IT-Security

Entwicklung sicherer Software durch Secure Coding Guidelines

Diplomarbeit

zur Erlangung des akademischen Grades

Diplom-Ingenieur

eingereicht von

Kevin Zabrana, BSc.

161061953

im Rahmen des

Studienganges IT-Security an der Fachhochschule St. Pölten

Betreuung

Betreuer: Ing. Dipl.-Ing. Richard Thron, BSc, CISSP

St. Pölten, 28. September 2018 _____ _____

(Unterschrift Verfasser/in) (Unterschrift Betreuer/in)

*

Fachhochschule St. Pölten GmbH, Matthias Corvinus-Straße 15, A-3100 St. Pölten,T: +43 (2742) 313 228, F: +43 (2742) 313 228-339, E:office@fhstp.ac.at, I:www.fhstp.ac.at

Ehrenwörtliche Erklärung

Ich versichere, dass

- ich diese Diplomarbeit selbständig verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und mich sonst keiner unerlaubten Hilfe bedient habe.
- ich dieses Diplomarbeitsthema bisher weder im Inland noch im Ausland einem Begutachter/einer Begutachterin zur Beurteilung oder in irgendeiner Form als Prüfungsarbeit vorgelegt habe.
- diese Arbeit mit der vom Begutachter/von der Begutachterin beurteilten Arbeit übereinstimmt.

Der Studierende/Absolvent räumt der FH St. Pölten das Recht ein, die Diplomarbeit für Lehre- und Forschungstätigkeiten zu verwenden und damit zu werben (z.B. bei der Projektevernissage, in Publikationen, auf der Homepage), wobei der Absolvent als Urheber zu nennen ist. Jegliche kommerzielle Verwertung/-Nutzung bedarf einer weiteren Vereinbarung zwischen dem Studierenden/Absolventen und der FH St. Pölten.

Ort, Datum *Unterschrift*

Kurzfassung

"Es gibt keine Sicherheit, nur verschiedene Grade der Unsicherheit" - Anton Neuhäusler, Professor für Philosophie und bayerischer Mundartdichter (1919 - 1997)[1]

So wie in diesem Zitat, ist auch Software nie vollkommen sicher. Diese wird aber immer bedeutender für Unternehmen, die sich sowohl an externe Vorgaben halten müssen, als auch ihre Daten und Informationen, und damit den Wert ihres Unternehmens, adäquat schützen möchten. Es ist daher oftmals gelebte Praxis, Entwicklern eine Guideline mit auf den Weg der Entwicklung zu geben, in dem definiert ist, auf welche sicherheitstechnischen Eckpunkte achtgegeben werden muss. Doch oftmals fehlt das Know-How um solche Regeln aufstellen zu können. Aus diesem Grund beschäftigt sich diese Diplomarbeit mit den beiden Forschungsfragen:

- Welche Inhalte können von Informationssicherheitsstandards und relevanten Publikationen angewendet werden?
- Wie können Regelwerke aussehen, um Software auf einem moderaten und einem erhöhten Sicherheitsniveau zu erstellen?

Zusätzlich zu diesen Inhalten wurden die untersuchten Standards und Publikationen nach definierten Kriterien bewertet. Aufgrund dieser Kriterien wurde ein Ranking erstellt, welche Publikationen sich am besten zur Erstellung einer Secure Coding Guideline eignen.

Durch die Ergebnisse der Arbeit soll es Unternehmen erleichtert werden, eine Secure Coding Guideline zu erstellen, die aktuellen Anforderungen entspricht. Durch die Validierung der Beispielguideline durch Experten, die entweder in der Softwareentwicklung oder im Bereich der Informationssicherheit bereits Jahre lang tätig sind, wird die Qualität, die Aktualität und die Anwendbarkeit bestätigt.

Abstract

"Es gibt keine Sicherheit, nur verschiedene Grade der Unsicherheit" - Anton Neuhäusler, professor for philosophy and Bavarian dialect poet (1919 – 1997)[1]

Just like stated in this quote, software can also never be completely secure. This security however, is gaining importance for companies that must do both, abide by external requirements and protect their data and information and with that the company's value adequately. Due to this it is established practice to give a guideline to software developers that defines the safety-relevant key points of importance in a project. However, oftentimes the person responsible for this in the company is lacking technical expertise which makes it almost impossible to establish these rules. Therefore, this diploma thesis deals with the following two research questions:

- What content of information security standards and relevant publications can be applied?
- What could policies for the development of software with a moderate and increased security level look like?

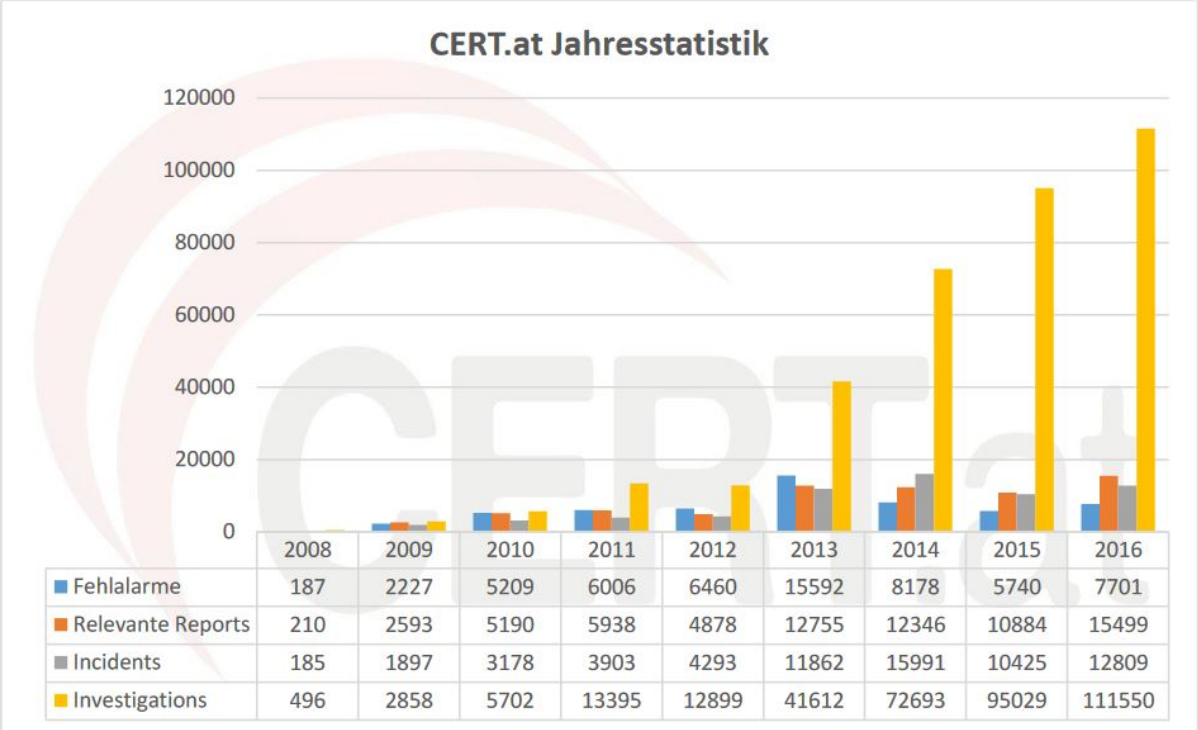
Additionally, the researched standards and publications were evaluated according to the criteria mentioned repeatedly in the literature used for research. They were then ranked based on these criteria to what extent which publication is suitable for the creation of a Secure Coding Guideline.

With the results of this thesis creating a Secure Coding Guideline that complies with the current requirements should be facilitated for companies. Through the validation of the example guideline by experts that have been working in either software development or the area of information security, the quality, topicality, and applicability of the results were ensured.

Inhaltsverzeichnis

1 Einleitung

Hackerangriffe und die Verbreitung von Schadsoftware sind Themen, die heutzutage nahezu täglich in den Medien präsent sind. Ob etwa Kreditkartendaten gestohlen oder eine Art von Verschlüsselungssoftware gefunden oder enttarnt wurde, überliest man dabei schon beinahe. Diesen Anstieg von Vorfällen bestätigt auch das österreichische Cyber Emergency Response Team Austria (cert.at) im Bericht 'Bericht Internet-Sicherheit Österreich 2016' wie in Graphik 1.1 dargestellt. Neben den Themen der Schuldzuweisung und der Haftung stellen sich aber immer mehr Informationssicherheitsverantwortliche, Geschäftsführerinnen und Geschäftsführer die Frage, wie man sich erfolgreich vor solchen Angriffen schützen kann.



	2008	2009	2010	2011	2012	2013	2014	2015	2016
Fehlalarme	187	2227	5209	6006	6460	15592	8178	5740	7701
Relevante Reports	210	2593	5190	5938	4878	12755	12346	10884	15499
Incidents	185	1897	3178	3903	4293	11862	15991	10425	12809
Investigations	496	2858	5702	13395	12899	41612	72693	95029	111550

Abbildung 1.1: Bericht Internetsicherheit 2016
[2]

Die Antwort auf diese Frage ist genauso schwierig zu geben, wie sie umfangreich sein müsste. Oftmals sind Schwachstellen in Software enthalten. Problematisch ist an dieser Stelle, dass es keinen einheitlichen Standard gibt, der zumindest Grundanforderungen an die Sicherheit von Software definiert. Dies hätte nicht nur den Vorteil, dass gegen diese Norm zertifiziert werden könnte, was eine Bescheinigung vor Externen oder vor Gericht darstellen könnte, sondern auch, dass in einer solchen Regelung Vorgehensweisen und Funktionen definiert und regelmäßig dem Stand der Technik angepasst werden können. Solche Standards sind nur in einigen Branchen vertreten, wie beispielsweise in der Automobilindustrie und Luftfahrt (ISO 26262 und DO-254).

Diese Diplomarbeit soll im Folgenden Standards und Publikationen auf ihre Anwendbarkeit zur Erstellung von sicheren Programmen und auf Aktualität überprüfen.
Außerdem sollen diese Werke verglichen und zur einfacheren Bearbeitung inhaltlich grob zusammengefasst werden.

Es wird eine Guideline definiert, die einer Entwicklerin oder einem Entwickler Anweisung geben soll, welche die Sicherheit von Software erhöht. Diese soll auf dem momentanen Stand der Technik passieren. Zur leichteren Anpassung werden die Parameter der Guideline, deren Anpassung wahrscheinlich ist, markiert. Durch die Validierung der Guideline durch Experten wird nicht nur die Aktualität sondern auch die Anwenderbarkeit gewährleistet.

Da großer Wert auf den Aufbau der Guideline gelegt wird, wird dieser separiert von der inhaltlichen Validierung von Kommunikationsexperten geprüft und verifiziert.

2 Einführung Literaturrecherche

Bei der Auswahl der analysierten Standards und Publikationen wurde darauf Wert gelegt, einen möglichsten guten Querschnitt über folgende Bereiche zu bekommen:

- Community vs. Standard
- Herkunft

So entstand die Auswahl folgender Werke:

Name	Kommerziell	Community	Standard	Herkunft
ISO 27001			X	INT
ISO 27002			X	INT
Microsoft - SDLC	X			USA
NIST Publikationen		X		USA
OWASP		X		INT
IT-Grundschutz		X		GER
CERT		X		USA
ISO 27034			X	INT
ÖNORM A7700			X	AUT
Java Secure Coding Guideline		X		INT
Summe	1	5	4	

Tabelle 2.1: Verteilung der analysierten Standards und Publikationen

Es wurde Wert darauf gelegt, Publikationen aus europäischen Ländern und Amerika zu bearbeiten, da möglicherweise unterschiedliche Ansätze zum Secure Coding verfolgt werden könnten. Weiters sind dort die momentan führenden Normungskremien und Communities rund um das Thema ‘Secure Coding’beheimatet. Da Hersteller oft nur Sicherheitsvorgaben zu eigenen Produkten liefern und meist keine allgemeinen Ratschläge zur Verfügung stellen, wurden diese außen vor gelassen. Aufgrund des hohen

Bekanntheitsgrades wurde der Microsoft SDLC dennoch in die Analyse mit aufgenommen.

Da die Welt der Informationstechnologie sehr schnelllebig ist und sich laufend verändert, werden nur Werke miteinbezogen und untersucht, die nicht älter als zehn Jahre sind, also deren Erscheinungsdatum nach 2008 liegt.

3 Bewertung der Publikationen

Ziel dieser Arbeit ist es unter anderem Standards, Publikationen und Ähnliches zu finden, welche beim Erstellen von Secure Coding Guidelines unterstützen können.
Grundlage für die Bewertung liefert außerdem eine Aufstellung von Kriterien, die aus dem OWASP Secure Coding Guide herausgelöst wurden. Auch wenn sich die Publikation auf Webentwicklung fokusiert, so sind die Risiken oft ähnlich. Ebenso werden Server-Client Applikationen oft durch einen Webserver mit Datenversorgt, dadurch wird sogar die selbe Schnittstelle verwendet um Informationen bereitzustellen. Deshalb können die Bestandteile zur Bewertung herangezogen werden.[3]:

- Logging
- Authentifizierung und Autorisierung
- Kryptographie
- Session Managemen
- Input Validation
- Output Escaping
- Datenbank Security
- Fehler Management - (Exception Handling)

Jeder der Teilbereiche wird mit Punkten von null bis vier bewertet:

- Null - Wird nicht behandelt
- Eins - Allgemeine Informationen zum Thema vorhanden - keine Anforderungen definiert
- Zwei - Teilweise definierte Anforderungen
- Drei - Genau definierte Anforderungen, aber keine Beispiele
- Vier - Exakt definierte Umsetzungsempfehlungen mit Code-Beispielen

Für das Alter der Publikation ergibt sich folgende Bewertung:

- Eins - acht bis zehn Jahre
- Zwei - sechs bis acht Jahre
- Drei - drei bis fünf Jahre
- Vier - ein bis zwei Jahre

Jeder Bereich wird mit einem Bemerkungsfeld versehen, in welchem bei Bedarf Anmerkungen festgehalten werden können.

Beispielskala:

Die in der Skala verwendeten Werte sind frei erfunden und dienen nur der Veranschaulichung des Bewertungssystems.

Bereich	Bewertung	Bemerkung
Logging	4	Exakte Definitionen mit Beispielen
Authentifizierung und Autorisierung	0	
Kryptographie	2	Integriert im Kapitel 'Technische Vorkehrungen'
Session Management	2	Integriert im Kapitel 'Technische Vorkehrungen
Input Validation	1	Nur allgemeine Ratschläge
Output Escaping	0	
Datenbank Sicherheit	0	
Fehler Management	2	Integriert im Kapitel 'Technische Vorkehrungen
Aktualität	4	Das Dokument wird laufend überarbeitet
Summe:	15	

Tabelle 3.1: Beispielskala

Am Ende der Bewertung werden die Punkte summiert - dadurch wird ein Ranking der Publikationen ermöglicht. Im Folgenden werden auch die Inhalte der bearbeiteten Publikation kurz wiedergegeben.

4 Standards

4.1 ISO 27001

Die ISO 27001 wurde von der Internationalen Normungsorganisation (International Standard Organisation) entwickelt, um Anforderungen für die Einrichtung, Implementierung, Wartung und laufende Verbesserung eines Informationssicherheitsmanagementsystems (ISMS) festzulegen. Ein Informationssicherheitsmanagementsystem schützt die Vertraulichkeit, die Unversehrtheit und die Verfügbarkeit von Informationen und sorgt durch einen Risikomanagementprozess dafür, dass Risiken erkannt und gemanagt werden. Da die Anforderungen, die in der Norm festgeschrieben sind, allgemein gehalten werden, sind diese für alle Organisationen, ungeachtet deren Größe, anwendbar.[4, pp.5-6]

4.1.1 Inhalt

Die Norm unterteilt sich in sechs Kapitel (Kapitel 5-10) die zur Anwendung kommen. Weitere Umsetzungspunkte finden sich im Annex A des Dokumentes. Diese werden jedoch im Kapitel 4.1.2 analysiert. Die ISO 27001 stellt Forderungen an ein Managementsystem und dessen Umfeld. Es wird definiert, welche Aufgaben die Unternehmensführung im Bezug auf Informationssicherheit wahrnehmen muss und was gewährleistet sein muss, damit ein solches System funktionieren kann. So wird beispielsweise definiert, dass das Informationssicherheitsmanagementsystem einem kontinuierlichen Verbesserungsprozess unterliegen muss. Auch die Bereitstellung von Ressourcen und die Zielsetzung sind in der Publikation genau definiert. Weiters wird der Betrieb eines funktionierenden und angemessenen Risikomanagements gefordert. Dies ist nötig, um Risiken auf passende Weise begegnen zu können und die Gefährdung des Unternehmens zu kennen. Aus den Ergebnissen können Maßnahmen durch die Unternehmensführung abgeleitet und umgesetzt werden. Genauso sind eine Abwälzung, Akzeptanz oder eine Vermeidung des Risikos denkbar.[4]

Da die Norm auf das Security Management und den Betrieb eines Informationssicherheitsmanagementsystems ausgelegt ist, sind keine Anforderungen in sichere Software oder Vorgaben für Programmierer selbiger enthalten. Ebenso finden sich keine Verweise auf Publikationen in denen solche Regelungen zu

finden sind.

4.1.2 Annex A

Im Annex A der ISO 27001 werden Ziele und Maßnahmen genannt, die einen Teil des Informationssicherheitsprozesses nach 6.1.3 der Norm darstellen. Dieser Teil der Norm wird von Unternehmen verwendet um ein Statement of Applicability (eine Anwendbarkeitserklärung) zu erstellen. Da diese Inhalte aber in der ISO 27002 näher erläutert werden, wird an dieser Stelle nicht näher darauf eingegangen sondern auf das Kapitel 4.2 verwiesen. Diese Punkte werden aus dem selben Grund auch nicht in der Bewertung berücksichtigt.[4, p. 17]

4.1.3 Bewertung

Bereich	Bewertung	Bemerkung
Logging	0	
Authentifizierung und Autorisierung	0	
Kryptographie	0	
Session Management	0	
Input Validation	0	
Output Escaping	0	
Datenbank Sicherheit	0	
Fehler Management	0	
Aktualität	3	Aktuelle Fassung aus 2013
Summe:	3	

Tabelle 4.1: Beispielskala

4.2 ISO 27002

Die ISO 27002 gehört genauso wie die im Kapitel 4.1 erwähnte ISO 27001 zum Regelwerk der ISO zum Thema Informationssicherheit. Die ISO 27002 bezieht sich auf die Controls im Annex A der ISO 27001 und ist daher als eine Guideline zur Umsetzung der Forderungen der ISO 27001 zu sehen. Da die Norm sehr umfangreich ist werden im Folgenden nur Thematiken behandelt, die auf sichere Softwareentwicklung anwendbar sind.

4.2.1 Inhalt

Die ISO 27002 ist ebenso wie die ISO 27001 auf ein Informationssicherheitsmanagement zugeschnitten. Da jedoch die Anforderungen genauer ausgeführt sind, lassen sich daraus Anforderungen an sichere Software ableiten.

Access Control

Die Norm fordert, dass gewährleistet sein muss, dass jede Benutzerin und jeder Benutzer nur Zugriff auf jene Daten hat, für die er auch autorisiert ist. Dieses Zugriffssystem muss rollenbasiert erfolgen. Diese Anforderungen müssen bereits in der Entwicklung von Applikationen berücksichtigt werden. Jeder User muss mit einer ID versehen sein. Das ermöglicht das Nachvollziehen der Aktionen. Diese ID muss systemweit einzigartig sein. Außerdem muss die Möglichkeit geschaffen werden, dass Benutzerkonten deaktiviert werden. Wenn die Authentifizierung über ein Passwort erfolgt, muss eine Möglichkeit bestehen, dem Benutzer ein Initialpasswort über eine sichere Verbindung mitzuteilen. Dieses Passwort muss bei der ersten Anmeldung geändert werden.[5, pp.19-28]

Die Benutzerin oder der Benutzer muss die Möglichkeit haben, das Passwort jederzeit zu ändern. Dabei soll nur ein Passwort gewählt werden können, das folgenden Anforderungen entspricht[5, pp.19-28]:

- ausreichende Länge
- einfach zu merken
- nicht anfällig für Wörterbuchangriffe
- nicht auf einfach zu erlangenden Informationen basierend (Telefonnummer, Geburtsdatum ,...)

Außerdem sollten, wenn vorhanden, Single-Sign-On Lösungen in Applikationen integriert werden. Dadurch werden sowohl die Sicherheit als auch die Verwendbarkeit gesteigert. Wenn die angezeigten und

verarbeiteten Daten von hoher Wichtigkeit oder hohem Wert sind, wird empfohlen, auf alternative Authentifizierungsmaßnahmen zu Passwörter zurückzugreifen. Dabei empfehlen sich Biometrie, Token und SmartCards.[5, pp.19-28]

Es ist außerdem wichtig, auch den Anmeldeprozess sicher in einer Applikation zu implementieren. Es dürfen beispielsweise nur generische Fehlermeldungen an einen Client übermittelt werden, da exakte Fehlerbeschreibungen oder Stack-Traces bereits wichtige Informationen für Angreifer enthalten können. Passwörter dürfen weder angezeigt werden können, noch dürfen sie unverschlüsselt übertragen werden. Außerdem muss auch im Bereich Session Management auf Sicherheit geachtet werden. Es ist wichtig, dass Sessions, die inaktiv sind - also nicht mehr aktiv verwendet werden - automatisch geschlossen werden. Außerdem ist es ratsam, dass Benutzerinnen und Benutzer nur zu bestimmten Zeiten Zugriff auf Systeme haben und nicht Rund um die Uhr. Dadurch können Angriffe auf Services eingeschränkt oder unterbunden werden.[5, pp.19-28]

Cryptography

Kryptographie beinhaltet nicht nur Chancen sondern auch Risiken. Daher ist es sinnvoll, dem Einsatz von kryptographischen Methoden mit Risikoanalysen zu begegnen. Dadurch wird erfasst, welche Risiken sich aus der Anwendung von Verschlüsselungstechnik ergeben, und welche Risiken damit behandelt werden. So ist besonders bei vertraulichen Informationen - wie beispielsweise Passwörter - auf eine starke Verschlüsselung Wert zu legen. Weiters sollte man folgende Einsatzmöglichkeiten bedenken:[5, p.23][5, pp.29-30]

- Vertraulichkeit - Zur Wahrung der Vertraulichkeit von Daten sollten Verschlüsselungsalgorithmen eingesetzt werden
- Integrität - Mittels Signatur und Prüfsummen kann die Integrität von Datensätzen gewährleistet werden
- Beweisbarkeit - Durch den Einsatz von kryptographischen Methoden können Ereignisse nachgewiesen werden.

4.2.2 Environmental Security

Logging

Logging ist wichtig, um Fehler im System nachzuvollziehen, beziehungsweise um Spuren von Angriffen zu finden und so den entstandenen Schaden richtig einzuschätzen, beziehungsweise den Angriffs-

punkt ausfindig zu machen. Die ISO 27002 schlägt vor, mindestens folgende Ereignisse in Logs zu dokumentieren:[5, pp.43-44]

- Benutzer ID
- System Aktivitäten
- Zeitpunkt von wichtigen Ereignissen, wie beispielsweise Anmeldung und Abmeldung
- Geräte ID und System Identifier
- Erfolgreiche und erfolglose Anmeldeversuche
- Erfolgreiche und erfolglose Datenzugriffe
- Veränderungen an der Systemkonfiguration
- Administrative Eingriffe
- Verwendung von Systemfunktionen
- Datei-Zugriffe
- Netzwerkadressen und Protokolle

Da Logs sensible Daten enthalten können müssen diese adäquat geschützt werden. Sie müssen daher sowohl vor Veränderung, als auch vor Löschung geschützt werden. Es soll auch für Administratoren nicht möglich sein, das Logging zu deaktivieren um die eigenen Aktivitäten zu verschleiern.[5, pp.44-45]

4.2.3 Bewertung

Bereich	Bewertung	Bemerkung
Logging	3	Genaue Definitionen ohne Beispiele
Authentifizierung und Autorisierung	3	Alternative Authentifizierungsmethoden empfohlen
Kryptographie	1	Keine exakten Protokolle oder Verfahren empfohlen
Session Management	3	Ausführliche Definition ohne Beispiele
Input Validation	1	Vereinzelte Vorgaben
Output Escaping	0	
Datenbank Sicherheit	0	
Fehler Management	1	Vereinzelte Vorgaben
Aktualität	3	Letzte Fassung 2013
Summe:	15	

Tabelle 4.2: Bewertung ISO 27002

4.3 ISO 27034

Die ISO 27034 ist ein Standard zum Thema ‘Application Security’ und besteht aus sechs Teilen. Der erste und älteste Teil existiert bereits seit 2011. Andere Teile folgten erst im Laufe der Jahre. Im Folgenden werden die Teile eins, zwei und sechs betrachtet und auf Anforderungen und Inhalte zum Erstellen einer Secure Coding Guideline eingegangen.

4.3.1 ISO 27034-1 - Overview and concepts

Ablauf und Aufbau

Dieser Teil des Standards ist stark an die ISO 27005 zum Thema Risikomanagement angelehnt. Sie ähneln sich in Aufbau und Herangehensweise. Teilweise finden sich sogar Vergleiche.

Ein wichtiges Statement der Norm ist, dass der Security Scope einer Applikation viel umfassender gestaltet sein muss, als der Scope der Applikation selbst. Das bedeutet, dass nicht nur Daten und Spezifikation berücksichtigt werden müssen, wie das für einen Programmierer oft der Fall sein kann, sondern auch Anforderungen aus dem Unternehmen, Ergebnisse aus Business Impact Analysen, technologische Grundlagen sowie Gesetze und Verträge die eingearbeitet werden müssen. Außerdem können noch Einflüsse aus einem Informationssicherheitsmanagementsystem wie beispielsweise Verwundbarkeitsanalysen oder dergleichen hinzukommen. Aus all diesen Faktoren fordert die Norm die Erstellung von Security Anforderungen und die Definition eines Kontextes, in welchem die Applikation betrieben werden soll.[6, pp. 6-8]
Aus diesen Gegebenheiten soll nun eine Risikoanalyse durchgeführt werden. Dadurch soll ein Niveau definiert werden, welchem die Applikation entsprechen muss. Dadurch wird gewährleistet, dass die Business Needs eingehalten werden und die Applikation allen Vorgaben genügt. Vor und im Betrieb fordert die Norm regelmäßige Überprüfungen, ob das geforderte Niveau eingehalten wird und wo es eventuell Verbesserungspotenzial gibt. Diese Faktoren fließen erneut in die Definition von Kontext und Security Anforderungen für Anpassungen und neue Applikationen mit ein, was den Kreis schließt und das System am Leben erhält.[6, pp. 10-14]

Organizational Normative Framework

Das Organizational Normative Framework - kurz ONF - ist die Grundlage für das Application Security Management. Es beinhaltet alle Security Best Practices, die im Unternehmen gelebt werden, sowie der Kontext aus Sicht des Unternehmens, Stand der Technologien, Verantwortlichkeiten und dergleichen.

Dieses Framework ist innerhalb eines Unternehmens einzigartig. Es kann für alle Softwareprojekte angewendet werden und muss laufend aktuell gehalten werden. Ein Herzstück des ONF stellt die ASC - Application Security Controls - Library dar. Darin enthalten sind Kontrollen, die im Unternehmen eingesetzt werden. Diese können aus vielen unterschiedlichen Quellen kommen (beispielsweise NIST 800-53 oder ISO 15408-3) und die Sicherheit einer Applikation überprüfen. Die Kontrollen sind in Sets organisiert, welche ein Sicherheitsniveau abbilden. Dadurch kann einfach ein Set herangezogen werden, um das Niveau einer Applikation zu definieren.[6, pp. 15-22]

Zu einzelnen Schritten wie der Definition von Anforderungen oder dem Durchführen von Überprüfungen und Audits stellt die Norm detaillierte Beschreibungen zur Verfügung, auf welche aber in diesem Zuge - aufgrund der Ähnlichkeit zu anderen Normen - nicht genau eingegangen wird.

4.3.2 ISO 27034-2 - Organizational normative framework

In diesem Teil der Norm wird definiert, wie ein ONF entstehen soll. Es wird geraten, alle Beteiligten im Prozess der Erstellung des ONF mittels eines RACI-Charts zu visualisieren. Es müssen die Verantwortlichkeiten definiert und die Arbeitsaufteilung exakt festgelegt werden. Weiters müssen bei der Erstellung des ONF sehr viele Teile des Unternehmens, wie beispielsweise Risikomanagement oder Datenschutzabteilung, miteinbezogen werden. Alle Inputs werden im ONF gesammelt und gruppiert. Es wird ein Zeitpunkt definiert, zu dem das ONF produktiv verwendet wird und für alle zukünftigen Applikationsprojekte Verwendung finden kann. Wichtig ist, dass dieses Framework immer am aktuellen Stand gehalten wird. So ist es bedeutend für die Funktion, dass jedes Element im ONF einen Besitzer hat, der für die Aktualität und Korrektheit Sorge trägt. Der Lebenszyklus des ONF unterliegt dem PDCA Zyklus. Nach der erfolgreichen Einführung wird weiter gemonitort, ob die Funktionsweise noch weiterhin gegeben ist. Sollte es dabei zu Abweichungen kommen, müssen Schritte unternommen werden, um die Funktion wiederherzustellen oder zu verbessern. Genauso schreibt die Norm vor, in regelmäßigen Abständen das ONF von Internen oder Externen auf die Wirksamkeit überprüfen zu lassen. Die Norm beschreibt, dass das ONF aus zumindest folgenden Punkten bestehen muss. Es kann natürlich beliebig erweitert werden[7, pp. 1-15]:

- Business context
- Regulatory context
- Technological context
- Application specification repository

- Roles, responsibilities and qualifications
- Organization ASC Library
- ONF and application security processes
- Application security lifecycle reference model
- Application security lifecycle model

4.3.3 Bewertung

Bereich	Bewertung	Bemerkung
Logging	0	
Authentifizierung und Autorisierung	0	
Kryptographie	0	
Session Management	0	
Input Validation	0	
Output Escaping	0	
Datenbank Sicherheit	0	
Fehler Management	0	
Aktualität	2	Aktuelle Fassung aus 2011
Summe:	2	

Tabelle 4.3: Bewertung ISO 27034

4.4 ÖNORM A7700

Die ÖNORM A 7700 wurde in den Jahren 2004 und 2005 vom Österreichischen Normungsinstitut, SEC Consult, Großbanken, Versicherungen, Behörden und Industrieunternehmen entwickelt. Auf den zwölf Seiten des Dokuments finden sich Sicherheitsvorgaben für die Entwicklung von Webapplikationen.[8]

4.4.1 Inhalt

Die Norm hat es zum Ziel, den Stand der Technik von sicheren Webapplikationen widerzuspiegeln und Themen, die durch Normen der 27000-Reihe nur gestreift werden, exakter zu definieren.[8]

Authentifizierung und Autorisierung

Die Norm schreibt vor - wenn die Anforderung besteht - geeignete Vorkehrungen einzusetzen um Benutzer eindeutig voneinander zu unterscheiden. Dies ist nötig, um entsprechend die Autorisierung der Benutzer vornehmen zu können. Es muss sichergestellt werden, dass sich eine Benutzerin oder ein Benutzer eindeutig mit einer vertraulichen Komponente identifiziert. Dazu kann beispielsweise ein Passwort verwendet werden, das ausreichend gegen Brute-Force Attacken geschützt ist und außerdem durch kryptografische Maßnahmen (Verschlüsselung und/oder Hash-Funktionen) geschützt ist. Außerdem muss die Webapplikation im Stande sein, eine Passwortpolicy des Unternehmens umzusetzen. Alle Funktionen, die eine Authentisierung erfordert, dürfen erst nach erfolgreicher Anmeldung freigeschalten werden.[8][pp.7-8]

Session Handling

Wenn Sessions verwendet werden, um Benutzern Daten zuzuordnen, muss eine eindeutige Session-ID verwendet und der Benutzerin oder dem Benutzer zugeordnet werden. Diese Session-Daten dürfen vom Client nicht manipulierbar sein, außerdem nicht in der URL übertragen werden und mit einer ausreichenden Entropie und Zufälligkeit erstellt werden. Außerdem muss die Webapplikation Funktionen zur Verfügung stellen, um eine Session zu beenden, und diese Funktion muss nach einer definierten Zeitspanne von Inaktivität der Benutzerin oder des Benutzers automatisch durchgeführt werden.[8][p.8]

Inputvalidierung

Die Norm schreibt vor, Eingaben, die von einer Benutzerin oder einem Benutzer in das System getätigt werden, hinsichtlich folgender Kriterien zu überprüfen:

- Inhalt

- Plausibilität
- gültiger Wertebereich
- erlaubte Zeichen
- Länge
- Typ

Werden die Eingaben verwendet, um Kommandos oder ähnliches zu generieren, so müssen Filter, Methoden und Kodierungen verwendet werden, um die Verarbeitung von Steuerzeichen zu vermeiden. Außerdem muss verhindert werden, dass durch Benutzereingaben Angriffsvektoren für die Webapplikation entstehen. All diese Validierungen müssen serverseitig stattfinden. Es muss gewährleistet werden, dass durch Benutzereingaben nicht die Integrität des internen Speichers verletzt wird. Bei Einbindung von externen Ressourcen rät die Norm, dies so minimal wie möglich zu tun, und unbedingt zu verhindern, dass durch Benutzereingaben darauf zugegriffen werden kann.[8][pp. 8-9]

Kryptographie

Es wird empfohlen, nur ausreichend getestete kryptographische Verfahren zu verwenden. Die Verfahren dürfen keine Schwachstellen im Sinne der Anwendung beinhalten. Außerdem schreibt die Norm vor, dafür Sorge zu tragen, dass verwendete Schlüssel eine ausreichend hohe Entropie und Zufälligkeit aufweisen. [8][p. 10]

4.4.2 Bewertung

Bereich	Bewertung	Bemerkung
Logging	0	
Authentifizierung und Autorisierung	3	Vorgaben sind genau definiert - keine Beispiele
Kryptographie	1	Keine exakten Vorgaben aber marginal behandelt
Session Management	0	
Input Validation	3	Benutzereingaben- und Fileupload-Vorgaben sind genau definiert - keine Beispiele
Output Escaping	1	Grobe Vorgaben
Datenbank Sicherheit	0	
Fehler Management	1	Grobe Vorgaben
Aktualität	1	Aktuelle Fassung aus 2008
Summe:	10	

Tabelle 4.4: Bewertung ÖNORM A 7700

5 Unabhängige Gremien

5.1 NIST 800-64 Revision 2

Die Publikation der NIST (National Institute of Standards and Technology[13])trägt den Titel ‘Security Considerations in the System Development Life Cycle’. Wie dieser Titel bereits vermuten lässt, wird ein Lebenszyklusmodell vorgestellt um sichere Systeme entwickeln zu können. Im ersten der folgenden Kapitel wird das Modell an sich vorgestellt. Im darauf folgenden werden zusätzlich sicherheitsrelevante Themen analysiert.

5.1.1 Secure Development Lifecycle

Initiation

Diese Phase befasst sich mit dem Festlegen von Anforderungen, Meilensteinen, Zusammenhängen und Abhängigkeiten. Außerdem soll in dieser Phase eine Risikoanalyse durchgeführt werden. Es ist wichtig, bereits in dieser Phase darauf einzugehen, da so im Laufe des Projektes Zeit und Kosten gespart werden können. Es wird geraten, auch Teile des Entwicklerteams mit in das Team der Risikoanalyse aufzunehmen. Zum Einen kann so Awareness geschaffen werden und andererseits ermöglicht dies den Entwicklern, die definierten Risiken richtig zu erfassen.[14, pp.13-14]

Initiate Security Planing - Es muss sowohl geklärt werden, wer welche Verantwortlichkeiten zum Thema Sicherheit tragen muss, als auch wie das Reporting an übergeordnete Stellen funktioniert, damit das Projekt in die richtige Richtung gelenkt werden kann. Außerdem muss erörtert werden, woher Anforderungen an die Sicherheit kommen und wie damit umgegangen wird. Außerdem können auch schon die wichtigsten Meilensteine aus Sicherheitssicht festgelegt werden.[14, pp.14-15]

The Information System - Die Kategorisierung von Informationssystemen stellt einen wichtigen Bestandteil dar um ein passendes Sicherheitsniveau gewährleisten zu können. Hier soll daher festgestellt werden, welche Teile des Systems oder der Software die Unternehmensziele stark beeinflussen und welche weniger. Die NIST stellt dazu auch ein eigenes Dokument (800-60) zur Verfügung, in welchem das

Thema weiter ausgeführt wird. Außerdem wird empfohlen, sich an FIPS 199 zu halten. Inputs für eine Kategorisierung können aus vielen Bereichen kommen. Einige davon sind sicher das Risikomanagement und das Enterprise Architecture Management.[14, pp.15-17]

Ensure Use of Secure Information System Development Processes - Da gerade im Anfangsstadium ein großer Teil der Verantwortung für die Sicherheit der Software beim Entwicklungsteam liegt, da es ein sehr tiefes Verständnis für das entstehende Produkt hat, müssen für die Arbeit genaue Vorgaben und Rahmenbedingungen geschaffen werden um das Entstehen von sicheren Produkten zu ermöglichen. All diese Punkte sollten in einem Dokument zusammengefasst werden[14, pp.18-20]:

- **Secure Concept of Operations** Ein Konzept zur Erarbeitung des Codes soll erstellt werden. Dabei muss beispielsweise definiert werden, wo und wie das Code Repository zu abgelegt und versioniert werden soll. Genauso ist für regelmäßige Sicherungen von Selbigem zu sorgen.
- **Security Training for Development Team** Das Entwicklerteam soll laufend oder in regelmäßigen Abständen geschult werden um über aktuelle Bedrohungen informiert zu sein und so sicheren Code schreiben zu können. Die Regelmäßigkeit und auch der Umfang dieser Schulung könnte in diesem Schritt definiert werden.
- **Secure Environment** Die Umgebung in der die Entwicklungsarbeit geleistet wird muss ebenfalls ein hohes Maß an Sicherheit bieten. Das betrifft sowohl Arbeitsplätze als auch Server und Netzwerkkomponenten. Genaue Kontrollpunkte dazu sind in der NIST Publikation 800-53 definiert.

Development/Acquisition

Dieses Kapitel der Publikation handelt von der Entwicklungsphase des Systems. Hier wird definiert, wie den festgestellten Risiken aus vorangegangenen Schritten gegenüberzutreten ist. Außerdem kann eine erneute Risikoanalyse bereits auf exakteren Daten durchgeführt werden. Dabei können genauere Ergebnisse entstehen, was wiederum Maßnahmen leichter planen und steuern lässt.[14, pp.21-27]

Außerdem wird in dieser Phase eine Planung zum Testen erstellt. Festzulegen ist, wie oft Sicherheits- und Funktionalitätstests durchgeführt werden sollen. Dabei sind Verantwortungen zu klären, als auch mitwirkende Personen und Abhängigkeiten von Rahmenbedingungen oder Meilensteinen. Wenn automatisierte Tests erforderlich sind bietet die NIST mit ihrer Security Content Automation Protocol (SCAP) eine Hilfestellung an. Auch können bereits grob Umfänge und Schwerpunkte von Tests definiert werden. Da diese Phase die gesamte Entwicklungsphase begleitet, können als Output bereits erste Testergebnisse

vorliegen.[14, pp.21-23]

Des Weiteren soll - in Zusammenarbeit mit Entwicklern - die Architektur festgelegt werden. Speziell ist hier erneut darauf zu achten, dass die strategischen Ziele des Unternehmens adäquat unterstützt werden können. Natürlich soll aber auch auf Schnittstellen zu externen Serviceanbietern geachtet werden, genauso wie auf Vorgaben aus Normen und Richtlinien. Auch die Auditierbarkeit und das Logging sollten in diesem Schritt mitbetrachtet werden.[14, pp.24-27]

5.1.2 Bewertung

Bereich	**Bewertung**	textbfBemerkung
Logging	1	Es wird immer wieder betont wie wichtig ein Konzept dafür ist - es gibt aber keine Vorgaben
Authentifizierung und Autorisierung	0	
Kryptographie	0	
Session Management	0	
Input Validation	0	
Output Escaping	0	
Datenbank Sicherheit	0	
Fehler Management	0	
Aktualität	1	Aktuellste Fassung aus 2008
Summe:	2	

Tabelle 5.1: Bewertung NIST SP 800-64

5.2 OWASP - Secure Coding Practices v2

Diese Publikation beschreibt eine Checkliste, die in einen Software Development Lifecycle eingearbeitet werden kann. [3]

5.2.1 Input Validation

Die Publikation empfiehlt, die Quellen, von denen das System Inputs bekommt, nach Vertrauenswürdigkeit zu klassifizieren. Speziell von Quellen, die als nicht vertrauenswürdig eingestuft wurden, muss der Input validiert werden. Es empfiehlt sich, für eine solche Validierung eine zentrale Routine zu definieren und bei einem Verstoß, den Input nicht anzunehmen und daher auch nicht weiter zu verarbeiten. Es empfiehlt sich für die Validierung eine Whitelist zu verwenden, welche den akzeptierten Input beschreibt. Besonders wichtig ist es, Nicht-ASCII Zeichen und Sonderzeichen, die oftmals nicht als Eingabe benötigt werden, zu detektieren, wie beispielsweise: < > # % ( ) & + ´ "
Für jede Eingabe sollte auch die Länge und der Datentyp definiert werden.[3, p.5]

5.2.2 Output Encoding

Auch für die Codierung von Ausgaben soll eine einheitliche, getestete Routine verwendet werden. Vor allem wenn Befehle an eine Datenbank oder ein Betriebssystem in eine nicht-vertrauenswürdige Zone gesendet werden, soll der Inhalt so definiert sein, dass das Zielsystem sicher damit arbeiten kann.[3, p.5]

5.2.3 Authentication and Password Management

Für alle Inhalte, die nicht für die Öffentlichkeit bestimmt sind, muss eine Authentifizierung nötig sein. Dabei sollte man auf getestete und weit verbreitete Algorithmen vertrauen. Es muss außerdem gewährleistet werden, dass das Authentifizierungssystem auch bei Fehlern in einem definierten Zustand bleibt. Zudem muss dieses System, von dem System dass die Zugriffsrechte verwaltet, getrennt sein. Selbst bei der Ausgabe von Fehlern soll darauf geachtet werden, dass die Fehlermeldung generisch ist und nicht auf die Fehlerursache schließen lässt. Weiters wird empfohlen, Anmeldedaten nicht über HTTP GET zu übermitteln. In sehr sensiblen Bereichen sollte eine Mehrfaktorenauthentifizierung eingesetzt werden.[3, pp.6-7]

Passwörter dürfen nur als Hash und nie im Klartext gespeichert werden. Außerdem sollten sie am User-Screen nicht im Klartext ersichtlich sein. Es wird dringend empfohlen, eine Mindestlänge von 8 Zeichen zu erzwingen, wobei 16 Zeichen ratsamer wären. Genauso sollten ein Mindestalter und ein Gültigkeits-

zeitraum für ein Passwort definiert sein. Es muss eine sichere Möglichkeit geben, wie eine Anwenderin oder ein Anwender das Passwort zurücksetzen kann. Über eine Passwortänderung oder den Versuch dessen, sollte die Benutzerin oder der Benutzer in Kenntnis gesetzt werden.[3, pp.6-7]

5.2.4 Session Management

Die Publikation rät, Frameworks für Session Management zu nutzen, sodass die Applikation nur noch die Information bekommt, ob die Session gültig ist oder nicht. Eine Session-ID muss immer auf einem vertrauenswürdigen System mittels geprüfter Algorithmen erstellt werden und einzigartig sein. Eine permanente Anmeldung sollte nicht möglich sein. Dies kann durch eine Logout-Funktion und einem automatischen Timeout erreicht werden. Alle Cookies, in denen Session-Daten gespeichert werden, müssen sowohl mit dem 'Secure' als auch mit dem 'HTTPOnly' Flag versehen sein. Je sensitiver der Bereich ist, für den die Session gültig ist, desto öfter kann eine neue Authentifizierung und damit eine Erneuerung des Session-ID nötig sein.[3, pp.7-8]

5.2.5 Access Control

Um die Entscheidung zu treffen, ob auf Ressourcen zugegriffen werden darf oder nicht, dürfen nur Daten herangezogen werden, die auf einem vertrauenswürdigen System gespeichert sind. Wenn möglich sollte man Frameworks oder Tools für diese Aufgabe heranziehen. Es ist besonders wichtig, Daten die nicht öffentlich sind, vor unbefugten Zugriff zu schützen. Dies kann beispielsweise Folgendes betreffen[3, pp.8-9]:

- Files oder ähnliche Ressourcen auf einem Server
- Geschützte Websites
- Geschützte Services
- Geschützte Applikationen
- Geschützte Konfigurationen
- etc.

Es muss außerdem möglich sein, Konten zu deaktivieren und zu löschen. Außerdem müssen Policies definiert werden, welche Rolle auf welche Ressourcen Zugriff bekommen soll.[3, pp.8-9]

5.2.6 Cryptographic Practices

Alle kryptographischen Funktionen sollen auf einem vertrauenswürdigen System implementiert sein. Kryptographie-Module sollen dem momentanen Stand der Technik entsprechen und auch bei Fehlern in einen definierten Zustand bleiben. Alle zufälligen Variablen müssen mittels eines Kryptographie-Modules erzeugt werden. Es empfiehlt sich, den Umgang mit Schlüsseln und Geheimnissen in einer Richtlinie zu definieren.[3, p.9]

5.2.7 Error Handling and Logging

An eine Benutzerin oder einen Benutzer dürfen nur generische Fehlermeldungen (ohne Stack-Trace oder Ähnlichem) ausgegeben werden. Selbst im Fehlerfall muss sichergestellt sein, dass das System in einem definierten Zustand bleibt und zugeordneter Speicher wieder freigegeben wird.[3, p.9]

Logs sind wichtig um Fehlerursachen finden zu können. Dies erfordert die Speicherung aller benötigten Daten. Wichtig ist jedoch, dass keine unnötigen, oder gar sensible Daten wie beispielsweise Passwörter in einem Log-File gespeichert werden.[3, p.9]

5.2.8 Data Protection

Zum Schutz der Daten sollte das Grundprinzip gelten, dass jeder nur die Rechte bekommt, die er auch benötigt. Sensitive Daten dürfen nur verschlüsselt und via HTTP POST übertragen werden. Dadurch lässt sich vermeiden, dass die Daten während der Übertragung verändert, beziehungsweise abgezogen werden. Alle Auto-Logon Funktionen sollten deaktiviert und alle für den Client sichtbaren Codeteile von Kommentaren befreit unleserlich gemacht werden. Das erhöht die Sicherheit der Daten. [3, p.10]

5.2.9 Communication Security

Sensible Kommunikation soll, wie eben erwähnt, nur verschlüsselt stattfinden. Die dazu verwendeten Methoden sollten korrekt implementiert sein und aktuelle Algorithmen erzwingen. Auch durch Fall-Back Funktionen sollte es nicht möglich sein, auf einen unsicheren Algorithmus zugreifen zu können. Werden Zertifikate verwendet (HTTPS), müssen diese gültig und mit dem korrekten Domain-Namen versehen sein.[3, p.10]

5.2.10 System Configuration

Um die Sicherheit des Systems gewährleisten zu können muss immer die aktuelle Version aller Komponenten in Betrieb und alle dazugehörigen Patches eingespielt sein. Genauso ist es wichtig, die Angriffsoberfläche zu minimieren. Dazu sollten beispielsweise nicht verwendete Funktionen und Services deaktiviert werden. Genauso ist es ratsam nur HTTP Methoden zu unterstützen, die auch in Verwendung sind, beziehungsweise Directory Listings eines Webservers zu deaktivieren. Um die Übersicht behalten zu können, kann es von Nöten sein, ein Asset Management einzuführen, das alle Teile des Systems abbildet und eventuell auch Veränderungen dokumentiert.[3, p.11]

5.2.11 General Coding Practices

Zusätzlich zu den eben erwähnten Punkten gibt es allgemein gehaltene Punkte deren Einhaltung für die Erhöhung des Sicherheitsniveaus dringend empfohlen wird. So ist es immer ratsam, bereits bestehenden und getesteten Code zu verwenden als selbst neuen Code zu implementieren. Genauso ist die Verwendung von API's ratsam um Services anderer Herkunft einzusetzen. Es muss gewährleistet werden, dass eine Anwenderin oder ein Anwender keinen eigenen Code ausführen beziehungsweise einfügen kann. Dies beinhaltet auch, dass geteilte Variablen einen erhöhten Schutzbedarf genießen müssen und Daten, die von der Benutzerin oder dem Benutzer verändert werden können, nicht direkt an dynamische Funktionen übergeben werden, sondern validiert sein müssen. Nicht zuletzt wird empfohlen, einen sicheren Update-Mechanismus in IT-Systeme zu implementieren um auch für zukünftig Verwundbarkeiten gewappnet zu sein und darauf reagieren zu können.[3, p.13]

5.2.12 Bewertung

Bereich	Bewertung	Bemerkung
Logging	3	Genaue Regeln - keine Beispiele
Authentifizierung und Autorisierung	3	Genaue Regeln - keine Beispiele
Kryptographie	2	Vorgaben sind nicht konkret, aber grobe Regeln definiert
Session Management	3	Genaue Regeln - keine Beispiele
Input Validation	3	Genaue Regeln - keine Beispiele
Output Escaping	2	Einige Regeln definiert
Datenbank Sicherheit	3	Genaue Regeln - keine Beispiele
Fehler Management	3	Genaue Regeln - keine Beispiele
Aktualität	2	Aktuelle Fassung aus 2011
Summe:	24	

Tabelle 5.2: Bewertung OWASP Secure Coding Pracitces v2

5.3 IT-Grundschutz

5.3.1 Gefährdungs- und Maßnahmenkataloge

Der IT-Grundschutz des deutschen BSI (Bundesamt für Sicherheit in der Informationstechnik) stellt ein kostenloses Framework für verschiedene Belangen der Informationssicherheit zur Verfügung. Die Kataloge können beispielsweise als Grundlage einer Risikoevaluierung genutzt oder für die Linderung einzelner Risiken herangezogen werden.[15] Im Bereich von Programmierung und Applikationssicherheit empfiehlt der IT-Grundschutz ein Vorgehen nach den folgenden Punkten:[16]:

Planung und Konzeption

Es ist unerlässlich, ein Softwareentwicklungsprojekt genau zu planen. Dabei ist es von großer Wichtigkeit, Rollen und Verantwortlichkeiten festzulegen. Es muss zumindest die Gesamtverantwortung für die Sicherheit im Softwareprojekt definiert werden. Diese Rolle ist für die Durchführung von Anforderungs- und Risikoanalysen zuständig, definiert Richtlinien, überprüft deren Umsetzung und begleitet das Projekt bis zur Finalisierung. Dabei ist es unerlässlich, dieser Rolle sowohl genügend Ressourcen (zeitlich, finanziell, personell) zur Verfügung zu stellen als auch Zugang zu Informationen rund um Informationssicherheit zu gewähren.[17][16]

Außerdem beschreibt der IT Grundschutz in dieser Phase die Auswahl eines geeigneten Modelles zur Softwareentwicklung, wie beispielsweise:[16][18]

- Wasserfallmodell
- Prototyping
- MDSD (Model-Driven Software Development)
- TDD (Test-Driven Development)

Die Belangen der Informationssicherheit müssen in das jeweilige Modell eingearbeitet werden, damit die korrekten Maßnahmen und Überprüfungen an den richtigen Stellen umgesetzt werden.[18]
Auch Anforderungen von Seiten der Compliance - beispielsweise aus Verträgen oder Gesetzen - müssen eingehalten werden. Daraus könnten erhebliche finanzielle und rechtliche Konsequenzen resultieren. Beispiele dafür sind die urheberrechtlich geschützten Programmteile und Bibliotheken. Insgesamt gilt es zusätzlich zu Verträgen und Gesetzen auch Anforderungen aus Normen oder Patenten einzuhalten. Bei Unklarheiten kann ein eventuell vorhandener Compliance-Beauftragter konsultiert werden.[18]

Beschaffung

Wenn die Verantwortlichkeiten geklärt und die Anforderungen definiert sind, ist es wichtig, für eine geeignete Entwicklungsumgebung zu sorgen und das benötigte Werkzeug dafür bereitzustellen. Die Auswahl der Entwicklungsumgebung soll hauptsächlich anhand der gewählten Programmiersprache und des Anwendungstypes festzulegen. Nicht außer Acht dürfen dabei jedoch Faktoren wie Lizenzkosten, Zusatzfunktionen und Anschaffungskosten gelassen werden. Auch Wartung und Bedienbarkeit sind zu bedenken.[19][16]

Genauso müssen auch die anderen Werkzeuge (zum Beispiel Compiler, Interpreter oder Debugger) definiert werden, welche im Entwicklungsprozess Anwendung finden sollen. Da diesen und ähnlichen Komponenten oft ein hohes Maß an Automatisierung (zum Beispiel in der Code-Generierung) zugestanden wird, ist es unerlässlich, dass diese fehlerfrei arbeiten und nicht unerkannt manipuliert werden können. [20][21][16]

Umsetzung

Während der Entwicklung von Software ist es wichtig, die gewählte Entwicklungsumgebung sicher einzusetzen. Es muss dafür gesorgt werden, dass Entwicklungs-, Test- und Produktivumgebung strikt von einander getrennt sind und sich nicht gegenseitig beeinflussen. Dies kann zum Beispiel durch Netztrennung oder Zugriffskontrollen passieren. Außerdem müssen Code-Repositories ausreichend geschützt werden um das Programmierte vor Veränderung zu bewahren. Bei hohen Sicherheitsanforderungen muss auch den Zutritt zu den Räumlichkeiten in denen die Entwicklungstätigkeiten stattfinden, zu begrenzen. Es empfiehlt sich darüber hinaus, Kommentare und ähnliche, für den Produktivbetrieb nicht relevante Daten, vor der Publizierung aus dem Code zu entfernen. Es könnte sich hierbei um wichtige Informationen für einen Angreifer handeln. Manche Entwicklungsumgebungen bieten dazu einen zumindest teilautomatisierten Ansatz.[22][16]

Es ist außerdem unerlässlich, schon beim Design des Systems auf Sicherheit zu achten. Der IT-Grundschutz empfiehlt, folgende Punkte umzusetzen um ein angebrachtes Sicherheitsniveau erreichen zu können[23]:

- Alle Eingabedaten müssen vor der Weiterverarbeitung serverseitig validiert werden.
- Die Datenübertragung zwischen Systemkomponenten sollte immer verschlüsselt sein.
- Das IT-System, auf dem Software betrieben wird, soll mit sicheren Einstellungen versehen werden. Dabei ist speziell auf das Betriebssystem und verwendete Module zu achten.

- Bei Fehlern oder Ausfall des Systems dürfen keine sensitiven Daten preisgegeben werden.
- Der Betrieb von Software muss mit möglichst geringen Benutzerrechten möglich sein.

Auch die Implementierung dieser und aller anderer Funktionen sollte sicher von statten gehen. Dazu ist es vor allem bei sicherheitskritischen Komponenten nötig, ein Vier-Augenprinzip anzuwenden oder den Code durch mehrere Entwickler überarbeiten zu lassen. Außerdem empfiehlt es sich, eine Versionskontrolle einzusetzen, welche die Dokumentation der Unterschiede zwischen zwei Versionen bietet und auch das Zurücksteigen auf Vorversionen ermöglicht. [24]

Ergebnisse aus der Softwareentwicklung, die in den Produktivbetrieb übernommen werden sollen, müssen ausreichenden Testverfahren unterzogen werden. Ein Teil davon ist die statische Analyse wie sie beispielsweise in einem Source Code Review umgesetzt wird. Anders sind dabei dynamische Tests, welche das Verhalten von Software zur Laufzeit testet. Es empfiehlt sich, parallel zur Entwicklung der Software, mit dem Design von Testfällen zu beginnen, da so Zeit vor der Produktivstellung gespart werden kann.[25]

Auch die Dokumentation von Softwareprojekten spielt eine wichtige Rolle. Sie macht die Ergebnisse auch zukünftig wartbar und deckt damit sicherheitsrelevante Aspekte auf. Man unterscheidet dabei in Projekt- und Systemdokumentation. Die Projektdokumentation sollte alles um das Projekt und zu Entscheidungsfindungen enthalten, wie zum Beispiel: Dokumentation des Projektverlaufes, Vergabeprozesse und Anforderungen an das System. Systemdokumentation ist die Dokumentation des Ergebnisses und sollte beispielsweise enthalten: [26]

- System-Spezifikation
- Schnittstellen Definitionen (inklusive Dokumentation der intern genutzten Bibliotheken, Entwicklungsumgebungen und weiterer Software)
- Codierungsrichtlinien (z. B. Namenskonventionen, Strukturierungsrichtlinien, etc.)
- Dokumentation der Qualitätssicherung und der Tests
- Dokumentation für die Installation und die Inbetriebnahme, Anleitungen für die Administratoren
- Bedienungsanleitung für die Anwender

IT-Grundschutz betrachtet auch die Schulung der Entwickler als wichtiges Element, um ein Produkt zu erhalten, das hohen Sicherheitsansprüchen genügt.

Betrieb

Nach erfolgreichem Testen der Software und ausreichender Schulung der Administratoren auf die Wartung des Produktes kann das Ergebnis des Softwareentwicklungsprojektes nun produktiv verwendet werden. Es möge berücksichtigt sein, dass für jegliche Problemstellung ein Ansprechpartner zur Verfügung steht. Es müssen nun die Benutzer auf das Produkt geschult und auf mögliche Dokumentation hingewiesen werden. Für den Installationsprozess selbst muss eine Anleitung existieren, die auch etwaige Fehlerquellen oder Abweichungen auf unterschiedlichen Zielsystemen beinhaltet.[27]

“Never touch a running system” - das ist der falsche Ansatz wenn es um den Erhalt der Informationssicherheit geht. Viele Schwachstellen, egal ob im Betriebssystem, der Firmware oder sogar der eingesetzten Hardware, werden erst im Laufe der Zeit erkannt. Deshalb ist es wichtig, dass ein Patchplan eingeführt wird, der beinhaltet, wann Updates eingespielt werden können und woher die Updates zu beziehen sind. Jedenfalls müssen Updates vor dem Einspielen immer, auch bei verifizierter Quelle, durch ein Antivirenprogramm überprüft werden, und zuerst auf einem Testsystem durchgeführt werden. Die Dokumentation des Patchvorganges ermöglicht es nachzuvollziehen, welche Version zu welchem Zeitpunkt im Einsatz war. [27][28]

Aussonderung

Bei der Deinstallation von Software ist auf das Rückgängigmachen aller Änderungen am System und an Systemdateien zu achten. Oft ist es dazu nötig, alle Änderungen die während der Installation und während Updates vorgenommen wurden aufzuzeichnen. Nur so ist sichergestellt, dass alle Änderungen rückgängig gemacht wurden. Speziell ist natürlich auf das Eliminieren von sensiblen Daten zu achten.[29]

5.3.2 Bewertung

Bereich	Bewertung	Bemerkung
Logging	0	
Authentifizierung und Autorisierung	1	Grundlegende Definitionen und Hinweise
Kryptographie	2	Wird im Teil 'Softwareentwicklung' nur marginal beschrieben, aber es gibt genaue Vorgaben in eigenem Abschnitt
Session Management	2	Wird im Teil 'Softwareentwicklung' nur marginal beschrieben, aber es gibt genaue Vorgaben in eigenem Abschnitt
Input Validation	2	Wird im Teil 'Softwareentwicklung' nur marginal beschrieben, aber es gibt genaue Vorgaben in eigenem Abschnitt
Output Escaping	0	
Datenbank Sicherheit	0	
Fehler Management	2	Wird im Teil 'Softwareentwicklung' nur marginal beschrieben, aber es gibt genaue Vorgaben in eigenem Abschnitt
Aktualität	2	Aktuelle Fassung aus 2010
Summe:	11	

Tabelle 5.3: Bewertung IT Grundschutz

5.4 CERT - Secure Coding Standard

CERT stellt eine Liste ‘Top 10 - Secure Coding Standards’ zur Verfügung die unterschiedlichste Bereiche abdecken. Dem Aufbau folgt auch dieses Dokument. Zu den einzelnen Einträgen der Liste werden Regeln aus dem ‘Coding Standard for Java’ erläutert. Die Regeln, die zur Anwendung kommen, werden dezitiert angeführt. Wenn eine Vielzahl von Regeln auf einen Punkt zutrifft, so wird nur eine repräsentative Auswahl davon angeführt und erläutert - als Grundlage dazu dient die risikobasierte Bewertung, die von CERT vorgenommen wurde.[30][31]

Input Validation

Das größte Risiko dieser Kategorie wird SQL-Injections zugeschrieben. Bei einer schlechten Implementierung der Authentifizierung können, durch Veränderung des Inputs, nicht erfolgreich authentifizierte Benutzerinnen und Benutzer Zugriff auf das System bekommen. Das wird durch die direkte Übergabe von Benutzereingaben an ein SQL Statement ermöglicht und kann wie folgt ausgenutzt werden. Wenn die Authentifizierung via folgender SQL-Query abgehandelt wird und der Input der Benutzerin oder des Benutzers nicht anderweitig überprüft wird. Kann via nachfolgendem Input im Feld Passwort Erfolg erzielt werden [30][31]:

```
SELECT * FROM db_user WHERE username='<USERNAME>' AND
                      password='<PASSWORD>'
```

```
validuser' OR '1'='1;
```

Das führt dazu, dass folgende Query gegen die Datenbank ausgeführt wird - welche sich immer als ‘TRUE’ herausstellt:

```
SELECT * FROM db_user WHERE username='<USERNAME>' AND password='' OR
   '1'='1'
```

Die einzige Voraussetzung ist, dass der Benutzer existiert.
Dem kann beispielsweise mittels eines Prepare-Statements entgegengewirkt werden. Diese müssen so eingesetzt werden, dass eine SQL-Injection nicht mehr möglich ist.
Zur Anwendung gekommene Regeln: IDS00-J. Prevent SQL injection

Default deny & Adhere to the principle of least privilege

Speziell sensible Daten sollten durch spezielle Mechanismen geschützt werden. Ein solcher Mechanismus könnte beispielsweise der von Java zur Verfügung gestellte Security Manager sein. Dieser ermöglicht es, Berechtigungen erneut zu überprüfen. So könnte beispielsweise sichergestellt werden, ob die Benutzerin oder der Benutzer ein File lesen darf, bevor er es angezeigt bekommt. Beispielhaft könnte eine Implementierung wie folgt aussehen:

```
SecurityManager sm = System.getSecurityManager();

if (sm != null) { // Check whether file may be read
  DTDPermission perm = new DTDPermission("/local/", "readDTD");
  sm.checkPermission(perm);
}
```

[30][31]

Zur Anwendung gekommene Regeln: SEC04-J. Protect sensitive operations with security manager checks

Sanitize data sent to other systems

Da Schnittstellen zwischen Komponenten nicht nur genau definiert sondern auch eingehalten werden müssen, ist es nötig, alle Inputs, die von einem externen System kommen und alle Outputs, die an ein externes System gesendet werden, zu überprüfen. Der CERT Coding Standard schlägt dazu beispielsweise die Validierung mittels regulären Ausdrücken vor, welche im Packet java.util.regex enthalten sind. Dadurch können Zeichenfolgen auf ein bestimmtes Muster hin überprüft werden.[31]
Daten, die in anderen Systemen (beispielsweise als Grundlage für HTML Content) verwendet werden, müssen validiert werden und dürfen nur aus ‘sicheren’ Zeichen bestehen. Es empfiehlt sich, diesen Zeichensatz vorab festzulegen und alle In-/Outputs mittels derselben Funktion überprüfen zu lassen, sodass immer ein konsistentes Ergebnis erwartet werden kann, so wie es beispielsweise von dieser Funktion übernommen wird[31]:

```
// Normalizes to known instances
 private String normalize(String input) {
   String canonical =
     java.text.Normalizer.normalize(input, Normalizer.Form.NFKC);
   return canonical;
 }
```

Zur Anwendung gekommene Regeln: IDS51-J. Properly encode or escape output / IDS08-J. Sanitize untrusted data included in a regular expression

Practice defense in depth

Es wird von CERT empfohlen, mehrere Sicherheitsschichten einzuziehen. Selbst wenn eine dieser Schichten versagen sollte, so ist die Sicherheit des Systems dennoch durch andere Schichten gegeben. So ist es beispielsweise ratsam, Return-Werte von Methoden zu überprüfen, sodass auf Fehler einfach reagiert werden kann und beim Versagen von einzelnen Funktionen andere Methoden greifen können. [30][31]

Genauso ist darauf zu achten, dass Variablen, die zum Speichern von sensiblen Daten verwendet werden mit eingeschränkten Zugriffsrechten ausgestattet sind, sodass niemand, der nicht dafür autorisiert ist, auf die Variable zugreifen kann. Um solche Variablen beeinflussen zu können empfiehlt es sich, Getter- und Setter-Methoden zu definieren. So kann auch sichergestellt werden, dass die Variablen nie Werte annehmen, welche nicht valide sind. Im Folgenden findet sich ein Beispiel für die Umsetzung: [31]

```
public class Widget {
  private int total; // Declared private

  public int getTotal () {
    return total;
  }

  // Definitions for add() and remove() remain the same
}
```

Zur Anwendung gekommene Regeln: EXP00-J. Do not ignore values returned by methods / OBJ01-J. Limit accessibility of fields

Use effective quality assurance techniques

Es wird darauf verwiesen, dass ausgiebiges Testen von Systemen und Applikationen Schwachstellen identifizieren und eliminieren kann. Um also die Qualität von Software sicherzustellen müssen Penetration-Tests, Software Reviews von Externen und Fuzztesting umgesetzt werden. Das soll dazu führen, dass sowohl die Anwendung als auch der Source Code selbst auf Schwachstellen untersucht wurde. Beim Fuzztesting geht es darum, die Anwendung mit Inputs zu bespielen, die keinen Sinn ergeben oder zu Problemen führen können. Damit soll die Stabilität der Anwendung festgestellt und nach Ableitung der

Maßnahmen auch verbessert werden. CERT stellt im Rahmen dieser Vorgaben auch viele Tools zur Erfüllung dieser Aufgaben zu Verfügung:[30][32]

- Basic Fuzzing Framework (BFF)
- Failure Observation Engine (FOE)
- DidFail
- CERT Linux Forensics Tools Repository
- CryptHunter
- Secure Coding Validation Suite for C

5.4.1 Bewertung

Bereich	Bewertung	Bemerkung
Logging	4	Konkrete Umsetzungen vorhanden inkl. Code-Snipets
Authentifizierung und Autorisierung	3	Genaue Vorgaben vorhanden
Kryptographie	4	Verschlüsselte Kommunikation inkl. Code-Snipets in mehreren Beispielen ausgeführt
Session Management	4	Genaue Vorgaben inkl. Code-Snippets vorhanden
Input Validation	4	Genaue Vorgaben inkl. Code-Snippets vorhanden
Output Escaping	3	Genaue Vorgaben - kleinerer Umfang
Datenbank Sicherheit	0	
Fehler Management	3	Genaue Vorgaben - kleinerer Umfang
Aktualität	4	wird laufend aktualisiert
Summe:	29	

Tabelle 5.4: Bewertung CERT

5.5 Java Secure Coding Guideline

Java wurde von SUN entwickelt, welche 2010 von Oracle gekauft wurde. Oracle stellt Leitfäden wie beispielsweise zum Thema Secure Coding zur Verfügung. Die Entwicklerinnen und Entwickler der Programmiersprache geben am Beginn der Guideline allgemeine Ratschläge zur Sicherheit von Software, danach gehen sie auf viele spezifische Punkte ein. Im Folgenden werden einige davon kurz zusammengefasst:

5.5.1 Input Validation

Die Validierung von Daten aus nicht vertrauenswürdigen Quellen, wie beispielsweise Inputs von Benutzerinnen und Benutzern oder Informationen aus Drittsystemen, zählt zu einer wichtigen Aufgabe für die Sicherheit von Applikationen. Wird der Input nicht ordnungsgemäß überprüft kommt es zu Risiken wie Overflows oder Path Traversals. Dadurch können weitere Systemschwächen erzeugt beziehungsweise ausgenützt werden. Es wird außerdem empfohlen auch die Outputs aus Funktionen anderer Klassen zu überprüfen da es durch die Komplexität von Software oft zu Problemen kommen kann. Eine weitere Empfehlung lautet, Wrapper über native Methoden zu verwenden. So können noch Checks definiert werden, bevor die Daten weiter verarbeitet werden. [33]

5.5.2 Injection

Oftmals werden, um Angriffe mittels Injections durchzuführen, Strings mit Sonderzeichen in Eingabefelder eingefügt. Was daher vor Injections schützen könnte ist die eben erwähnte Validierung von Input und das damit verbundene filtern von Sonderzeichen. Weiters empfiehlt Oracle dynamische SQL abfragen wo möglich zu verhindern oder mittels der Klassen java.sql.PreparedStatement oder java.sql.CallableStatement zu verwenden. Dadurch kann SQL Injection Angriffen entgegengewirkt werden. Jedoch nicht nur beim Input von Daten muss auf de Sicherheit geachtet werden. Ebenso ist dies bei Ouputs von Systemen erforderlich. Besonders bei Ausgaben von HTML oder XML Daten können Risiken auftreten. Daher ist es nötig Daten zuerst zu filtern oder kodiert werden. Da es sich dabei nicht um eine triviale Aufgabe handelt wird geraten auf Tools und Libraries zu setzen. Außerdem ist es sehr ratsam, dafür zu sorgen, dass keine Daten aus nicht vertrauenswürdigen Quellen als CLI Befehle ausgeführt werden können. Wenn eine solche Funktion nötig ist muss der Input ganz genau überprüft werden, um das Risiko einzuschränken.[33]

5.5.3 Vertrauliche Daten

Vertrauliche Daten können einer Angreifer oder einem Angreifer wichtige Informationen über das System geben. Die kann auch schon bei einfachen Tätigkeiten wie das Öffnen eines Files passieren. Beim Versuch ein File zu öffnen das es nicht gibt entsteht eine FileNotFoundException. Es würde sich natürlich anbieten, die Fehlermeldung an die Benutzerin oder den Benutzer auszugeben. Jedoch werden dabei möglicherweise viele Informationen preisgegeben. So kann man beispielsweise aus dem Aufbau des Pfades des nicht gefundenen Files eruieren, welches Betriebssystem verwendet wird. Außerdem ist es möglich, den Benutzernamen auszulesen wenn beispielsweise auf das eigene Homeverzeichnis zugegriffen wird. Es ist außerdem wichtig, dass Sicherheitsfunktionalitäten nicht auf Exceptions angewiesen sind, diese könnten sich nämlich zukünftig ändern was möglicherweise Sicherheitsvorkehrungen außer Kraft setzt.[33]
Nicht nur um Angreifer nicht mit relevanter Information auszustatten, sondern auch aus Datenschutzgründen muss auf vertrauliche Daten acht gegeben werden. Daten wie beispielsweise die Sozialversicherungsnummer oder Passwörter von Benutzerinnen und Benutzer dürfen nicht in Logfiles aufgenommen werden. Diese Daten sind für die Administratorin oder den Administrator nicht relevant und sollten daher auch nicht zugänglich sein. Außerdem sollte in Betracht gezogen werden, diese Daten nach der Verarbeitung auch aus dem Speicher zu löschen. Dadurch kann verhindert werden, dass diese Daten in Core Dumps oder dergleichen enthalten sind.[33]

5.5.4 Denial of Service

Unter Denial of Service versteht man das überfordern und damit das nicht-verwendbar-machen von Applikationen. Ressourcen die oft betroffen sind, sind beispielsweise CPU, Memory oder Speicherplatz.[33]

Folgende Aktivitäten und Datentypen sollten soweit als möglich eingeschränkt beziehungsweise unterbunden werden, da sie sehr ressourcenintensiv sind und daher für solche Angriffe gut geeignet sind[33]:

- Abrufen von großen Vektorgraphiken wie beispielsweise SVG-Files
- ZIP-Bombs - Das sind Files die stark komprimiert sind und beim entpacken große Mengen an Speicherplatz verbrauchen
- XML entitiy expansion kann dazu führen, dass XML Files sehr groß werden. Daher sollte das XMLConstants.FEATURE_SECURE_PROCESSING Feature aktiviert sein.
- Oftmaliges Eintragen von Werten mit identen Hashes in eine Hashtable

- XPath kann sehr viel CPU nötig machen
- Detailiertes Logging von unübichen Aktivitäten führt zu großem Speicherverbrauch

Außerdem ist es sehr wichtig, Ressourcen wieder freizugeben sobald sie nicht mehr benötigt werden. Beispiele dafür sind manuell allokierter Speicherplatz oder geöffnete Files und Verbindungen wie beispielsweise zu Datenbanken. Außerdem ist darauf zu achten, dass Routinen, welche die Verfügbarkeit von Ressourcen überprüfen nicht auf von Integer Overflows betroffen sein können. Dadurch würden die Ergebnisse verfälscht und das gewünschte verhalten wahrscheinlich nicht erreicht werden können.[33]

5.5.5 Bewertung

Bereich	Bewertung	Bemerkung
Logging	2	Vorgaben sind in andere Kapitel eingearbeitet - keine exakten Definitionen
Authentifizierung und Autorisierung	4	Genaue Erklärung des SecurityManagers und des Zugriffskonzept mit Beispielen und Veranschaulichungen
Kryptographie	0	
Session Management	0	
Input Validation	4	Detailierte Vorgaben mit Beispielen vorhanden
Output Escaping	2	In Kapitel 'Injection and Inclusion'eingearbeitet
Datenbank Sicherheit	4	Im Kapitel'Injection and Inclusion'eingearbeitet - genaue Vorgaben und Beispiele vorhanden
Fehler Management	2	Ungenaue Vorgaben in Kapiteln eingearbeitet
Aktualität	4	Wird laufend aktualisiert
Summe:	22	

Tabelle 5.5: Bewertung Java Secure Coding Guideline

6 Microsoft - Secure Development Lifecycle

Microsoft veröffentlicht - als großer Softwarehersteller - ein Modell, um Sicherheit im Lebenszyklus von Software zu gewährleisten, wie in folgender Graphik ersichtlich. Microsoft teilt diesen Lebenszyklus in sieben Elemente ein, die im Folgenden analysiert werden:

Abbildung 6.1: Microsoft Secure Development Lifecycle[9]

6.0.1 1. Training

In dieser Phase soll gewährleistet werden, dass alle technischen Beteiligten (Tester, Entwickler, ...) geschult werden um Gefahren zu erkennen, und Fähigkeiten im Sicherheitsumfeld wie beispielsweise Secure Coding oder Threat Modeling umsetzen zu können. Mindestens einmal jährlich sollte eine solche Schulung von allen technisch Mitwirkenden besucht und abgeschlossen werden.[9]

6.0.2 2. Requirements

In diesem Bereich sollen Anforderungen sicherheitstechnischer Natur festgeschrieben werden. Dies kann beispielsweise über einen Fragebogen erfolgen, wie er von Microsoft zur Verfügung gestellt wird [10]. Was dabei entstehen soll ist ein Mindestniveau an Privacy- und Sicherheitszielen, welche von der Applikation erfüllt werden müssen. Des Weiteren sollen in diesem Schritt Bug Bars und Quality Gates definiert werden, um Sicherheitslücken einstufen und damit umgehen zu können. In einer solchen Bug Bar wird definiert welche Kategorien von Schwachstellen existieren und nach welchen Kriterien diese erfüllt werden. In den Quality Gates könnte beispielsweise definiert sein, dass eine Software mit kritischen Sicherheitslücken nicht verteilt werden darf, jedoch bis zu 5 hochpriore Sicherheitslücken toleriert werden können.[9]

Um oben genannte Entscheidungen treffen zu können, sieht der Microsoft Secure Lifecycle eine Risikoanalyse vor. [9]

6.0.3 3. Design

In dieser Phase wird die Umsetzung der Anforderungen, die im Schritt zuvor definiert wurden, geplant. Alle Entscheidungen die in dieser Phase getroffen werden, beeinflussen die darauf folgenden Schritte stark. Ein Mittel, um das Design zu verbessern, ist die Anlage eines Threat Model. Dabei wird visualisiert, wie die einzelnen Programmkomponenten zusammenspielen, wo es Schnittstellen geben muss und an welchen Stellen sich eine weitere Sicherheitsvorkehrung bezahlt machen könnte. Beispielsweise kann man für diese Aufgabe ein Tool von Microsoft verwenden, das von Microsoft momentan kostenfrei bezogen werden kann.[11, pp.4-5][9] Außerdem sollte die Angriffsoberfläche des geplanten Programmes genau betrachtet werden. Eventuell kann sie verringert werden um es einem Angreifer schwerer zu machen in das System einzudringen. Dadurch können zwar nicht alle Schwachstellen beseitigt werden, jedoch trägt dieser Schritt maßgeblich zu einem erhöhten Sicherheitsniveau bei.[11, pp.4-5][9]

6.0.4 4. Implementation

Auch in der Implementierungsphase muss auf die Sicherheit geachtet werden. Um den Code überprüfen zu können empfiehlt Microsoft die Verwendung einiger - teilweise hauseigener - Produkte, wie beispielsweise[12]

- **Binskim Binary Analyzer** - dieses Tool kann Binaries auf Programmfehler untersuchen, welche möglicherweise zu Schwachstellen führen können
- **FxCop** - Es handelt sich um ein statisches Tool zum Analysieren von .NET Applikationen auf Sicherheitslücken und Performanceschwächen.
- **SiteLock ATL Template** - unterstützt bei der Entwicklung von ActiveX-Applikationen.

In diesem Schritt sollte außerdem darauf geachtet werden, dass der produzierte Code keine veralteten oder unsicheren Funktionen enthält. Es wird empfohlen, sich an Best Practices zu halten und den Code gegen sogenannte ‘Banned Lists’ zu überprüfen.
Da mithilfe diverser Tools Schwachstellen bereits früh gefunden werden können, wird der Einsatz von statischen Code Analyse Tools - wie zum Beispiel das oben erwähnte Tool ‘FxCop’ - , empfohlen. [9]

6.0.5 5. Verification

An dieser Stelle kann der Sourcecode bereits mittels dynamischen Analysetools überprüft werden um den Blickwinkel eines Angreifers auf die Applikation in die Analyse miteinbeziehen zu können. Dabei können zum Beispiel Auffälligkeiten wie Memory Corruption festgestellt werden. Das wiederum bietet die Möglichkeit, auf diese Fehler bereits vor der Veröffentlichung der Applikation reagieren zu können und so das Sicherheitsniveau weiter in die Höhe zu treiben. Weiters können an dieser Stell Fuzz-Tests durchgeführt werden. Dabei werden absichtlich falsche oder zufällige Werte in die Applikation eingepflegt, um das Verhalten der Applikation zu überprüfen und mögliche Abstürze oder Fehlverhalten herbeizuführen, die ein späterer Angreifer ausnützen könnte.[9]

6.0.6 6. Release

Neue Angriffsszenarien werden im Laufe der Zeit auf die Applikation einwirken. Es ist wichtig, darauf vorbereitet zu sein. Abhilfe schafft hier bereits ein früh implementierter Incident Response Plan. Darin soll definiert werden, wie auf neu auftretende Gefährdungen reagiert wird und wer an unterschiedlichen Stellen im Programm für daraus resultierende Anpassungen verantwortlich ist. Außerdem muss festgelegt werden, wer bei eventuellen Partnern der Ansprechpartner in solchen Belangen ist und wann mit diesem in Kontakt getreten werden muss. Das abschließende Security Review kann möglicherweise auch an externe Firmen ausgelagert werden, um einen neuen Blickwinkel auf die Applikation und die damit verbundene Gefährdungslage zu bekommen. Aufgrund der vorab festgelegten Bug Bars und Quality Gates kann genau definiert werden, ob aus Sicherheitssicht die Software publiziert werden kann oder ob es nötig ist, noch Veränderungen und Anpassungen vorzunehmen.
Der letzte Schritt, bevor die Software publiziert werden kann, sollte noch die Archivierung des Source Codes und aller anderen Dokumente sein, die in diesem Prozess entstanden sind. Das dient sowohl der Nachvollziehbarkeit als auch der Dokumentationspflicht.[9]

6.0.7 7. Response

Dieser Schritt erfolgt bereits zur ‘Lebenszeit’ des Produktes. Hier muss entsprechend der im Vorschritt festgelegten Response Pläne gearbeitet werden. Dies gewährleistet, dass die Software auch im Laufe der Zeit aktuell und sicher bleibt. Die Pläne müssen gewartet werden um eventuell Verbesserungen vornehmen zu können und am Zahn der Zeit zu bleiben.[9]

6.0.8 Bewertung

Bereich	Bewertung	Bemerkung
Logging	0	
Authentifizierung und Autorisierung	0	
Kryptographie	0	
Session Management	0	
Input Validation	0	
Output Escaping	0	
Datenbank Sicherheit	0	
Fehler Management	0	
Aktualität	4	Wird laufend aktualisiert
Summe:	4	

Tabelle 6.1: Bewertung Microsoft SDLC

7 Ranking

Alle Standards und Publikationen, die innerhalb der Arbeit analysiert und bewertet wurden, werden im folgendem Ranking der Reihe nach angeführt. Die Reihung erfolgt absteigend, gemessen an der innerhalb des Bewertungsschema vergebenen Punktezahl:

Reihung	Publikationsname	Punkte
1.	CERT - Secure Coding Standard	29
2.	OWASP - Secure Coding Practices v2	24
3.	Java Secure Coding Guideline	22
4.	ISO 27002	15
5.	IT-Grundschutz	11
6.	ÖNORM A7700	10
7.	Microsoft Secure Development Lifecycle	4
8.	NIST 800-64 Rev. 2	2
9.	ISO 27001	3
10.	ISO 27034	2

Tabelle 7.1: Ranking

8 Secure Coding Guideline

Aus den bearbeiteten Standards und Publikationen und einigen weiteren Quellen wird im Folgenden eine Secure Coding Guideline abgeleitet. Die Guideline ist speziell für die Programmierung einer Server-Client Applikation in Java angepasst.

8.1 Aufbau einer Guideline

Wichtig für das Verständnis von Vorgaben ist die Art der Aufbereitung. Klare Regeln und Standards sind immer dann erforderlich, wenn entweder viel Geld von Erfolg oder Misserfolg abhängt oder Menschenleben gefährdet sein können, so wie es beispielsweise in der Medizin der Fall ist. Viele Regelungen zum Aufbau von Guidelines kommen daher aus diesem Bereich.

Im Allgemeinen ist es wichtig, dieselbe Sprache wie das Zielpublikum der Regelungen, zu sprechen. Dies bezieht sich sowohl auf die Sprache als auch auf die Wortwahl. Beides muss korrekt gewählt sein, damit eine Nachricht richtig von der Empfängerin oder dem Empfänger aufgenommen und verarbeitet werden kann. Es bietet sich an, kurze, einfache Sätze zu verwenden. Außerdem ist es oft einfach, Inhalte mittels Grafiken zu visualisieren. [34]

Jedes Kapitel, wie auch die Guideline an sich, soll mit einer kurzen Einführung beginnen in der die Wichtigkeit und damit verbunden auch die Verbindlichkeit des Schriftstückes in den Vordergrund gebracht werden.
Außerdem muss am Beginn der Guideline kommuniziert werden, dass es sich um ein lebendes Dokument handelt, und laufende Aktualisierungen nötig sind. Diese Aktualisierungen sollten in einem Register geführt werden und die Guideline mit einer Versionsnummer versehen sein. So kann Missverständnissen entgegengewirkt werden. Auf diese Formalitäten wird in der folgenden Beispielguideline verzichtet, da die Gepflogenheiten des Unternehmens fortgeführt werden und Anforderungen aus dem Qualitätsmanagement starken Einfluss nehmen (Kopf- und Fußzeilen, Formatierung, Register,...). Daher erscheint eine beispielweise Umsetzung nicht zielführend.

Weiters ist es unerlässlich, der Leserin oder dem Leser nicht nur Regeln zu liefern sondern auch eine Erklärung, um den Sinn des Mehraufwands, der durch diese Regelungen entstehen könnte, erfassen zu können. Das erleichtert nicht nur das Verständnis der Regeln sondern stellt auch eine Awarenessmaßnahme dar. Außerdem ist es ratsam, viele Beispiele mit in die Richtlinie einzubauen. Das ermöglicht es, auch schwierige Regelungen zu erfassen und zu verstehen.[35][36]

Aufgrund dieser Vorgaben wurde für die Secure Coding Guideline folgende Form gewählt: Das Dokument beginnt mit einer Einführung. Danach werden Themen der Reihe nach abgehandelt. Jedes Thema weißt dabei folgenden Aufbau auf:

1. Einführung
2. Regelungen
3. Tools (optional)
4. Beispiele (Dos and Don't)
5. Kontrollfragen

Dieser Aufbau soll es ermöglichen, dass das Dokument leicht verstanden und umgesetzt werden kann. Die Werte, die variiert werden können, sind mit State-of-the-Art Werten befüllt. Eine Fußnote erklärt, welche Varianten die Sicherheit steigern oder senken könnten.

8.2 Einleitung

Sicherheit von Daten und Informationen wird nicht nur zum Schutz von personenbezogenen Daten von der DSGVO (GDPR) gefordert, sondern gewinnt für Unternehmen immer mehr an Bedeutung, um die jeweiligen Erfolgsgeheimnisse zu schützen und so konkurrenzfähig zu bleiben. Denkt man an die unterschiedlichen Stakeholder, so hat jeder andere Beweggründe, warum Informationssicherheit wichtig ist. Beispielsweise:

- Unternehmer - Will Daten geschützt wissen / Will Werte des Unternehmens wahren / Will Kosten senken / Will effiziente Arbeit ermöglichen / Will konkurrenzfähig bleiben
- Entwickler - Will qualitativ hochwertige Software liefern / Will effizient arbeiten
- Kunde/Anwender - Will die eigenen Daten geschützt wissen / Will einfach zu verwendende Software / Möchte kostengünstige Software
- Aufsichtsbehörde - Will den Stand der Technik umgesetzt haben

Diese Liste lässt sich wahrscheinlich endlos weiterführen und für jede Stakeholdergruppe wird es ein gerechtfertigtes Interesse geben, warum der sichere Umgang mit Informationen wichtig ist.

8.2.1 Parametrisierung

Da das Sicherheitsniveau von Software oft von den verwendeten Parametern für Regelungen abhängt wurden in der folgenden Guideline die Parameter, die variiert werden können, kursiv dargestellt. Als Fußnote findet sich eine Beschreibung, wie sich das Sicherheitsniveau ändert wenn der Parameter angepasst wird.

8.2.2 Generelle Ratschläge

- Setzen auf Bestehendes und Getestetes (Die Verwendung von Tools spart viel Zeit - außerdem sind diese oftmals ausgiebig getestet)
- Verwendung von bestehenden Strukturen (Beispielsweise LDAP oder bestehende Richtlinien)

8.3 Secure Coding Guideline

8.3.1 Gültigkeit

Die folgende Guideline wurde für eine javabasierte Server-Client Applikation entworfen. Das Regelwerk soll als Grundlage dienen, um sichere Software zu schreiben. Behandelt werden nur die Themengebiete 'Development' und 'Operation'. Dadurch hält man sich an die typischen Inhalte von Secure Coding Guidelines. Architekturthemen werden beabsichtigt nicht behandelt, um die Anwendbarkeit der Guideline nicht einzuschränken.

8.3.2 Logging

Einführung

Logging (systemgenerierte Nachrichten) ist unerlässlich um Ursachen für Systemfehler finden zu können, beziehungsweise um das Verhalten des Systems zu dokumentieren. Außerdem werden Logs verwendet, um Handlungen von Benutzerinnen und Benutzern aufzuzeichnen. Dies ist oft nötig, um Aktionen nachweisbar zu machen. Dadurch ergibt sich natürlich auch der Schutzbedarf dieser Daten. Entscheidend ist nicht die Menge der Daten, sondern die Qualität. Da es oft nicht nötig ist, personenbezogene beziehungsweise sensible Daten in Logfiles aufzunehmen, muss weitgehenst darauf verzichtet werden. Außerdem müssen diese Daten aus Datenschutzgründen so früh als möglich gelöscht werden. [3][37][38, pp.1315-1317]

Regelungen

- Für Loggingfunktionalitäten ist die Klasse java.util.logging zu verwenden. Alternativ kann auch ein Tool verwendet werden - siehe Empfehlungen in diesem Kapitel.
- Logeinträge müssen folgendem Aufbau entsprechen[3]
  - Zeitstempel - Wird bereits von der Logging-API zur Verfügung gestellt.(Zeitzonen beachten)
  - Loglevel (Config, Info, Warning, Severe, ...)
  - Markierung für sicherheitsrelevante Log-Informationen: <SECURITY>
  - Identifikation des Auslösers (zum Beispiel: Account bei dem das Event erzeugt wurde)
  - Event-Erfolg (success/failure)
  - Beschreibung des Events

Es muss gewährleistet sein, dass zumindest folgende Events in Logfiles erfasst und mit einer Security-Markierung versehen werden[3][5][pp. 36-37]:

  - Validierungsfehler
  - Erfolgreiche und fehlgeschlagene Anmeldeversuche
  - Administrative Eingriffe (sowohl server- als auch clientseitig)
  - Erfolgreiche und fehlgeschlagene Passwortwechsel
  - Alle Applikationsfehler
  - Fehler aus Verschlüsselungsmodulen

- Logs dürfen nicht ausführbar sein! [3]
- Logs müssen vor unbefugten Zugriffen geschützt werden. [3]
- Logs dürfen nur solange vorgehalten werden wie nötig (Compliance-Vorgaben, ...)[39]
- Personenbezogene und sensible Daten dürfen nur dann in Logs gespeichert werden, wenn dies unbedingt nötig ist, und es eine Grundlage dafür gibt (gemäß den geltenden Datenschutzvorgaben). Ebenso muss die Aufbewahrungszeit angepasst werden - beispielweise zur Wahrung von Dokumentations- und Aufbewahrungspflichten. [39]
- Die Integrität der Logs muss beispielsweise mittels Hashes und Prüfsummen kontrollierbar sein. Noch besser wäre es, die Logs auch in einer zentralen Logmanagement Lösung zu verwalten, da sie dort vor Veränderung besser geschützt werden können.[3][40]
- Nur überprüfter User-Input darf in Logfiles gespeichert werden (wenn dies nötig ist). Siehe Kapitel 8.3.6.[37]
- Logs dürfen nicht in Verzeichnissen eines Webservers selbst, oder in Verzeichnissen, die von einem solchen ausgeführt werden, gespeichert werden. Logs können sonst schnell vom Webserver ausgeführt beziehungsweise angezeigt und so zur Gefahr werden.[40]

Tools

Eines der bekanntesten Tools um Logging in einer Java-Applikation umzusetzen ist Log4J.[41] Es gibt jedoch eine Vielzahl an Alternativen:

- Apache Commons Logging [42]
- SLF4J [43]

Beispiele

Logging Mechanismus

Falsch

Hier wird der Fehler auf Standard-Error geschrieben, das entspricht nicht den Vorgaben und ist damit falsch. Durch die Verwendung von STDERR kann nicht gewährleistet werden, dass Severity-Levels vergeben werden und Logs entsprechend geschützt werden. [44]

```
try {
} catch (SecurityException se) {
  System.err.println(se);
}
```

Richtig

Korrekt sind Logs so zu erzeugen:

```
try {
} catch(SecurityException se) {
  logger.log(Level.SEVERE, se);
}
```

Validierte Benutzereingabe

Falsch

Die Eingabe der Benutzerin oder des Benutzers wird hier direkt aus der Eingabe in das Logfile geschrieben. Außerdem ist der Benutzername personenbezogen und darf nur in Ausnahmefällen aufgezeichnet werden. [45].

```
if (loginNotSuccessful) {
  logger.info("User login failed for: " + username);
}
```

Richtig

Die Funktion müsste korrekterweise so aussehen, wobei die Funktionalität von sanitize(Input) sowohl die Datenschutzbelangen löst, als auch die Filterung auf Zeichen, die eventuell nicht zulässig sind[1] [45]

```
if (loginNotSuccessful) {
  logger.severe("User login failed for: " + sanitizeUser(username));
```

[1] Diese Zeichen sollten in einer Whitelist erfasst werden, wie im Beispiel ersichtlich. Wenn dies nicht möglich ist müssen zumindest Blacklists verwendet, und so Zeichen verboten werden, die für Logeinträge nicht nötig sind.

```
}

public String sanitizeUser(String username) {
  return Pattern.matches("[A-Za-z0-9_]+", username))
     ? username : "unauthorized user";
}
```

Kontrollfragen

Kontrollfrage	JA / NEIN
Werden alle wichtigen System- und Useraktionen in Logs gespeichert?	
Sind alle Logs als ‘nicht ausführbar’ markiert?	
Wird der beschriebene Aufbau von Logeinträgen konsistent eingesetzt?	
Wird soweit möglich auf die Speicherung von personenbezogenen und/oder sensiblen Daten verzichtet?	
Ist der Schutz der Logs vor Manipulation, Löschung oder Diebstahl der Daten (CIA) gegeben?	
Ist die Integrität der Daten nachweisbar?	
Ist gewährleistet, dass kein Userinput direkt in Logs gespeichert wird?	

8.3.3 Authentifizierung und Autorisierung

Einführung

Authentifizierungsmethoden sind unabdingbar, um nur berechtigten Benutzerinnen und Benutzern Zutritt zum System zu gewähren. Wenn möglich ist es sowohl einfacher als auch sicherer, bereits vorhandene Strukturen und Systeme zu verwenden, wie beispielsweise eine Authentifizierung über ein Active Directory (Single Sign On).[3][46]

Regelungen

- Eine Authentifizierung muss für alle Programmteile vorausgesetzt werden, außer für jene, die explizit als 'öffentlich' deklariert sind.[3][p. 6]
- Authentifizierung muss immer serverseitig umgesetzt werden und darf nie clientabhängig sein.[3][p. 6]
- Es muss ein rollenbasiertes Berechtigungsmanagement Anwendung finden. [3][pp. 6.7]
- Passwörter dürfen nur als kryptographisch starke und gesalzene Hashes gespeichert werden.[3][p. 6][37][5][p. 37]
- Wird das Passwort via HTTPS übermittelt, so darf nur die POST Methode verwendet werden. Die unverschlüsselte Übertragung via HTTP ist ausgeschlossen.[3][p. 6]
- Bei einer fehlgeschlagenen Authentifizierung dürfen nur generische Fehlermeldungen erzeugt werden.[3][p. 6]
- Wenn dieselbe Session-ID länger als *zwölf*[2] Stunden verwendet wird, ist eine neuerliche Authentifizierung nötig.[47]
- Nach *fünf*[3] fehlgeschlagenen Anmeldeversuchen ist der Benutzeraccount für *15 Minuten*[4] zu sperren.[3][p. 6][48]
- Es müssen Möglichkeiten existieren, um das Passwort neu zu generieren. Möglich ist beispielsweise der Versand eines Links zur Neuvergabe eines Passwortes über eine vorab hinterlegte E-Mailadresse oder Versand eines Einmalpasswortes per SMS über eine hinterlegte Telefonnummer beziehungsweise ebenso per E-Mail.[3][pp.6-7][49][50]

[2] Wenn dieser Parameter gesenkt wird, steigt das Sicherheitsniveau
[3] Wird dieser Wert erhöht wird das Sicherheitsniveau gesenkt, eine Senkung des Wertes erhöht das Sicherheitsniveau marginal
[4] Dieser Wert kann angepasst werden. Ein Senken des Wertes senkt auch das Sicherheitsniveau und umgekehrt

- Es muss ein Timeout eingeführt sein, welches eine Benutzerin oder einen Benutzer bei Inaktivität automatisch abmeldet. Das Timeout sollte so kurz wie möglich sein.[5][3][p. 6]
- Das Passwort darf während der Eingabe nicht im Klartext sichtbar sein. [5][p.36]
- Die gültige Authentifizierung muss bei jedem Aufruf überprüft werden. [3][p. 7]
- Der Zugriff einer jeden Benutzerin oder eines jeden Benutzers muss soweit wie möglich eingeschränkt sein. Jeder darf nur die Rechte haben die sie oder er benötigt. Es sollte ein entsprechend granulares Berechtigungskonzept in der Applikation vorgesehen sein.[33][3][p. 7] [5][pp. 30-37]
- Der Zugriff auf Systemfunktionen muss für nicht-administrative Benutzerinnen und Benutzer gesperrt sein. [5][p. 36]
- Für Benutzeraccounts müssen Passwörter folgende Anforderungen erfüllen[3][p. 6]:
  - Mindestlänge: *16* Zeichen[6]
  - Muss folgende Zeichen beinhalten[51][47][3][p. 6]
    - Großbuchstaben
    - Kleinbuchstaben
    - Ziffern
    - Sonderzeichen
  - Mindestalter: *ein*[7] Tag
  - Gültigkeitsdauer: *drei*[8] Monate
  - Passworthistorie: *20*[9] Passwörter
  - Der Benutzername, oder Teile (ab *4*[10] Zeichen) davon dürfen nicht im Passwort vorkommen

[5] Da dieser Wert stark von der Art der Anwendung abhängt, kann hier keine Empfehlung abgegeben werden. Jedenfalls sollte das Timeout so kurz als möglich gesetzt werden.

[6] Absolute Untergrenze sind momentan 8 - 16 Zeichen werden aber empfohlen und sind damit 'Stand der Technik'. Kürzere Passwörter senken das Sicherheitsniveau

[7] Eine Passwortrichtlinie ohne Mindestalter würde das Sicherheitsniveau drastisch senken, da es so einer Benutzerin oder einem Benutzer ermöglicht wird, dasselbe Passwort zu behalten, indem sie oder er nach Ablauf der Gültigkeitsdauer das Passwort ausreichend oft ändert bis die Passworthistorie überschritten ist und dann das gewünschte Passwort wieder wählt. Eine Erhöhung des Wertes hebt das Sicherheitsniveau nur marginal.

[8] Eine Senkung dieses Parameters steigert das Sicherheitsniveau - eine Erhöhung senkt das Sicherheitsniveau

[9] Eine drastische Senkung dieses Wertes senkt das Sicherheitsniveau, wobei eine Erhöhung nur eine minimale Steigerung des Sicherheitsniveaus hervorruft

[10] Wird dieser Parameter gesenkt, sinkt das Sicherheitsniveau ebenso, umgekehrt steigt es etwas, wenn der Wert erhöht wird.

- Das Passwort muss gegen ein Dictionary (mögliche Dictionaries finden sich beispielsweise unter [52]) beziehungsweise gegen Listen oft verwendeter Passwörter[11] geprüft werden (darf darin nicht vorkommen)

Beispiele

Wenn kein Tool beziehungsweise eine Bibliothek oder dergleichen verwendet wird sondern die Authentifizierung neu umgesetzt werden muss, liefert folgendes Beispiel wichtige Anhaltspunkte:[53]

```
import java.security.GeneralSecurityException;
import java.security.SecureRandom;
import java.security.spec.KeySpec;
import javax.crypto.SecretKeyFactory;
import javax.crypto.spec.PBEKeySpec;

final class Password {
 private SecureRandom random = new SecureRandom();

 /* Setzt das Passwort neu */
 void setPassword(char[] pass)
    throws IOException, GeneralSecurityException {
  byte[] salt = new byte[12];
  random.nextBytes(salt);
  saveBytes(salt, "salt.bin");
  byte[] hashVal = hashPassword( pass, salt);
  saveBytes(hashVal,"password.bin");
  Arrays.fill(hashVal, (byte) 0);
 }
 /* Ueberprueft ob das bereitgestellte Passwort korrekt ist */
 boolean checkPassword(char[] pass)
    throws IOException, GeneralSecurityException {
  byte[] salt = loadBytes("salt.bin");
  byte[] hashVal1 = hashPassword( pass, salt);
  // Laedt den Hashwert aus password.bin
  byte[] hashVal2 = loadBytes("password.bin");
  boolean arraysEqual = timingEquals( hashVal1, hashVal2);
  Arrays.fill(hashVal1, (byte) 0);
```

[11] Solche Listen werden beispielsweise von Herstellern von Sicherheitsprodukten veröffentlicht

```
    Arrays.fill(hashVal2, (byte) 0);
    return arraysEqual;
  }

  /* Verschluesselt gesalzenes Passwort */
  private byte[] hashPassword(char[] pass, byte[] salt)
      throws GeneralSecurityException {
    KeySpec spec = new PBEKeySpec(pass, salt, 65536, 128);
    Arrays.fill(pass, (char) 0);
    Arrays.fill(salt, (byte) 0);
    SecretKeyFactory f =
        SecretKeyFactory.getInstance("PBKDF2WithHmacSHA256");
    return f.generateSecret(spec).getEncoded();
  }
  /**
   * Ueberprueft ob beide Arrays gleich gross sind,
   * benoetigt aber die selbe Zeit, egal ob sie gleich- oder
      unterschiedlich lang sind.
   * So werden Timing Attacks verhindert.
   */
  public static boolean timingEquals(byte b1[], byte b2[]) {
    boolean result = true;
    int len = b1.length;
    if (len != b2.length) {
      result = false;
    }
    if (len > b2.length) {
      len = b2.length;
    }
    for (int i = 0; i < len; i++) {
      result &= (b1[i] == b2[i]);
    }
    return result;
  }

}
```

Kontrollfragen

Kontrollfrage	JA / NEIN
Ist die Interaktion mit nicht-öffentlichen Bereichen erst nach erfolgreicher Authentifizierung möglich?	
Kann ausgeschlossen werden, dass die Authentifizierung umgangen werden kann?	
Findet die Authentifizierung ausschließlich am Server statt?	
Werden Passwörter nur gehasht und gesalzen gespeichert?	
Werden sensible Daten wie Benutzername und Passwort ausschließlich über verschlüsselte Verbindungen übertragen?	
Werden ausschließlich generische Fehlermeldungen ausgegeben?	
Kann sich die Benutzerin oder der Benutzer selbstständig abmelden und geschieht dies nach Inaktivität automatisch?	
Werden sichere Möglichkeiten geboten um ein neues Passwort zu wählen?	
Wird der Account nach Falschanmeldungen gesperrt?	
Werden die gültigen Passwortrichtlinien erzwungen?	
Wird die gültige Authentifizierung bei jeder Kommunikation mit dem Server überprüft?	
Ist der Zugriff auf Systemressourcen gesperrt?	
Ist ein rollenbasiertes Berechtigungsmanagement eingeführt?	

8.3.4 Kryptographie

Einführung

Kryptographie - also die Lehre der Verschlüsselungstechnik - ist unersetzbar wichtig für die Informationssicherheit. Dabei handelt es sich jedoch um ein Umfeld, in dem aktiv Forschung betrieben wird. Daher ist die laufende Evaluierung von Methoden oder Algorithmen nötig, um am Stand der Technik zu bleiben.[54]
Es gilt folgender Leitsatz:

Kryptographie soll nicht neu erfunden werden - Verwende bestehende Funktionen und Algorithmen!

Regelungen

- **Zufallszahlen**: Diese müssen mittels der Klasse java.security.SecureRandom erzeugt werden.[55]
- **Hashing**: Es dürfen ausschließlich folgende Hashingfunktionen verwendet werden: SHA-256, SHA-384 und SHA-512, SHA-3-256 und SHA-3-512[56, p. 40]
- **Encryption**: Die Verschlüsselung muss auf sichere Algorithmen setzen. Man unterscheidet zwischen symetrischer und asymetrischer Kryptographie:[56, pp.23-26][57][58]
  - **Asymetrisch**: Asymetrische Kryptographie - also die Verwendung von unterschiedlichen Schlüsseln zur Ver- und Entschlüsselung von Information - wird beispielsweise zum Schlüsselaustausch verwendet. Dieser muss via Diffie-Hellmann-Algorithmus erfolgen, wahlweise unter Miteinbeziehung von elliptischen Kurven.
  - **Symetrisch**: Da symetrische Kryptographie - also die Verwendung von identen Schlüsseln zur Ver- und Entschlüsselung von Informationen - viel performanter ist, wird sie zur Verschlüsselung der Daten verwendet. Dies muss mittels AES256 oder AES192 passieren. Dabei werden die Modi GCM und CBC unterstützt.
- **SSL/TLS**: SSL darf keine Anwendung in der Verschlüsselung von Übertragung finden. Es werden nur die Protokolle TLS 1.2 und 1.3 unterstützt. (TLS 1.2 ist momentan State-of-the-Art - zukünftig wird jedoch auf TLS 1.3 gesetzt werden, deshalb sollen beide implementiert werden)[56][59]
- Wenn technisch umsetzbar sind ausschließlich folgende Modi zur Verschlüsselung zu wählen [60]:
  - TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA
  - TLS_DHE_RSA_WITH_AES_128_CBC_SHA

Dadurch wird Perfect Forward Secrecy unterstützt, was vor Angriffen schützt, bei dem der Inhalt von Nachrichten im Nachhinein entschlüsselt werden soll.

Beispiele

Erstellen von Zufallszahlen

Falsch

Hier wird die Funktion java.util.Random verwendet, weil die Ergebnisse vorhersehbar sein können.

```
import java.util.Random;

Random number = new Random(123L);

for (int i = 0; i < 10; i++) {
  // Generiert Zufallszahlen im Bereich [0, 20]
  int n = number.nextInt(21);
  System.out.println(n);
}
```

Richtig

```
import java.security.SecureRandom;
import java.security.NoSuchAlgorithmException;
// ...

public static void main (String args[]) {
  SecureRandom number = new SecureRandom();
  // Generiert Zufallszahlen im Bereich [0, 20]
  for (int i = 0; i < 20; i++) {
    System.out.println(number.nextInt(21));
  }
}
```

Hashing SHA-256

Richtig Das folgende Beispiel zeigt eine mögliche Implementierung eines SHA-256 Hashes [61]

```
MessageDigest digest = MessageDigest.getInstance("SHA-256");
byte[] encodedhash = digest.digest(
  originalString.getBytes(StandardCharsets.UTF_8));
```

Java Client HTTP-Verbindung unter Verwendung von TLS 1.2

Clientseitig muss die Applikation mit den Parametern:

```
 -Dhttps.protocols=TLSv1.2 -Djdk.tls.client.protocols=TLSv1.2
```

aufgerufen werden. Alternativ kann auch in der Applikation folgender Aufruf verwendet werden:

```
CloseableHttpClient client = HttpClientBuilder.create()
.setSSLSocketFactory(new
    SSLConnectionSocketFactory(SSLContext.getDefault(),
new String[] { "TLSv1.2" }, null,
    SSLConnectionSocketFactory.getDefaultHostnameViewer())).build();
```

Kontrollfragen

Kontrollfrage	JA / NEIN
Werden ausschließlich angeführte Algorithmen verwendet?	
Wird zur Generierung von Zufallszahlen ausschließlich die Funktion java.security.SecureRandom verwendet?	

8.3.5 Session Management

Einführung

Wie bereits in vorangegangenen Kapiteln erwähnt müssen sensible Daten, wie beispielsweise Benutzername und Passwort gesichert übertragen werden. Dieser Blickwinkel muss in diesem Kapitel sogar noch ausgeweitet werden. Denn um Anwendungen komfortabler zu gestalten, werden Methoden eingesetzt, die eine Neuanmeldung mittels beispielsweise Benutzername und Passwort, so selten wie möglich nötig machen. So werden beispielsweise Session-IDs (oftmals JSESSIONID) vergeben um Benutzerinnen und Benutzer mit einer aktiven Sitzung voneinander trennen zu können, beziehungsweise um vor jedem Aufruf überprüfen zu können, ob die oder der Aufrufende dazu berechtigt ist. Das Passwort muss so nur einmal, und nicht bei jedem Aufruf eingegeben werden.
Um solche oder ähnliche Funktionalitäten sicher umsetzen zu können sind folgende Punkte einzuhalten:

Regelungen

- Soweit nicht wichtige Gründe dagegen sprechen (Vorgaben des Managements oder Ähnliches), müssen Session Management Frameworks (Tools) verwendet werden - Eigenentwicklungen sind oft fehleranfällig - die Entscheidung über den Tooleinsatz sollte vor Entwicklungsbeginn erfolgen.[3][pp. 7-8]
- Session-IDs müssen auf dem Server (nicht auf dem Client) erstellt werden.[3][pp. 7-8]
- Die Session-ID muss mindestens *30*[12] Byte lang sein und unter Zuhilfenahme der im Kapitel 8.3.4 erwähnten Funktion java.security.SecureRandom erzeugt werden.[3][pp. 7-8]
- Wenn Cookies zum Session Management eingesetzt werden, müssen diese mit dem Secure- und dem Httponly-Flag versehen sein. Außerdem sollte im Cookie das Attribut SameSite ‘strict’gesetzt, um den Abfluss von Daten zu verhindern.[62][63][pp.95-97]
- Es muss einfach möglich sein, sich von der Applikation abzumelden. - Dadurch wird die Möglichkeit für Session-Hijacking - also das übernehmen einer aktiven Session - eingeschränkt.[3][pp. 7-8]
- Immer wenn eine Benutzerin oder ein Benutzer sich authentifiziert, muss eine neue Session-ID generiert und vergeben werden.[3][pp. 7-8]

[12] Eine Erhöhung des Parameters sorgt für eine minimale Steigerung des Sicherheitsniveau wobei eine Senkung viele Angriffe erleichtert und damit das Sicherheitsniveau senkt

- Bei Inaktivität muss nach *20*[13] Minuten die Session automatisch terminiert werden.[3][pp. 7-8]
- Die Session-ID darf nicht in der URL ersichtlich sein, darf also nur via HTTP-POST übertragen werden. Es wird jedoch empfohlen, die SessionID in einem Cookie zu übertragen.[64][3][pp. 7-8]
- Bei der Übertragung von Cookies oder anderen sensiblen Daten muss eine verschlüsselte Übertragung implementiert sein. Empfohlen wird die Verwendung von HSTS.[3][pp. 7-8][63][pp.95-97]
- Session-IDs müssen einzigartig sein und mit einer hohen Entropie erstellt werden.[3][pp. 7-8][63][pp.95-97]

Tools

Es gibt viele, auch proprietäre Tools, um unter anderem Session-Management in Java umzusetzen (Identity-Management). Genauso gibt es aber auch Tools, die kostenfrei einsetzbar sind, wie beispielsweise:

- Apache Shiro[65]
- Enterprise Security API (OWASP) [66]
- Spring Security[67]

Beispiele

Cookie Config Beispiel-Konfiguration eines Tomcat Servers ab Version 7 für die Verwendung von Secure, HTTP-Only Cookies. Folgender Code muss in das File web.xml im Bereich 'session-config' eingefügt werden:

```
<session-config>
   <cookie-config>
      <http-only>true</http-only>
      <secure>true</secure>
   </cookie-config>
 </session-config>
```

[13] Wird der Parameter drastisch gesenkt, so steigt das Sicherheitsniveau der Applikation minimal, eine Steigerung senkt das Sicherheitsniveau minimal

Kontrollfragen

Kontrollfrage	JA / NEIN
Wird beim Login die eventuell bestehende, alte Session-ID verworfen und eine neue generiert?	
Werden Cookies wie Anmeldedaten behandelt und wird daher sichergestellt, dass sie nur über eine verschlüsselte Verbindung übertragen werden?	
Wird ein Tool zum Session-Management verwendet?	
Werden Session-IDs nur via HTTP Post beziehungsweise als Cookie übertragen?	
Kann sich eine Benutzerin oder ein Benutzer einfach abmelden?	

8.3.6 Input Validierung

Einführung

Viele Sicherheitslücken entstehen durch die fehlerhafte Validierung von Daten. Das hängt sicherlich nicht zuletzt damit zusammen, dass die Eingabe von fehlerhaften oder 'bösartigen' Daten einfach und ohne erweitertem Wissen möglich ist, da es Datenfelder gibt, die für alle Benutzerinnen und Benutzer zur Verfügung stehen müssen um die Interaktion zu ermöglichen. Ungeachtet der Erreichbarkeit von Eingabemöglichkeiten ist es unerlässlich, jeden Input zu validieren.

Regelungen

- Es muss eine zentrale, serverseitige Funktion zum Validieren von Inputs geben, die für alle Eingabefelder zur Anwendung kommt.[3][p. 5][63][pp.54-61]
- Eine Validierung kann auch clientseitig erfolgen, muss aber serverseitig wiederholt werden. Eine clientseitige Überprüfung dient nur der Usability und der Entlastung der Serverinfrastruktur.[63][pp.54-61]
- Bei manchen Eingabefeldern kann es jedoch sein, dass eine eigene Überprüfungsroutine nötig ist, wie beispielsweise bei E-Mailadressen (besonderer Aufbau) oder bei Fileuploads (hier ist nicht nur die Filegröße sondern auch der Datentyp und die Überprüfung auf Viren vor der Verarbeitung wichtig). Außerdem muss die Ausführung hochgeladener Files verhindert werden.[3][p. 5][63][pp.54-61]
- Die Verarbeitung einer Eingabe darf erst nach erfolgter Validierung erfolgen.[3][p. 5][63][pp.54-61]
- Es wird ein *Whitelisting*[14]-Ansatz gewählt - alle erlaubten Zeichen werden festgelegt.[3][p. 5][63][pp.54-61]
- Es empfiehlt sich, vor der Validierung die Daten zu konvertieren beziehungsweise zu typisieren, sodass Validierungsfehler eingegrenzt werden können.[3][p. 5]
- Passwortfelder können von der Validierung ausgenommen werden, da der Inhalt nur gehasht gespeichert wird und daher die eigentliche Eingabe nicht relevant ist.[63][pp.54-61]

[14] Alternativ kann auch ein Blacklisting-Ansatz angewendet werden. Dies ist zwar einfacher - senkt aber potenziell das Sicherheitsniveau. Das Ausmaß ist nicht abschätzbar, da es stark von den Listen selbst abhängig ist

- Wird der Input in einem SQL-Statement verwendet, muss immer ein Prepared-Statement Anwendung finden. Alternativ kann auch mit Stored Procedures gearbeitet werden.[3][p. 5][63][pp.54-61]
- Für jedes Eingabefeld muss außerdem eine maximale Länge definiert werden. Diese ist natürlich abhängig vom geplanten Inhalt des Datenfeldes.[3][p. 5]
- Alle Eingaben, unabhängig davon ob sie von einem anderen Service, einer externen Datenbank, einer Eingabe einer Benutzerin oder eines Benutzers kommen oder Ähnliches müssen gleich behandelt und damit validiert werden.[63][pp.54-61]
- Eingaben, welche die Validierung negativ ablegen, müssen verworfen und dürfen nicht weiterverarbeitet werden. - Das Ereignis soll jedoch in Logs aufgezeichnet werden, um das Verhalten des Systems nachvollziehen zu können.[3][p. 5][63][pp.54-61]

Tools

Auch für die Input Validation können Tools verwendet werden. Zwei davon sind[63][p.62]:

- OVal Framework[68]
- Apache Common Validator[69]

Beispiele

Prepared Statement Eine beispielhafte Implementierung eines Prepared Statements:[70]

```
PreparedStatement pstmt = con.prepareStatement("UPDATE EMPLOYEES
                          SET SALARY = ? WHERE ID = ?");
pstmt.setBigDecimal(1, 153833.00)
pstmt.setInt(2, 110592)
```

Inputvalidierung mit Regex Die Validierung eines Datum könnte wie folgt gestaltet werden.[71]

```
class GregorianDateMatcher implements DateMatcher {

   private static Pattern DATE_PATTERN = Pattern.compile(
     "^((2000|2400|2800|(19|2[0-9](0[48]|[2468][048]|[13579][26])))-02-29)$"
     + "|^(((19|2[0-9])[0-9]{2})-02-(0[1-9]|1[0-9]|2[0-8]))$"
     +
         "|^(((19|2[0-9])[0-9]{2})-(0[13578]|10|12)-(0[1-9]|[12][0-9]|3[01]))$"
```

```
      + "|^(((19|2[0-9])[0-9]{2})-(0[469]|11)-(0[1-9]|[12][0-9]|30))$");

   @Override
   public boolean matches(String date) {
      return DATE_PATTERN.matcher(date).matches();
   }
}
```

Negativ Beispiel ohne Validierung In folgendem Beispiel ist ersichtlich, wie ohne Validierung Daten verarbeitet werden können. Das Beispiel dient als Negativbeispiel:[72]

```
class DirList {
  public static void main(String[] args) throws Exception {
    String dir = System.getProperty("dir");
    Runtime rt = Runtime.getRuntime();
    Process proc = rt.exec(new String[] {"sh", "-c", "ls " + dir});
    int result = proc.waitFor();
    if (result != 0) {
      System.out.println("process error: " + result);
    }
    InputStream in = (result == 0) ? proc.getInputStream() :
                                     proc.getErrorStream();
    int c;
    while ((c = in.read()) != -1) {
      System.out.print((char) c);
    }
  }
}
```

Kontrollfragen

Kontrollfrage	JA / NEIN
Wird gewährleistet, dass jeder Input mit externer Herkunft serverseitig validiert wird?	
Wird vor der Validierung eine Konvertierung eingesetzt?	
Wird für Datenbankabfragen ein Prepared-Statement verwendet?	
Werden Eingaben, die nicht validiert werden können, verworfen?	
Wird ein Whitelisting-Ansatz für die Inputvalidierung verwendet?	

8.3.7 Output Escaping

Einführung

Das Output-Escaping soll verhindern, dass Daten, die aus dem System kommen, ausführbaren Code enthalten und so Gefahr von ihnen ausgeht. Damit gehört das Output-Escaping sowohl zu einer Security als auch zu einer Safety-Maßnahme. Große Bedeutung wird diesem Bereich zu Teil, wenn Ausgaben eines Systems in anderen Systemen in Form von Schnittstellen zur Verfügung gestellt und dort weiterverarbeitet wird.

Output-Escaping ist genauso wichtig wie Input Validierung[37]

Regelungen

- Unbedingt auf geprüfte Tools und geprüfte Routinen zurückgreifen - Diese sind meist gut getestet und bieten umfangreiche Funktionen.[3][p. 5][63][pp.66-67][33]
- Es dürfen nur überprüfte Ausgaben aus dem System erfolgen.[3][p. 5][63][pp.66-67]
- Output-Escaping ist kontextsensitiv (unbedingt auf mögliche Skript- und Programmiersprachen achten)[63][pp.66-67]
- Für Schnittstellen zu anderen Applikationen ist immer eine exakte Definition zu erstellen und zu beachten damit die Fehlerrate niedrig gehalten werden kann.

Tools

- Enterprise Security API (OWASP) [66] - Dort existieren Methoden, wie beispielsweise: encodeForHtml() oder encodeForCSS() die für das Escaping hilfreich sein können.
- Coverity Security Library [73]

Beispiele

Beispiel für implementiertes einfaches Output-Escaping:[74]

```
public class ValidateOutput {
  // Erlaubt nur alphanumerische Zeichen and Whitespaces
  private static final Pattern pattern =
      Pattern.compile("^[a-zA-Z0-9\\s]{0,20}$");

  // Validiert das Input-Feld anhand einer Whitelist
  public String validate(String name, String input) throws
      ValidationException {
    String canonical = normalize(input);

    if (!pattern.matcher(canonical).matches()) {
      throw new ValidationException("Improper format in " + name + " field");
    }

    // Fuehrt das Output-Encoding fuer nicht valide Zeichen durch
    canonical = HTMLEntityEncode(canonical);
    return canonical;
  }
}
```

Kontrollfragen

Kontrollfrage	JA / NEIN
Wurde ein Tool zum Output-Escaping verwendet?	
Kann sichergestellt werden, dass nur codierte Outputs ausgegeben werden?	
Existiert zu jeder Schnittstelle zu anderen Applikationen eine Definition und wird diese eingehalten und überprüft?	

8.3.8 Datenbank Sicherheit

Einführung

Datenbanken beherbergen meist das Herzstück von Applikationen, nämlich, wie der Name ohnehin vermuten lässt - Daten. Auch wenn die Sicherheit der Datenbank nur ein Randgebiet des Themas um die sichere Programmierung ist, so handelt es sich dabei oftmals um ein Einfallstor für Angreifer. Sowohl aus der Ecke des Datenschutzes als auch aus Sicherheitsgründen sind folgende Regeln einzuhalten.

Regelungen

- Applikationsbenutzer dürfen keine Datenbankbenutzer sein. Es müssen Serviceaccounts (Accounts, die von anderen Applikationen verwendet werden) mit sicheren Zugangsdaten konfiguriert werden (siehe Definition im Kapitel 8.3.3.[3]
- Es darf immer nur mit der kleinstmöglichen Berechtigung auf die Datenbank zugegriffen werden.[75]
- Wie bereits im Kapitel 8.3.6 beschrieben muss bei Zugriff auf Datenbanken auf Prepared-Statements zurückgegriffen werden. Alternativ kann auch die Klasse java.sql.CallableStatement verwendet werden.[3][76]
- Es sind Views zu erstellen, die Anwenderinnen und Anwender nur auf definierte Teile von Datenbanken Berechtigungen erteilen.[3]
- Verbindungen zur Datenbank müssen, sobald sie nicht mehr benötigt werden, geschlossen werden.[3]
- Bei der Einrichtung der Datenbank ist darauf zu achten, dass alle Standardcredentials ersetzt werden.[75][3]
- Datenbank-Connection-Strings dürfen weder hardgecodet sein noch unverschlüsselt gespeichert werden. Die Connection-Strings müssen verschlüsselt sein und dürfen erst zur Laufzeit entschlüsselt werden.[75][3][77]
- Zum Zugriff auf Datenbanken kann JDBC verwendet werden.[33]
- Wenn eine verschlüsselte Ablage von Credentials beziehungsweise des Connection-Strings nicht möglich ist, kann unter Einsatz von Middleware eine Alternative geschaffen werden.

Beispiele

Ein Beispiel zu Prepared-Statements findet sich bereits im Kapitel 8.3.6. Aufgrund dessen, dass viele der Regeln nicht direkt für den Code relevant sind, werden keine weiteren Beispiele benötigt.

Kontrollfragen

Kontrollfrage	JA / NEIN
Wurden alle Standardcredentials beim Einrichten der Datenbank geändert?	
Sind alle Connection-Strings auf vertrauenswürdigen Systemen verschlüsselt abgelegt?	
Werden ausschließlich Prepared-Statements für Datenbankabfragen verwendet?	
Sind Views eingerichtet, um nur Teildaten einsichtig zu machen?	
Werden alle Datenbankverbindungen geschlossen sobald sie nicht mehr benötigt werden?	

8.3.9 Fehler

Einführung

Fehler können immer auftreten - wichtig ist nur, ihnen entsprechend gegenüberzutreten. Fehler können jedoch auch von Angreiferinnen und Angreifern absichtlich erzeugt werden, um Informationen zu erlangen oder sich daraus ergebenden Schwachstellen auszunutzen. Um zu verhindern, dass diese Fehler kritische Auswirkungen auf die Funktionalität der Applikation und dessen Sicherheitsstandards haben, sind folgende Regeln zu beachten:

Regelungen

- Alle potenziellen Fehler müssen behandelt werden, sodass das Programm immer in einem konsistenten Zustand bleibt. [3]
- Alle auftretenden Fehler müssen in Form von Logeinträgen dokumentiert werden um nachvollzogen werden zu können.[3][63][pp.70-72]
- An Benutzerinnen und Benutzer dürfen nur generische Fehlermeldungen ausgegeben werden, da sonst sicherheitsrelevante Informationen enthalten sein können.[3]
- Alle Fehlerquellen müssen einzeln behandelt werden. Es darf nicht nur die Klasse “Throwable” behandelt werden, von der alle Exceptions erben, da jeder Fehler eine spezifische Behandlung benötigt.

Beispiele

Negativ Beispiel Im folgenden Beispiel wird der Stacktrace selbst ausgegeben, welches eine potenzielle Angreiferin oder einen potenziellen Angreifer mit Informationen versorgen kann:[78]

```
try {
  //...
} catch (IOException ioe) {
  ioe.printStackTrace();
}
```

Beispiel FileNotFoundException In diesem Beispiel wird der Benutzer so lange nach der Eingabe einer gültigen Datei gefragt, bis dies erfolgt ist. [78]

```
volatile boolean validFlag = false;
do {
  try {
    // Wenn das angegebene File nicht existiert --> IOException
    // Wenn das File existiert --> validFlag = true
    validFlag = true;
  } catch (IOException e) {
    // User wird nach einem anderen File gefragt
  }
} while (validFlag != false);
// File wird verwendet
  catch (IOException e) {
// Behandelt den Fall, dass das File zwischenzeitlich manipuliert wurde.
```

Kontrollfragen

Kontrollfrage	JA / NEIN
Werden alle möglichen Fehler behandelt?	
Ist für jeden möglichen Fehler gewährleistet, dass die Applikation in einem definierten Zustand bleibt und dass ein Logeintrag erzeugt wird?	
Werden der Benutzerin oder dem Benutzer nur generische Fehlermeldungen angezeigt?	

8.3.10 Libraries

Einführung

Wie bereits im Kapitel 8.3.3 erwähnt, ist es oftmals nicht nötig, das Rad neu zu erfinden. Es ist oft einfacher und sicherer auf bereits bestehende und geprüfte Verfahren zu setzen. Diese Tools können einzelne Aufgaben, wie in folgender Matrix ersichtlich, erleichtern. Die Verwendung dieser Tools ist nicht obligat und muss im Einzelfall entschieden werden.

Regelungen

- Die Verwendung von Libraries und Tools muss dokumentiert werden.
- Es ist dafür zu Sorgen, dass die verwendeten Libraries und Tools auf dem aktuellen Stand gehalten werden und wenn nötig von aktuellen Bibliotheken abgelöst werden.

Tool	Logging	Authentifizierung	Kryptographie	Session Management	Input Validation	Output Escaping	Datenbank Security	Fehler Management
OWASP Enterprise Security API [63] [66]	X	X	X	X	X	X		X
Coverity [63] [73]					X	X		
Apache Shiro [79] [65]		X	X	X				
Spring Security [79][63] [67]		X		X				
OACC [79] [80]		X						
Keycloak [79] [81]		X						
Keyczar [79] [82]			X					
Jasypt [79] [83]			X					
OVal Framework [68]					X			

Tabelle 8.1: Tools

8.3.11 Lesbarkeit

Einführung

Lesbarer Code ist zwar nur im weitesten Sinne relevant für die Sicherheit von Applikationen, dennoch stellt sie die Grundlage für die Wartbarkeit von Software dar. Dadurch ist die Lesbarkeit von immenser Bedeutung um Code auch in Zukunft anpassen zu können und so auf Sicherheitslücken und neue Angriffsvektoren reagieren zu können. Es ist daher nötig, folgende Regeln einzuhalten:[84][85]

Regelungen

- Eine Funktion darf nie mehr als *vier*[15] Parameter erwarten.
- Eine Funktion darf immer nur die Lösung einer einzigen Aufgabe zum Ziel haben. Um weitere Aufgaben zu lösen, müssen weitere Funktionen definiert werden.
- Um Funktionen einfach zu gestalten darf eine Funktion nicht umfangreicher als 75 Codezeilen sein [16].
- Variablennamen dürfen nur Kleinbuchstaben und Ziffern enthalten
- Die Namen von konstanten Werten dürfen nur Großbuchstaben und Ziffern enthalten.
- Klassennamen dürfen nur Buchstaben und Namen enthalten und müssen mit einem großen Buchstaben beginnen.
- Interfacenamen müssen genauso vergeben werden wie die Namen von Klassen, müssen jedoch mit der Endung *'_int'*[17] versehen werden.
- Funktionen müssen mit sprechenden Namen versehen werden, sodass auf die Funktion geschlossen werden kann. Die Namen dürfen nur Kleinbuchstaben und Ziffern beinhalten.
- Das Trennzeichen '_'[18] kann zur Verbesserung der Lesbarkeit in allen Namen verwendet werden. Alternativ kann auch die CamelCase-Notation verwendet werden.[86]
- Der Code muss durch Kommentare so erklärt werden, dass Funktionen für eine unabhängige Entwicklerin oder einen unabhängigen Entwickler klar ersichtlich sind.

[15] Anzahl der Parameter kann abgeändert werden - eine Erhöhung lässt die Komplexität und damit die Lesbarkeit steigen, und eine Verminderung lässt die Komplexität sinken
[16] Anzahl der Codezeilen pro Funktion kann abgeändert werden - eine Erhöhung lässt die Komplexität und damit die Lesbarkeit steigen und eine Verminderung lässt die Komplexität sinken
[17] Die Abkürzung kann beliebig angepasst werden.
[18] Es können bei Bedarf Sonderzeichen hinzugefügt werden.

- Es muss eine Sprache für Kommentare festgelegt werden.

Kontrollfragen

Kontrollfrage	JA / NEIN
Hat jede Funktion weniger als 75 Codezeilen?	
Hat keine Funktion mehr als vier Parameter?	
Haben alle Konstanten nur Großbuchstaben und Ziffern im Namen?	
Haben alle Interfaces die Endung '_int'?	
Beginnen alle Klassennamen mit einem Großbuchstaben?	
Enthalten alle Variablen im Namen nur Kleinbuchstaben und Ziffern?	
Wurden sprechende Funktionsnamen vergeben?	

9 Ausblick

Um die Arbeit in zukünftigen Projekten weiterzuführen könnte angedacht werden, weitere Standards und Publikationen einer Bewertung zu unterziehen. So könnten diese auf die Anwendung als Grundlage für technische und organisatorische Vorgaben geprüft und in mit Hilfe des Bewertungsschemas bewertet werden. Zusätzlich könnten mehrere beispielhafte Dokumente aufgrund der Standards und Publikationen erstellt werden, um Anwendern die Umsetzung von Vorgaben zur sicheren Entwicklung zu erleichtern:

- **Risikoanalyse**:
  Es könnten beispielhafte Risikoanalysen umgesetzt werden, um eine Grundlage für die Definition des Ziel-Sicherheitsniveaus zu schaffen.
- **Prozess Secure Coding**:
  Ein Prozess zur Entwicklung von sicherer Software könnte dargestellt werden. Dieser könnte neben dem Ablauf ebenso Risiken und Kontrollen beinhalten, die berücksichtigt werden.
- **Rollendefinitionen und Zuständigkeiten**:
  Abhängig von multiplen Faktoren, wie beispielsweise Unternehmensphilosophie, Projektgröße und Ähnliches, müssen Rollen und Zuständigkeiten definiert werden, damit ein Softwareprojekt sicher ablaufen kann. Dies dient sowohl der Qualitätssicherung als auch dem Projekterfolg selbst.

Ebenso könnte das Bewertungsschema erweitert und auch andere Faktoren berücksichtigt werden. Dadurch wäre eine genauere Bewertung der Standards und Publikationen möglich.

10 Inhaltliche Validierung

Im Folgenden sollen durch die Meinung der Experten die Qualität, die Anwendbarkeit und die Aktualität der Guideline sichergestellt werden.

Software Entwicklung ist wie IT Security ein sehr komplexes Thema. Bereits ein kleiner Tipp- oder Logikfehler kann zu schwerwiegenden Sicherheitsproblemen führen. Um diesen Risiken entgegenzuwirken ist es entscheidend, den Entwicklern entsprechende Guidelines zur sicheren Softwareentwicklung zur Seite zu stellen. Genau diese Idee wird von Hrn. Zabrana in seiner Arbeit „Entwicklung sicherer Software durch Secure Coding Guidelines“ aufgegriffen. Die darin beschriebenen Richtlinien für die Erstellung sicherer Java Anwendungen zeigen durch die vielen Beispiele praxisorientiert, welche primären Fehlerklassen zu beachten sind und wie die beschriebenen Probleme vermieden werden können. Durch die Orientierung an aktuellen Publikationen und Regelwerken entspricht die Guideline dem Stand der Technik und bietet eine gute Grundlage für die Entwicklung sicherer Software.

Florian Bogner, Information Security Experte bei Bee IT Security Consulting e.U. - Bug (Bounty) Hunter

---

Um die Sicherheit von Software gewährleisten zu können, müssen potentielle Sicherheitsprobleme bereits während der Entwicklung oder sogar der Planung vermieden werden. Deshalb ist es unverzichtbar, Entwickler über mögliche Gefahren aufzuklären. Die von Herrn Zabrana im Rahmen seiner Diplomarbeit Ëntwicklung sicherer Software durch Secure Coding Guidelinesërstellte Richtlinie, bietet einen weitreichenden Überblick über wichtige sicherheitsrelevante Fehlerkategorien. Häufig begangene Fehler werden durch verständlich formulierte Regeln abgedeckt und durch praktische Beispiele veranschaulicht. Somit unterstützt seine Richtlinie als aktuelle Wissensgrundlage die Entwicklung sicherer Software.

Daniel Marth, Softwareentwickler, Sicherheitsexperte mit mehrjähriger Erfahrung und Gewinner von mehreren nationalen und internationalen Cyber Security Challenges

Die von Herrn Zabrana in seiner Diplomarbeit Ëntwicklung sicherer Software durch Secure Coding Guidelinesërstellten Richtlinien bündeln etablierte Maßnahmen zur Absicherung von Java- Anwendungen in klarer und verständlicher Weise. Die aufgeführten Regeln entsprechen dem Stand der Technik und werden durch konkrete Beispiele verdeutlicht. Die Richtlinien bieten dadurch eine wertvolle Unterstützung für die Entwicklung sicherer Software.

Dr. Arnaud Fietzke - Software-Berater, -Analyst, -Architekt und -Entwickler bei itestra und mit langjähriger Erfahrung in der Analyse und Entwicklung komplexer, geschäftskritischer IT-Systeme

11 Validierung des Aufbaus

Um einen Aufbau der Guideline nach aktuellen Vorgaben sicherzustellen, wurde dieser durch Kommunikationsexperten überprüft und verifiziert.

Die von Kevin Zabrana verfasste Diplomarbeit „Entwicklung sicherer Software durch Secure Coding Guidelines“ wurde unter Berücksichtigung der angewandten Methodik evaluiert. Die Guideline ist im Aufbau einfach gehalten und auch sprachlich verständlich geschrieben, was die Grundvoraussetzung erfüllt, um angenommen zu werden. Angeführte Negativbeispiele helfen, die Kernaussage verständlich zu machen. Die vor jedes Kapitel gesetzte Einführung ermöglicht eine gute Übersicht und liefert meines Erachtens nach bessere Ergebnisse um nachhaltig im Gedächtnis zu bleiben (Grundsätzliches Verstehen und Möglichkeit von Ankerpunkten). Auch der sprachliche Stil mit kurzen, einfachen Sätzen ist zielgruppengerecht und sollte einem Missverstehen der Punkte vorbeugen.

Mag. Stefan Reischl ist Kommunikationsexperte und beschäftigt sich mit der Vermittlung von komplexen Inhalten (Informationssicherheit) an Mitarbeiter aller Wissenslevels. Gemeinsam mit Alexander Krenn, MSc. hat er 2018 die nextbeststep GmbH ins Leben gerufen (www.nextbeststep.at),

---

Kevin Zabrana stellt sich bewusst der Problematik, welche Kriterien Security Coding Guidelines erfüllen müssen und wie diese aufgebaut sein sollten, um zur in der Praxis gut zur Anwendung zu kommen. Genau dieser entscheidende Blick über den Tellerrand ist für eine erfolgreiche SW-Entwicklung entscheidend.

Hannes Sekyra ist international als Unternehmensberater und Top Executive Coach tätig. Er begleitet Führungskräfte in Organisationen, wenn es um Change Initiativen oder strategische Neuausrichtung geht. Meist sind nicht die inhaltlichen Fragen das Problem. Vielmehr gilt es, emotionale Widerstände, Tabus und überholte Lösungsmuster zu erkennen und anders zu gestalten.

Appendix A

An
Kevin Zabrana
Kellergasse 10
A-2000 Stockerau

Einschätzung „Secure Coding Guidelines"

Software Entwicklung ist wie IT Security ein sehr komplexes Thema. Bereits ein kleiner Tipp- oder Logikfehler kann zu schwerwiegenden Sicherheitsproblemen führen. Um diesen Risiken entgegenzuwirken ist es entscheidend, den Entwicklern entsprechende Guidelines zu sicheren Softwareentwicklung zur Seite zu stellen.

Genau diese Idee wird von Hrn. Zabrana in seiner Arbeit „Entwicklung sicherer Software durch Secure Coding Guidelines“ aufgegriffen. Die darin beschriebenen Richtlinien für die Erstellung sicherer Java Anwendungen zeigen durch die vielen Beispiele praxisorientiert, welche primären Fehlerklassen zu beachten sind und wie die beschriebenen Probleme vermieden werden können. Durch die Orientierung an aktuellen Publikationen und Regelwerken entspricht die Guideline dem Stand der Technik und bietet eine gute Grundlage für die Entwicklung sicherer Software.

Florian Bogner
Schweinern, 24.07.18

Diese Stellungnahme wurde durch Florian Bogner, Information Security Experte bei Bee IT Security Consulting e.U. erstellt. Florian unterstützt hauptberuflich seine Kunden beim Aufbau und der Umsetzung von technischen Sicherheitskonzepten und Strategien. Ziel dabei ist es, gemeinsam die Lösungen mit den größten Sicherheitsgewinnen für das jeweilige Unternehmen zu finden und umzusetzen.

Wann immer möglich, hält er Workshops und Vorträge rund um die Themen IT Security, Informationssicherheit und Security Awareness. In seiner Freizeit ist er als Bug (Bounty) Hunter unterwegs und veröffentlicht Teile der dabei identifizierten Schwachstellen auf seinem Blog https://bogner.sh.

Bee IT Security Consulting e.U.
Nibelungenstraße 37, A-3123 Schweinern
ATU73286259, FN 492303a, LG St. Pölten

Florian Bogner
+43 660 123 9454
florian@bee-itsecurity.at

Catalysts

Catalysts Digital Solutions GmbH, Huemerstraße 23, A-4020 Linz

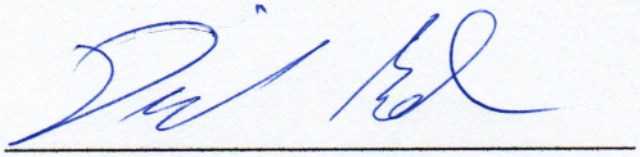
Kevin Zabrana
Kellergasse 10
A-2000 Stockerau

Wien, 01.08.2018

Statement "Secure Coding Guidelines"

Um die Sicherheit von Software gewährleisten zu können, müssen potentielle Sicherheitsprobleme bereits während der Entwicklung oder sogar der Planung vermieden werden. Deshalb ist es unverzichtbar, Entwickler über mögliche Gefahren aufzuklären.

Die von Herrn Zabrana im Rahmen seiner Diplomarbeit "Entwicklung sicherer Software durch Secure Coding Guidelines" erstellte Richtlinie, bietet einen weitreichenden Überblick über wichtige sicherheitsrelevante Fehlerkategorien. Häufig begangene Fehler werden durch verständlich formulierte Regeln abgedeckt und durch praktische Beispiele veranschaulicht. Somit unterstützt seine Richtlinie als aktuelle Wissensgrundlage die Entwicklung sicherer Software.



Daniel Marth
Leitung Catalysts Security
https://www.catalysts.cc/pentest/

Daniel Marth ist professioneller Softwareentwickler und Sicherheitsexperte mit mehrjähriger Erfahrung. Er leitet das Penetration Testing Team von Catalysts seit seiner Gründung im Jahr 2013 und berät Kunden bei der Entwicklung von sicheren Softwareanwendungen. Seine Expertise konnte er bereits bei vielen technischen Bewerben unter Beweis stellen (Sieger Austrian Cyber Security Challenge von 2013 bis 2017, Sieger European Cyber Security Challenge 2014 und 2015, Sieger OWASP AppSecEU University Challenge 2016 und 2017, Sieger Google Hardcode Secure Coding Competition 2013).

itestra
be excellent

itestra GmbH • Destouchesstr. 68 • 80796 München

Kevin Zabrana
Kellergasse 10
A-2000 Stockerau

München, den 17.08.2018

Statement „Secure Coding Guidelines“

Qualitativ hochwertige Software liefert ihren Nutzern hohen Mehrwert bei geringen Kosten, sowohl in der Entwicklung als auch im Betrieb. Dieses Ziel kann nur erreicht werden, wenn von Anfang an klar definierte und am konkreten Bedarf ausgerichtete Prinzipien durchgängig eingehalten werden.

Insbesondere im Bereich IT-Sicherheit ist "Security by Design", also die universelle Anwendung etablierter Sicherheitsprinzipien auf allen Ebenen des Systems notwendige Voraussetzung für robuste Software, die Verfügbarkeit sowie Vertraulichkeit und Integrität der von ihr verarbeiteten Daten sicherstellt.

Die von Herrn Zabrana in seiner Diplomarbeit "Entwicklung sicherer Software durch Secure Coding Guidelines" erstellten Richtlinien bündeln etablierte Maßnahmen zur Absicherung von Java-Anwendungen in klarer und verständlicher Weise. Die aufgeführten Regeln entsprechen dem Stand der Technik und werden durch konkrete Beispiele verdeutlicht. Die Richtlinien bieten dadurch eine wertvolle Unterstützung für die Entwicklung sicherer Software.

Dr. Arnaud Fietzke
itestra GmbH

Dr. Arnaud Fietzke arbeitet als Software-Berater, -Analyst, -Architekt und -Entwickler bei itestra und besitzt langjährige Erfahrung in der Analyse und Entwicklung komplexer, geschäftskritischer IT-Systeme für namhafte Unternehmen in verschiedensten Branchen, u.a. Banken, Versicherungen und Automobilindustrie. Kernkompetenz der itestra GmbH ist Softwarequalität, welche IT-Sicherheit als essentielle Komponente mit einschließt.

nextbeststep GmbH
Werkgasse 15
2232 Deutsch Wagram

Kevin Zabrana
Kellergasse 10
2000 Stockerau

Deutsch-Wagram, 22.09.2018

Evaluierung Diplomarbeit „Guidelines für Secure Coding“

Die von Kevin Zabrana verfasste Diplomarbeit „Entwicklung sicherer Software durch Secure Coding Guidelines“ wurde unter Berücksichtigung der angewandten Methodik evaluiert.

Die Guideline ist im Aufbau einfach gehalten und auch sprachlich verständlich geschrieben, was die Grundvoraussetzung erfüllt, um angenommen zu werden. Angeführte Negativbeispiele helfen, die Kernaussage verständlich zu machen. Die vor jedes Kapitel gesetzte Einführung ermöglicht eine gute Übersicht und liefert m.E. nach bessere Ergebnisse um nachhaltig im Gedächtnis zu bleiben (Grundsätzliches Verstehen und Möglichkeit von Ankerpunkten). Acuh der sprachliche Stil mit kurzen, einfachen Sätzen ist zielgruppengerecht und sollte einem Missverstehen der Punkte vorbeugen.

Stefan Reischl, nextbeststep GmbH

--

Mag. Stefan Reischl *ist Kommunikationsexperte und beschäftigt sich mit der Vermittlung von komplexen Inhalten (Informationssicherheit) an Mitarbeiter aller Wissenslevels. Gemeinsam mit Alexander Krenn, MSc. hat er 2018 die nextbeststep GmbH ins Leben gerufen (www.nextbeststep.at), die Kommunikations-Konzepte zur Vermittlung von Informationssicherheitsinhalten an Mitarbeiter im Corporate-Umfeld erstellt. Ziel ist dabei vorrangig, die Menschen dort abzuholen, wo Sie stehen und dabei einen persönlichen Nutzen erkennbar zu machen.*

THE REALLY EFFECTIVE GROUP
HIGHLIGHT THE ESSENTIAL

Kevin Zabrana

Kellergasse 10
A-2000 Stockerau

28.09.2018

Statement „Security Coding Guidelines“

Als Unternehmensberater erlebe ich im Zuge der Digitalisierung die steigende Relevanz von Datensicherheit über alle Branchen hinweg. Softwarelösungen sollen einerseits rasch dem Kunden zur Verfügung stehen - Stichwort Agilität - und gleichzeitig den hohen Qualitätsanforderungen und dabei insbesondere den aktuellen Daten-Sicherheitserwartungen genügen. Die hohe Komplexität von Software verlangt nach gut strukturierten, einfach anzuwendenden Regelwerken, damit Security gleich zu Beginn im Coding hinreichend Beachtung findet.

Herr Zabrana zeigt mit seiner Arbeit, wie dies möglich ist. Er stellt sich bewusst der Problematik, welche Kriterien Security Coding Guidelines erfüllen müssen und wie diese aufgebaut sein sollten, um zur in der Praxis gut zur Anwendung zu kommen. Genau dieser entscheidende Blick über den Tellerrand ist für eine erfolgreiche SW-Entwicklung entscheidend.

Hannes Sekyra

Hannes Sekyra ist international als Unternehmensberater und Top Executive Coach tätig. Er begleitet Führungskräfte in Organisationen, wenn es um Change Initiativen oder strategische Neuausrichtung geht. Meist sind nicht die inhaltlichen Fragen das Problem. Vielmehr gilt es, emotionale Widerstände, Tabus und überholte Lösungsmuster zu erkennen und anders zu gestalten.

Geboren in Linz/ Donau, studierte er in Graz Umweltsystemwissenschaften mit den Schwerpunkten Betriebswirtschaft und Verfahrenstechnik. Er ist ausgebildeter Gruppendynamiker und Hypnosystemiker.

Nach mehrjähriger Erfahrung als Manager im internationalen Umfeld (Europa, USA, Taiwan) und als Unternehmer in der Consumer Goods Industry ist er seit über 12 Jahren erfolgreich als selbständiger Consultant tätig. Zu seinen Kunden zählen sowohl Familienunternehmen im Mittelstand als auch internationale börsennotierte Konzerne.

Really Effective Interventions GmbH, Hauptstrasse 26, 8832 Wollerau am Zürichsee, Schweiz
Handelsregister Schwyz CH-130.4.013.590-2, Unternehmens-ID: CHE-114.360.132 MWST
Raiffeisenbank Höfe, 8832 Wollerau, Bankenclearing: 81356, Swift BIC RAIFCH22
EURO Konto IBAN CH0581356000008048834, CHF Konto IBAN CH0581356000008048812
at +43 664 1599555
h.sekyra@reallyeffective.ch
www.reallyeffective.ch

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[1] "Aphorismen," Website. [Online]. Available: https://www.aphorismen.de/zitat/60103

[2] Cert.at, "Bericht internet-sicherheit Österreich 2016," *CERT.AT Bericht 2016*, p. 60, 2016. [Online]. Available: https://www.cert.at/static/downloads/reports/cert.at-jahresbericht-2016.pdf

[3] "Owasp - secure coding practices v2," Document, June 2017. [Online]. Available: https://www.owasp.org/images/0/08/OWASP_SCP_Quick_Reference_Guide_v2.pdf

[4] I. O. for Standardization, "Informationstechnologie — sicherheitstechnik — informationssicherheits-managementsysteme — anforderungen," International Organization for Standardization, Geneva, CH, Standard, Mar. 2013.

[5] ——, "Information technology — security techniques — code of practice for information security controls," International Organization for Standardization, Geneva, CH, Standard, 2013.

[6] ——, "Information technology – security techniques – application security - part 1: Overview and concepts," International Organization for Standardization, Geneva, CH, Standard, 2011.

[7] ——, "Information technology – security techniques – application security - part 2: Organization normative framework," International Organization for Standardization, Geneva, CH, Standard, 2015.

[8] "Oenorm a 7700," Website. [Online]. Available: http://a7700.org/

[9] "Microsoft secure development lifecycle," Website, June 2017. [Online]. Available: https://www.microsoft.com/en-us/sdl/

[10] Web. [Online]. Available: https://download.microsoft.com/download/F/7/A/F7AFE1FA-2080-415F-B3C5-0187A0C6A2D0/Security%20Requirements%20Questionnaire.docx

[11] Microsoft, "The security development lifecycle developer starter kit," Website. [Online]. Available: https://www.microsoft.com/en-us/SDL/adopt/customsdltraining.aspx

[12] ——, Website, 2017. [Online]. Available: https://www.microsoft.com/en-us/SDL/adopt/tools.aspx

[13] 07 2017. [Online]. Available: https://www.nist.gov/

[14] N. N. I. of Standards and Technology, "Security considerations in the system development life cycle," *NIST 800-64*, p. 67, 2008. [Online]. Available: http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-64r2.pdf

[15] BSI, "It grundschutz," Website. [Online]. Available: https://www.bsi.bund.de/DE/Themen/ITGrundschutz/ITGrundschutzKataloge/itgrundschutzkataloge_node.html;jsessionid=EA248CB21118D9C0027B35F5E442EE2E.1_cid369

[16] "B 5.27 software-entwicklung," Website, 2017. [Online]. Available: https://www.bsi.bund.de/DE/Themen/ITGrundschutz/ITGrundschutzKataloge/Inhalt/_content/baust/b05/b05027.html?nn=6604968

[17] "M 2.569 definition von rollen und verantwortlichkeiten bei der software-entwicklung," Website, 2ß17. [Online]. Available: https://www.bsi.bund.de/DE/Themen/ITGrundschutz/ITGrundschutzKataloge/Inhalt/_content/m/m02/m02569.html;jsessionid=4F985A1B9AF7F16E962850A1C59218F2.2_cid351?nn=6604968

[18] "M 2.570 auswahl eines vorgehensmodells zur software-entwicklung," Website, 2017. [Online]. Available: https://www.bsi.bund.de/DE/Themen/ITGrundschutz/ITGrundschutzKataloge/Inhalt/_content/m/m02/m02570.html;jsessionid=4F985A1B9AF7F16E962850A1C59218F2.2_cid351?nn=6604968

[19] "M 4.493 auswahl einer entwicklungsumgebung für die software-entwicklung," Website. [Online]. Available: https://www.bsi.bund.de/DE/Themen/ITGrundschutz/ITGrundschutzKataloge/Inhalt/_content/m/m04/m04493.html;jsessionid=4F985A1B9AF7F16E962850A1C59218F2.2_cid351?nn=6604968

[20] "M 2.572 beschaffung von werkzeugen zur software-entwicklung," Website. [Online]. Available: https://www.bsi.bund.de/DE/Themen/ITGrundschutz/ITGrundschutzKataloge/Inhalt/_content/m/m02/m02572.html;jsessionid=4F985A1B9AF7F16E962850A1C59218F2.2_cid351?nn=6604968

[21] "M 2.567 auswahl vertrauenswürdiger entwicklungswerkzeuge," Website. [Online]. Available: https://www.bsi.bund.de/DE/Themen/ITGrundschutz/ITGrundschutzKataloge/Inhalt/_content/m/m02/m02567.html;jsessionid=4F985A1B9AF7F16E962850A1C59218F2.2_cid351?nn=6604968

[22] "M 4.494 sicherer einsatz einer entwicklungsumgebung," Website, 2017. [Online]. Available: https://www.bsi.bund.de/DE/Themen/ITGrundschutz/ITGrundschutzKataloge/Inhalt/_content/m/m04/m04494.html?nn=6604968

[23] "M 4.495 sicheres systemdesign bei der software-entwicklung," Website, 2017. [Online]. Available: https://www.bsi.bund.de/DE/Themen/ITGrundschutz/ITGrundschutzKataloge/Inhalt/_content/m/m04/m04495.html?nn=6604968

[24] "M 2.573 einhaltung einer sicheren vorgehensweise bei der software-entwicklung," Website. [Online]. Available: https://www.bsi.bund.de/DE/Themen/ITGrundschutz/ITGrundschutzKataloge/Inhalt/_content/m/m02/m02573.html?nn=6604968

[25] "M 2.568 testverfahren für software," Website. [Online]. Available: https://www.bsi.bund.de/DE/Themen/ITGrundschutz/ITGrundschutzKataloge/Inhalt/_content/m/m02/m02568.html?nn=6604968

[26] "M 2.574 ausführliche dokumentation der software-entwicklung," 2017. [Online]. Available: https://www.bsi.bund.de/DE/Themen/ITGrundschutz/ITGrundschutzKataloge/Inhalt/_content/m/m02/m02574.html?nn=6604968

[27] "M 4.496 sichere installation der entwickelten software," Website. [Online]. Available: https://www.bsi.bund.de/DE/Themen/ITGrundschutz/ITGrundschutzKataloge/Inhalt/_content/m/m04/m04496.html?nn=6604968

[28] "M 4.78 sorgfältige durchführung von konfigurationsänderungen," Website. [Online]. Available: https://www.bsi.bund.de/DE/Themen/ITGrundschutz/ITGrundschutzKataloge/Inhalt/_content/m/m04/m04078.html?nn=6604968

[29] "M 2.89 deinstallation von standardsoftware," Website. [Online]. Available: https://www.bsi.bund.de/DE/Themen/ITGrundschutz/ITGrundschutzKataloge/Inhalt/_content/m/m02/m02089.html?nn=6604968

[30] "Top 10 secure coding practices," August 2017. [Online]. Available: https://www.securecoding.cert.org/confluence/display/seccode/Top+10+Secure+Coding+Practices

[31] "Java rules," Website, 2017. [Online]. Available: https://www.securecoding.cert.org/confluence/display/java/Java+Rules

[32] "Cert tools," Website, August 2017. [Online]. Available: https://www.cert.org/engage/tools.cfm?

[33] "Secure coding guidelines for java se," Website. [Online]. Available: https://www.oracle.com/technetwork/java/seccodeguide-139067.html#5

[34] "How to write effective community guidelines," Website. [Online]. Available: http://thecommunitymanager.com/2013/09/26/how-to-write-effective-community-guidelines/

[35] "6 rules for writing ui guidelines," Website. [Online]. Available: https://arnhem.luminis.eu/6-rules-for-writing-ui-guidelines/

[36] "The guidelines manual," Website. [Online]. Available: https://www.nice.org.uk/process/pmg6/chapter/writing-the-clinical-guideline-and-the-role-of-the-nice-editors

[37] "Sei cert oracle coding standard for java," Website, August 2017. [Online]. Available: https://www.securecoding.cert.org/confluence/display/java/SEI+CERT+Oracle+Coding+Standard+for+Java

[38] C. Ullenboom, *Java 7 - Mehr als eine Insel*, C. Ullenboom, Ed. Galileo Press Gmbh, 2012. [Online]. Available: https://www.amazon.com/Java-Mehr-als-eine-Insel/dp/3836215071?SubscriptionId=0JYN1NVW651KCA56C102&tag=techkie-20&linkCode=xm2&camp=2025&creative=165953&creativeASIN=3836215071

[39] L. Feiler and N. Forgó, *EU-DSGVO: EU-Datenschutz-Grundverordnung*, 1st ed. Verlag Österreich, Dezember 2016.

[40] "Logging cheat sheet," Website. [Online]. Available: https://www.owasp.org/index.php/Logging_Cheat_Sheet#Protection

[41] "Apache log4j 2," Website. [Online]. Available: http://logging.apache.org/log4j/2.x/

[42] "Apache commons," Website. [Online]. Available: https://commons.apache.org/proper/commons-logging/

[43] "Simple logging facade for java (slf4j)," Website. [Online]. Available: https://www.slf4j.org/

[44] "Do not log sensitive information outside a trust boundary," Website. [Online]. Available: https://wiki.sei.cmu.edu/confluence/display/java/FIO13-J.+Do+not+log+sensitive+information+outside+a+trust+boundary

[45] "Do not log unsanitized user input," Website. [Online]. Available: https://wiki.sei.cmu.edu/confluence/display/java/IDS03-J.+Do+not+log+unsanitized+user+input

[46] "Ids00-j. prevent sql injection," Website. [Online]. Available: https://www.securecoding.cert.org/confluence/display/java/IDS00-J.+Prevent+SQL+injection

[47] "Nist special publication 800-63b," Website. [Online]. Available: https://pages.nist.gov/800-63-3/sp800-63b.html#sec5

[48] "Account lockout duration," Website. [Online]. Available: https://docs.microsoft.com/en-us/windows/security/threat-protection/security-policy-settings/account-lockout-duration

[49] "Zurücksetzen von passwörtern," Website. [Online]. Available: https://www.bsi.bund.de/DE/Themen/ITGrundschutz/ITGrundschutzKataloge/Inhalt/_content/m/m02/m02402.html

[50] "Forgot password cheat sheet," Website. [Online]. Available: https://www.owasp.org/index.php/Forgot_Password_Cheat_Sheet

[51] "Regelung des passwortgebrauchs," Website. [Online]. Available: https://www.bsi.bund.de/DE/Themen/ITGrundschutz/ITGrundschutzKataloge/Inhalt/_content/m/m02/m02011.html

[52] "Seclists," Website. [Online]. Available: https://github.com/danielmiessler/SecLists/tree/master/Passwords/Common-Credentials

[53] "Store passwords using a hash function," Website. [Online]. Available: https://wiki.sei.cmu.edu/confluence/display/java/MSC62-J.+Store+passwords+using+a+hash+function

[54] "Kryptografie," Website. [Online]. Available: https://www.bsi.bund.de/DE/Themen/Kryptografie_Kryptotechnologie/Kryptografie/kryptografie_node.html

[55] Cert.at, Website. [Online]. Available: https://www.securecoding.cert.org/confluence/display/java/MSC02-J.+Generate+strong+random+numbers?focusedCommentId=168099997#comment-168099997

[56] "Serverseitige verwendung von ssl/tls," Website. [Online]. Available: https://www.bsi.bund.de/DE/Themen/ITGrundschutz/ITGrundschutzKataloge/Inhalt/_content/m/m05/m05177.html

[57] "Msc61-j. do not use insecure or weak cryptographic algorithms," Website. [Online]. Available: https://www.securecoding.cert.org/confluence/display/java/MSC61-J.+Do+not+use+insecure+or+weak+cryptographic+algorithms

[58] "Technische richtlinie – kryptographische algorithmen und schluesellaengen," Website. [Online]. Available: https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/TechnischeRichtlinien/TR02102/BSI-TR-02102.pdf?__blob=publicationFile&v=8

[59] "Guidelines for the selection, configuration, and use of transport layer security (tls) implementations," Website. [Online]. Available: https://csrc.nist.gov/publications/detail/sp/800-52/rev-2/draft

[60] "Jdk 8 will use tls 1.2 as default," Website. [Online]. Available: https://blogs.oracle.com/java-platform-group/jdk-8-will-use-tls-12-as-default

[61] "Messagedigest class in java," Website. [Online]. Available: https://www.baeldung.com/sha-256-hashing-java

[62] "Samesite," Website. [Online]. Available: https://www.owasp.org/index.php/SameSite

[63] E. D. Schadow, *Java Web Security*, E. D. Schadow, Ed. dpunkt.verlag, 2014.

[64] "M 4.394 session-management bei webanwendungen und web-services," Website. [Online]. Available: https://www.bsi.bund.de/DE/Themen/ITGrundschutz/ITGrundschutzKataloge/Inhalt/_content/m/m04/m04394.html

[65] "Apache shiro framework," Website. [Online]. Available: https://shiro.apache.org/

[66] "Owasp enterprise security api," Website. [Online]. Available: https://www.owasp.org/index.php/Category:OWASP_Enterprise_Security_API#tab=Downloads

[67] "Spring: the source for modern java," Website. [Online]. Available: https://spring.io/

[68] "Oval," Website. [Online]. Available: http://oval.sourceforge.net/

[69] "Apache commons validator," Website. [Online]. Available: https://commons.apache.org/proper/commons-validator/

[70] "Interface preparedstatement," Website. [Online]. Available: https://docs.oracle.com/javase/7/docs/api/java/sql/PreparedStatement.html

[71] "Regex for matching date pattern in java," Website. [Online]. Available: http://www.baeldung.com/java-date-regular-expressions

[72] "Ids07-j. sanitize untrusted data passed to the runtime.exec() method," Website. [Online]. Available: https://wiki.sei.cmu.edu/confluence/display/java/IDS07-J.+Sanitize+untrusted+data+passed+to+the+Runtime.exec%28%29+method

[73] "coverity-security-library," Website. [Online]. Available: https://github.com/coverity/coveritysecurity-library

[74] "Ids51-j. properly encode or escape output," Website. [Online]. Available: https://wiki.sei.cmu.edu/confluence/display/java/IDS51-J.+Properly+encode+or+escape+output

[75] "Introducing database security for application developers," Website. [Online]. Available: https://docs.oracle.com/cd/B14117_01/network.101/b10773/apdvntro.htm

[76] "Secure coding guideline," Website. [Online]. Available: http://www.oracle.com/technetwork/java/seccodeguide-139067.html

[77] "Use of hard-coded cryptographic key," Website. [Online]. Available: https://www.owasp.org/index.php/Use_of_hard-coded_cryptographic_key

[78] "Err00-j. do not suppress or ignore checked exceptions," Website. [Online]. Available: https://wiki.sei.cmu.edu/confluence/display/java/ERR00-J.+Do+not+suppress+or+ignore+checked+exceptions

[79] "Owasp - third party projects," Website, 2017. [Online]. Available: https://www.owasp.org/index.php/Category:Java#tab=Related_3rd_Party_Projects

[80] "Oacc," Website. [Online]. Available: http://oaccframework.org/

[81] "Keycloak - open source identity and access management," Website. [Online]. Available: https://www.keycloak.org/

[82] "Picketlink," Website. [Online]. Available: http://picketlink.org/

[83] "Java simplified encryption," Website. [Online]. Available: http://www.jasypt.org/

[84] J. P. L. C. I. of Technology, "Jpl institutional coding standard for the c programming language," *Coding Guideline for Curiosity*, 2009. [Online]. Available: https://lars-lab.jpl.nasa.gov/JPL_Coding_Standard_C.pdf

[85] "Ibm clean java code," Website. [Online]. Available: https://www.ibm.com/developerworks/library/j-perry-writing-good-java-code/index.html

[86] "Java naming conventions," Website. [Online]. Available: https://www.javatpoint.com/java-naming-conventions