Gottfried Wilhelm
Leibniz Universität Hannover
Fakultät für Elektrotechnik und Informatik
Institut für Praktische Informatik
Fachgebiet Software Engineering

Aufstellen eines Qualitätsmodells aus sicherheitskritischen Anforderungen realer Softwareprojekte

Establishing a quality model from security-critical requirements of real software products

Bachelorarbeit

im Studiengang Informatik

von

Lena Huszar

Prüfer: Kurt Schneider
Zweitprüfer: Jil Klünder
Betreuer: Alexander Specht

Hannover, 04.09.2023

Erklärung der Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Bachelorarbeit selbständig und ohne fremde Hilfe verfasst und keine anderen als die in der Arbeit angegebenen Quellen und Hilfsmittel verwendet habe. Die Arbeit hat in gleicher oder ähnlicher Form noch keinem anderen Prüfungsamt vorgelegen.

Hannover, den 04.09.2023

Lena Huszar

Zusammenfassung

Sicherheit in der Informationstechnologie ist ein wichtiger Qualitätsaspekt, welcher den Erfolg oder Misserfolg eines Softwareprojekts maßgeblich beeinflusst. Während weniger kritische Schwachstellen das Image sowohl der Entwickelnden als auch der Auftraggebenden beeinflussen können, ist bei größeren Schwachstellen mit deutlich weiter reichenden Folgen zu rechnen. Je nach Schwere eines Angriffs könnten sogar Menschenleben von den Auswirkungen betroffen sein. Aufgrund dessen sollte eine Software umfangreich getestet werden, um Sicherheitslücken und das damit verbundene Risiko durch die Ausnutzung dieser zu vermeiden.

Allerdings wird Software in der heutigen Zeit immer komplexer und umfangreicher, beinhaltet viele zu schützende Informationen und bietet somit zunehmend größere Angriffsflächen. Dadurch wird es schwieriger sicherheitskritische Faktoren zu identifizieren und die Sicherheit einer Software zu gewährleisten. Grundsätzlich existieren Qualitätsmodelle wie z.B. die ISO25010, um einen Überblick über Qualitätsanforderungen zu erhalten und eine Hilfestellung für die Umsetzung dieser zu bieten. Jedoch gibt es nur wenige Modelle, die sich spezifisch mit der Sicherheit beschäftigen, alle Aspekte abdecken und in der Realität zum Einsatz kommen.

Deshalb wird in dieser Bachelorarbeit ein solches prototypisches Qualitätsmodell im Bezug auf sicherheitskritische Anforderungen eines realen Softwareprojekts aufgestellt, wofür die Herangehensweise “Bottom-up“ genutzt und untersucht wird.

Die hierbei erlangten Erkenntnisse zeigen, dass das Qualitätsmodell erfolgreich auf Software anzuwenden ist, welche Ähnlichkeiten zur Corona-Warn-App (CWA) aufweist. Auch durch den Bottom-up Ansatz enstanden einige Vorteile. Dieses Wissen kann für zukünftige Forschungen und Arbeiten genutzt werden.

Abstract

Security in information technology is an important quality aspect, which affects the success or failure of a software project significantly. Whereas less critical flaws could influence the image of developers as well as clients, bigger vulnerabilities will possibly have a considerable extensive impact. Depending on the severity of an attack, even human life could be affected by the consequences. On the basis of that, software should be comprehensively tested to avoid security flaws and to prevent the connected risk of exploitation.

However in this day and age, software gets increasingly complex and large-scaled, entails a lot of information which has to be protected and therefore opens up a wide surface for attacks. Thereby it gets more difficult to identify security-critical factors and to ensure the security of a software. Fundamentally such quality models like the ISO25010 do exist to get an overview of the quality requirements and to assist when implementing these. Yet there are few models that specifically engage with security, cover all aspects and are applied in practice.

Hence in this bachelor's thesis such a prototypical quality model wil be established, covering security-critical requirements of a real software project. In doing so the "Bottom-up" approach will be used and investigated.

Here acuired findings show, that the quality model can be successfully applied to software that bears resemblence to the CWA. There are also some advantages originated due to the bottom-up approach. That knowledge can be used for future research and work.

Akronyme

BSI	Bundesamt für Sicherheit in der Informationstechnik
CWA	Corona-Warn-App
PEPP-PT	Pan-European Privacy-Preserving Proximity Tracing
DP-3T	Decentralized Privacy-Preserving Proximity Tracing
BLE	Bluetooth Low Energy
SAST	Static application security testing
DAST	Dynamic application security testing
Pentest	Penetrationstest
DoS	Denial of Service
TCN	Temporary Contact Numbers
IaC	Infrastructure as Code
CSRF	Cross-Site Request Forgery
XSS	Cross-Site Scripting
TLS	Transport Layer Security
RKI	Robert Koch-Institut
BBB	BigBlueButton

Inhaltsverzeichnis

Kapitel 1

Einleitung

IT-Sicherheit ist ein bedeutendes Qualitätsziel. In einer Zeit, in der nahezu jeder Mensch Zugriff auf ein Smartphone, einen Computer oder ein ähnliches elektronisches Gerät haben kann, wird es immer wichtiger, dass sich Privatpersonen wie auch Unternehmen schützen. Tatsächlich nutzen weltweit ca. 3,9 Milliarden Menschen ein Smartphone (Stand 2021 [1]), in Deutschland sind es ca. 62,6 Millionen [2], also 88,8% [3] der Bevölkerung (Stand 2021) und die Zahl steigt weiterhin. Mit Computern verhält es sich recht ähnlich, sie werden von 84% der deutschen Bevölkerung im privaten Haushalt sowie auf der Arbeit genutzt (Stand 2017 [4]). Mit der wachsenden Anzahl an Nutzenden wächst allerdings auch die Anzahl der potentiellen Opfer für Cyberkriminalität. Bei einer Umfrage des Bundesamt für Sicherheit in der Informationstechnik (BSI) im Jahr 2022 gaben 29% der Befragten an, bereits Opfer von Kriminalität im Internet gewesen zu sein [10]. Im Arbeitsumfeld ist es besonders kritisch, wenn dadurch Informationen nach außen dringen. Cyberkriminelle agieren weltweit und können ihre Spuren verschleiern, sodass es schwierig ist eine Straftat zu jemandem zurückzuverfolgen, so Viegas et al. [32].

Zudem haben Angreifende eine beträchtliche Angriffsfläche, denn auch die Anzahl an Schwachstellen in Softwareprodukten nimmt jährlich zu [10, 32]. So wurde z.B. Mitte Dezember 2021 eine kritische Schwachstelle in der weit verbreiteten Java-Bibliothek Log4j[5] bekannt. Mit geringem Aufwand war es möglich diese Schwachstelle auszunutzen, da Angreifende die Kontrolle

[1] https://de.statista.com/statistik/daten/studie/309656/umfrage/prognose-zur-anzahl-der-smartphone-nutzer-weltweit/, letzter Zugriff: August 2023

[2] https://de.statista.com/statistik/daten/studie/198959/umfrage/anzahl-der-smartphonenutzer-in-deutschland-seit-2010/, letzter Zugriff: August 2023

[3] https://de.statista.com/statistik/daten/studie/585883/umfrage/anteil-der-smartphone-nutzer-in-deutschland/, letzter Zugriff: August 2023

[4] https://de.statista.com/statistik/daten/studie/3102/umfrage/quote-der-computernutzer-in-deutschland-seit-2004/, letzter Zugriff: August 2023

[5] https://www.cve.org/CVERecord?id=CVE-2021-44228, letzter Zugriff: August 2023

über Log Messages erhielten und somit eigenen Code für weitere Angriffe ausführen konnten. Laut dem BSI, ist es häufig eher schwierig festzustellen, ob ein Produkt von einer solchen Schwachstelle betroffen ist, da insbesondere kommerzielle Software meist nicht detailliert darlegt, welche Bibliotheken und Teilkomponenten genutzt werden [10]. Laut einem Artikel von Wirth [33] ist davon auszugehen, dass mehrere Millionen Server und Systeme von Log4j betroffen sind und die Auswirkungen könnten noch jahrelang zu spüren sein.

Furrer [17] beschreibt in seinem Buch, dass Software inzwischen allgegenwärtig ist und als große Hilfestellung dient. Allerdings ist unsere Gesellschaft deshalb auch auf sie angewiesen. Die Steuerung eines Flugzeugs wird z.B. durch den Autopiloten deutlich vereinfacht, sodass ein Pilot oder eine Pilotin sich anderen Aufgaben widmen oder trotz schlechten Sichtverhältnissen landen kann. Dazu muss der Autopilot aber verlässlich sein, denn fällt dieser unerwartet aus oder arbeitet fehlerhaft, könnte dies Menschenleben gefährden [18]. Auch in Zukunft ist davon auszugehen, dass Software immer mehr genutzt wird [16]. Während der Corona-Pandemie sind z.B. Plattformen für Videokonferenzen immer populärer geworden und auch ohne Isolationspflicht werden diese weiterhin im Home-Office genutzt [10, 32]. Auch hier können Sicherheitslücken auftreten, wie z.B. im Jahr 2022 bei dem Konferenz-Tool BigBlueButton (BBB). Durch zwei verschiedene Schwachstellen[6][7] ist es möglich gewesen einen Cross-Site Request Forgery (CSRF) Angriff durch Cross-Site Scripting (XSS) auf Nutzende der Plattform durchzuführen und so z.B. die Identität des Opfers zu stehlen [21]. Außerdem werden KI-Tools wie ChatGPT immer beliebter und aufgrund des Potenzials und der hohen Nachfrage stetig weiterentwickelt [25]. Mängel in einer Software können also maßgeblich den Komfort, Geschäfte und Interaktionen negativ beeinflussen, für welche die Software eigentlich eine Erleichterung darstellen soll.

Im Ernstfall kann eine Sicherheitslücke in einer Software auch zum Ausfall infrastrukturell relevanter Organisationen wie z.B. Krankenhäusern, Ämtern oder Energieversorgern führen und betrifft somit neben Eigentum auch Gesundheit und Existenz der Menschen [16, 14]. Solche Ausfälle gab es z.B. in Deutschland durch einen Hackerangriff im Sommer 2021, welcher die Landkreisverwaltung von Sachsen-Anhalt beeinträchtigte, oder auch im Zusammenhang mit dem russischen Angriffskrieg gegen die Ukraine. Das BSI äußert sich dazu in dem Bericht zur Lage der IT-Sicherheit in Deutschland 2022:

> "Die Gefährdungslage im Cyber-Raum ist so hoch wie nie."
>
> BSI [10] (S. 7)

Da Fehler ein natürliches Phänomen sind und selbst kleine Fehler bereits umfangreiche Auswirkungen haben können, ist eine Qualitätssicherung in Hinsicht auf die Sicherheit eines Produktes unabdingbar.

[6] https://www.cve.org/CVERecord?id=CVE-2022-27238
[7] https://www.cve.org/CVERecord?id=CVE-2022-26497

1.1 Problemstellung

Mit steigender Komplexität und vor allem bei besonders umfangreichen Softwareprojekten wird es immer schwieriger die Sicherheit in dem geforderten Maße zu garantieren, so Viegas et al.[32].

Eine Software ist im Idealfall gegen sämtliches unerwartetes Verhalten abgesichert. Sowohl auf der Software- und Hardwareebene als auch auf physischer Ebene und auf Ebene der Nutzenden, wie Schneier aufzeigt [29]. Außerdem muss nach Viegas et al. [32] diese Sicherung fortlaufend geschehen. Es reicht nicht aus, dass eine Software zum Zeitpunkt ihrer Veröffentlichung sicher ist, da Angreifende neue Methoden und Werkzeuge entwickeln können. Allerdings ist es mühsam sämtliche mögliche Sicherheitslücken ausfindig zu machen und anspruchsvoll jede Art von Angriff zu bedenken. Andererseits brauchen laut Schneier [29] Angreifende lediglich eine Sicherheitslücke um einen Angriff zu ermöglichen, was ihnen einen gewissen Vorteil verschafft. Nicht jede Schwachstelle bedeutet, dass ein schwerwiegender Angriff erfolgen muss, aber das Risiko im Allgemeinen sollte möglichst vermieden werden.

Gleichzeitig müssen Unternehmen auch andere Qualitätsaspekte im Auge behalten, welche aber nicht immer gut mit der Sicherheit vereinbar sind, wie z.B. Usability [24]. Um eine Balance zwischen Sicherheit und anderen wichtigen Aspekten zu behalten und trotzdem systematisch einen gewissen Grad an Sicherheit zu gewährleisten, bietet es sich an ein Qualitätsmodell zu nutzen, welches als Leitfaden bei dieser Aufgabe dient. Damit können Sicherheitsanforderungen im Laufe eines Softwareprojekts verfolgt und deren Erfüllung ermittelt werden. Spezifisch für die Sicherheit einer Software gibt es jedoch noch nicht sehr viele Qualitätsmodelle, welche alle Aspekte abdecken und in der Praxis verwendet werden.

1.2 Lösungsansatz

In dieser Bachelorarbeit wird deshalb ein prototypisches Qualitätsmodell speziell im Bezug auf sicherheitskritische Aspekte aufgestellt. Dieses soll systematisch Probleme identifizeren, Sicherheit gewährleisten und vor allem häufige Fehler, welche zu Sicherheitslücken führen, vermeiden. Dazu wird die Open Source Software “Corona-Warn-App“ herangezogen und deren bekannte Sicherheitslücken und Risiken bzw. die vorgenommenen präventiven sowie reaktiven Gegenmaßnahmen evaluiert und kategorisiert. Aus diesen Ergebnissen können sicherheitskritische Anforderungen und eventuell weitere Abwehrmechanismen abgeleitet werden. Dieses Vorgehen wird als Bottom-up bezeichnet: Aus einer spezifischen Software (Corona-Warn-App) wird ein allgemeines Modell abgeleitet. Normalerweise ist die Vorgehensweise bei der Erstellung eines Qualitätsmodells Top-down, also ohne die Betrachtung einer Software von allgemeinen Aspekten hin zu detaillierteren Unterkategorien.

Die hier genutzte umgekehrte Herangehensweise ist deshalb vielversprechend, weil das Modell anhand eines realen Softwareprojekts aufgestellt wird und somit von den Fehlern aber auch dem Wissen anderer gelernt werden kann.

Als Leitfäden zur Umsetzung des Lösungsansatzes dienen folgende Forschungsfragen:

1.2.1 Forschungsfrage 1

Um einen Überblick zu erhalten, welche Sicherheitsanforderungen an eine Software gestellt werden, ist es hilfreich sich reale Sicherheitslücken der CWA genauer anzusehen, die entweder im Verlauf der Entwicklung oder der Weiterentwicklung aufgetreten sind oder sogar zum Betrachtungszeitpunkt noch existieren. Im Zuge dessen kann untersucht werden, welche Angriffe durch die gefundenen Schwachstellen theoretisch möglich werden oder tatsächlich eingetreten sind. Zusammengefasst stellt sich so die erste Forschungsfrage:

RQ-1

Welche Sicherheitslücken und damit einhergehende mögliche Attacken sind bei der Corona-Warn-App aufgetreten?

1.2.2 Forschungsfrage 2

Im nächsten Schritt werden die vorgenommenen Sicherheitsmaßnahmen gesammelt, welche entweder direkt zu Beginn eingesetzt wurden oder als Gegenmaßnahme zu entstandenen Sicherheitslücken fungieren. Diese können später bei der Anwendung des Qualitätsmodells auf andere Software bei der Umsetzung der Sicherheitsanforderungen helfen. Entsprechend lautet die zweite Forschungsfrage:

RQ-2

Was für Sicherheitsmaßnahmen konnten vorgenommen werden und welche Sicherheitsanforderungen erfüllen sie?

1.2.3 Forschungsfrage 3

Ist das Qualitätsmodell schließlich aufgestellt sollte noch überprüft werden, dass kein wesentlicher Aspekt vergessen wurde. Dies ist wichtig, damit sich bei der Verwendung des Modells bei zukünftigen Projekten darauf verlassen werden kann, dass es seinen Zweck erfüllt und Schwachstellen effektiv vermieden werden können. Die dritte Forschungsfrage lautet also:

RQ-3

Wie kann sichergestellt werden, dass das Qualitätsmodell die wichtigsten Aspekte der CWA beinhaltet?

1.2.4 Forschungsfrage 4

Zuletzt soll dargestellt werden, welche Vorteile sich aus dieser Herangehensweise ergeben und welche Gründe es gibt, das so erstellte Qualitätsmodell anderen Modellen vorzuziehen. Somit schließt sich die vierte Forschungsfrage an, welche lautet:

RQ-4

Welche Vorteile bietet die Herangehensweise “Bottom-up“ im Vergleich zu anderen Qualitätsmodellen?

1.3 Struktur der Arbeit

Diese Arbeit teilt sich in insgesamt sieben Kapitel auf. Nach diesem einleitenden Kapitel werden zuerst einige Grundlagen erläutert, welche die Basis der weiteren Arbeit bilden. Danach wird in Kapitel 3 erklärt, wie die Corona-Warn-App evaluiert wurde und welche Ergebnisse sich daraus ergeben haben. Daraus leitet sich das verallgemeinerte Qualitätsmodell ab, welches im darauf folgenden Kapitel vorgestellt wird. Die im vorherigen Abschnitt erläuterten Forschungsfragen werden in Kapitel 5 beantwortet. In Kapitel 6 findet sich die Diskussion der Ergebnisse und die Limitierungen dieser Arbeit werden dargestellt. Abschließend werden die wesentlichen Ergebnisse noch einmal in Kapitel 7 zusammengefasst.

Kapitel 2

Grundlagen

2.1 Die Corona-Warn-App

2.1.1 Überblick

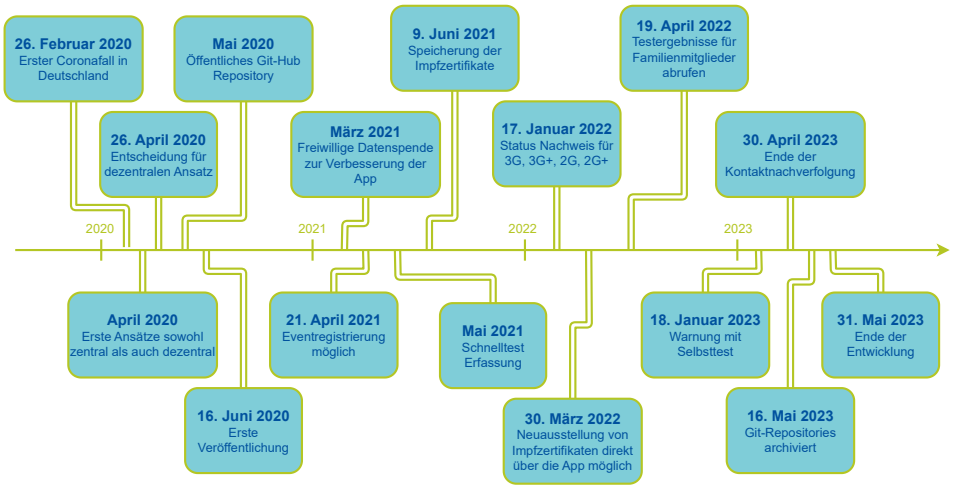


Abbildung 2.1: Historie der CWA

Nach dem ersten Coronafall in Deutschland im Februar 2020 (siehe Abb. 2.1) und die Erklärung zur Pandemie durch die WHO im März [34] wurde die Dringlichkeit von Gegenmaßnahmen deutlich. Deshalb beauftragte die deutsche Bundesregierung die Entwicklung einer App zur Kontaktnachverfolgung, für die zunächst ein zentraler Ansatz (Pan-European Privacy-Preserving Proximity Tracing (PEPP-PT)) und ein dezentraler Ansatz (Decentralized Privacy-Preserving Proximity Tracing (DP-3T)) betrachtet wurde [13]. Schließlich entschieden sich die Verantwortlichen aufgrund des Datenschutzes und der höheren Akzeptanz durch die Bevölkerung für eine

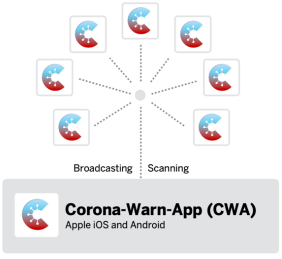
dezentralisierte Variante. Durch die Zusammenarbeit der Firma SAP und der Deutsche-Telekom-Tochter T-Systems sowie weiteren Partnern entstand innerhalb von weniger als zwei Monaten die CWA.

Wie im Science Blog der CWA [12] angegeben, stand die App ab dem 16. Juni 2020 zum Download zur Verfügung. Seit dieser ersten veröffentlichten Version der CWA wurden einige weitere optionale Funktionen hinzugefügt, wie z.B. die Registrierung bei Events (z.B. ein Restaurant-/Konzertbesuch) oder das Hinterlegen von Schnelltests, Selbsttests, Impfzertifikaten und Genesenennachweisen (siehe Abb. 2.1). Durch die Angabe von Schnelltestergebnissen, durchgeführten Impfungen usw. enthielt die App weitere personenbezogene Daten, die ebenso wie die Ergebnisse der PCR-Tests Datenschutz erforderten. Dafür boten die Funktionen weitere praktische Erleichterungen im Alltag während der Corona-Pandemie.

Angesichts der inzwischen hohen Immunität der Bevölkerung und der Rückkehr zu einem alltäglichen Leben mit nahezu keinen Corona-Maßnahmen wurde die Kontaktnachverfolgung und die Weiterentwicklung der CWA im Mai 2023 (siehe Abb. 2.1) beendet und die App in den Ruhezustand versetzt.

2.1.2 Funktionsweise

Wie auf der offiziellen Website [23] angegeben ist die CWA ein Hilfsmittel, um Infektionsketten des SARS-CoV-2 (COVID-19-Auslöser) nachverfolgen und unterbrechen zu können. Die App wurde primär in Deutschland genutzt. Durch den öffentlich zur Verfügung stehenden Source Code dient sie aber auch als Vorbild zur Programmierung von Apps in anderen Ländern[1]. Außerdem ist sie mittlerweile mit anderen europäischen Warn-Apps kompatibel und in andere Sprachen übersetzt worden.



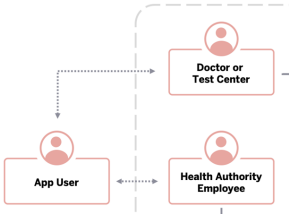
Die CWA soll Nutzende informieren, wenn sie mit einer infizierten Person mit einem bestimmten Abstand über eine festgelegte Dauer in Kontakt standen. Dies funktioniert aber nur, wenn alle Beteiligten die CWA verwenden, ihr Smartphone bei sich trugen, die infizierte Person auf COVID-19 getestet wurde und das positive Testergebnis in der App hinterlegt hat.

Um den Kontakt zu Infizierten festzustellen, sendet das Exposure Notification System der CWA in regelmäßigen Abständen eine kurzlebige zufällige Bluetooth-ID (Rolling Proximity ID) an Smartphones in ihrem Umfeld und sucht per Bluetooth Low Energy (BLE) IDs von anderen Geräten. Diese alle 10-15 Minuten aus dem zufälligen Geräteschlüssel erzeugten IDs werden dann temporär lokal

[1] https://www.helmholtz.de/newsroom/artikel/wie-sicher-ist-die-corona-warn-app/

auf den Empfängergeräten gespeichert und nach 14 Tagen wieder vom Gerät gelöscht.

Wenn nun eine infizierte Person ein positives Testergebnis nach einem Corona-Test bei einem Arzt oder einem Testcenter erhält und per QR-Code oder TeleTAN in der App registriert, sendet die CWA die Registrierung weiter an den Portal Server (siehe Abb. 2.2). Dieser leitet die Registrierung an den Verification Server weiter, wo die Registrierung verifiziert wird. Ist die Registrierung verifiziert, werden alle von der App in den letzten 14 Tagen erzeugten zufälligen Geräteschlüssel sowie der Schlüssel des Folgetages auf den CWA Server hochgeladen.



Der Server wiederum sendet die Schlüssel an alle anderen Geräte mit installierter CWA. Dort werden sie mit den lokal gespeicherten kurzlebigen zufälligen Bluetooth-IDs abgeglichen. Bei einer Übereinstimmung werden die Betroffenen über den Kontakt benachrichtigt. Außerdem wird durch eine definierte Risikoformel ein individueller Risikoscore berechnet und eine entsprechende Handlungsanweisung angezeigt, so beschrieben vom Robert Koch-Institut (RKI) [5]

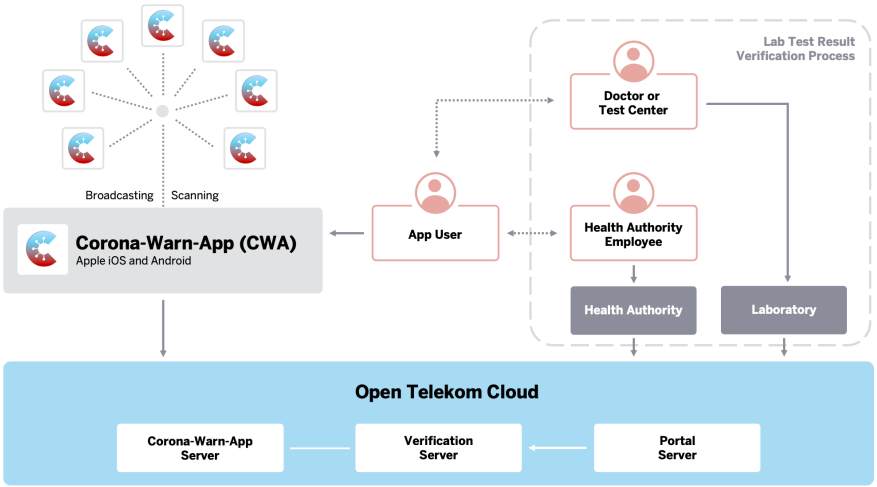


Abbildung 2.2: Komponenten der CWA [3]

2.2 STRIDE

STRIDE [4] ist ein von Microsoft entwickeltes Modell, welches Sicherheitsrisiken in sechs Kategorien unterteilt:

- Spoofing (Verschleierung der Identität)
- Tampering (Manipulation von Software/Daten)
- Repudiation (Verleugnen getätigter Aktionen)
- Information Disclosure (Offenlegung von Informationen)
- Denial of Service (Lahmlegung der Software)
- Elevation of Privilege (Ausweitung der Rechte)

Dieses Modell kann zur Identifikation von Bedrohungen für und deren mögliche Auswirkungen auf eine Software genutzt werden. Dazu wird es auf einzelne Teilkomponenten der Software angewandt. Dabei wird untersucht, ob eine oder mehrere der Kategorien zutreffen.

Ein Beispiel für **Spoofing** ist das illegale Zugreifen und der anschließende Missbrauch der Authentifizierungsdaten anderer Nutzender, also z.B. der Diebstahl von Anmeldenamen und Passwörtern. **Tampering** bezeichnet das maliziöse Verändern von Daten. Dies betrifft z.B. Daten aus einer Datenbank oder Daten, die während der Kommunikation übertragen werden.

Repudiation ist die Verleugnung getätigter Aktionen in einem System oder von Interaktionen mit einer Software. Hier kann hinterher nicht nachgewiesen werden, dass diese Aktion tatsächlich stattgefunden hat. Unter **Information Disclosure** ist die Exponierung von Daten zu verstehen, z.B. wenn Nutzende Dateien lesen können, zu denen sie eigentlich keinen Zugriff haben sollten, oder wenn Angreifende eine Übertragung abfangen und so an Informationen gelangen. **Denial of Service (DoS)** wird ein Angriff genannt, bei dem es dem System nicht mehr möglich ist wie üblich zu arbeiten, also z.B. Server nicht mehr zu erreichen sind oder bestimmte Funktionen vorübergehend nicht mehr für verifizierte Nutzende zur Verfügung stehen. Die letzte Bedrohung wird **Elevation of Privilidge** genannt. Hierbei erlangen Angreifende den Zugriff auf Funktionen oder zu Bereichen einer Software, für die sie nicht authorisiert sind. Dadurch ist es ihnen möglich das System zu gefährden oder sogar zu zerstören, da sie fälschlicherweise als vertrauenswürdig gelten.

Alle diese Risiken können einzeln oder in Kombination auftreten, häufig wird auch durch ein Risiko ein anderes begünstigt.

2.3 Qualitätsmodelle

Software-Qualitätsmodelle werden genutzt, um die Qualität einer Software sicherzustellen [28]. Sie dienen als Orientierung bei der Feststellung, welche Aspekte für die Qualität einer Software relevant sind und zu welchem Grad sie schließlich erfüllt werden müssen. Danach kann mit Hilfe von Metriken überprüft werden, ob diese Ziele erreicht wurden.

Eines der früheren Qualitätsmodelle für Software wurde von McCall entwickelt [19]. Dieses war eher allgemein und betrachtete ein Software Produkt ganzheitlich. Dazu enthielt es eine Reihe von Metriken und Checklisten, um die Qualität der Software zu messen und zu überprüfen.

D.6

METRIC	REQMTS		DESIGN		IMPLEMENTATION	
	SOURCE	TOOL	SOURCE	TOOL	SOURCE	TOOL
ET. 1 ERROR TOLERANCE CONTROL CHECKLIST:						
(1) Any concurrent processing centrally controlled.			PDS DDS	MAN	CODE	MAN
(2) Errors should be fixable and processing continued.			PDS DDS	MAN	CODE	MAN
(3) When an error condition is detected, it should be passed up to calling routine.			PDS DDS	MAN	CODE	MAN

Abbildung 2.3: Checkliste für Error Tolerance [19]

Ein weiteres bekanntes Beispiel ist die ISO9162 [2], welche 2001 veröffentlicht und später überarbeitet wurde. Im Jahr 2007 entstand daraus die ISO25010 [3]. Die Nutzung dieser ISO-Norm gilt als "best practice" und diente neben dem Modell von Firesmith als Inspiration für das in dieser Arbeit aufgestellte Qualitätsmodell. Ab dem Jahr 2001 tauchten auch spezifischere Modelle auf, mit denen es möglich war Komponenten einer Software einzeln zu bewerten. Diese wurden aus den Basis-Modellen (ISO9162, ISO25010 etc.) erstellt, indem Aspekte hinzugefügt, modifiziert und spezialisiert wurden. In der folgenden Abbildung (siehe Abb. 2.4) ist eine Übersicht der zeitlichen Entwicklung mit einigen bekannteren Qualitätsmodellen dargestellt:

[2] https://www.iso.org/standard/22749.html
[3] https://iso25000.com/index.php/en/iso-25000-standards/iso-25010?ref=123

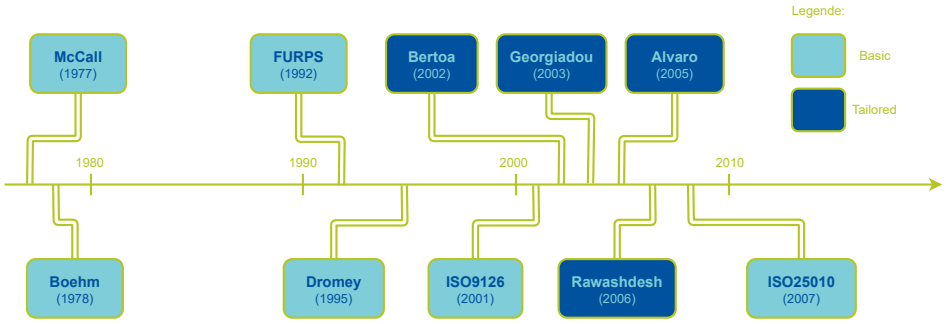


Abbildung 2.4: Evolution der Qualitätsmodelle[22]

2.4 Modell von Firesmith

Ein Beispiel für ein Qualitätsmodell, welches spezifisch auf Sicherheit bezogen ist, ist das Modell von Donald G. Firesmith [14]. Firesmith verwendet Begriffe wie sie auch in der ISO25010 unter den Qualitätsaspekten Security und Reliability zu finden sind, fügt aber auch einige Kategorien wie z.B. Privacy hinzu, oder teilt Begriffe in Unterkategorien auf. Firesmith betrachtet im Gegensatz zu dem in dieser Arbeit aufgestellten Modell auch physische und soziale Aspekte, wie Hardware und Personnel Integrity oder Physical Protection. Zudem separiert er Safety und Survivability von der Security, wobei Safety den Grad des fahrlässigen Schadens an einer Software beschreibt und Security den vorsetzlichen. Survivability ist nach Firesmith der Grad, zu dem beide Arten von Schaden vermieden, identifiziert und behoben werden. In dieser Arbeit hingegen wird nicht unterschieden, ob die Sicherheit versehentlich oder absichtlich verletzt wurde. Zum einen ist die Motivation nicht immer klar feststellbar und zum anderen können die Auswirkungen die gleichen sein, unabhängig von der Ursache. Sie müssen folglich auch gleichermaßen vermieden werden.

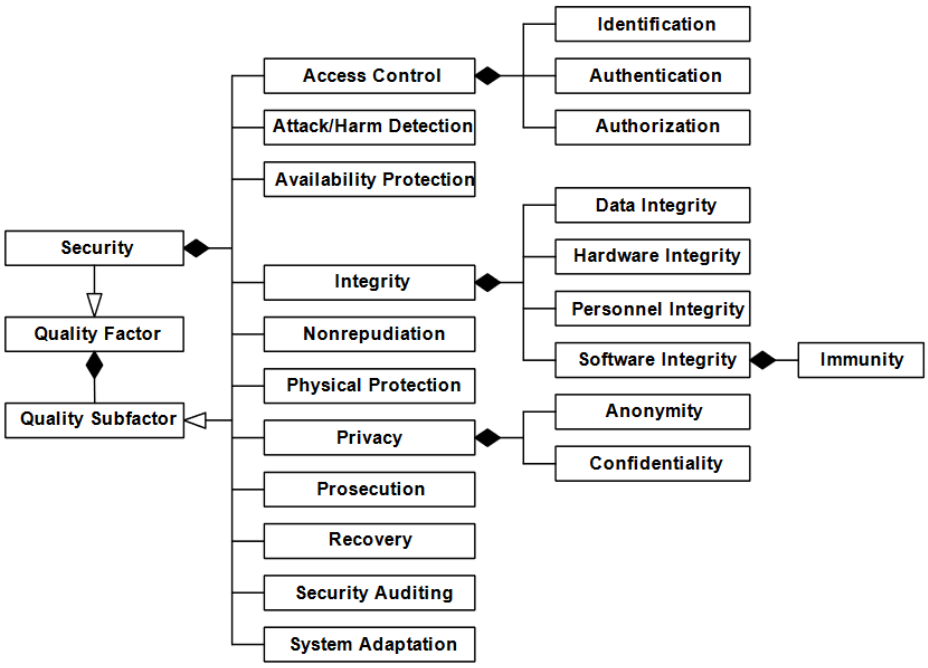


Abbildung 2.5: Dekomposition der Sicherheit in Subfaktoren der Qualität von Firesmith [14]

Kapitel 3

Evaluation der Corona-Warn-App

In diesem Kapitel wird zunächst beschrieben, wieso die CWA zur Analyse ausgewählt und wie bei der Evaluierung vorgegangen wurde. In den weiteren Abschnitten werden die Ergebnisse der Evaluierung dargestellt, welche aus den Sicherheitslücken und Sicherheitsmaßnahmen bestehen.

3.1 Auswahl der App und Vorgehen

Die CWA hat sich zur Evaluation angeboten, da sie eine aktuelle und bekannte Software und ihr Source Code sowie ihre umfangreiche Dokumentation und Entwicklung öffentlich auf GitHub einsehbar ist. Zudem hat die Sicherheit bei der Software eine zentrale Rolle eingenommen, insbesondere im Hinblick auf den Datenschutz der Nutzenden. Es wurde von Anfang an darauf Wert gelegt, diese Anforderungen umfangreich umzusetzen, obwohl die Entwicklung unter einem gewissen Zeitdruck geschehen musste. Zeitdruck ist nach Furrer [16] ein wesentlicher Faktor, was die Entstehung von sicherheitskritischen Fehlern betrifft. Demnach ist davon auszugehen, dass im Verlauf der Entwicklung der CWA Fehler entstanden sind, die sich zur Analyse eignen.

Diese Fehler lassen sich in den Repositories des Projekts auf Git-Hub finden, indem z.B. die Commits oder der Code nach bestimmten Stichwörtern durchsucht werden (s. Tabelle A). Es gab aber auch Diskussionen von Mitarbeitenden und Interessierten, in denen mögliche Sicherheitslücken diskutiert wurden. Die Stichwörter zur Suche können naheliegend sein, wie “CVE“ oder “Sicherheitslücke“, es war auch festzustellen, dass in den Beiträgen häufig das Wort “Rating“ verwendet wurde, welches zu weiteren Treffern geführt hat. Auch bei der Recherche von Papern tauchten bestimmte Wörter im Kontext von Sicherheitslücken häufig auf, welche als Inspiration für weitere Suchen nützlich waren. In einem Paper zu der SQUARE Methode befand sich

z.B. eine Liste mit sicherheitsbezogenen Wörtern, die zur Kommunikation innerhalb eines Software Engineering Teams vorgeschlagen wurde [20]. Diese eigneten sich zur Suche, da die Wörter häufig verwendet werden und ergänzten die bereits gesuchten Stichwörter, ausgenommen der Dopplungen. So ergab sich ein vielfältiges Set von Begriffen, allerdings erzielte nicht jedes Stichwort einen Treffer und einige brachten auch dieselben Ergebnisse hervor. Die so durchgeführten Suchen wurden als Schwachstelle aufgenommen, wenn entweder Mitarbeitende der CWA diese als Schwachstelle bezeichnet haben oder eindeutig eine oder mehrere Sicherheitsanforderungen nicht erfüllt wurden und somit eine nach STRIDE kategorisierbare Sicherheitslücke vorlag. Zudem wurde bei vielen Schwachstellen beschrieben, wie genau sie behoben wurden und auch zu Beginn etablierte Sicherheitsmaßnahmen lassen sich in der Dokumentation des Projekts verfolgen. Das daraus erlangte Wissen lässt sich ebenfalls zur Qualitätssicherung nutzen und wurde deshalb in die Evaluation aufgenommen. In den folgenden Abschnitten findet sich eine detaillierte Beschreibung ausgewählter Sicherheitslücken und eine Übersicht über die getroffenen Maßnahmen.

3.2 Sicherheitslücken

Ein wesentliches Ziel einer Anwendung besteht darin fehlerfrei und sicher zu laufen [16]. Trotzdem können im Entwicklungsprozess Fehler entstehen, z.B. durch zeitlichen oder finanziellen Druck auf die Mitarbeitenden. Aber auch fehlendes Verantwortungsbewusstsein, unausgeglichene Priorisierung von Qualitätsaspekten oder mangelndes Verständnis der Relevanz von Software-Sicherheit können laut Furrer [16] zu Sicherheitslücken führen. Meistens können diese vor ihrer Ausnutzung behoben werden, jedoch sollten sie möglichst von Anfang an vermieden werden, um das mögliche Risiko zu umgehen. Im Folgenden sind einige Beispiele für Sicherheitslücken in der CWA näher beschrieben, eine Tabelle mit allen im Rahmen dieser Arbeit gefundenen Schwachstellen findet sich im Anhang (s. Tabelle A).

Wie in der Einleitung (s. Abschnitt 1) bereits erwähnt, gab es im Jahr 2021 eine besonders kritische Schwachstelle in der Java-Bibliothek Log4j. Diese Bibliothek wurde auch für die CWA genutzt, wodurch die App folglich auch davon betroffen war. Durch die Schwachstelle wäre es Angreifenden möglich gewesen, eigenen Code in die Log-Messages der betroffenen CWA-Server einzuschleusen, auszuführen und somit die Grundlage für weitere Angriffe zu schaffen. Das Manipulieren solcher Meldungen war auch noch durch andere Sicherheitslücken möglich und hätte dort genutzt werden können, um bestimmte Aktionen von Mitarbeitenden zu erwirken, wie z.B. der Neustart eines Systems. Es kann aber auch verwendet werden, um sensible Daten durch eine WebRequest in die Meldungen schreiben zu lassen und sie so auszulesen.

Durch eine andere externe Bibliothek namens “CBOR“ war es aufgrund einer Schwachstelle möglich, einen DoS Angriff auszuführen und die App in eine Endlosschleife zu versetzen. Auch durch andere invalide Eingaben war eine solche Endlosschleife möglich, da im Code noch keine entsprechende Eingabeprüfung bzw. ein Timeout als Schutz etabliert war. Auch allzu umfangreiche Anfragen konnten das System überlasten, sodass es zeitweise keine neuen Anfragen bearbeiten konnte. Durch fehlende Prüfung der Ausgabe einer Funktion wiederum konnte es passieren, dass existierende Dateien ungewollt überschrieben werden.

Außerdem gab es eine Reihe von Sicherheitslücken im Bezug auf die TeleTAN. Diese hätte durch eine CSRF Attacke von Angreifenden im Namen von Nutzenden in großer Anzahl generiert werden können. Dies war möglich, da die Anzahl zunächst nicht begrenzt und schließlich durch eine Verwechslung der Zeichen “<=“ und “<“ auf eine höhere Anzahl begrenzt war als angegeben. Zusätzlich war in der Dokumentation zwar beschrieben, dass die TeleTANs lediglich eine Stunde gültig sein sollten, in der Praxis konnte aber gezeigt werden, dass sie auch nach einem Tag noch gültig waren, da die beschriebene Funktion nicht richtig implementiert war. Mit Hilfe einer Brute-Force Attacke wäre es so deutlich einfacher für Angreifende gewesen eine valide TeleTAN zu identifizieren und sie hätten zudem ausreichende Zeit dazu gehabt.

So hätte jemand sich als infiziert ausgeben können, ohne ein positives Testergebnis erhalten zu haben. Wird nun zusätzlich das vom Smartphone ausgehende Signal zur Warnung anderer verstärkt, bekommen diese eine Risikowarnung, obwohl sie keinem Risiko ausgesetzt waren und begeben sich in Quarantäne. Besonders in infrastrukturell wichtigen Einrichtungen (z.B. Krankenhäuser, Polizeistationen, politische Institutionen oder IT-Sicherheitsunternehmen), dessen Mitarbeitende mitunter verpflichtet sind sich in diesem Fall zu isolieren, kann dies verheerende Auswirkungen haben. Diese Schwachstelle könnte aber auch dazu ausgenutzt werden, um z.B. Produktionen eines konkurrierenden Unternehmens zum Erliegen zu bringen und dem eigenen Unternehmen einen Vorteil auf dem Markt zu verschaffen.

Theoretisch wäre das gleiche Szenario auch ohne TeleTAN möglich, wenn jemand einen Empfänger beispielsweise neben einem Corona-Test-Center platzieren würde, so positive Pakete abfängt und diese weitersenden würde. Auf diese Weise abgefangene Pakete haben jedoch einen Zeitstempel, welcher alle 10 Minuten wechselt. Es bräuchte also eine gewisse Anzahl an Paketen und diese müssten innerhalb des gegebenen Zeitraums weiterverbreitet werden, um die gleiche Wirkung zu erzielen.

Es gab auch einige Fälle, die es möglich gemacht haben an sensible Informationen zu kommen, ohne einen wirklichen Angriff dafür ausführen zu müssen, da einige Daten fälschlicher Weise offen zugänglich waren. Dies betraf z.B. Login Daten, die sich im Code befanden oder die Versionsnummer sowie genaue Fehlerbeschreibungen eines Servers, welche in bestimmten

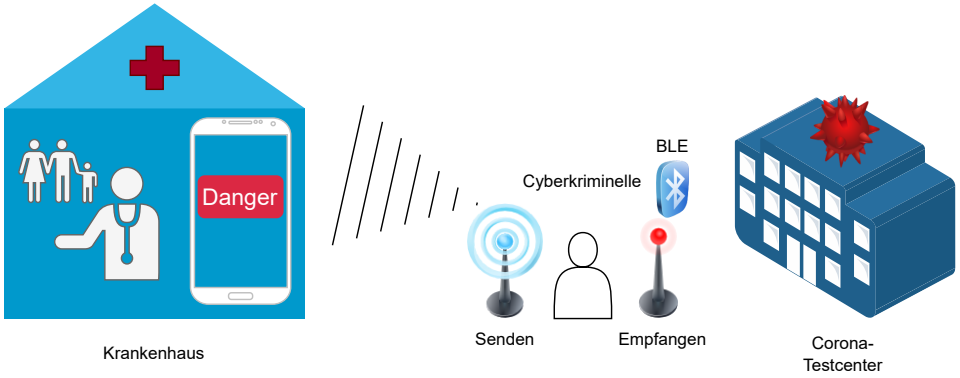


Abbildung 3.1: Möglicher Angriff ohne TeleTAN

Fehlermeldungen offenbart wurden und ausgenutzt hätten werden können, um einen Angriff zu erleichtern. Zudem hatten einige externe Anwendungen in früheren Versionen Schwierigkeiten fehlerhafte Anfragen richtig zu interpretieren und haben infolgedessen das falsche Ziel ausgewählt, um Anfragen zu bearbeiten. Manche haben hingegen die Standardeinstellung gehabt, neue Dateien für alle Nutzende eines Systems sichtbar zu machen, obwohl die Person, welche die Datei verfasst hat, dies vielleicht nicht intendierte.

Weitere Sicherheitslücken entstanden dadurch, dass die Programmierenden Root-Rechte an Nutzende vergeben haben, obwohl dies nicht nötig war, oder ihre Ergebnisse direkt in den Main-Branch gepushed haben, sodass das Testen umgangen wurde und mögliche Fehler im Code nicht gefunden werden konnten, bevor der Code genutzt wurde. Wenn die Sicherheitsmaßnahmen jedoch genutzt und nicht umgangen wurden, haben sie viele der genannten Sicherheitslücken behoben.

Da die CWA bis auf Weiteres nicht mehr aktiv genutzt wird, ist sie für zukünftige Angriffe nicht mehr ganz so attraktiv. Dennoch ist es jederzeit möglich, dass neue Schwachstellen entdeckt und ausgenutzt werden.

3.3 Sicherheitsmaßnahmen

Die Privatsphäre und die Sicherheit stehen bei der CWA im Vordergrund. Generell werden möglichst wenige personenbezogene Daten verarbeitet, um das Risiko zu minimieren und alle Nutzenden bestimmen individuell, welche Daten sie zu welchem Zweck preisgeben möchten. Zudem wurde zu Beginn der Entwicklung ein umfassendes Risk Assessment sowie Threat Modeling durchgeführt und der Code während der Entwicklung und im weiteren Verlauf regelmäßig getestet, um mit Gegenmaßnahmen bereits entstandene Lücken auszubessern. Im folgenden werden die Maßnahmen beschrieben,

welche bei der CWA zum Einsatz kamen (siehe Abb. 3.2).

3.3.1 Testen

Primär wurden drei Arten von Tests angewandt, welche zuvor im Risk Assessment gewählt wurden:

- Static application security testing (SAST)
- Dynamic application security testing (DAST)
- Penetrationstest (Pentest)

Unter SAST sind White-Box Tests zu verstehen, bei dem der Source Code regelmäßig durch ein automatisiertes Tool auf Schwachstellen untersucht wird.

Beim DAST hingegen werden Black-Box Tests verwendet. Der Source Code wird hierbei nicht betrachtet, stattdessen werden durch ein Tool simulierte Angriffe auf die Software durchgeführt, um Sicherheitslücken zu finden. Ein Pentest existiert als Black-, Grey- und White-Box Test, je nachdem wie viele Informationen den Testenden z.B. in Form von Dokumentation oder Quellcode zur Verfügung stehen. Da Pentests im Gegensatz zum SAST oder DAST manuell durchgeführt werden, können sie Aspekte beinhalten, die automatisierte Tests übersehen oder nicht bieten können. Sie sind aber auch auf die Fähigkeiten der jeweiligen Testenden angewiesen, so in einem Buch von Rohr [27] beschrieben.

Alle Varianten beschäftigen sich nicht nur mit der Software selbst, sondern insbesondere auch mit den Schnittstellen der Software, z.B. zu Servern oder Datenbanken.

Dadurch, dass der Source Code der CWA öffentlich auf GitHub einzusehen ist, kommt auch noch eine unvollständige Analyse von Teilkomponenten der CWA durch alle Interessierten, die sich den Code dort ansehen, hinzu. Finden diese gezielt oder zufällig eine vermeintliche Schwachstelle im Code, kann diese den Mitarbeitenden mitgeteilt werden. Diese prüfen dann, ob tatsächlich eine Verwundbarkeit vorliegt und was ggf. dagegen unternommen werden kann. Durch die große Anzahl an Freiwilligen in Kombination mit den offiziellen Mitarbeitenden konnten so auch schon einige Sicherheitslücken behoben oder zumindest auf Missverständlichkeiten im Code oder der Dokumentation aufmerksam gemacht werden.

Allerdings bedeutet das auch zusätzliche Arbeit für die Mitarbeitenden, da alle Hinweise auf GitHub überprüft werden müssen. Da alle Interessierten unabhängig vom individuellen Kenntnisstand die Möglichkeit haben eine Vulnerabilität zu melden, gab es auch einige Situationen, in denen über einen längeren Zeitraum über eine vermeintliche Schwachstelle diskutiert wurde, die letztendlich keine war (siehe z.B. Issue 65 [1]).

[1] https://github.com/corona-warn-app/cwa-testresult-server/issues/65

3.3.2 Dokumentation

Es bietet sich zudem an, vor dem Beginn oder zumindest zum Anfang der Implementierung zu schützende Attribute zu identifizieren und mögliche Risiken für die Anwendung zu bedenken und dementsprechend weitere individuelle Sicherheitsmaßnahmen zu planen. Dies wurde für die CWA umgesetzt (durch Risk Assessment und Threat Modeling) und während der Entwicklung konnte auf diese Erkenntnisse zurückgegriffen werden, was als Leitfaden für die Programmierung diente und Ungereimtheiten aufdeckte.

Der Source Code wiederum sollte immer mit der dazugehörigen Dokumentation übereinstimmen. Das heißt, wenn eine neue Funktion zur CWA hinzugefügt wurde, sollte die Dokumentation aktualisiert werden, um widerzuspiegeln, dass es diese Funktion gibt und was für eine Aufgabe sie erfüllt. Wenn im Laufe der Entwicklung eine geplante Funktion verworfen oder auf eine andere Weise umgesetzt wird, muss auch das in der Dokumentation angepasst werden, um keine falschen Erwartungen zu wecken. Grundsätzlich wurde auch diese Maßnahme in der CWA umgesetzt, allerdings nicht kontinuierlich, wodurch Missverständnisse entstanden (s. Abschnitt 3.2).

3.3.3 Source Code

Saubere Kommunikation sowie einheitliche Terminologie sind nicht nur in der Dokumentation, sondern auch im Source Code unerlässlich, da eine Vielzahl an Mitarbeitenden zusammen an der CWA arbeitet und wie zuvor bereits verschiedentlich erwähnt, Missverständnisse entstehen können, welche die Sicherheit der Software beeinträchtigen. So wurden in der CWA z.B. die Begriffe “Passwort“ und “Key“ an einer Stelle synonym verwendet. Da für diese Begriffe unterschiedliche Bedingungen gelten, kann dies zu falschen Annahmen führen. Wird beispielsweise für einen Key eine Zeichenlänge von 32 verlangt, für ein Passwort aber nur eine Länge von 16 und diese beiden Begriffe werden vertauscht, könnte die Länge des Keys durch unzureichende Überprüfung 16 betragen, worunter die Komplexität und somit die Sicherheit leidet. Ein ähnlicher Fall ist tatsächlich auch bei der CWA aufgetreten (s. Tabelle A), wurde aber sehr schnell wieder korrigiert.

Die Überprüfung des Inputs und Outputs ist im allgemeinen wichtig, um unerwartetes Verhalten zu erkennen und zu vermeiden. Auch das Verwenden von Variablen als Konstante sollte vermieden werden, wenn es nicht intendiert ist, um die Änderung des Codes zu vereinfachen. Solche Variablen oder Kommentare im Code sollten unter keinen Umständen unverschlüsselte sensible Informationen enthalten. Da sie auf GitHub öffentlich zugänglich sind, muss auch im Code der Datenschutz gewahrt werden und Angreifende sollten so nicht an Informationen gelangen, welche Angriffe erleichtern könnten.

3.3.4 Schnittstellen

Die Schnittstellen der CWA wie z.B. die eingebundene Java-Bibliothek Log4j wurden regelmäßig aktualisiert, denn die neuesten Versionen beinhalten wichtige Fehlerbehebungen oder zumindest provisorische Lösungen für neu aufgetauchte Sicherheitslücken. Auch wenn Bibliotheken, Programme und Server dritter Parteien nicht von dem Team der CWA selbst gewartet werden, ist das Team trotzdem in der Verantwortung, die Sicherheit der CWA, welche auch auf den Schnittstellen basiert, zu gewährleisten. Werden also Schwachstellen in genutzten Schnittstellen bekannt, werden diese umgehend auf neue sicherere Versionen aktualisiert. Falls Schnittstellen nicht die Sicherheitsanforderungen erfüllen oder keine Updates mehr bereitstellen, sollten diese nicht oder nicht mehr genutzt werden.

Des Weiteren geschieht die Kommunikation mit externen Programmen oder zu eigenen Servern, um z.B. die zufälligen Geräteschlüssel zu übermitteln, immer verschlüsselt, sodass es nicht möglich ist, Rückschlüsse auf die infizierte Person zu ziehen oder aus abgefangenen Paketen ohne Entschlüsselung sonstige Informationen zu erlangen. Zur Verschlüsselung benutzt die CWA unter anderem das Verschlüsselungsprotokoll Transport Layer Security (TLS), welches früher auch als Secure Sockets Layer (SSL) bezeichnet wurde.

Das Hochladen von positiven Testergebnissen bzw. des dafür benötigten Geräteschlüssels wird geschützt, indem zusätzlich zu den gültigen Schlüsseln Dummy-Pakete verschickt werden, welche den Schlüsseln ähnlich sehen, aber keine Funktion besitzen [5, 13]. Diese werden vom System generiert und den tatsächlichen Schlüsseln beigemischt, sodass immer eine Mindestanzahl an Schlüsseln übertragen wird. Dies war vor allem wichtig, wenn nur wenige Menschen die App nutzten. So ist es deutlich schwieriger valide Geräteschlüssel ausfindig zu machen, sollten Pakete abgefangen werden. Außerdem geschieht das Hochladen in einem Mix Netzwerk, ein Konzept eingeführt von David L. Chaum [11], wobei der Schlüssel in einer Kaskade aus Mixen mehrmals verschlüsselt, weitergeleitet und am Ende vom Empfänger wieder entschlüsselt wird. Dadurch wird verschleiert, wer mit wem kommuniziert, also in diesem Fall, welche Person infiziert ist.

3.3.5 Architektur

Wie bereits in den Grundlagen zur CWA erwähnt, gab es in Deutschland zu Anfang zwei unterschiedliche Ansätze für die Struktur, einen dezentralen und einen zentralen Ansatz (s. Abschnitt 2.1.1). Es kam jedoch zunehmend Widerstand gegen die zentrale Speicherung auf, z.B. unterzeichneten Organisationen wie die Gesellschaft für Informatik (GI) e.V. und andere einen offenen Brief an die Regierung, in dem es hieß:

> "Diese Entscheidung stößt bei uns zwischenzeitlich auf großes Unverständnis, da gerade dies der problematischste unter den vorliegenden Entwürfen ist." [15]

Der geringe Datenschutz des zentralen Ansatzes und der damit einhergehende mit geringem Aufwand mögliche Missbrauch von Gesundheitsdaten führe zu wenig Vertrauen und Akzeptanz in der Bevölkerung und sei nicht akzeptabel.

Deshalb wurde dieser Ansatz in Deutschland verworfen und die Konzentration auf das DP-3T Protokoll gelenkt, welches die Technologie BLE nutzt und neben dem Temporary Contact Numbers (TCN) [2] Protokoll und dem Privacy-Preserving Contact Tracing von Apple und Google, laut eigenen Angaben [1] die CWA inspirierte. Hierbei wird immer noch ein zentraler Server benötigt, welcher z.B. den Geräteschlüssel bei Angabe eines positiven Testergebnisses temporär speichert und an andere Geräte sendet. Jedoch werden alle personenbezogenen Daten, wie z.B. ein registriertes Impfzertifikat, nur lokal auf den jeweiligen Smartphones der Nutzenden gespeichert, was es für Angreifer deutlich erschwert an diese Daten zu kommen, da sie nicht nur einen Server angreifen müssten, sondern gezielt einzelne Geräte. Es ist also deutlich aufwendiger viele Daten zu sammeln und ein Angriff ist dadurch weniger attraktiv. Zudem können die lokalen Daten individuell verwaltet werden und die Nutzenden haben die Möglichkeit, die Daten im Falle eines Angriffs zu löschen und ihr Passwort zu ändern, wenn dieses geleakt wurde, um den Schaden möglichst gering zu halten.

Generell wurde für den Aufbau der Struktur der CWA ein Verfahren namens "Infrastructure as Code (IaC)" genutzt, wodurch der Vorgang anhand eines allgemeinen Entwurfs automatisiert wird und so die häufigsten Fehler der manuellen Erstellung vermieden werden können.

All diese Maßnahmen sind als Übersicht verallgemeinert zu einer Grafik zusammengefasst (siehe Abb. 3.2).

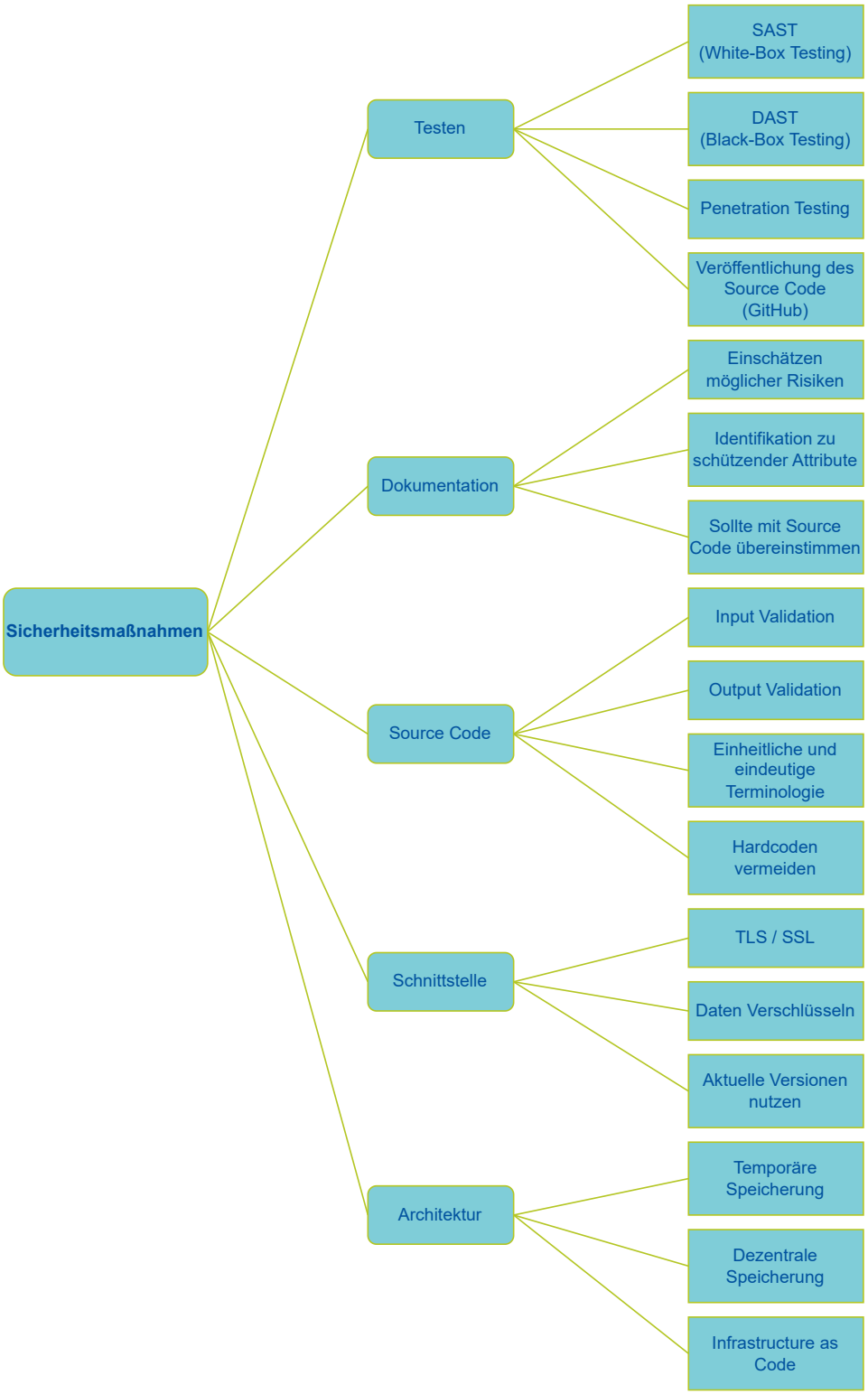


Abbildung 3.2: Sicherheitsmaßnahmen

Kapitel 4

Qualitätsmodell

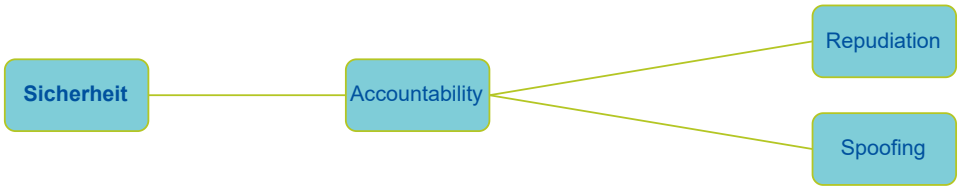
Aus der Analyse der CWA ergaben sich schließlich Kategorien, welche die Grundlage des hier erstellten Qualitätsmodells bilden. Jede Sicherheitslücke kann einer oder mehreren Arten von Risiken zugeordnet werden und die Sicherheitsmaßnahmen sollen sicherstellen, dass bestimmte Anforderungen an die App erfüllt werden, sodass die entsprechenden Risiken nicht entstehen.

4.1 Aufbau

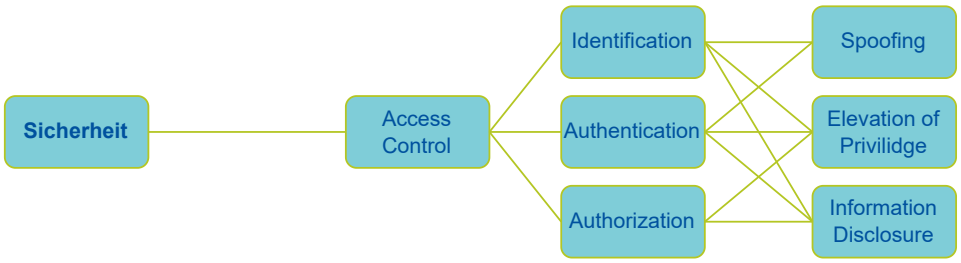
Die Sicherheit einer Software gliedert sich in diesem Modell in sieben Oberkategorien von Sicherheitsanforderungen (siehe Abb. 4.1). Im Wesentlichen sind diese Kategorien inspiriert von der ISO 25010 [6], STRIDE [4] sowie von dem Modell von Firesmith [14].

Non-Repudiation ist die Anforderung, dass für jede getätigte Aktion und jedes Ereignis in Zusammenhang mit einer Software nachgewiesen werden kann, dass sie sich in dieser Weise ereignet haben. Wird diese Anforderung nicht ausreichend erfüllt, besteht das Risiko für Repudiation. Dies ist die Verleugnung einer Aktion oder eines Ereignisses, welches negative Auswirkungen auf die Software haben kann. Bleibt deshalb eine böswillige Handlung unentdeckt, da sie nicht als externe oder schädliche Interaktion identifiziert wird, können Gegenmaßnahmen eventuell nicht rechtzeitig ergriffen werden. Dadurch entstandener Schaden hätte eingedämmt oder vermieden werden können, wäre der Angriff frühzeitig als solcher erkannt worden. Wenn zudem nicht nachgewiesen werden kann, dass die Ursache nicht die eigene Software war, kann dies z.B. rechtliche Konsequenzen mit

sich ziehen oder das Image des oder der beteiligten Unternehmen schädigen, so Parkin [26].



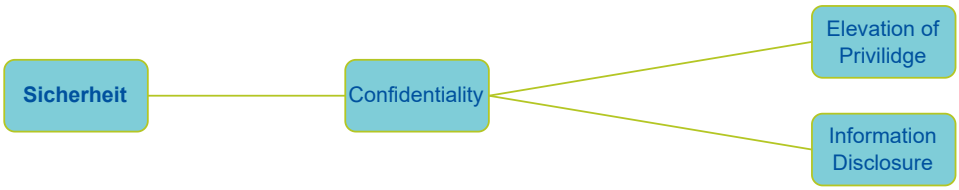
Das Verfolgen einer solchen Aktion zu der Person, die sie getätigt hat, ist in der Anforderung **Accountability** enthalten, dementsprechend ist auch hier das Risiko der Repudiation zuzuordnen. Außerdem soll damit das Risiko Spoofing verhindert werden, also dass sich eine Person als eine andere ausgibt, wodurch die eigentlichen Schuldtragenden nicht ausfindig gemacht werden können.



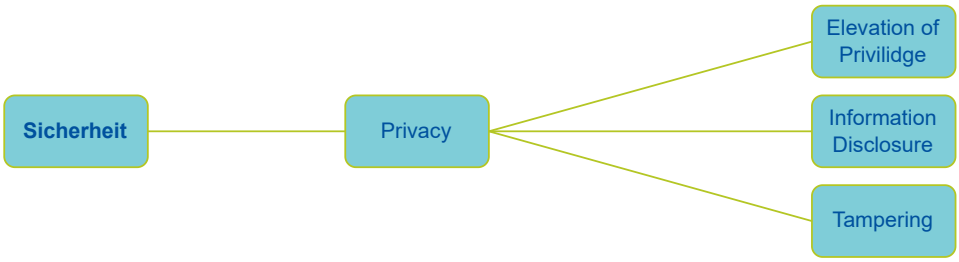
Die **Access Control**, also der Grad, zu dem ein System auf authorisierte Nutzende beschränkt ist, soll ebenfalls dabei unterstützen Spoofing zu vermeiden und kann in drei Unterkategorien gegliedert werden:

- **Identification** ist der Grad, zu dem externe Nutzende identifiziert werden können, bevor der Zugriff gewährt wird.
- **Authentication** ist der Grad, zu dem eine Identität als authorisierter Nutzer verifiziert werden kann.
- **Authorization** ist der Grad, zu dem einer authorisierten Identität die richtigen Rechte zugeordnet werden können.

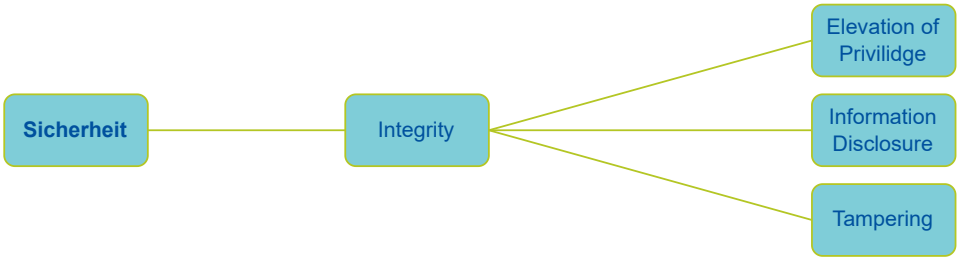
Die Unterkategorien können zudem den Risiken Elevation of Priviledge und Information Disclosure zugeordnet werden, da sie die Rechte überprüfen und Nutzende somit nur Zugang zu Seiten und Informationen erhalten, für die sie authorisiert sind.



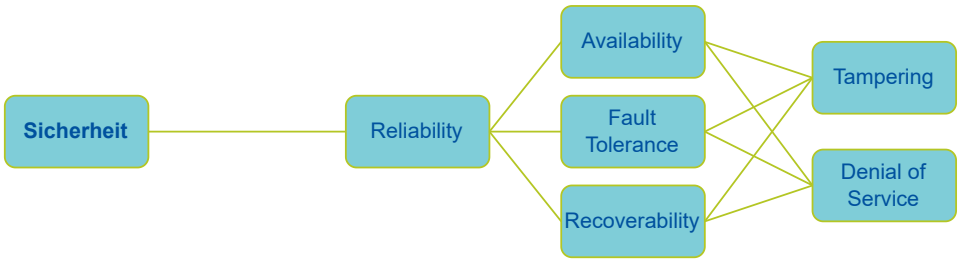
In diesem Punkt schließt sich die **Confidentiality** an, welche eng mit den vorherigen Anforderungen vernetzt ist. Hierbei liegt der Fokus vor allem bei den Daten, welche nur zu Verfügung stehen sollen, wenn Nutzende durch die Access Control dazu berechtigt sind.



Ähnlich ist es mit der **Privacy** Anforderung, hier geht es allerdings umgekehrt, um personenspezifische Daten, welche dem System nur zur Verfügung gestellt werden sollen, wenn die jeweiligen Nutzenden dies erlauben. Außerdem muss sichergestellt werden, dass die Daten akkurat sind, also nicht von anderen geändert werden können und vor anderen Nutzenden geheim gehalten werden um die Anonymität zu wahren.



Dies muss wiederum auch für andere Daten gewährleistet werden, wofür es die Anforderung **Integrity** gibt. Dies soll den Zugriff und die Modifikation von Code oder Systemdaten durch unauthorisierte Nutzende verhindern, und kann somit den Risiken Elevation of Privilidge, Information Disclosure und Tampering zugeordnet werden.



Zuletzt besteht noch die Anforderung **Reliability**, welche den Verlass auf die Software sicherstellen soll und sich ebenfalls in drei Unterkategorien aufteilt:

- **Availability** ist der Grad der Verfügbarkeit der Software, wenn sie genutzt werden soll.

- **Fault Tolerance** ist der Grad, zu dem Fehler oder Angriffe auf die Software toleriert werden und sie trotzdem noch wie gewohnt arbeitet.

- **Recoverability** ist der Grad, zu dem die Software und betroffene Daten nach dem Ausfall durch einen Fehler oder nach einem Angriff wieder in den gewünschten Zustand zurück versetzt werden können.

Alle drei Unterkategorien schützen bzw. minimieren den Schaden durch Tampering oder DoS Angriffe, welche das System stilllegen.

Den Risiken wiederum sind beispielhaft konkrete Methoden zugeordnet, wie ein Angriff in diesem Bereich geschehen könnte. Diese sollen bei der Anwendung des Modells als Hilfestellung dienen. Auch die Sicherheitsmaßnahmen aus dem vorherigen Kapitel (s. Abschnitt 3.3) können bei der Umsetzung der Sicherheitsqualität dienlich sein. Aus der Kombination all dieser Anforderungen, Risiken und deren Zuordnung untereinander ergibt sich zusammengefügt folgendes Qualitätsmodell:

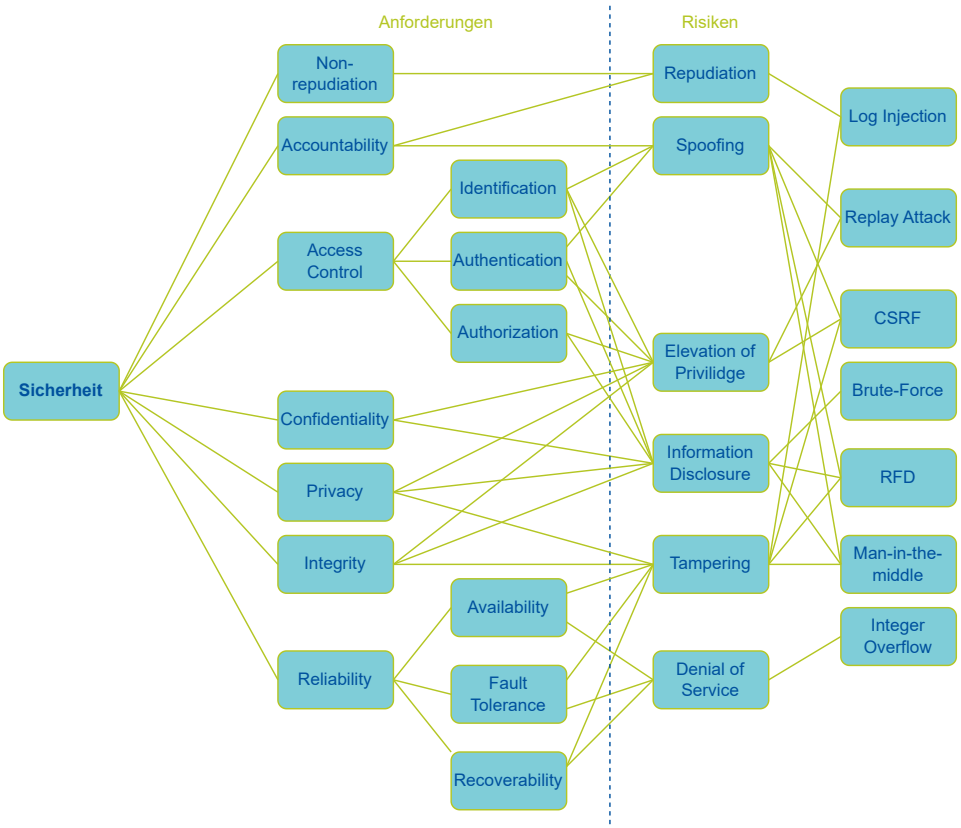


Abbildung 4.1: Qualitätsmodell

4.2 Allgemeine Anwendung

Risiken entstehen nur, wenn die entsprechenden Anforderungen nicht vollständig erfüllt sind und können somit, wie im vorherigen Abschnitt gezeigt, einer oder mehreren Anforderungen zugeordnet werden. Ganz zu Anfang ist es wichtig, dass alle Beteiligten sich mit dem Modell befasst haben und die Relevanz und Bedeutung der Sicherheitsanforderungen gleichermaßen wahrnehmen. Denn laut Braz et al. [9] gilt die Sicherheit häufig noch als nebensächlich. Das liegt unter anderem daran, dass die meisten Unternehmen keine Schulungen oder Weiterbildungen in Bezug auf die Sicherheit anbieten. Zusätzlich fehlen effiziente, einfach zu nutzende Werkzeuge in Form von unterstützenden Programmen und die Etablierung von Routinen zur Qualitätssicherung [31]. Schlussendlich wird eine Software nie 100% sicher sein, da Unternehmen konkurrenzfähig bleiben wollen und Sicherheitsmaßnahmen oft nur durchgeführt werden, wenn sie kosteneffektiv sind, so Parkin [26]. Generell gilt, dass eine Software sicherer wird, je mehr Geld dafür investiert

wird. Allerdings ist das Kosten-Nutzen-Verhältnis ab einem bestimmten Grad der Sicherheit nicht mehr ökonomisch [8]. Dadurch entstehen Abstriche bei der Qualität, besonders wenn die Entwicklungskosten möglichst gering bleiben sollen [16]. Auch die Wahrscheinlichkeit für einen Angriff und der Verlust für das Unternehmen spielt dabei eine Rolle. Ist ein Szenario eher unwahrscheinlich und mit geringen Schäden verbunden, wird selten eine Gegenmaßnahme implementiert [8]. Außerdem werden an Stelle von Sicherheitsbeauftragten und Entwickelnden automatisierte Programme auf die Qualitätssicherung angesetzt [9]. Diese sparen Kosten und Zeit, können aber nicht zwingend alle Aspekte abdecken und das menschlische Urteilungsvermögen somit nicht ersetzen [31].

Wie bei der CWA zu sehen war, gab es zwar umfassende Sicherheitsmaßnahmen, diese wurden aber nicht immer von den Entwickelnden genutzt oder umgangen, wie z.B. beim direkten Pushen von Neuerungen in den Main Branch ohne die Nutzung der vorhandenen Tests in untergeordneten Branches (s. Abschnitt 3.2). Dies könnte unbeabsichtigt passiert sein, als Ursachen sind aber auch Bequemlichkeit, Zeitdruck oder fehlende Kenntnis über die Sicherheitsmaßnahmen möglich. Es wurden auch Bots zur Identifikation von Schwachstellen genutzt, diese allein waren aber nicht in der Lage alle Schwachstellen aufzudecken. Daher ist die Erkenntnis der Relevanz und Notwendigkeit von Sicherheit der erste Schritt in der Anwendung des Modells, damit solche Fehler vermieden werden und die weiteren Schritte des Modells ihren Zweck erfüllen können.

Außerdem besteht das Problem, dass Entwickelnde verschiedenes Grundlagen- und Fachwissen besitzen, je nachdem aus welchem Bereich sie kommen und welche Erfahrungen sie im Vorhinein gemacht haben. Deshalb ist es möglich, dass Anforderungen anders interpretiert und missverstanden werden könnten und infolgedessen nicht einheitlich umgesetzt werden [20]. Gegebenenfalls ist es hilfreich sich die Definitionen der Anforderungen nochmal in eigenen Worten zu formulieren und mit Mitarbeitenden abzugleichen, um eine effektive und klare Kommunikation sicherzustellen und eine “Symmetry of Ignorance“ zu vemeiden. Denn schon simple Formulierungen wie “Lesbarkeit von Code“ können unterschiedlich interpretiert werden [28].

Um das Modell nun auf eine Software anwenden zu können, ist zunächst festzustellen, welche Anforderungen für die individuelle Software besonders relevant sind und mit welchen anderen Anforderungen sie eventuell konfligieren. Nutzt eine Software z.B. gar keine personenbezogenen Daten, ist die Anforderung “Privacy“ in diesem Fall nicht relevant. Die Priorisierung ist deshalb relevant, weil einige Anforderungen besonders bei Projekten mit Zeitdruck womöglich nicht vollständig umgesetzt werden können, oder weil sie Anforderungen aus anderen Bereichen, wie z.B. Usability zu sehr beeinträchtigen.

Danach oder zeitgleich sollten Use Cases für die verschiedenen Funktio-

nen der Software erstellt werden, falls diese nicht ohnehin schon vorhanden sind. Mithilfe der Risiken und der beispielhaften Szenarien im Modell können dann passende Misuse Cases aufgestellt werden. Damit kann einfacher identifiziert werden, welche Bedrohungen für die Software bestehen und welche Attribute geschützt werden müssen.

Schließlich muss noch betrachtet werden, wo in der Architektur der Software die Anforderungen umzusetzen sind und welche Rollen des Teams dafür zuständig sein werden, da sie die benötigte Kenntnis über den Bereich besitzen.

Mit den Ergebnissen der vorherigen Schritte ist es nun möglich Maßnahmen aufzustellen, mit denen die Anforderungen umgesetzt und Sicherheitslücken vermieden werden können. Diese können später falls nötig ergänzt werden, z.B. wenn die Software größere Veränderungen durchläuft. Aber auch reaktive Maßnahmen auf entstandene Lücken z.B. durch eingebundene Bibliotheken sollten dokumentiert werden.

Am Ende dieser Schritte sollten alle Ergebnisse daraus in einem Dokument zusammengefasst werden, um für die Entwicklung der Software einen Leitfaden für die Sicherheit zu erhalten, auf den jederzeit zurückgegriffen werden kann.

Abschließend sollten alle Ergebnisse auf ihre Richtigkeit überprüft werden. Wenn alle Parteien mit dem Endergebnis übereinkommen, wurde das Qualitätsmodell vollständig angewandt und die Erkenntnisse daraus können bei der Entwicklung der Software genutzt werden.

Insgesamt unterteilt sich die hier beschriebene Anwendung also in sieben Schritte, die mitunter zeitgleich ausgeführt werden:

1. Gemeinsames Verständnis des Qualitätsmodells und damit zusammenhängender Definitionen erarbeiten
2. Software-spezifische Priorisierung der Anforderungen
3. Misuse Cases und/oder Szenarien erarbeiten
4. Mögliche Bedrohungen und zu schützende Komponenten identifizieren
5. Anforderungen nach Bereichen der Software und Zuständigkeit kategorisieren
6. Sicherheitsmaßnahmen aufstellen
7. Zusammenfassung und Prüfung der Ergebnisse

4.3 Anwendung an einer Software

Im Folgenden wird das Modell zur Demonstrierung an der Luca-App angewendet. Diese ähnelt der CWA in einigen Funktionen, wie z.B. der Event-Registrierung und eignet sich daher gut zum Vergleich. Die App dient laut Angaben der Website[1] zur Gestaltung des gesellschaftlichen Lebens der Nutzenden. Sie dient zum Finden von Restaurants, Abrufen von Informationen wie Speisekarten und zum Bezahlen, bietet aber z.B. auch die Möglichkeit Impf- und Testzertifikate zu hinterlegen. Ein Account wird für die Nutzung nicht gefordert, ist aber für viele der Funktionen notwendig. Darin werden personenbezogene Daten wie der Name, der Standort, die Bankdaten und weitere gespeichert.

4.3.1 Gemeinsame Definition und Anforderungserhebung

Würde man das Qualitätsmodell wie oben beschrieben bei der Entwicklung der Luca-App anwenden, sollten sich zunächst alle an der Entwicklung Beteiligten genau mit dem Modell befassen. Die Anforderungen können ähnlich priorisiert werden wie bei der CWA. Auch für die Luca-App ist Privacy sehr wichtig, damit Nutzende der App ihre Daten anvertrauen können. Non-repudiation und Accountability sind im Kontext der Bezahlung von Relevanz, um diese sicher und reibungslos durchführen zu können. Generell wäre ein Ausfall der App für kurze Zeit nicht sehr kritisch, da es auch anders möglich ist Restaurants zu finden und z.B. in Bar zu bezahlen. Während einer Bezahlung hat die Availability allerdings einen höheren Stellenwert, da Probleme entstehen könnten, wenn der Vorgang unterbrochen wird. Sollte dies passieren, ist es wichtig, dass die Daten wiederhergestellt werden können, um den Vorgang korrekt abzuschließen, das erfordert eine gute Recoverability. Keine der Anforderungen ist unwichtig, da aber häufig nicht alles vollständig umgesetzt werden kann, könnte eine Priorisierung wie folgt aussehen:

1. Privacy
2. Accountability
3. Non-repudiation
4. Reliability
5. Access Control
6. Integrity
7. Confidentiality

[1] https://luca-app.de/faq/

4.3.2 Beispiel für ein Szenario

Ein Szenario eines Angriffs könnte wie folgt beschrieben werden:

Jedesmal, wenn Nutzende sich bei einer Location an- oder abmelden, besteht eine direkte Verbindung zwischen dem Server der Luca-App und dem Mobilgerät. Dabei fragt der Server Informationen ab, unter anderem auch die derzeitige IP-Adresse des Geräts.

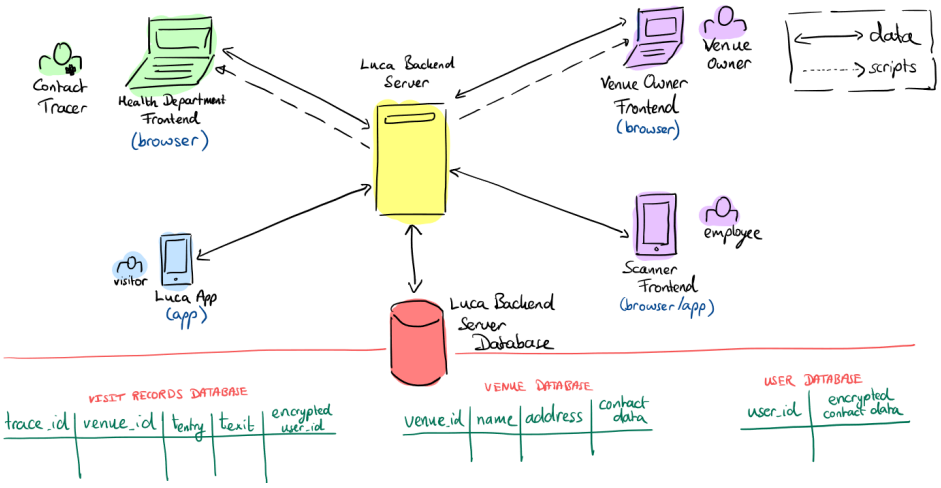


Abbildung 4.2: Überblick über die Luca-App[30]

Durch den Check-in ist bereits die derzeitige Position des Mobilgeräts bekannt und solange sich die IP-Adresse nicht verändert, kann der Standort bei weiteren Registrierungen in Restaurants auf bestimmte Individuen zurückverfolgt werden. Gelingt es nun Angreifenden Zugriff auf den Server zu erhalten, können diese nachvollziehen, wo sich eine Person zu einem bestimmten Zeitpunkt befindet und welches Pseudonym sie verwendet [30]. Der Datenschutz wäre somit verletzt und die Informationen könnten für weitere Angriffe ausgenutzt werden.

4.3.3 Identifikation, Zuweisung und Dokumentation

Die zu schützenden Komponenten wären in diesem Beispiel somit die (Standort-)Daten der Nutzenden bzw. der Server, der sie abruft. Zuständig dafür wären demnach diejenigen, die den Server einrichten und warten, sowie die Entwickler der eigentlichen App, welche die Informationen abspeichern. Eine mögliche Maßnahme wäre die Verschlüsselung der IP-Adresse, des Gerätetyps und sonstiger relevanter Daten. Außerdem wird derzeit eine zentralisierte Architektur genutzt. Würden die Daten wie bei der CWA lediglich dezentralisiert gespeichert werden, wäre ein Angriff auf den Server

nicht ganz so kritisch [30].

Würde das Modell wie in diesem Beispiel vollständig auf die Luca-App angewandt, die Erkenntnisse dokumentiert und die Anforderungen während der Entwicklung getraced werden, könnte dies maßgeblich zur Sicherheit beitragen.

4.4 Bottom-up Herangehensweise

Während bislang Qualitätsmodelle Top-down erstellt wurden, basiert das in dieser Arbeit entwickelte Qualitätsmodell auf der CWA, wurde also mit dem Bottom-up Ansatz aufgestellt. Das Prinzip Bottom-up bezeichnet eine Herangehensweise, welche z.B. bei der Erstellung von Modellen angewandt werden kann. Dabei wird von einem konkreten Ausgangspunkt hin zu einer reduzierten und allgemeineren Abbildung der Realität gearbeitet. Den Ausgangspunkt in diesem Qualitätsmodell stellten die Schwachstellen der CWA dar. Diese wurden nach Gemeinsamkeiten und Überschneidungen kategorisiert und so zusammengefasst. Für die einzelnen Kategorien wurden im Anschluss allgemein gültige Begriffe gefunden, um diese zu beschreiben. Einige dieser Kategorien, wie z.B. Fault Tolerance und Recoverability konnten wiederum zu einer übergeordneten Kategorie zusammengefasst werden. So konnten Schritt für Schritt Anforderungen aus den gefundenen Lücken abstrahiert und unter dem Begriff "Sicherheit" als allgemeines Modell konstruiert werden. Das Gegenprinzip "Top-down" hingegen bezeichnet den umgekehrten Prozess, beginnt also bei abstrakten Objekten und teilt diese Schritt für Schritt in detailliertere Unterkategorien auf. Für ein sicherheitsbezogenes Qualitätsmodell würde man dementsprechend mit dem zentralen Begriff der Sicherheit beginnen und darauf aufbauend detailliertere Unterkategorien finden, in welche sich die Sicherheit aufgliedert.

4.4.1 Vorteile

Aus dieser Herangehensweise ergeben sich einige Vorteile. Durch das Heranziehen eines realen und zeitlich aktuellen Software Projekts werden vor allem Aspekte betrachtet, die in der Praxis relevant sind. So lassen sich die einzelnen Segmente der untersten Ebene des Modells einfach feststellen und messen. Es wird deutlich, welche Fehler häufig auftreten, welche besonders kritisch sind und welche Defizite jene Modelle aufweisen, die bereits existieren und zur Qualitätssicherung im Projekt genutzt wurden. Auf diese Weise ist es einfacher solche Defizite gezielt auszubessern. Auch aus den angewandten Sicherheitsmaßnahmen können Schlüsse für zukünftige Projekte gezogen werden. Bereits funktionierende Arbeitsweisen wiederzuverwenden und die Erfahrungen und Methoden anderer zu nutzen ist einfacher und schneller, als grundlegend neue Wege zur Erreichung desselben Ziels zu erdenken. Sowohl die Fehler als auch die Erfolge eines Projekts können so konstruktiv

dazu genutzt werden ein verbessertes Qualitätsmodell aufzustellen und so zukünftige Software sicherer zu machen. Zudem besteht die Möglichkeit, dass für Entwickelnde noch deutlicher wird, wie bedeutend Sicherheit und damit präzises und sorgfältiges Arbeiten für ein Projekt ist. Da anhand der Realität aufgezeigt wird wie viele Fehler passieren können, welche Auswirkungen diese hatten oder haben könnten und dass stetig an der Verbesserung gearbeitet werden muss, werden fahrlässige Fehler vielleicht verringert. Zuletzt ist es durch die Richtung des Vorgehens vom spezifischen zum allgemeinen einfacher, alle Aspekte zu integrieren und zusammenzusetzen und somit Kohärenz in dem Modell zu schaffen. Dadurch wird das Modell sehr strukturiert und übersichtlich, wodurch es wiederum einfacher anzuwenden ist. Im Ganzen gesehen ist die Herangehensweise Bottom-up durch die einfache Messbarkeit, dem Erkenntnisgewinn aus der betrachteten Software, der problemlosen Anwendbarkeit und der Nachvollziehbarkeit durch die Nähe zur Praxis effektiv.

Kapitel 5

Beantwortung der Forschungsfragen

Die im Verlauf der Arbeit erlangten Erkenntnisse werden nun in diesem Kapitel zusammenfassend präsentiert, indem die in der Einleitung aufgestellten Forschungsfragen beantwortet werden.

5.1 Forschungsfrage 1

RQ-1

Welche Sicherheitslücken und damit einhergehende mögliche Attacken sind bei der Corona-Warn-App aufgetreten?

Wie im Kapitel 3 (s. Abschnitt 3.2) sowie im Anhang (s. Abschnitt A) detailliert beschrieben, konnten in der CWA einige Sicherheitslücken festgestellt werden. Diese sind entweder vor der Veröffentlichung oder im Prozess der Weiterentwicklung der App identifiziert worden und wurden mehrheitlich bereits wieder behoben. Zu den Ursachen der Schwachstellen gehören unter anderem die eingebundenen Bibliotheken Log4j und CBOR, aber auch die fehlerhafte Vergabe von Rechten oder die von der Dokumentation abweichende Programmierung. Die Ausnutzung dieser Lücken hätte z.B. zur Offenbarung sensibler Daten der Nutzer oder systemrelevanter Daten wie Versionsnummern oder der TeleTAN führen können. Weiterführend hätten diese Informationen dazu beitragen können, die CWA in eine Endlosschleife zu versetzen oder anderweitig funktionsunfähig zu machen, sodass die Warnfunktion nicht mehr erfüllt werden könnte. Andererseits hätten Angreifende die Kontrolle über die App erlangen und ungültige Warnungen aussenden können. Dies waren die zwei schwerwiegendsten Probleme der App, da sie einen großen Teil der Gesellschaft hätten betreffen können, unabhängig davon, ob all diese Personen der Nutzung der App zugestimmt

haben. Zusammengefasst hätten die Schäden durch die Sicherheitslücken das Leben vieler Menschen betreffen und weitreichende Folgen haben können. Insbesondere durch den möglichen Verlust der Reputation und der damit einhergehenden geringeren Nutzung der App in der Bevölkerung könnte das ursprüngliche Ziel, die Ausbreitung von COVID-19 zu verlangsamen, nicht erreicht werden.

5.2 Forschungsfrage 2

RQ-2

Was für Sicherheitsmaßnahmen konnten vorgenommen werden und welche Sicherheitsanforderungen erfüllen sie?

Die Sicherheitsmaßnahmen der CWA werden ebenfalls im dritten Kapitel dargelegt (s. Abschnitt 3.3). Sie lassen sich in folgende Oberbegriffe gliedern:

- Testen
- Dokumentation
- Source Code
- Schnittstelle
- Architektur

Die meisten Sicherheitsmaßnahmen, wie z.B. das Testen, erfüllen zeitgleich mehrere Sicherheitsanforderungen, dennoch lässt sich bei einigen Maßnahmen ein Hauptziel eingrenzen. Die dezentrale Serverarchitektur und die temporäre Speicherung soll z.B. im wesentlichen die Anforderung “Privacy“ erfüllen. Die Sicherheitsmaßnahmen greifen aber auch ineinander und erfüllen in Kombination theoretisch alle Sicherheitsanforderungen des Qualitätsmodells (siehe Abb. 4.1). Für die CWA waren insbesondere die Anforderungen Privacy, Access Control und Reliability relevant. Dies spiegelt sich in der Dokumentation, in der öffentlichen Debatte und vor allem in den Sicherheitsmaßnahmen und -lücken wieder. Wie im Modell dargestellt, sind Anforderungen meist mit mehreren Risiken verknüpft und andersherum. So können z.B. Sicherheitsmaßnahmen, die sowohl Privacy als auch Access Control erfüllen sollen, gemeinsam dazu beitragen das Risiko der Offenlegung von Informationen (Information Disclosure) entgegenzuwirken. Dafür können beispielsweise die Daten der Nutzenden verschlüsselt und zusätzlich der Zugang zu den Daten kontrolliert werden. Das Zusammenwirken der Maßnahmen erzielt hier den größten Effekt, aber es besteht auch noch ein gewisser Schutz sollte eine der Maßnahmen fehlschlagen.

5.3 Forschungsfrage 3

RQ-3

Wie kann sichergestellt werden, dass das Qualitätsmodell die wichtigsten Aspekte der CWA beinhaltet?

Das Qualitätsmodell basiert auf bereits existierenden Qualitätsmodellen, welche umstrukturiert und durch Sicherheitsanforderungen wie z.B. Reliability, die insbesondere für die CWA relevant sind, ergänzt wurden. Zudem wurde bei jeder untersuchten Sicherheitslücke mindestens eine der sieben Anforderungen des Modells verletzt. Das zeigt, dass die wesentlichen Aspekte der CWA im Kontext der gefundenen Schwachstellen enthalten sein müssen. Es gibt also keine gefundene Schwachstelle, die nicht durch die vollständige Erfüllung der aufgestellten Anforderungen vermieden werden könnte. Eine weitere Unterteilung der Aspekte wäre zudem wenig nutzbringend, da die Übersichtlichkeit des Modells darunter leidet und keine neuen Informationen daraus hervorgehen würden.

5.4 Forschungsfrage 4

RQ-4

Welche Vorteile bietet die Herangehensweise “Bottom-up“ im Vergleich zu anderen Qualitätsmodellen?

Bei dieser Herangehensweise wird eine direkte Orientierung zur Aufstellung des Qualitätsmodells genutzt. Statt ein grundlegend neues Modell zu erdenken wird untersucht, welche Aspekte sich besonders auf die Sicherheit auswirken und häufig in der Praxis vorkommen. Die Anforderungen an die Sicherheit lassen sich so einfach feststellen und messen. Das Wissen über bereits funktionierende Maßnahmen zur Qualitätssicherung kann direkt eingebunden, Fehler können analysiert und deren Auslöser vermieden werden. So kann später bei der Anwendung des neuen Modells ein möglichst großer Nutzen erzielt werden. Außerdem kann das Vertrauen in das Modell gesteigert werden, da durch die Nähe zur Praxis und die Erfahrungen und Fehlschläge anderer die Bedeutung für die Sicherheit zukünftiger Projekte erkennbar wird. Durch den Aufbau des Modells von unten nach oben, also von spezifischen Segmenten zu übergeordneten Kategorien, ist es außerdem leichter die Aspekte zu kategorisieren und zusammenzuführen. Dadurch entsteht ein übersichtliches Gesamtbild für eine effiziente Anwendung.

Kapitel 6

Diskussion

In den folgenden Abschnitten werden die zuvor dargestellten Ergebnisse kritisch betrachtet und diskutiert. Darüber hinaus werden die Grenzen dieser Arbeit aufgezeigt.

6.1 Nachteile von Bottom-up

Wie in Kapitel 4 (s. Abschnitt 4.4) gezeigt, bietet die Herangehensweise Bottom-up bei der Aufstellung eines Qualitätsmodells einige Vorteile. Allerdings bestehen auch Nachteile, die betrachtet werden müssen. Da das Qualitätsmodell anhand einer ausgewählten realen Software entwickelt wurde, ist es zwar sehr gut für Software, die der CWA ähnlich ist, geeignet, allerdings ist das Modell nicht zwangsläufig für jede Software generalisierbar. Es besteht die Möglichkeit, dass bestimmte Aspekte nicht im Qualitätsmodell enthalten sind, da sie für die CWA nicht relevant waren, für andere Software könnten aber genau diese ausgelassenen Aspekte wesentlich sein. Außerdem könnten bei einer anderen Software diverse Probleme eine Rolle spielen, die bei der CWA nie aufgetreten sind und deshalb nicht betrachtet wurden. Hinzu kommt, dass bei der Evaluierung der CWA im Wesentlichen die Softwareebene betrachtet wurde. Physische und soziale Sicherheitsrisiken, wie z.B. die Bestechung oder Bedrohung eines Mitarbeitenden, um an geschützte Informationen zu gelangen, können in diesem Modell nicht beachtet werden. Prinzipbedingt kann anhand des Codes nicht ermittelt und ausgewertet werden, inwiefern dagegen vorgegangen wird. Zusammengefasst ist ein Modell durch die Bottom-up Methode sehr detailliert und spezifisch, dadurch könnte es problematisch sein das Modell bei einer Software anzuwenden, die sich elementar von der CWA unterscheidet.

6.2 Zuordnung der Lücken in Kategorien

Für die Einordnung der gefundenen Sicherheitslücken in Kategorien und auch für die darauffolgende Erstellung des Qualitätsmodells wurde STRIDE genutzt. Die Einteilung in Kategorien war allerdings nicht in jedem Fall eindeutig, da die Risiken häufig in Kombination auftreten bzw. sich gegenseitig begünstigen.

Angenommen, es liegt ein Risiko durch Spoofing vor, also das unautho-risierte Zugreifen auf die Zugangsdaten eines Nutzenden und den Diebstahl der Identität des Opfers: Haben Angreifende z.B. Zugangsdaten mit Root-Rechten erlangt, liegt zudem ein Elevation of Priviledge Risiko vor, denn die Angreifenden haben in diesem Fall die Kontrolle über große Teile des Systems. Das kann auch bedeuten, dass sie Zugriff auf Dokumente haben, die nicht öffentlich zugänglich sind (Information Disclosure) und sie könnten auch die Möglichkeit haben solche Dokumente zu ändern oder eigenen Code einzuschleusen, was als Tampering eingeordnet wird. Führt der injizierte Code beispielsweise zu einer Endlos-Schleife, sodass das Programm keine anderen Anfragen mehr entgegennimmt, handelt es sich um einem DoS-Angriff. Außerdem könnten Angreifende Code einschleusen, der es unmöglich macht ihre illegalen Aktionen aufzuspüren, sodass ihnen die Straftat hinterher nicht nachgewiesen werden kann (Repudiation).

Dieses Beispiel zeigt, dass alle Risiken auf eine bestimmte Weise verknüpft sein können und ein Risiko auch andere begünstigen kann. Deshalb ist es umso wichtiger Software vor möglichst allen Risiken zu schützen. Um STRIDE hier trotzdem verwenden zu können, wurden jeweils eine oder mehrere Kategorien, welche die Sicherheitslücke am besten beschreiben, dieser zugeordnet. Dies dient vor allem dazu ein grundlegendes Bild von der Schwachstelle auf möglichst einfache Weise zu vermitteln, da diese mitunter komplex und sehr spezifisch für die CWA waren. Durch die Zuordnung ist es zudem einfacher aus den Schwachstellen zu lernen und somit Schlüsse für andere Software zu ziehen.

6.3 Grenzen der Arbeit

Während der Analyse und Evaluation der CWA ist ein wesentliches Problem aufgetreten, welches Zeit gekostet und somit den Umfang der gefundenen Informationen eingeschränkt hat.

Bei zu vielen hintereinander durchgeführten Suchanfragen oder zu schnellem Blättern durch die Treffer, zeigte GitHub eine Fehlermeldung an, die dazu aufforderte ein paar Minuten bis zu einer Stunde zu warten, da das Limit der Anfragen erreicht wurde. Diese Meldung trat relativ häufig auf und verzögerte so die Recherche.

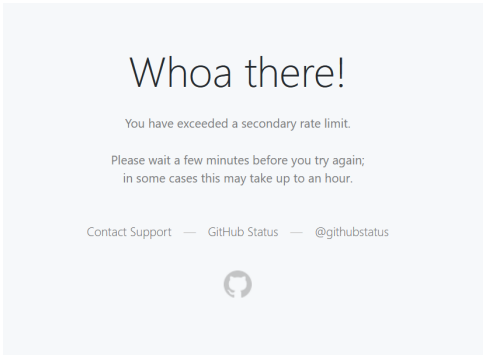


Abbildung 6.1: Fehlermeldung von GitHub

Zudem ist die CWA sehr komplex und in verschiedenen Programmiersprachen entwickelt worden, daher beschränkte sich die Suche hauptsächlich auf die in Java geschriebenen Anteile des Codes. Gravierende Sicherheitslücken oder solche, für die keine Kenntnis der Programmiersprache nötig war, da sie in natürlicher Sprache z.B. in Diskussionen beschrieben wurden, sind jedoch trotzdem mit aufgenommen worden.

Allerdings erzielten nicht alle Suchen einen Treffer und obwohl die Liste der Stichwörter sehr umfangreich war, besteht die Möglichkeit, dass Wörter nicht bedacht wurden.

Zudem ist nicht auszuschließen, dass entweder in der Suche oder in den Ergebnissen Tippfehler vorkamen und deshalb kein Ergebnis zustande kam. Aufgrund dieser Tatsache und da nicht sichergestellt werden kann, ob Teile der CWA eventuell nicht veröffentlicht wurden, wie z.B. genutzte externe Programme, ist die Vollständigkeit der aufgelisteten Sicherheitslücken nicht garantiert.

Außerdem besteht die Möglichkeit, dass Beiträge wieder gelöscht wurden und nicht mehr einsehbar sind, diese konnten ebenfalls nicht betrachtet werden.

Abschließend ist anzumerken, dass die Suche und die Evaluierung lediglich von einer Person durchgeführt wurden. Das heißt, die Kapazitäten für die Suche waren begrenzt und es gab keinerlei Kontrollinstanz, welche die Objektivität der Ergebnisse sicherstellen könnte. Trotzdem kann gewährleistet werden, dass die Ergebnisse allgemein gültig sind. Da das Material und die Quellen, welche als Basis des Modells dienten von anderen Personen verfasst wurden und die Aussagen des Autors untermauern. So wurden gefundene Sicherheitslücken z.B. mehrheitlich auch von Mitarbeitenden

der CWA als solche bezeichnet oder haben sogar eine explizite CVE-Kennzeichnung. Eine Überprüfung der Ergebnisse durch weitere Personen wäre möglich, um Subjektivität in vollem Umfang auszuschließen. Es ist allerdings nicht zu erwarten, dass dabei grundlegend abweichende Ergebnisse festgestellt werden.

Kapitel 7

Zusammenfassung und Ausblick

Dieses Kapitel fasst die Ergebnisse der Bachelorarbeit abschließend zusammen und gibt einen Ausblick auf mögliche zukünftige Forschungen, die im Zusammenhang mit der Arbeit stehen.

7.1 Zusammenfassung

Typischerweise werden Qualitätsmodelle mit der Top-down Methode erstellt. In dieser Arbeit wurde das Gegenprinzip Bottom-up verwendet, um anhand einer realen Software, der CWA, ein prototypisches Qualitätsmodell aufzustellen. Dazu wurde untersucht, welche Sicherheitslücken in der Praxis häufig auftreten, welche Sicherheitsmaßnahmen gängig genutzt werden und welche Anforderungen sie abdecken. Anschließend wurde evaluiert, ob das auf diese Weise erstellte Modell die wesentlichen sicherheitskritischen Aspekte der CWA und ähnlicher Software enthält und welche Vorteile sich durch die Nutzung des Bottom-up Ansatzes ergeben. Es konnte bei der Suche mit Hilfe von Stichwörtern eine Vielzahl an Sicherheitslücken bei der CWA festgestellt werden. Eine Liste mit allen gefundenen Schwachstellen liegt der Arbeit bei. Davon wurden viele behoben, indem zusätzliche Sicherheitsmaßnahmen implementiert oder Fehler im Code ausgebessert wurden. Ein Beispiel für eine bereits zu Beginn etablierte Maßnahme war das Testen der Software durch diverse Black-, Grey- und White-Box-Tests sowie durch Interessierte bei Git-Hub. Zudem konnte demonstriert werden, dass sich alle wesentlichen Aspekte der CWA im Qualitätsmodell widerspiegeln und das Modell somit für die App zutreffend ist. Bei der Anwendung des Modells auf die Luca-App zeigte sich, dass das Qualitätsmodell auch für andere, der CWA ähnliche Software geeignet ist. Bekannte Sicherheitslücken hätten frühzeitig identifiziert und vermieden werden können. Dies gilt auch für die CWA: Wären beispielsweise externe Bibliotheken besser überprüft und abgesichert

worden, wäre die App vermutlich nicht von Log4j betroffen gewesen. Neben den Vorteilen des Bottom-up Ansatzes wurden auch Nachteile diskutiert, diese Thematik wird im folgenden Abschnitt erneut aufgegriffen.

7.2 Ausblick

Mit den Ergebnissen dieser Arbeit entstehen auch neue Fragen und Denkanstöße, welche in zukünftigen Arbeiten und Forschungen näher betrachtet werden könnten. Für ein Qualitätsmodell ist es essentiell, dass es richtig angewandt und genutzt wird, da es seine Funktion nur dann auch erfüllen kann. Deshalb wäre es von Nutzen, verschiedene Maßnahmen zu betrachten, welche das Umgehen von Sicherheitsmaßnahmen, wie es bei der Entwicklung der CWA geschehen konnte, effektiv vermeiden. Denkbar wäre z.B., dass die von den Entwickelnden genutzten Programme eine entsprechende automatische Erinnerungsfunktion erhalten oder sogar automatisch erkennen, wenn neuer Code noch nicht getestet wurde. Zudem sollte das hier entstandene Qualitätsmodell für diverse Software geprüft werden, um festzustellen, inwieweit es im Allgemeinen anwendbar ist. Auch eine Ergänzung des Qualitätsmodells mit Hilfe des Top-down Ansatzes könnte vielversprechend sein, wenn die Vorteile von beiden Ansätzen kombiniert werden können. Andererseits könnte eine iterative Verbesserung des Qualitätsmodells erprobt werden. Beide Möglichkeiten würden womöglich fehlende Faktoren ergänzen und somit Software näher an ein Optimum im Bezug auf die Sicherheit bringen.

Anhang A

Übersicht der Suchergebnisse

Tabelle 1: Sicherheitslücken

Nr.	Kategorie	Stichwort	Link	Beschreibung
1	Elevation of Privilidge, Tampering	Log4j	https://github.com/corona-warn-app/cwa-ppa-server/pull/370 & https://github.com/corona-warn-app/cwa-server/pull/1639	Apache Log4j2 2.0-beta9 through 2.15.0 (excluding security releases 2.12.2, 2.12.3, and 2.3.1) JNDI features used in configuration, log messages, and parameters do not protect against attacker controlled LDAP and other JNDI related endpoints. An attacker who can control log messages or log message parameters can execute arbitrary code loaded from LDAP servers when message lookup substitution is enabled. *Aus <https://www.cve.org/CVERecord?id=CVE-2021-44228>*
2	Tampering	Sicherheit slücke	https://github.com/corona-warn-app/cwa-documentation/issues/661	If a user takes a RAT on the 01.06. at 12h but only registers the test result in the Corona-Warn-App on the 02.06. at 12h, the counter, which is shown under Android when a RAT returns negative, does not count the hours since the test result is available (and correctly shows that the test result is available since 24h in the example) but counts the time since the user has registered the test in the app. *Aus <https://github.com/corona-warn-app/cwa-documentation/issues/661>*
3	Information disclosure, Tampering	insecure	https://github.com/corona-warn-app/cwa-app-android/issues/402	Also insecure TLS versions are used for this CDN-Connection (TLS 1-0 - 1.3). In my opinion all public TLS attacks (BEAST, POODLE; FREAK...) only affect the confidentiality of an packet, but in case if an attacker can also tamper with the integrity of an packet (breaking the TLS MAC protection) resulting of an invalid signature, an **update force loop** can be achieved. Then a user follows the PlayStore link but can not find any new version if he has already the current version. So this (i know very theoretical) attack can make the app unusable for some time. Also the same problem occurs if the signature on the CDN server is calculated incorrect because of some technical/administration problem on the server.

				Aus <https://github.com/corona-warn-app/cwa-app-android/issues/402>
4	Information disclosure	insecure	https://github.com/corona-warn-app/cwa-quick-test-backend/issues/68	1.**Misleading Variable and Configuration Name** The encryption key is internally called Password, both in code in form of a variable as well as in the server's configuration. However, passwords and keys have different requirements in terms of complexity, length, or storage. During future development, this confusion could lead to false assumptions and potential security problems. To give an example, the fact that the key's length is checked to be either 16, 24, or 32 is an unusual, rather insecure requirement for a password, but valid for a block cipher key. 2.**CBC Mode with Hardcoded Static IV** The CBC mode in combination with a static hardcoded initialization vector leads to the issue that the same values (plaintexts) result in the same ciphertexts. Given that each column is encrypted separately, all positive test results, for example, have the same value in the database. This also applies to the other fields, such as first names, last names, email, phone number, or the date of birth. This highly impacts the confidentiality of the encryption. Various statistical attacks could be run against the database, such as a frequency distribution over the first names. In combination with the many different properties of a test result, it can be possible to infer sensitive user information. *Aus <https://github.com/corona-warn-app/cwa-quick-test-backend/issues/68>*
5	Spoofing, Tampering	insecure	https://github.com/corona-warn-app/cwa-log-	The cwa-log-upload server is vulnerable to a log injection attack at two individual endpoints. This enables an unauthenticated (as well as an authenticated) attacker to violate the integrity of the cwa-log-upload server logs by injecting

			upload/issues/24	arbitrary log messages. Note that this does not affected the user-supplied log file ZIP archives, but the actual server logging output. The injected log entries are not easily distinguishable to legitimate entries that have been appended by the cwa-logupload server itself. This issues could be abused for social engineering attacks on administrative personnel by injecting malicious messages into log files. These could include false error conditions that instruct administrators to, for example, contact someone or interact with services or systems by restarting them. In this specific case, the endpoints /api/logs and /portal/search (which requires an authenticated portal user) are affected. *Aus <https://github.com/corona-warn-app/cwa-log-upload/issues/24>*
6	Spoofing, Tampering Information Disclosure	insecure	https://github.com/corona-warn-app/cwa-verification-portal/issues/70	In this case, a CSRF attack can be used to generate TeleTANs on behalf of a logged in user. The nature of a CSRF attack does not allow the attacker to retrieve the generated TANs. However, as the generation of TeleTANs is not rate limited, an attacker can force a user, using a CSRF attack, to create a large number of TeleTANs. This results in a strongly decreased search space for the available TeleTANs. In a second step, this circumstance simplifies brute-force attacks on a valid TeleTAN. The more TeleTANs are created, the higher the probability to brute-force a correct one. *Aus <https://github.com/corona-warn-app/cwa-verification-portal/issues/70>*
7	Information Disclosure,	leak	https://github.com/corona-warn-	The Verification Portal does not offer the possibility for a user to change

	Elevation of Privilidge		app/cwa-verification-portal/issues/71	their password. This limits the user to the password they initially chose. In case a user's password leaks in any way, the affected user will not be able to change the password. *Aus <https://github.com/corona-warn-app/cwa-verification-portal/issues/71>*
8	Information Disclosure	leak	https://github.com/corona-warn-app/cwa-server/pull/74	This is intended to prevent the WebRequest to leak client data into the log, which could have happen in the old code. *Aus <https://github.com/corona-warn-app/cwa-server/pull/74>*
9	Tampering	tampering	https://github.com/corona-warn-app/cwa-documentation/issues/463	An attacker could place a receiver at a corona test centre, so there is a high chance, that there will be positive packets received. Then he retransmits them with a high signal level at crowded places. He could for example direct the signal with a high gain directional antenna to an office building (or even at the health department, police or other sensitive areas). Then a lot of employees get warnings of a possible infection and self quarantine. For that time these places can work only very limited, very bad if this is health department, police or any other critical infrastructure. *Aus <https://github.com/corona-warn-app/cwa-documentation/issues/463>*
10	Denial of Service	dos	https://github.com/corona-warn-app/cwa-server/issues/970	The download service component of the cwa-server backend can be forced into an endless loop by a malicious or faulty federation gateway server. *Aus <https://github.com/corona-warn-app/cwa-server/issues/970>*

11	Information disclosure	ble	https://github.com/corona-warn-app/cwa-server/issues/620	If the adversary is able to deploy dense network of BLE scanners then no matter how we pad / shift the data it would still be easily possible to connect Diagnosis Keys across multiple days based on movement patterns of individuals (where they live, where day work etc., days of most of us have something in common). *Aus <https://github.com/corona-warn-app/cwa-server/issues/620>*
12	Information Disclosure, Tampering	rating	https://github.com/corona-warn-app/cwa-verification-server/issues/225	The SSL/TLS configuration of the affected services does not validate the validity of certificates when connecting to other servers. This may allow attackers to read and manipulate the communication. However, the attacker needs to be in a man-in-the-middle position. Furthermore, not all services use encrypted connections. *Aus <https://github.com/corona-warn-app/cwa-verification-server/issues/225>*
13	DoS	rating	https://github.com/corona-warn-app/cwa-quick-test-backend/issues/130	The cwa-quick-test-backend relies on the third-party com.upokecenter CBOR library. The pom.xml specifies the version 4.0.1 which was released in late 2019. The current version is 4.4.3 released in May 2021. In the context of another (mobile) application known to the testers, which uses the same library, a denial-of-service vulnerability could be identified. In this context, it was possible to construct a CBOR object which caused the app to be caught in an endless loop. The issue was mitigated by updating to the most recent version of the library. *Aus <https://github.com/corona-warn-app/cwa-quick-test-backend/issues/130>*
14	!Not a real risk, just wrong documentation!	rating	https://github.com/corona-warn-app/cwa-testresult-server/issues/65	The testresult insertion point does not seem to be authenticated. According to the docu there should be JWT-Token based auth. We couldn't find an implementation of it yet.

				Aus <https://github.com/corona-warn-app/cwa-testresult-server/issues/65> • The endpoint for importing test results is open This issue will be fixed, it was a temporary opening of the firewall to ease testing • No JWT validation This is a wrong documentation/specification. The connection is using mTLS client certificates for authentication, no JWT. Documentation is fixed in PR *Aus <https://github.com/corona-warn-app/cwa-testresult-server/issues/65>*
15	DoS	rating	https://github.com/corona-warn-app/cwa-verification-server/issues/55	The server as provided by the Docker file does not limit the request size. This enables an attacker to send arbitrary long requests, that have to be cached by the server in memory. In addition, the server uses stream processing of the supplied JSON data. By repeatedly sending the payload {"": we can create lots of nested dictionaries, that will be processed by the server. This yields a *10x amplification in memory usage* on the server in comparison to the traffic sent by the attacker. After sending ~164MB, the server CPU usage goes to 100% and the server becomes unresponsive in this configuration. As a single request was used, rate-limiting is not sufficient. *Aus <https://github.com/corona-warn-app/cwa-verification-server/issues/55>*
16	Information Disclosure	rating	https://github.com/coron	Multiple code and config files include hardcoded [login] credentials.

			a-warn-app/cwa-testresult-server/issues/30	*Aus <https://github.com/corona-warn-app/cwa-testresult-server/issues/30>*
17	Information Disclosure	rating	https://github.com/corona-warn-app/cwa-verification-server/issues/152	The Nginx web server discloses its version in the HTTP Server header field and HTTP error pages. Disclosing version numbers of used software components could allow attackers to prepare further attacks. For instance, an attacker could use publicly known vulnerabilities to harm the service. They could also try to find new vulnerabilities in the code of this exact version. *Aus <https://github.com/corona-warn-app/cwa-verification-server/issues/152>*
18	Information Disclosure	rating	https://github.com/corona-warn-app/cwa-verification-server/issues/153	The TeleTAN, which is produced by the public health department, is per documentation only valid for one hour. However, during the evaluation, it was determined that this TeleTAN is still valid after one day. Due to the fact that, additionally, no protection exits against brute-force attacks on the use of the TeleTAN to generate a registration token, an attacker would have plenty of time to identify a valid TeleTAN. It is therefore that this vulnerability was rated as high risk. *Aus <https://github.com/corona-warn-app/cwa-verification-server/issues/153>*
19	Tampering	rating	https://github.com/corona-warn-app/cwa-verification-server/issues/32	The documentation mentions that currently only two TANs per session can be created. This is also formalized in the application.yml configuration file (appsession: tancountermax: 2). However, the code that is checking the TAN counter is implemented incorrectly. It checks whether the current TAN count is less-than-or-equal (<=) to

				the specified tancountermax. However, the counter starts off with the value 0 and is incremented after a TAN is generated. Thus, the check is insufficient and allows *three* instead of *two* tans to be created by one session. The operator should be "<" instead of "<=". *Aus <https://github.com/corona-warn-app/cwa-verification-server/issues/32>* -> Attacker could create their own third Tan?
20	Tampering	rating	https://github.com/corona-warn-app/cwa-server/issues/196	A missing check of a return value could lead to unexpected behavior. The method createNewFile() returns a boolean (see https://docs.oracle.com/javase/7/docs/api/java/io/File.html#createNewFile()) which is not checked. The function would return false if the file already exists. In the current implementation this is not an issue (file name is a constant) but could lead to unexpected behavior in the future. It could potentially lead to unintentionally overwriting existing files. The function name and the function doku should indicate that this function ignores existing files. *Aus <https://github.com/corona-warn-app/cwa-server/issues/196>*
21	Information disclosure	rating	https://github.com/corona-warn-app/cwa-verification-server/issues/4	The server responds with verbose error messages and details about occured exceptions. This can help an attacker to better understand the running services/applications and debug the problem the processing component is running into. The information collected by the attacker can possibly be used to exploit the component or let the component behave unintended. *Aus <https://github.com/corona-warn-app/cwa-verification-server/issues/4>*

22	Elevation of Privilidge, Information disclosure	Access control	https://github.com/corona-warn-app/cwa-documentation/issues/93	The current design does not yet describe the communication channels' security properties between the app and the servers. Based on the current design, it seems that the messages (i) GUIDs registration, (ii) checking for results, (iii) Uploading diagnosis key, and (iv) TAN and token retrieval are sensitive data items that should be protected from passive/active attackers. Also, intermediary entities such as CDN nodes and in-path web caches should not be allowed to access the content of messages too. *Aus <https://github.com/corona-warn-app/cwa-documentation/issues/93>*
23	Information disclosure	Access control	https://github.com/corona-warn-app/cwa-documentation/issues/223	By the architecture and protocol design of the verification process, both Verification Server (VS) and Corona-Warn-App Server (CWAS) can identify the Corona-Warn-App (CWA) uniquely within the protocol, since TAN is unique – which is of course necessary at some point to effect the verification. If VS and CWAS are under authority of or directly associated with the network operator (MNO), MNO and CWAS, respectively, VS can collude to identify a user submitting a valid (TAN, Diagnosis Key (DK)) pair uniquely, i.e., they can identify infected persons. *Aus <https://github.com/corona-warn-app/cwa-documentation/issues/223>*
24	Security guidelines umgehen -> Alle Kategorien möglich	threat	https://github.com/corona-warn-app/cwa-server/pull/124	• We include the cron trigger to ensure that there's always at least one analysis per week. While that's redundant while there's active development going on, it may be the only analysis which happens if active development ceases. Since the threat landscape is constantly changing, this ensures you learn about new threats in either case. • For the push event, we recommend that you run on all branches so that you can learn about newly introduced security

				issues in your pull requests. Running only on master means that the vulnerabilities can be introduced to your mainline before we detect them. *Aus <https://github.com/corona-warn-app/cwa-server/pull/124>*
25	Information Disclosure	Brute force	https://github.com/corona-warn-app/cwa-verification-server/issues/54	According to the architecture-overview, the teleTAN consists out of digits and uppercase letters only, in the code, one can see that it consists of digits, uppercase and lowercase letters. The implementation might be disadvantegeous when communicating the TAN over something like the phone, but increases security in terms of brute force attacks guessing a tan. *Aus <https://github.com/corona-warn-app/cwa-verification-server/issues/54>*
26	Tampering, DoS	mitigate	https://github.com/corona-warn-app/cwa-server/pull/1050	An integer overflow vulnerability exists with the length of websocket frames received via a websocket connection. An attacker would use this flaw to cause a denial of service attack on an HTTP Server allowing websocket connections. *Aus <https://www.cve.org/CVERecord?id=CVE-2020-27813>*
27	Information disclosure, Access Control, Elevation of Privilidge	mitigate	https://github.com/corona-warn-app/cwa-server/pull/914	In JUnit4 from version 4.7 and before 4.13.1, the test rule TemporaryFolder contains a local information disclosure vulnerability. On Unix like systems, the system's temporary directory is shared between all users on that system. Because of this, when files and directories are written into this directory they are, by default, readable by other users on that same system. This vulnerability does not allow other users to overwrite the contents of these directories or files.

				Aus <https://www.cve.org/CVERecord?id=CVE-2020-15250> Apache HttpClient versions prior to version 4.5.13 and 5.0.3 can misinterpret malformed authority component in request URIs passed to the library as java.net.URI object and pick the wrong target host for request execution. *Aus <https://www.cve.org/CVERecord?id=CVE-2020-13956>*
28	DoS	mitigate	https://github.com/corona-warn-app/cwa-server/pull/690	The payload length in a WebSocket frame was not correctly validated in Apache Tomcat 10.0.0-M1 to 10.0.0-M6, 9.0.0.M1 to 9.0.36, 8.5.0 to 8.5.56 and 7.0.27 to 7.0.104. Invalid payload lengths could trigger an infinite loop. Multiple requests with invalid payload lengths could lead to a denial of service. *Aus <https://www.cve.org/CVERecord?id=CVE-2020-13935>*
29	Information Disclosure, Spoofing	mitigate	https://github.com/corona-warn-app/cwa-server/pull/796	In Spring Framework versions 5.2.0 - 5.2.8, 5.1.0 - 5.1.17, 5.0.0 - 5.0.18, 4.3.0 - 4.3.28, and older unsupported versions, the protections against RFD attacks from CVE-2015-5211 may be bypassed depending on the browser used through the use of a jsessionid path parameter. *Aus <https://www.cve.org/CVERecord?id=CVE-2020-5421>*
30	Elevation of Privilidge	mitigate	https://github.com/corona-warn-app/cwa-server/pull/597	Use non-root user in distroless base images to run the applications. Due to the nature of distroless images, the potential threat should not be as high as for non-distroless images, but we want to mitigate this nevertheless. *Aus <https://github.com/corona-warn-app/cwa-server/pull/597>*

Tabelle 2: Sicherheitsmaßnahmen

Kategorie	Sicherheitsmaßnahmen (Behebungs-/Präventionsmöglichkeiten)	Beschreibung
Testen	We need to enforce static code scans to help us find security related issues early. *Aus <https://github.com/corona-warn-app/cwa-server/issues/13>* Static application security testing (SAST) *Aus <https://github.com/corona-warn-app/cwa-documentation/blob/main/overview-security.md#threat-tampering-diagnosis-keys>*	to secure software by reviewing the source code of the software to identify sources of vulnerabilities (White-Box Testing) *Aus <https://en.wikipedia.org/wiki/Static_application_security_testing>* Hilft nur wenn Code gescannt wird bevor er im master gepusht wird https://github.com/corona-warn-app/cwa-server/pull/124
Testen	Dynamic application security testing (DAST) *Aus <https://github.com/corona-warn-app/cwa-documentation/blob/main/overview-security.md#threat-tampering-diagnosis-keys>*	Unlike SAST tools, DAST tools do not have access to the source code and therefore detect vulnerabilities by actually performing attacks. (Black-Box Testing) *Aus <https://en.wikipedia.org/wiki/Dynamic_application_security_testing>*
Testen	Penetration testing *Aus <https://github.com/corona-warn-app/cwa-documentation/blob/main/overview-security.md#threat-tampering-diagnosis-keys>*	Umfassenden Sicherheitstest einzelner Rechner oder Netzwerke oder Datenbanken. Werkzeuge bilden bei Penetrationstests Angriffsmuster nach, die sich aus zahlreichen bekannten Angriffsmethoden ableiten lassen. *Aus <https://de.wikipedia.org/wiki/Penetrationstest_(Informatik)>*

Architektur & Testen	• Open-source software security testing • Regular open source software security • Usage of GitHub security alerts *Aus <https://github.com/corona-warn-app/cwa-documentation/blob/main/overview-security.md#threat-tampering-diagnosis-keys>* Add initial templates for contribution guidelines, code of conduct, and security policy. Contribution guidelines needs to be further refined with specifics from this implementation (e.g. code style/formatting). *Aus <https://github.com/corona-warn-app/cwa-server/pull/40>* Der Quellcode der App und der des Servers liegen öffentlich auf GitHub. • Öffentlicher Code -> Fehler können von allen entdeckt und nachgebessert aber auch leichter ausgenutzt werden • Außerdem: Potentielle Nutzer können sich ansehen inwiefern die Sicherheit garantiert ist und vertrauen der Software deshalb eher, oder weniger wenn aus ihrer Sicht nicht genug Schutz besteht • Lange und teilweise unsachliche Diskussionen über die Sicherheit müssen von Entwicklern reviewt werden um einen Nutzen daraus ziehen zu können • Transparenz an sich gut, es muss aber kontrolliert	Es gibt einige Diskussionen mit dem Label "review" die nie geschlossen wurden. https://github.com/corona-warn-app/cwa-documentation/issues/463 Szenario: Gefunden wurde die Lücke eher zufällig durch GitHubs Security Lab. Dessen Forscher hatten nach Mustern für "Java Bean Validation"-Lücken gesucht, um die Erkennungsmuster in die automatischen Code-Scanning-Werkzeuge der Plattform zu integrieren. Bei der Suche fanden sie auch die Lücke im Code für die Server der Corona-Warn-App. Dort wurde die Ausgabe einer Fehlermeldung als Code interpretiert. *Aus <https://www.heise.de/news/Corona-Warn-App-Sicherheitsluecke-im-Server-4964853.html>* Nginx Server gibt seine Version preis https://github.com/corona-warn-app/cwa-verification-server/issues/152 Server gibt in Exceptions zu viele Informationen über seine Funktionsweise preis https://github.com/corona-warn-app/cwa-verification-server/issues/4

	werden was alles öffentlich sein darf	
Architektur	Use of decentralized architecture based on Google, Apple and DP-3T approach *Aus <https://github.com/corona-warn-app/cwa-documentation/blob/main/overview-security.md#threat-tampering-diagnosis-keys>*	Bei Erstellung des Corona-Impfzertifikats werden Ihre Daten einmalig durch die Impfstelle erhoben und zur Signierung an das RKI übermittelt. Die Daten werden dort sofort wieder gelöscht. Wenn Sie das Corona-Impfzertifikat in der App hinzufügen, werden ihre persönlichen Daten nur lokal (dezentral) auf ihrem Smartphone gespeichert. Sie können selbst entscheiden, ob und wann Sie diese Daten auf Ihrem Gerät löschen. *Aus <https://github.com/corona-warn-app/cwa-website/issues/1311>*
Sourcecode	Input validation *Aus <https://github.com/corona-warn-app/cwa-documentation/blob/main/overview-security.md#threat-tampering-diagnosis-keys>* Output validation	Z.B. Begrenzung der Eingabelänge https://github.com/corona-warn-app/cwa-verification-server/issues/55 Was passiert wenn ein unerwarteter Wert zurückgegeben wird? https://github.com/corona-warn-app/cwa-server/issues/196
Schnittstelle	• TLS certificate pinning • TLS certificate validation *Aus <https://github.com/corona-warn-app/cwa-documentation/blob/main/overview-security.md#threat-tampering-diagnosis-keys>* • TLS verification • TLS encryption	Transport Layer Security (TLS) Secure Sockets Layer (SSL) Prevents man-in-the-middle attacks https://github.com/corona-warn-app/cwa-verification-server/issues/225 Kommunikation ist geschützt https://github.com/corona-warn-app/cwa-documentation/issues/93
Architektur	Infrastructure as code *Aus <https://github.com/corona-warn-app/cwa-documentation/blob/main/overview-*	Infrastructure as Code (IaC) is one of the main tenets of DevOps. Previously, manual configuration via cloud providers' UI consoles and physical hardware used to take place. But now, with the concept of IaC, the IT infrastructure can be automated by using

	security.md#threat-tampering-diagnosis-keys>	blueprints that are easily readable by machines. *Aus <https://www.sciencedirect.com/science/article/abs/pii/S1361372320301093#preview-section-abstract>*
Schnittstelle	TAN as one-time token for upload of diagnosis keys • Increase teleTAN complexity • Decrease teleTAN lifetime *Aus <https://github.com/corona-warn-app/cwa-documentation/blob/main/overview-security.md>* Begrenzen der maximalen Anzahl an TeleTANs	Wer positiv getestet wurde, kann dieses Ergebnis mittels eines QR-Codes oder einer teleTAN in die App eingeben. Den QR-Code erhält er bei der Probenentnahme, damit miss- bräuchliche Meldungen (Fehlalarme) vermieden werden. Dem- selben Zweck dient die teleTAN, die über eine Hotline abgefragt werden kann. Dix, A. Die deutsche Corona Warn-App – ein gelungenes Beispiel für Privacy by Design? . *Datenschutz Datensich* **44**, 779–785 (2020). https://doi.org/10.1007/s11623-020-1366-1 *Aus <https://link-1springer-1com-1skv2yfk3024f.shan01.han.tib.eu/article/10.1007/s11623-020-1366-1#citeas>* TeleTan zu lange gültig https://github.com/corona-warn-app/cwa-verification-server/issues/153 Mehr TeleTANs erstellbar als erlaubt https://github.com/corona-warn-app/cwa-verification-server/issues/32
Schnittstelle	• Gibt keine Telefonnummern oder persönliche Daten weiter -> zufällige Zahlenketten, gewissermaßen Pseudonyme *Aus <https://www.helmholtz.de/newsroom/artikel/wie-sicher-ist-die-corona-warn-app/>*	Zwei Arten von Zufallscodes: • Tages-/Geräteschlüssel (wechselt alle 24h) • kurzlebige Bluetooth-IDs (wechseln alle 10-15 min), kryptografisch aus dem Tagesschlüssel abgeleitet *Aus <https://www.rki.de/DE/Content/InfAZ/N/Neuartiges*

	• Nutzt permanent Bluetooth Low Energy	*Coronavirus/WarnApp/Funktion_Detail.pdf?__blob=publicationFile>*
Architektur	Temporäre Speicherung -> Geringerer Zeitraum um sensible Daten auszulesen	Speicherung der Zufallscodes: Die Zufallscodes werden für 14 Tage auf den Smart- phones der Nutzer/innen gespeichert. 14 Tage sind der Zeitraum, in dem von der App erfasste Begegnungen mit Corona-positiven Personen epidemiologisch relevant sein können. *Aus <https://www.rki.de/DE/Content/InfAZ/N/Neuartiges_Coronavirus/WarnApp/Funktion_Detail.pdf?__blob=publicationFile>*
Schnittstelle	Verschleierung des Zufallscodes um Rückschluss auf Infizierte Person zu vermeiden • Dummy packages for diagnosis key upload • Use mix network for anonymity during diagnosis keys upload *Aus <https://github.com/corona-warn-app/cwa-documentation/blob/main/overview-security.md>*	Übertragene Tagesschlüssel und Anzahl der bereit- gestellten Schlüssel: Aus Datenschutzgründen werden den übertragenen Schlüsseln weitere vom System erzeugte Schlüssel beigemischt. Dadurch wird gewährleistet, dass immer eine Mindestanzahl an Schlüsseln übertragen wird und keine Rückschlüsse auf einzelne Personen möglich sind. *Aus <https://www.rki.de/DE/Content/InfAZ/N/Neuartiges_Coronavirus/WarnApp/Funktion_Detail.pdf?__blob=publicationFile>*
Schnittstelle	Eingebundene Bibliotheken und Programme aktuell halten -> wenn diese kritische Sicherheitslücken beinhalten welche nicht durch neuere Versionen entfernt werden muss die Nutzung komplett unterbunden werden.	z.B. Log4j, 3rd-party-provider NETSYNO *Aus <https://github.com/corona-warn-app/cwa-documentation/issues/661>*

		https://github.com/corona-warn-app/cwa-quick-test-backend/issues/130
Sourcecode	Saubere Programmierung -> einheitliche Terminologie um Zusammenarbeit zu erleichtern -> eindeutige Variablennamen -> Hardcoden vermeiden wenn es nicht nötig ist	z.B. "key" und "password" wurden synonym benutzt obwohl signifikante Unterschiede bestehen was zu Verwirrung führen kann, hardgecodete Initialisierungen erleichtern das entschlüsseln weil Regelmäßigkeiten entstehen. *Aus <https://github.com/corona-warn-app/cwa-quick-test-backend/issues/68>* Hardcoded login credentials https://github.com/corona-warn-app/cwa-testresult-server/issues/30 Mehr TeleTans erstellbar durch "<=" anstatt "<" -> beginnt bei 0 zu zählen https://github.com/corona-warn-app/cwa-verification-server/issues/32
Sourcecode	Alle Nutzerdaten sollten Rücksetzbar/Änderbar/löschbar sein um mögliche Schäden bei einem Angriff zu minimieren	Passwort war nicht änderbar *Aus <https://github.com/corona-warn-app/cwa-verification-portal/issues/71>*
Dokumentation / Sourcecode	Die Dokumentation sollte mit der Implementierung übereinstimmen und aktuell gehalten werden	Verwirrung durch Angaben die der Code gar nicht erfüllt wird vermieden, man geht nicht fälschlicher Weise davon aus, dass etwas geschützt ist. *Aus <https://github.com/corona-warn-app/cwa-testresult-server/issues/65>* TeleTan laut Dokumentation nur eine Stunde gültig, in der Praxis länger https://github.com/corona-warn-app/cwa-verification-server/issues/153 Mehr Tans erstellbar als in Dokumentation angegeben https://github.com/corona-warn-app/cwa-verification-server/issues/32 Angeben wenn eine Funktion eine Anforderung nicht erfüllt bzw. präzise beschreiben was sie erfüllt

		https://github.com/corona-warn-app/cwa-server/issues/196 TeleTans bestehen nicht aus den Buchstaben und Zahlen wie in der Spezifikation angegeben https://github.com/corona-warn-app/cwa-verification-server/issues/54

Glossar

Bluetooth-ID Mehrmals pro Stunde erzeugter eindeutiger Schlüssel, der aus dem zufälligen Geräteschlüssel abgeleitet und über Bluetooth zwischen benachbarten Smartphones ausgetauscht wird. Synonyme: Bluetooth-ID; wechselnde Entfernungsschlüssel, Empfangsschlüssel, Sendeschlüssel; Rolling-Proximity-Identifier; Rotating-Proximity-Identifier Aus: CWA-Glossar [2]. 8, 9

Geräteschlüssel Täglich neu erzeugter, eindeutiger Schlüssel, aus dem mehrfach pro Stunde neue kurzlebige zufällige Bluetooth-IDs abgeleitet werden. Eine infizierte Person kann die zufälligen Geräteschlüssel der letzten 14 Tage als Positivkennungen freigeben und so mittels Risiko-Mitteilung andere über eine Begegnung mit einer Corona-positiv gestesteten Person informieren. Synonyme: Tagesschlüssel Aus: CWA-Glossar [2]. 8, 9, 21, 22

PCR-Test Ein Labortest zum Nachweis des SARS-CoV-2 Virus basierend auf der Polymerase-Kettenreaktion (polimerase chain reaction). Dieser Test wurde wegen seiner hohen Genauigkeit und Sensibilität genutzt. Im späteren Verlauf der Pandemie wurden auch Antigen-Schnelltests und Selbsttests zur Diagnose verwendet. [7]. 8

Risk Assessment Die Identifikation von Risiken und zu schützenden Attributen einer Software und Aufstellung von möglichen Sicherheitsmaßnahmen. Diese wird im Vorfeld der Entwicklung durchgeführt und kann während der Entwicklung bei signifikanten Veränderungen ergänzt und aktualisiert werden. . 18–20

STRIDE Ein Threat Model entwickelt von Microsoft um Bedrohungen zu gruppieren. Die Anfangsbuchstaben stehen für: Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, Elevation of Privilidge. (s. Abschnitt 2.2). 10, 25, 42

Symmetry of Ignorance Das Phänomen, dass alle Teilnehmenden einer Kommunikation davon ausgehen, dass die jeweils anderen einen Sach-

verhalt genauso verstehen wie sie, da es eine scheinbare Selbstverständlichkeit ist. Tatsächlich gibt es aber Missverständnisse dadurch, dass nicht jeder das gleiche Hintergrundwissen und die gleiche Perspektive besitzt. Solche Missverständnisse werden womöglich erst spät aufgedeckt und können zu Fehlern führen.. 30

TeleTAN Eine menschenlesbare Transaktionsnummer, welche über die telefonische Verifikations-Hotline der CWA ausgegeben wird und dazu dient ein positives Testergebnis in der App zu verifizieren. Sie dient als Alternative zum QR-Code mit der gleichen Funktion.. 9, 17, 18, 69

Threat Modeling Ähnelt dem Risk Assessment, ist aber spezifischer und wird durchgeführt sobald eine neue Komponente der Software hinzugefügt wird. Dabei werden Szenarien aus der Sicht von Angreifenden betrachtet, Risiken und Assets identifiziert und geeignete Gegenmaßnahmen direkt implementiert.. 18, 20

Abbildungsverzeichnis

Literaturverzeichnis

[1] Corona-Warn-App: Documentation. https://github.com/corona-warn-app/cwa-documentation. letzter Zugriff: Juni 2023.

[2] Corona-Warn-App: Glossar. https://www.bundesregierung.de/resource/blob/975226/1760836/fffa3f10830fe90d6d26453d6634af43/2020-06-16-glossar-data.pdf?download=1. letzter Zugriff: Juni 2023.

[3] CORONA-WARN-APP SOLUTION ARCHITECTURE. https://github.com/corona-warn-app/cwa-documentation/blob/main/solution_architecture.md. letzter Zugriff: Juni 2023.

[4] The STRIDE Threat Model. https://learn.microsoft.com/en-us/previous-versions/commerce-server/ee823878(v=cs.20)?redirectedfrom=MSDN, November 2009. letzter Zugriff: Juni 2023.

[5] So funktioniert die Corona-Warn-App im Detail. https://www.rki.de/DE/Content/InfAZ/N/Neuartiges_Coronavirus/WarnApp/Funktion_Detail.pdf?__blob=publicationFile, Juli 2020. letzter Zugriff: Juni 2023.

[6] ISO/IEC 25010. https://iso25000.com/index.php/en/iso-25000-standards/iso-25010, 2022. letzter Zugriff: Juli 2023.

[7] I. M. Artika, Y. P. Dewi, I. M. Nainggolan, J. E. Siregar, and U. Antonjaya. Real-time polymerase chain reaction: Current techniques, applications, and role in covid-19 diagnosis. *Genes*, 13(12), 2022.

[8] S. Bistarelli, F. Fioravanti, and P. Peretti. Defense trees for economic evaluation of security investments. In *First International Conference on Availability, Reliability and Security (ARES'06)*, pages 8 pp.–423, 2006.

[9] L. Braz and A. Bacchelli. Software security during modern code review: The developer's perspective. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2022, page 810–821, New York, NY, USA, 2022. Association for Computing Machinery.

[10] Bundesamt für Sicherheit in der Informationstechnik. Die Lage der IT-Sicherheit in Deutschland 2022. https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/Lageberichte/Lagebericht2022.pdf?__blob=publicationFile&v=8, Oktober 2022. letzter Zugriff: Mai 2023.

[11] D. L. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2):84–90, 1981.

[12] CWA-Team. Science-Blog. https://www.coronawarn.app/de/science/. letzter Zugriff: Juni 2023.

[13] A. Dix. Die deutsche Corona Warn-App – ein gelungenes Beispiel für Privacy by Design? *Datenschutz und Datensicherheit - DuD*, November 2020.

[14] D. G. Firesmith. *Common concepts underlying safety, security, and survivability engineering.* Carnegie Mellon University, Software Engineering Institute Pittsburgh, Pa, USA, 2003.

[15] Z. für digitalen Fortschritt. Verein für liberale Netzpolitik, Forum InformatikerInnen für Frieden und Gesellschaftliche Verantwortung, Gesellschaft für Informatik, Chaos Computer Club, Stiftung Datenschutz (2020) Offener Brief: Geplante Corona-App ist höchst problematisch. 24. April. https://www.ccc.de/system/uploads/299/original/Offener_Brief_Corona_App_Bundeskanzleramt.pdf, 2020. letzter Zugriff: August 2023.

[16] F. J. Furrer. Software. *Informatik-Spektrum*, 40:264–269, Juni 2017.

[17] F. J. Furrer. *Software Everywhere*, pages 3–10. Springer Fachmedien Wiesbaden, Wiesbaden, 2019.

[18] D. Manzey. *Systemgestaltung und Automatisierung*, pages 333–352. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[19] J. A. McCall, P. K. Richards, and G. F. Walters. Factors in software quality. volume 2. metric data collection and validation. Technical Report ADA049015, GENERAL ELECTRIC CO SUNNYVALE CA, November 1977. letzter Zugriff: August 2023.

[20] N. Mead, E. Hough, and T. S. II. Security quality requirements engineering technical report. Technical Report CMU/SEI-2005-TR-009, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 2005.

[21] mgm security partners GmbH. Ergebnisse der Studie zur statischen Codeanalyse ausgewählter Opensource Software. https://www.

`bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/Studien/P486-Codeanalyse/Videokonferenzsysteme.pdf?__blob=publicationFile&v=3`, Juni 2023. letzter Zugriff: August 2023.

[22] J. P. Miguel, D. Mauricio, and G. Rodríguez. A review of software quality models for the evaluation of software products. *International Journal of Software Engineering & Applications*, 5(6):31–53, nov 2014.

[23] Mitwirkende Personen des Open-Source-Projekts für Corona-Warn-App. Corona-Warn-App Open-Source-Projekt. `https://www.coronawarn.app/de/`, 2020-2023. letzter Zugriff: Juni 2023.

[24] B. Naqvi, A. Seffah, and A. Abran. Framework for examination of software quality characteristics in conflict: A security and usability exemplar. *Cogent Engineering*, 7(1):1788308, 2020.

[25] F. E. Oguz, M. N. Ekersular, K. M. Sunnetci, and A. Alkan. Can chat gpt be utilized in scientific and undergraduate studies? *Annals of Biomedical Engineering*, pages 1–3, 2023.

[26] R. K. Parkin. The importance of it security. *Computer Fraud & Security*, 1998(3):12–15, 1998.

[27] M. Rohr. *Sicherheitsuntersuchungen von Webanwendungen*, pages 345–431. Springer Fachmedien Wiesbaden, Wiesbaden, 2018.

[28] K. Schneider. *Abenteuer Softwarequalität: Grundlagen und Verfahren für Qualitätssicherung und Qualitätsmanagement*. dpunkt. verlag, 2012.

[29] B. Schneier. Cryptographic design vulnerabilities. *Computer*, 31(9):29–33, 1998.

[30] T. Stadler, W. Lueks, K. Kohls, and C. Troncoso. Preliminary analysis of potential harms in the luca tracing system. *CoRR*, abs/2103.11958, 2021.

[31] T. W. Thomas, M. Tabassum, B. Chu, and H. Lipford. Security during application development: An application security expert perspective. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, CHI '18, page 1–12, New York, NY, USA, 2018. Association for Computing Machinery.

[32] V. Viegas and O. Kuyucu. *The Cybersecurity Challenge*, pages 1–16. Apress, Berkeley, CA, 2022.

[33] A. Wirth. Log jam: Lesson learned from the log4shell vulnerability. *Biomedical Instrumentation & Technology*, 56(3):72–76, 2022.

[34] World Health Organization. Timeline: Who's covid-19 response. https://www.who.int/emergencies/diseases/novel-coronavirus-2019/interactive-timeline#event-72, 2023. letzter Zugriff: Juni 2023.