. Security Engineering nach BSI-Grundschutz . . . 32
  - 3.1.2. Agiles Vorgehen nach Secure Software Development Lifecycle . . . 35
  - 3.1.3. OWASP . . . 38
  - 3.1.4. Ergebnis . . . 40
- 3.2. Vorstellung des verwendeten Entwicklungsmodells . . . 41
  - 3.2.1. Scrum . . . 41

3.2.2. Security-Gruppe . . . 42
3.3. Vorstellung der Anwendung . . . 43
3.3.1. Fachlichkeit & Anforderungen . . . 43
3.3.2. Implementation . . . 43
3.3.3. Deployment & Infrastruktur . . . 45
3.4. Untersuchung des Reifegrades der Entwicklung mit OWASP SAMM . . . 46
3.4.1. Exemplarische Untersuchung der Business Function Verwaltung („Governance“) . . . 47
3.4.2. Ergebnis . . . 48
3.5. Folgerungen aus der Analyse . . . 49

4. Umsetzung **50**
4.1. Schutzgüter . . . 50
4.1.1. Werte der Anwendung . . . 50
4.1.2. Schutzziele . . . 51
4.2. Konkrete Änderungen im agilen Prozess . . . 52
4.2.1. Risikomanagement . . . 52
4.2.2. Weitere Verbesserungen . . . 59
4.3. Modellierung der Anwendung . . . 63
4.3.1. Baseline . . . 63
4.3.2. Exemplarische Detailmodellierung der Login-Komponente . . . 65
4.4. Bedrohungs- und Risikoanalyse . . . 67
4.4.1. STRIDE in Aktion . . . 67
4.4.2. Risikobewertung . . . 68
4.4.3. Eine beispielhafte Evil User Story . . . 71
4.4.4. Nützlichkeit der IT-Grundschutzkataloge . . . 72
4.5. Besonderheiten bezüglich der SPA-Aspekte . . . 72
4.5.1. Node.js & npm . . . 72
4.5.2. Kompartmentalisierung . . . 73
4.5.3. Einfluss der Technologie auf Risikobewertung . . . 74

5. Ausleitung **75**
5.1. Zusammenfassung der Ergebnisse . . . 75
5.1.1. Grundsätzliches . . . 75
5.1.2. Modellierung, STRIDE & Risikobewertung . . . 76
5.1.3. Folgerungen für den agilen Prozess . . . 76
5.2. Bewertung und Fazit . . . 77
5.3. Ausblick . . . 79

A. Anhang **81**

Abbildungsverzeichnis

1.1. Kosten - Umfang - Dauer 2
1.2. Auswertung einer Wahl auf verschiedenen Ebenen 5
1.3. SPA Kommunikation 8
1.4. Calibre E-Book viewer: Reference Mode 10

2.1. Bedrohungen & DFD-Typen 29
2.2. SDL: Risiko-Level 30

3.1. Erstellung eines Sicherheitskonzepts nach BSI-Grundschutzkatalogen 32
3.2. Traditioneller Microsoft-Entwicklungsprozess 35
3.3. SDL-Entwicklungsprozess 36
3.4. Exemplarisches Datenflussdiagramm aus dem SDL 37
3.5. OWASP Risikobewertung 39
3.6. Scrum, vereinfacht dargestellt 42
3.7. SAMM: Education & Guidance 47

4.1. In den Prozess zu integrierende Schritte 55
4.2. Prozesserweiterung in der Variante A priori 57
4.3. Prozesserweiterung in der Variante A posteriori 58
4.4. Minimales DFD der Anwendung 64
4.5. DFD Level 0 der Login-Komponente 65
4.6. DFD Level 1 der Login-Komponente 66

A.1. SDL-Entwicklungsprozess 82
A.2. Infrastrukturdiagramm 83
A.3. DFD Level 1 der Login-Komponente 84

1. Einleitung

IT-Sicherheit wird häufig sehr technisch entlang konkreter Einzelfälle gelehrt. Zur Vermittlung ihrer Konzepte werden in der Regel spezielle Probleme einzelner Protokolle, Schwachstellen spezifischer Algorithmen bis hin zu einzelnen, manipulierbaren Bitfolgen betrachtet. Ohne solche reduktionistischen Betrachtungen ist zwar nur schwer ein ganzheitliches Verständnis dieses Forschungsfeldes zu vermitteln, es fehlt jedoch eine Übertragung in die Realität der professionellen, agilen Softwareentwicklung. In diesem stark termingetriebenen Umfeld kann sich ein Team eine einzelfallorientierte Vorgehensweise zeitlich nicht erlauben. Auch aus wirtschaftlichen Erwägungen muss ein Team seine Zeitressourcen so einsetzen, dass nicht unzählige Arbeitsstunden in die Behebung von Missständen fließen, die von Dritten mit deutlich geringerem finanziellen Aufwand aufgespürt und ausgenutzt werden können. Es bedarf daher eines Brückenschlags zwischen diesen beiden Welten.

Denn beim Beurteilen der Qualität von Softwareprodukten stellt die IT-Sicherheit einen wichtigen Gesichtspunkt dar. Da die zur Entwicklung aufwendbaren Ressourcen jedoch endlich sind, muss das Produkt beim Aufkommen zeitlicher Probleme entweder im Umfang oder der Qualität beschränkt werden, um bei einem festen Budget im Zeitplan zu bleiben. Dem CAP-Theorem nicht unähnlich handelt es sich hierbei um einen klassischen, dreipoligen Zielkonflikt. Das Theorem verweist auf die Unmöglichkeit, gleichzeitig mehr als zwei der drei Eigenschaften Konsistenz, Verfügbarkeit und Ausfallsicherheit bzw. Partitionstoleranz zu gewährleisten. In einem Softwareprojekt ist die Qualität in ähnlicher Weise beschränkt durch Kosten, Umfang und Dauer.

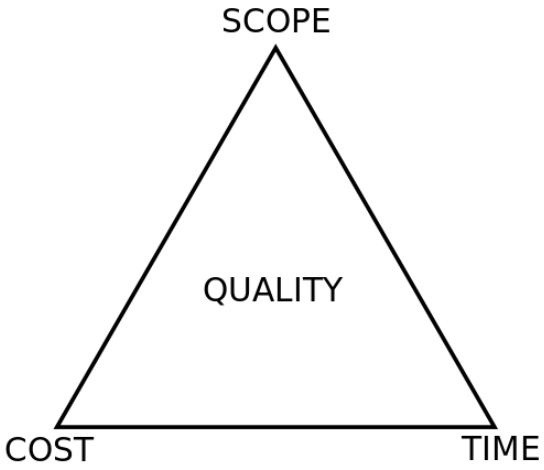


Abbildung 1.1.: Kosten - Umfang - Dauer Wikimedia Commons (2007)

Bei naiver Betrachtung bietet die IT-Sicherheit hier Einsparpotential, da sie selten offen in Erscheinung tritt und so ihre An- oder Abwesenheit für den Kunden nicht offensichtlich ist. So kann auf diese Weise vermeintlich Zeit gespart werden, ohne die Abläufe der Entwicklung aufwändig neu auszurichten. Wenn Maßnahmen der IT-Sicherheit nicht von Anfang an als explizite Anforderungen mit in den Entwicklungsprozess eingebunden sind, steht dem Vernachlässigen der IT-Sicherheit auch methodisch nichts im Weg. Möchte man also verhindern, dass diese unter einem Sparzwang leidet, muss folgerichtig der Entwicklungsprozess von Anfang an so gestaltet sein, dass die Sicherheit betreffende Maßnahmen integraler Bestandteil desselben sind.

Auch sprechen ganz handfeste rechtliche Gründe dafür, sich planvoll der Thematik zu nähern: Bereits seit 1998 gilt das Gesetz zur Kontrolle und Transparenz im Unternehmensbereich. Dieses verpflichtet Vorstände von Aktiengesellschaften und die Geschäftsführungen von GmbHs, „geeignete Maßnahmen zu treffen“, d.h. nach gängiger Lesart z.B. Risikomanagement für IT-Systeme zu betreiben (Deutscher Bundestag, 1998, § 91 Abs. 2 Aktiengesetz). Auch im Telemediengesetz finden sich Passagen mit Bezug auf IT-Sicherheit: Es sieht vor, dass „der Nutzer Telemedien gegen Kenntnisnahme Dritter geschützt in Anspruch nehmen kann“ (Bundesministerium der Justiz und für Verbraucherschutz, 2007, § 13 Abs. 4 Nr. 3). Zur im Sommer 2018 in Kraft getretenen Datenschutzgrundverordnung fehlen bislang Urteile, die als Referenz der Auslegung des Textes durch Gerichte dienen können. Die Einschätzungen gehen auseinander, was etwa die Abmahnbarkeit von Verstößen durch kleinere Webseiten betrifft (spiegel.de (2018)).

Bemerkenswert eindeutig hingegen ist Artikel 32 der DSGVO zur Verpflichtung von „Verantwortliche[n] und Auftraggeber[n]“ zum Schutzniveau bei der Datenverarbeitung. Ihnen wird unter anderem auferlegt, personenbezogene Daten zu verschlüsseln und/oder zu pseudonymisieren, Vertraulichkeit, Integrität und Verfügbarkeit (auf die ich noch zu sprechen kommen werde) zu gewährleisten und Risikomanagement zu betreiben (Amtsblatt der Europäischen Union, 2016, Art. 32).

Die Strafen bei Verstößen gegen die DSGVO sind im Vergleich zu den Gesetzeswerken, die sie ersetzt, außerdem empfindlich höher: Bis zu 4 % des weltweiten Jahresumsatzes können verhängt werden (Amtsblatt der Europäischen Union, 2016, Art. 83 Abs. 5).

1.1. Beispiele bekannter, offensichtlicher Sicherheitsprobleme

Um meine Motivation für diese Arbeit zu verdeutlichen, stelle ich im folgenden Abschnitt als Negativbeispiel zwei Softwareprojekte vor, die in der jüngeren Vergangenheit einige Aufmerksamkeit erfuhren bzw. dies immer noch tun. An ihnen soll veranschaulicht werden, dass selbst heute noch auch in Projekten mit teilweise hohen Budgets und kritischen Fachlichkeiten das Vorhandenseins eines den Sachzwängen angemessenen Sicherheitskonzepts bei weitem keine Selbstverständlichkeit ist.

1.1.1. Das besondere elektronische Anwaltspostfach

Im Januar 2018 sollten Anwälte verpflichtet werden, über das besondere elektronische Anwaltspostfach (nachfolgend in Eigenschreibweise kurz „beA“) Nachrichten mindestens empfangen zu können. Ziel war es, ein Standardverfahren zur elektronischen Kommunikation von Mitgliedern der Bundesrechtsanwaltskammer zu etablieren. Mit Blick auf Anwaltsgeheimnis und das Vertrauensverhältnis zwischen den Kommunikationspartnern ist leicht zu ersehen, dass ein solches System hohe Vertraulichkeitsanforderungen erfüllen können müsste.

Dies sollte geschehen, indem ein sich Rechtsbeistand mit Chip-Karte und PIN an einem PC authentifiziert, woraufhin alle weitere Kommunikation verschlüsselt ablaufen sollte. Die Client-Software mit dem Namen „Client Security“ leistet hierfür die Kommunikation mit dem Chipkartenlesegerät und stellt ein Webinterface zur Verfügung, mit dessen Hilfe Nutzer*innen des beA ihre Korrespondenz abwickeln sollten.

Ab dieser Stelle treten essenzielle Schwächen im Konzept zu Tage: Für den Betrieb des Webinterfaces muss Client Security einen Webserver betreiben. Um Warnungen bei unverschlüsselten HTTP-Verbindungen zu vermeiden, muss dieser Webserver HTTPS anbieten. Damit aktuelle Browser diese Verbindung akzeptieren, ist allerdings ein kryptografisches Server-Zertifikat

vonnöten. Leider verteilte man in der Software jedoch anstelle des öffentlichen den privaten Schlüssel.

Ein solcher in die Öffentlichkeit gelangter privater Schlüssel unterläuft den Zweck einer per Zertifikat durchgeführten Serverauthentifizierung, da sich mit seiner Hilfe beliebige Dritte als legitime Kommunikationspartner des beA-Webinterfaces ausgeben können: Ein Man-in-the-Middle-Angriff wird ermöglicht. Ein öffentlich gewordener und damit der exklusiven Kontrolle des Erstellenden entzogener privater Schlüssel gilt außerdem technisch als kompromittiert. Kurz nachdem Markus Drenger vom Chaos Computer Club das Trust Center Telesec, die das von Client Security ausgelieferte Zertifikat ausgestellt hatten, auf die vorliegenden Missstände hingewiesen hatte, zog Telesec das fragliche Zertifikat zurück. In der Folge konnte das beA nun überhaupt nicht mehr benutzt worden.

In Atos' nachfolgender öffentlichen Kommunikation fand das eigene Verschulden indes keinerlei Widerhall: So schob man zunächst den ehrenamtlichen Auditoren die Schuld zu, diese hätten ein Zertifikat „gehackt". Später war von einem „ungültig gewordenen Zertifikat" die Rede. Technisch versuchte Atos, dem Problem beizukommen, indem man in einem Patch der Software ein neues, selbst signiertes Zertifikat verteilte und ihre Nutzerschaft aufforderte, entgegen aller Warnhinweise dieses als Root-Zertifikat zu installieren - was zur Folge gehabt hätte, dass mit Hilfe dieses Zertifikats beliebige weitere hätten signiert werden können. Gleichzeitig wurde der private Schlüssel für das Zertifikat erneut mitgeliefert. Damit hätte ein Angreifer Zertifikate dieses mal sogar für beliebige andere Internetverbindungen erstellen können, deren mangelnde Echtheit für den Browser nicht mehr zu erkennen gewesen wäre. Die vermeintliche Lösung hatte das Problem also dramatisch verschlimmert.

Diese Erkenntnisse beschränken sich auf den öffentlich verfügbaren Client, und die aufgezählten Fehler beziehen sich allein auf die lokale Absicherung der Kommunikation zwischen Nutzer*in und Client. Die Serversoftware ist hingegen nicht öffentlich verfügbar, so dass über die Qualität der Implementation einzelner Details keine Aussagen getroffen werden können. Gleichwohl steht im Konzept des beA, Nachrichten seien Ende-zu-Ende verschlüsselt, obwohl auf den Servern eine sog. „Umschlüsselung" stattfinden soll: Anstatt also eine Nachricht mit dem öffentlichen Schlüssel des Adressaten zu verschlüsseln, so dass nur dieser sie mit seinem privaten Schlüssel entschlüsseln kann, findet die Verschlüsselung beim beA nur zwischen Client und Server statt, wo die Nachrichten an einzelne Teilnehmer*innen neu verschlüsselt werden - was funktional bestenfalls einer Transportverschlüsselung entspricht, nicht aber Ende-zu-Ende. Der Unterschied ist in diesem Fall besonders brisant, da auf diese Weise die Betreiber der Infrastruktur des beA zwingend Zugriff auf die privaten Schlüssel zur Kommunikation haben, die vom Anwaltsgeheimnis besonders geschützt ist (CCC (2018)).

1.1.2. PC-Wahl

Anders als der Name suggeriert, ist „PC-Wahl“ keine Software zur eigentlichen Durchführung einer Abstimmung, sondern zur korrekten Verrechnung & Zusammenführung der händisch ausgezählten Stimmen aus bundesweit über 70.000 Wahllokalen in 16 Bundesländern mit jeweils eigenen Reglements. Zuletzt wurde PC-Wahl bei der Bundestagswahl 2017 zur Erfassung von (nach Herstellerangaben) bis zu 33 Millionen Stimmen eingesetzt, weist jedoch ähnlich schwere konzeptionelle Sicherheitsprobleme auf wie das beA.

Abbildung 1.2.: Auswertung einer Wahl auf verschiedenen Ebenen
Neumann, Tschirsich, Schröder (2017)

Von Anfang an sperrte sich der Entwickler vote iT gegen ein Audit seiner Software, unter anderem mit dem Argument, eine Prüfung der Software durch den Bundeswahlleiter sei ausreichend. Auf eigene Faust durchgeführte Untersuchungen per Reverse-Engineering durch Mitglieder des Chaos Computer Clubs festigten jedoch immer mehr das Bild, dass damit eklatante Sicherheitsprobleme verschleiert werden sollten. Etwa wurde die Vertraulichkeit von Passwörtern nicht besonders ernst genommen: Die Wahlergebnisse z.B. aus Hessen wurden seit Jahren auf einen FTP-Server geladen, dessen Betreiber die Klartext-Zugangsdaten in öffentlich zugängliche Dokumente schrieb. An anderer Stelle waren Zugangsdaten lediglich durch dezimal repräsentierte Hexzahlen verschleiert, nicht aber kryptografisch vor Zugriff geschützt, oder waren auf leicht erratbare Standardwerte wie „test“ (Passwort und Nutzername) festgeschrieben.

Auch war die Übertragung der Wahlergebnisse über das Internet in keiner Weise abgesichert, so dass Angreifer diese schon an dieser Stelle hätten manipulieren können. Ebenso wurden übermittelte Wahlergebnisse nicht signiert, so dass es möglich gewesen wäre, übertragene Ergebnisse nachträglich zu verändern oder gar mit eigenen Dateien zu überschreiben, ohne dass dies verhindert oder im Nachhinein hätte erkannt werden können.

Erschwerend kamen Schwachstellen der Update-Funktion von PC-Wahl hinzu. Auf dem Webserver, den die Software nach verfügbaren Patches anfragte, fand der CCC nach eigenen Angaben mindestens vier Sicherheitslücken, die das Überschreiben von Updatepaketen ermöglicht hätten. Auf Codesigning hatte vote iT verzichtet, so dass diese manipulierten Updates von

PC-Wahl nicht hätten erkannt werden können. Die ausführbaren Programmdateien lagen auf dem Updateserver zwar nicht im Klartext vor. Jedoch hatte sich der Entwickler nicht auf ein bewährtes Standardverfahren verlassen, sondern einen eigenen Algorithmus implementiert, der vom CCC in kürzester Zeit überwunden werden konnte.

Mit einem Patch wurde den Administrator*innen der Software zwar eine Anleitung geliefert, wie die Verifizierung der Updates von Hand geschehen könnte, bei der aber wiederum nur auf Vorhandensein, nicht aber auf Echtheit der Zertifikate geprüft wurde. Der CCC konnte zeigen, dass ein beliebiges eigenes Zertifikat diese Prüfung unbeanstandet durchläuft. Auch sollte das Programm seine Integrität nun über einen Selbstcheck sicherstellen - was keinerlei Aussagekraft besitzt, da im Zweifelsfall die sich selbst überprüfende Programmdatei und damit die Selbstcheck-Routine zum Ausführungszeitpunkt bereits manipuliert worden ist.

Als der CCC auf diese Probleme hinwies und vote iT eine „Open Source-Spende“ in Form eines eigenen Patches des Update-Mechanismus anbot, deaktivierte der Hersteller die Updatefunktion von PC-Wahl übers Internet stattdessen komplett. Dies löst keineswegs die bestehenden Probleme fehlender Verifikation der Updates, sondern erschwert lediglich Zugang zu diesen (heise (2017)).

1.2. Zielgruppe, Voraussetzungen und Anspruch

Auf den folgenden Seiten beschäftige ich mich intensiv mit Softwareentwicklung. Kenntnisse auf diesem Feld setze ich daher bei der Leserschaft voraus. Ob diese durch ein theoretisches Studium oder berufliche Praxis erworben wurden, ist dabei zweitrangig. Insbesondere die grundlegenden Abläufe agiler Entwicklungsmodelle sollten jedoch bekannt sein, sowie die wichtigsten technischen Begriffe typischer Java-Backends und von Web-Technologien.

Grundlagenwissen der IT-Sicherheit ist von Vorteil, auf konkrete Begriffe und Konzepte werde ich jedoch an entsprechender Stelle in gegebener Tiefe eingehen.

1.3. Ziel: Bewertung etablierter IT-Sicherheitskonzepte für SPA-Entwicklung

1.3.1. Zielsetzung

Ziel dieser Arbeit ist der Einsatz geeigneter Techniken zum methodischen Schutz einer Single Page Application, die im Rahmen eines Kundenprojekts meines Arbeitgebers entwickelt wird, und die Verallgemeinerung der dabei gewonnen Erkenntnisse für den Entwicklungsprozess kommender Projekte.

In einer inhaltlichen Auseinandersetzung mit für den konkreten Anwendungsfall relevanter Fachliteratur werde ich eine Auswahl an Erkenntnissen zusammenstellen, die der Verbesserung eines bestehenden agilen Entwicklungsprozesses hinsichtlich Maßnahmen der IT-Sicherheit dienen können. Mit den auf diese Weise gewonnenen Werkzeugen werde ich die bisherige Arbeit im Projekt bewerten.

Zu diesem Zweck werde ich zwar auch die konkrete Implementation der Anwendung, mehr noch aber die zu ihr führenden Prozesse in Augenschein nehmen. Aus diesen werde ich verallgemeinern, was entweder in zukünftigen Projekten wiederholt oder aber vermieden werden soll. Auf diese Weise soll das bei der Entstehung dieser Arbeit gewonnene Wissen zur Wiederholung herangezogen werden können und so einen Nutzen über die akademische Beschäftigung mit dem Thema hinaus erbringen.

1.3.2. Exkurs: Single Page Application

Gegenstand der folgenden Betrachtung ist ein Softwareprojekt in Form einer Single Page Application, kurz SPA. Hierbei handelt es sich um eine Webseite, die auf Interaktion anders reagiert als eine statische.

Bei einer klassischen Webseite lädt der Browser nach Klick auf einen Link das mit der URL referenzierte Objekt herunter. Anschließend verwirft er den aktuellen Seiteninhalt und stellt das neue Objekt dar. Es ist hierbei nicht ohne weiteres möglich, z.B. Benutzereingaben in einem Formular über den Ladevorgang hinweg zu halten. Eine SPA besteht hingegen aus einem einzelnen HTML-Dokument, dessen Inhalt dynamisch erweitert wird. Dies geschieht, indem Inhalte über „Asynchronous JavaScript and XML“, kurz „Ajax“, oder Websockets im Hintergrund nachgeladen werden. So kann sich der Inhalt einer Webseite ändern, ohne dass sie neu geladen werden muss und Kontext verloren geht.

Zu den evidenten Vorteilen dieser Technik gehört, dass der Interaktionsfluss mit einer Webseite nicht durch Neuladen behindert wird und Daten, die sich im Verlauf der Interaktion nicht ändern, nur einmal geladen werden müssen. Tatsächlich muss der Client stets nur die Daten beim Server anfragen, die für eine Interaktion gerade benötigt werden. Damit reduziert sich die Zahl nötiger Requests & der Datenverkehr. Früher waren dafür Browserplugins wie Flash nötig, die oft Probleme mit Sicherheit und Performance mit sich brachten. Auch skaliert eine SPA vorteilhaft mit häufigeren Seitenabrufen: Da das Rendering auf den Clientgeräten stattfindet, wird der Webserver von dieser Tätigkeit entlastet Santamaria (2015).

Ein prominentes Beispiel für eine SPA ist Google Mail: Nicht nur lädt die Seite nicht komplett neu, wenn man etwa z.B. vom Posteingang aus beginnt, eine neue E-Mail zu schreiben. Auch eventuell gesetzte Filter auf den E-Mails bleiben ebenfalls bestehen.

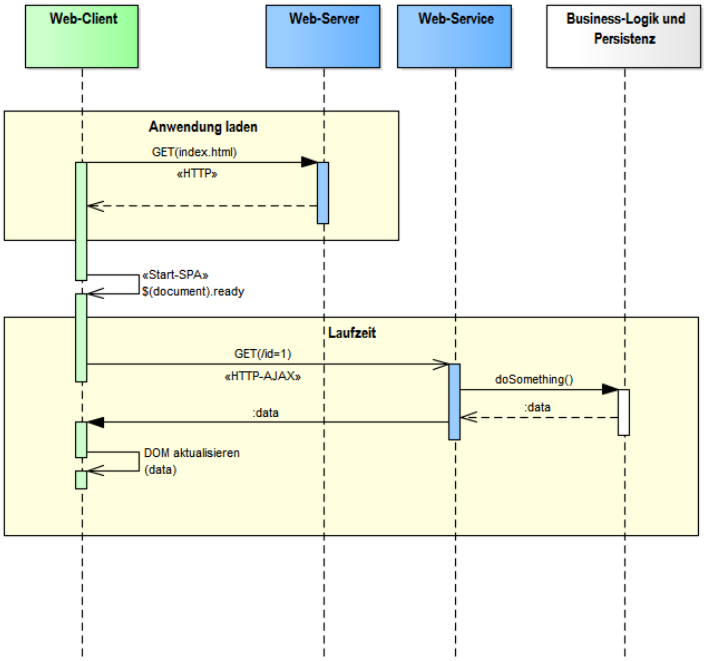


Abbildung 1.3.: Client-Server-Kommunikation bei Interaktion mit einer SPA
Wikimedia Commons (2015)

Diese Technik hat auch Nachteile: Zum einen ist der „Zurück“-Button in seiner Funktion eingeschränkt. Wenn der Browser nur ein einziges Dokument lädt, kann zwischen verschiedenen Stadien der Interaktion nicht über den Verlauf unterschieden werden.

Auch legt eine Ajax-basierte Webseite große Teile ihrer Benutzerführung in die Hände der implementierenden Entwickler*innen, was zu Brüchen mit der Erwartungshaltung des Nutzers führen kann: Jakob Nielsen führt als Beispiel hierfür Radio-Buttons an, die beim Anklicken unerwarteterweise sofort eine Aktion auslösen, anstatt eine Option zu setzen, die erst bei Betätigung eines OK-Buttons ausgelesen wird (Nielsen (1999)).

Die starke Einbindung von JavaScript bedingt natürlich eine gewisse Anfälligkeit für XSS- oder Cross-Site-Scripting. Angriffe dieser Kategorie stehen seit Jahren in der Top 10 der gefährlichsten Bedrohungen für Webanwendungen des Open Web Application Security Projects (OWASP Foundation, 2017, S. 4). Auf deren Platz 1 stehen die Injection-Angriffe, die insbesondere für eine SPA mit ihrer hohen Reaktivität und Interaktivität ein besonderes Thema darstellen: Nutzereingaben müssen angenommen, verarbeitet und dargestellt werden. Ein Framework muss daher neben der Bereitstellung der eigentlichen Funktion auch Sorge dafür tragen, solche Angriffe zu erschweren. In Angular, das im Projekt zum Einsatz kommt, sind hierfür bereits ab

Werk mehrere Mechanismen eingebaut (Angular Documentation, 2018, Guide: Security). Es existieren, vor allem für Versionen ab 2.0 (die vorherigen heißen AngularJS, die technisch noch deutlich anders funktioniert), erstaunlich wenig bekannte Sicherheitslücken - insbesondere angesichts der Verbreitung des Frameworks (https://cve.mitre.org (2018)).

1.4. Übersicht über die weitere Arbeit

In der Einleitung habe ich anhand prominenter Beispiele beispielhaft Gefahren aufgezeigt, die aus fehlerhafter Planung und/oder Umsetzung von IT-Sicherheitsmaßnahmen folgen. Ziel war es, die Wahrnehmung der Leserschaft für Konsequenzen typischer Fehler in Konzeption und Umsetzung von Aspekten der IT-Sicherheit und deren Häufigkeit zu schärfen sowie meine Motivation erklären.

Im Rahmen einer Begriffsdefinition erfolgt daraufhin u.a. eine Abgrenzung von IT- und Informationssicherheit. Diese scheinbar sehr ähnlich gelagerten Ausdrücke haben tatsächlich ausreichend unterschiedliche Bedeutungen, um dezidiert in Bezug zueinander gesetzt zu werden. Darauf folgend gebe ich einen Überblick über verschiedene Leitfäden zur IT-Sicherheit in der Softwareentwicklung. Diese werden zum Zwecke einer Übersicht zunächst allgemein vorgestellt. Im Anschluss daran stelle ich einige Verfahren des Risikomanagements heraus, die ich für besonders wichtig halte.

Das folgende dritte Kapitel enthält Überlegungen, wie IT-Sicherheit und agile Vorgehensweisen miteinander verbunden werden können. Ziel des ersten Abschnittes ist es, aus den im zweiten Kapitel zusammengetragenen Erkenntnissen einen Prozess zu extrahieren, mit dem ein bestehendes agiles Entwicklungsmodell sicherheitstechnisch, sowohl in Bezug auf das Produkt selbst als auch die Entwicklung, aufgewertet werden kann. Danach erläutere ich die Besonderheiten des bei meinem Arbeitgeber implementierten Entwicklungsmodells sowie der Anwendung, die geschützt werden soll. Ziel ist es hierbei nicht, alle ihre Einzelheiten aus allen Blickwinkeln zu beleuchten, sondern der Leserschaft einen Überblick zu geben, wo bei einem Anwendungsfall wie dem hier dargelegten die Hauptproblemfelder liegen und wie der Status quo aussieht.

Der Umsetzungsteil beschreibt, wie das vorgestellte Entwicklungsmodell methodisch erweitert und verbessert werden kann. An dessen Anfang steht eine Auseinandersetzung mit den Werten der Anwendung. Danach beschreibe ich die konkrete Umsetzung der vorher beschriebenen Verbesserungen und führe ihre Schritte anhand eines exemplarischen Beispiels einmal vor. Anschließend gehe ich gesondert auf technische Eigenheiten ein, die sich bei der Entwicklung einer SPA und mithilfe des Angular-Frameworks ergeben.

Im abschließenden Fazit nehme ich eine Bewertung vor, die den auf diesen Seiten verschriftlichten Vorgang in seiner Effektivität beurteilt sowie Probleme in der Umsetzung aufzeigt und gegebenenfalls deren Ursachen. Zuletzt gebe einen Ausblick auf Handlungsmöglichkeiten über den Rahmen dieser Arbeit hinaus und werde meine bei der Bearbeitung dieses Themas gewonnene Haltung darlegen, welchen Nutzen die einzelnen Interessengruppen aus dem auf diesen Seiten geschilderten Prozess ziehen können.

1.5. Technische Anmerkung zu Referenzen auf Bücher in EPUB-Form

In dieser Arbeit beziehe ich mich mehrfach auf E-Books im Format EPUB. Anders als bei PDFs, in denen Text fest einer Position im Dokument zugeordnet ist, besteht EPUB aus nahtlosem Fließtext und passt sich daher der gewählten Darstellungsgröße an. Dadurch verlieren Seitenzahlen bei EPUB ihre identifizierende Funktion. Um dennoch Textstellen eindeutig referenzieren zu können, wende ich eine von der Boldeian Library der Universität Oxford empfohlene Technik an. Bodleian-Libraries (2018) Diese lokalisiert Textstellen in einem Dokument über die Nummerierung von Absätzen und Unterabsätzen. Eine Software wie z.B. Calibre bietet hierfür einen sog. „Reference Mode“, der die Referenzen automatisch und reproduzierbar zählt.

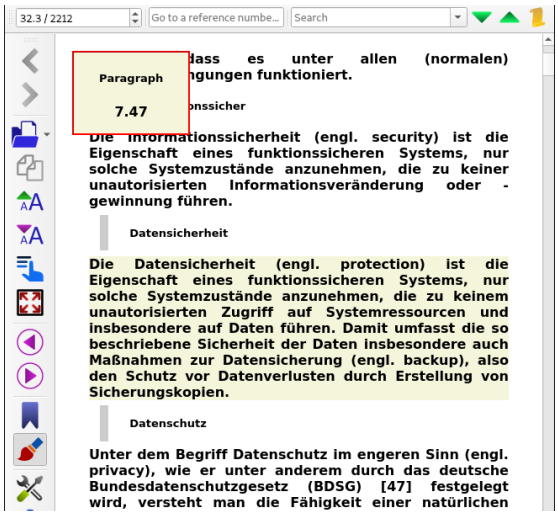


Abbildung 1.4.: Calibre E-Book viewer: Reference Mode

2. Bestandsaufnahme zur IT-Sicherheit in der agilen Softwareentwicklung

In diesem Kapitel stelle ich gewissermaßen den Werkzeugkoffer für die nachfolgenden Betrachtungen zusammen. Zunächst führe ich wichtige Begriffe ein. Dann stelle ich über bereits allgemein akzeptierte Best Practices hinaus deutende zentrale Erkenntnisse aus drei Leitfäden zur IT-Sicherheit in der Softwareentwicklung zusammen, die von besonderem Wert für die Beurteilung des untersuchten Projektes und der Entwicklungsprozesse sind. Besonders nehme ich dabei das Risikomanagement in den Blick, da ich hier das größte Potential sehe für die Schwerpunktsetzung meiner Arbeit auf größtmöglichen Gewinn für den hauseigenen Entwicklungsprozess. Genauer führe ich diese Entscheidung zu Beginn von 2.3 aus.

2.1. Begriffseinführungen

Einige Ausdrücke im Themengebiet der IT-Sicherheit sind oft unpräzise definiert, werden von verschiedenen Autor*innen unterschiedlich ausgelegt oder sind u.U. erklärungsbedürftig für Leser*innen, deren Hauptbetätigungsfeld ein außerhalb der IT-Sicherheit liegt. Daher lege ich im folgenden Abschnitt die genaue Semantik von Begriffen fest, nach der sie im Verlauf der Arbeit verwendet werden.

2.1.1. Subjekt und Objekt

Dieses Begriffspaar ist in der Fachliteratur weit verbreitet und bietet meines Erachtens eine sinnvolle unterscheidende Beschreibung der Entitäten in IT-Systemen.

Ein passives Objekt, etwa eine Datei, speichert und repräsentiert die Daten eines Systems. Ein aktives Objekt, z.B. ein Prozess, kann Daten nicht nur speichern, sondern auch verändern. Subjekt hingegen ist alles, was die Daten eines Systems nutzt, also etwa die Benutzer*innen, ein anderes System, oder ein aktives Objekt im Auftrag eines Nutzers. (Eckert, 2014, Abs. 7.23)

2.1.2. Risiko und Bedrohung/Gefährdung

Im Alltag wird innerhalb dieser Begriffe oft nicht differenziert, in der Fachsprache der IT-Sicherheit jedoch sehr wohl.

Eine Bedrohung ist nach den IT-Grundschutzkatalogen des Bundesministeriums für Sicherheit in der Informationstechnik ein Umstand oder Ereignis, das eines der Schutzgüter beeinträchtigen kann; z.B. höhere Gewalt, technisches Versagen oder Handeln in Schadabsicht. Durch das Wirken über eine Schwachstelle wird aus einer Bedrohung eine Gefährdung: Viren im Internet sind allgemein betrachtet eine Bedrohung. Nutzt eines dieser Programme aber eine konkrete, im Zielsystem vorhandene Schwachstelle, spricht man von einer Gefährdung. „Risiko“ erweitert die vorgenannten Definitionen um eine Bewertung des Schadensszenarios. (BSI, 2016, S. 99, 101 & 106)

2.1.3. Schutzgüter/Schutzziele

Zur Differenzierung wird in der Frage, wovor ein System eigentlich geschützt werden soll, wird zwischen den sog. Schutzgütern bzw. Schutzzielen unterschieden. Das Verständnis derselben wird im Verlauf dieser Arbeit an verschiedenen Stellen hilfreich sein, weswegen ich sie kurz erläutern werde. Ich nehme dabei eine Einteilung in allgemeine und spezielle Schutzziele vor. Diese findet sich unter anderem im OWASP DevGuide (OWASP Foundation, 2018a, S. 12). Ich halte sie für gewinnbringend, da sich die allgemeinen Schutzziele bei genauer Betrachtung in praktisch jeder Anwendung mit Mehrbenutzerschnittstelle finden dürften, die speziellen hingegen nicht.

Confidentiality/Vertraulichkeit Unterscheidung von Datenzugriffen und Übertragungen in berechtigt und nicht berechtigt, Verhinderung von letzterem

Integrity/Integrität Schutz von Daten & Systemen vor unautorisierter Veränderung & Nachverfolgbarkeit von Änderungen

Availability/Verfügbarkeit Gewährleistung von Zugriff, Verhinderung von Ausfällen

Die nun beschriebenen speziellen Schutzziele folgen nicht unmittelbar aus den allgemeinen, und es gibt durchaus Systeme, bei denen sie nur eine untergeordnete oder sogar gar keine Rolle spielen.

Authenticity/Authentizität Sicherstellung der „Echtheit“ und damit Vertrauenswürdigkeit von Daten

Non Repudiation/Nichtabstreitbarkeit Sichere Attribuierbarkeit von Aktionen zu einzelnen Subjekten, so dass ein solches nicht abstreiten kann, eine Aktion durchgeführt zu haben

Accountability/Zurechenbarkeit Verlässliche Zuordnung von Objekten und Aktionen zu einzelnen Subjekten

Anonymity/Anonymität Das diametrale Gegenteil des Vorgenannten: Verhinderung der Zuordnung von Subjekten zu Aktionen und/oder Objekten
(Eckert, 2014, Abs. 7.60)

Ersichtlich ist, dass die letzten beiden Ziele dieser Kategorie in direktem Konflikt zueinander stehen. Je nach Anwendungsfall kann das eine Ziel höher zu gewichten sein als das andere: Beim Design eines Bankingsystem wird wohl beispielsweise Zurechenbarkeit ein hoher Wert beigemessen werden - bei einem Whistleblower-Postfach etwa hingegen der Anonymität.

2.1.4. Abgrenzung IT-, Daten-, Cyber- und Informationssicherheit

Bisweilen findet man alle obenstehenden Begriffe in synonymer Verwendung. Für eine genauere Betrachtung muss jedoch zunächst der Begriff „Sicherheit“ für diesen Kontext geklärt werden. Eine allgemeine Positivdefinition ist insbesondere in der deutschen Sprache nicht ohne Schwierigkeiten zu leisten. Im Englischen wird etwa zwischen „Safety“ und „Security“ unterschieden: Das erste wird i.d.R. etwas sperrig mit „funktionssicher“ übersetzt, was bedeutet, dass ein System nur Zustände einnimmt, die in seiner Spezifikation auch vorgesehen sind. Mit anderen Worten: Das System funktioniert wie geplant, bzw. die Implementation stimmt mit der Spezifikation überein. Absicherungen hinsichtlich der Safety bedeuten damit die Verhütung von Unfällen oder Störungen. Security ist anders gelagert: Hier wird ein System gegen intentionale Störungen, also Angriffe, abgesichert. (Eckert, 2014, Abs. 7.57)

Als Beispiel aus der physischen Welt sei das Steuerungssystem eines Flugzeugs genannt: Dieses System ist funktionssicher, wenn Kommandos aus dem Cockpit die korrekten Aktionen in den Aktoren der Maschine auslösen. Ein Angriff könnte diese Funktion jedoch unterbinden, wenn sich womöglich jemand Zugriff zu den elektrischen Leitungen des Flugzeugs verschafft und die Steuerimpulse blockiert.

Auch wäre zu unterscheiden zwischen objektiver und subjektiver Sicherheit: Das eine bezeichnet Sicherheitseigenschaften, die mess- und quantifizierbar sind, letzteres solche, die ausschließlich die Empfindung von Sicherheit stärken.

Bei Claudia Eckert findet man darüber hinaus noch einen dritten Aspekt der Sicherheit:

> „Die Datensicherheit (engl. protection) ist die Eigenschaft eines funktionssicheren Systems, nur solche Systemzustände anzunehmen, die zu keinem unautorisierten Zugriff auf Systemressourcen und insbesondere auf Daten führen." (Eckert, 2014, Abs. 7.47)

In dieser Definition ist der autorisierte Zugriff das entscheidende Begriff: Ein System gilt als datensicher, wenn gewährleistet werden kann, dass ein Subjekt genau dann Zugriff auf ein Objekt erhält, wenn es seine (für den Zugriff berechtigte) Identität nachweisen kann. Als Zugriff zu verstehen ist, der Subjekt-Objekt-Abgrenzung folgend, das Nutzen der Schnittstelle eines Systems zur Interaktion mit dessen Daten. Zu erkennen ist ein direkter Bezug zu den Schutzzielen: Für einen Zugriff muss die Verfügbarkeit des Systems sichergestellt sein, und Datensicherheit ist nur unter der Voraussetzung der Authentifizierung der Subjekte möglich.

Dieser Definition ist eine Zweckbindung des Zugriffs hinzuzufügen: Polizist*innen etwa dürfen gewisse Daten einsehen, jedoch nur im Rahmen ihrer jeweiligen Tätigkeit und nicht z.B. zur Überwachung von Beziehungspartnern.

Doch auch sind Daten nicht immer Informationen: Letztere ergeben sich aus der Interpretation von Ersteren. So kann etwa das Datum einer nackten Zahl durch Interpretation als „Kontostand" zu einer Information werden. (Eckert, 2014, Abs. 7.27) Weder Informationen noch Daten müssen hierbei jedoch elektronisch, sondern können auch auf Papier oder nur in Köpfen vorliegen. (BSI, 2016, S. 112)

Folglich ist das Manipulieren von Informationen nur dann möglich, wenn Daten manipuliert werden können. Aus der Möglichkeit zur Manipulation von Daten ergibt sich aber nicht notwendigerweise die intentionale Manipulierbarkeit von Informationen: Ein System könnte Informationen aus redundanten Datenquellen interpretieren oder die Integrität der Daten beispielsweise über Hashes verifizieren und inkonsistente Daten verwerfen. Somit ist zwar jedes datensichere System ein informationssicheres, aber nicht jedes informationssichere System ein datensicheres. Bei strenger Betrachtung ist außerdem ein System, dessen Informationen über unsichere Kanäle wie z.B. das Internet fließen, nie datensicher. Ist jedoch die Übertragung kryptografisch gesichert und damit die Interpretationsvorschrift der Daten vor Mithörenden hinreichend geheim gehalten, kann das System dennoch als informationssicher gelten.

Für ein informationssicheres System sind demzufolge die Schutzziele Authentizität und Integrität essentiell; sie stellen in Kombination den Schutz der Informationen im System vor unberechtigter Manipulation sicher. Weitere Schutzziele wie etwa die Verfügbarkeit sind deswegen nicht weniger wichtig, aber nicht konstituierend für die Informationssicherheit.

Der Begriff IT-Sicherheit ist breiter aufgestellt. Er bezeichnet die Aufgabe, die ein IT-System betreibende Organisation und deren Werte vor Schäden zu bewahren, die bei Verletzung der „CIA"-Schutzziele Vertraulichkeit, Integrität und Verfügbarkeit des Systems und seiner Daten durch absichtsvolles Handeln Dritter entstehen. Einfacher ausgedrückt: IT-Sicherheit bedeutet die Umsetzung aller drei Schutzziele. (Eckert, 2014, Abs. 7.4)

„Cybersicherheit" ist hingegen nicht stringent definiert und taucht meist an Stellen auf, wo eigentlich IT-Sicherheit gemeint ist, das Wort aber anscheinend aufgewertet werden soll (vgl. z.B. bsi.bund.de). Der Ausdruck qualifiziert sich damit weitestgehend als Schlagwort bzw. Buzzword.

2.1.5. Verfahren, Konzepte, Modelle, Techniken

Um Unklarheiten zu vermeiden, was ich in den kommenden Kapiteln mit den obenstehenden Ausdrücken meine, definiere ich hier meine Verwendung dieser Nomenklatur. Ziel dieses Abschnittes ist keine Begriffsdefinition von lexikalischem Wert, sondern das Ausräumen begrifflicher Unschärfen.

Verfahren Mit „Verfahren" meine ich eine Festlegung im Entwicklungsprozess, wie z.B. dass sich ein Team darauf einigt, regelmäßige Code-Reviews zu veranstalten. Da der Fokus dieser Arbeit auf Prozessen in der Anwendungsentwicklung liegt, ist dies ein zentraler Begriff dieses Kapitels.

Technik Nicht zu verwechseln mit „Technologie", ist unter diesem Begriff ein bestimmtes Vorgehen zu verstehen. So fällt etwa eine Firewall selbst in die Kategorie der Technologien. Was hingegen mit ihr getan wird, etwa Filterung von Netzwerkzugriffen oder Deep Packet Inspection, ist eine Technik.

Sicherheitsmodell Wie bei jedem anderen Modell auch steht hier eine vereinfachende Abstraktion eines Ausschnitts der Wirklichkeit im Mittelpunkt. Anhand dieser sollen sich leichter Eigenschaften des Modellierungsgegenstandes ablesen lassen, etwa neuralgische Punkte eines Prozesses, die im konkreten Fall des Sicherheitsmodells erhöhter Aufmerksamkeit bedürfen.

Prominente Beispiele für Sicherheitsmodelle finden sich in der Zugriffskontrolle: Da das zugrundeliegende Problem in praktisch jedem größeren IT-System gelöst werden muss, existieren hierfür gleich mehrere bewährte Modelle. Exemplarisch zu nennen wären hier etwa das Zugriffsmatrix-Modell oder La-Padula. (Eckert, 2014, Abs. 13.2)

Sicherheitskonzept Als ein solches verstehe ich in diesem Kontext eine projektweite Festlegung zu sicherheitsrelevanten Fragen. Die Art und Weise, wie man zu solchen Festlegungen kommt, kann dabei ebenso unterschiedlich sein wie deren eigentlicher Inhalt. Jedes Softwareprojekt hat nach diesem Verständnis ein solches Konzept, da selbst die formale Abwesenheit eines solchen eine Haltung zu dessen Gegenstand darstellt. Diese Lesart des Begriffes deckt sich weitestgehend mit der des Bundesministeriums für Sicherheit in der Informationstechnik (BSI, 2016, S. 107).

Abzugrenzen ist der Begriff damit beispielsweise von in Programmiersprachen eingebauten Techniken etwa zur automatischen Speicherbereinigung, die bisweilen auch als Sicherheitskonzept bezeichnet werden.

2.1.6. Risikomanagement in der Softwareentwicklung

Da ich Prozesse behandle und dabei auf Verallgemeinerbarkeit abziele, ist das Risikomanagement von besonderem Interesse für meine Erwägungen. Dass absolute Sicherheit in der IT nie erreicht werden kann, ist eine Binsenweisheit. Daraus folgt, dass auch bei noch so großen Sicherheitsanstrengungen stets Gefahren drohen; man spricht hier von einem Restrisiko. Aufgabe des Risikomanagements muss es sein, Orientierung in der Frage zu geben, wie das Restrisiko unter wirtschaftlichem Einsatz verfügbarer Ressourcen minimiert werden kann. Die International Organization for Standardization (ISO) beschreibt den Vorgang des Risikomanagements als bestehend aus Analyse und Bewertung von Risiken, wobei der erste Schritt wiederum zu unterteilen ist in Identifikation und Einschätzung eines Risikos (International Organization for Standardization, 2008, S. 9).

2.2. Evaluierung dreier Leitfäden

IT-Systeme sind heute stärker verteilt und vernetzt als vor einigen Jahrzehnten, und bieten damit eine drastisch erhöhte Angriffsfläche. Wie gezeigt werden leider auch heute noch Softwareprojekte teilweise unter Missachtung ihrer Exponiertheit und Angreifbarkeit geplant, umgesetzt und betrieben.

Prominente Branchenmitglieder versuchen heute, Lehren aus den Fehlern der Vergangenheit zu ziehen. Nachfolgend stelle ich eine Auswahl von Dokumenten vor, die diese Informationen zu bündeln versuchen. Dabei strebe ich danach, in vertretbarem Umfang eine Zusammenfassung einer Auswahl aus der relevanten Fachliteratur zu geben. Da mein Ziel darin besteht, einen konkreten, agilen Entwicklungsprozess und die daraus hervorgehende Anwendung zu untersuchen und gegebenenfalls abzusichern, sind für mich prozessbezogene und auf moderne Vorgehensweisen anwendbare Hinweise am wertvollsten und rein theoretische Werke weniger interessant. Meine Wahl fiel daher auf die IT-Grundschutzkataloge des Bundesamts für Sicherheit in der Informationstechnik, den Secure Software Development Lifecycle von Microsoft und das Open Web Application Security Project, kurz OWASP.

Die Grundschutzkataloge sind besonders breit aufgestellt und daher entsprechend umfangreich. Bei weitem nicht jeder Artikel der Kataloge ist relevant für die Absicherung einer agil entwickelten Webanwendung, jedoch finden sich zu praktisch allen Aspekten der Softwareentwicklung entsprechende Handreichungen. Da die Grundschutzkataloge von einer inländischen Behörde erarbeitet wurden, nehmen Sie außerdem Besonderheiten der Softwareentwicklung für den deutschen Markt in den Blick.

Der Secure Software Development Lifecycle hingegen stammt aus einem Softwarehaus, das im Ruf steht, in der Vergangenheit zwar besonders viel falsch gemacht, daraus aber die richtigen Schlüsse gezogen zu haben. Auch ist Microsofts Portfolio breit aufgestellt und reicht von Betriebssystemen für verschiedenste Hardwareplattformen über Büroapplikationen bis zu Webanwendungen. Die Autoren können dementsprechend aus einem enormen Erfahrungsschatz schöpfen (Howard und Lipner, 2006, Abs 3.5).

Das OWASP ist eine Non-Profit-Organisation, die verschiedene Dokumente und Tools bereitstellt, um die Sicherheit in IT-Anwendungen zu verbessern. Man versteht sich als eine Gemeinschaft Gleichgesinnter, deren Hinweise herstellerübergreifend gültig sein sollen. owasp.org (2018) Wie der Name bereits verrät, geht es dem OWASP allen voran um Webanwendungen.

2.2.1. BSI-Grundschutz-Kataloge

Das Bundesamt für Sicherheit in der Informationstechnik (kurz „BSI“) bietet mit den IT-Grundschutzkatalogen auf über 5000 Seiten ein besonders detailliertes Dokument. Dieses richten sich primär an die Verantwortlichen der Umsetzung von operativen Sicherheitsmaßnahmen. Supplementiert wird es durch ans Management gerichtete Dokumente in Form von Online-Schulungen und als PDF abrufbaren Dokumenten. Diese nehmen jedoch eine unternehmensstrategische Perspektive auf IT-Sicherheit ein und sind daher für meine Betrachtung weniger relevant.

Der Fokus der Grundschutzkataloge ist der weiteste der hier vorgestellten Werke. Aufgrund des enormen Umfangs kann hier nur punktuell eine inhaltliche Auseinandersetzung geleistet werden, ohne den Rahmen dieser Arbeit zu sprengen.

Das Gesamtwerk gliedert sich in Baustein-, Gefährdungs- und Maßnahmenkataloge. Verweise auf Artikel aus diesen Katalogen erfolgen über Angabe der Katalogskategorie (B für Baustein-, G für Gefährdungs- und M für Maßnahmenkataloge), eine führende Zahl für den Katalog und eine weitere Zahl für den Artikel. Beispiel: G 0.1 verweist auf die Gefährdungskataloge, dort den ersten (oder eben „nullten“) Katalog „Elementare Gefährdungen“ und darin auf den ersten Artikel „Feuer“.

Bausteinkataloge Die Bausteinkataloge enthalten Kurzbeschreibungen der verschiedenen Aspekte von IT-Sicherheit. Zu jedem Baustein erfolgt eine Einschätzung der Bedrohungslage und ein daraus abgeleitetes Bündel an Maßnahmen zur Verhütung des Schadenfalles. In den Bedrohungen wie auch den Maßnahmen wird auf die zugehörigen Einträge in den Gefährdungs- bzw. Maßnahmenkatalogen verwiesen. Die Bausteinkataloge setzen sich aus folgenden Schichten zusammen:

1. Übergreifende Aspekte
2. Infrastruktur
3. IT-Systeme
4. Netze
5. Anwendungen

Den Bezeichnungen ist zu entnehmen, dass sich für meine Schwerpunktsetzung auf den Entwicklungsprozess Schicht zwei bis vier zu explizit auf konkrete Hard- sowie Softwareinstallationen beziehen und daher Schicht 1 aus prozessualer Sicht am interessantesten ist (auf die fünfte gehe ich anschließend kurz ein).

Der Katalog zu übergreifenden Aspekten gliedert sich in 18 Bausteine, u.a. Sicherheitsmanagement, Organisation und Notfallmanagement. Dem selbst formulierten ganzheitlichen Anspruch der Kataloge folgend, wird in jedem der Bausteine festgehalten, dass seine Umsetzung nicht isoliert erfolgen kann, sondern ebenfalls das ins Werk setzen anderer Bausteine erforderlich ist. (BSI, 2016, S. 111 ff.)

Besonders hervorheben möchte ich noch einen Baustein der Schicht 5: Anwendungen.

B 5.21 Webanwendungen Dieser Baustein verdient besondere Betrachtung, da sein Fokus mit dem dieser Arbeit zusammenfällt. Das BSI definiert eine Webanwendung als Software, deren Funktionen und dynamische Inhalte über eine Webschnittstelle zur Verfügung gestellt werden. Die Eigenschaften der hierzu verwendeten Frameworks hinsichtlich in der IT-Sicherheit müssen bereits in der Planungs- und Architekturphase berücksichtigt werden. Ebenfalls identifiziert dieser Baustein mit Web- und Applikationsservern sowie Hintergrunddienste wie Datenbanken mehrere praktisch immer vorhandene Systemkomponenten.

Die zu lösenden Probleme decken sich in Teilen mit den bereits eingeführten Schutzzielen und werden folgendermaßen identifiziert bzw. formuliert:

- Authentisierung
- Autorisierung
- Validierung von Ein- und Ausgabedaten
- Session-Management (Verwalten eines Sitzungsstatus über das zustandslose HTTP hinaus)
- Fehlerbehandlung (Funktionssicherheit auch im Fehlerfall)
- Protokollierung

Die Gefährdungslage von Webanwendungen ist dem BSI zufolge im Grunde identisch mit der für allgemeine Geschäftsanwendungen, da erstere genauso Geschäftslogiken nachbilden wie letztere. Hinzu kommen jedoch einige spezifische Bedrohungen für Anwendungen, deren Endpunkte öffentlich exponiert sind (z.B. G 5.87: „Web-Spoofing“).

Unter den empfohlenen Maßnahmen befinden sich u.a. die zum sicheren Betrieb von Servern im Allgemeinen und Webservern im Speziellen sowie Hinweise zum Datenschutz für den Fall, dass die Anwendung personenbezogene Daten verarbeitet.

Zur Aufrechterhaltung des definierten Sicherheitsniveaus sei es darüber hinaus erforderlich, Prozesse einzurichten, um über Sicherheitslücken informiert zu bleiben und Softwareupdates einzuspielen. Zusätzlich soll sichergestellt werden, dass die Anwendung Unbefugten gegenüber keine sicherheitsrelevanten Informationen preisgibt (BSI, 2016, S. 414 ff.).

Gefährdungskataloge

Die einzelnen Artikel formulieren oder referenzieren keine Gegenmaßnahmen, sondern beinhalten allein die Beschreibung der Gefahren. Die Kataloge sind sehr umfangreich, weshalb

ich mich bei ihnen auf eine grobe Inhaltsangabe beschränken und nur exemplarisch einen einzelnen Artikel vorstellen werde.

G 0: Elementare Gefährdungen Dieser Katalog enthält „verallgemeinerte und auf das wesentlicher reduzierte grundlegende Gefährdungen“ (BSI, 2016, S. 47) und wird vom BSI als mögliche Grundlage für Risikoanalysen vorgestellt. Die Bandbreite reicht dabei von physischen Gefahren wie Feuer, Verschmutzung und Naturkatastrophen über technische Probleme, etwa Infrastrukturausfälle, Störstrahlung und Ressourcenmangel, sowie organisatorische Themen, z.B. Personalausfall oder Fehlplanungen, bis hin zu intentionalen Angriffen wie Social Engineering, Erpressung und Ausspähung von Informationen (BSI, 2016, S. 443 ff.).

G 1: „Höhere Gewalt“ Der Fokus dieses Katalogs liegt auf Bedrohungen, die auch bei äußerster Sorgfalt nicht ausgeräumt werden können. Dies beinhaltet u.a. Natur- und sonstige Katastrophen, Beeinträchtigung durch Großveranstaltungen, Ausfall von Dienstleistern, Personal oder Infrastrukturen. (BSI, 2016, S. 491 ff.).

G 2: „Organisatorische Mängel“ Hier werden Gefahren eingehender beschrieben, die sowohl planerischer, juristischer wie auch technischer Natur sein können. Beispielhaft zu nennen wären für diesen Katalog fehlende oder unzureichende Regeln, Betriebsmittel, Dokumentation oder Wartung, unpassende Dimensionierung von Infrastruktur oder Hardwarekapazitäten und Fehler in Kosten-Nutzen-Kalkulationen (BSI, 2016, S. 511 ff.).

G 3: „Menschliche Fehlhandlungen“ Dieser Katalog listet Fehler auf, die Menschen in der Bedienung technischer Systeme, aber auch beim Ausführen von Prozessschritten unterlaufen können. Gemein ist allen Artikeln des Katalogs in Abgrenzung zu G 5: „Vorsätzliche Handlungen“, dass sie eben ohne Vorsatz stattfinden, wenn auch bisweilen fahrlässig. Der Katalog nennt z.B. die nicht intentionale Weitergabe von Interna, versehentliches Beschädigen oder Ausschalten von Geräten, fehlerhafte Konfiguration oder fehlende Akzeptanz für Sicherheitsmaßnahmen (BSI, 2016, S. 795 ff.).

G 4: „Technisches Versagen“ Hier finden sich Gefährdungen mit technischer Ursache. Es gibt Schnittmengen zu anderen Katalogen: Der Ausfall der Stromversorgung etwa wird ebenfalls in G 1: Höhere Gewalt behandelt. Darüber hinaus finden sich hier Datenverluste durch überalterte oder unbemerkt vollgelaufene Speichermedien ebenso wie schädliche Stromflüsse, unsichere Default-Konfigurationen oder allgemein Schwachstellen in Software. (BSI, 2016, S. 952 ff.)

G 5: „Vorsätzliche Handlungen“ Die in diesem Katalog beschriebenen Gefahren beschreiben mögliche Angriffe auf IT-Systeme mit unterschiedlichen Zielen: Informationsabflüsse, finanziell motivierter Betrug oder Funktionsausfall sind nur einige der möglichen Intentionen. Diese sind naturgemäß zum Teil sehr technisch, wie etwa Missbrauch von Bestandteilen des IP-Protokolls oder spezieller Betriebssystemdienste, an anderer Stelle jedoch auch diesbezüglich recht niedrigschwellig, wie z.B. Vandalismus oder Einbruch. Darüber hinaus finden sich in diesem Katalog „Klassiker“ der IT-Sicherheit wie Trojaner, Social Engineering, Makro-Viren oder Passwort-Bruteforcing. (BSI, 2016, S. 1091 ff.)

Maßnahmenkataloge

Diese Kataloge enthalten die Handlungsanweisungen, welche in den vorherigen Katalogen referenziert werden. Die einzelnen Kataloge heißen wie folgt:

1. Infrastruktur
2. Organisation
3. Personal
4. Hardware und Software
5. Kommunikation
6. Notfallvorsorge

Eine Maßnahme beginnt stets mit einer Feststellung, wer verantwortlich ist, sie zu initiieren und wer sie umzusetzen hat. Es folgt eine kurze Begründung, warum die Maßnahme sinnvoll ist bzw. wogegen sie schützt und die Beschreibung der Maßnahme selbst mit Anforderungen und Durchführungshinweisen, sowie Referenzen auf andere Maßnahmen. Am Ende steht eine Liste von Prüffragen, die nach erfolgreicher Umsetzung der Maßnahme positiv beantwortet werden können sollen.

Die Maßnahmenkataloge sind anteilsmäßig das Schwergewicht der Sammlung mit 1609 Maßnahmen auf über 3700 Seiten, weshalb ich auf die Vorstellung einzelner Artikel an dieser Stelle verzichte und ggf. auf die Kataloge verweisen werde, wenn ich mich auf sie beziehe.

2.2.2. Security Software Development Lifecycle

Dieser Leitfaden (im Folgenden „SDL“ genannt) wurde von Microsoft erstmalig 2004 veröffentlicht, ist 2006 als eBook erschienen und wird seitdem halbjährlich aktualisiert. Wie der

Name bereits verrät, betrachtet er den gesamten Zyklus der Entwicklung einer Software von der Anforderungsaufnahme bis zum Betrieb & Support. Die Autoren liefern eine Metrik, um zu bestimmen, ob sich ein Projekt für die Methodiken des SDL eignet, benennen jedoch an keiner Stelle, auf welche Art von Software der SDL beschränkt sei. Damit ist auch der SDL von der Ausrichtung her ein allgemein gehaltenes Konzept.

Das in Teilen bis heute bestehende, von den SDL-Autoren bezeugte schlechte Image Microsofts, was die Qualität und Sicherheit seiner Softwareprodukte betrifft, stammt zu wesentlichen Teilen aus den Jahren unmittelbar vor dem Erscheinen des SDL (Howard und Lipner, 2006, Abs. 4.7). Da außerdem Howard und Lipner selbst schreiben, dass die drastische Kurswende zwischen 2002 und 2005 dem SDL zu verdanken sei, lässt sich mit einiger Rechtfertigung spekulieren, dass hierfür weniger einzelne Entwickler*innen verantwortlich zeichneten, sondern ein Problem in den Prozessen vorlag. In dieses Bild fügt sich ein, dass sich der SDL nicht explizit an Entwickler*innen, sondern primär Designer*innen/Architekt*innen und die Managementebene richtet. Diese Trennung verschwimmt in der agilen Softwareentwicklung, was den SDL für diese Arbeit besonders interessant macht.

Der SDL besteht aus drei Teilen: Howard und Lipner begründen zunächst die Notwendigkeit und Vorteile eines solchen Dokumentes. Es folgt im Hauptteil das eigentliche Konzept, abschließend werden weiterführende Themen und Details behandelt. Im Kern wird der klassische Entwicklungsprozess bei Microsoft beschrieben und um spezielle, sicherheitsfokussierte Aktivitäten erweitert. In der Konsequenz besteht der Gesamtprozess aus diesen Stadien folgenden Inhalts, sinngemäß übersetzt und zusammengefasst:

0. **Education & Awareness (Schulung & Problembewusstsein)**: Fortwährende Weiterbildung, Arten von Security Trainings, Fragen zur Quantifizierung von Kompetenz, Implementation eigener Trainings, Schlüsselfaktoren für Erfolg & Metriken

1. **Project Inception (Projektbeginn)**: Ernennung einer/eines Hauptverantwortlichen für Security, Aufbauen eines Führungsteams, Sicherheitsmaßnahmen ins Bugtracking einbinden, Schwere von zu behandelnden Bugs definieren

2. **Define and Follow Design Best Practices (definieren und befolgen bewährter Methoden)**: Verbreitete Leitsätze zu sicherem Softwaredesign, Analyse und Reduzierung von Angriffsflächen

3. **Product Risk Assessment (Produktrisikoeinschätzung)**: Abschätzung von Sicherheitsrisiken, Bewertung der Ansprüche an die Privatsphäre

4. **Risk Analysis (Risikoanalyse, Hauptteil: Bedrohungsmodellierungen)**: Ergebnisse der Bedrohungsmodellierung festhalten, sich auf Wesentliches beschränken, Durchführungsschritte und Prozess der Modellierung, Unterstützung von Code Review und Testing durch Modelle
5. **Creating Security Documents, Tools, and Best Practices for Customers (Erstellen von Sicherheitsdukumentation, Werkzeugen und bewährten Methoden für Kunden)**: Sinn und Zweck von Dokumentation und Tools, Präskriptive Dokumentation bewährter Ansätze, Erstellung von Werkzeugen
6. **Secure Coding Policies (Strategien zur sicheren Programmierung)**: Verwendung aktueller Versionen von Compilern und Hilfswerkzeugen, Hilfestellungen des Compilers nutzen, Werkzeuge der Codeanalyse, Vermeidung ausnutzbarer Codekonstrukte & Designs
7. **Secure Testing Policies (Strategien zum sicheren Testen)**: Fuzzing, Penetrationstests, Laufzeitverifikation, ggf. Aktualisierung der Bedrohungsmodelle und Angriffsflächen
8. **The Security Push (Sicherheitsinitiativen)**: Vorbereitung und Übungen, Aktualisierung von Bedrohungsmodellen, Testen, Bereinigen von Angriffsflächen und Dokumentation
9. **The Final Security Review (Die abschließende Security-Prüfung)**: Koordination im Team, Überprüfung von Bedrohungsmodellen sowie nicht behobenen Bugs, Nachbereitung des Reviews
10. **Security Response Planning (Vorbereitung auf Umgang mit Sicherheitslücken)**: Vorbereitung von Reaktionen, Teamkompetenzen
11. **Product Release (Veröffentlichung)**: Durchführung der firmeninternen Prozeduren zum Release
12. **Security Response Execution (Ausführen der in Kapitel 10 beschlossenen Maßnahmen)**: Wie man festgelegten Plänen folgt, Umgang mit unerwarteten Vorfällen, was ausgelassen werden darf

Im dritten Teil des SDL machen die Autoren in einem eigenen Kapitel Vorschläge, wie der SDL und agile Vorgehensmodelle miteinander kombiniert werden können. Zum Erscheinungszeitpunkt des SDL war der Entwicklungsprozess bei Microsoft noch mehrheitlich traditionell

linear organisiert und agile Methoden noch nicht so weit verbreitet wie heute. Bei Microsoft war man sich offenbar nicht sicher, dass mit ihnen größere Projekte überhaupt zu handhaben wären (Howard und Lipner, 2006, Abs. 42.5). Mit Blick auf den Fokus dieser Arbeit stelle ich hier einige der wesentlichen Vorschläge der Autoren heraus. Besondere Aufmerksamkeit widme ich hierbei Aspekten, die aus meiner Sicht nicht bereits als Best Practices in der Breite angekommen sind.

Wissensbildung Insbesondere empfiehlt der SDL, in agilen Projekten die Weiterbildung von Entwickler*innen in Sicherheits- und Privatsphärefragen ernstzunehmen. Hier stünden verhältnismäßig geringe Kosten großen wirtschaftlichen Risiken gegenüber - insbesondere, da in agilen Teams auch kritische Entscheidungen bisweilen von dessen Mitgliedern anstatt vom Management getroffen werden. Beispiel für ein finanziell erhebliches Risiko können unwissentlich begangene Verstöße gegen die DSGVO sein.

Risikoanalyse Da die Zyklen in agilen Projekten viel kürzer sind als in traditionellen, sind auch deren Architekturen stärker in Bewegung. Dies stellt ein Problem für klassische Methoden der Risikoanalyse dar, die etwa Flussdiagrammzeichnungen der Architektur zu Hilfe nehmen. Diese entstehen bei Scrum oder Xtreme Programming im Vergleich zu traditionellen Entwicklungsmodellen jedoch seltener, später oder in eher rudimentärer, isolierter Form. Der SDL rät daher, beispielsweise dezidierte Design-Sprints einzulegen.

Richtlinien für sicheres Programmieren und Testen Howard und Lipner sehen die Notwendigkeit, in agilen Projekten Codingrichtlinien festzulegen und empfehlen, mit statischen Analysetools deren Umsetzung auch zu erzwingen. Positiv stellen sie heraus, dass Extreme Programming bereits vorsieht, dass für gefundene Bugs stets Tests zu schreiben sind, und legen Wert darauf, dieses Paradigma insbesondere für sicherheitsrelevante Bugs anzuwenden. Ebenfalls empfiehlt der SDL Defensive Programmierung und tägliche Fuzzing-Tests.

Security Response Execution In den meisten agilen Vorgehensmodellen wird ein gefundener Bug in der nächsten Iteration behoben. Dieses Vorgehen halten die Autoren ungeeignet für sicherheitsrelevante Fehler, da etwa eine unbeabsichtigterweise exponierte Datenbank sofort und nicht erst im nächsten Sprint geschützt werden muss. Um den Prozess dadurch nicht zu unterlaufen, raten sie zum Einlegen eines sog. Spikes, die normalerweise dazu dienen, schwer zu schätzende Aufwände wie etwa das Einarbeiten in neue Technologien abzubilden.

Anschließend geben Howard und Lipner mit umgekehrter Blickrichtung an, was man für agile Vorgehensweisen aus dem SDL ableiten kann. So sollen etwa beim Schreiben der User Stories auch die Sicherheitsanforderungen des Kunden berücksichtigt werden, indem sie als Constraints gemäß „User Stories Applied“ von Mike Cohn behandelt werden (Cohn, 2004, S. 77 ff.). Im Rahmen von Refactorings sollen darüber hinaus typische Fehler wie Nutzung „verbotener“ Funktionen oder schwacher kryptografischer Verfahren in altem Code behoben werden. Außerdem wird u.a. der Einsatz von Pair Programming empfohlen.

2.2.3. OWASP

OWASP ist ein Community-Projekt mit dem Ziel, Erkenntnisse über die Entwicklung sicherer Webanwendungen aufzubereiten und sie Interessierten zugänglich zu machen. Bekannt ist OWASP vor allem für die breit rezipierte Top-10-Liste der kritischsten Sicherheitsrisiken für Webanwendungen. OWASP Foundation (2017) Auf der Projekthomepage finden sich Verlinkungen zahlreicher Unterprojekte, beispielsweise das ZAP, ein Tool für automatische Penetrationstests. Ebenfalls dort zu finden sind diverse Guides mit unterschiedlichen Schwerpunktsetzungen.

Development Guide

Dieses Dokument ist schon wegen des Namens wahrscheinlich oft die erste Anlaufstelle bei der Suche nach Anleitung zur Entwicklung sicherer(er) Webanwendungen. Tatsächlich werden hier auch wertvolle entwicklungsbegleitende Erkenntnisse gebündelt. Inhaltlich wird jedoch wenig Gewicht auf den Entwicklungsprozess gelegt. Aufgrund der Prominenz des Guides gebe ich dennoch einen kurzen Überblick über seinen Inhalt.

Der Development Guide (Eigenschreibweise: DevGuide) gliedert sich in Grundlagen, Design, Umsetzung und operative Sicherheit. Die Abschnitte sind individuell durchaus unterschiedlich aufgebaut. Im Grundlagenabschnitt finden sich bereits größtenteils eingeführte Definitionen sowie Standards & Richtlinien, Lebenszyklen, Training und, extrem knapp gefasst, Risikomanagement. Unter „Design“ finden sich mit „Grundlagen sicherer Designs“ Erklärungen zu Designkonzepten wie Least Privilege und Defense in Depth. „Sicherheitsarchitektur“ wirft einen Blick auf den größeren Begriff der Softwarearchitektur mit Ausrichtung auf deren sicherheitsrelevanten Aspekte. „Umsetzung“ ist der umfangreichste Abschnitt und behandelt unter anderem Authentifizierung, Session Management, Zugriffskontrolle und Kryptografie. Die meisten Einträge in diesem Kapitel beginnen mit Hintergründen zum Thema und liefern dann sog. „Principles“, also „Gesetzmäßigkeiten“. Gemeint sind Anforderungen, die eine Software hinsichtlich eines technischen Themas erfüllen muss. Bei Authentifizierung sind dies

z.B. Erzwingen derselben von Subjekten und das Schützen der übertragenen und persistierten Identifikationsdaten. Es folgt eine Liste von Modellen zur Lösung des jeweiligen Problems, wie beispielsweise Single und Multi Factor- sowie biometrische Authentifizierung. Jedes dieser Modelle wird anhand von Beispielen kurz erklärt und die Risiken werden aufgezählt. Wenn Standards existieren, die ein passendes Verfahren beschreiben, nennt der Guide diese. Zur Veranschaulichung des Prozesses der User-Registrierung bietet der Guide ein UML-Ablaufdiagramm. (OWASP Foundation, 2018a, S. 29)

Der Risikomanagementteil ist weniger als eine halbe Seite lang (OWASP Foundation, 2018a, S. 16) und enthält wenig mehr als eine Begriffsdefinition und die vier Möglichkeiten, mit einem (Rest-) Risiko umzugehen: Akzeptanz als das bewusste Eingehen eines Risikos; Linderung/Entschärfung als Installation technischer Gegenmaßnahmen; Transfer als ein anderes Wort für Versicherung (allgemeiner gesprochen: Auslagerung des Risikos); und zu guter Letzt Eliminierung, was bedeutet, die Quelle des Risikos gänzlich auszuschalten (ergo z.B. ein Feature nicht zu implementieren).

SAMM

Das Software Assurance Maturity Model ist eine Metrik zur Bestimmung des sicherheitsbezogenen Reifegrades eines Entwicklungsprozesses. Es besteht aus vier Teilen, den sog. Business Functions: Governance („Verwaltung“), Construction („Umsetzung“), Verifikation und Operations („Betrieb“). Jedem dieser Teile sind jeweils drei einzelne Tätigkeitsbereiche gewidmet, die Security Practices heißen. Zur Wahrung der Konsistenz werde ich deren Titel bei der Erläuterung übersetzen, wenn dies angemessen ist. Den Inhalt werde ich erläutern, wenn sich dieser nicht aus dem Begriff selbst ergibt.

Die Inhalte von Governance sind präskriptiver Natur (Chandra, 2017, S. 3). Hier findet sich mit „Weiterbildung und Anleitung“ ein Aspekt, der so ähnlich auch im SDL zu finden ist. Die Unterteilung von SAMM impliziert bereits eine Ausrichtung an unterschiedlichen Zielgruppen, und dieser Teil zielt auf das Projektmanagement. Damit ist er für eine genauere Untersuchung an dieser Stelle uninteressant - die administrative Hoheit über einen Großteil der Fragen dieses Bereichs liegt außerhalb der Reichweite Scrum-Teams.

Construction besteht aus Bedrohungsabschätzung, Sicherheitsanforderungen und sicherer Architektur. Dies sind die Haupttätigkeiten, auf die ich mich in dieser Arbeit konzentriere. Teil der Verifikation sind Design Review, Implementations-Review und Security Testing. Der letzte Teil, Operations, behandelt Fehlerfallbehandlung, Härtung der Infrastruktur „Operational Enablement“, was sich kaum unfallfrei ins Deutsche übersetzen lässt. Gemeint sind Tätigkeiten, die die mit dem Betrieb betreuten Mitarbeiter*innen in die Lage versetzen sollen, ihrer Tätigkeit

sinnvoll nachzugehen, etwa Zusammenstellen und Aufbereiten von Leistungsdaten und Betrieb von Software zur Unterstützung.

In den einzelnen Tätigkeitsbereichen kann eine Organisationsstruktur Punkte erzielen, die den Reifegrad des Tätigkeitsbereichs darstellen. Diese kulminieren zu einem von drei erreichbaren Reifegradleveln: In der Bedrohungsabschätzung beispielsweise ist die niedrigschwelligste Handlung, wahrscheinliche Bedrohungen überhaupt zu bedenken, und darauf aufbauend, sie zu dokumentieren. Zum Erreichen von Level 1 in der Bedrohungsabschätzung müssen diese beiden Handlungen durchgeführt werden.

Wikis

Ertragreicher als der DevGuide hinsichtlich der Schwerpunkte dieser Arbeit sind die verschiedenen auf der Projektwebseite zu findenden, jeweils unter einem Oberthema gruppierten Wikis. Auf der „Projects“-Kategorie der Hauptseite sind alle aufgelistet (owasp.org, 2018, Abschnitt „Projects“). Insbesondere der Testing Guide und das Code Review Project sind hier zu nennen. Viele derselben lassen sich als PDFs herunterladen, die jedoch den Gewinn an Übersicht mit verringertem Detailgrad erkaufen. Bei Bezugnahme auf diese Unterprojekte beziehe ich mich daher auf die Onlineversionen in den Wikis. Ansonsten ist das zur Verfügung stehende Material zwar umfangreich und allgemeinverständlich aufbereitet, jedoch nicht immer nachvollziehbar strukturiert. So dauert es z.B. nicht lange, bis man fast identische Inhalte an zwei verschiedenen Stellen findet.

Angesichts des Umfangs der vorangegangenen Schilderungen wäre der Gewinn an zusätzlichen Informationen begrenzt, wenn an dieser Stelle in größerer Ausführlichkeit auf eine Vielzahl von Empfehlungen der verschiedenen Guides eingegangen würde. Ich belasse es hier daher bei dieser knappen Übersicht. Eine detailliertere Auseinandersetzung mit einzelnen Punkten findet sich in 3.1.3.

2.3. Übersicht über Risikomanagementverfahren

Am unteren Ende der Skala möglicher Abstraktionsgrade befindet sich das technische Vorgehen am konkreten Einzelfall. Dies ist für die Schwerpunktsetzung dieser Arbeit zu kleinteilig und erfüllt nur selten den Anspruch auf Verallgemeinerbarkeit. Auch besteht eine andere Gefahr: Die übermäßige Konzentration auf eine konkrete technische Schwachstelle birgt das Risiko, aus dem Blick zu verlieren, dass diese ggf. für den jeweiligen Anwendungsfall irrelevant ist (sei es etwa eine unzureichende Routine zur Verschlüsselung einer Bibliotheksfunktion, die aber gar nicht genutzt wird), oder verhindert die Beschäftigung mit viel dringenderen Problemen

der Anwendung. Am anderen Extrem stehen Managementprozesse, der allgemeine Aufbau von Organisationen und letztlich rechtliche Fragen. Diese fallen jedoch in die Kompetenzen anderer Wissenschaften als der Informatik.

Daher richtet sich mein Blick speziell auf Techniken zur Verbesserung des Entwicklungsprozesses von Software. Besonders wichtig werden diese Verfahren in einem agilen Projekt dadurch, dass die mit der Entwicklung beauftragten Teams häufig nur Rechenschaft über ihre Ergebnisse ablegen müssen und weitestgehend frei sind in der Wahl ihrer Vorgehensweisen. Wie eine Software konkret entwickelt wird, ist somit eine Teamentscheidung. Um die Sicherheit von Softwareprodukten effektiv und effizient zu verbessern, muss daher der Prozess optimiert werden, in dem Software entwickelt wird. Aus wirtschaftlicher Sicht müssen dazu Prozessschritte identifiziert werden, deren positiver Einfluss auf die IT-Sicherheit im Verhältnis zu ihrem Aufwand der größtmögliche ist. Hier ist das Risikomanagement besonders hervorzuheben: Sein Ziel im Kontext der Softwareentwicklung ist es, die begrenzten Entwicklungsressourcen mit maximalem Nutzen für die Sicherheit einzusetzen. Weniger abstrakt heißt das, Entwickler*innen ihre Zeit und Aufmerksamkeit den jeweils dringlichsten Problemen widmen zu lassen.

Als Konsequenz dieser Erkenntnis stelle ich im folgenden einige zentrale Verfahren des Risikomanagements heraus, um im weiteren Verlauf der Arbeit Bezug auf sie nehmen zu können.

2.3.1. STRIDE

STRIDE ist eine Entwicklung von Microsoft und dient zur Identifikation und Klassifizierung von Bedrohungen einer Anwendung. Es kann benutzt werden, um Antwort auf die Frage zu erhalten, zu welchen Problemen es bei Betrieb der Anwendung kommen könnte. Zu den möglichen Reaktionen auf eine Bedrohung gehört, die Probleme technologisch zu lösen, bevor sie auftreten, ihre Auswirkungen mildern oder ggf. ein Feature gänzlich umzustoßen (Howard und Lipner, 2006, Abs. 24.368).

Bei STRIDE denkt man aus der Perspektive einer angreifenden anstatt wie bei CIA verteidigenden Partei. Auch der Name dieses Verfahrens ist eine Merkhilfe für die folgenden Bedrohungen.

Spoofing Identity Vortäuschung einer anderen Identität: Ein Angreifer gibt sich als legitime Kommunikationsgegenstelle aus, z.B. als Anwendungsserver oder ein eingeloggter User.

Tampering Modifikation von Daten und/oder Code in Schadabsicht

Repudiation Abstreiten, eine Aktion ausgeführt zu haben

Information Disclosure Offenlegung ansonsten geschützter Daten gegenüber unbefugten Dritten

Denial of Service Direkte oder indirekte Angriffe auf die Verfügbarkeit eines Dienstes

Elevation of Privilege Erlangung von umfassenderen Rechten als vorgesehen

Sobald die Komponenten der Anwendung sowie ihre Daten-Ein- und Ausgänge einmal herausgearbeitet sind, können die erkannten Entitäten den Bedrohungskategorien in einer Matrix gegenübergestellt werden.

DFD Element Type	S	T	R	I	D	E
External Entity	X		X			
Data Flow		X		X	X	
Data Store		X	†	X	X	
Process	X	X	X	X	X	X

Abbildung 2.1.: Bedrohungen & Elemente von Datenflussdiagrammen

Das Kreuz in der Grafik (Data Store/Repudiation) kennzeichnet hierbei einen Dienst, der nur in Spezialfällen von Attacken der entsprechenden Kategorie betroffen ist.

Am Ende des Vorgangs kann diese Tabelle nach Bedrohungen umgestellt werden, so dass jeder Bedrohungskategorie eine Liste betroffener Komponenten zugeordnet werden kann. Diese Liste bietet einen Startpunkt, um in einem nachgeordneten Arbeitsschritt das Risiko einzelner Bedrohungen einzuschätzen und so später Gegenmaßnahmen zu priorisieren.

2.3.2. DREAD

Das den Risikomanagementverfahren zuzuordnende DREAD wurde ebenso wie STRIDE ursprünglich von Microsoft eingesetzt. Das Akronym steht für Schadenspotential (Damage Potential), Reproduzierbarkeit, Ausnutzbarkeit (Exploitability), betroffene Nutzer (Affected Users)

und Entdeckbarkeit (Discoverability) (Microsoft (2007)). Für eine identifizierte Bedrohung werden alle der genannten Aspekte numerisch bewertet und die erzielten Punkte zu einer Summe zusammen gerechnet, welche die Bewertung darstellt. Bei Microsoft wird DREAD jedoch spätestens seit 2008 nicht mehr verwendet, da die Ergebnisse offenbar schwer zu verallgemeinern und stark von der persönlichen Einschätzung abhängig sind, die das Verfahren durchführen (Microsoft (2008)).

2.3.3. Risk Levels

Generell rät der SDL zur Vorsicht mit numerisch bewertenden Verfahren: Das Risiko eines Angriffs ist, wenn es mit Eintrittswahrscheinlichkeit · Auswirkungen beschrieben wird, stark von der Abschätzung determiniert, wann ein erfolgreicher Angriff stattfinden wird. Dies ist nach Howard und Lipner nicht seriös zu leisten, da niemand die Zukunft vorhersagen könne. Eine solche Risikobewertung gilt ihnen daher gewissermaßen als Ratespiel. Sie empfehlen daher eine an die Security Bulletin Rankings des Microsoft Security Response Center angelehnten Bewertungsskala, den Risiko-Levels von 1 bis 4 (je niedriger die Zahl, desto höher das Risiko). In diese Akkumulation fließt u.a. ein, ob eine Schwachstelle auf Server- oder Clientseite ausgenutzt wird, dies nur lokal oder auch aus der Ferne geschehen kann und ob in dies standardmäßig möglich ist oder nicht.

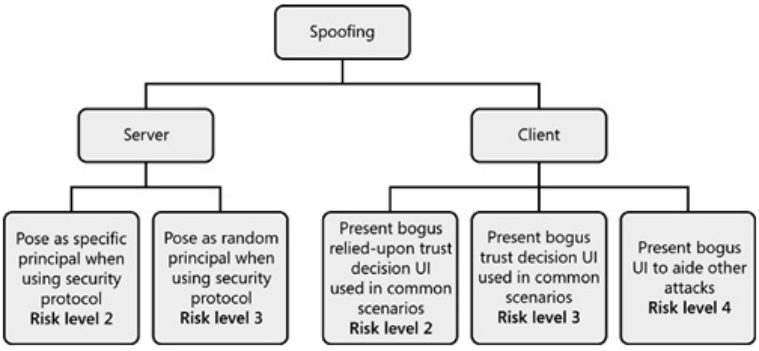


Abbildung 2.2.: SDL: Risiko-Level am Beispiel von Spoofing
(Howard und Lipner, 2006, Abs. 24.358)

3. Erarbeitung der Vorgehensweise

Im vorangegangenen Kapitel habe ich wesentliche Konzepte moderner Methodiken der IT-Sicherheit vorgestellt. Das folgende Kapitel steht nun im Zeichen des Security Engineerings, also der methodischen Erarbeitung eines Entwicklungsprozesses zur Konstruktion sicherer Systeme (Eckert, 2014, Abs. 11.2).

Von besonderer Bedeutung dafür sind für meine Schwerpunktsetzung wie erwähnt Techniken des Risikomanagements. Bevor Entwickler*innen sich mit Sicherheitsproblemen beschäftigen können, müssen diese naturgemäß bekannt und beschrieben sein. Dies leistet ein Risikomanagement. Da hundertprozentige Sicherheit in der Praxis außerdem nie zu erreichen ist, müssen die endlichen verfügbaren Entwicklungsressourcen dort zum Einsatz gebracht werden, wo sie maximale Wirkung erzielen - genau diese Stellen lassen sich mit Risikomanagement aufspüren. Auf diese Weise kann vermieden werden, sich in der Bearbeitung von Schwachstellen zu verlieren, die für den Anwendungsfall gar nicht relevant sind, womit Zeit und Geld gespart werden können. Zwischenergebnis eines guten Risikomanagementprozesses auf dem Weg zur Minimierung des Restrisikos ist nämlich das Wissen, was wann getan werden muss und was nicht. Es bildet die Grundlage, sich der IT-Sicherheit methodisch zu nähern. Das Wissen, wogegen man sich absichern muss und wogegen nicht, schlägt eine Brücke zum Agilen Manifest. Dort heißt es:

> „Einfachheit - die Kunst, die Menge nicht getaner Arbeit zu maximieren - ist essenziell." (Beck u. a.)

Risikomanagement und agile Softwareentwicklung schließen sich also nicht aus, sondern sind in Teilen ihrer Zielsetzungen sogar identisch.

3.1. Zusammenstellung der Prozessschritte

Es existieren zahlreiche Best Practices z.B. zum sicheren Programmieren. Das Security Engineering jedoch ist bis zum heutigen Zeitpunkt nicht hinreichend formalisiert. (Eckert, 2014, Abs. 11.2) Auf den kommenden Seiten werde ich mich daher unter Rückgriff auf die im vorherigen Kapitel zusammengetragenen Informationen diesem Vorgang methodisch nähern.

3.1.1. Security Engineering nach BSI-Grundschutz

Die Grundschutzkataloge benennen mit der sog. Sicherheitskonzeption das Zusammenstellen geeigneter Sicherheitsmaßnahmen und ihr Zusammenführen in einem Sicherheitskonzept (BSI, 2016, S.107). Dieser Schritt baut auf den Ergebnissen der Modellierung des Informationsverbundes auf. Als solcher wird die Gesamtheit der als System zusammenwirkenden Infrastruktur, Organisation, Personen und Technik bezeichnet. Noch vor der Modellierung erfolgt die sog. Strukturanalyse, während der die diesbezüglichen Informationen zusammengetragen werden. Es folgt die Schutzbedarfsfeststellung, bei der alle Anwendungsteile auf mögliche Beeinträchtigungen hinsichtlich primärer Schutzziele untersucht werden. Mithilfe einer Abschätzung möglicher Folgeschäden soll hier eine Einteilung in die Schutzbedarfskategorien normal, hoch und sehr hoch erfolgen, was dem Vorgang einer Risikoanalyse entspricht. Diese Schritte gelten universell sowohl für geplante als auch bereits in Betrieb befindliche Informationsverbände.

Das BSI legt nahe, komplexere Systeme in die bereits in 2.2.1 dargestellten Schichten Übergreifende Aspekte, Infrastruktur, IT-Systeme, Netze und Anwendungen zu unterteilen.

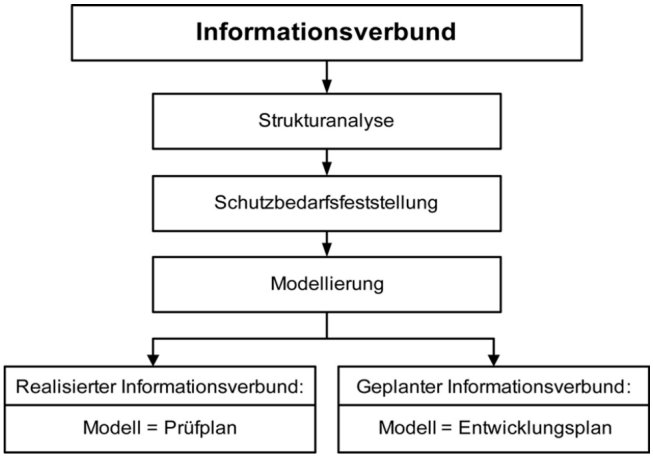


Abbildung 3.1.: Erstellung eines Sicherheitskonzepts nach BSI-Grundschutzkatalogen (BSI, 2016, S. 83)

Schicht 1: Übergreifende Aspekte

Das Ergebnis dieses Vorgangs leistet eine Abbildung zwischen den Bausteinen der Bausteinkataloge und den sicherheitskritischen Teilen des Systems oder Informationsverbundes. Hierfür können die Bausteine auch mehrfach verwendet werden. Spezialfälle stellen die folgenden da, die wegen ihrer allgemeinverbindlichen Natur nur einmal für die gesamte Betrachtung angewandt werden sollen:

- B 1.0: Sicherheitsmanagement
- B 1.4: Datensicherungskonzept
- B 1.6: Schutz vor Schadprogrammen
- B 1.13: Sensibilisierung und Schulung zur Informationssicherheit
- B 1.15: Löschung und Vernichten von Dateien
- B 1.16: Anforderungsmanagement
- B 1.18: Identitäts- und Berechtigungsmanagement

Einige Bausteine *müssen* dabei *immer mindestens einmal* herangezogen werden und dürfen somit für keinen Informationsverbund fehlen. Bei hinreichend autonomen Unterorganisationen sind sie für jede derselben anzuwenden. Dies betrifft:

- B 1.1: Organisation
- B 1.2: Personal
- B 1.9: Hard- und Softwaremanagement
- B 1.10: Standardsoftware

Bei Outsourcing gelten besondere, in weiterführenden Dokumenten festgehaltene Regeln. Bei größeren Verbünden sollen außerdem grundsätzlich folgende Bausteine angewandt werden:

- B 1.3: Notfallmanagement (auch speziell für Komponenten mit hohem Verfügbarkeits-Schutzbedarf)
- B 1.14: Patch- und Änderungsmanagement

Für besonders beachtenswert halte ich mit Bezug auf Webanwendungen darüber hinaus folgende Bausteine der Schicht 1:
B 1.5: „Datenschutz“ bezieht sich speziell auf die deutsche Rechtslage und soll Orientierung bieten, falls bei der Schutzbedarfsfeststellung Komponenten identifiziert wurden, die personenbezogene Daten verarbeiten. Auch ist zu prüfen, ob der Baustein einmal für den gesamten Verbund oder einzeln auf dessen Teile anzuwenden ist.
B 1.8: „Behandlung von Sicherheitsvorfällen“ kommt zur Anwendung bei Identifikation von Komponenten mit hohem oder sehr hohem Schutzbedarf bezüglich eines der drei zentralen

Schutzgüter Vertraulichkeit, Integrität und Verfügbarkeit, oder wenn dem Gesamtausfall des Systems ein erhöhtes Schadenspotential beigemessen wird.
B 1.17: „Cloud-Nutzung“ muss bei geplanter oder bereits aktiver Inanspruchnahme von Cloud Services bedacht werden, und zwar dergestalt, dass jeder einzelne genutzte Cloud Service sowie deren Schnittstellen untereinander einzeln modelliert wird.
(BSI, 2016, S. 84 ff.)

Schicht 5: Sicherheit in Anwendungen

Zu dieser Schicht der Bausteinkataloge merkt das BSI an, dass heutzutage Anwendungen praktisch nicht mehr allein betrachtet werden können, da gerade größere Systeme häufig aus diversen mehr oder weniger autonomen Komponenten bestehen - beispielsweise Webserver und Datenbanksystem. Diese müssen separat voneinander modelliert werden.

Für die Mehrheit der Bausteine dieser Schicht gilt: Ist eine entsprechende Anwendung im Informationsverbund, ist der jeweilige Baustein anzuwenden. Für jeden Verbund sollen laut BSI die folgenden zu Rate gezogen werden:

- Mobile Datenträger
- Protokollierung (zumindest bei größeren Systemen)

Die verbreitetsten und damit relevantesten übrigen Bausteine seien hier der Übersicht halber aufgezählt.

- Allgemeiner Client (bezeichnet verschiedene Client-Server- und Peer-to-Peer-Topologien)
- Webserver
- Datenbanken
- Telearbeit
- MS Exchange/Outlook
- SAP System
- Internetdienste (betrifft auch Nutzung durch Mitarbeiter*innen am Arbeitsplatz)
- Webanwendungen

(BSI, 2016, S. 90 ff.)
Am Ende empfiehlt das BSI, die Modellierung anhand einer aussagekräftigen Übersicht des Informationsverbundes wie etwa einem Netzplan auf Vollständigkeit zu prüfen: Jede Komponente muss dabei einer Gruppe (von vereinheitlichten anderen Komponenten)zugehören oder explizit modelliert sein.

Zusammenfassung

Für eine Vorgehensweise nach IT-Grundschutz ist die Modellierung des Systems bzw. seiner Komponenten anhand der Bausteinkataloge entscheidend. Sinnvollerweise wird hierzu das System bzw. der Informationsverbund in seiner Gesamtheit einmal zu Beginn modelliert und, agilen Methoden folgend, iterative Änderungen an bestehenden Komponenten im Verlauf der Sprints nachgetragen sowie der Vorgang für neue Bestandteile erneut vorgenommen.

Sind die Komponenten des Verbunds auf diese Weise modelliert, ob vor Projektbeginn oder während der Sprintplanung, lässt sich mit der Schutzbedarfsfeststellung gewichten, an welchen Stellen Angriffe besonders schwere Auswirkungen hätten. Anhand dieser Priorisierung können die relevanten Bedrohungen bzw. Gefährdungen identifiziert werden, indem aus der Liste der Bausteine abgelesen wird, welche Einträge der Gefährdungskataloge mit ihnen assoziiert sind. Auf gleiche Weise gelangt man zu geeigneten Maßnahmen, die als Stories formuliert werden und Eingang ins Backlog finden können.

3.1.2. Agiles Vorgehen nach Secure Software Development Lifecycle

Howard und Lipner betrachten zwar, wie bereits erläutert, nicht primär iterative, zyklische Vorgehensmodelle, sondern einen linearen Prozess.

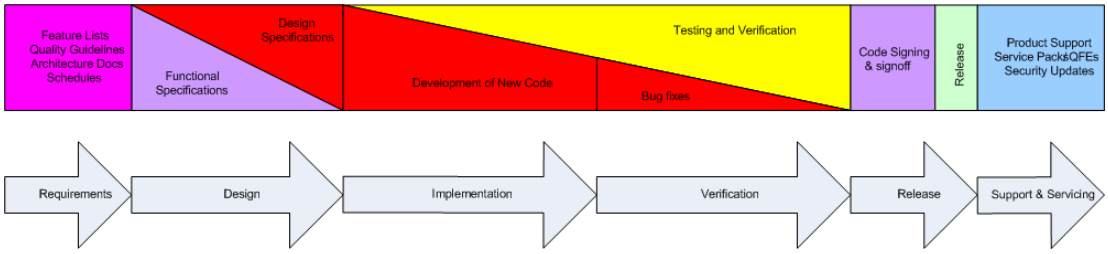


Abbildung 3.2.: Der traditionelle Entwicklungsprozess bei Microsoft
Howard und Lipner (2005)

Dies ist der Prozess, mit dem Microsoft bis etwa zum Jahr 2002 Software entwickelte (Howard und Lipner, 2006, Abs. 4.5 ff.). Agile Vorgehensmethoden und insbesondere Scrum betrachten eine Iteration bzw. einen Sprint dabei als eine Art Mini-Projekt gleicher Länge, in dem genau wie in einem regulären geplant, umgesetzt und abgeliefert werden soll (Schwaber und Sutherland, 2017, S. 9). Die letzte Phase, Support & Servicing, muss bei einem agilen Projekt jedoch anders bedacht werden: Wegen der iterativen Herangehensweise entsteht hier bereits frühzeitig eine

lauffähige Version des Produkts, die ggf. auch schon live betrieben wird und so einen zur Weiterentwicklung parallelen Supportbetrieb nötig macht.

Mit diesem Gedanken im Kopf sehen wir nun, wo Howard und Lipner welche Prozessschritte zur Verbesserung der Sicherheit hinzufügen. Die Grafik stellt dabei keine Differenz zur obenstehenden dar, sondern enthält nur die neuen Schritte:

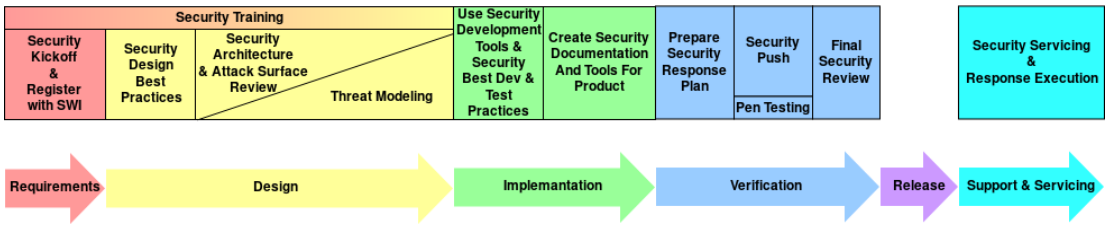


Abbildung 3.3.: Erweiterungen des Microsoftprozesses nach SDL, größere Darstellung: A.1 (Howard und Lipner, 2005, Nachbildung der Originalgrafik zur Verbesserung der optischen Qualität)

Risk Modeling

> „Wenn man uns die Hände auf dem Rücken zusammengebunden hätte (was nicht der Fall ist) und wir so auf eine Maßnahme beschränkt wären, um die Anwendungssicherheit zu verbessern, würden wir jeden Tag der Woche Threat Modeling betreiben." (Howard und Lipner, 2006, frei übersetzt, Abs. 24.9)

Die primäre Empfehlung des SDL ist klar: Threat Modeling. Grundlage hierfür sollen realistische, produktbezogene Bedrohungsszenarien sein, wie z.B. ein gestohlenes Endgerät für eine mobile Anwendung. Hierzu zu berücksichtigen sind unterschiedliche Rollen potentieller Angreifer und deren Interessen. Ebenfalls zu bedenken sind externe Abhängigkeiten wie Frameworks und Webserver, sowie getroffene Annahmen u.a. über deren Sicherheit: Es muss beispielsweise überprüft werden, ob ein System, das kryptografische Schlüssel speichern soll, dies auch in einer angemessenen Weise leisten kann. Bestehen Einschränkungen zu einem oder mehreren der genannten Aspekte, müssen diese dokumentiert werden.

Wie bereits in 2.2.2 beschrieben, ist das bevorzugte Werkzeug von Howard und Lipner zum Threat Modeling das Datenflussdiagramm (kurz: DFD). Wichtigstes Element darin sind Trust- oder Privilege-Boundaries; gemeint sind Kommunikationsflüsse, in denen Daten zwischen Kontexten fließt, in denen ihnen unterschiedlich stark vertraut wird bzw. die verschieden hohe Privilegien genießen. Dargestellt werden diese durch gestrichelte Linien.

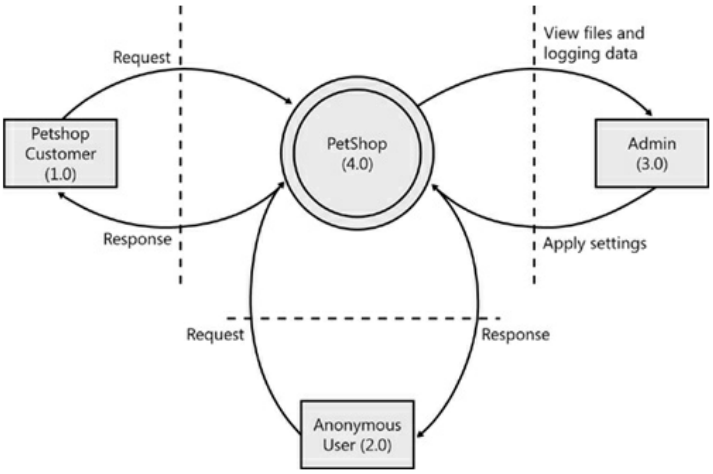


Abbildung 3.4.: Exemplarisches Datenflussdiagramm aus dem SDL, Trust/Privilege-Boundaries als gestrichelte Linie aufgetragen
(Howard und Lipner, 2006, Abs. 24.131)

An den durch Boundaries gekennzeichneten Grenzen müssen Anfragen wie auch die übergebenen Daten stets auf Korrektheit des Inhalts und der Berechtigung geprüft werden. Auch etwaige Fehlermeldungen dürfen nicht unbehandelt über diese Kanäle fließen. Die Prozesse direkt hinter den Boundaries stellen die primären Ziele für Injectionangriffe und Quellen für unberechtigte Datenabflüsse dar. Gemäß SDL muss aller Code für diese Schnittstellen von Menschen geprüft werden.

Um ein solches DFD zu erzeugen, schlagen die Autoren vor, von einem Kontextdiagramm ausgehend alle komplexen Prozesse zu zerlegen und die relevanten Kommunikationsflüsse zu modellieren, und diesen Vorgang zu wiederholen, bis keine hinreichend komplexen Prozesse mehr als Black Boxes existieren (Level-0- und Level-1-DFD). Gegebenenfalls sind in diesem Stadium Entitäten nachträglich einzupflegen, an deren Schaffung bislang vielleicht niemand gedacht hat. Zur abschließenden Vereinfachung können alle Entitäten mit identischen Profilen innerhalb einer Trust Boundary zusammengefasst werden.

In einem agilen Projekt sind Architekturen stärker in Bewegung als z.B. bei Anwendung des Wasserfallmodells. Das Gegenteil einer einmal aufgestellten, starren Architekturskizze ist jedoch nicht das vollständige Fehlen einer solchen, sondern eine iterative Anpassung derselben. Das steht keineswegs im Konflikt zur obenstehenden Modellierung; Wie die Autoren selbst anführen, sind kleinere Pakete von Änderungen ohnehin vorteilhaft bezüglich einer Vielzahl von Aspekten der Sicherheit, so auch des Designs (Howard und Lipner, 2006, Abs. 24.59).

Steht dieses Modell, können mit dem bereits bei 2.3.1 beschriebenen STRIDE Bedrohungen identifiziert und diese anschließend bewertet werden. Ergänzend hierzu bietet der SDL sog. „Bug Bars“: Dies sind in tabellarische Aufstellungen, welche Klassen von Fehlern wie dringend zu beheben sind, aufgeteilt in Server- und Clientseite. Für Server beispielsweise beinhaltet die höchste Dringlichkeitsstufe sämtliche Fehler, die zu Elevation of Privilege oder Denial of Service führen können.

Zusammenfassung

Zentrale Bedeutung hat bei Howard und Lipner das Threat Modeling. Dies geschieht auf Basis der Modellierung sicherheitskritischer Kommunikationsflüsse in einem Datenflussdiagramm. Kriterium hierfür ist das Überschreiten von Trust Boundaries, also Grenzen zwischen verschiedenen Kontexten der Anwendung. An diesen Stellen wird mit STRIDE eruiert, mit welchen Bedrohungen zu rechnen ist und diese werden z.B. mit Risiko-Levels bewertet.

Im agilen Prozess kann dies umgesetzt werden, indem während der Planungsphase ggf. sogar noch vor dem ersten eigentlichen Sprint eine „Baseline“ modelliert und das Modell bei der Umsetzung einzelner Komponenten punktuell verfeinert wird. Auf diese Weise kann der obenstehende Prozess zu Beginn einer jeden Implementation wiederholt und so eine graduell immer genauer werdende Modellierung erreicht werden, um zu einer stets aktuellen und angemessen priorisierten Liste von neuralgischen Punkten der Anwendung und den sich daraus ergebenden Notwendigkeiten zu gelangen. Aus diesen kann das Team leicht technische Anforderungen an das System ableiten und diese etwa in Form von User Stories methodisch in den Entwicklungsprozess integrieren.

3.1.3. OWASP

Ein Blick auf den Artikel zum Risk Rating Methodology im Testing Guide (OWASP Foundation, 2018b, Risk Rating Methodology) zeigt eine Vorgehensweise, die sich gut in die bereits in den zwei vorangegangenen Abschnitten zu den Grundschutzkatalogen und Microsofts SDL gesehenen einfügt:

Im ersten Schritt müssen wie immer Risiken identifiziert werden. Notwendig hierfür sind Informationen darüber, welche Auswirkungen ein Exploit, also das Ausnutzen einer Schwachstelle, hätte, über die Verwundbarkeiten selbst, verwendete Angriffe und Threat Agents. Letzteres unterscheidet in der OWASP-Terminologie die Bedrohungen gemäß ihrer Natur z.B. in Achtlosigkeit und Wettbewerber.

Darauffolgend werden in Schritt zwei die Faktoren bestimmt, die das Eintreten der Bedrohung in seiner Wahrscheinlichkeit bestimmen. Hierzu zählen u.a. die Fähigkeiten, die zur Ausnutzung

einer Schwachstelle nötig sind, Motive und notwendige Gelegenheiten. Die einzelnen Faktoren können numerisch oder mit niedrig, mittel und hoch bewertet werden.

Im Schritt drei werden die sog. „Impact Factors“ beurteilt, also die potentiellen Auswirkungen eines Exploits. Diese zerfallen dabei in technische sowie „Business Impact Factors“. Die ersteren bestehen aus dem Verlust jeweils eines der CIA-Schutzgüter sowie der Zurechenbarkeit und sind vor allem interessant durch ihre Konsequenzen für die zweite Kategorie. Hier steht der unmittelbare Schaden für die Organisation im Mittelpunkt, die das in Frage stehende Produkt einsetzt. Die Faktoren bestehen aus finanziellem Schaden, Rufschädigung, „Non-Compliance“ (Verletzung von Richtlinien, z.B. Verträgen) und Preisgabe persönlicher Daten.

Schritt vier besteht wie in der klassischen Risikokalkulation aus dem Zusammenführen der vorangegangenen Schätzungen zu Eintrittswahrscheinlichkeit und Auswirkungen eines Risikos. Dies kann in kleineren Projekten „informell“ per individueller Einschätzung geschehen, oder „wiederholbar“: Hierzu werden die in den vorangegangen Schritten beschriebenen Werte errechnet und in einer Matrix zueinander in Beziehung gesetzt.

Overall Risk Severity				
Impact	HIGH	Medium	High	Critical
	MEDIUM	Low	Medium	High
	LOW	Note	Low	Medium
		LOW	MEDIUM	HIGH
	Likelihood			

Abbildung 3.5.: OWASP Risikobewertung: Beziehung zwischen Eintrittswahrscheinlichkeit und Auswirkungen

In Schritt fünf müssen basierend auf dieser Risikobewertung die Entscheidungen zum einen dafür fallen, was überhaupt behoben werden soll, und zum anderen, wie diese zu priorisieren sind. So sollen höher bewertete Risiken selbst dann bevorzugt behandelt werden, wenn niedrigere schneller, billiger und einfacher zu beheben wären. Außerdem soll der Aufwand ins Verhältnis zum Nutzen gesetzt werden: Eine Auswirkung von maximal 2000$ jährlich unter Aufwendung von 100.000$ zu verhindern, amortisiert sich erst nach 50 Jahren Betriebsdauer.

Abschließend können in Schritt sechs noch weitere Impact Factors hinzugefügt werden (z.B. Gefährdung oder gar Verlust von Menschenleben bei Betrieb kritischer Infrastrukturen), die Klassifikationen der Auswirkungen für die Impact Factors den real vorzufindenden Gegebenheiten angeglichen oder deren Gewichtung dem Geschäftsmodell angepasst werden.

Zusammenfassung

Insgesamt setzt der vom OWASP formulierte Prozess an etwas höherer Abstraktionsstelle an als in der restlichen Literatur. So benennt er keine dezidierte Methodik, um das betreffende System

zu modellieren, fügt dem Prozess des Risikomanagements jedoch mit Threat Agents einen Faktor hinzu, der so sonst nirgends auftaucht. Diese einzigartige Komponente ermöglicht eine feiner granulierte Auflösung und Kategorisierung der Bedrohungen nach nicht-technischen Ursachen.

Der eigentliche Prozess der Risikobewertung ist den Vorgenannten nicht unähnlich und kann je nach Bedarf entweder numerisch erfolgen oder nach weniger formeller Klassifizierung in niedrig, mittel und hoch. Danach erfolgt noch eine Abwägung, ob gewisse Gegenmaßnahmen nicht ggf. sogar unterlassen werden, etwa weil der potentielle Schaden in keinem wirtschaftlichen Verhältnis zum Aufwand steht. Insgesamt kann die OWASP Risk Rating Methodology als eine aufgebohrte Version von Eintrittswahrscheinlichkeit · Auswirkungen gesehen werden, in der allerdings die Schätzungen auf beiden Seiten des Operators durch feinere Auflösung und vorgegebene Skalen weniger subjektiv ausfallen soll.

3.1.4. Ergebnis

Ein verbindendes Element aller drei Werke betrifft die Handhabung von Risiken, denen die Anwendung ausgesetzt ist. An einer Stelle ist diesbezüglich die Rede von Risk/Threat Assessment, an anderer von Bedrohungsanalyse. Zur Vereinfachung benutze ich daher im Folgenden die Bezeichnung Risikomanagement.

Essenziell ist hierfür eine Modellierung kritischer Systemkomponenten. In Details weichen die Empfehlungen hierzu voneinander ab, z.B ist die Modellierung der Komponenten nach IT-Grundschutz deutlich statischer als die der Kommunikationsflüsse nach SDL, über die Sinnhaftigkeit einer Modellierung allgemein lassen die verschiedenen Autor*innen jedoch keinen Zweifel. Das OWASP selbst nennt keine explizite Modellierungstechnik, aufgrund der Natur der sonstigen Hinweise ist aber klar, dass für diese ein Modell vorausgesetzt wird. Der Nutzen eines solchen Modells ist dabei primär Komplexitätsreduktion: Menschen können nur wenige Informationen gleichzeitig im Gedächtnis halten, und Code sowie Infrastruktur sind schon bei weniger aufwändigen Anwendungen in ihrer Gesamtheit kaum mehr zu überblicken. Für ein methodisches Vorgehen ist ein Modell daher praktisch Pflicht.

Im Kern ebenfalls einig sind sich alle Autor*innen, dass auf dem Modell aufbauend Bedrohungen & Risiken zunächst identifiziert und in einem zweiten Schritt bewertet sollen. Hierbei wird Wichtiges von Unwichtigem getrennt und aus der Gewichtung ergibt sich die Reihenfolge des Umgangs mit den Bedrohungen. Jeweils eigene Vorstellungen existieren davon, wie genau diese Gewichtung vorzunehmen sei: Das BSI legt hierfür die Schutzbedarfsfeststellung zugrunde, der SDL mahnt Skepsis gegenüber numerischen Risikobewertungsverfahren an &

nennt als Alternative die vierstufigen Risiko-Level. Der Prozess des OWASP kann diesbezüglich numerisch oder informell ausformuliert werden.

Die Artefakte sowohl der Modellierung als auch des Risikomanagements lassen sich zusätzlich auch zur Dokumentation nutzen. Nicht nur eignen sich die Diagramme als Architekturdokumentation, die Gesamtheit der Ergebnisse des beschriebenen Prozesses macht die Entscheidungsfindung im Team transparent und begründet sie nachvollziehbar. Auf einer solchen Basis lässt sich gerade im Nachhinein sehr viel fundierter vertreten, warum eine technische Gegenmaßnahme in die Wege geleitet werden musste als durch ein erratisches Bauchgefühl einzelner Entwickler*innen.

3.2. Vorstellung des verwendeten Entwicklungsmodells

Im Folgenden Abschnitt lege ich in knapper Form dar, wie die Entwicklung der Anwendung bislang organisiert ist. Hierzu gehe ich auf das verwendete Entwicklungsmodell ein und beschreibe bisherige externe Einträge ins Team zum Thema der IT-Sicherheit. Spezifika erläutere ich an späterer Stelle bei der eingehenderen Untersuchung des Prozesses auf Verbesserungsmöglichkeiten.

3.2.1. Scrum

Vorgehen

Basis für das Vorgehen ist der bekannte Scrum-Ablauf: In einem täglichen Standup-Meeting von maximal einer Viertelstunde berichten alle Teammitglieder über am Vortag erreichte Fortschritte, ob es relevante Hindernisse gab, besprechen die Aufgabenverteilung für den Tag und führen die Fortschrittsmessung am Storyboard durch. Die hier im Fortschritt verfolgten Userstories werden zu Beginn der im Projekt zuerst zwei-, später dreiwöchigen Sprints aus den Stories im Project-Backlog in einem gemeinsam durchgeführten Sprint-Planning erzeugt. Hierbei findet eine Aufwandsschätzung unter Berücksichtigung der Velocity des Teams statt.

Die Stories für das Project-Backlog werden vom PO in Zusammenarbeit mit dem Kunden und ausgewählten Teammitgliedern mit großer Projekterfahrung erstellt.

Verwaltet werden die Stories in einem Jira-System mit mehreren Boards: Es existiert je eins für den aktuellen Sprint, das auch physisch mit Karteikarten auf Pinnwänden visualisiert wird, für gefundene Bugs und Support. Auf letzteres hat der Kunde Zugriff und kann dort eigene Beobachtungen an den ihm zugänglichen Test- und Produktivsystemen festhalten, was nach einer langsamen Anlaufphase auch zunehmend genutzt wird.

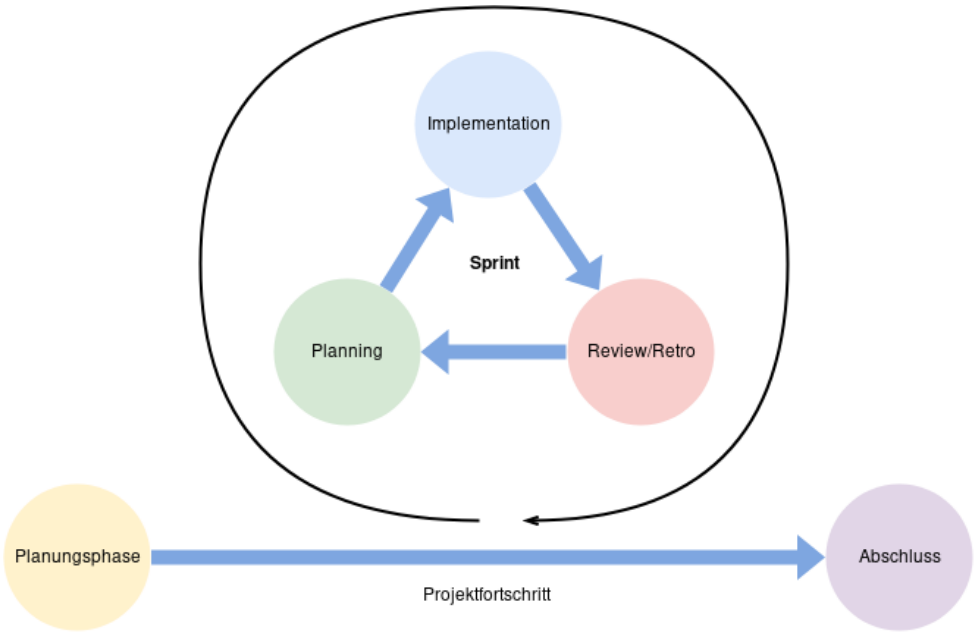


Abbildung 3.6.: Scrum, vereinfacht dargestellt

Team

Die Teamgröße ist von anfangs vier auf heute zehn Entwickler*innen gestiegen. Es existieren die üblichen Scrum-Rollen des Project Owners (kurz „PO“) und Scrum-Masters. Die letztgenannte Position wird in Teilzeit von einem Entwickler ausgefüllt. Die Teammitglieder verfügen über verschiedene Hintergründe und individuell unterschiedliches Vorwissen, darüber hinaus es gibt im Team jedoch keine Spezialisierungen.

3.2.2. Security-Gruppe

In unregelmäßigen Abständen finden die Treffen einer projektübergreifenden Arbeitsgruppe zum Thema Security statt. Hier werden z.B. von einem Vortragenden (Angriffs-) Techniken vorgeführt und an dem aktuellen Stand der entwickelten Software ausprobiert. Mit OWASP-ZAPP wurden sogar bereits Probleme im BLOB-Speicher der Anwendung gefunden: Die URLs sowohl zum Abrufen einer Referenz auf Binärobjekte als auch aller UUIDs waren versehentlich nicht gegen unbefugten Zugriff gesichert worden. Ein Angreifer hätte somit in der Theorie Daten aus dem System extrahieren können. Als Ursache stellte sich das irrtümliche Fehlen einer Annotation an einem Webservice heraus.

```
1  public class BlobWebService {
2
3      @GET
4      // @Secured // war zum Testen des Services auskommentiert
5      @Path("/blob/{id}")
6      public Response getBlobRef(@PathParam("id") @NotNull UUID id) {
7      StatusFileDTO statusFile = documentService.getStatusFileWS(id);
8          return Response.ok(statusFile).build();
9      }
10 }
```

Listing 3.1: Versehentlich ungesicherte Methode des BLOB-Webservices

Die Ergebnisse der Treffen werden ins Team getragen, vor allem durch den PO, der festes Mitglied der Arbeitsgruppe ist.

3.3. Vorstellung der Anwendung

In diesem Abschnitt gebe ich einen Überblick über die Funktionalität der entwickelten Software und die Fachlichkeit und gehe auf einige sicherheitsrelevante Details ein.

3.3.1. Fachlichkeit & Anforderungen

Der Kunde vermietet und verkauft weltweit Tankcontainer. Mit der zu entwickelnden Software möchte er seine bisherige Lösung zur Verwaltung von Stammdaten wie Informationen über Mitarbeiter, Geschäftskontakte und Container, sowie zur Erstellung von Angeboten und Rechnungen ersetzen. Hierfür möchte er verschiedenen Mitarbeiter*innen verschiedene Rechte gewähren. Dies alles soll von verschieden Standorten und Endgeräten aus geschehen können. Des Weiteren sollen die Altdaten aus dem bestehenden System in das neue migriert werden.

3.3.2. Implementation

Die Geschäftslogik wird umgesetzt in einem Java-Backend unter Verwendung des Hibernate-Frameworks für die Persistenz, des Maven-Buildsystems zur Kompilierung und Management von Abhängigkeiten sowie des WildFly-Application Servers.

Da die Anwendung im Browser und von mehreren Standorten bedient werden soll, muss das Frontend eine HTML-Seite sein, die zumindest theoretisch in die ganze Welt ausgeliefert

wird. Hierfür kommt das bereits erwähnte Angular-Framework zum Einsatz, als Webserver dient nginx.

Datenmodell

Entitäten, die in der Datenbank gespeichert werden, benötigen als identifizierenden Schlüssel häufig IDs. Eine naive Implementation eines solchen Mechanismus inkrementiert hierfür schlicht Integerzahlen. Dies bringt Nachteile mit sich, u.a. sind solche IDs leicht aufzuzählen und damit zu raten. Ein Angreifer kann dieses Wissen etwa zur Konstruktion von URLs ausnutzen.

Dieser Umstand wurde im Projekt schon früh erkannt, woraufhin die Umstellung aller IDs auf UUIDs, Universally Unique Identifier, erfolgte. Vereinfacht ausgedrückt bieten diese in RFC 4122 standardisierten Bezeichner einen Namensraum von solcher Größe, dass Kollisionen sogar über Systemgrenzen hinweg auf Dauer hinreichend unwahrscheinlich sind. So kann eine UUID nicht sinnvoll geraten werden.

Binärdatenspeicher

Die verwendete Datenbank Maria-DB ist als relationales System nur begrenzt geeignet, Daten in binärer Form zu speichern. Der Kunde möchte jedoch binär kodierte Dokumente, etwa PDFs, in der Anwendung speichern, beispielsweise ein mit einem Vertrag assoziiertes Rechnungsdokument. Zu diesem Zweck speichert die Uploadfunktion der Anwendung die Daten nicht selbst, sondern reicht sie an ein Storage-System des jeweiligen Cloud-Anbieters (siehe 3.3.3) weiter und legt in der eigenen Datenbank nur eine Referenz auf das BLOB an.

Nutzerauthentisierung

Wie in praktisch jeder Webanwendung erzeugt eine Nutzeraktion mit Datenzugriff vom Frontend einen HTTP-Request gegen eine URL des Backends. Weil HTTP bekanntermaßen keine Zustände kennt, muss das Backend bei jeder solchen Anfrage sicherstellen, dass das Subjekt seine Identität nicht nur vortäuscht.

Zu diesem Zweck speichert die Datenbank nicht nur das (gehashte) Passwort und ein Salt, sondern es kommen JWT (JSON Web Token) zum Einsatz. Beim Login generiert das Backend ein solches Token und schickt es in seiner Antwort, transportverschlüsselt, dem Frontend. Im Rest der Session weist sich der Client durch Mitsenden des Tokens aus. Es kann bei Bedarf erneuert werden, wird beim Logout gelöscht und läuft nach einer gewissen Zeit von selbst ab,

so dass nicht jede unabsichtlich unbeendete Session dem nächsten Nutzer des Endgeräts in die Hände fällt.

Rechte und Rollen

Verschiedene Objekte der Anwendung sollen nicht jedem Subjekt zur Verfügung stehen. Die schiere Zahl der Objekte und wiederkehrende Muster in der Objekt-Subjekt-Relation legen die Verwendung von RBAC, Role Based Access Control, nahe. Und so gehören zu einem Nutzer-Objekt der Anwendung stets eine oder mehrere Rollen, denen jeweils eine Zahl Rechte zugeordnet wird. Die Administrationsrolle beispielsweise darf Nutzeraccounts, die Sales-Rolle Angebote anlegen und verwalten.

Im Frontend dient dieses Modell dazu, bereits die Navigation zu Seiten, die für die jeweilige Nutzerrolle nicht zugänglich sein sollen, zu unterbinden. Umgesetzt wird dies im Projekt über die Angular-Technologie des Route-Guards: Dieser besteht aus einer Funktion, die zu wahr oder falsch ausgewertet wird und in Abhängigkeit davon entweder die angeforderte oder eine Fehlerseite anzeigt (Angular Documentation, 2018, Fundamentals: Routing & Navigation: Milestone 5, Route Guards).

Input Sanitizing

Das Frontend der Anwendung ist öffentlich zugänglich und nimmt bereits zum Login notwendigerweise Daten entgegen. Zur Abwehr diverser Formen von Injection-Angriffen müssen diese Inputs „sanitized“ werden, als von Zeichen bereinigt, die andernfalls in die Anwendung gelangen und dort als Code ausgeführt werden würden. Im konkreten Fall stellt das Frontend die erste diesbezügliche Verteidigungslinie dar: Angulars Standardverhalten ist es bereits, Usereingaben ohne weiteres Zutun von Entwicklerseite von gefährlichen Zeichen zu säubern (Angular Documentation, 2018, API: Core: Sanitizer).

3.3.3. Deployment & Infrastruktur

Versioniert wird die Software in Git, es kommt dabei das Git-Flow-Modell zum Einsatz. Die Builds der Software finden auf einem Jenkins-Build-Server im Haus statt und erfolgen automatisch nach jedem Commit auf den Development-Branch. Hierbei werden stets die Tests ausgeführt und Fehlschläge den commitenden Entwickler*innen per Email mitgeteilt. Nach erfolgreichem Build kann das Deployment in eine von vier Stages (Test, Preview, QA & Produktiv) erfolgen: Hierfür erzeugt der Build-Server ein Docker-Image mit dem Kompilat der Anwendung und pusht es in die Registry von Kubernetes-Instanzen, die bei unterschiedli-

chen Anbietern gehostet werden können (Amazon, Microsoft und Google). Die Pods, also die virtuellen Maschinen in den Clustern, sind konfiguriert, neue Images per Rolling-Update auszurollen, können aber natürlich auch von Hand neu gestartet werden. Als Webserver kommt nginx in der zu Projektbeginn aktuellen Version 1.13.5 zum Einsatz. Die Zertifikate für HTTPS-Verbindungen werden darüber hinaus per Let's Encrypt generiert und automatisch erneuert.

Seit Frühjahr 2018 befindet sich die Anwendung in einem für den Kunden in Grundzügen nutzbaren Stadium. Seitdem kann dieser mit der als Produktivsystem deployten Instanz zu arbeiten und auf dem Testsystem neue Funktionen auszuprobieren; hier existiert ein separater Datenbestand und nach außen reichende Funktionen sind „entschärft", versendete Email z.B. landen bei einer Auffangadresse anstelle der wirklichen Adressaten. Der finale Livegang einer ersten vollumfänglichen Version ist noch einige Sprints entfernt.

3.4. Untersuchung des Reifegrades der Entwicklung mit OWASP SAMM

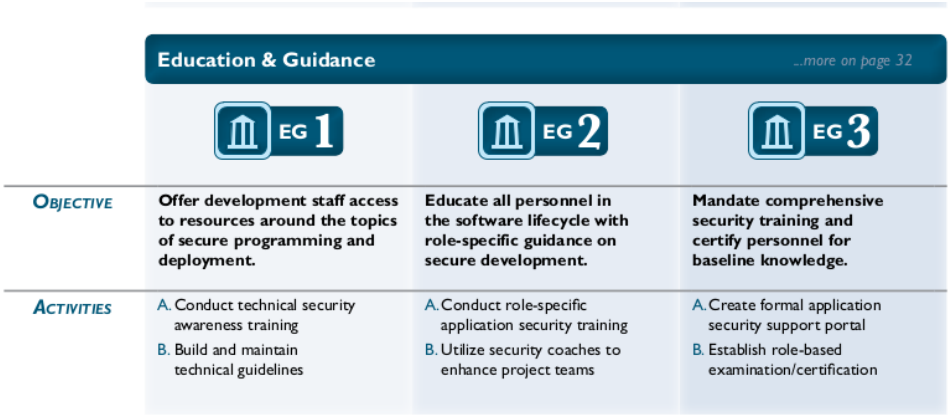
Von sich aus macht das Team nach allgemein akzeptierten Kriterien bereits das allermeiste richtig im Sinne allgemeiner Produkt- und Prozessqualität. Mit zehn Personen ist die Zahl der Entwickler*innen zwar streng genommen etwas zu groß für Scrum (Schwaber und Sutherland, 2017, S. 7), im derzeitigen Entwicklungsstand können die einsetzenden Reibungsverluste jedoch abgefangen werden, indem sich einzelne Entwickler*innen auf parallel zur Kernentwicklung durchführbare Tätigkeiten wie Entwicklung von Integrationstests konzentrieren.

Dass außer dem unter 3.2.2 beschriebenen Vorfall bislang keine wesentlichen Sicherheitsprobleme zutage getreten sind, trägt dazu bei, dass das kollektive Bewusstsein für die IT-Sicherheit im Team begrenzt ist. Wie in dieser Arbeit bereits gezeigt wurde, ist dies jedoch nicht in jedem Fall als Indiz dafür zu werten, dass es keine Sicherheitsprobleme gibt bzw. geben wird. Es bedarf daher einer objektiveren Metrik der Prozessqualität als nur eines individuellen Bauchgefühls. Von den bisher vorgestellten Werkzeugen drängt sich für diesen Anlass das OWASP SAMM geradezu auf.

Das Software Assurance Maturity Model wurde in 2.2.3 vorgestellt und soll nun zur Anwendung kommen. Hierbei werde ich mich im Interesse der Lesbarkeit & Kürze auf die Bestimmung des jeweiligen Levels beschränken und auf die Berechnung eines genauen Scores verzichten, sowie nur exemplarisch anhand der ersten relevanten Aktivität die Arbeitsweise dokumentieren. Für die verbleibenden Aktivitäten liefere ich den Level am Ende dieses Abschnitts in einer Übersicht.

3.4.1. Exemplarische Untersuchung der Business Function Verwaltung („Governance“)

Wie bereits in 2.2.3 ausgeführt, ist hier die Security Practice „Weiterbildung und Anleitung“ von besonderem Interesse. Hier ein beschreibender Auszug aus dem Originaldokument:



Education & Guidance			
	EG 1	EG 2	EG 3
Objective	**Offer development staff access to resources around the topics of secure programming and deployment.**	**Educate all personnel in the software lifecycle with role-specific guidance on secure development.**	**Mandate comprehensive security training and certify personnel for baseline knowledge.**
Activities	A. Conduct technical security awareness training B. Build and maintain technical guidelines	A. Conduct role-specific application security training B. Utilize security coaches to enhance project teams	A. Create formal application security support portal B. Establish role-based examination/certification

Abbildung 3.7.: Beispiel für SAMM-Activities: Governance „Ausbildung und Anleitung“ (Chandra, 2017, S.)

Zur Veranschaulichung ermittle ich diesen Score exemplarisch ausführlich.

1 A In den Treffen der Security-Arbeitsgruppe werden unterschiedliche Themen adressiert. Das Untersuchen der Anwendung auf Schwachstellen mit OWASP ZAPP kann als technisches Bewusstseinstraining gezählt werden, wenngleich die Treffen unregelmäßig und unter freiwilliger Teilnahme stattfinden.

1 B Es existieren durchaus technische Richtlinien im Projekt, etwa der Code- und Import-Formatter. Ebenso sind wesentliche technische Schritte dokumentiert wie das korrekte Aufsetzen der Entwicklungsumgebung. Andererseits sind viele Festlegungen im Projekt informeller Natur: Unter welchen Bedingungen z.B. Code auf den develop-Branch gemerged werden darf, ist nur rudimentär niedergeschrieben und gar nicht verschriftlicht ist die Konventionen, wann statt einem Merge ein Rebase stattfinden soll (gleichwohl gibt es sie). Zu Datenmodellen und Architektur gibt es ebenfalls nur in Einzelfällen Diagramme, und diese sind oft veraltet.

2 A Definierte Rollen gibt es im Projekt, abgesehen von den zur Parallelisierung abgegrenzten Testentwickler*innen, nicht, weswegen auch weder auf sie zugeschnittene Trainings noch Prüfungen und Zertifizierungen stattfinden können.

2 B Security Coaches und ein Security Support Portal gibt es in dieser Form nicht.

3 A & B Da die Voraussetzungen für Level 2 nicht erfüllt wurden, beachte ich Level 3 nicht weiter.

Insgesamt ergibt sich damit bei nicht allzu strenger Auslegung Level 1.

3.4.2. Ergebnis

Um die Auflösung des Ergebnisses zu verbessern, stelle ich in der Tabelle auch halbe Level dar, wenn von zwei nötigen Punkten nur einer umgesetzt wird. Diesen Ansatz verfolgt das SAMM in ähnlicher Weise selbst zum Errechnen des genauen Scores (Chandra, 2017, S. 9).

		Level	Durchschnitt
Verwaltung	Weiterbildung und Anleitung	1	1
Umsetzung	Bedrohungsabschätzung	0	0
	Sicherheitsanforderungen	0	
	sichere Architektur	0.5	
Verifikation	Design Review	0	0.5
	Implementations-Review	0.5	
	Security Testing	0.5	
Betrieb	Fehlerfallbehandlung	1	1

Diese Bewertung sieht auf den ersten Blick ernüchternd aus. Die Arbeitsweise des Teams bislang jedoch nicht auf IT-Sicherheit allgemein oder die SAMM-Metrik speziell optimiert worden. Daran gemessen ist das Erreichen von zwei mal Level 1 kein schlechtes Ergebnis.

Es lassen sich anhand der Tabelle jedoch mehrere bislang offenbar wenig bearbeitete Themenkomplexe identifizieren. Über Level 0 nicht hinaus kommt das Projekt in den Bereichen Bedrohungsabschätzung, Sicherheitsanforderungen (die sich direkt aus den vorgenannten ergeben), und Design Reviews. Zu letzteren ist zu erwähnen, dass im Projekt zwar durchaus Designs in Review-artigen Vorgängen diskutiert werden, dabei aber de facto nie über Sicherheitsaspekte gesprochen wird, so dass hierfür auch kein halber Punkt vergeben werden kann.

3.5. Folgerungen aus der Analyse

Aus der Bewertung ist zu schließen, dass sich Prozessverbesserungen besonders im Feld des Risikomanagements anbieten: Das Kriterium der Sicherheitsanforderungen zum Beispiel hängt explizit vom Vorhandensein der Bedrohungseinschätzung ab, und im weitesten Sinn damit auch alle in der Tabelle aufgeführten Aktivitäten, der Level < 1 war.

Die Ergebnisse der Untersuchung mit dem SAMM deuten damit in die selbe Richtung wie die im Vorfeld geäußerten Gedankengänge hinsichtlich der effektivsten und effizientesten Optionen zur Verbesserung des Entwicklungsprozesses. Das bisherige Ausbleiben von Sicherheitsvorfällen kann nicht als Eigenschaft des Prozesses angesehen, sondern muss der Erfahrung und den individuellen Fähigkeiten der Entwickler*innen im Team zugeschrieben werden. Das Vorhandensein dieser Qualität ist selbstredend zu begrüßen, kann aber aus verschiedenen Gründen nicht zur Grundlage verlässlicher, sicherer Softwareentwicklung erhoben werden. Wenn die Qualität des Produkts allein von den Qualifikationen einzelner Mitarbeiter abhängt, führt dies unweigerlich zu Problemen, wenn:

- sich die Zusammensetzung eines Entwicklungsteams entscheidend ändert
- in anderen Projekten gänzlich andere Personen miteinander arbeiten
- „Spezialisten“ durch Urlaub, Krankheit oder Team-/Arbeitgeberwechsel ausfallen

Aus diesem Grund muss der Entwicklungsprozess seinen Teil beitragen, die Qualität sicherzustellen. Dies gilt selbstverständlich genauso für IT-Sicherheit wie für jeden anderen Aspekt der Prozess- und Produktqualität.

4. Umsetzung

Im vorangegangenen Kapitel wurden der zu analysierende Entwicklungsprozess sowie die damit entwickelte Software beschrieben und ermittelt, wie und an welcher Stelle Änderungen stattfinden sollen. Auf den folgenden Seiten beschreibe ich nun die Durchführung der angedachten Änderungen.

4.1. Schutzgüter

Noch bevor man mit dem eigentlichen Risikomanagement beginnt, ist es sinnvoll sich zu vergegenwärtigen, welche Werte tatsächlich geschützt werden sollen. Letzten Endes dient alle eingesetzte Technik stets einem Business-Interesse und ist demzufolge kein Selbstzweck. Aus der diversen Natur möglicher Anforderungen an IT-Systeme und Anwendungen (vgl. 2.1.3) können sich unterschiedliche Schutzbedürftigkeiten ergeben.

4.1.1. Werte der Anwendung

Der Kunde möchte mit dem entwickelten System selbstverständlich Geld verdienen, was nur möglich ist, wenn er damit wirtschaftlich arbeiten und Rechnungen erzeugen kann. Zu diesem Zweck benötigt er zuallererst natürlich seine Datenbestände. Diese beinhalten Informationen über die Benutzer*innen des Systems, Kunden & und Dienstleister*innen, Container und diverse weitere fachliche Entitäten, sowie Angebote, Verträge und Rechnungen.

Aufgrund der Komplexität der Rechnungserstellung ist deren maschinelle Durchführung ebenfalls ein Kernaspekt der Software und damit ein zentraler Wert für den Kunden: Die Zahl der in einem einzelnen Vertrag involvierten Container erreicht in der Praxis bisweilen dreistellige Bereiche, von denen jeder einzelne zu verschiedenen Konditionen sowie für verschiedene Zeiträume gebucht werden kann und an dieser Stelle auch noch auf Plausibilität und Durchführbarkeit geprüft wird.-Außerdem können die Verträge versioniert werden, wobei beim Abrechnen der aktuellen Version keine bereits abgerechneten Vermietungen älterer Versionen erneut in Rechnung gestellt werden sollen. Dies ist ohne Hilfe der Maschine kaum zu leisten, was explizit einen der Gründe für den Wunsch des Kunden nach einer solchen Software

darstellt. Dieser maschinellen Hilfe zuzurechnen ist nicht nur die Businesslogik im Backend, die die eigentliche Abrechnung durchführt, sondern auch das Frontend, welches die User z.B. bei der Dateneingabe und dem Zusammenstellen der Verträge unterstützt. Das eigentliche Rechnungsdokument muss ebenfalls erzeugt und dem User zugänglich gemacht bzw. versandt werden können.

Der Wert der gesamten Anwendung steht und fällt mit der Bereitstellung dieser Kombination von Funktionen.

4.1.2. Schutzziele

Die primären CIA-Schutzziele Vertraulichkeit, Integrität und Verfügbarkeit sind hier leicht auszumachen. Insbesondere die Vertraulichkeit findet sich auf allen Ebenen: Der Zugriff auf die Daten in der Datenbank darf nur bestimmten Subjekten gestattet werden. Gemäß des in 3.3.2 dargestellten RBAC-Konzepts zu Rollen und ihren Rechten erhalten Subjekte darüber hinaus nur Zugriff auf eine Teilmenge der Objekte. Doch nicht nur die persistierten Daten müssen im ruhenden Zustand gegen Verletzungen der Vertraulichkeit gesichert werden, sondern auch die transienten, also in Übertragung befindlichen.

Die Integrität der Daten ist analog zur Vertraulichkeit einzuhalten sowohl für ruhende als auch transiente Zustände: Weder in der Datenbank, noch während der Übertragung darf ihre nichtintentionale Veränderung unbemerkt bleiben oder zugelassen werden. Erneut ist hier das Rechte-Rollen-Konzept zu berücksichtigen. Da außerdem besonders unter Einsatz agiler Entwicklungsmodelle oft teil- oder vollautomatisierte Deployments stattfinden sollen, muss ein Weg gefunden werden, einen maschinellen Prozess sicher zu einem autorisierten Subjekt zu erheben, das die in Ausführung befindlichen Kompilate ersetzen und somit Einfluss auf das Verhalten der Anwendung nehmen darf. Die Forderung nach Integrität muss also nicht nur für die Daten erfüllt sein, sondern für auch die Anwendung selbst.

Bei einer Webanwendung ist darüber hinaus offensichtlich, wo mit der Anforderung der Verfügbarkeit anzusetzen ist: Daten und Logik liegen bzw. werden ausgeführt auf einem zentralen Server(-cluster), von wo auch das Frontend ausgeliefert wird. Fällt diese Instanz aus, kann die Anwendung nicht benutzt werden.

Von den erweiterten Schutzzielen ist die Nichtabstreitbarkeit herauszuheben: Änderungen an Entitäten und Geschäftsdaten sollen auf expliziten Kundenwunsch nachverfolgbar sein. Wie erwähnt, steht dies im Widerspruch zum Schutzziel der Anonymität, die dementsprechend in diesem Fall keinen Wert der Anwendung darstellt.

4.2. Konkrete Änderungen im agilen Prozess

4.2.1. Risikomanagement

Zentrales Ergebnis aller bisherigen Herleitungen ist: Der erste und wichtigste Schritt, um in Softwareprojekten reproduzierbare Ergebnisse bezüglich der IT-Sicherheit zu erzielen, muss die Installation einer Form des Risikomanagements sein. Dabei darf nicht nur die Wirksamkeit der Maßnahmen Einfluss auf deren Gestaltung nehmen, es muss auch deren Akzeptanz im Team bedacht werden: Ihre Ausführung findet später in zeitlicher Konkurrenz zur übrigen Planung sowie der eigentlichen Entwicklungstätigkeit statt. Wenn die Prozessschritte den Entwickler*innen nicht einleuchten oder als unnötig wahrgenommen werden, ist die Wahrscheinlichkeit groß, dass hierfür kein „Commitment“ stattfinden wird, das Team also diese Verantwortung nicht annimmt. Besonderes Gewicht bekommt dieser Umstand angesichts der Tatsache, dass nicht wenige agil arbeitende Teams in zum Teil erheblichem Maße autonom arbeiten. Die wahrscheinlich meisten Entwickler*innen wohl neigen dazu, lieber zu implementieren als zu planen, und Risikomanagement fällt zweifelsohne in die weniger beliebte zweite Kategorie. Gleichwohl darf der Akzeptanz nicht die Effektivität geopfert werden. Die Sinnhaftigkeit dieses Vorgangs ist in den vorherigen Kapiteln hinreichend dargelegt worden.

Wenn also u.a. Bedrohungsmodellierung stattfinden soll, hat diese in einem Umfang zu erfolgen, der dem vorgesagten Rechnung trägt.

Modell

Wie bereits hergeleitet wurde, ist die Grundlage fürs Risikomanagement ein Modell, das die Komplexität des Untersuchungsgegenstandes beherrschbar macht. Für einen mit Scrum vergleichbaren Prozess formuliert heißt dies, dass zunächst eine Basis-Modellierung stattfinden muss, die im Verlauf der Entwicklung und Explizierung einzelner Komponenten anzupassen und zu verfeinern ist. Im vorliegenden Fall liegt der Projektbeginn schon einige Zeit zurück, so dass bei der Modellierung die real implementierte Anwendung als Grundlage verwendet werden kann anstelle abstrakter Architekturskizzen.

Zunächst muss jedoch über die Art der Modellierung entschieden werden. Die unterschiedlichen vorgestellten Herangehensweisen weisen Eigenschaften auf, die sich als Vor- und Nachteile ausprägen können. Hierfür ist zuerst die Frage zu stellen, welchen Umfang das erzeugte Modell haben soll. Ein minimales Modell stellt nur die Kommunikationsabläufe innerhalb der Anwendung dar, ein erweitertes bezieht auch wesentliche Infrastrukturkomponenten mit ein und die Modellierung nach IT-Grundschutz zielt schon gar nicht mehr auf Systeme allein, sondern modelliert unter der Bezeichnung des Informationsverbundes sogar die ein System

bedienenden Menschen. Relevant bezüglich einer Entscheidung für eine Modellierungstechnik sind mit Blick auf Scrum als Vorgehensmodell besonders die folgenden Fragen:

- Kosten-Nutzen: Ist der Mehraufwand eines detaillierteren bzw. umfassenderen Modells wirtschaftlich und/oder technisch sinnvoll?
- Akzeptanz: Das Baseline-Modell muss gepflegt und erweitert werden. Ist das im gewählten Detailgrad und Umfang während eines Sprints leistbar?
- Vorhandene Informationen: Können über jeden Modellierungsgegenstand verlässliche Aussagen getroffen werden?
- Mehrfachnutzen: Unterläuft eine Art des Modellierens vielleicht die Weiterverwendung des Modells für einen anderen Zweck (z.B. wie hilfreich ist ein Datenflussdiagramm wirklich für die Dimensionierung von Infrastruktur)?

Für das Vorgehen nach IT-Grundschutz spricht zunächst die Reputation BSI. Der Name ist in Deutschland zumindest nicht fachfremden Kreisen durchaus geläufig als eine Instanz im Feld der IT-Sicherheit und es böte sich daher aus Marketingperspektive an, zum Beispiel mit einer Aussage wie „IT-Sicherheit nach Grundschutz des BSI“ zu werben. Ein weiterer Vorteil sind die Bausteine der Kataloge, die sich explizit auf den Datenschutz beziehen. Der rechtliche Rahmen hierzu ist in Deutschland weitaus enger als in weiten Teilen der restlichen Welt, was deutsche Softwareentwickler*innen vor spezielle Herausforderungen stellt. Mit einer Modellierung, die dem Rechnung trägt, ließe sich die Zahl der Unsicherheiten in diesem Bereich reduzieren. Hinzu kommt die Möglichkeit, seine Software vom BSI zertifizieren zu lassen.

Andererseits bedeutet eine Zertifizierung einen nicht unerheblichen organisatorischen wie auch finanziellen Aufwand, kann je nach Umfang längere Zeit dauern und bezieht sich notwendigerweise auf genau eine Version der Software. Schon ein einzelner Hotfix reicht aus, damit die eingesetzte Version des Produkts nicht mehr der zertifizierten entspricht. Dies ist kaum mit der Realität agiler Softwareentwicklung in Deckung zu bringen. Auch ist die Modellierung anhand der IT-Grundschutz-Bausteine nicht darauf ausgelegt, um bedarfsgerecht grob oder fein zu modellieren und das Modell entlang von Sprints weiter auszubauen.

Hierfür sind Datenflussdiagramme die bessere Wahl: Ihre iterative Qualität ermöglicht eine schrittweise Annäherung einer groben Baseline an detaillierte Komponenten. Dadurch kann im Sprint die Kosten-Nutzen-Abwägung am Einzelfall getroffen werden, was die Akzeptanz im Team steigern kann. Außerdem müssen hierfür, anders als bei der Modellierung eines ganzen Informationsverbunds, nicht zwangsläufig spekulative Annahmen etwa über

unbekannte Akteure im Personenkreis des Kunden getroffen werden. Die Diagramme lassen sich darüber hinaus mit überschaubarem Aufwand in andere Diagrammtypen überführen, etwa in ein Komponentendiagramm - der Mehrfachnutzen bleibt daher ebenso erhalten. Auch sind wesentliche Entscheidungen im konkreten Projekt bereits getroffen worden, bei denen ein anderes Modell eine Hilfestellung hätte sein können - wie etwa die erwähnte Dimensionierung der Infrastruktur. Dieser potentielle Nutzen anderer Modelle ist also vernachlässigbar.

In Summe präsentiert sich daher die Modellierung mit Datenflussdiagrammen zumindest im hier untersuchten Einzelfall als sinnvoller anpassbar an die Anforderungen agiler Softwareentwicklung.

Bedrohungsanalyse

Steht einmal das Modell, so können die Anordnung der Trust Boundaries als erste Anlaufstelle zur Gewichtung ihrer Relevanz dienen. Nach oder von „außen“ gerichtete bzw. kommende Kommunikationen, also solche mit einer Verbindung zu sog. externen Entitäten (z.B. User), stellen sozusagen die erste Verteidigungslinie dar, der naturgemäß die höchste Aufmerksamkeit gebührt: Wann immer die Anwendung unter Druck gerät, ist es erstrebenswert, unerwünschte Handlungen so früh wie möglich zu unterbinden oder gar ins Leere laufen zu lassen. Jeder nicht schon hier verhinderte Zugriff ermöglicht einem Angreifer mindestens ungewollte Rückschlüsse oder lässt gar eine Tür z.B. für Denial of Service offen. Am anderen Ende einer solchen Liste von Datenflüssen steht beispielsweise die Kommunikation zwischen Anwendung und Infrastruktur, etwa, was die JVM auf ihrem Docker-Container tut.

Entlang dieser priorisierten Liste kann nun eines der Werkzeuge zur Bedrohungsanalyse angewandt werden. Das prominenteste und gut bewährt ist zweifelsohne das schon beschriebene STRIDE: Für jeden eine Trust Boundary überquerenden Datenfluss wird dazu aus Sicht eines potentiellen Angreifers geprüft, ob hier einer der das Akronym konstruierenden Angriffsvektoren zur Verletzung der Schutzziele führen und Werte der Anwendung in Gefahr bringen kann. Triviales Beispiel: Ein ungeschützter Lesezugriff auf die Datenbank ermöglicht Information Disclosure, was mit in Konflikt mit dem Schutzziel der Vertraulichkeit steht.

Risikobewertung

Die anhand der minimalen Modellierung mit STRIDE ermittelten Bedrohungen lassen sich in Kombination mit der Aufstellung von Werten & Schutzzielen bereits für einen ersten Durchlauf der Risikobewertung einsetzen. Hierfür stehen wiederum mehrere mögliche Vorgehensweisen zur Verfügung. DREAD und Auswirkungen · Eintrittswahrscheinlichkeit bieten dafür zwar

eine erste Näherung; beide stehen jedoch im Ruf, dass sich ihre Ergebnisse nur schlecht reproduzieren lassen (Howard und Lipner, 2006, Abs. 24.346). Zur Auswahl stehen damit noch die Schutzbedarfsfeststellung des BSI, die Risiko Level nach SDL und die Risk Rating Methodology des OWASP (vgl. 3.1.3). Die erstgenannte ist im BSI-Prozess eng verknüpft mit der Baustein-Modellierung, gegen die jedoch bereits im vorangegangenen Abschnitt entschieden wurde. Dies engt die Auswahl auf die letzten beiden Kandidaten ein. Von diesen berücksichtigt nur die OWASP-Methode Eigenschaften potentieller Angreifer, die es an späterer Stelle jedoch ermöglichen würden, mit einem Blick Bedrohungen und Risiken auszusortieren, wenn beispielsweise eine Schwachstelle nur von einem gewissen Personenkreis ausgenutzt werden könnte.

Im Ergebnis gelangt man durch Anwendung der obenstehenden Schritte zu einer Art Liste von Negativ-Anforderungen an die Anwendung, welche sich in spezielle User Stories überführen lassen, wie ich nachfolgend zeigen werde.

Agile Umsetzung

Die exakte Umsetzung des Vorgesagten in einem agilen Vorgehensmodell wird von der Fachliteratur nirgends vorformuliert. Jedoch ist im Verlauf der Bestandsaufnahmen und Analysen in dieser Arbeit bereits erkennbar geworden, wie dies funktionieren kann. Das zu lösende Problem besteht darin zu bestimmen, an welcher zeitlichen Stelle das Risikomanagement in den Scrum-Prozess einzugliedern ist.

Abbildung 4.1.: Die zusätzlichen Schritte („OWASP RRM“: OWASP Risk Rating Methodology)

Hierzu schildere ich im Folgenden zwei mögliche Varianten. Bei beiden findet die Modellierung der Anwendungen in einem DFD statt. Dies geschieht iterativ und bedarfsabhängig - zu Beginn reicht eine grobe Vereinfachung in einem Level 0-Übersichtsdiagramm, ggf. ergänzend zu oder auf Basis von Architekturskizzen.

A priori Hierbei finden die Verfahrensschritte vor dem Sprint Planning statt: Wenn anhand des Product Backlogs klar wird, dass im nächsten Sprint eine Komponente zu implementieren sein wird, die nicht klar erkennbar ausschließlich innerhalb Trust Boundaries operiert, muss

das Basismodell um diese zu verfeinert bzw. muss sie gesondert modelliert werden. Daraufhin wird am Modell STRIDE in der beschriebenen Weise angewandt: Alle Datenflüsse über eine Trust Boundary hinweg werden untersucht. Ergebnis ist eine Liste von Bedrohungen, die jede für sich mit der Risk Rating Methodology des OWASP bewertet werden.

Das Vorhandensein dieser Einschätzung vor Beginn des Sprint-Plannings ist entscheidend, denn hier wird anhand der Risikobewertung über den Umgang mit den Risiken entschieden. Es kann z.B. beschlossen werden, dass zum jeweiligen Zeitpunkt das Risiko einzugehen ist, weil beispielsweise bereits alle wirtschaftlich und technisch sinnvollen eindämmenden Maßnahmen getroffen wurden, die Behebung kann auf einen späteren Sprint verschoben werden, oder aus dem Risiko können sog. Evil User Stories abgeleitet werden.

Dies ist ein Konzept des OWASP (owasp.org, 2018, Stichwort „Evil User Stories“) und beinhalten anstelle einer Anforderung eine aus Angreifersicht formulierte Gefährdung. Während also eine Story des Backlogs in Tasks aufgebrochen wird, können anhand der mit ihr assoziierten Risiken die Evil User Stories geschrieben und wie andere Implementationstätigkeiten als Task erfasst werden. Der von einer solchen Story repräsentierte Aufwand besteht darin, zu verhindern, dass ihr Inhalt eintreten kann. Ein Beispiel von der Webseite: „Als Hacker kann ich in einer Weise Daten in eine URL einbetten, so dass mir Daten oder Funktionalitäten zugänglich werden, zu denen ich normalerweise nicht berechtigt bin“ (frei übersetzt). Diese Story gilt dann als erledigt, wenn keine Injection-Angriffe mehr über URLs möglich sind. Da User Stories aus der Perspektive eines Stakeholders bzw. einer Benutzerrolle (Engl. „User Role“, (Cohn, 2004, S. 31)) geschrieben sind, kann hier der in der OWASP Risk Rating Methodology ermittelte Threat Agent verwendet werden.

Der Aufwand zur Absicherung einer Softwarekomponente gegen ein auf einer Evil User Story vermerktes Risiko wird dabei nicht separat auf dieser notiert, sondern ist der Zeitschätzung für die eigentliche Implementation hinzuzurechnen. Dies soll der Versuchung entgegenwirken, erst zu implementieren und dann abzusichern. Auf diese Weise wird gefördert, dass Entwickler*innen beides gemeinsam denken.

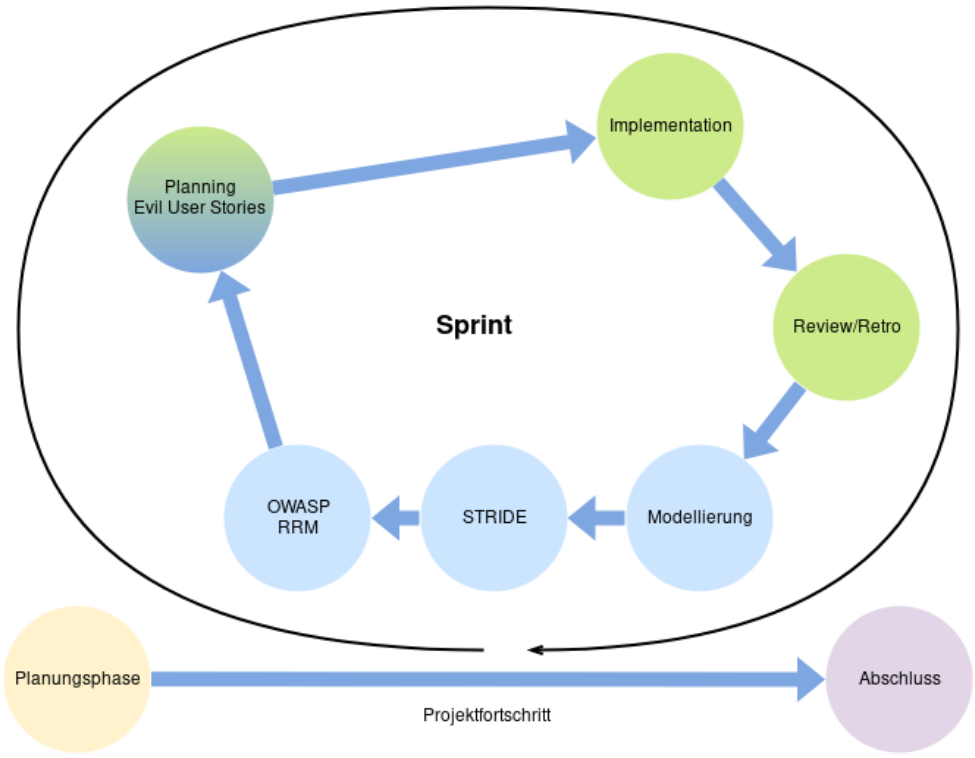


Abbildung 4.2.: Prozesserweiterung in der Variante A priori, Planning und das Schreiben der Evil User Stories wegen zeitlicher Einheit gemeinsam dargestellt

A posteriori Eigentlich sieht Scrum jedoch vor, dass erst im Sprint Planning die eigentlichen Design-Arbeiten zur Umsetzung einer Story stattfinden (Schwaber und Sutherland, 2017, S. 11). Das vorgelagerte Modellieren & Bewerten einer Architektur widerspricht diesem Gedanken und könnte schwer umzusetzen sein, wenn von dem im Scrum Guide beschriebenen Vorgehen nicht abzuweichen ist.

Eine andere Möglichkeit der Umsetzung macht sich die Erweiterbarkeit von Datenflussdiagrammen zunutze: Während des Plannings erstellt ein Teammitglied parallel zum Gespräch über eine Komponente ein basales DFD derselben. Sind daran kritische Kommunikationsflüsse ablesbar, werden die möglichen Bedrohungen im Anschluss an den Designvorgang vom gesamten Team mit STRIDE ermittelt und grob & informell mit Eintrittswahrscheinlichkeit · Auswirkungen gewichtet. Für jedes Risiko, das dem Team relevant erscheint, wird bestimmt, ob Gegenmaßnahmen zu treffen sind. Der Mehraufwand in der Entwicklung wird abgeschätzt,

wie bereits beschrieben zum Aufwand der Implementation hinzu addiert und eine Evil User Story geschrieben.

Das entstandene DFD kann im Bedarfsfall zu Sprintbeginn noch weiter ausformuliert und die Risikoeinschätzungen mit der OWASP-Methode noch präzisiert werden. Insbesondere bei Kernfunktionalitäten einer Anwendung kann dies geboten sein, oder wenn bereits mit der informellen Methode außergewöhnlich hohe Risikoeinschätzungen zustande gekommen sind.

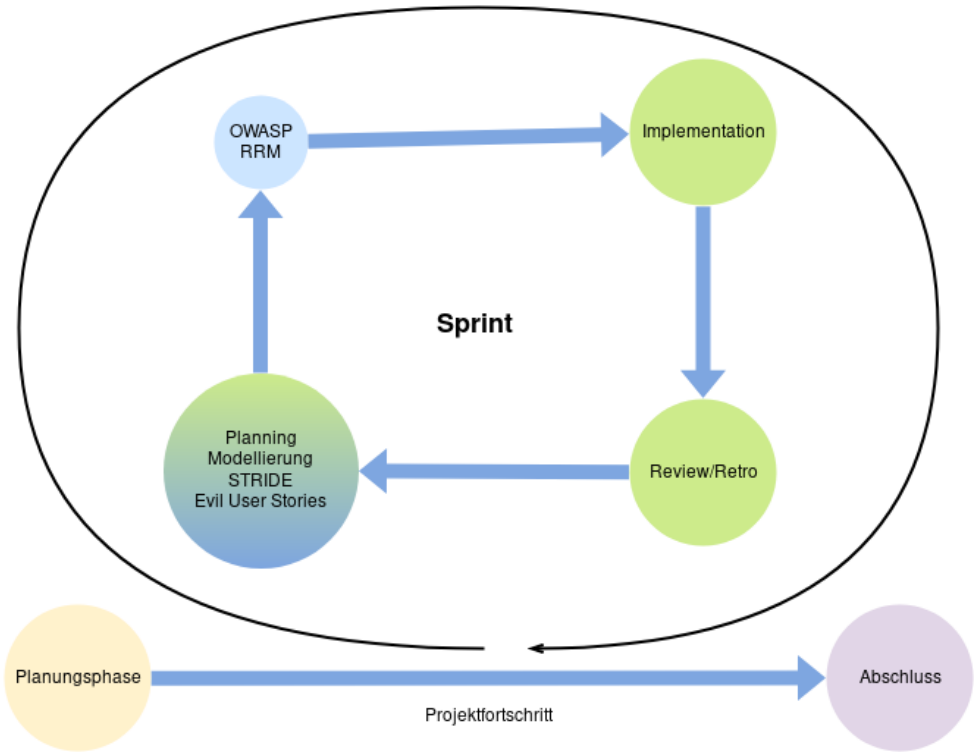


Abbildung 4.3.: Prozesserweiterung in der Variante A posteriori, außer Risikobewertung finden alle Schritte gemeinsam mit dem Planning statt

Unabhängig von der gewählten Variante ist es der Objektivität und Transparenz getroffener Entscheidungen dienlich, frühzeitig den Umgang mit Risiken festzulegen. Dies umfasst u.a., ab welcher Risikostufe unbedingt Gegenmaßnahmen zu treffen sind, welche Risiken eingegangen werden können, sowie den Umgang mit Restrisiken: Denn weder setzt das Installieren technischer Hürden für Angreifer die Risikostufe automatisch auf null bzw. niedrig (vgl. 4.4.2), noch kann eine solche Einschätzung als ewig gültig betrachtet werden. Zu schnell ändern sich technische Rahmenbedingungen, bisweilen wird Software eines Tages anders eingesetzt als

ursprünglich vorgesehen und getroffene Annahmen können sich als falsch oder nicht mehr aktuell herausstellen.

Daher sollte Teil der Projektdokumentation eine Art Risikoregister sein. Ein solches umfasst eine Auflistung identifizierter Risiken und nennt jeweils Gegenmaßnahmen. Auf diese Weise kann im Schadensfall schnell und adäquat reagiert werden, wodurch das SDL-Stadien 10: „Security Response Planning“ und 12: „Security Response Execution“ implementiert bzw. ermöglicht werden.

4.2.2. Weitere Verbesserungen

Auch außerhalb des Risikomanagements lässt sich ein Entwicklungsprozess. An einigen Stellen werden günstige Nebeneffekte durch ihre Kombination mit dem Obenstehenden deutlich.

Security Champions

Dieser mit der Organisation der Security-Arbeitsgruppe hochgradig kompatibler Vorschlag stammt ebenfalls aus dem OWASP: In einem Team soll ein sog. Security Champion benannt werden, der in einer Art Doppelfunktion einerseits aktiv entwickelt und gleichfalls teilhat am „AppSec“-Team - also der Security-Arbeitsgruppe. Dies bedeutet u.a. die regelmäßige Teilnahme an deren Meetings zum Zwecke der Weiterbildung, und im Team selbst die Übernahme der Hauptverantwortung zu Fragen der IT-Sicherheit. Damit soll sichergestellt werden, dass das Team stets produktiv bleiben kann, weil es stets einen Ansprechpartner zu Sicherheitsthemen hat (owasp.org, 2018, Stichwort „Security Champions“).

Konkret kann mit einem Security Champion ein*e Hauptverantwortliche*r für die Modellierung und Risikoanalyse bestimmt werden. Diese geschieht am besten nicht in Einzelarbeit, sondern mindestens zu zweit: Die Qualität eines Modells gewinnt in gleicher Weise durch ein zweites Paar Augen wie Code, und wie beim Pair Programming werden auf diese Weise die nötigen Fähigkeiten im Team gestreut.

Die Mitglieder eines nach Scrum arbeitenden Teams sollen jedoch besonders nicht von außen andere Rollen als Scrum Master, Project Owner oder Entwickler*in zugewiesen bekommen, um stärker crossfunktional arbeiten zu können (Schwaber und Sutherland, 2017, S. 7). Um diese Prinzipien nicht zu unterlaufen, muss die streng definierte Rolle aufgeweicht werden. Ein gut zusammengestelltes, funktionierendes Team organisiert sich im Idealfall selbstständig in einer Weise, dass am Thema interessierte Mitglieder die Funktionen der Security Champions auch ohne formale Rollenzuweisung zu übernehmen. Disfunktionale Teams müssen jedoch ohnehin erst die Probleme lösen, die sie disfunktional machen.

Schulungen

Insbesondere der SDL betont die Notwendigkeit, Entwickler*innen regelmäßig in Sachen IT-Sicherheit zu schulen. Wie bereits geschildert ist das Sicherheitsbewusstsein des Teams noch ausbaufähig, wofür Schulungen ebenfalls eine Abhilfe darstellen können.

Die hauseigene Security-Arbeitsgruppe stellt einen ersten Ansatz der Implementation dieser Verbesserung dar. Gemäß SDL sollen allerdings alle Entwickler*innen mindestens einmal im Jahr an einer solchen Schulung teilnehmen, was durch die Arbeitsgruppe bislang nicht in vollem Umfang erreicht wird. Zu diesem Zweck böten sich stattdessen bzw. ergänzend regelmäßige, verpflichtende Workshops zu diesem Thema an. Während einer diesbezüglichen Diskussion in dieser Arbeitsgruppe wurde das Vorhaben formuliert, solche Veranstaltungen teamübergreifend, aber mit Fokus auf die aktuell verwendeten Technologien von jeweils einem oder mehreren Mitarbeiter*innen des Hauses in wechselnder Besetzung durchführen zu lassen.

Als Gegenstand der Schulungen wurden Techniken zur Erstellung und Vermittlung des Nutzens von Datenflussdiagrammen, Risikoanalyse sowie Paradigmen und Best Practices vorgeschlagen.

Secure by Design & Default-Paradigma

Wie bereits in vorangegangenen Kapiteln beschrieben, formuliert der SDL von Microsoft zwei wesentliche Sicherheitsparadigmen für die Softwareentwicklung, nämlich Secure by Design (Mitdenken der IT-Sicherheit bereits in der Konzeptionsphase) und Secure by Default (Ausgehen vom Vorhandensein von Sicherheitslücken und entsprechend vorsichtige Standardeinstellungen).

Das Secure by Design-Paradigma lässt sich im Projekt an verschiedenen Stellen wiederfinden. So war die Verwendung wesentlicher Basistechniken zur Absicherung der Software schon vor Entwicklungsbeginn in der Planung vorgesehen, wie etwa das Rollout in ein Kubernetes-Cluster mit vorgelagertem Load-Balancer zur Sicherstellung der Verfügbarkeit oder der Einsatz von Let's Encrypt zum Erzeugen & erneuern gültiger Zertifikate für transportverschlüsselte HTTP-Verbindungen, jeweils ohne aktives menschliches Eingreifen. Die Verwendung der UUIDs fand ebenfalls planvoll statt, nachdem das Problem der enumerierbaren Integer-IDs erkannt wurde.

Die Lehren, die aus dem in 3.2.2 gezeigten Fall gezogen wurden, sind ein exzellentes Beispiel dafür, wie Secure by Default im Projekt praktiziert wird. Daher hebe ich dieses Paradigma besonders hervor. Zur Erinnerung: Für Webservices, die eine Authentisierung über das Mitsen-

den eines Tokens erfordern, existierte eine Annotation, die dies leistete: Sie ermittelte aus dem übermittelten Token die User-Rolle, bestimmte die zur Ausführung der Aktionen des Dienstes nötigen Berechtigungen und gewährte oder verweigerte die Ausführung in Abhängigkeit davon, ob mit der Rolle die nötigen Berechtigungen assoziiert waren.

Nachteil dieser Vorgehensweise ist, dass die Annotation aktiv einem Service hinzugefügt werden muss. Dies kann wie gesehen zu Fehlern führen: Abgesehen davon, dass es einfach ist, zu vergessen, die Annotation wieder einzukommentieren oder sie überhaupt zu verwenden, ist das Wissen um ihre Funktion nicht notwendigerweise bei allen Entwickler*innen vorhanden. Wenige Tage nach dem Vorfall entschloss sich daher das Team, die entsprechende Logik umzudrehen und stattdessen eine Annotation zu entwickeln, die Services von der Absicherung ausnimmt. Notwendig bleiben nicht abgesicherte Webservices zum Login und zur Anzeige von Versionsinformationen auf der Startseite der Anwendung. Umgesetzt wurde die Absicherung über einen sog. ContainerRequestFilter: Implementierungen dieses Interfaces aus dem javax-Paket werden automatisch erkannt und auf eingehende Requests angewandt. Auf diese Weise ist nun der Standardfall sicher.

An anderer Stelle besteht hingegen noch Spielraum für Verbesserungen: Das Konzept von Rechten und Rollen ist der Komplexität der Anwendung angemessen, letztere könnten allerdings noch feiner granuliert werden. So gibt es in der Anwendung einzelne Rollen, deren Set an Berechtigungen vergleichsweise groß ist. Accounts, die diese Rollen innehaben, wären damit attraktive Ziele für Identitätsdiebstahl & Spoofing.

Coding Guidelines

Von Anfang an existierte eine teamweite Festlegung zu Richtlinien der Programmierung. Auf unterster Ebene besteht diese in einem Formatter für die IDE, der im Git mitversioniert wird. Auf diese Weise werden Imports stets in gleicher Reihenfolge angeordnet, Einrückungen sind immer gleich lang und die Verwendung von Leerzeichen vor oder hinter Klammern und anderen Steuerzeichen sind überall identisch und es wird verhindert, dass wechselweise mit Tabs und Leerzeichen eingerückt wird. Dies mag in erster Anschauung trivial anmuten. Ohne standardisiertes Formatting werden jedoch bei Commits oft große Teile des Codes ohne Wirkung verändert, so dass es schwerer wird, die tatsächlich wirksamen Änderungen zu erkennen und so beispielsweise Bugs zu finden.

Darüber hinaus verfügen moderne Entwicklungsumgebungen wie Eclipse oder IntelliJ über eingebaute Werkzeuge zur Analyse des Quellcodes, die auf typische Fehler hinweisen können (z.B. ungenutzte Klassenmethoden). Diese lassen sich durch Plugins noch erweitern, so dass etwa Warnhinweise ab einer gewissen Verschachtelungstiefe angezeigt oder nicht

abgefangene potentielle NullPointerExceptions bereits zur Compilezeit erkannt werden. Die Konfiguration, wann die IDE welche Hinweise anzeigt, lässt sich ebenfalls exportieren und projektweit versionieren.

Das selbe Feld bespielen Werkzeuge zur statischen Codeanalyse wie SonarQube. Technisch besonders interessant ist, dass einige der eben erwähnten Plugins Schnittstellen bieten, um die Codeanalyse in der IDE nicht lokal, sondern durch einen SonarQube-Server durchzuführen. Auf diese Weise lassen sich einheitliche Standards besonders einfach durchsetzen.

Alle Warnhinweise in der IDE sind jedoch nichts wert, wenn sie nicht beachtet werden. Gemäß der „Broken-Window"-Theorie sinkt die Bereitschaft des Einzelnen, eigene Konventionsverstöße zu vermeiden oder gar zu beheben, drastisch mit dem offensichtlichen Vorhandensein solcher Verstöße in dessen Umfeld. Zu diesem Zweck kann es geboten sein, regelmäßige Refactorings in den Sprints zu planen. Essenziell hierfür sind natürlich einheitliche Standards und ggf. eine etablierte Sammlung von Patterns, so dass für identische Probleme kein Wildwuchs individueller Lösungen entsteht. Schlechte Erfahrungen wurden dagegen mit „harten" Überprüfungen geäußert, die etwa das Einchecken von Code technisch verhindern, wenn dieser gegen die Konventionen verstößt. Z.B. im Falle einer Notfallmaßnahme gelten andere Prioritäten als im Entwicklungsalltag, so dass die erzwungene Einhaltung der Konventionen beispielsweise über Git-Hooks im Ernstfall mehr Schaden als Nutzen bedeuten kann.

Sicheres Anwendungs-Template

Zur Vereinheitlichung von Architekturstandards bietet es sich an, zu Beginn eines neuen Projekts nicht alle Infrastrukturfragen neu zu stellen, sondern auf durchdachte und gehärtete Vorlagen beispielsweise für Kubernetes und Docker zurückgreifen zu können. Hier lassen sich in aktuell laufenden und vergangenen Projekten gemachte Erfahrungen bündeln. Um unterschiedlichen Technologiestacks gerecht zu werden, müssten entweder individuell angepasste Templates vorliegen, oder diese hinreichend modular aufgebaut werden, so dass beispielsweise C♯- und Java-Teams lediglich das Docker-Basisimage für den Businesslogik-Container austauschen müssten.

Automatisches Penetration-Testing

Das schon erwähnte OWASP Zed Attack Proxy (kurz „ZAP") lässt sich zum automatischen Penetration-Testing benutzen. Für den Jenkins Build-Server existiert außerdem ein Plugin, um ZAP in die Build-Pipeline zu integrieren. So ergibt sich die Möglichkeit, zum Beispiel jede Nacht ein festes Set an Tests durchzuführen oder gar Builds als fehlgeschlagen zu betrachten, wenn ZAP eine neue Schwachstelle findet bzw. die Nichtbehebung einer bekannten belegt.

Das Team kann damit die Definition of Done einer Evil User Story um einen Satz von Testfällen für ZAP erweitern. Auf diese Weise lassen sich die Ergebnisse der Arbeiten an einer solchen Story messen und überprüfen. Gleichzeitig werden wie bei Unit-Tests ungewollte Seiteneffekte deutlich, wenn Änderungen negative Auswirkungen an anderer Stelle haben.

4.3. Modellierung der Anwendung

Die textuelle Beschreibung des Aufbaus der Anwendung in 3.3 wird durch die Abbildung A.2 im Anhang ergänzt. Eine Anmerkung vorweg: Ich bezeichne die Grafik als Infrastrukturdiagramm in dem Wissen, dass der UML-Standard diesen Begriff für einen anderen Diagrammtyp benutzt, der hier aber nicht gemeint ist. Gleichwohl beschreibt der Name am besten den Inhalt der Darstellung, weshalb ich der Leserschaft diese Irritation zumute.

Der im Diagramm zu sehende Pfeil, der die Datenflussrichtung beschreibt, deutet zwar in Richtung User, die entgegengesetzt ablaufenden Anfragen von seiner Seite verlaufen jedoch grundsätzlich entlang des selben Pfades. Der grundlegende Kommunikationsfluss beginnt also mit dem Auflösen der Anwendungs-URL zu einer festen IP auf einem der dargestellten Cloud-Dienste. Hier wird der Verkehr erstinstanzlich von einer Firewall gefiltert. Danach gelangt er zum HAProxy, wo der Verkehr entschlüsselt, an die verschiedenen Endpunkte weiter verteilt und die Skalierung der Anwendung in Form der Zahl hochgefahrener Container vorgenommen wird. Um das Thema Orchestrierung von Kubernetes-Clustern nicht zum Hauptgegenstand der nächsten Seiten zu machen, soll an dieser Stelle der Hinweis genügen, dass die nun folgenden Komponenten der eigentlichen Anwendung in Form von Frontend, Backend und Datenbank jeweils in Docker-Containern laufen und die jeweilig höhere Schicht die URL der darunterliegenden kennt.

Das heißt, dass alle User-Anfragen beim Frontend landen. Die dort laufende Angular-Anwendung nimmt eine Vorverarbeitung der Requests vor und stellt ihrerseits Anfragen ans Backend. Hier findet gemäß des traditionellen Drei-Schichten-Modells die Abwicklung der eigentlichen Business-Logik statt. Für Lese- und Schreiboperationen greift das Backend wiederum auf die Persistenzschicht in Form der MariaDB zu. Die Antwort nimmt genau den umgekehrten Weg: Die abgefragten Daten laufen von der Datenbank ins Backend, dieses führt damit seine Business-Operationen aus und beantwortet mit dem Ergebnis die eröffnete Anfrage des Frontends, das die Daten in einem View rendert und diesen dem User darstellt.

4.3.1. Baseline

Die nebenstehende Grafik reduziert die oben stehende auf die essenziellen Kommunikationsflüsse innerhalb der Anwendung. Erläuterungsbedürftig ist die zusätzliche Trust Boundary zwischen Front- und Backend:

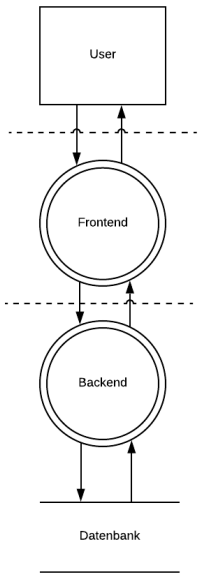


Abbildung 4.4.: Minimales Datenflussdiagramm Level 0: Gesamtübersicht über die Anwendung

Die Komponenten der Software laufen innerhalb eines Kubernetes-Clusters. Der Webserver, der das Frontend ausliefert, ist Endpunkt des Routings nach DNS-Auflösung. Daher mag der Umstand kontraintuitiv erscheinen, dass das Backend ebenfalls öffentlich exponiert sein muss. Das Frontend ist jedoch als Single Page Application in JavaScript ausgeführt: Es wird daher tatsächlich nicht wie eine serverseitige PHP-Anwendung auf dem ausliefernden Webserver zur Ausführung gebracht, sondern vom Client heruntergeladen, woraufhin es in dessen Browser läuft. Requests gegen das Backend stellt folglich das Endgerät des Users, das sich außerhalb des Clusters befindet - um mit dem Backend kommunizieren zu können, muss dieses daher von außen erreichbar sein. Diese Erkenntnis ist folgenreich für die spätere Analyse der Bedrohungen an dieser Stelle. Die Kommunikation mit der Datenbank hingegen findet allein innerhalb des Clusters statt. Wenn ein Angreifer also nicht in der Lage ist, in einen solchen Cluster einzubrechen (was zum aktuellen Zeitpunkt keine realistische Bedrohung darstellt), muss dieser Datenfluss nicht extra vor ihm z.B. durch Verschlüsselung gesichert werden.

Gemeinsam stellen diese beiden Boundaries die wichtigsten Kontextübergänge der gesamten Anwendung dar, da hier Usereingaben das erste mal verarbeitet werden. Streng ausgelegt geschieht dies natürlich bereits an der Firewall, doch aus Sicht der Softwareentwicklung interessiert hier vor allem der Teil der Verarbeitung, der in der Anwendung geschieht.

Im Folgenden gilt es nun, die Abläufe innerhalb von Front- und Backend weiter auszumodellieren.

4.3.2. Exemplarische Detailmodellierung der Login-Komponente

Der Login eines Users involviert alle Schichten der Anwendung und beinhaltet sowohl das Verarbeiten von Eingabedaten als auch die Erzeugung und Rückübermittlung eines Ergebnisses. Er steht damit stellvertretend für eine Vielzahl von Business-Prozessen, die sich auf diesem Grad der Abstraktion nur durch die Natur der übermittelten Daten unterscheiden.

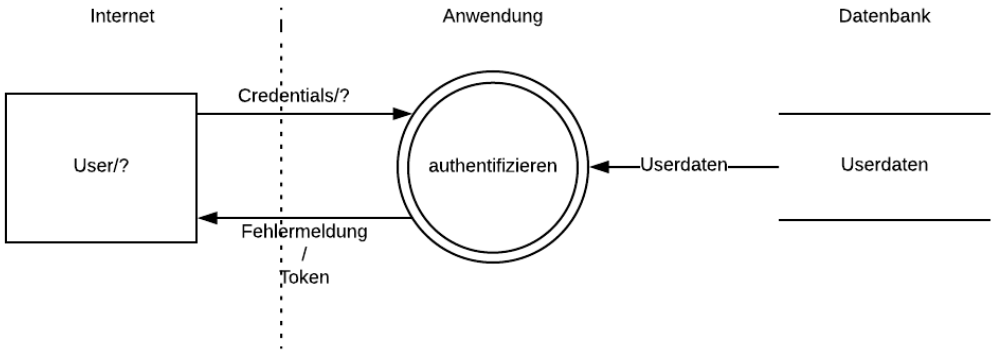


Abbildung 4.5.: Datenflüsse beim Login, Level 0

Das Rechteck links symbolisiert eine externe Entität, das seitlich offene am rechten Rand ein Datenquelle, in diesem Fall die Userdaten-Tabelle in der Datenbank, und der mittige Doppelkreis eine Komposition verschiedener Prozesse. Zumindest in einem Level 0-Diagramm können innerhalb einer Trust Boundary mehrere hinreichend ähnliche Prozesse durch eine solche Darstellung zusammengefasst werden.

Nicht sofort ersichtlich mag hier sein, warum das Frontend nicht modelliert wurde: Aus bereits genannten Gründen entzieht sich jeglicher Kontrolle, welche Gegenstelle mit dem Backend kommuniziert. Selbiges muss für das Frontend erreichbar sein, aber es gibt zumindest keinen technischen Grund, darauf zu vertrauen, dass eingehende Requests vom auf einem Client laufenden Frontend stammen und nicht von einer Drittpartei mit unklaren Motiven. Diese Ungewissheit drückt sich aus in den Fragezeichen hinter der Entität wie auch dem Datenfluss in Richtung Anwendung. Diese Trust Boundary, die die Daten dabei überqueren, ist daher die primäre der ganzen Software.

Das Backend gleicht die erhaltenen Daten mit den Beständen in der Datenbank ab und gibt entweder eine Fehlermeldung zurück, falls die Kombination aus Name und Passwort nicht

zusammenpassen, oder stößt die Erzeugung eines Tokens an. Im Erfolgsfall wird dieses dann an die die Anfrage stellende Instanz zurück übermittelt.

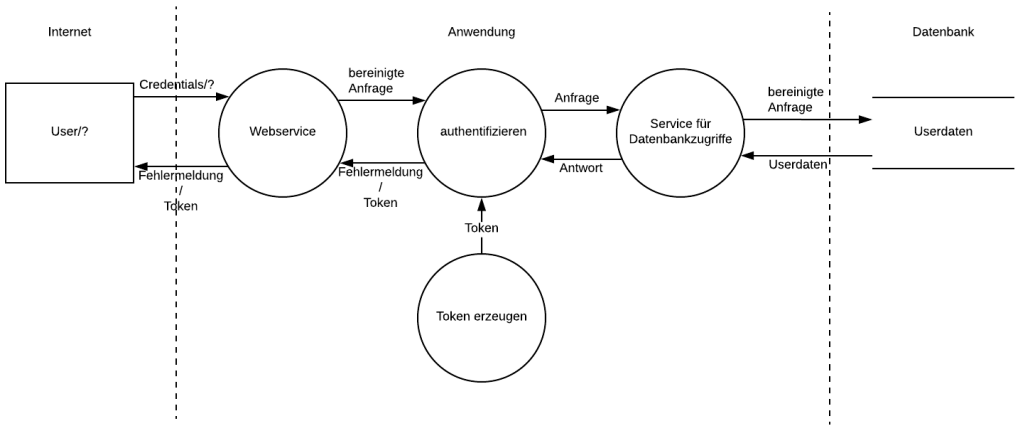


Abbildung 4.6.: Datenflüsse beim Login, Level 1, größere Darstellung in A.3

In dieser Darstellung des selben Vorgangs sind alle in der oberen Grafik noch durch Doppelkreise repräsentierten zusammengefassten Prozesse dekomponiert. Ebenfalls wurde eine weitere Trust Boundary in Richtung der Datenbank eingeführt: Man könnte zwar einerseits argumentieren, dass Angriffe durch z.B. SQL-Injection bereits im Webservice erkannt werden sollten. Andererseits lassen es die Paradigmen der Defense in Depth und Separation of Concerns sinnvoll erscheinen, an diesem zentralen Zugangsservice zur Datenbank im Zweifelsfall noch eine Überprüfung, in diesem Fall exklusiv auf Anzeichen von SQL-Injections, durchzuführen.

Ebenfalls wird deutlich, dass alle Datenströme, welche die Trust Boundary zwischen Internet und Anwendung überschreitet, bereits im Webservice validiert werden sollten, also noch bevor sie in den eigentlichen Authentifizierungsprozess gelangen.

4.4. Bedrohungs- und Risikoanalyse

4.4.1. STRIDE in Aktion

Äußere Trust Boundary

Ich betrachte an dieser Stelle zunächst den mit „Credentials/?“ bezeichneten Datenfluss von der externen Entität in Richtung des Webservices. Zur Erinnerung sei noch einmal darauf hingewiesen, dass Untersuchungsgegenstand der aktuelle Stand der Implementation ist, nicht die abstrakte Planung.

- Spoofing: Ja, eine Dritte Person könnte versuchen, sich als autorisierte*r Nutzer*in bzw. als Anwendungsserver auszugeben
- Tampering: Nein, die Daten werden mit TLS verschlüsselt übertragen, böswillige Änderungen erkennt bereits das Protokoll
- Repudiation: Eher nein, außer in Verbindung mit Spoofing, daher aber zumindest nicht als eigene Bedrohung
- Information Disclosure: Nein, denn wegen der TLS-Verschlüsselung können die über diese Schnittstelle übertragenen u.a. Benutzername & Passwort nicht in fremde Hände geraten
- Denial of Service: Ja, die Schnittstelle ist nach außen exponiert und daher prinzipiell Angriffen auf die Verfügbarkeit ausgesetzt
- Elevation of Privilege: Nein, der Webservice läuft als Teil einer Java-Anwendung innerhalb der Sandbox der JVM auf einem Docker-Container und ist damit hinreichend isoliert.

Auf dem Rückkanal gelange ich zu folgender Bewertung:

- Spoofing: Ja, denn wenn eine gefälschte Absenderadresse eingeschleust wurde, geht die Rückantwort an ein falsches Ziel
- Tampering: Nein, s.o. (andere Richtung der Kommunikation)
- Repudiation: Nein, s.o.
- Information Disclosure: Nein, denn das Token muss zwingend geheim gehalten werden, was aber wegen der Verschlüsselung ohne Hinzunahme anderer Bedrohungen gewährleistet ist

- Denial of Service: Ja, denn mit Slowloris-Angriffen usw. kann der Dienst massiv ausgebremst werden
- Elevation of Privilege: Nein

Innere Trust Boundary

Diese Kommunikation ist im Level 0-Diagramm unilateral modelliert, obwohl sie technisch betrachtet natürlich ebenfalls bilateral funktioniert. Es findet während des Logins jedoch nur lesender Zugriff auf die Datenbank statt, weswegen ich mich auf die Kommunikation in Richtung der Antwort konzentriere. Die Datenbank vor böswilligen Schreibzugriffen zu schützen, muss Gegenstand anderer Analysen sein.

- Spoofing: Nein, die Datenbank antwortet schlicht auf Anfragen und der Prozess kennt die Adresse der Datenbank aus der Konfiguration der Anwendung
- Tampering: Nein, die Kommunikation findet innerhalb des Clusters statt und kann daher nicht ohne weiteres manipuliert werden
- Repudiation: Nein, s. äußere Trust Boundary.
- Information Disclosure: Nein, s. Tampering
- Denial of Service: Nein, da die Datenbank nur auf Anfragen antwortet, die der fragende Service selbst gestellt hat
- Elevation of Privilege: Nein

Wie man sieht, könnte auf die innere Trust Boundary im Prinzip verzichtet werden, da keine einzige der STRIDE-Bedrohungen festzustellen ist. Dieser Umstand tritt allerdings erst durch die Modellierung und Analyse zutage.

4.4.2. Risikobewertung

Alle soeben festgestellten Bedrohungen mit der OWASP-Methode zu bewerten, würde den Rahmen dieser Arbeit sprengen. Daher beschränke ich mich auf die Eingangsrichtung der äußeren Trust Boundary und stelle die Berechnung nur für Spoofing dar.

Ich gehe dabei die vom OWASP vorgegebenen Faktoren durch und gebe dabei jeweils meine Bewertung in Form einer Zahl in Klammern ab. Zu jedem Faktor liefert die Methode eine Hand voll Optionen der Bewertung, die einer Zahl von 0 - 9 zugeordnet sind. 0 bedeutet hierbei kein

und 9 ein extrem hohes Risiko. Im Zweifelsfall soll stets der schlimmste Fall angenommen werden.

Der erste Schritt der OWASP Risk Rating Methodology, die Identifikation eines Risikos, ist bereits getan: Spoofing bedeutet in diesem Kontext an dem Login-Endpunkt der Anwendung, dass ein Angreifer sich als autorisierte*r Nutzer*in der Anwendung ausgibt. Um diese exemplarische Risikobewertung auf einen einzelnen Fall zu beschränken, lasse ich das Web-Spoofing aus: Ebenfalls könnte ein Angreifer beispielsweise durch Phishing versuchen, einem User eine manipulierte Adresse unterzuschieben, die diesen auf eine vom Angreifer kontrollierte Seite leitet, wo der User durch optische Nachahmung der eigentlichen Anwendung verleitet wird, vertrauliche Daten wie Passwörter o.ä. preiszugeben (BSI, 2016, S. 1190). Dies entspräche einem Man in the Middle-Angriff.

Nötig für ein Spoofing zwischen Angreifer und Anwendung ist hingegen die korrekte Kombination aus Benutzername und Passwort eines anderen Users. Der Rahmen dieser Bedrohung muss dabei auf technische Faktoren eingeschränkt werden, da z.B. Social Engineering durch die Softwareentwicklung fast gar nicht und Phishing nur teilweise beeinflusst werden können.

Es folgt die Bestimmung der Likelihood, also der Eintrittswahrscheinlichkeit, mit den sog. Threat Agent Factors:

- Da nicht klar einzugrenzen ist, wer an dieser Stelle versuchen würde, einen Spoofing-Angriff durchzuführen, ist über das Fähigkeitsniveau möglicher Angreifer zunächst keine Aussage möglich - ich klammere diesen Faktor daher von dieser Bewertung aus.
- Als Belohnung winkt der Zugang zu Geschäftsdaten, wenn auch nicht z.B. denen einer Banking-Anwendung oder zu hochsensiblen privaten Inhalten (4).
- Die notwendigen Ressourcen sind, wie schon erwähnt, ein gültiger Satz fremder Nutzerdaten. Da wir als Untersuchungsgegenstand die aktuelle Implementation der Software gewählt haben und nicht ein theoretisches Planungskonstrukt, muss die Erlangung eines solchen Datensatzes aus technischer Perspektive als extrem schwer betrachtet werden, solange nicht bereits ein Zugang über einen Administrator-Account vorhanden ist (0).
- Prinzipiell möglich ist der Versuch jedoch für alle Internetteilnehmer (9).

Hinzu kommen die Vulnerability Factors genannten Faktoren:

- Zur Entdeckung der Backend-Adresse müssen entweder interne Informationen vorliegen oder der Frontendcode analysiert werden, was wegen des zu JavaScript kompilierten TypeScript-Codes der Angular-Anwendung keine leichte Aufgabe ist (3).

- Da an dieser Stelle nach bisherigem Kenntnisstand technisch sauber gearbeitet wurde und daher keine Schwachstelle vorliegt, ist die Übernahme einer fremden Identität ohne Maßnahmen wie Social Engineering etc. allenfalls theoretisch möglich (1).
- Die Anwendung ist nur dem Mitarbeiterstamm des Kunden bekannt, dieser potentielle Einstiegspunkt für Spoofing-Angriffe darf also bislang wohl als unbekannt gelten (1).
- Sollte es dennoch zu einem erfolgreichen Spoofing kommen, so kann dies durch das Logging erkannt werden - bisher existiert aber keine Review-Praxis für Logs (8).

Insgesamt ergibt dies eine Eintrittswahrscheinlichkeit von $\frac{4+0+9+3+1+1+8}{7} = 3,7$, was der **mittleren** von drei Stufen entspricht.

Als nächstes steht die Bewertung der Impact Factors an:

- Bei erfolgreichem Spoofing können signifikante Mengen vertraulicher Daten in falsche Hände geraten, wenn auch aufgrund des Rechte- und Rollenmodells nie alle auf einmal (7).
- Gleiches gilt für die Integrität der Daten (7).
- Die Funktionalität der gesamten Anwendung hängt aus Geschäftsperspektive in hohem Maß vom Vorhandensein und der Korrektheit des Datenbestandes ab (7).
- Die Kombination aus Logging und der bereits implementierten Nachverfolgbarkeit von Änderungen in den Daten lässt allerdings eine weitgehende Zuordnung eines solchen Spoofings zu (1).
- Da die Anwendung auf eine Datenredundanz vorbereitet ist, hielten sich die finanziellen Auswirkungen höchstwahrscheinlich in überschaubaren Grenzen (3).
- Der Kunde kommuniziert aktiv die Qualität seiner Dienstleistung über das Vorhandensein der vorliegenden Software, so dass ein Sicherheitsvorfall wahrscheinlich zum Verlust einzelner Kunden führen könnte (4).
- Die Einhaltung von eingegangenen Verträgen dürfte wegen der genannten Redundanz der Geschäftsdaten allenfalls minimal in Gefahr geraten (2).
- Auf der anderen Seite beinhalten die möglicherweise offengelegten Daten persönliche Angaben von Personen, deren Zahl sich im niedrigen dreistelligen Bereich bewegt (5).

In Summe bedeutet dies Impact Factors der Stärke $\frac{7+7+7+1+3+4+2+5}{8} = 4,5$, also ebenfalls **mittel**, so dass das Risiko **insgesamt gleichfalls als mittelgroß** (mittel · mittel) bewertet wird.

Im folgenden Schritt würde nun anhand der Bewertung anderer Risiken sowie technischer und wirtschaftlicher Erwägungen im Sprint Planning evaluiert, wie mit diesem Risiko zu verfahren ist.

Dabei ist der soeben ermittelte Wert bemerkenswert hoch: Obwohl auf technischer Ebene alle wesentlichen Vorkehrungen getroffen wurden, bleibt der Endpunkt nun mal öffentlich im Internet exponiert. Auf diese Weise trifft eine hohe technische Hürde auf minimale Zugangsbeschränkungen für diesbezügliche Versuche. Dies ist ein Beispiel für ein Restrisiko, das auf jeden Fall dokumentiert werden muss, z.B. im angesprochenen Risikoregister. Würde im Laufe des Livebetriebs z.B. eine Schwäche in einer verwendeten Krypto-Bibliothek bekannt, darf nicht erst langwierig evaluiert werden müssen, wie mit ihr zu verfahren ist. Es muss stattdessen eine Sofortmaßnahme bereitstehen, etwa IP-Whitelisting oder gar die vorübergehende Abschaltung des öffentlichen Zugangs. Solche Maßnahmen sind bei der Entwicklung bereits vorzubereiten.

4.4.3. Eine beispielhafte Evil User Story

Da die Anwendung bereits recht weit entwickelt ist und das Team über eine gewisse Erfahrung auch in der Implementation sicherheitskritischer Softwarekomponenten verfügt, bestehen zum jetzigen Zeitpunkt zumindest keine offensichtlichen technischen Optionen mehr, das konkrete Spoofing-Risiko beim Login weiter einzudämmen. Würde sich das Team im Planning dennoch dazu entscheiden (oder z.B. die Komponente von Grund auf neu entwickeln), wäre hierfür ein Task in Form einer Evil User Story zu schreiben und der Gesamtstory „Login Komponente“ anzuhängen. Nachfolgend tue ich daher so, als gäbe es die oben erwähnte Implementation noch nicht.

In diesem Fall ist, wie bereits dargelegt, die Rolle eines Angreifers nicht klar ersichtlich. Es ist denkbar, dass die Backendadresse von Softwaretools entdeckt wird, die ganze IP-Adressbereiche im Internet nach offenen Ports scannen und automatisch mal mehr, mal weniger zufällige Zeichenkombinationen an Nutzername- und Passwortfelder in gefundenen Login-Formularen senden. Gleichsam könnte ein böswilliger Konkurrent des Kunden versuchen, ihm zu schaden. Schon in diesen beiden Fällen unterscheidet sich die jeweilige Motivation der Angreifer und die Aussagen, die über ihre Fähigkeiten getroffen werden können, dramatisch. Daher kann dessen Rolle hier nicht über einen abstrakten „Hacker“ hinaus konkretisiert werden.

Der Angriffsvektor selbst hingegen ist unzweideutig, und eine aus dem Spoofing-Risiko abgeleitete Evil User Story könnte lauten:

„Als Hacker kann ich auf technischem Wege an Logindaten anderer User gelangen."

Während des Plannings wären dann bereits, wie bei jeder anderen Story, Lösungsansätze zu diskutieren und die Definition of Done zu bestimmen: Um zu vermeiden, dass ein Angreifer im selben Netz wie ein legitimer User einfach dessen Netzwerkverkehr belauscht, wäre die ausschließlich verschlüsselte Übertragung aller Kommunikation mit dem Backend sicherzustellen. Zum Verhindern von Brute-Force bzw. Wörterbuchangriffen, also letztlich des Ratens der Logindaten, kann die Einführung einer Verzögerung der Backendantwort und/oder gar die Sperrung von Accounts nach einer zu bestimmenden Zahl fehlgeschlagener Login-Versuche gegeben sein - was jedoch seinerseits einen Denial of Service-Vektor öffnen würde. Als DoD könnte ein Integrations- oder Frontendtest festgelegt werden, der die korrekte Implementation überprüft.

4.4.4. Nützlichkeit der IT-Grundschutzkataloge

Um von einer Evil User Story zu einer Liste von umzusetzenden Schutzmaßnahmen zu gelangen, könnte man nun versuchen, dies anhand der Grundschutzkataloge zu tun. Jedoch sind die auf STRIDE aufbauenden Analysen und Tasks aus der Perspektive möglicher Angreifer und die Kataloge aus der der Verteidiger gedacht und die Gefährdungskataloge leisten leider keine Verknüpfung zwischen Gefährdungen und Maßnahmen. Beispielsweise könnte in G 5.87: „Web-Spoofing" auf passende Einträge der Maßnahmenkataloge verwiesen werden, dies findet jedoch nicht statt. Für die beschriebene Vorgehensweise sind die IT-Grundschutzkataloge daher nur von begrenztem Wert.

4.5. Besonderheiten bezüglich der SPA-Aspekte

Mit Facebook, Twitter und den Produkten der Google-Familie bestehen heute wesentliche Bestandteile des Internets aus Single Page Applications. Grund genug, an dieser Stelle noch einige abschließende Erwägungen zu ihren Eigenheiten auszuführen.

4.5.1. Node.js & npm

SPAs werden selten ohne technische Unterstützung in Form von Frameworks entwickelt. Zum Einsatz im Projekt kommt wie erwähnt Angular in der Version 5. Eingangs wurde ebenfalls bereits ausgeführt, dass gemessen an seiner Verbreitung die üblichen Datenbanken für Sicherheitslücken verhältnismäßig wenige Einträge für Angular enthalten, zumal sich der Großteil

derselben auf ältere Versionen bezieht. Gleichwohl teilt Angular eine prinzipielle Schwäche mit allen Technologien, die z.B. Node.js & den Paketmanager npm einsetzen: Es ist nicht nur verführerisch, sondern teilweise auch ökonomisch geboten, für eine in der Entwicklung zu lösende Aufgabe externe Repositories nach einer Drittbibliothek zu durchsuchen. Ein gedankenloser Umgang hiermit birgt jedoch nicht zu unterschätzende Risiken. Für die Nutzer*innen einer npm-Bibliothek ist oft nicht ersichtlich, von welcher Qualität der Code ist, den sie gerade ins Projekt einbinden. Dies eigenhändig zu überprüfen ist zeitlich nur möglich, wenn sich die Nutzung von Fremdcode in Grenzen hält. Exponentiell erschwert wird dies dadurch, dass für die meisten Bibliotheken auch noch Abhängigkeiten mit installiert werden müssen.

Der Node Package Manager bietet seit Version 6 mit `npm audit` ein eigenes Kommando zum Zweck, bekannte Anfälligkeiten in den Abhängigkeiten eines Projekts aufzuspüren (npm Documentation, 2018, Getting Started, 21 - How to run a security audit with npm audit). Mit `npm audit fix` kann npm zwar versuchen, automatisch kompatible Updates für problematische Lücken zu finden, doch der Erfolg dieser Maßnahme hängt davon ab, ob es erstens überhaupt Updates gibt und zweitens selbige das Verhalten der Bibliothek nicht entscheidend ändern.

Daher ist in Projekten, die auf Node.js aufbauen, eine Regelung für den Umgang mit Abhängigkeiten zu finden und wie Coding Guidelines projektweit festzuhalten. Dies kann zB. bedeuten, dass Pakete aus dem npm-Repository nur benutzt werden dürfen, wenn für sie keine CVE vorliegt, oder die Bibliothek nur verwendet wird, wenn begründet werden kann, dass die konkrete Schwachstelle im Projekt keine Bedrohung darstellt (ggf. durch Anwendung von STRIDE).

4.5.2. Kompartmentalisierung

Front- und Backend sind beim Einsatz von SPA-Frameworks weniger stark miteinander gekoppelt als bei vielen anderen Frontendtechnologien. Bei der aktuellen Projektgröße drängt es sich nicht auf, bei Einsatz von deutlich mehr als zehn Entwickler*innen böte sich jedoch die Aufteilung in Teams für Front- und Backend an. Die Grenze zur Microservice-Architektur ist an dieser Stelle fließend, so dass mit zunehmender Kompartmentalisierung die Lehren dieser Disziplin an Bedeutung gewinnen. Ab einer gewissen Grenze ist es ggf. sinnvoller, Datenflussdiagramme für einzelne Anwendungsteile zu gestalten, anstatt stets den Weg einer Anfrage durch alle Schichten zu modellieren. Gleichwohl darf die Gesamtübersicht nicht aus dem Blick geraten, da Microservices ihre ganz eigene Art von Komplexität in ein Projekt bringen.

Hingegen erleichtert es die Kompartmentalisierung, Dienste weniger plattformabhängig zu gestalten. Die Webinterfaces der eingangs genannten großen Anbieter sind letztlich nur ein

Weg von mehreren, mit ihren Diensten zu kommunizieren. Zusätzlich stehen für viele Services native Anwendungen auf verschiedenen Zielplattformen zur Verfügung, die jedoch alle stets mit dem selben Business-Backend kommunizieren. Die Oberfläche der Anwendungen ist damit inhärent austauschbar. Die Kehrseite einer solcher Diversifizierung ist gezwungenermaßen eine Vergrößerung der Angriffsoberfläche, weil jeder einzelne Client problematische Schwachstellen aufweisen kann und eine Kette von Sicherheitsmaßnahmen stets nur so stark ist wie ihr schwächstes Glied.

4.5.3. Einfluss der Technologie auf Risikobewertung

Der JavaScript-Code des Frontends kommt wie geschildert nicht auf der Infrastruktur eines Hostinganbieters zur Ausführung, sondern auf den Clients. Dies hat erheblichen Einfluss nicht nur auf die Modellierung der Anwendung, sondern damit auch auf die Bewertung von Risiken. Die Präsentationsschicht der Software liegt dem SPA-Paradigma folgend vollständig auf der Clientseite. Dies beinhaltet nicht nur das Rendering und die Anpassung des Views an veränderliche Inhalte, sondern auch die Verwaltung von Sitzungsstatus und Zuständen. Diese sind von erheblicher Bedeutung für die Sicherheit der Anwendung in jeder Hinsicht: Da Clients nur im Ausnahmefall in gleichem Maße vertraut werden kann wie z.B. einem eigenhändig administrierten Application-Server, muss die zusätzliche Trust Boundary (vgl. 4.4.1) stets bedacht werden.

Twitters seit 2012 in der Presse zu beobachtenden Bemühungen, den API-Zugang für Drittanbieter-Clients einzuschränken, verleiht diesem Problem Ausdruck: Der Dienst spielt durch Werbung Umsatz ein. Für Drittanbieter besteht hingegen ein Anreiz, Twitters Dienst auf API-Ebene zu benutzen und Benutzern in eigenen (z.T. bezahlten) Clients keine Werbung anzuzeigen. Twitter begründet die Einschränkungen der API interessanterweise primär mit der DSGVO, also dem Datenschutz, und der Sicherheit (Twitter Developer Blog (2018)). Tatsächlich ist fraglich, ob dies wirklich die Gründe für die Änderungen der API sind. Es macht jedoch deutlich, dass es die SPA-Architektur erschwert zu kontrollieren, auf welche Weise mit dem eigenen Backend kommuniziert wird. Bei z.B. Facebook und YouTube hat das Problem andere Dimensionen, doch auch für diese Dienste existieren Drittanbieter-Apps.

Dieser Punkt muss bedacht werden, wenn man beispielsweise STRIDE für die Kommunikation zwischen externen Entitäten und seiner Backend-Schnittstelle durchführt; Lastangriffe etwa, die früher auf serverseitig ausgeführte Scripte abzielten, funktionieren zwar nicht mehr, jedoch kann nicht mehr die Gutartigkeit von Datenströmen „hinter“ dem Frontend angenommen werden.

5. Ausleitung

5.1. Zusammenfassung der Ergebnisse

5.1.1. Grundsätzliches

Theoretische Erwägungen ergaben bereits zu Beginn der Arbeit, dass ihr primärer Fokus auf der Etablierung von Verfahren des IT-Risikomanagements innerhalb eines Vorgehensmodells wie Scrum liegen würde. Die im zweiten Kapitel erarbeitete Übersicht über den Inhalt der IT-Grundschutzkataloge, den SDL und Teile des OWASP legte offen, dass diese Zusammenführung bislang kaum formalisiert worden ist. Gleichwohl ergab die Beschäftigung mit diesen Leitfäden Hinweise auf später angewandten Werkzeuge. Ein herauszuhebendes Ergebnis das Software Assurance Maturity Model des OWASP, das die benötigte Metrik zur Bestimmung des sicherheitstechnischen Reifegrades eines Entwicklungsprozesses lieferte.

Im dritten Kapitel habe ich die Essenz der drei Werke bzw. Sammlungen bezüglich des Risikomanagements und das untersuchte Projekt vorgestellt. Auf dieser Basis konnte ich mit dem SAMM die Ergebnisse meiner Vorüberlegungen bekräftigen: Ein methodisches Einflechten von Risikomanagementverfahren in den Entwicklungsprozess ist nicht nur prinzipiell sinnvoll, sondern im untersuchten Projekt stellt die Abwesenheit eines solchen sogar eine der größten methodischen Schwachstellen zur Vermeidung von IT-Sicherheitsvorfällen dar.

Insgesamt fällt die Bewertung mit der SAMM-Metrik keineswegs vernichtend aus, deutet aber auf einen Umstand hin, der sich künftig zu einem Problem auswachsen könnte: Die Qualität der technischen Umsetzung erwächst hier ganz offensichtlich zumindest nicht hauptsächlich aus einem elaborierten, verallgemeinerbaren Prozessmodell. Das heißt, dass sie mit der Konstellation, Qualifikation, der Erfahrung und dem Wissen der Entwickler*innen steht und fällt. Obwohl selbstverständlich nichts gegen ein fähiges und gut organisiertes Team einzuwenden ist, muss dieses direkte Abhängigkeitsverhältnis kritisch gesehen werden: Personalausfälle sind oft nicht zu verhindern und Arbeitgeberwechsel insbesondere in heutigen Angestellten- und Freelancerverhältnissen eher Regel als Ausnahme. Nur, wenn die Sicherheit einer entwickelten Software auch durch einen geeigneten Entwicklungsprozess unterstützt wird, ist die Softwareentwicklung einigermaßen gefeit vor derartigen Unwägbarkeiten.

Klar wurde außerdem, dass die Basis zur methodischen Identifikation von Bedrohungen ein Modell der Anwendung sein muss, anhand dessen STRIDE durchgeführt werden kann, um die daraus gewonnenen Erkenntnisse mit der Risk Rating Methodology des OWASP bewertet zu können. Als besonders geeignete ergänzende Maßnahmen zur Begleitung der Entwicklung ermittelte ich in Abstimmung mit der Security-Arbeitsgruppe regelmäßige Schulungen aller Entwickler*innen in Sachen IT-Sicherheit und Regularien zu Paradigmen sicheren Designs, Coding-Guidelines und die Evaluierung eines Templates für Anwendungen, das grundlegende Fehler zu vermeiden hilft. Um den Erfolg umgesetzter Schutzmaßnahmen im Projekt zu ermitteln, versprach automatisiertes Penetration-Testing mit dem OWASP ZAP-Tool in die Build-Pipeline zu integrieren den größten Erfolg.

5.1.2. Modellierung, STRIDE & Risikobewertung

Das Erstellen eines Modells muss innerhalb eines Sprints stattfinden können, was gewisse Anforderungen an die Modellierungstechnik stellt. Gegenüber standen sich hierzu die Modellierung nach IT-Grundschutz und Datenflussdiagramme nach SDL. Im vierten Kapitel fiel die Entscheidung für letztgenannte, weil sie besser mit dem iterativen Scrum-Vorgehen zusammenpasst.

Neben einer stark vereinfachten Zusammenfassung der gesamten Anwendung, an der aber bereits sicherheitsrelevante Details bezüglich der Kommunikation zwischen Front- und Backend abzulesen waren, modellierte ich exemplarisch den Login-Vorgang aus. Anhand dieses Modells konnte ich die wesentlichen Kommunikationsflüsse der Komponente bestimmen (die in ihrem Kommunikationsverhalten stellvertretend für viele Business-Logiken der Anwendungen stehen kann). Das dem SDL entnommene STRIDE-Verfahren ergab verschiedene Bedrohungen, von denen ich beispielhaft die Gefahr des Spoofings mit der OWASP-Methode bewertet habe.

5.1.3. Folgerungen für den agilen Prozess

Aus dem Obenstehenden formulierte ich Verbesserungen für das agile Vorgehen mit Scrum in zwei Varianten, genannt A priori und A posteriori. In der ersten wird vor dem Sprint Planning eine in der nächsten Iteration umzusetzende Story aus dem Product Backlog wie beschrieben modelliert, anhand des Modells eine Bedrohungsanalyse mit STRIDE durchgeführt und diese Bedrohungen nach der OWASP Risk Rating Methodology bewertet. Im Sprint Planning wird entschieden, wie mit dem Risiko zu verfahren ist. Wenn technische Gegenmaßnahmen beschlossen werden, wird beim Zerlegen der User Story in Tasks für jedes Risiko, das weiterer technischer Aufmerksamkeit bedarf, eine Evil User Story als Task angelegt. Eine solche enthält

einen aus Perspektive eines Angreifers formulierten Weg, die Schutzgüter der Anwendung in Gefahr zu bringen. Dieser Task ist wie jeder andere im nächsten Sprint abzuschließen, bevor eine Gesamt-Story als fertig gelten kann. Zusätzliche Aufwände werden nicht auf dem Task selbst vermerkt, sondern auf die Implementationstasks aufgeschlagen, um zu vermeiden, dass Entwicklung und Absicherung als getrennte Tätigkeiten wahrgenommen werden.

Die zweite Variante unterscheidet sich von der vorgenannten darin, an welcher Stelle die Erweiterungen in den Scrum-Prozess eingehängt werden. Anstatt vorher bereits architektonische Tätigkeiten durchzuführen, wird im Planning selbst ein grundlegendes DFD erstellt, Bedrohungen vom Team gemeinsam mit STRIDE modelliert und diese grob in Auswirkung und Eintrittswahrscheinlichkeit bewertet. Dies stellt die Arbeitsgrundlage für Evil User Stories dar, die wie in der vorgenannten Variante geschrieben und bearbeitet werden. Bei Bedarf können besonders wichtige oder Hochrisikokomponenten zu Sprintbeginn noch detaillierter modelliert und die Risikoeinschätzung mit der OWASP-Methode präzisiert werden.

In beiden Fällen ist über das (Rest-) Risiko Buch zu führen, wozu ich die Aufnahme eines Risikoregisters in die Projektdokumentation vorschlage.

Als zusätzliche Rolle kann ein agiles Team außerdem einen Security Champion bestimmen, welcher hauptverantwortlich zeichnet für Fragen der IT-Sicherheit in der Entwicklung. Zu dessen Kompetenzen gehört die Bedrohungs- und Risikoanalyse inklusive der Modellierung, die der Champion jedoch nicht allein durchführen sollte, sondern im Zeichen des Wissenstransfers und der Ergebnisqualität mit mindestens einer weiteren Person. Je nach Strenge der vorhandenen Scrum-Umsetzung muss hierbei das Verständnis der Rolle aufgeweicht werden, weil feste Rollenzuweisungen insbesondere von außen der Methodologie von Scrum widersprechen.

5.2. Bewertung und Fazit

Die vorgestellten Ergebnisse wurden in der Security-Arbeitsgruppe diskutiert. Die A posteriori-Variante der Erweiterungsimplementation in Scrum ist ein Zugeständnis an die hier geäußerten Anregungen, wie die neuen Verfahrensschritte besser mit dem Vorgehen nach Scrum in Deckung zu bringen sein könnten. Die grundlegende Herangehensweise fand dabei jedoch durchaus Zustimmung.

Zur A posteriori-Vorgehensweise ist anzumerken, dass durch die Verschränkung von Modellierung, STRIDE und Risikobewertung mit dem Planning dieses u.U. sehr zeitaufwändig werden kann. Da der Scrum-Guide eine maximale Dauer des Plannings von acht Stunden vorsieht (Schwaber und Sutherland, 2017, S. 10), das Risikomanagement jedoch nur begrenzt sinnerhaltend beschleunigt werden kann, wäre hier die einzig verbleibende Stellschraube die

Länge der Sprints, die nach unten korrigiert werden müsste. Auf diese Weise verringert sich der Umfang der Planning-Inhalte, so dass die Zeitbeschränkung auch mit den zusätzlichen Arbeitsschritten eingehalten werden kann.

Als die beschriebenen Prozesserweiterungen einen diskutierbaren Stand erreicht hatten, war die Entwicklung im Projekt bereits in eine Phase eingetreten, in der zum einen Komponenten nicht mehr von Grund auf neu, sondern „nur“ noch weiterentwickelt wurden, und der nahende Termin des Livegangs experimentelle Änderungen am Prozess verbot. Eine Bewährung des hier beschriebenen in der Praxis steht daher noch aus.

Darüber hinaus bedeutet die Implementation der vorgestellten Änderungen selbstverständlich keine automatische Verbesserung der Ergebnisse eines Teams. Die Prozesserweiterungen sind keineswegs voraussetzungslos: Das Team muss allem voran Akzeptanz für sie zeigen, da sonst Modellierung und Risikomanagement höchstwahrscheinlich nur halbherzig durchgeführt werden. Auch sollten sich mindestens ein, besser aber zwei oder mehr Mitglieder berufen fühlen, die mit der Rolle des Security Champions verbundenen Verantwortlichkeiten übernehmen. Wenn nicht von vornherein unter den Mitgliedern ein gewisses Bewusstsein und Interesse für IT-Sicherheit vorherrscht, ist dies keine Selbstverständlichkeit.

Auch löst das beschriebene Vorgehen nicht alle denkbaren Probleme: Der Fokus liegt mit priorisierter Behandlung der größten Risiken bereits, der Pareto-Verteilung folgend, darauf, in 20% der für eine vollumfängliche Absicherung gegen alle auffindbaren Risiken nötigen Zeit die dringendsten 80% davon zu beheben, weshalb das Verfahren so stark auf die Trust Boundaries abzielt. Dass in der Menge niedrig priorisierter, nicht adressierter Risiken keine relevanten Schwachstellen mehr vorhanden sind, ist jedoch selbstverständlich ebenfalls kein Naturgesetz. Auch haben Datenflussdiagramme ihre Grenzen: Sie können beispielsweise überhaupt nicht darstellen, ob das beschriebene RBAC-Modell im Verlauf der Entwicklung ausgehöhlt oder unterlaufen wird, indem z.B. aus Bequemlichkeit übermäßig viele Rechte in einer einzelnen Rolle akkumuliert werden oder viel zu viele Accounts Administratorrechte erhalten. Überdies besteht die Gefahr, mit der Umsetzung einer technischen Gegenmaßnahme weitere Schwachstellen zu erzeugen. Ein entsprechendes Beispiel habe ich in 4.4.3 beschrieben. Ein solcher Vorgang muss vom Team rechtzeitig, am besten noch beim Planen der Gegenmaßnahme, bemerkt werden: Zum Zeitpunkt der Implementation von Gegenmaßnahmen ist die Bedrohungsanalyse der fraglichen Komponente bereits abgeschlossen. Formell lässt sich dies nur abfangen, indem nach Planung der Maßnahmen ein erneuter Zyklus der Risikobewertung durchlaufen wird. Was dies für Akzeptanz und Zeitaufwand bedeutet, muss die Praxis zeigen.

In einer „perfekten Welt“, in der jede*r Entwickler*in stets aufmerksam und sich aller möglichen, relevanten Angriffe bewusst ist sowie in der sich agile Teams ideal selbst organisieren,

sind die dargestellten Prozesserweiterungen darüber hinaus selbstverständlich nicht notwendig. Wenn ein Team von sich aus gute Ergebnisse liefert, das schon längere Zeit zusammen arbeitet und hinreichend eingespielt ist, besteht nur begrenzter Anlass, den Prozess wie dargestellt auszubauen. Gleichwohl sind, wie beschrieben, Verhältnisse wie Personalkonstellationen heutzutage selten lange stabil. Da IT-Sicherheitsvorfälle von teuren Haftungsfragen begleitet werden können, kann es sinnvoll sein, auch ohne das Auftreten konkreter Probleme in Erwägung zu ziehen, das Entwicklungsmodell auf den Prüfstand zu stellen.

Insgesamt beurteile ich den Versuch, etablierte Best Practices der IT-Sicherheit und agile Softwareentwicklung nach Scrum miteinander zu vereinen, als zumindest theoretisch gelungen. Die in den Prozess eingeflossenen Techniken wie z.B. STRIDE haben sich allesamt jeweils bereits in der Praxis bewiesen und sind gut dokumentiert. Von ihrer Kombination und zeitlichen Anordnung im Prozess erwarte ich bei ernsthafter Anwendung eine deutlich verringerte Zahl von Sicherheitsvorfällen. Der Erfolg wird jedoch schwer zu messen sein: Erstens sind A/B-Tests nicht möglich, da man schlecht die selbe Anwendung mehrfach entwickeln lassen kann. Zweitens sind Softwareprojekte aufgrund vieler Eigenschaften selten sinnvoll miteinander zu vergleichen.

5.3. Ausblick

Die vorgeschlagenen Prozesserweiterungen könnten freilich noch weiter entwickelt werden. So ließe sich formalisieren, wie Komponenten ähnlichen Verhaltens modellhaft zueinander in Bezug gesetzt werden können. Bisher gibt das Verfahren selbst jedenfalls keinen Anlass, beispielsweise bei der Entwicklung zweier Webservices auf jeden Fall nur ein mal die Folge von Modellierung, Bedrohungs- und Risikoanalyse & Evil User Stories zu durchlaufen. Sinnvollster Ansatzpunkt wäre hierfür bereits das Modell: Wenn schon bei dessen Erstellung festgestellt werden könnte, dass ähnliche Fälle bereits abgehandelt wurden, ließen sich die folgenden Schritte ggf. einsparen. Hier könnte man weiterdenken, ob und wie die Modelle in Graphen zu überführen und Ähnlichkeitsberechnungen auf ihnen anzustellen wären. Zu ähnlichen Zwecken existieren bereits mathematische Grundlagen in Form von Graphgrammatiken und Ersetzungssystemen. Dies übersteigt jedoch den Rahmen dieser Arbeit bei weitem.

Des Weiteren wäre aber auch auf einer niedrigeren Abstraktionsebene eine Softwareunterstützung für die beschriebenen Prozessschritte denkbar. Ein Datenflussdiagramm von begrenztem Umfang lässt sich zwar bereits mit heute verfügbaren Tools in vertretbarer Zeit erstellen, doch um hieran STRIDE durchzuführen und dessen Ergebnisse zu bewerten, müssen bislang

klassisch Stift & Papier oder Excel herhalten. Dabei müssen im Verlauf des Vorgangs für jede identifizierte Bedrohung jeweils zwei Durchschnitte aus je acht Zahlen ermittelt werden, was aufgrund menschlicher Faktoren wiederum zeitaufwändig und fehleranfällig ist. Eine Software könnte dies unterstützen, indem sie z.B. aus einem Diagramm automatisch alle Kommunikationsflüsse für STRIDE extrahiert, Eingabemasken für die Einschätzungen generiert und schließlich die Durchschnitte ermittelt.

Aus der Organisation des untersuchten Vorgehensmodells ergibt sich außerdem, dass die Ergebnisse dieser Arbeit zum Teil recht spezifisch auf Scrum ausgerichtet sind. Um anderen Prozessmodellen gerecht zu werden, sind hier ggf. Anpassungen nötig. Bei Kanban beispielsweise sind Spezialistenrollen wie die des Security Champions unproblematisch, was es vereinfacht, nah am Vorschlag des OWASP zu bleiben. Iterationen bzw. Sprints hingegen sind optional, so dass ohne sie ein Team individuell entscheiden muss, wann das Risikomanagement betrieben werden soll.

A. Anhang

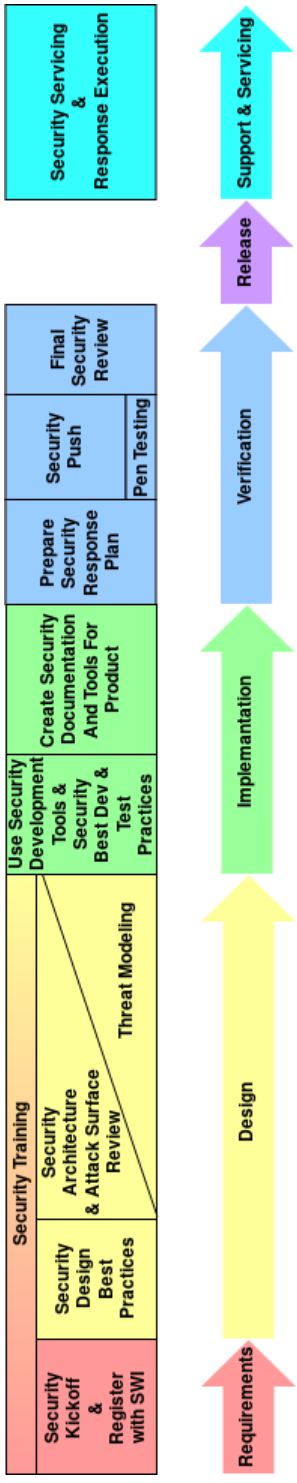


Abbildung A.1.: Die Erweiterungen des Microsoftprozesses nach SDL
(Howard und Lipner, 2005, Nachbildung der Originalgrafik zur Verbesserung der optischen Qualität)

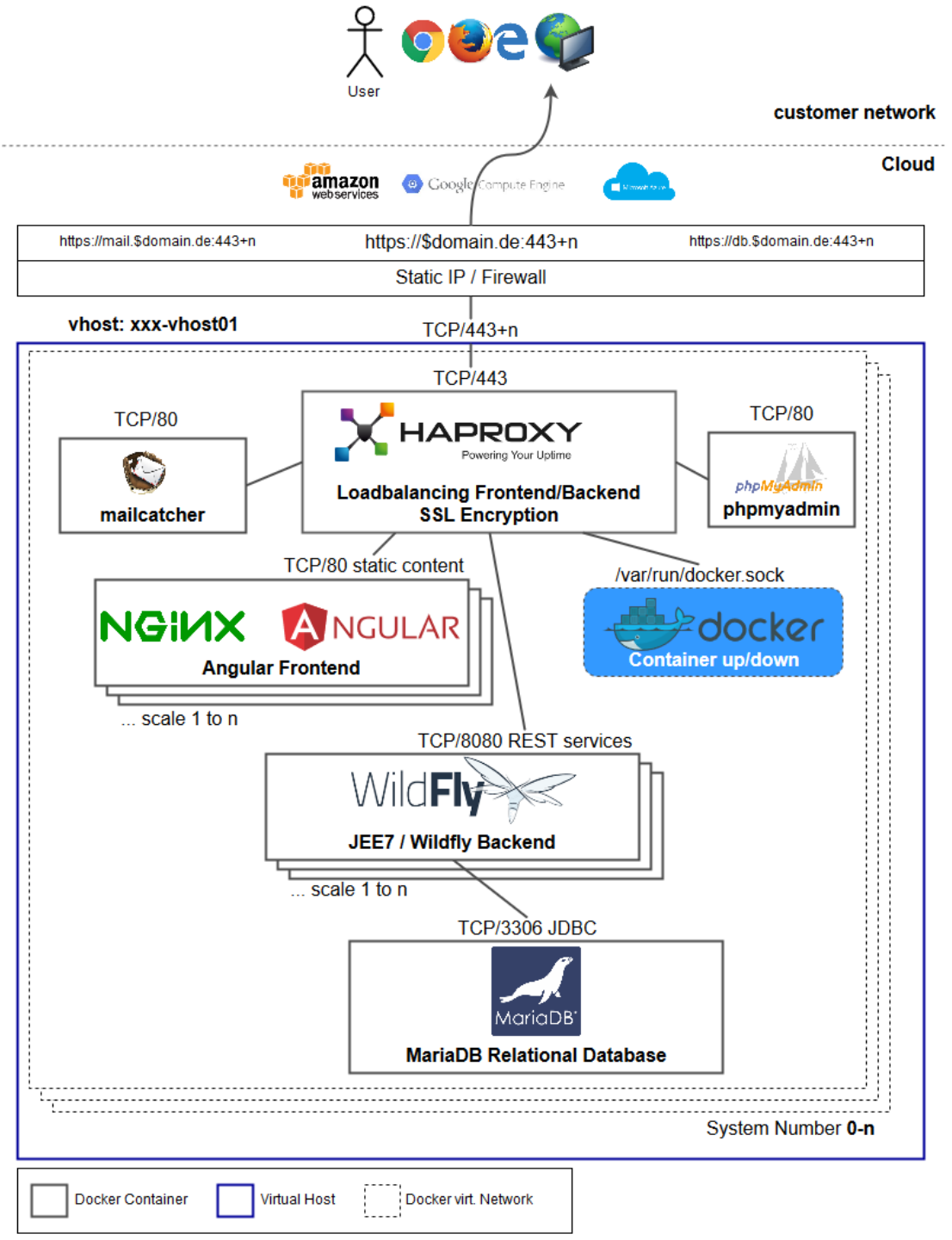


Abbildung A.2.: Übersicht über die Anwendung: Infrastrukturdiagramm

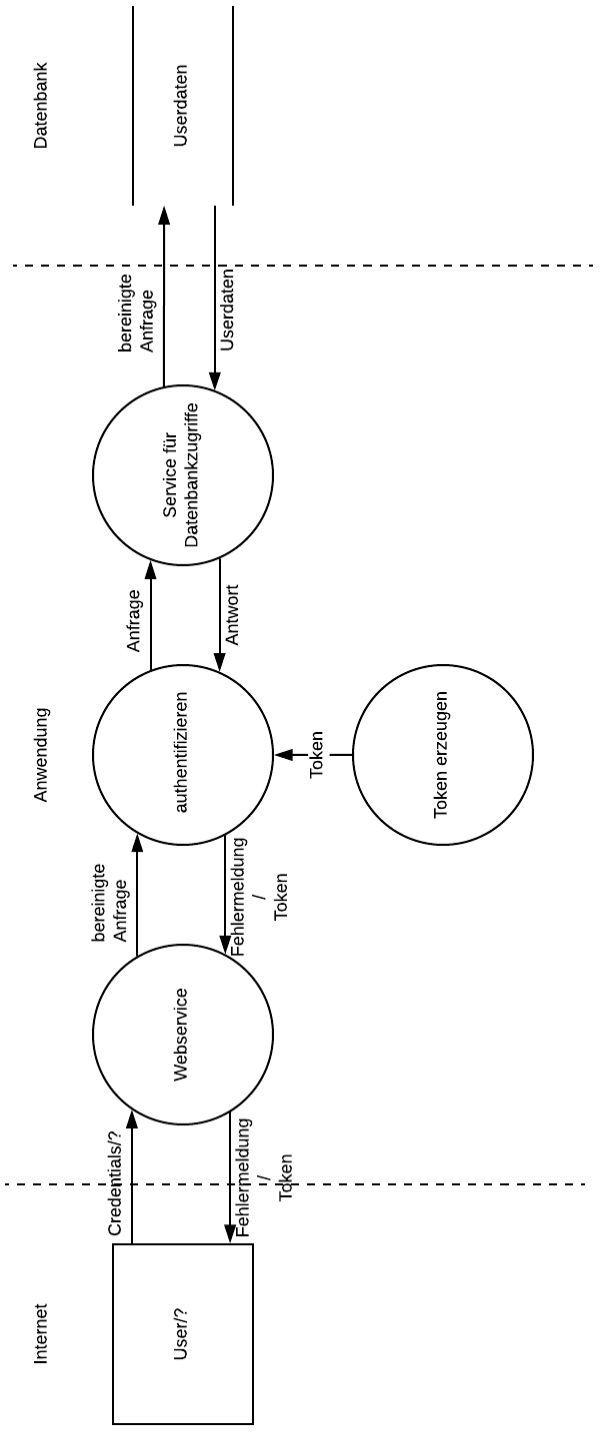


Abbildung A.3.: Datenflüsse beim Login, Level 1

Literaturverzeichnis

[Amtsblatt der Europäischen Union 2016] Amtsblatt der Europäischen Union: *Datenschutz-Grundverordnung*. 2016. – URL https://eur-lex.europa.eu/legal-content/DE/TXT/PDF/?uri=CELEX:32016R0679. – Zugriffsdatum: 2018-05-28

[Angular Documentation 2018] Angular Documentation: *Angular Documentation*. 2018. – URL https://angular.io/guide/. – Zugriffsdatum: 2018-07-30

[Beck u. a.] Beck, Kent ; Beedle, Mike ; Bennekum, Arie van ; Cockburn, Alistair ; Cunningham, Ward ; Fowler, Martin ; Grenning, James ; Highsmith, Jim ; Hunt, Andrew ; Jeffries, Ron ; Kern, Jon ; Marick, Brian ; Martin, Robert C. ; Mellor, Steve ; Schwaber, Ken ; Sutherland, Jeff ; Thomas, Dave: *Principles of the Agile Manifesto*. – URL http://agilemanifesto.org/principles.html. – Zugriffsdatum: 2018-08-18

[Bodleian-Libraries 2018] Bodleian-Libraries: *e-Books: Citing e-books*. 2018. – URL https://libguides.bodleian.ox.ac.uk/e-books/citing. – Zugriffsdatum: 01.06.2018

[BSI 2016] BSI: *IT-Grundschutz-Kataloge*. 2016. – URL https://www.bsi.bund.de/DE/Themen/ITGrundschutz/ITGrundschutzKataloge/Inhalt/_content/bezug/bezug.html. – Zugriffsdatum: 2018-06-09

[bsi.bund.de] bsi.bund.de: *bsi.bund.de: Cybersicherheit*. – URL https://www.bsi.bund.de/DE/Themen/Cyber-Sicherheit/cyber-sicherheit_node.html

[Bundesministerium der Justiz und für Verbraucherschutz 2007] Bundesministerium der Justiz und für Verbraucherschutz: *Telemediengesetz*. 2007. – URL https://www.gesetze-im-internet.de/tmg/__13.html. – Zugriffsdatum: 2018-06-18

[CCC 2018] CCC: *Besonderes elektronisches Anwaltspostfach muss Freie Software werden.* 2018. – URL https://www.ccc.de/de/updates/2018/bea. – Zugriffsdatum: 12.05.2018

[Chandra 2017] Chandra, Pravir ; Arciniegas, Fabio (Hrsg.) ; Bartoldus, Matt (Hrsg.) ; Deleersnyder, Sebastien (Hrsg.) ; Carter, Jonathan (Hrsg.) ; Challey, Darren (Hrsg.) ; Chess, Brian (Hrsg.) ; Cruz, Dinis (Hrsg.) ; Derry, Justin (Hrsg.) ; Win, Bart D. (Hrsg.) ; McGovern, James (Hrsg.) ; Meucci, Matteo (Hrsg.) ; Payne, Jeff (Hrsg.) ; Peterson, Gunnar (Hrsg.) ; Piper, Jeff (Hrsg.) ; Steingruebl, Andy (Hrsg.) ; Steven, John (Hrsg.) ; Thunberg, Chad (Hrsg.) ; Watson, Colin (Hrsg.) ; Williams, Jeff (Hrsg.): *Software Assurance Maturity Model 1.5.* 2017. – URL https://opensamm.org/downloads/SAMM-1.0.pdf. – Zugriffsdatum: 2018-08-12

[Cohn 2004] Cohn, Mike: *User Stories Applied: For Agile Software Development.* 2004. – ISBN 0-321-20568-5

[Deutscher Bundestag 1998] Deutscher Bundestag: *Gesetz zur Kontrolle und Transparenz im Unternehmensbereich (KonTraG).* 1998. – URL https://www.bgbl.de/xaver/bgbl/text.xav?SID=&tf=xaver.component.Text_0&tocf=&qmf=&hlf=xaver.component.Hitlist_0&bk=bgbl&start=%2F%2F*%5B%40node_id%3D%27279779%27%5D&skin=pdf&tlevel=-2&nohist=1. – Zugriffsdatum: 2018-06-18

[Eckert 2014] Eckert, Claudia: *IT-Sicherheit: Konzepte - Verfahren - Protokolle.* De Gruyter, 2014 (De Gruyter Studium). – URL https://books.google.de/books?id=NUzoBQAAQBAJ. – ISBN 9783110399103

[heise 2017] heise, Jürgen Schmidt: *PC-Wahl: CCC demonstriert erneut einen Angriff und bietet Open-Source-Hilfe.* 2017. – URL https://www.heise.de/security/meldung/PC-Wahl-CCC-demonstriert-erneut-einen-Angriff-und-bietet-Open-Source-H
html. – Zugriffsdatum: 12.05.2018

[Howard und Lipner 2005] Howard, Micheal ; Lipner, Steve: *Entwicklungszyklus für sichere Software.* 2005. – URL https://msdn.microsoft.com/de-de/library/ms995349.aspx. – Zugriffsdatum: 27.05.2018

[Howard und Lipner 2006] Howard, Micheal ; Lipner, Steve: *The Security Development Lifecycle.* Microsoft, 2006. – URL https://blogs.msdn.microsoft.com/microsoft_press/2016/04/19/

free-ebook-the-security-development-lifecycle/. – Zugriffsdatum: 28.05.2018

[https://cve.mitre.org 2018] HTTPS://CVE.MITRE.ORG: *Search for „angular“ on https://cve.mitre.org*. 2018. – URL https://cve.mitre.org/cgi-bin/cvekey.cgi?keyword=angular. – Zugriffsdatum: 2018-08-06

[International Organization for Standardization 2008] INTERNATIONAL ORGANIZATION FOR STANDARDIZATION: Information technology — Security techniques — Information security risk management / International Organization for Standardization. März 2008. – Standard

[Microsoft 2008] MICROSOFT: *MSDN Forums: Do you use DREAD as it is?* 2008. – URL https://social.msdn.microsoft.com/Forums/en-US/c601e0ca-5f38-4a07-8a46-40e4adcbc293/do-you-use-dread-as-it-is?forum=sdlprocess. – Zugriffsdatum: 2018-08-27

[Microsoft 2007] MICROSOFT, David Leblanc: *DREADful*. 2007. – URL https://blogs.msdn.microsoft.com/david_leblanc/2007/08/14/dreadful/. – Zugriffsdatum: 12.05.2018

[Neumann, Tschirsich, Schröder 2017] NEUMANN, TSCHIRSICH, SCHRÖDER: *Der PC-Wahl-Hack*. 2017. – URL https://media.ccc.de/v/34c3-9247-der_pc-wahl-hack

[Nielsen 1999] NIELSEN, Jakob: *The Top 10 Web Design Mistakes of 1999*. 1999. – URL https://www.nngroup.com/articles/the-top-ten-web-design-mistakes-of-1999/. – Zugriffsdatum: 2018-06-04

[npm Documentation 2018] NPM DOCUMENTATION: *npm Documentation*. 2018. – URL https://docs.npmjs.com/. – Zugriffsdatum: 2018-08-30

[OWASP Foundation 2017] OWASP FOUNDATION, The: *OWASP Top 10 - 2017*. 2017. – URL https://www.owasp.org/images/7/72/OWASP_Top_10-2017_%28en%29.pdf.pdf. – Zugriffsdatum: 2018-07-26

[OWASP Foundation 2018a] OWASP FOUNDATION, The: *OWASP Developer Guide*. URL https://github.com/OWASP/DevGuide. – Zugriffsdatum: 2018-07-26, 2018

[OWASP Foundation 2018b] OWASP Foundation, THe: *OWASP Testing Guide Wiki*. 2018. – URL https://www.owasp.org/index.php/OWASP_Testing_Guide_v4_Table_of_Contents. – Zugriffsdatum: 2018-08-10

[owasp.org 2018] owasp.org: *Main Page*. 2018. – URL https://www.owasp.org/index.php/Main_Page. – Zugriffsdatum: 2018-07-30

[Santamaria 2015] Santamaria, Jose Maria A.: *The Single Page Interface Manifesto*. 2015. – URL http://itsnat.sourceforge.net/php/spim/spi_manifesto_en.php. – Zugriffsdatum: 2018-06-04

[Schwaber und Sutherland 2017] Schwaber, Ken ; Sutherland, Jeff: *The Scrum Guide*. 2017. – URL https://www.scrumguides.org/download.html. – Zugriffsdatum: 2018-08-09

[spiegel.de 2018] spiegel.de: *Sascha Lobo: der Debatten-Podcast Nummer 41: Albrecht, Lobo und die DSGVO*. 2018. – URL http://www.spiegel.de/netzwelt/netzpolitik/datenschutz-grundverordnung-dsgvo-sascha-lobo-vs-jan-philipp-albrecht-html. – Zugriffsdatum: 2018-07-28

[Twitter Developer Blog 2018] Twitter Developer Blog ; Piper, Andy (Hrsg.): *Upcoming changes to the developer platform*. 2018. – URL https://blog.twitter.com/developer/en_us/topics/tools/2018/upcoming-changes-to-the-developer-platform.html

[Wikimedia Commons 2007] Wikimedia Commons ; Mapto (Hrsg.): *Project-triangle-en*. 2007. – URL https://commons.wikimedia.org/wiki/File:Project-triangle-en.svg. – Zugriffsdatum: 2018-06-09

[Wikimedia Commons 2015] Wikimedia Commons ; Eatthis86 (Hrsg.): *Skizzierung des Ablaufs einer Single-Page-Webanwendung*. 2015. – URL https://commons.wikimedia.org/wiki/File:SPA_Start.png#/media/File:SPA_Start.png. – Zugriffsdatum: 2018-06-09

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 19. September 2018 Jakob Ledig