cyberagentur

HOCHSCHULE MITTWEIDA
University of Applied Sciences

BACHELORARBEIT

Frau
Joline Wochnik

Untersuchung der Entwicklung von IT-Sicherheitslücken anhand von CVE-Daten

Mittweida, September 2023

Fakultät **Angewandte Computer- und Biowissenschaften**

BACHELORARBEIT

Untersuchung der Entwicklung von IT-Sicherheitslücken anhand von CVE-Daten

Autorin:
Joline Wochnik

Studiengang:
Allgemeine und Digitale Forensik

Seminargruppe:
FO20w2-B

Erstprüfer:
Prof. Dr. rer. nat. Michael Spranger

Zweitprüfer:
Prof. Dr. rer. nat. Christian Hummert

Einreichung:
Mittweida, 18.09.2023

Verteidigung/Bewertung:
Mittweida, 2023

Faculty of **Applied Computer Sciences and Biosciences**

BACHELOR THESIS

Investigation of the growth of vulnerabilities based on CVE data

Author:
Joline Wochnik

Course of Study:
Forensics

Seminar Group:
FO20w2-B

First Examiner:
Prof. Dr. rer. nat. Michael Spranger

Second Examiner:
Prof. Dr. rer. nat. Christian Hummert

Submission:
Mittweida, 18.09.2023

Defense/Evaluation:
Mittweida, 2023

Bibliografische Beschreibung:

Wochnik, Joline:
Untersuchung der Entwicklung von IT-Sicherheitslücken anhand von CVE-Daten. – 2023. – 67 S.
Mittweida, Hochschule Mittweida – University of Applied Sciences, Fakultät Angewandte Computer- und Biowissenschaften, Bachelorarbeit, 2023.

Referat:

Die vorliegende Arbeit beschäftigt sich mit der Fragestellung ob und warum die Anzahl von IT-Sicherheitslücken exponentiell steigt. In Zuge dessen wird der Zusammenhang zwischen der Entwicklung von Sicherheitslücken und der Entwicklung der Code-Länge über die Zeit untersucht. Um die Forschungsfrage zu beantworten, wurden CVE-Daten ausgewertet. Dabei wird nicht nur die allgemeine CVE-Entwicklung, sondern auch ausgewählte Software-Systeme im Speziellen betrachtet. Um die Code-Länge zu untersuchen, wurden die LOC der einzelnen Software-Systeme analysiert. Die Untersuchung ergab, dass die Entwicklung der CVE gesamt einen exponentiellen Trend verfolgt. Die CVE-Entwicklung der einzelnen Software-Systeme verfolgt im Gegensatz dazu in der Mehrheit einen linearen Trend. Auch die LOC-Entwicklung der Software-Systeme passt sich diesem Trend an. Somit zeigt sich, dass die drastische Entwicklung der Sicherheitslücken nicht ausschließlich durch eine drastische LOC-Entwicklung beeinflusst wird, sondern von verschiedenen Faktoren abhängt, die sich untereinander verstärken.

Abstract:

This bachelor thesis addresses the question of whether and why the number of IT security vulnerabilities is increasing exponentially. In the course of this, the correlation between the development of security vulnerabilities and the development of code length over time is examined. To answer the research question, CVE data was analyzed. This includes not only the general CVE evolution but also the CVE evolution of selected software systems in particular. To investigate the code length, the LOC of the software systems were examined. The investigation revealed that the overall CVE development follows an exponential trend. In contrast, the CVE and LOC development of the majority of the individual software systems follow a linear trend. Thus, it can be seen that the drastic development of security vulnerabilities is not exclusively influenced by a drastic LOC development but depends on various factors that reinforce each other.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

AML ..... Alhazmi-Malaiya Logistic Model

BSI ..... Bundesamt für Sicherheit in der Informationstechnik

CISA ..... Cybersecurity and Infrastructure Security Agency

CISQ ..... Consortium for Information & Software Quality

CNA ..... CVE Numbering Authority

CNNVD ..... Chinese National Vulnerability Database

CVE ..... Common Vulnerabilities and Exposures

CVSS ..... Common Vulnerability Scoring System

CWE ..... Common Weakness Enumeration

DHS ..... US Department of Homeland Security

FIRST ..... Forum of Incident Response and Security Teams

IDE ..... integrierte Entwicklungsumgebung

JVN ..... Japan Vulnerability Notes

LoC ..... Lines of Code

NCF ..... National Cybersecurity Federal Funded Research and Development Center

NIAC ..... National Infrastructure Advisory Council

NIST ..... National Institute of Standards and Technology

NVD ..... National Vulnerability Database

OSBA ..... Open Source Business Alliance

OSVDB ..... Open Source Vulnerability Database

SSDF ..... Secure Software Development Framework

URL ..... Uniform Resource Locator

VDM ..... Vulnerability Discovery Model

Web ..... World Wide Web

1 Einleitung

1.1 Motivation

„The hacker didn’t succeed through sophistication. Rather he poked at obvious places, trying to enter through unlock[ed] doors. Persistence, not wizardry, let him through.“ [1, S. 126]

Tatsächlich haben die meisten Softwareprojekte so viele Sicherheitslücken, dass es eher eine Frage der Zeit ist, bis ein Angreifer eine der Sicherheitslücken ausnutzt, als der Kompetenz des Angreifers. Es kann nicht ausgeschlossen werden, dass eine vorhandene Sicherheitslücke nicht ausgenutzt wird, wodurch sie eine Risiko für die Nutzenden darstellen. [2, nach Derin/Golla, NJW 2019, 1111 (1115)]

Die Agentur für Innovation in der Cybersicherheit GmbH[1], im Sprachgebrauch Cyberagentur genannt, versucht „Schlüsseltechnologien und bahnbrechende Innovationen nutzbar zu machen, die dabei helfen, die innere und äußere Sicherheit zu ermöglichen und zu verbessern“ [3]. Um dies zu erreichen, werden wagnisbehaftete Forschungen beauftragt und von den Experten der Cyberagentur begleitet. Dabei wird der Fokus auf die Cybersicherheit Deutschlands und Europas in 10 bis 15 Jahren gerichtet [3]. Es geht darum, die Zukunft des Cyberraums Deutschlands sicherer zumachen. Auch IT-Sicherheitslücken sind dabei ein wichtiges Thema. Unter Betreuung der Abteilung „Sichere Gesellschaft“ der Cyberagentur befasst sich diese Arbeit mit der Analyse der Entwicklung von IT-Sicherheitslücken.

1.2 Ziele

Die Arbeit beschäftigt sich mit der Forschungsfrage, ob und warum die Anzahl von IT-Sicherheitslücken exponentiell wächst. Dazu wird ein möglicher Zusammenhang von Common Vulnerabilities and Exposures (CVE)- und Lines of Code (LoC)-Entwicklung erforscht.

Folgende Thesen werden geprüft:

These 1: Die Anzahl der IT-Sicherheitslücken steigt exponentiell.

These 2: Die Anzahl der IT-Sicherheitslücken steigt exponentiell, weil die Code-Länge exponentiell wächst.

These 3: Wenn die Entwicklung der Code-Länge und das Auftreten von Sicherheitslücken von Software über die Zeit korreliert, dann ist ein bestimmender Faktor für das Wachstum der Sicherheitslücken das Wachstum des Codes einer Software.

[1] https://www.cyberagentur.de/

1.3 Aufbau der Arbeit

Folgend wird der Aufbau der Arbeit erläutert. In Kapitel 2 wird ein Überblick zu IT-Sicherheitslücken, deren Lebenszyklus, Detektion und Verwaltung gegeben. Anschließend werden in Kapitel 3 die genutzten Methoden schrittweise aufgeführt. Die Ergebnisse dieser Methoden werden in Kapitel 4 zusammengefasst und nachfolgend in Kapitel 5 ausführlich diskutiert. Abschließend wird in Kapitel 6 ein Fazit gezogen und ein kurzer Ausblick in mögliche weitere Forschung gegeben.

2 Grundlagen

Der Grundlagenteil dient als Überblick zu IT-Sicherheitslücken. Der nachfolgende Abschnitt 2.1 beschäftigt sich mit ihrer Definition. Abschnitt 2.2 widmet sich dem Lebenszyklus von IT-Sicherheitslücken und deren Bezug zum Lebenszyklus von Software-Versionen, während Abschnitt 2.3 die Gefährdung durch IT-Sicherheitslücken betrachtet. Anschließend befassen sich die Abschnitte 2.4 und 2.5 mit ihrer Detektion und Verwaltung. Abschnitt 2.6 thematisiert das Thema Sicherheit bei der Software Entwicklung. Die abschließend in Abschnitt 2.7 aufgeführten Untersuchungsgrundlagen bilden die Basis für das Kapitel 3, welches die verwendeten Methoden aufführt.

2.1 IT-Sicherheitslücken

IT-Sicherheitslücken, auch Schwachstellen genannt, sind eine häufige Ursache für Angriffe auf Software und können zu beträchtlichen Schäden führen [4][5]. Ihre Ausnutzung bildet einen Großteil aller Sicherheitsvorfälle in Software-Produkten [6], wodurch sie ein erhebliches Sicherheitsrisiko darstellen [7]. Sicherheitslücken werden vom Bundesamt für Sicherheit in der Informationstechnik (BSI)[2] daher als „eine der wesentlichen Ursachen für IT-Sicherheitsvorfälle“ benannt [8, S. 1]. Die Ziele der Angreifer variieren zwischen Kontrolle über das System zu erlangen und für ihre Zwecke interessante Informationen zu beziehen [5]. Zu den potentiellen Schäden durch Sicherheitslücken zählt auch die Gefährdung der Schutzziele der IT-Sicherheit, wie Verfügbarkeit oder Integrität des Produktes, durch beispielsweise Systemzusammenbrüche, Veränderung der Daten oder ungewollte Verhaltensweisen des Systems [4], [6]. Im Allgemeinen gilt, je später eine Sicherheitslücke gefunden wird, desto aufwendiger ist ihre Beseitigung und die Behebung des entstandenen Schadens [5].

Für Sicherheitslücken gibt es keine einheitliche Definition, weshalb viele Gelehrte eigene Beschreibungen aufgestellt haben [9]. Zu den bekanntesten zählen die Definition von Schultz et al. sowie die Definition von Charles P. Pfleeger. 1990 beschrieben Schultz et al. Sicherheitslücken als „defect which enables an attacker to bypass security measures” [10]. Pfleeger definierte Sicherheitslücken als „a weakness in the security system that might be exploited to cause loss or harm“ [11]. Der National Infrastructure Advisory Council (NIAC)[3] nutze 2004 in ihrem Vulnerability Disclosure Framework Abschlussbericht folgende Definition: „A vulnerability is defined as a set of conditions that may lead to an implicit or explicit failure of the confidentiality, integrity, or availability of an information system“ [12, S. 13]. In einem 2018 veröffentlichen Paper fassen Y. Singh und P. Kumar nach Betrachtung der verschiedenen Definitionen, Sicherheitslücken folgendermaßen zusammen. Nach ihnen sind Sicherheitslücken „an imperfection in the evolution of software that can be exploited by an assailant with a specific end goal to get a few

[2] https://www.bsi.bund.de/DE/Home/home_node.html

[3] https://www.cisa.gov/resources-tools/groups/presidents-national-infrastructure-advisory-council-niac

benefits in the information system“ [9, S. 16]. Die unterschiedlichen Definitionen führen im Kern zum gleichen Ergebnis. Sicherheitslücken können als Eingang in die jeweilige Software betrachtet werden [5][9].

Zusammenfassend wird eine Software somit als riskanter angesehen, je mehr Sicherheitslücken im Softwaresystem enthalten sind [4]. Um dies normalisiert darzustellen gibt es das Maß der *Vulnerability Density* (dt. Schwachstellendichte, V_d). Dieses Maß ergibt sich aus der Anzahl der Sicherheitslücken (V) pro Codeeinheit (S) und bildet damit eine Vergleichsgrundlage für Qualität und Sicherheit von Softwarecode [7].

$$V_d = \frac{V}{S}$$

Dabei ist zu beachten, dass stets von einer unbekannten Menge an noch nicht entdeckten Sicherheitslücken ausgegangen werden muss. Unentdeckte Sicherheitslücken stellen somit, neben allen bekannten Sicherheitslücken, ein hohes, schwer einschätzbares Risiko für Software-Systeme dar. [13]

2.2 IT-Sicherheitslücken: von Entdeckung bis Behebung

2.2.1 Software-System in Bezug auf IT-Sicherheitslücken

Sicherheitslücken machen eine Software angreifbar [9]. Allerdings ist das Entstehen von Sicherheitslücken bei der Software-Entwicklung, durch die gestiegene, hohe Komplexität von gegenwärtiger Software, kaum vermeidbar [8] und die Fehlerfreiheit von Programmen kann nicht garantiert werden [5][14][13]. Um die Sicherheit von Software und Anwendern zu verbessern, müssen Sicherheitslücken identifiziert und die Fehler korrigiert werden [6]. Dabei gilt, auch wenn ein Großteil von Sicherheitslücken isoliert werden kann, ist es doch unmöglich, sie vollständig zu beseitigen [13]. Je mehr Sicherheitslücken in einer Software existieren und je mehr Risiken dadurch entstehen, desto unsicherer gilt die Software [15]. Die Anzahl der Sicherheitslücken kann somit als ein „Schlüsselattribut“ [4, S. 247] der Sicherheit der jeweiligen Software betrachtet werden. Auf Grund dessen kann die Entdeckungsrate von Sicherheitslücken mit der Zuverlässigkeit (engl. reliability) der Software gleichgestellt werden [4], die nach Michael R. Lyu der wichtigste Aspekt der Qualität einer Software ist [16].

Die Bemühungen Sicherheitslücken zu erkennen und auszunutzen, variieren im Laufe des Lebens einer Software [4]. Laut Alhazmi et al. [13] ist die Aufdeckungsrate von Sicherheitslücken eng mit der Nutzungsumgebung der Software verbunden: Je mehr Versionen für eine Software veröffentlicht werden, desto mehr beginnt die Zahl der verschiedenen installierten Versionen zu wachsen. Je größer diese installierte Basis ist, desto größer ist die Belohnung für das Auffinden von Sicherheitslücken. Somit steigt das Ausmaß der Bemühungen eine Sicherheitslücke zu finden, wodurch wiederum die Anzahl der entdeckten Sicherheitslücken steigt. lhazmi et al. gehen daher davon aus, dass,

nachdem die Software lange genug im Einsatz ist, ein „Großteil der ursprünglich vorhandenen Sicherheitslücken“ [13, S. 2] entdeckt wurde. Dabei bilden diese ursprünglichen Sicherheitslücken einen Großteil aller insgesamt entdeckten Sicherheitslücken.

Unter Betrachtung dieser Abhängigkeit von Sicherheitslücken einer Software und ihrer Nutzungsumgebung ist, unter Einbezug der Entdeckungen von Sicherheitslücken, ein Rückschluss auf den Entwicklungsstand der Software möglich [13]. Omar H. Alhazmi und Yashwant K. Malaiya fassen diese Beobachtung, als Beziehung zwischen der Gesamtmenge an entdeckten Sicherheitslücken und Zeit, in einem „time-based known-vulnerability discovery model“ [7] zusammen. Das Modell stützt sich auf die Veränderung der Nutzerbasis im Laufe der Zeit und den damit verbundenen Bemühungen Sicherheitslücken aufzudecken [13]. Obwohl in dem 2005 erschienenen Paper noch von Betriebssystemen die Rede ist, wird es in Papern darauffolgender Jahre gleichsam für Software-Systeme angewendet [4] . Nach dem beschriebenen logistischen Modell mit dem Namen Alhazmi-Malaiya Logistic Model (AML) durchläuft ein System während seines Lebens verschiedene Phasen, darunter Veröffentlichung, steigende Popularität, Stabilität, abnehmende Popularität und schließlich Veraltung [7]. AML fasst dies in drei Phasen zusammen [4], [7], [13].

In Abbildung 2.1 ist das allgemeine AML 3-Phasen Modell dargestellt und wird nachfolgend in Bezug auf Software-Systeme beschrieben. Die Übergänge zwischen den drei Phasen sind dabei als „Transition point“s gekennzeichnet. In der ersten Phase, der Lern-Phase, sammeln Software-Tester Erfahrungen mit dem System und beginnen Sicherheitslücken zu entdecken. Somit steigt die Rate der Entdeckungen und Phase zwei, die lineare Phase, beginnt. Mit wachsender Nutzeranzahl nimmt die Anzahl der Entdeckungen von Sicherheitslücken zu. Die Zunahme an Entdeckungen von Sicherheitslücken verläuft, bis die Software auf ihrem Popularitäts-Höhepunkt angekommen ist, linear. In der linearen Phase ist das Entdecken von Sicherheitslücken, aufgrund der hohen Nutzung des Systems, am lohnendsten. Nach einiger Zeit wird die Software durch eine neuere Version abgelöst. Dies beschreibt den Start der dritten Phase des Modells. Durch das Vorhandensein einer neueren Version nimmt die Popularität der derzeitigen Software-Version ab. Durch die geringere Nutzung der nun veralteten Version wird die Entdeckung von Sicherheitslücken weniger belohnend, wodurch weniger Mühe in deren Aufdeckung gesteckt wird. Diese Phase trägt daher den Namen Sättigungs-Phase. [4], [7], [13]

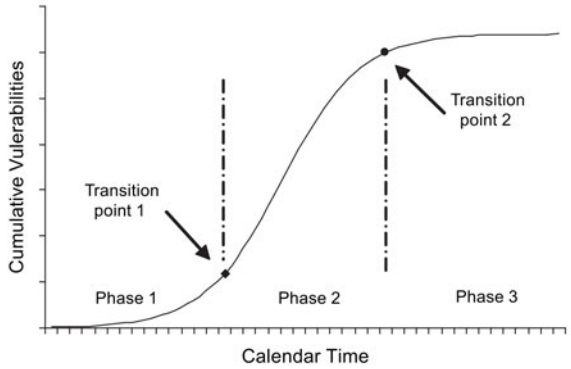


Abbildung 2.1: Drei-Phasen-Model für die Realtion von Sicherheitslücken-Entdeckungsrate und Zeit [13]

In einem Paper von 2007 [13] erweitern Omar H. Alhazmi und Yashwant K. Malaiya ihr vorangegangenes Modell. Dabei beziehen sie sich auf den Anwendungsfall, dass eine ältere Version, welche den benannten Prozess und die damit verbundenen Phasen durchlaufen hat, eine beträchtliche Anzahl an Sicherheitslücken mit einer späteren Version teilt. Da bereits eine Menge an Sicherheitslücken im Code der alten Version, welcher teils in die neuere Version übernommen wurde, entdeckt wurden, kommt es nach Alhazmi und Malaiya zu mehr als einer S-Kurve oder auch Überlagerungen in dem Lebensverlauf der neueren Version. Sie schließen, dass sich der geteilte Code verschiedener Versionen auf die Entdeckungsrate ihrer Sicherheitslücken auswirkt. [13]

Um das Risiko potentieller Sicherheitslücken einer Software sowie die zur Behebung nötigen Ressourcen abschätzen zu können, werden unter anderem Vulnerability Discovery Model (VDM)s eingesetzt [4]. VDMs stellen einen Versuch dar, die Rate, mit der Sicherheitslücken einer Software entdeckt werden, zu modellieren [17], wodurch ein Einblick in die Sicherheit der Software ermöglicht wird [18].

2.2.2 Lebenszyklus

„Jede Sicherheitslücke ist einzigartig, aber alle durchlaufen einen vorhersehbaren Lebenszyklus“ [12, S. 7]. Der Zyklus des Lebens von Sicherheitslücken variiert dabei, je nachdem welche Intention ihr Entdecker verfolgt und welches Selbstverständnis diese Person hat [8]. Grundsätzlich kann zwischen gut gemeinter Behebung und böswilligen Akteuren unterschieden werden. Es gibt keinen bindenden Standard, wie, wann und an wen die Entdeckung von Sicherheitslücken offengelegt und weitergegeben werden soll [12]. Neben der Motivation des Entdeckers ist ebenfalls die Reaktion der Produktverantwortlichen ein bedeutender Faktor, der die Gefährdung, die von der Sicherheitslücke ausgeht, beeinflusst. Es kommt auf korrekte Einschätzung, schnelles Handeln und resolute Abhilfe an. [8]

Das BSI hat 2018 zu den aus den verschiedenen Entdeckungsarten resultierenden unterschiedlichen Lebenszyklen von Sicherheitslücken ein Schema veröffentlicht [8]. Wie in Abbildung 2.2 dargestellt, startet die Betrachtung einer Sicherheitslücke mit ihrer Entdeckung. In dem, vom BSI vorgestellten, Schema wird von dem allgemeinen Fall ausgegangen, dass die Sicherheitslücke von einem Dritten entdeckt wird.

Darauf aufbauend gibt es verschiedene Möglichkeiten, wie der Produktverantwortliche von der Existenz der Sicherheitslücke erfahren kann [8]. Bei einem sogenannten *Full Disclosure* (dt. Vollständige Offenlegung von Informationen) veröffentlicht der Entdecker alle Informationen, die er über die Sicherheitslücke erlangt hat. Wenn der Entdecker vor einer Veröffentlichung in direkten Kontakt mit dem jeweiligen Produktverantwortlichem tritt und dieser somit die Möglichkeit hat, die Veröffentlichung zu beaufsichtigen oder auch vorerst zu unterbinden, wodurch nicht alle oder gar keine Informationen über die Sicherheitslücke an die Öffentlichkeit kommen, wird von *Coordinated Disclosure* (dt. Koordinierte Offenlegung von Informationen) gesprochen. Es ist zudem möglich, dass die Sicherheitslücke dem Produktverantwortlichem vorenthalten und auch nicht direkt

mit allen Informationen veröffentlicht wird, zu Gunsten des Entdeckers mit meist böswilliger Intention. Diese Art von Sicherheitslücke wird *Zero-Day-Exploit* genannt. Erst durch ihre Ausnutzung und den daraus resultierenden Angriffen erfährt der Produktverantwortliche von ihrer Existenz. Wenn Informationen über eine Sicherheitslücke nicht direkt veröffentlicht werden, kann der Produktverantwortliche auch ohne direkte Angriffe von ihr über sogenannte *Schwachstellen-Broker* (dt. Schwachstellen-Händler) erfahren. Schwachstellen-Broker betreiben zu eigenem Profit Zwischenhandel mit Informationen zu Sicherheitslücken. Um die bereitwillige Freigabe von Informationen zu gefundenen Sicherheitslücken zu unterstützen, gibt es von manchen Produktverantwortlichen und Herstellern sogenannte *Bug Bounties*, durch welche das Melden von Sicherheitslücken als *Coordinated Disclosure* geldlich entlohnt wird.

IT-Sicherheitsexperten stellen das Zusammenwirken mittels *Coordinated Disclosure* als notwendig dar. Hindernisse einer erfolgreichen Inkenntnissetzung der Produktverantwortlichen bezüglich Sicherheitslücken ihrer Produkte sind allerdings häufig die Identifikation von Produktverantwortlichen und ihren Ansprechpartner sowie unzureichende Kommunikation. [2]

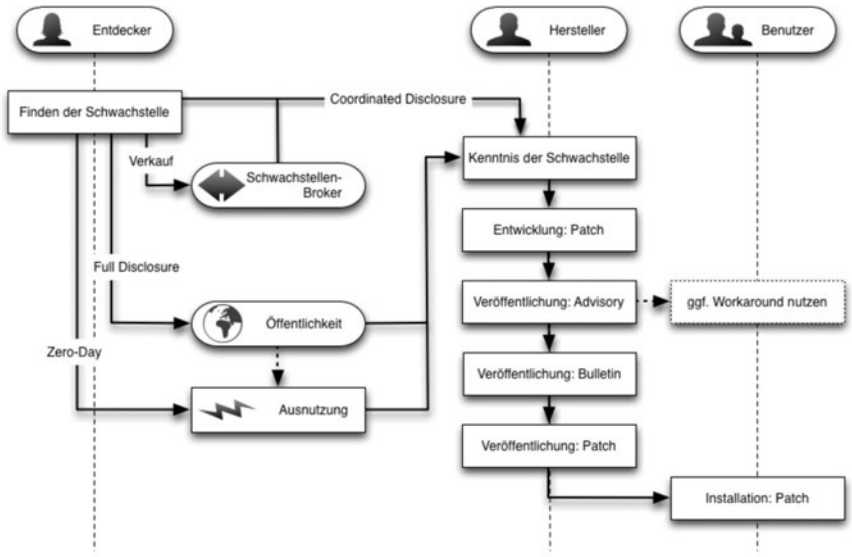


Abbildung 2.2: Schematischer Lebenszyklus einer Sicherheitslücke [8]

Dem Verlauf von Abbildung 2.2 folgend, wird mit der Entwicklung eines Patches (dt. flicken) begonnen, nachdem der Produktverantwortliche Kenntnis von der Existenz der Sicherheitslücke erlangt hat. Durch diesen soll die Sicherheitslücke „geflickt“ und die Gefährdung, die mit ihrer Existenz einhergeht, behoben werden. Vor einem Patch wird ein sogenanntes Advisory veröffentlicht. Advisorys sind Warnungen von Produktverantwortlichen, bei denen der Öffentlichkeit die Sicherheitslücke und die dadurch entstehenden Gefahren bekanntgemacht werden. Advisorys können auch erst gleichzeitig mit dem Patch veröffentlicht werden, je nach Einschätzung der Gefährdungs-Lage durch den jeweiligen Produktverantwortlichen. Zusammen mit dem Patch eine passende Beschreibung („Bulletin“) veröffentlicht. Mit der Installation des Patches endet die Darstellung des Lebenszyklus der Sicherheitslücke. [8]

Das Leben einer Sicherheitslücke wird als abgeschlossen betrachtet, wenn ein hoher Prozentsatz der Nutzer der betroffenen Systeme den dazugehörigen Patch installiert hat, die Software veraltet ist oder die Sicherheitslücke sowie deren Patch seit langem bekannt sind [12].

Meist folgt die Veröffentlichung eines Patches allerdings nicht direkt auf die Entdeckung einer Sicherheitslücke. Nach einer Messung an über 3000 Sicherheitslücken fand eine Studie heraus, dass ein Drittel aller Sicherheitslücken erst drei Jahre nach ihrer Entdeckungen gepatchet werden. Zudem erhielten 7% der analysierten Sicherheitslücken gar keinen Patch. Dies stellt ein großes Risiko dar, da Patches für den Schutz von Daten und Nutzenden wesentlich sind. [19]

Es ist anzumerken, dass das Schema nur einen allgemeinen Ablauf darstellt, der sich dynamisch ändern kann [8]. Zudem können nicht immer alle Sicherheitslücken geschlossen werden. Auf diesen Fakt aufbauend sowie bekennend der riesigen Anzahl auftretender Sicherheitslücken, erklärt das Bundesministerium des Inneren, für Bau und Heimat in ihrer „Cybersicherheitsstrategie für Deutschland 2021“, dass Bedarf an verbesserten Qualitätssicherungsprozessen besteht. Diese sollen der Verminderung des Aufkommens von Sicherheitslücken vor der Veröffentlichung der jeweiligen Produkte dienen. [20]

2.3 Gefahren durch IT-Sicherheitslücken

2.3.1 Common Vulnerability Scoring System

Um die Behebung von Sicherheitslücken mit den potenziell schwerwiegendsten Auswirkungen zu priorisieren und somit die kritische Infrastruktur weitest möglich schützen zu können, ist eine einheitliche Methode zur zuverlässigen Bewertung von Sicherheitslücken maßgeblich [12]. 2004 wurde daher, zum besseren Verständnis von Sicherheitslücken und deren Auswirkungen, von dem NIAC ein „offenes und allgemeingültiges System zur Bewertung von Sicherheitslücken“ [21, S. 2] für die internationale Anwendung vorgestellt: das Common Vulnerability Scoring System (CVSS)[4]. Es basiert auf der Definition von Sicherheitslücken, welche der NIAC [12] in dem Vulnerability Disclosure Framework Abschlussbericht formulierte. CVSS ist ein offenes [22] Framework zur Einstufung von Sicherheitslücken. Dabei wird für Sicherheitslücken jeweils eine Punktzahl, engl. Score, bestimmt, die den Schweregrad sowie das Risiko, welches von den Sicherheitslücken ausgeht, darstellt. [21]

Der Begriff „Risiko“ wurde von dem National Institute of Standards and Technology (NIST)[5] im Zusammenhang mit Sicherheitslücken als die „relativen Auswirkungen, die eine ausgenutzte Sicherheitslücke auf die Umgebung eines Benutzers haben würde“ [22, S. 4] beschrieben.

[4] https://www.first.org/cvss/
[5] https://www.nist.gov/

CVSS steht unter der Obhut des Forum of Incident Response and Security Teams (FIRST)[6] [22]. Es ist gegenwärtig der am weitesten verbreitete und am meisten genutzte Standard zur quantitativen Bewertung des Schweregrades von Sicherheitslücken [23] und ist derzeit in Version CVSS v3.1 verfügbar [24].

Abbildung 2.3 zeigt den Aufbau des CVSS-Systems. Alle Berechnungen werden automatisch im Hintergrund ausgeführt, sind dabei aber für den Nutzer transparent. CVSS wird durch ein modulares System mit drei verschiedenen Gruppen beschrieben. Die Gruppen verfügen jeweils über eine separate Formel, zum Kombinieren und Gewichten der Metriken der zu bewertenden Sicherheitslücke [21]. Metriken beschreiben Merkmale bzw. Komponenten einer Sicherheitslücke, welche quantitativ oder qualitativ gemessen werden können.

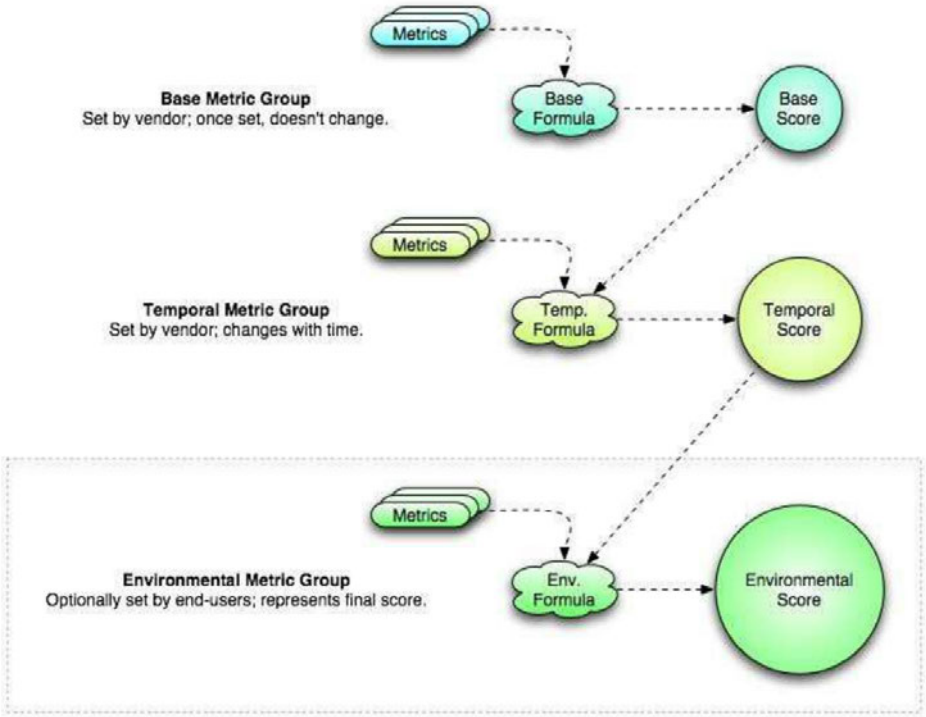


Abbildung 2.3: Schematisches CVSS Modell [21]

Wie in Abbildung 2.4 dargestellt, befassen sich die drei Gruppen mit jeweils unterschiedlichen Metriken. In der Abbildung sind die verschiedenen zu bewertenden Metriken als graue Ovale innerhalb der farbigen Kästen der verschiedenen Gruppen abgebildet. Die Basisgruppe, in Abbildung 2.4 lila hinterlegt, fasst alle Metriken einer Sicherheitslücke zusammen, die ihr innewohnen und sich im Laufe der Zeit nicht ändern. Die temporale Gruppe, in Abbildung 2.4 gelb hinterlegt, fasst hingegen alle Metriken zusammen, die sich bei der Alterung der jeweiligen Sicherheitslücke ändern. Die Umgebungsgruppe, in Abbildung 2.4 hellblau hinterlegt, beschäftigt sich mit allen Metriken, die implementierungsabhängig und somit umgebungsspezifisch sind. [21], [22]

[6] `https://www.first.org/`

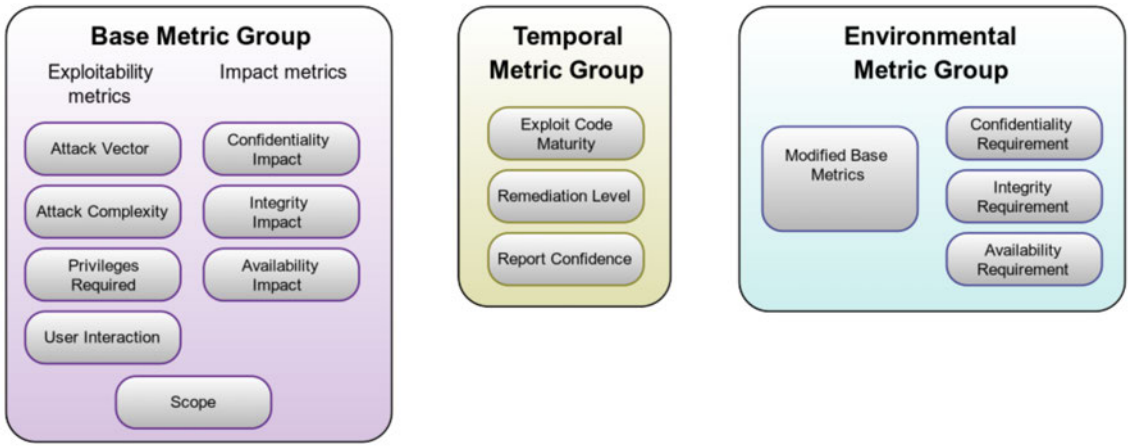


Abbildung 2.4: Unterteilung der Metriken in die drei CVSS-Gruppen [24]

Die Bewertung der Metriken der Basisgruppe ergeben einen Score von 0 bis 10 [25]. Dabei werden Auswirkungen der Sicherheitslücke jeder Metrik separat bewertet. Der Score der Basisgruppe ergibt sich aus der Betrachtung der Bewertung aller Metriken der Gruppe. Das Ergebnis fließt in die Formel der temporalen Gruppe ein und wird im Hinblick auf die temporalen Metriken angepasst. Zur Bestimmung der finalen CVSS-Scores fließt das Ergebnis der temporalen Gruppe, welches wieder ein Score zwischen 0 und 10 ist, in die Formel der Umgebungsgruppe ein und wird dabei im Hinblick auf die umgebungsbezogenen Metriken angepasst. Der endgültige CVSS-Score zwischen 0 und 10 zeigt dann das Risiko, welches die jeweilige Sicherheitslücke für die jeweilige Umgebung darstellt. [21], [22]

Der Basis-Score kann in manchen Fällen schon ausreichend zur Beschreibung des Schweregrades einer Sicherheitslücke sein, weshalb die genannte Veränderung des Basis-Score durch Anpassung in der temporalen und Umgebungsgruppe optional und nicht notwendig ist [22].

Es ist anzumerken, dass CVSS ausschließlich den separaten Schweregrad von Sicherheitslücken bewertet [21], [22] und nicht zur Bewertung der gesamten Sicherheit einer Software, welche immer eine Vielzahl von Sicherheitslücken enthält [23], verwendet werden kann [15], [26].

Schweregrad einer Sicherheitslücke Das qualitative Rating von Sicherheitslücken verknüpft den CVSS-Score einer Sicherheitslücke mit ihrem Schweregrad. Der CVSS-Score ist mindestens 0 und maximal 10. Ab CVSS v3.0 werden fünf mögliche Schweregrade einer Sicherheitslücke aufbauend auf ihrem CVSS-Score definiert. Wie in Tabelle 2.1 dargestellt, werden Schweregrade von Sicherheitslücken in „kein“ Schweregrad, „niedriger“, „mittlerer“, „hoher“ und „kritischer“ Schweregrad unterschieden. Kein Schweregrad liegt vor, wenn ein CVSS-Score von 0.0 für die Sicherheitslücke berechnet wird. Bei CVSS-Score 0.1 bis 3.9 liegt ein leichter Schweregrad vor, bei 4.0 bis 6.9 ein mittlerer, bei 7.0 bis 8.9 ein hoher und bei 9.0 bis 10.0 ein kritischer Schweregrad. [24]

Tabelle 2.1: Einteilung der CVSS-Scores ab CVSS v3.0 [25]

Schweregrad	Basis Score-Bereich
keine	0.0
leicht	0.1-3.9
mittel	4.0-6.9
schwer	7.0-8.9
kritisch	9.0-10.0

2.3.2 Gefährdungsverlauf

Der Schweregrad einer Sicherheitslücke korreliert mit ihrem Lebenszyklus und wird dahingehend auch von der Art der Kenntnisnahme des Produktverantwortlichen über die Sicherheitslücke beeinflusst [8]. Abbildung 2.5 (A.1) stellt die Relation zwischen dem Verlauf des Lebens einer Sicherheitslücke und der Einschätzung ihrer potenziellen Gefährdung bei Ausnutzung für das System und seine Anwender, nachfolgend als Risiko bezeichnet, dar. Die Abbildung orientiert sich an der Darstellung des BSI [8] sowie der Darstellung von Kapur, Yadavali und Shrivastava [4]. Die Einteilung in „keine“, „leicht“, „mittel“, „hoch“ und „kritisch“ basiert auf der Schweregrad-Einteilung einer Sicherheitslücke nach CVSS [8]. Die blaue Linie steht für den Gefährdungsverlauf einer Sicherheitslücke in dem allgemeinen Fall, dass Informationen über eine Sicherheitslücke der Öffentlichkeit vor dem Produktverantwortlichem bekannt gemacht werden. Die rosa Linie steht dahingegen für einen *Coordinated Disclosure*-Fall.

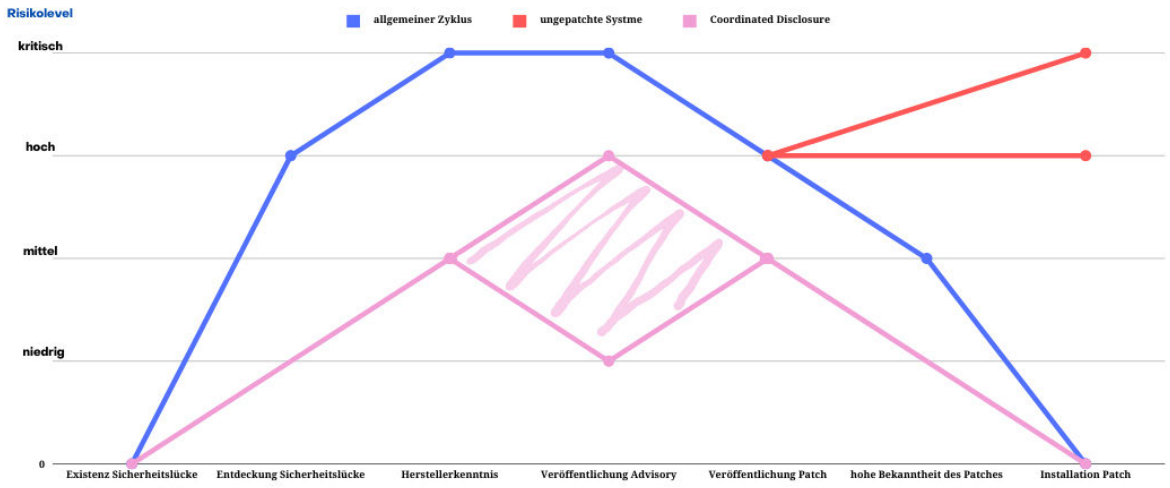


Abbildung 2.5: Gefährdungsverlauf einer Sicherheitslücke, angelehnt an [4] und [8]

Sobald eine Sicherheitslücke entdeckt wird, steigt ihr potenzielles Risiko an. Falls es sich nicht um ein *Coordinated Disclosure* handelt und somit die Informationen über die Sicherheitslücke der Öffentlichkeit noch vor dem Produktverantwortlichem zur Verfügung stehen, kann, bis der Produktverantwortliche Kenntnis von der Sicherheitslücke erlangt, von einem Risiko ausgegangen werden, dass einem kritischem Schweregrad entspricht. Durch schnelles Erfassen der Situation durch den Produktverantwortlichen und eine angemessene Behandlung der Sicherheitslücke mit Advisorys, sinkt das von der Sicherheitslücke ausgehende Risiko im Allgemeinen schrittweise ab. Von einem Risiko unter

einem hohen Schweregrad kann allerdings erst durch die Verfügbarkeit eines Patches ausgegangen werden. Mit der Installation des Patches sollte die Gefährdung der Sicherheitslücke idealerweise behoben sein. Ohne Installation des Patches kann das Risiko, das von der Sicherheitslücke ausgeht, allerdings ungehindert weiter ansteigen. Der potentielle Gefährdungsverlauf ohne Installation des Patches wird ab dessen Veröffentlichung mit roten Linien dargestellt. [4], [8]

Bei *Coordinated Disclosure* hängt der Verlauf der Gefährdung durch eine Sicherheitslücke mit der öffentlichen Verfügbarkeit von Informationen zu der Sicherheitslücke zusammen. Je mehr Informationen der Öffentlichkeit bekannt sind, desto höher ist das Risiko bis zur Veröffentlichung eines Patches. Dabei liegt das Risiko meist zwischen „niedrig“ und „hoch“. In der Abbildung wird dies durch die möglichen unterschiedlichen Verläufe der rosa Linie dargestellt. Der gekennzeichnete rosa Bereich steht stellvertretend für diese Variabilität. [8]

Abbildung 2.5 dient lediglich als Orientierung für den potentiellen Gefährdungsverlauf einer Sicherheitslücke. Die jeweilige Gefährdung einer Sicherheitslücke ist immer vom Einzelfall abhängig und sollt nie pauschalisiert betrachtet werden. [8]

Auf dem 19. Deutschen BSI-Sicherheitskongress sprach Dr. Ralf Schneider, Alliance Group CIO, in der Podiumsdiskussion zum Thema „Digital sicher in eine nachhaltige Zukunft“ davon, dass heutzutage Patches immer effizienter entwickelt und schneller veröffentlicht werden, wodurch Sicherheitslücken zügig geschlossen werden können [27].

2.4 Aufspüren von IT-Sicherheitslücken

Um Sicherheitslücken in einem System zu vermeiden oder zu beseitigen [5], wird ihrer Detektion eine hohe Priorität zugeordnet [9]. Zum Aufspüren von Sicherheitslücken innerhalb von Software, werden Detektionsmethoden benötigt, welche das Erkennen und Lokalisieren unentdeckter Sicherheitslücken in einer Software ermöglichen [6], [9]. Die Detektion von Sicherheitslücken trägt dazu bei, dass Produktverantwortliche sowie Anwender von der Sicherheitslücke erfahren, um entsprechend handeln und vorbeugende Maßnahmen treffen zu können [5]. Zur Detektion von Sicherheitslücken gibt es eine große Anzahl unterschiedlicher Tools, Modelle und Techniken [6]. Diese Detektionsmethoden sind in die Kategorien „statisch“, „dynamisch“ und „hybrid“ unterteilt [6], [9]. Viele Techniken sind zudem automatisiert [14].

Statische Methoden zeichnen sich dadurch aus, dass die Software nicht ausgeführt werden muss, um potenzielle Sicherheitslücken detektieren zu können. Sie fokussieren sich auf die Analyse des Quellcodes sowie allen Kontrollabläufen, mit dem Ziel Informationen direkt aus dem Code schließen zu können. Durch diese Vorgehensweise wird die Ausführung von möglicherweise vorhandenen schädlichen Programmen verhindert. Statische Methoden gelten als sehr effektiv für die Detektion von Sicherheitslücken in der frühen Phase der Entwicklung einer Software. Sie sind schnell und wiederholbar. Da sie nur auf der Analyse des Quellcodes beruhen, gibt es im Allgemeinen eine große Anzahl

falsch-positiver Ergebnisse, welche schwer auszumachen und auszuschließen sind. Zu den statischen Sicherheitslücken-Detektions-Methoden zählen unter anderem die lexikalische Analyse, die Datenflussanalyse, das „Pattern Matching“ sowie die Modellprüfung. [5], [6], [9]

Im Gegensatz zu statischen Methoden wird bei dynamischen Methoden der Code der Software zum Zweck der Analyse und Detektion der Sicherheitslücken ausgeführt. Fuzzing-Tests, Fehlerinjektion und „Sanitization“ sind Vertreter von dynamischen Techniken. Durch die Analyse der Software bei ihrer Ausführung, gibt es nur wenige falsch-positive Fehlalarme. Allerdings sind dynamische Methoden in den meisten Fällen mit hohen Kosten, Problemen bei der Effizienz und geringer Abdeckung verbunden. [5], [6], [9]

Durch Verbindung von statischen und dynamischen Ansätzen, ermöglichen hybride Methoden einen Ausgleich der Nachteile beider Varianten [9]. Ein Beispiel für eine hybride Methode für die Detektion der Heartbleed Sicherheitslücke[7] ist FLINDER-SCA [28]. Bei diesem Tool werden im ersten Schritt statische Methoden angewandt, um potenzielle Sicherheitslücken aufzuspüren. Im letzten Schritt wird der dynamische Ansatz verwendet und das Programm ausgeführt, wobei die detektierten Sicherheitslücken nachgeprüft werden. [28]

Eine allgemeingültige Methode, mit der sich alle Sicherheitslücken von jeder Software detektieren lassen, gibt es allerdings nicht [6]. Um Sicherheitslücken aufzuspüren werden daher gegenwärtig häufig, anstelle einer einzelnen Detektionsmethode, eine Vielzahl verschiedener Techniken und Tools zur Detektion von Sicherheitslücken kombiniert genutzt [6], [9].

2.5 Verwaltung und Systematisierung von Sicherheitslücken

Um die Sicherheit von Software zu verbessern, werden Informationen über gefundene Sicherheitslücken gesammelt, gepflegt und verbreitet [29]. Diesem Akt der Verwaltung von Sicherheitslücken haben sich verschiedene internationale Organisationen verschrieben [9].

2.5.1 CVE

CVE[8] ist ein Industriestandard, welcher der einheitlichen Benennung von Sicherheitslücken und der systematischen Sammlung aller, über die jeweilige Sicherheitslücke verfügbaren, Informationen dient [8]. Durch die Kennung von CVEs, sogenannte CVE-IDs [30], wird eine standardisierte Identifizierung von Sicherheitslücken ermöglicht [13]. Wie in Abbildung 2.6 dargestellt, ist der aktuelle CVE-ID-Syntax am 01.01.2014 in Kraft getreten und wird folgendermaßen beschrieben:

„CVE“ + Jahr + Sequenznummer

[7] `https://www.cvedetails.com/cve-details.php?t=1&cve_id=CVE-2014-0160`
[8] derzeitige Website: `https://cve.mitre.org/` ; neue Website: `https://www.cve.org/`

Die Sequenznummer, welche für jedes neue Jahr wieder ab 1 startet, kann aus vier oder mehr Ziffern bestehen. Ein allgemeines Beispiel einer CVE mit vier-stelliger Sequenznummer könnte folgendermaßen aussehen: CVE-YYYY-NNNN. [30]

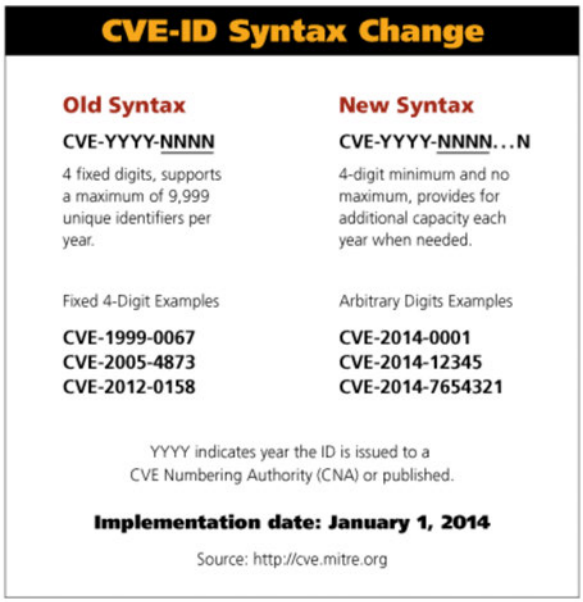


Abbildung 2.6: Aktueller Syntax der CVE-IDs [30]

Durch die Vereinheitlichung der Benennung von Sicherheitslücken durch den CVE-Standard kann sichergestellt werden, dass eine Sicherheitslücke nicht unter verschiedenen Bezeichnungen getrennt voneinander betrachtet wird [8]. Die Verwendung solch einer gemeinsamen Namenskonvention, wie die des CVE-Projektes, wird deshalb von dem NIAC empfohlen [12].

Verwaltung CVE kann kostenlos von jedem genutzt werden und enthält zu jeder aufgeführten Sicherheitslücke eine CVE-ID, eine Beschreibung und mindestens einen öffentlichen Verweis. Das CVE-Repository untersteht dem National Cybersecurity Federal Funded Research and Development Center (NCF) [29] und wird von der MITRE Corporation[9] verwaltet. Sponsoren sind das US Department of Homeland Security (DHS)[10] und die Cybersecurity and Infrastructure Security Agency (CISA)[11]. CVE unterliegt ständiger Aktualisierung und Weiterentwicklung. Das Hinzufügen neuer CVE-Einträge beruht dabei auf der Mitarbeit der Gemeinschaft. Organisationen, die sich am CVE-Programm beteiligen wollen, können sich kostenlos als CVE Numbering Authority (CNA) registrieren lassen. CNAs vergeben CVE-IDs für neu entdeckte Sicherheitslücken, welche in ihren Verantwortungsbereich bezüglich Identifizierung und Veröffentlichung von Sicherheitslücken fallen [31]. Zum Hinzufügen eines Eintrages für eine Sicherheitslücke erstellt ein CNA eine leere CVE-ID. Diese wird anschließend mit den bekannten Informationen zu der Sicherheitslücke gefüllt. Diese Beschreibung der Sicherheitslücke wird nicht als final angesehen und kann, wenn nötig, ergänzt werden. Sollte eine CVE samt ID entfernt werden, wird sie als „Rejected“ (dt. abgelehnt) weiterhin im CVE- Repository aufgeführt. [32]

[9] https://www.mitre.org/
[10] https://www.dhs.gov/
[11] https://www.cisa.gov/

Insgesamt sind CVEs seit dem Jahr 1999 erfasst [29] und können unter anderem auf der cvedetails-Website[12] abgerufen werden.

2.5.2 NVD

Die National Vulnerability Database (NVD) ist eine der einflussreichsten und am häufigsten verwendeten [29] Sicherheitslücken-Datenbank [22]. Sicherheitslücken-Datenbanken sammeln, verwalten und verbreiten Informationen über enthaltene Sicherheitslücken [29]. NVD ist mit CVE synchronisiert und enthält einen umfangreichen Katalog „stabiler“ [32, S. 220] CVE-Sicherheitslücken und deren spezifische Informationen. Benutzer können Sicherheitslücken in der NVD über eine abgestufte Suchmaschine finden [22].

NVD ist die „offizielle Datenbank der US-Regierung zum Management von Sicherheitslücken“ [29, S. 7], wurde vom NIST entwickelt und wird regelmäßig basierend auf CVE aktualisiert. Dabei wird jede neu hinzugefügte Sicherheitslücke analysiert und anschließend überprüft [31]. Neben manchen Sicherheitslücken, die bei diesem Prozess von NIST ausgeschlossen werden, werden den neuen Sicherheitslücken anhand der Analyseergebnisse detaillierte Informationen hinzugefügt. [32]

Die Analysten der NIST fügen den CVEs folgende Metadaten hinzu: die Bewertung ihres Schweregrades mittels CVSS v3.1 und CVSS v2.0 [25], Reference Tags und Konfigurationen. Weitergehend wird jeder CVE als Klassifizierung [32] eine Common Weakness Enumeration (CWE) zugeordnet. [31]

CVSS wird als Standard für die Schweregrad-Bewertung von vielen staatlichen US-Behörden empfohlen [33]. Die Schweregrad-Bewertung der Sicherheitslücken mittels CVSS-Scores in der NVD wird von NVD Analysten durchgeführt [22], [23], [31] und ist somit von der US-Regierung validiert [22].

In Abbildung 2.7 ist der vereinfachte Ablauf der Einbindung eines CVSS-Scores in die NVD dargestellt. Nach Auffinden einer Sicherheitslücke wird für diese im Allgemeinen eine CVE angefordert. Die MITRE Corporation regelt diesen Austausch. Die Sicherheitslücken, die dabei zum Archivieren akzeptiert und aufgenommen werden, werden nach Überprüfung [32] in der NVD veröffentlicht. Parallel zur Koordinierungsarbeit bezüglich der CVEs sowie deren Aufnahme, werden die Sicherheitslücken von einem Analysten der NVD mittels CVSS v2.0 und 3.1 [25] auf Basis ihres Schweregrades bewertet. Nach Abschluss der Bewertung werden die CVE-Sicherheitslückeninformationen in der NVD aktualisiert und um den CVSS-Score ergänzt. Durch die in der Regel später als die CVE-Daten veröffentlichten Schweregrad-Informationen, entsteht dabei eine uneinheitliche zeitliche Differenz zwischen Veröffentlichung der Sicherheitslücken und Aufnahme der Schweregrad-Bewertung in die Datenbank. [33]

[12] https://www.cvedetails.com/

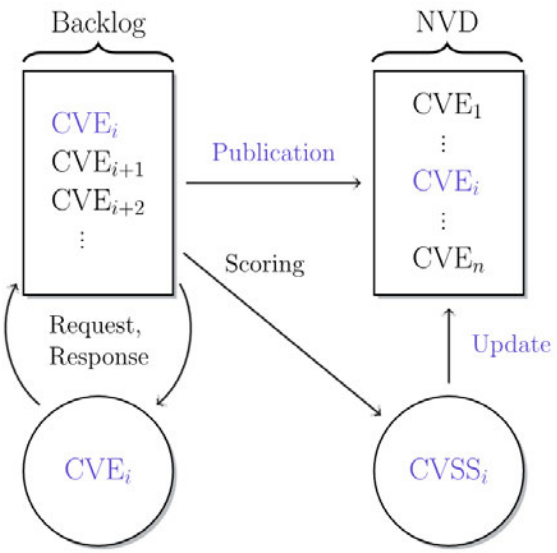


Abbildung 2.7: vereinfachter CVE- und CVSS-Prozess der NVD [33]

Weitere Vertreter von Sicherheitslücken-Datenbanken sind unter anderem die Open Source Vulnerability Database (OSVDB) und IBM X-Force Exchange[13]. Zudem verfügen einige Länder auch über nationale Sicherheitslücken-Datenbanken. Beispiele hierfür sind Japan Vulnerability Notes (JVN)[14] in Japan und Chinese National Vulnerability Database (CNNVD)[15] in China. [29]

2.6 Entwicklung sicherer Software

Durch den immer stärker werdenen Einfluss, den Software in der Gesellschaft hat, ist ihre Sicherheit und Vertrauenswürdigkeit, wie in Abschnitt 2.1 und Abschnitt 2.2 beschrieben, eine wichtige Thematik [34]. Das NIST hat diesbezüglich 2022 das Secure Software Development Framework (SSDF) [35] als Framework für eine sichere Entwicklung von Software veröffentlicht. Die Kerninhalte werden nachfolgend zusammengefasst. Das Framework stellt Praktiken für eine sichere Softwareentwicklung vor und soll Softwareentwickler unterstützen, die Anzahl von Sicherheitslücken in ihrer Software zu reduzieren. Es soll helfen, die Ursachen der Sicherheitslücken beseitigen und die potenziellen Schäden durch ihre Ausnutzung verringern zu können. Der Fokus liegt dabei auf den Ergebnissen der Praktiken und nicht ihrer Umsetzung. Es liegt dabei an den Unternehmen zu bestimmen, welche Praktiken für sie angemessen und effektiv sind. Bei der Auswahl sollten Risiken, Kosten, Durchführbarkeit und Anwendbarkeit beachtet werden.

Es wurden vier Empfehlungen für eine sichere Softwareentwicklung benannt [35]:

- Es soll sichergestellt werden, dass Mitarbeiter, Prozesse und verwendete Technologien auf eine sichere Entwicklung von Software vorbereitet sind.
- Software-Komponenten sollen vor Manipulation und unbefugtem Zugriff geschützt werden.

[13] https://www.ibm.com/de-de/products/xforce-exchange
[14] https://jvn.jp/
[15] https://www.cnnvd.org.cn/

- Es soll gut gesicherte Software mit einer minimalen Anzahl von Sicherheitslücken produziert werden.
- Verbliebene Sicherheitslücken sollen identifiziert, angemessen behandelt und beseitigt werden.

Die einzelnen Praktiken sind, an die Empfehlungen angelehnt, in vier Gruppen unterteilt. Es gibt Praktiken um die Organisation auf sichere Software Entwicklung vorzubereiten, Praktiken um die Software zu schützen, Praktiken um gut-gesicherte Software zu produzieren und Praktiken um auf Sicherheitslücken angemessen reagieren zu können. Alle Praktiken finden sich im Anhang des SSDF.

An diese Praktiken anschließend, hat die Universität Bonn *Best Praktices* (dt. beste Praktiken) für eine sichere Softwareentwicklung zusammengefasst [34]. Dabei sind sie zu dem Schluss gekommen, dass es bereits eine große Auswahl guter Best Practices gibt und dabei automatisierte Werkzeuge eine große Rolle bei der Steigerung der Software-Sicherheit spielen.

Weitere Empfehlungen für die Entwicklung sicherer Software wurden beispielsweise 2022 von dem Consortium for Information & Software Quality (CISQ) in [14] veröffentlicht.

2.6.1 Sicherheitsvergleich: Open Source Software vs Proprietäre Software

Im Zuge der Betrachtung von sicherer Softwareentwicklung, hat sich die Universität Bonn im Auftrag von der Open Source Business Alliance (OSBA) damit beschäftigt, die Sicherheit von Open Source Software der Sicherheit proprietärer Software gegenüberzustellen. Die Ergebnisse der im Juni 2023 veröffentlichten „Studie zum Vergleich der Sicherheit von Open-Source-Software und Proprietärer Software“ [34] werden nachfolgend zusammengefasst.

Unterschieden werden die Software-Formen bezüglich ihres Entwicklungs- und Vertreibungsansatzes. Ist eine Software transparent quelloffen entwickelt und von jedem uneingeschränkt und ohne Zahlungsbarriere nutzbar, so handelt es sich um Open Source Software. Die Software kann dabei je nach Wahl der Lizenz von allen Nutzenden untersucht oder auch modifiziert werden, wodurch jeder Nutzende zusätzlich die Möglichkeit hat, Verbesserungen einzureichen. Wird der Quelltext im Gegensatz zum Ansatz der Open Source Software geheimgehalten und die Software ist nur unter bestimmten Voraussetzungen, beispielsweise dem Besitz einer Nutzungslizenz, einsetzbar, so handelt es sich um proprietäre Software.

Open Source Software ist in Bezug auf Einsatz in Firmen klar auf dem Vormarsch und etabliert sich zunehmendend auf dem Markt. Laut Studien setzen 71% der befragten deutschen Unternehmen Open Source Software ein [36], in Großbritannien 89% [37] und in der Schweiz 97% [38]. 77% der in einem 2022 erschienenen Bericht der CISQ [14] befragten Unternehmen, gaben an, dass der Einsatz von Open Source-Software steigt.

Im Gegensatz zu proprietärer Software bietet Open Source Software in ihrer Entwicklung Transparenz und Nachverfolgbarkeit. Zudem stehen sie im Gegensatz zu den meisten proprietären Software-Systemen kostenlos zur Verfügung. Die eingeschränkte Nutzung proprietärer Software bringt hingegen gegenüber des frei verfügbaren Open Source Quellcodes den Vorteil, dass nur autorisierte Personen Änderungen am Quellcode vornehmen können. Da der Quellcode nur autorisierten Personen zur Verfügung steht, liegt es allerdings auch an ihnen, die Sicherheit der Software zu gewährleisten und Patches zu veröffentlichen. Dieser Prozess ist nicht transparent. Bei Open Source Software kann indes jeder Nutzer, wie oben beschrieben, Verbesserungen mitgestalten, wodurch regelmäßige Software Updates und Sicherheitspatches ermöglicht werden, die aus der Zusammenarbeit der Community und des Entwicklerteams entstehen. Häufig fehlt es Open Source Software allerdings an Support, da dieser nicht wie bei den meisten Vertretern proprietärer Software vertraglich geregelt und umfangreich vorhanden ist. Somit bietet proprietäre Software rechtliche Sicherheit und Support, wohingegen Open Source Software Transparenz, Unabhängigkeit und Einsparung von Kosten verspricht.

Open Source Software und proprietäre Software vertreten folglich zwei verschiedene Philosophien. Trotz dessen ist die Sicherheit der Software für beide Entwicklungsansätze gleichsam eine zentrale Aufgabe. Die Softwaresicherheit kann dabei nicht vom Entwicklungsansatz der Software abhängig gemacht werden. Die Qualität der Sicherheit wird von der treibenden Projektorganisation und den Entwicklungspraktiken hinter der Software maßgeblich mitbestimmt.

Die eindeutige Grenze zwischen den beiden Software Formen verschwimmt gegenwärtig immer weiter. In proprietärer Software werden zu großen Teilen Open Source Komponenten verbaut. Bei Software Komponenten handelt es sich um vorgeschriebenen Code, der zu Software hinzugefügt werden kann [39]. Ein 2023 veröffentlichter Bericht zeigt, dass gegenwärtig (2022) 96% der untersuchten proprietären Codebasen mindestens eine Open Source Komponente enthalten [39]. Software-Systeme mit weniger als einer Millionen LoC enthalten dabei durchschnittlich 200 bis 300 Drittanbieter-Komponenten [14]. Die Open Source Komponenten bilden dabei typischerweise die Grundlage proprietärer Software, auf welche dann marktdifferenzierende Merkmale hinzugefügt werden.

Die Verwebung von proprietärer Software mit Open Source, ermöglicht die Schlussfolgerung, dass proprietäre Software nicht sicherer als die verbauten Open Source Komponenten und somit Open Source Software sein kann. Auch nach dem BSI ist Open Source Software mindestens so sicher wie proprietäre Software [40], wodurch beide Schlussfolgerungen den gleichen Kern widerspiegeln. Open Source Software ist gegenwärtig den Ausführungen der Quellen folgend, mindestens so sicher wie proprietäre Software. Aber heißt das, dass Open Source Software im Allgemeinen sicherer ist?

Einige der in Abschnitt 2.6 erläuterten Kriterien der sicheren Software Entwicklung sind für proprietäre Software auf Grund des eingeschränkten Zugriffs nicht überprüfbar. Während die Qualität von Open Source Software wie ihre Implementierung weitgehend beurteilt und geprüft werden kann, ist dies bei proprietärer Software somit nur eingeschränkt möglich. Beide Software-Entwicklungsansätze können allerdings auch durch Code-Reviews nicht vor Manipulation des Quellcodes geschützt werden. Bei Open Source

Software kann die Manipulation aber von allen Anwendern entdeckt und gelöst werden. Dies ist bei proprietärer Software nicht möglich, wodurch die Sicherheit der Software von außen nicht sichergestellt werden kann.

Die Studie der Universität Bonn kommt zu dem Schluss, dass die Orientierung an Praktiken sicherer Software Entwicklung für sichere Software entscheidend ist. Ob und wie dies geschieht, lässt sich in Open Source Software aufgrund ihrer Transparenz besser nachweisen, als in proprietärer Software. Nichtsdestotrotz weisen beide Software-Entwicklungsansätze viele Gemeinsamkeiten bezüglich der Produktion sicherer Software auf und die klaren Grenzen zwischen ihnen verschwimmen immer weiter. In Bezug auf Sicherheit hat somit keiner der Ansätze einen unanfechtbaren Vorteil, wodurch sie unter Betrachtung des Sicherheitsaspektes als gleichwertig betrachtet werden können. Kein Ansatz kann klar als der sicherere Ansatz bestimmt werden. Die Sicherheit einer Software unabhängig vom Entwicklungsansatz ist damit verbunden, wie viel unternommen wird, um die Software sicher zu gestalten. Für Sicherheitsbetrachtungen sollten somit für Open Source Software und proprietäre Software die gleichen Maßstäbe gelten.

2.7 Untersuchungsgrundlagen

Um Sicherheitslücken allumfassend betrachten zu können, können sie und ihr Auftreten statistisch analysiert werden. Dabei ist das Mining von Sicherheitslücken-Datenbanken ein oft genutzter Ansatz [29]. Die folgenden Untersuchungsgrundlagen bilden die Basis für das sich anschließende Kapitel zu den Methoden.

2.7.1 Extraktion von Informationen

Das World Wide Web (Web), vergrößert sich unablässig [41]. Gegenwärtig ist das Web die „größte öffentlich zugängliche Datenquelle der Welt“ [42, S. VII]. Es bildet durch den „explosiven Anstieg“ [43, S. 98] zur Verfügung stehender Daten, eine nie dagewesene Möglichkeit zur Erkenntnisgewinnung und bildet dabei eine beispiellose Grundlage für Data Mining [43]. Die Ergebnisse des Minings sind dabei jedoch stark von der Qualität der verfügbaren Informationen abhängig [29].

Data Mining, welches auf die Extraktion von Wissen aus dem Web abzielt, wird Web Mining genannt [43]. Die Extraktion von Webdaten ermöglicht es, dieses Wissen effizient und mit begrenztem menschlichen Aufwand aus Webquellen zu extrahieren und zu sammeln [44]. Web Mining beschreibt dahingehend das Analysieren von Webseiten in Bezug auf statistische Eigenschaften sowie die Analyse der Daten der Webseite [41]. Web Mining kann in drei Kategorien eingeteilt werden. Um die Verknüpfungsstrukturen von Webseiten darzustellen, wird Web Structure Mining genutzt. Um das Benutzerverhalten auf Webseiten darzustellen, kann Web Usage Mining genutzt werden. Zur Extrahierung von Informationen aus den Webseiten wird Web Content Mining eingesetzt. [42], [43], [45]

Um im Web aufgeführte Informationen untersuchen zu können, müssen ihre Informationen vor der Verarbeitung und abschließenden Untersuchung aus dem Web extrahiert werden. Als Basis für die Untersuchung von Informationen, die im Web über Sicherheitslücken zur Verfügung stehen, wird sich im Weiteren somit auf das Web Content Mining fokussiert.

Abbildung 2.8 zeigt den typischen Verlauf von Web Content Mining. Das Sammeln von Daten bildet die Grundlage für den Prozess.

Abbildung 2.8: Ablauf von Web Content Mining-Prozessen, angelehnt an [43]

Crawling Zum Abrufen und Sammeln von Daten nutzen Web Content Mining-Anwendungen unter anderem *Web Crawler*. *Web Crawler*, auch *Spider* genannt [45], sind Programme, die Webseiten automatisiert durchsuchen, indem sie Hyperlinks dieser Webseiten ausfindig machen und ihnen folgen. [43]

Sie ermöglichen das massenhafte Herunterladen von für den Nutzer interessanten Webseiten. Durch diese Funktionalität werden *Crawler* neben Web-Data-Mining unter anderem für Web-Suchmaschinen und Web-Archivierung eingesetzt. [41]

Miteinander verknüpfte Hypertext-Dokumente bilden die Grundstruktur des Webs [45]. Der Ausgangspunkt für *Web Crawler* sind typischerweise URLs, die als Startpunkt festgelegt wurden. Ein *Web Crawler* lädt alle von der Uniform Resource Locator (URL) adressierten Webseiten herunter und extrahiert dabei Hyperlinks. In einem iterativen Prozess geht der *Crawler* alle auf diese Weise gefundenen URLs auf die gleiche Weise durch. [41]

Mit Abschluss dieses Prozesses werden die gecrawlten Daten lokal gespeichert [45], wodurch eine Weiterverarbeitung ermöglicht wird [43].

Laut Olston et al. [41] liegt die Daseinsberechtigung von *Web Crawlern* darin, dass das Web keinen zentral gemanagten Speicherort für alle enthaltenen Information darstellt. Es gibt im Gegensatz dazu hunderte Millionen unabhängige Web-Anbieter, die sich an bestimmte Protokolle und Datenformate halten sollen, aber doch unterschiedliche Dienste anbieten, wodurch eine Anwendung zum Abgreifen dieser verteilten Daten notwendig ist. So wurde schon 1993, demselben Jahr, indem der erste grafikfähige Webbrowser Mosaic[16] auf den Markt kam, der erste *Web Crawler* implementiert. [41]

Scraping Um gezielt bestimmte Daten zu extrahieren und somit genauere Content-Mining-Ergebnisse zu garantieren, wird *Web Scraping* genutzt. Bei *Scraping* werden die Zieldaten so extrahiert, dass strukturierte Daten erzeugt werden, die weiter verarbeitbar sind. [43] *Crawling* und *Scraping* können als separate Aufgaben hintereinander laufen

[16] `http://mosaic.mcom.com/`

oder gleichzeitig ausgeführt werden. Bei gleichzeitiger Ausführung werden auf den Seiten, die der *Crawler* holt, direkt die Daten im Sinne eines *Scrapers* extrahiert. Ein *Web Crawler*, der diese Funktionsweise umsetzt kann somit auch als *Web Scraper* bezeichnet werden. [43]

2.7.2 R als Untersuchungsumgebung

R R[17] [46] ist eine frei verfügbare und „hoch effektive Softwareumgebung für statistische Analysen[,] [. . .] Datenverarbeitung“ [43, S. 99] und Visualisierungen. Neben einer Vielzahl an verfügbaren graphischen und statistischen Funktionen, kann R zusätzlich mit verschiedenen Pakten erweitert werden, um zusätzliche Funktionen zu erhalten. [47]

R ist open-source und ermöglicht Nutzern, im Hinblick auf die Analyse von Daten, Code zu schreiben und auszuführen. „R“ bezieht sich dabei sowohl auf die Programmiersprache R als auch auf die Software R, die das Ausführen des Codes ermöglicht. Beispielsweise nutzt die integrierte Entwicklungsumgebung (IDE) Rstudio die Programmiersprache R. [48]

Rcrawler Rcrawler ist ein, in R verfügbares, Paket für domänenbasiertes Multi-*Threaded-Crawling* und *Scraping* von Webseiten. Mit einer festgelegten URL als Startpunkt der Untersuchung, crawlt und parst Rcrawler automatisiert alle URLs der Domäne. Dabei können mit Hilfe von XPath-Mustern gezielt Informationen von den Webseiten extrahiert werden. *Crawling* und *Scraping* kann dabei getrennt voneinander oder gleichzeitig ausgeführt werden. Die Tiefe des *Crawlings* von der Start-URL aus kann dabei vom Nutzer vorgegeben werden, sodass nur die notwendigen Seiten gecrawlt werden. Die Ergebnisse des Prozesses werden in strukturierten und zur Weiterverarbeitung direkt nutzbaren Datenstrukturen zurückgegeben. Rcrawler hebt sich von anderen in R verfügbaren Pakten für Datensammlung hervor, da crawlen, parsen und scrapen von Webseiten möglich gemacht wird. Andere Pakete, wie scrapeR und Rvest, bieten dahingegen nur eine bis zwei dieser Funktionen an. Zur Bearbeitung von Web-Content-Mining-Problemen ist Rcrawler somit eine leicht anwendbare Variante. [43]

Abbildung 2.9 zeigt die Funktionsweise von Rcrawler. Rcrawler startet mit der vorgegebenen URL, legt ihre Daten ab und extrahiert dabei vorhandene URLs. Die URLs, die in den, vom Nutzer vorgegebenen, Filter passen, werden in einer URL-Liste abgelegt. Der *Crawling*-Prozess ist beendet, wenn alle URLs der Liste abgearbeitet sind.

Durch die Erkennung von Duplikaten schon gespeicherter Webseiten können die Verarbeitungskosten, der Speicherbedarf sowie die Netzwerkbandbreite reduziert werden [43].

[17] https://www.r-project.org/

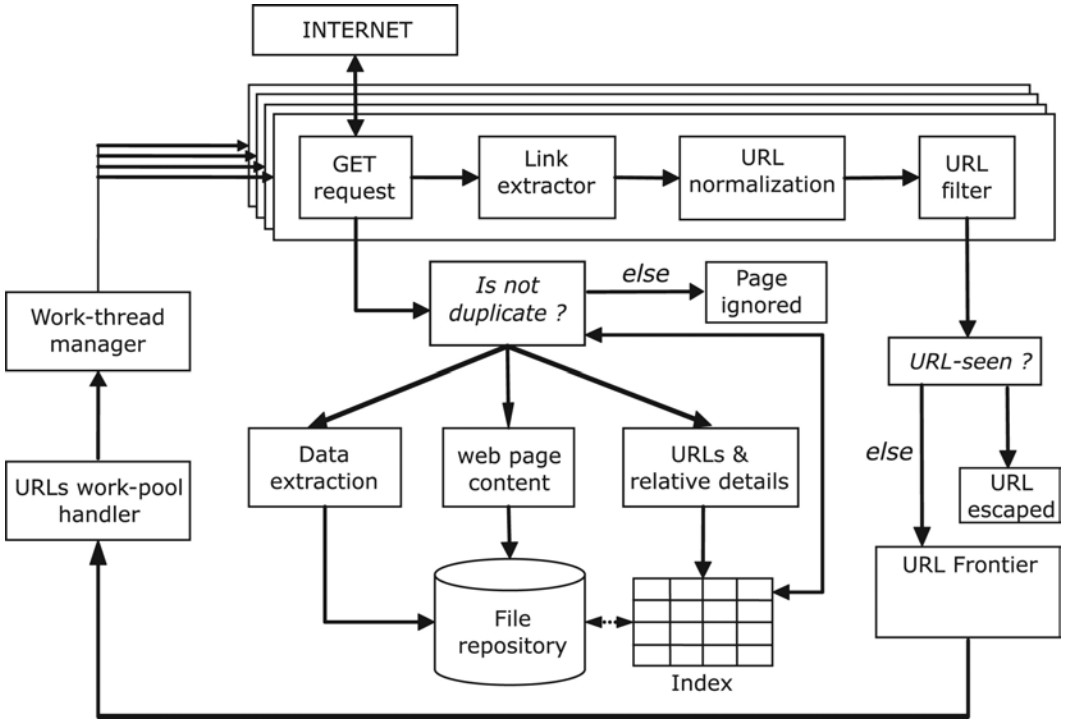


Abbildung 2.9: schematische Darstellung der Funktionsweise des Rcrawlers [43]

3 Methoden

Um einen Überblick über die Entwicklung von Sicherheitslücken sowie potenziell beeinflussenden Faktoren zu erhalten, werden dazugehörige Daten analysiert, ausgewertet und verglichen. Dazu werden Rstudio, Excel und Bash-Befehle genutzt.

Programmierumgebung R Zur Analyse, Verarbeitung und Veranschaulichung der Daten werden in Rstudio folgende Pakete verwendet:

- Rcrawler[18]
- tidyverse[19]

Rcrawler wird zum Extrahieren von Daten aus dem Web verwendet. Tidyverse wird zum Verarbeiten der Datensätze genutzt. Dabei werden unter anderem mit Hilfe des tidyverse-Pakets ggplot2 Diagramme über die bereinigten Datensätze erstellt.

3.1 CVE-Entwicklung

Extraktion Die Analyse des Auftretens von Sicherheitslücken wird anhand von CVE-Daten durchgeführt. Die Daten werden der Website `https://www.cvedetails.com/` entnommen. Es handelt sich dabei um eine Sicherheitslücken-Datenbank, welche nach CVE klassifizierte Sicherheitslücken enthält[20].

Zur Extraktion der Daten der knapp 200.000 Sicherheitslücken wurde ein Webcrawler geschrieben. Der Crawler greift auf alle Seiten der oben genannten Webseite zu, welche CVE-Listen der Jahre 1999 bis 2023 Jahre enthalten. Diese Seiten sucht er nach Hyperlinks ab, welche den einzelnen CVEs zu geordnet sind und somit genaue Informationen der jeweiligen CVEs beinhalten. Von den gecrawlten Seiten werden anschließend Informationen bezüglich des Namens, des CVSS-Scores und des Meldedatums der jeweiligen CVE gescrapet und in einen Datensatz für das dazugehörige Jahr gefügt. Beim Crawlen der Webseiten ist zu beachten, dass den URLs der verschiedenen Jahre jeweils unterschiedliche SHA-Hashwerte angehangen sind und dies somit auch bei der Gestaltung des Crawlers Beachtung finden muss. Ansonsten können nur die ersten 50 CVE-Daten extrahiert werden.

```
CVE1999 <- map dfr(URL list 1999, ~{

 Rcrawler(Website = .,
        crawlUrlfilter = "/cve/[^'java']+$",
        dataUrlfilter = "/cve/[^'java']+$",
        crawlZoneXPath = "//div[@id='searchresults']",
        ExtractXpathPat = c("//*[contains(concat( ' ', @class, ' ' ), concat( '
            ', 'col-md-12', ' ' ))]//a", #Name CVE
```

[18] https://www.rdocumentation.org/packages/Rcrawler/versions/0.1.9-1
[19] https://tidyverse.tidyverse.org/
[20] Nach der letzten Nutzung der Datenbank am 07.07.2023 wurde diese grundlegend überarbeitet. Die nachfolgenden Pfade der Crawler (crawlZoneXPath, ExtractXpathPat) wurden dahingehend angepasst.

```
                    "//*[contains(concat( ' ', @class, ' ' ), concat( ' ',
                        'ps-2', ' ' )) and (((count(preceding-sibling::*) + 1)
                        = 1) and parent::*)] | //*[contains(concat( ' ',
                        @class, ' ' ), concat( ' ', 'cvssbox', ' ' ))]", #CVSS
                    "//*[contains(concat( ' ', @class, ' ' ), concat( ' ',
                        'd-inline-block', ' ' )) and
                        (((count(preceding-sibling::*) + 1) = 1) and
                        parent::*)]"), # Datum
        PatternsNames = c("CVE", "CVSS"),
        no cores = 4,
        no conn = 4,
        MaxDepth = 1,
        RequestsDelay = 0.1,
        saveOnDisk = FALSE)

 CVElist 1999 <- do.call("rbind", DATA) %>% data.frame()
})
```

Verarbeitung Nach ihrer Extraktion werden die CVE-Daten verarbeitet. Dabei wird die CVE-Gesamtanzahl für jedes Jahr erfasst. Zusätzlich werden für jedes Jahr das Vorkommen von CVEs nach ihrem CVSS-Scores gezählt. Dazu wird sich an der Schweregrad-Einteilung von CVSS (v2) orientiert. Es gibt somit drei Kategorien. CVEs mit CVSS-Score von mindestens fünf, CVEs mit CVSS-Score von mindestens sieben und CVEs mit CVSS-Score von mindestens neun.

```
## CVSS 5+ ##
anzahl <- 0

for (i in 1999:2022) {
 Di <- filter(CVE, Jahr==i) %>% data.frame()
 anzahl <- length(which(Di$CVSS>=5))
 AnzahlCVE 5 <- append(AnzahlCVE 5, anzahl)
}
```

Es wird zusätzlich das Vorkommen der CVE inklusive der Kategorien kumuliert erfasst.

```
## CVSS 5+ kumulativ ##
anzahl c <- 0

for (i in 1999:2022) {
 Di <- filter(CVE, Jahr==i) %>% data.frame()
 anzahl c <- length(which(Di$CVSS>=5)) + anzahl c
 AnzahlCVE 5 c <- append(AnzahlCVE 5 c, anzahl c)
}
```

Die Ergebnisse beider Varianten werden als zwei eigenständige Datensätze abgelegt.

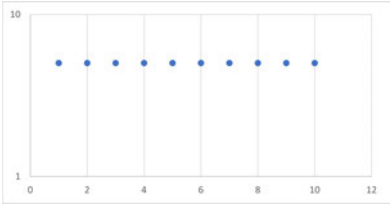
Visualisierung Die somit neu angelegten Datensätze über die CVEs der verschiedenen Jahre, bilden die Grundlage für die Analyse der Anzahl und Entwicklung der CVEs der vergangenen Jahre. Es werden CVEs mit Veröffentlichungsdatum von 1999 bis 2022 betrachtet. Das aktuell laufende Jahr 2023 wird außer Betrachtung gelassen.

Zur Veranschaulichung der Daten werden mit ggplot2 zwei Diagramme erstellt, welche die Anzahl der CVE pro Jahr bzw. kumuliert darstellen. Dazu wird die Anzahl der CVEs mit einem CVSS-Score mindestens fünf, mindestens sieben und mindestens neun pro Jahr bzw. über die Jahre kumuliert visualisiert. Für jede Kategorie wird mittels der Funktion *geom_point* ein Punktdiagramm erstellt, über welches anschließend zur Veranschaulichung des Trends eine Trendlinie mittels der Funktion *geom_smooth* gelegt wird.

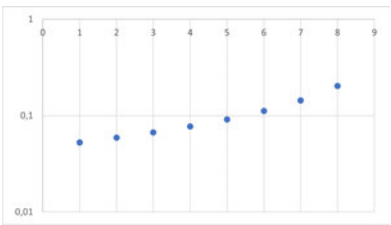
```
ggplot(alle, aes(x=Jahr)) +
 geom point(aes(y=allCVE, color="all CVE"), alpha=0.2) +
     geom smooth(aes(y=allCVE), color = 'blue', se=F) +
 geom point(aes(y=CVSS5, color="CVSS 5+"), alpha=0.2)+geom smooth(aes(y=CVSS5),
     color='green', se=F) +
 geom point(aes(y=CVSS7, color="CVSS 7+"), alpha=0.2)+geom smooth(aes(y=CVSS7),
     color='orange', se=F) +
 geom point(aes(y=CVSS9, color="CVSS 9+"), alpha=0.2)+geom smooth(aes(y=CVSS9),
     color='red', se=F) +
 labs(x="Jahr", y="Anzahl CVEs") +
 scale color manual("",
                breaks = c("all CVE", "CVSS 5+", "CVSS 7+", "CVSS 9+"),
                values = c('blue', 'green', 'orange', 'red'))
```

Analyse Um herauszufinden, welchen Trend die CVE-Entwicklung (gesamt) verfolgt, werden Log-Plots über die Differenzen der Auftreten von CVEs für jedes Jahr erstellt. Es handelt sich somit um Log-Plots der ersten Ableitung der Funktion, welche die Entwicklung der CVE-Daten von 1999 bis 2022 beschreibt. Die Jahre sind die x-Werte. Die Logarithmen der Differenzen bilden die y-Werte. Die Plots werden mit Excel erstellt. Bilden die Punkte im Plot eine Gerade mit positivem Anstieg, handelt es sich um eine e-Funktion. Dies kann als technischer Indikator für einen exponentiellen Trend gewertet werden. Bilden die Punkte im Plot eine Gerade ohne Anstieg, kann dies als technischer Indikator für einen linearen Trend gewertet werden.

Um diese Abhängigkeit zu Veranschaulichen, wurden Log-Plots von Ableitungen einer linearen (Abb. 3.1a) und einer exponentiellen Funktion (Abb. 3.1b) erstellt[21].



(a) Log-Plot Ableitung lineare Funktion



(b) Log-Plot Ableitung exponentielle Funktion

Abbildung 3.1: Vergleich der Log-Plots von Ableitungen

[21] für Datengrundlage siehe Anhang B.1

Zusätzlich werden zwei Plots in R erstellt, welche eine lineare Funktion bzw. eine exponentielle Verteilung neben der CVE-Entwicklung enthalten, um einen optischen Vergleich der beiden Varianten mit der CVE-Entwicklung zu ermöglichen.

3.2 Vergleich von CVE und LoC

Um die Entwicklung von Sicherheitslücken in Bezug auf die Entwicklung von Code-Größe zu betrachten, werden vier Software-Systeme analysiert. Bei den Software-Systemen handelt es sich um WordPress[22], QEMU[23], FFmpeg[24] und Linux Kernel[25]. Der Code von mehreren Versionen der Systeme ist offen zugänglich auf GitHub verfügbar, was eine Untersuchung des Quellcodes der Software-Systeme ermöglicht. Zur Analyse der Entwicklung der Software-Systeme werden CVEs betrachtet, die den jeweiligen Software-Systemen zugeordnet sind. Um die Entwicklung der Code-Größe zu untersuchen, werden die auf GitHub verfügbaren Versionen bezüglich ihrer LoC ausgewertet. CVE- und LoC-Entwicklung der Software-Systeme werden einander vergleichend gegenübergestellt.

Nachfolgend wird das Vorgehen am Beispiel von WordPress beschrieben.

CVE-Extraktion Im ersten Schritt wird `https://www.cvedetails.com/`, gleichsam der Extraktion aller CVE-Daten, gecrawlt. Dabei werden alle CVEs mit dazugehörigem CVSS-Score und Veröffentlichungsdatum extrahiert, die dem „vendor“ (dt. Anbieter) WordPress zugeordnet sind. Diese Zuordnung erfolgt auf der Webseite über die Vendor-ID 2337[26].

```
CVE_wp <- map_dfr(URL_list_wp, ~{

 Rcrawler(Website = .,
        crawlUrlfilter = "/cve/[^'java']+$",
        dataUrlfilter = "/cve/[^'java']+$",
        crawlZoneXPath = "//div[@id='searchresults']",
        ExtractXpathPat = c("//*[contains(concat( ' ', @class, ' ' ), concat( '
                    ', 'col-md-12', ' ' ))]//a", #Name CVE
                    "//*[contains(concat( ' ', @class, ' ' ), concat( ' ',
                        'ps-2', ' ' )) and (((count(preceding-sibling::*) + 1)
                        = 1) and parent::*)] | //*[contains(concat( ' ',
                        @class, ' ' ), concat( ' ', 'cvssbox', ' ' ))]", #CVSS
                    "//*[contains(concat( ' ', @class, ' ' ), concat( ' ',
                        'd-inline-block', ' ' )) and
                        (((count(preceding-sibling::*) + 1) = 1) and
                        parent::*)]"), # Datum
        PatternsNames = c("CVE", "CVSS", "Datum"),
        no_cores = 4,
        no_conn = 4,
        MaxDepth = 1,
        RequestsDelay = 0.1,
```

[22] https://wordpress.com/de/
[23] https://www.qemu.org/
[24] https://ffmpeg.org/
[25] https://www.kernel.org/
[26] https://www.cvedetails.com/vendor/2337/Wordpress.html

```
        saveOnDisk = FALSE)

 CVElist_wp <- do.call("rbind", DATA) %>% data.frame()
})

CVE_wp <- CVE_wp %>%
 mutate(across(everything(), as.character)) %>%
 mutate(across(everything(), ~replace(., . == "." , ","))) %>%
 mutate(across(CVSS, as.numeric)) %>%
 mutate(PageID = 1:nrow(.)) %>%
 tibble::remove_rownames()
```

Daten-Verarbeitung CVE Da die Veröffentlichungsdaten nur als Strings gecrawlt werden können, welche noch weitere für die Analyse uninteressante Informationen enthalten (siehe B.2), werden die Daten im nächsten Schritt bereinigt. Die Bereinigung der Veröffentlichungsdaten-Spalte wird in Excel mit der Funktion *TEIL* durchgeführt, durch welche ausschließlich ausgewählte Stellen einer Zeichenfolge zurückgegeben werden. Aus der Veröffentlichungsdaten-Spalte wird das Datum der Veröffentlichung ausgeschnitten und im Weiteren für die Analyse verwendet. Die originale Spalte entfällt.

Der bereinigte Datensatz wird in R eingelesen. Dort wird die Einteilung in die Schweregrad-Kategorien mindestens fünf, mindestens sieben und mindestens neun vorgenommen. Dies wird durch eine for-Schleife realisiert, welche prüft, in wie vielen Felder jeweils ein CVSS-Score größer/gleich fünf, sieben oder neun steht, gleichsam dem in Abschnitt 3.1 (Verarbeitung) aufgeführten Code.

LoC-Extraktion Um die Betrachtung der LoC-Entwicklung von WordPress zu ermöglichen, wird das GitHub-Repository von WordPress[27] heruntergeladen. Die verschiedenen Versionen von WordPress sind ab Version 1.5 als branches (dt. Zweige) in dem Repository verfügbar[28].

Zur Extraktion der LoC aller Code-Dokumente der verfügbaren Versionen, wird die Git-Bash für Windows verwendet. Mit dem Befehl

$ git switch branch-name

kann zwischen den Versions-Branches gewechselt werden.

Für jeden Versions-Branch wird der Befehl

$ git ls-files | grep -v ".jpg"| grep -v ".png"| grep -v ".gif"| grep -v "readme"| grep -v "license"| xargs wc -l | grep "total"

[27] `https://github.com/WordPress/WordPress`
[28] Bei Linux sind die Versionen nicht in branches sondern in tags (dt. Markierungen) abgelegt.

ausgeführt, welcher die über alle enthaltenen Dokumente summierte LoC des jeweiligen Branches und somit der jeweiligen Version aufführt. Die aufgeführte LoC-Anzahl beinhaltet Leerzeilen. Alle enthaltenen Bilder und Gifs sowie die README- und Lizens-Dateien werden bei der Extraktion der LoC nicht berücksichtigt.

Die extrahierten LoC der verschiedenen Versionen von WordPress werden mit den dazugehörigen Versionen in einen Datensatz eingepflegt.

Versionsdaten-Extraktion Um die LoC-Daten in ihrer Entwicklung den CVE-Daten gegenüberstellen und vergleichen zu können, werden die Veröffentlichungsdaten der verschiedenen WordPress Versionen benötigt. Die Veröffentlichungsdaten der Versionen von WordPress sind auf der offiziellen WordPress-Webseite[29] verfügbar. Die Veröffentlichungsdaten werden im erstellten Datensatz, der bereits die WordPress-Versionen und ihre LoC-Anzahl enthält, zugeordnet. Somit lässt sich eine Verbindung zwischen LoC und dem Veröffentlichungsdatum der jeweiligen Version herstellen, was einen Vergleich mit den CVE von WordPress ermöglicht.

Visualisierung Zur Visualisierung der LoC-Entwicklung wird in R mittels ggplot2 ein Diagramm erstellt, welches die LoC der Versionen zu ihren Veröffentlichungsdaten darstellt. Die Grundlage bildet dabei ein Punktediagramm, bei welchem jeder Punkt eine Version repräsentiert. Über die Punkte wird mit der Funktion *geom_smooth* eine Trendlinie gelegt, um den Trend der Entwicklung veranschaulichen zu können.

```
# WordPress LOC Verlauf

datum <- wordpress.df$Datum %>% as.Date(tryFormats = "%d.%m.%Y")

ggplot(wordpress.df, aes(x=datum)) +
  geom point(aes(y=LOC, color="LOC"), alpha=0.2) +
  geom smooth(aes(y=LOC), color = 'black', se=F) +
  labs(x="Jahr", y="LOC WordPress Versionen") +
  scale color manual("",
                breaks = c("LOC"),
                values = c('black'))
```

Zur Gegenüberstellung der LoC-und CVE-Entwicklung wird mittels der plot-Funktion von R ein Liniendiagramm mit zwei y-Achsen erstellt. Die linke y-Achse umfasst den CVE-Wertebereich, während die rechte y-Achse den LoC-Wertebereich abbildet. Die Linien der einzelnen Kategorien werden mit *lines* erstellt. Die Achsen werden mit *axis* initialisiert.

```
# WordPress LOC und CVE

par(mar=c(5,4,4,4) + 0.3)

plot(jahr, cve, type = "l", xlab = "Jahr", ylab = "CVE Anzahl", axes = F, col =
    "blue")
axis(1, at=seq(2005,2023, by=1))
axis(2, at=seq(0,450, by=50), las=1)
```

[29] https://wordpress.org/documentation/article/learn-about-wordpress-and-version-history/

```
lines(jahr, cve5, col = "green")
lines(jahr, cve7, col = "orange")
lines(jahr, cve9, col = "red")
par(new = T)
plot(jahr, loc, type = "l", col = "black", axes = F, xlab = "", ylab = "")
axis(side = 4, at = seq(0, 4, by=0.2), las=1)
mtext("LOC", side = 4, line = 3)

legend("topleft",
     legend = c("CVE gesamt", "CVSS 5+", "CVSS 7+", "CVSS 9+", "LOC"),
     pch = c(20, 20, 20, 20, 20),
     col = c("blue", "green", "orange", "red", "black"))
```

Analyse der Entwicklung Um den Trend der Entwicklung der CVE bzw. LoC der einzelnen Software-Systeme zu bestimmen, wird wie in Abschnitt 3.1 (Analyse) aufgeführt ein Log-Plot von der jeweiligen ersten Ableitung der CVE bzw. LoC jedes Software-Systems gebildet.

Analyse der Korrelation der CVE und LoC Zum Testen der Korrelation zwischen CVE und LoC wird die R-Funktion *cor.test*[30] verwendet. Mit Hilfe dieser Funktion wird auf „Assoziation zwischen gepaarten Stichproben unter Verwendung von Pearson's Produkt-Moment-Korrelationskoeffizient [über] Kendall's τ oder Spearman's ρ" [49] getestet. Da die der Funktion zugrundeliegenden Datensätze gepaart sein müssen, wird jeweils die letzte Version, welche in einem Jahr veröffentlicht wurde, als Repräsentant des Jahres genommen. Somit wird ein neuer Datensatz angelegt, welcher die LoC der jeweilig letzten Version jedes Jahres dem Jahr gegenüberstellt. Der neue Datensatz der LoCs ist auf den Datensatz der Gesamt-CVE angepasst, welcher für jedes Jahr die Anzahl der CVE enthält. Es wird dabei der CVE-Datensatz mit der kumulierten CVE-Anzahl genutzt. Beide Datensätze werden der *cor.test*-Funktion übergeben. Da sich Kendalls τ gut für den Umgang mit kleinen Stichprobengrößen eignet [50], wird die *cor.test*-Funktion zur Überprüfung der Korrelation der CVE und LoC mit der Methode Kendall durchgeführt. Als Ergebnis wird eine Zahl im Bereich von -1 bis 1 geliefert [50]. Je näher τ dabei an 1 ist, desto wahrscheinlicher liegt eine Korrelation vor.

```
## Korrelation CVE LOC

cor.test(cve, loc)

cor.test(cve, loc,
       alternative = "two.sided",
       method = "kendall")
```

Normalisierung von LoC und CVE Zur Veranschaulichung werden die LoC- und CVE-Daten normalisiert. Dabei werden als erstes die LoC für jedes Jahr über die CVE normalisiert, wodurch sich die durchschnittliche Anzahl von LoC auf eine CVE ergibt. Die

[30] https://www.rdocumentation.org/packages/stats/versions/3.6.2/topics/cor.test

dadurch entstehende Liste wird ab dem erstem Auftreten einer CVE bis 2022 geplottet. Um ein unendliches Ergebnis zu vermeiden, werden alle Einträge, bei denen eine CVE-Anzahl von Null vorliegt zu Null.

```
wp norm <- list()

i <- 1
for (i in 1:length(cve)) {
  if(cve[i]==0) {
    norm <- 0
  }
  else {
    norm <- loc[i]/cve[i]
  }
  wp norm <- append(wp norm, norm)
}

plot(jahr, wp norm, xlab = "Jahr", ylab = "LOC pro CVE", type = "b")
```

Des Weiteren wird die *Vulnerability Density*, wie in Abschnitt 2.1 beschrieben, für jedes Jahr berechnet und visualisiert. Genutzt wird hierbei die „known vulnerability density“, welche die Anzahl der (öffentlich) bekannten Sicherheitslücken bezüglich der Code-Größe beschreibt [7]. Die Code-Größe wird durch die LoC-Anzahl der letzten Version jedes Jahres repräsentiert. Alle Einträge, bei denen keine LoC-Anzahl vorliegt, werden zu Null, um einen Fehler bei der Ausgabe zu vermeiden.

```
wp norm <- list()

i <- 1
for (i in 1:length(cve)) {
  if(is.na(loc[i])) {
    norm <- 0
  }
  else {
    norm <- cve[i]/loc[i]
  }

  wp norm <- append(wp norm, norm)
}

plot(jahr, wp norm, xlab = "Jahr", ylab = "Vulnerability Density", type = "b")
```

3.3 Entwicklung der CVE mit kritischem Schweregrad

Um die Entwicklung der Sicherheitslücken mit einem kritischen Schweregrad im Vergleich zu der Entwicklung aller Sicherheitslücken zu betrachten, werden mit Hilfe von Excel-Tabellen Grafiken erstellt, die den prozentualen Anteil dieser Beziehung darstellen. Dazu wird die Anzahl der CVE mit einem CVSS-Score von mindestens neun durch

die Anzahl aller CVE für jedes Jahr geteilt. Das Ergebnis davon wird in Prozent umgerechnet. Es werden Grafiken für die Gesamt-CVE sowie für die einzelnen betrachteten Software-Produkte erstellt, welche für jedes Jahr die Anzahl aller entdeckten CVE, die Anzahl aller entdeckten CVE mit CVSS-Score von mindestens neun und den sich daraus ergebenden prozentualen Anteil der CVE mit CVSS-Score von mindestens neun in Bezug auf die Gesamt-CVE des Jahres darstellt. Dies wird für die nicht kumulierten sowie die kumulierten Daten durchgeführt.

4 Ergebnisse

Zur Bearbeitung der Forschungsfrage, ob und warum die Anzahl der IT-Sicherheitslücken in Form von CVE exponentiell wächst, werden die Ergebnisse der Methoden ausgewertet und nachfolgend dargelegt.

4.1 CVE-Entwicklung

Um die Entwicklung der Sicherheitslücken in den vergangenen Jahren auszuwerten, wurden alle vorhandenen CVE-Daten einer Sicherheitslücken-Datenbank extrahiert, verarbeitet und visualisiert.

Entwicklung CVE Abbildung 4.1 zeigt die entdeckten CVEs für jedes Jahr von 1999 bis 2022. Die blauen Punkte stehen für das Aufkommen der gesamten CVEs eines Jahres, für jedes Jahr von 1999 bis 2022. Die grünen Punkte stellen das Aufkommen aller CVEs mit einem CVSS-Score von mindestens fünf für jedes Jahr dar. Die orangenen Punkte zeigen alle CVEs mit einem CVSS-Score von mindestens sieben für jedes Jahr und die roten Punkte das Aufkommen aller CVEs mit einem CVSS-Score von mindestens neun. Alle Jahresgesamtaufkommen der einzelnen Kategorien in Form der Punkte, werden jeweils durch die dazugehörige gleichfarbige Trendlinie überlagert.

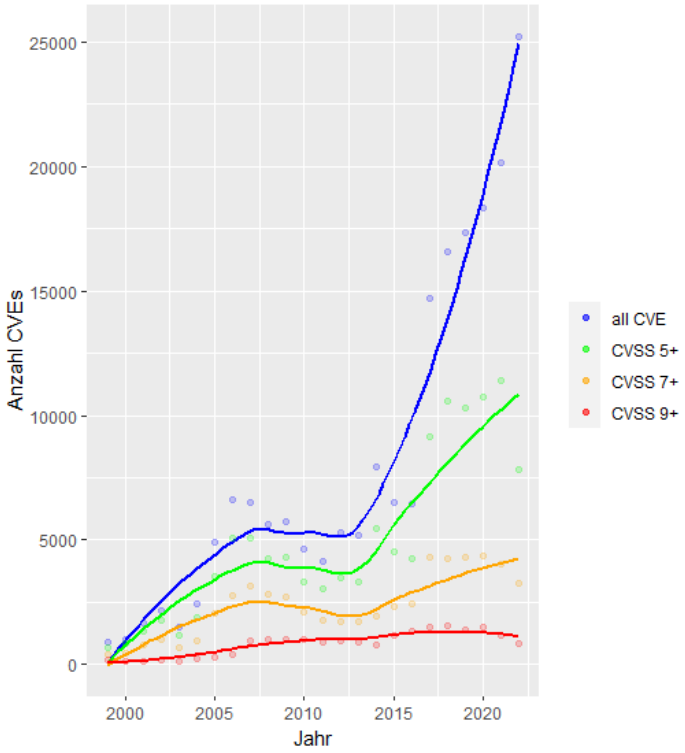


Abbildung 4.1: Entwicklung CVE

Die blauen Punkte zeigen an, dass die gesamt entdeckten CVE jedes Jahres erst gemäßigt ansteigen, von 2007 bis 2011 allerdings wieder leicht abfallen. Ab 2013 steigt das Aufkommen der gesamten CVE für jedes weitere Jahr stark an. Optisch könnte es sich

bei diesem Anstieg um exponentielles Wachstum handeln. Der abrupte Anstieg spiegelt sich ebenfalls in abgeflachter Form in der Entwicklung der CVE mit einem CVSS-Score von mindestens fünf wieder. Die CVE mit einem hohen bzw. kritischen CVSS-Score steigen hingegen nur langsam und leicht an. Bei den CVE mit CVSS-Score von mindestens sieben ist zwischen 2007 und 2013 ein Abstieg erkennbar, danach steigt ihr Aufkommen allerdings ebenfalls wieder etwas stärker an. Wie in Abbildung C.1 dargestellt, werden 2005, 2014, 2017 und 2022 die größten Differenzen der CVE-Aufkommen zwischen den Jahren aufgezeichnet.

Um die Art der CVE-Entwicklung zu überprüfen, wurde ein Log-Plot über die Differenzen der CVE zwischen den Jahren erstellt. Der Plot wird in Abbildung 4.2 dargestellt. Es ist eine Gerade mit positivem Anstieg erkennbar. Die Gerade ist von großer Streuung und Abweichungen z.B. 2007 geprägt. Der Log-Plot kann als technischer Indikator gewertet werden, dass es sich bei der Entwicklung von CVE mit hinreichender Wahrscheinlichkeit um einen annähernd exponentiellen Trend handelt.

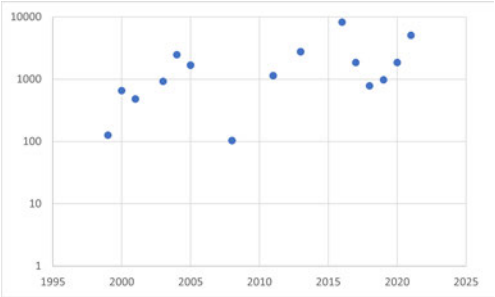
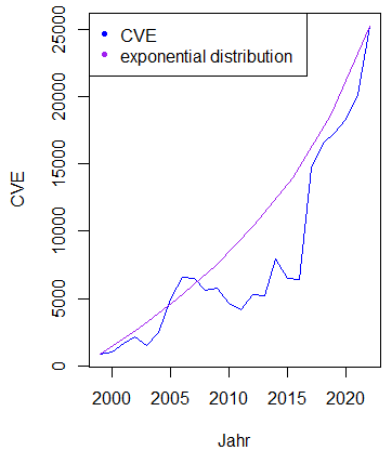
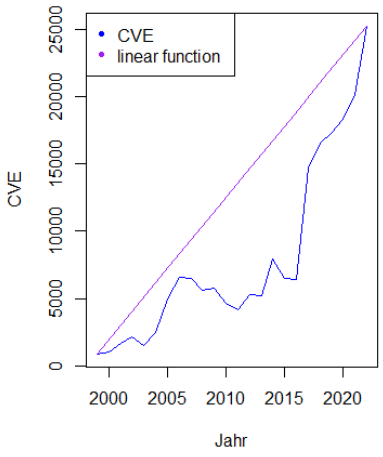


Abbildung 4.2: Log-Plot CVE-Entwicklung

Abbildung 4.3 stellt den CVE-Entdeckungen einer exponentiellen (4.3a) und linearen Funktion (4.3b) gegenüber. Die CVE-Entwicklung passt sich dabei passend zu dem Ergebnis des Log-Plots optisch der exponentiellen Funktion mehr an als der linearen Funktion.



(a) CVE Entwicklung vs exponentielle Funktion



(b) CVE Entwicklung vs lineare Funktion

Abbildung 4.3: Entwicklung CVE optischer Abgleich

Entwicklung CVE kumulativ Abbildung 4.4 zeigt die Entwicklung der von 1999 bis 2022 entdeckten CVE für jedes Jahr kumuliert. Es wurde die gleiche Farbverteilung wie in Abbildung 4.1 gewählt.

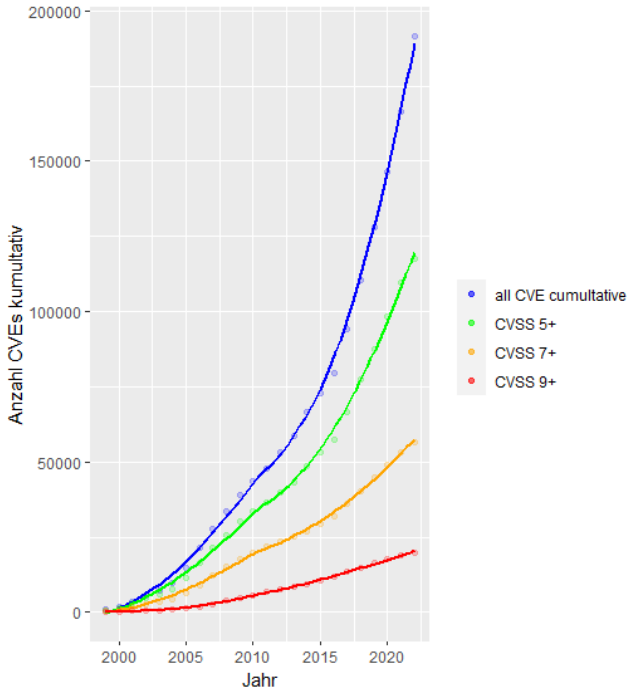


Abbildung 4.4: Entwicklung CVE kumulativ

Die Abbildung zeigt einen monotonen Anstieg aller CVE über den betrachteten Zeitraum. Die Gesamt-CVE steigen rapide an und scheinen dabei optisch einen exponentiellen Trend zu verfolgen. Der starke Anstieg ist in abgeflachter Form auch bei den CVE mit einem CVSS-Score von mindestens fünf erkennbar, während die CVE mit CVSS-Score von mindestens sieben und mindestens neun einen leichteren Anstieg zeigen.

Um zu überprüfen, ob der Anstieg der Anzahl der kumulierten CVE über den Zeitraum hinweg exponentiell ist, wurde ein Log-Plot über die Differenzen der kumulierten CVE zwischen den Jahren erstellt. Die Punkte in diesem Plot bilden, wie in Abbildung 4.5 zu erkennen, eine Gerade mit positivem Anstieg. Dies kann als technischer Indikator gewertet werden, dass die kumulierte Entwicklung der CVE mit hinreichender Wahrscheinlichkeit einem exponentiellen Trend folgt.

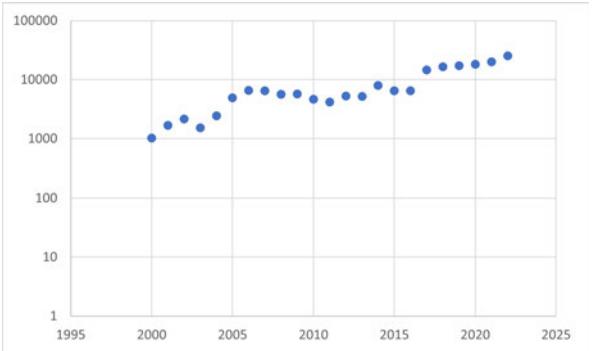


Abbildung 4.5: Log-Plot kumulierte CVE

4.2 LoC und CVE

Um einen Zusammenhang zwischen LoC und CVE herstellen zu können, wurden verschiedene Schritte an vier exemplarischen Software-Systemen durchgeführt. Im ersten Schritt wurde die Entwicklung der CVE der jeweiligen Software-Systeme betrachtet, im zweiten die Entwicklung der LoC der Software-Systeme. Anschließend wurde die Entwicklungen einander gegenübergestellt sowie auf Korrelation getestet. Abschließend wurden CVE und LoC normalisiert und der dazugehörige Verlauf betrachtet.

4.2.1 WordPress

Das erste betrachtete Software-System ist WordPress, ein Content-Management-System zur Veröffentlichung und Verwaltung von Inhalten, welches auf PHP und MySQL basiert [51]. Gegenwärtig wird WordPress von über 43% aller Webseiten verwendet [51, 52]. Der Quellcode des Open-Source-Projekt der Versionen 1.5 bis 6.2 ist auf GitHub verfügbar[31].

CVE Entwicklung WordPress Abbildung 4.6 visualisiert die Entwicklung der in WordPress entdeckten CVE ab dem ersten Fund von 2004 bis Ende 2022. In Abbildung 4.6a ist die Anzahl der entdeckten CVE für jedes Jahr abgebildet. Die Verteilung der Anzahl weist keine klare Struktur im Auftreten der CVE über die Jahre auf. Zudem sind viele Ausreißer von der Trendlinie zu erkennen. Bei Betrachtung der Abbildung 4.6b, welche die kumulative Entwicklung der Entdeckung von CVE bei WordPress zeigt, ist hingegen bei den Gesamt-CVE (blau) ist ein klarer linear wirkender Anstieg zu betrachten. Ein Anstieg kann auch bei allen anderen CVE-Kategorien betrachtet werden. Dabei steigen CVEs mit CVSS-Score über sieben nur leicht. Bei CVE mit CVSS-Score über neun scheint das Wachstum zu stagnieren. Bei den Trendlinien sind keine Ausreißer zu betrachten. Alle Zahlen finden sich Nahe der Trendlinie.

(a) CVE WordPress **(b)** CVE WordPress kumulativ

Abbildung 4.6: Entwicklung CVE WordPress

[31] https://github.com/WordPress/WordPress

Um die Art der Entwicklung der CVE in WordPress zu analysieren, wurde ein Log-Plot über die Differenzen der kumulierten CVE zwischen jedem Jahr erstellt. Die Punkte des Plots bilden, wie in Abbildung 4.7 dargestellt, keine klare Linie. Da die Punkte sich allerdings, bis auf zwei Ausreißer in den Jahren 2005 und 2011, auf einer Ebene zu befinden scheinen, könnte dies grob als Gerade ohne Anstieg mit viel Streuung gewertet werden. Dies würde als technischer Indikator mit hinreichender Wahrscheinlichkeit für einen linearen Trend der CVE-Entwicklung in WordPress sprechen.

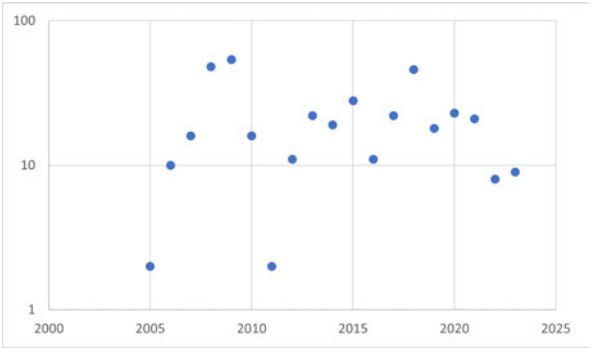


Abbildung 4.7: Log-Plot CVE WordPress

LoC Entwicklung WordPress Die Entwicklung der LoC von WordPress ab dem Veröffentlichungsdatum der ersten auf GitHub verfügbaren Version 2005 bis 2022 ist in Abbildung 4.8 visualisiert. Die LoC stehen für jedes Jahr einzeln und sind nicht kumuliert. Nichtsdestotrotz steigt die LoC-Anzahl von rund 50.000 im Jahr 2005 auf mehr als 1.5 Millionen im Jahr 2022 stetig stark an, ohne merkliche Stagnation oder Rücklauf. Der Verlauf der Kurve lässt ein lineares bis exponentielles Wachstum vermuten.

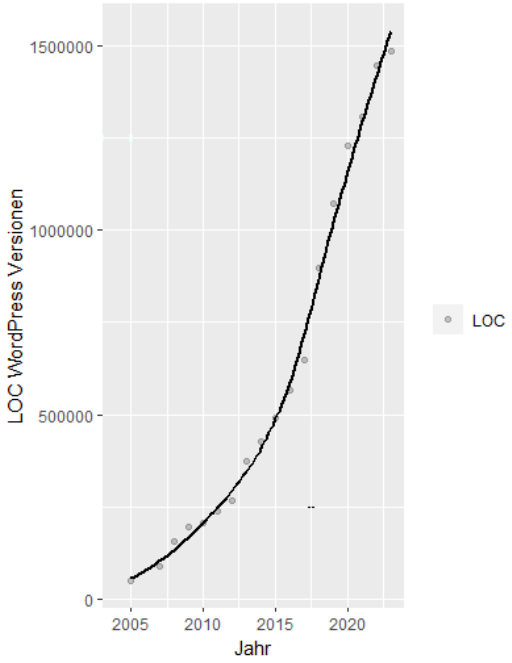


Abbildung 4.8: Entwicklung LoC WordPress

Um die Art des Wachstums der LoC zu überprüfen, wurde ein Log-Plot über die Differenzen der letzten LoC jedes Jahres erstellt. Wie Abbildung 4.9 zeigt, bilden die Punkte des Plots eine Linie. Diese scheint teils leicht anzusteigen, wirkt aber in der Gesamtbe-

trachtung wie eine Gerade ohne Anstieg. Dies kann als technischer Indikator gewertet werden, dass die LoC-Entwicklung von WordPress im großen Trend mit hinreichender Wahrscheinlichkeit linear ist.

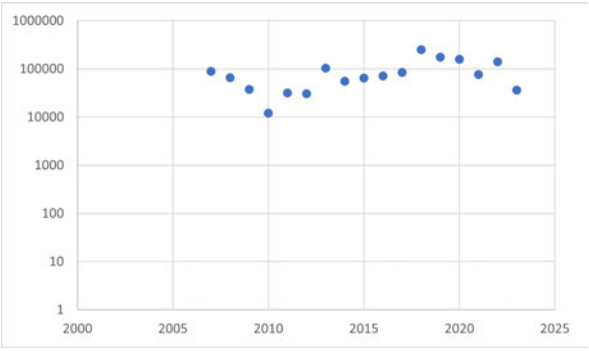


Abbildung 4.9: LogPlot LOC WordPress

Gegenüberstellung von CVE und LoC von WordPress Abbildung 4.10 stellt die Entwicklung der LoC von WordPress den kumulierten gefundenen CVE gegenüber. Dabei wird ein ähnlicher Kurvenverlauf zwischen LoC und Gesamt-CVE deutlich.

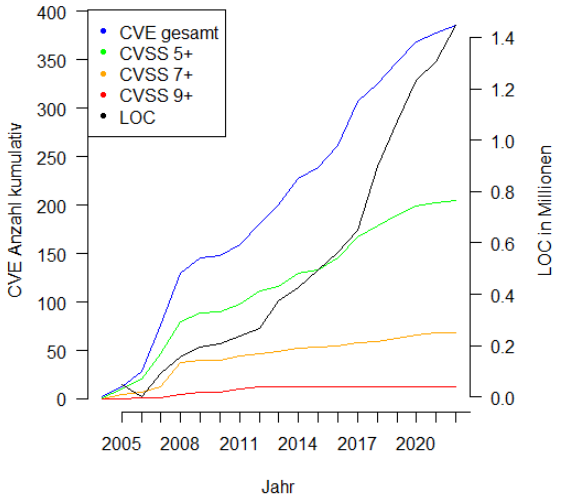
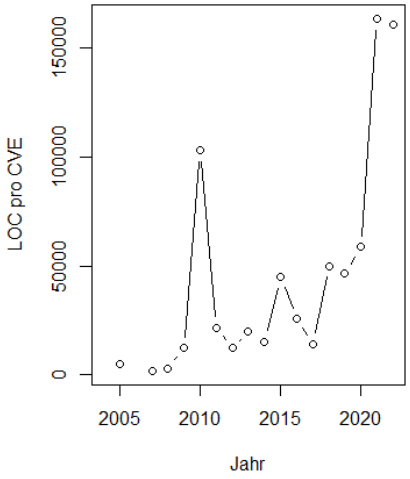


Abbildung 4.10: Vergleich der CVE- und LoC-Entwicklung von WordPress

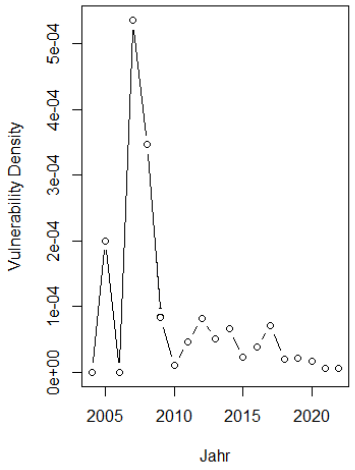
Die vermutete Ähnlichkeit / Assoziation der LoC und CVE wurde mit Hilfe eines Korrelationstestes weitergehend geprüft. Die Assoziationsmaße der verwendeten Methoden liegen jeweils alle im Bereich [-1,1], wobei 1 bedeutet, dass eine Assoziation vorliegt. Das Ergebnis der Methode „Kendall“ für τ ist 1. Es existiert somit eine Korrelation zwischen LoC und CVE von WordPress.

Normalisierung von LoC und CVE Um die Entwicklung der Anzahl der CVE im Bezug auf die LoC im Verlauf der Jahre anhand von vergleichbaren Werten betrachten zu können, wurden die CVE- und LoC-Anzahlen normalisiert. Dabei wurde als erstes eine Normalisierung der LoC über die CVE von WordPress durchgeführt. Abbildung 4.11a zeigt die durchschnittliche Anzahl der LoC pro CVE einer WordPress-Version für jedes

Jahr. Die Abbildung weist einen Trend in Richtung immer mehr LoC im Durchschnitt auf eine CVE. Der Anteil der CVE im Code sinkt somit. Dieser Trend wird auch in Abbildung 4.11b deutlich, welche die *Vulnerability Density* von WordPress über die Jahre hinweg darstellt. Während 2007 die höchste Sicherheitslückendichte aufgezeichnet wurde, sinkt die Dichte über die folgenden Jahre ab. Abbildung 4.11 zeigt somit, dass der Anteil der Sicherheitslücken bezüglich der Code-Länge einen abnehmenden Trend verfolgt.



(a) Normalisierung LoC/CVE



(b) Vulnerability Density (CVE/LoC)

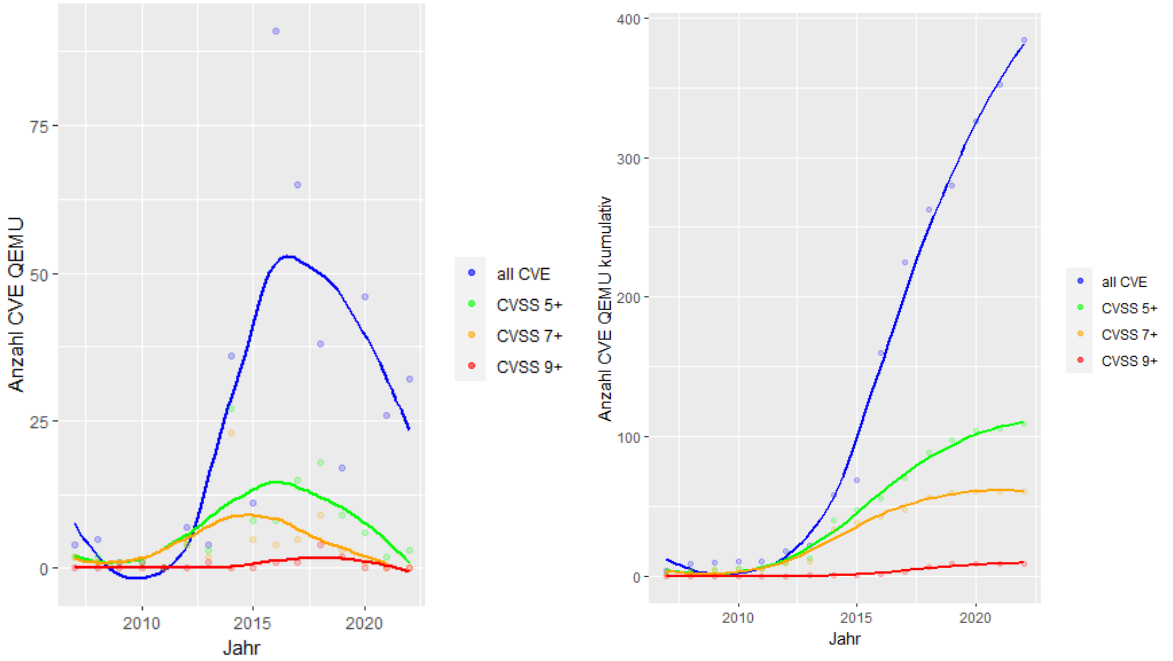
Abbildung 4.11: Normalisierter Vergleich WordPress

4.2.2 QEMU

QEMU ist „ein generischer und quelloffener Maschinenemulator und Virtualisierer“ [53]. Der Quellcode der Versionen 0.10 bis 8.0 ist mit einigen Ausnahmen der neueren Versionen fast vollständig auf GitHub verfügbar[32].

CVE Entwicklung QEMU Abbildung 4.12 visualisiert die Entwicklung der in der Software QEMU gefundenen CVE vom ersten Fund 2007 bis 2022. In Abbildung 4.12a ist die Anzahl der entdeckten CVE für jedes Jahr einzeln aufgezeigt. Während 2007 bis 2013 nur wenige CVE gemeldet wurden, steigt ihre Anzahl ab 2014 deutlich an. Im Jahr 2016 wurden die gegenwärtig meisten CVE aufgezeichnet. Auch wenn die anschließende Trendlinie ein große Streuung aufweist, ist ab 2017 ein deutlicher Abfall der Entdeckungen von CVE erkennbar. Der selbe Trend wird auch in den anderen Kurven der CVE mit CVSS-Score ab fünf widergespiegelt, ist dabei aber weniger ausgeprägt. Werden die Kurvenverläufe miteinander verglichen, so besitzt (mit hoher Wahrscheinlichkeit) die Mehrheit der CVE einen CVSS-Score von unter fünf. In Abbildung 4.12b sind die entdeckten CVE über die Jahre kumuliert abgebildet. Auch in dieser Abbildung ist ab 2014 ein starker Anstieg zu betrachten, der ab 2017 leicht abflacht. Dabei scheint optisch weder ein linearer noch exponentieller Trend verfolgt zu werden. Es ist auffällig, dass es bei QEMU kaum CVE mit CVSS-Score von mindestens neun zu geben scheint.

[32] `https://github.com/qemu/qemu`



(a) CVE QEMU

(b) CVE QEMU kumulativ

Abbildung 4.12: Entwicklung CVE QEMU

Um die Art der Entwicklung der CVE in QEMU zu analysieren, wurde ein Log-Plot über die Differenzen der kumulierten CVE zwischen jedem Jahr erstellt. Wie in Abbildung 4.13 zu sehen, bilden die Punkte des Plots keine Linie. Dies kann als technischer Indikator gewertet werden, dass die Entwicklung der CVE in QEMU mit hinreichender Wahrscheinlichkeit, gleichsam zur optischen Beobachtung, keinen exponentiellen oder linearen Trend verfolgt.

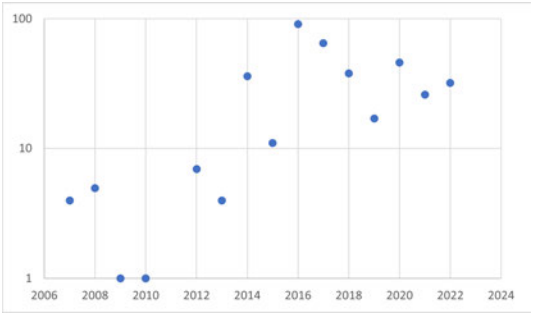


Abbildung 4.13: Log-Plot CVE QEMU

LoC Entwicklung QEMU Abbildung 4.14 zeigt die Entwicklung der LoC von dem Veröffentlichungsdatum der ersten auf GitHub verfügbaren Version von QEMU 2009 bis 2022. Es ist ein stetiger Anstieg zu beobachten, der ab 2016 steiler wird. Die Entwicklung scheint optisch einen linearen bis exponentiellen Trend zu verfolgen.

Um die Art des Wachstums der LoC in QEMU zu analysieren, wurde ein Log-Plot über die Differenzen der letzten LoC jedes Jahres erstellt. Die Punkte des Plots bilden, wie in Abbildung 4.15 ersichtlich eine Gerade ohne erkennbaren Anstieg. Dies kann mit hinreichender Wahrscheinlichkeit als technischer Indikator für einen linearen Trend der Entwicklung der LoC gewertet werden.

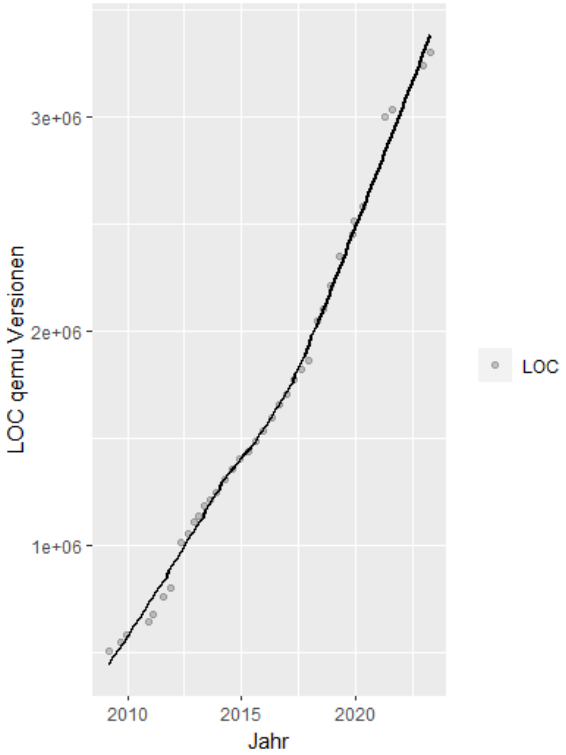


Abbildung 4.14: Entwicklung LoC QEMU

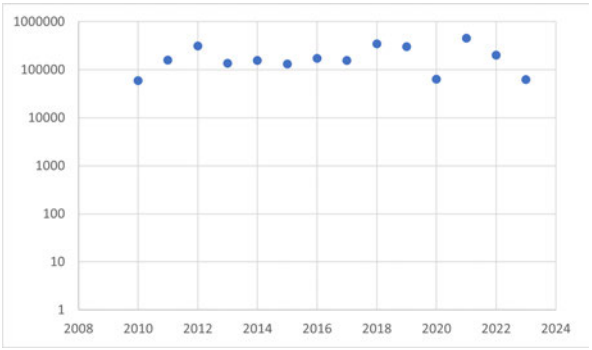


Abbildung 4.15: Log-Plot LOC QEMU

Gegenüberstellung von CVE und LoC von QEMU Die Gegenüberstellung der Entwicklung der LoC von QEMU und der kumulierte Entwicklung der CVE von QEMU ist in Abbildung 4.16 visualisiert. Während die Entwicklung der CVE mit CVSS-Score von mindestens fünf nur sehr langsam zu wachsen scheint, zeigen die Entwicklung der LoC und Gesamt-CVE wiederum eine gewisse Ähnlichkeit.

Zur Überprüfung dieser Ähnlichkeit wurde erneut ein Korrelationstest durchgeführt, welcher testet, ob zwischen LoC-Entwicklung und Gesamt-CVE Entwicklung eine Assoziation möglich ist. Das Ergebnis des Korrelationstest mit der Methode „Kendall“ für τ ist 0.9944903. Somit scheint eine Korrelation zwischen LoC-Entwicklung und CVE-Entwicklung von QEMU möglich.

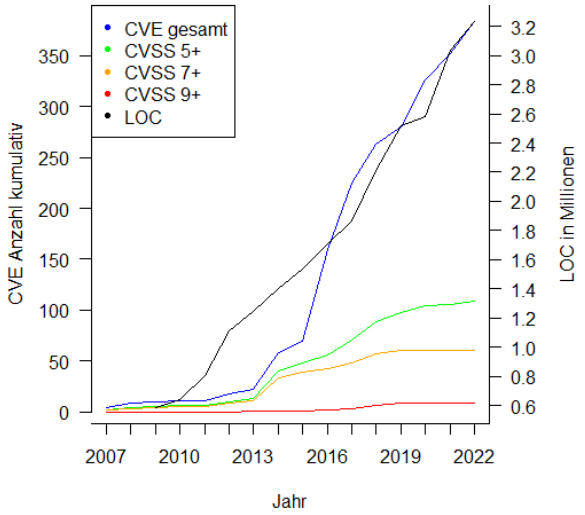
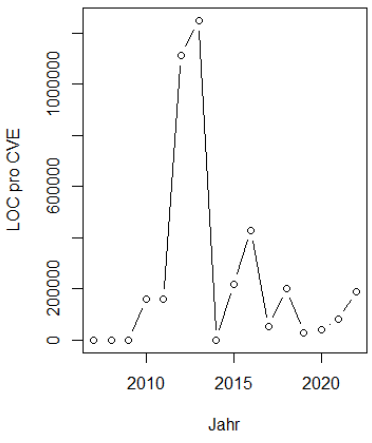
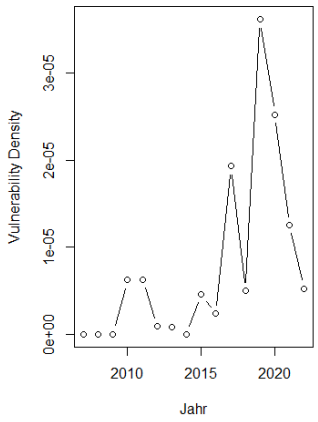


Abbildung 4.16: Vergleich der CVE- und LoC-Entwicklung von QEMU

Normalisierung von LoC und CVE In Abbildung 4.17 werden die Entwicklungen LoC und CVE von QEMU normalisiert dargestellt. Abbildung 4.17a zeigt die normalisierten LoC pro CVE für die letzte QEMU-Version jedes Jahres ab der Entdeckung der ersten CVE bis 2022. Der Höhepunkt mit rund 1.500.000 LoC pro CVE liegt dabei im Jahr 2013. Mit einiger Streuung in den Jahren 2016 und 2018 scheint der Trend anschließend abzusinken. Ab 2020 ist allerdings erneut eine leichte Erhöhung der Anzahl der LoC pro CVE zu erkennen. Auch wenn dies nur bis gegenwärtig 200.000 LoC pro CVE führt, scheint sich der Trend dahin zu Entwickeln, dass mehr Code auf eine CVE kommt. Abbildung 4.17b zeigt die *Vulnerability Density* von QEMU im Laufe der Jahre. Hierbei verdeutlicht sich, dass die höchste Sicherheitslückendichte 2019 aufgezeichnet wurde. Seitdem nimmt die *Vulnerability Density* stetig ab. Abbildung 4.17 verdeutlicht somit, dass der Anteil der CVE in Code bis 2019 zugenommen hat. Ab 2020 sinkt der Anteil.



(a) Normalisierung LoC/CVE



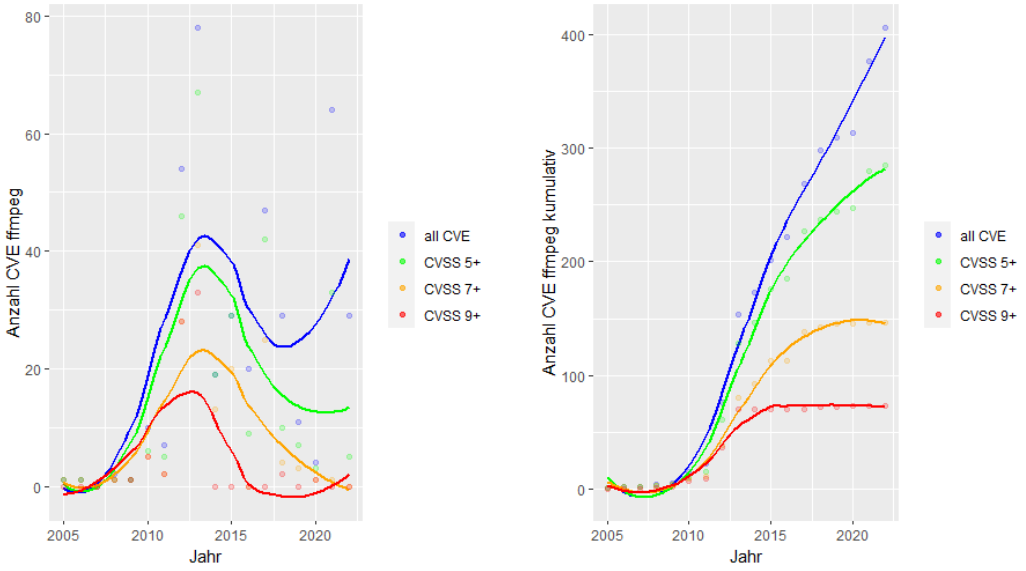
(b) Vulnerability Density (CVE/LoC)

Abbildung 4.17: Normalisierter Vergleich QEMU

4.2.3 FFmpeg

FFmpeg ist eine „vollständige, plattformübergreifende Lösung zum Aufnehmen, Konvertieren und Streamen von Audio und Video“ [54]. Der Quellcode ist von Version 0.5 bis zur neusten Version 6.0 auf GitHub verfügbar[33].

CVE Entwicklung FFmpeg Abbildung 4.18 zeigt die Entwicklung der Anzahl der entdeckten CVE in FFmpeg ab der Entdeckung der ersten FFmpeg-CVE 2005.
In Abbildung 4.18a ist die Anzahl der in FFmpeg gefundenen CVE für jedes Jahr einzeln dargelegt. Es wird ein starker Anstieg der CVE ab 2008 deutlich. 2013 wurden mit einer Anzahl von annähernd 80 die meisten CVE gefunden. Darauf folgend sinkt die Entdeckungsrate für jedes Jahr leicht, wobei die Trendlinie ab 2017 wieder ansteigt. Durch die große Anzahl an Ausreißern, ist kein klaren Trend bestimmbar. Der Verlauf der Trendlinie des Gesamt-CVE findet sich auch in den anderen Kategorien für CVE mit CVSS-Score ab fünf wieder. 2013 ist in allen Kategorien die Anzahl der Entdeckungen am höchsten. Auffällig ist, dass ab 2015 nur noch kaum bis sehr wenige CVE mit CVSS-Score von mindestens sieben bzw. mindestens neun verzeichnet werden.
In Abbildung 4.18b ist das Wachstum der Entdeckungen der CVE über die Jahre kumuliert dargestellt. Bei den Gesamt-CVE lässt sich ab 2008 ein starker Anstieg verzeichnen, der ab 2015 schwächer zu werden scheint. Die Entdeckungen der CVE mit CVSS-Score von mindestens fünf sind bis 2015 fast exakt gleich mit den Gesamt-CVE, wodurch deutlich wird, dass bis 2015 fast alle entdeckten CVE einen CVSS-Score von mindestens fünf haben. Ab 2016 wird die Differenz der Trendlinien größer. Auch in dieser Abbildung fällt auf, dass die CVE mit CVSS-Score von über sieben ab 2020 zu stagnieren scheinen. Bei den CVE mit CVSS-Score von mindestens neun ist dieser Trend schon ab 2014 zu beobachten. Optisch betrachtet, scheint die CVE-Entwicklung weder linearen noch exponentiellen Trend zu verfolgen. Die Kurve ähnelt einer Sättigungsfunktion.



(a) CVE FFmpeg **(b)** CVE FFmpeg kumulativ

Abbildung 4.18: Entwicklung CVE FFmpeg

[33] https://github.com/FFmpeg/FFmpeg

Um die Art der Entwicklung der CVE in FFmpeg zu analysieren, wurde ein Log-Plot über die Differenzen der in WordPress entdeckten kumulierten CVE zwischen jedem Jahr erstellt. Wie in Abbildung 4.19 zu sehen, bilden die Punkte des Plots keine Gerade. Dies kann als technischer Indikator gewertet werden, dass die Entwicklung der CVE in FFmpeg mit hinreichender Wahrscheinlichkeit, gleichsam zur optischen Beobachtung, keinen klar exponentiellen oder linearen Trend verfolgt.

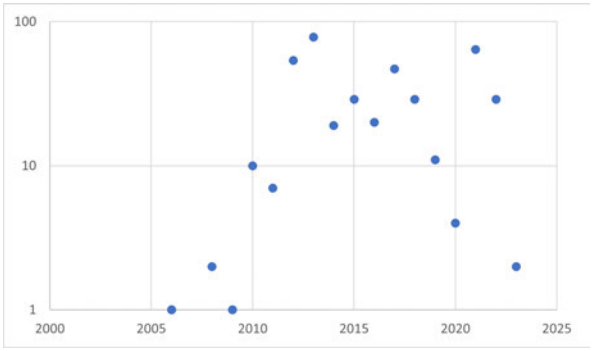


Abbildung 4.19: Log-Plot CVE FFmpeg

LoC Entwicklung FFmpeg Abbildung 4.20 visualisiert die Entwicklung der LoC von FFmpeg ab dem Veröffentlichungsdatum der ersten auf GitHub verfügbaren Version 2009 bis 2022. Dabei wird ein Wachstum von annähernd 60.000 LoC im Jahr 2009 bis knapp 1.9 Millionen LoC im Jahr 2022 verzeichnet. Bis 2017 verläuft das Wachstum dabei stetig steil, ab 2017 flacht der Trend ab.

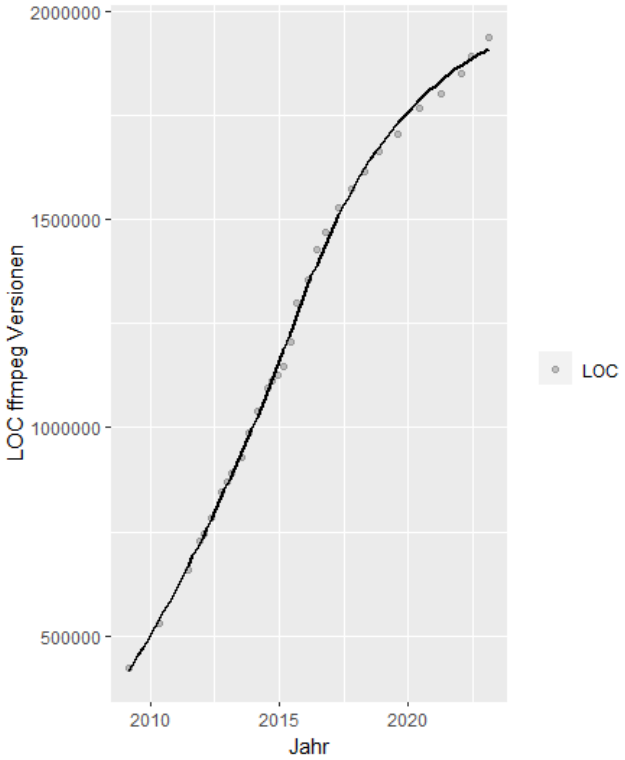


Abbildung 4.20: Entwicklung LoC FFmpeg

Um die Art des Wachstums der LoC von FFmpeg über die Jahre zu bestimmen, wurde ein Log-Plot über die Differenzen der letzten LoC jedes Jahres erstellt. Wie in Abbildung 4.13 zu sehen, bilden die Punkte des Plots eine Gerade mit negativem Anstieg. Der negative Anstieg kann als technischer Indikator gewertet werden, dass die Entwicklung der CVE in QEMU mit hinreichender Wahrscheinlichkeit keine e-Funktion ist und somit keinen exponentiellen Trend verfolgt. Zudem spricht der Indikator gegen einen linearen Trend der LoC-Entwicklung.

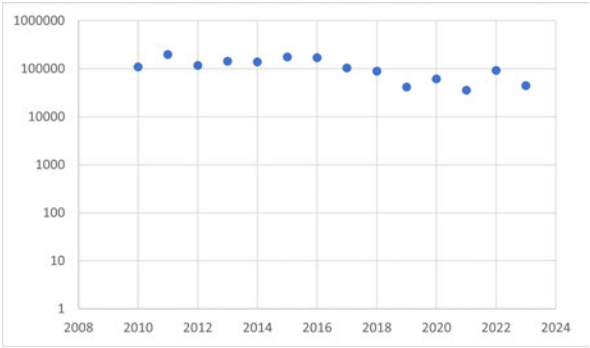


Abbildung 4.21: Log-Plot LOC FFmpeg

Gegenüberstellung von CVE und LoC von FFmpeg In Abbildung 4.22 wird die Entwicklung der LoC von FFmpeg der Entwicklung der kumulierten Anzahl der Entdeckungen der CVE gegenübergestellt. Es ist eine klare Ähnlichkeit zwischen der Entwicklung der LoC und der Entwicklung der kumulierten CVE erkennbar.

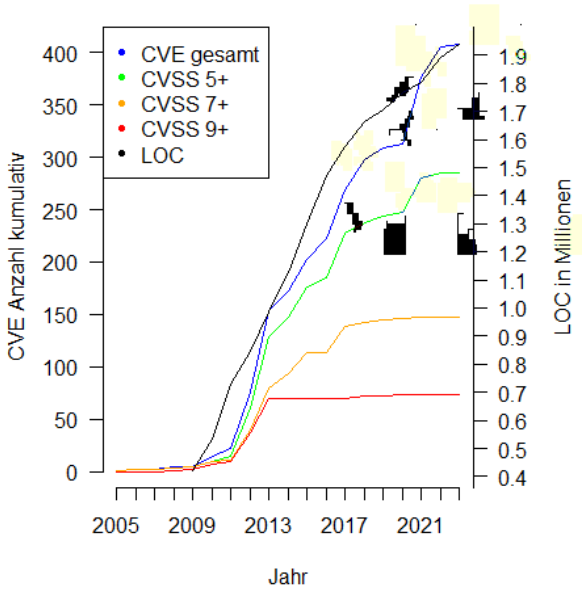
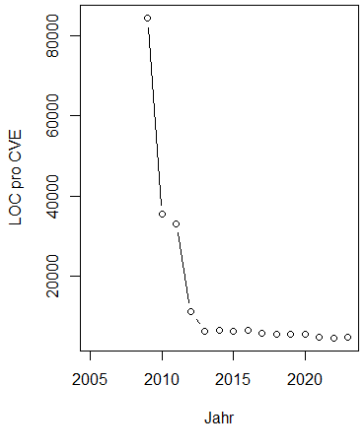


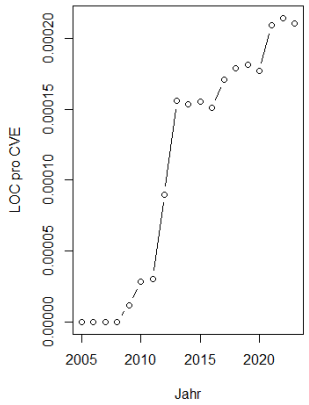
Abbildung 4.22: Vergleich der CVE- und LoC-Entwicklung von FFmpeg

Um eine Assoziation zwischen Entwicklung der LoC und CVE zu überprüfen wird ein Korrelationtest durchgeführt. Das Ergebnis des Korrelationstest mit der Methode „Kendall“ für τ ist 1. Somit existiert eine nachweisbare Korrelation zwischen LoC-Entwicklung und CVE-Entwicklung von FFmpeg.

Normalisierung von LoC und CVE Abbildung 4.23a stellt die Normalisierung der LoC pro CVE in FFmpeg für jedes Jahr ab der ersten Entdeckung einer CVE in FFmpeg bis 2022 dar. Es ist ein klarer Trend erkennbar, dass die LoC pro CVE über die Jahre stark abgenommen haben und gegenwärtig auf niedrigem Level zu stagnieren scheinen. Es gibt somit einen hohen Anteil von CVE im Quellcode bezogen auf durchschnittliche LoC pro CVE. Abbildung 4.23b verdeutlicht diesen Trend. Die *Vulnerability Density* nimmt über die Jahre hinweg zu. Die größten Sprünge der Dichte werden dabei zwischen 2010 und 2012 verzeichnet. Abbildung 4.23 zeigt somit, dass der Anteil der CVE im Code von FFmpeg zugenommen hat.



(a) Normalisierung LoC/CVE



(b) Vulnerability Density (CVE/LoC)

Abbildung 4.23: Normalisierter Vergleich FFmpeg

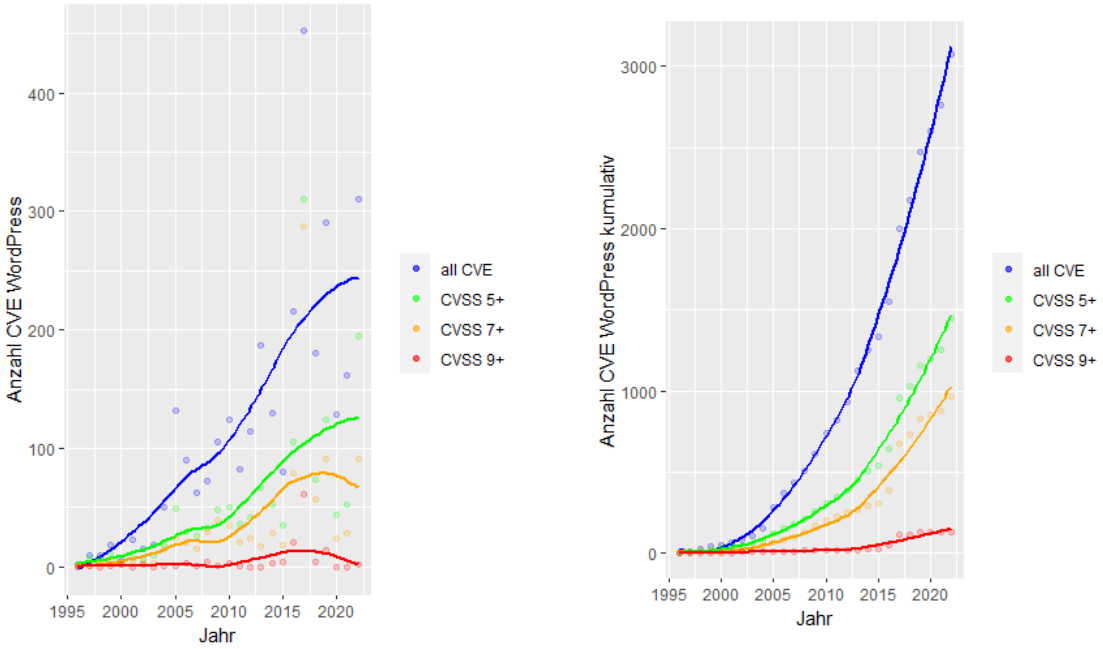
4.2.4 Linux Kernel

Linux Kernel ist eine oft genutzte Komponente von Linux Systemen [55]. Der Quellcode der Versionen 2.6.11 bis 6.5 ist auf GitHub verfügbar[34].

CVE Entwicklung Linux Kernel Abbildung 4.24 zeigt die Entwicklung der Anzahl der von der ersten entdeckten CVE 1996 bis 2022 in Linux Kernel entdeckten CVE. In Abbildung 4.24a sind die Entdeckungszahlen von CVE für jedes Jahre dargestellt. Die Entwicklung über die Jahre ist von viel Streuung und Ausreißern in den Jahren 2017, 2020 und 2021 geprägt, zeigt aber einen steigenden Trend. Die meisten CVE werden im Jahr 2017 verzeichnet. Auch die CVE mit höheren Schweregrade zeigen um diesen Zeitpunkt herum einen Anstieg. CVE mit CVSS-Score von mindestens neun scheinen im Linux Kernel im Vergleich zu allen anderen CVE wenig vertreten zu sein und haben ihr größtes Aufkommen von 2016 bis 2019. Die restlichen Jahre schwankt ihr Aufkommen jeweils zwischen 0 und 5 CVE pro Jahr. In Abbildung 4.24b sind die über die Jahre kumulierten CVE-Anzahlen dargestellt. Die Anzahl der CVE scheint rapide zu steigen. Es könnte sich optisch um exponentielles Wachstum handeln. In den Kurven der CVE mit

[34] `https://github.com/torvalds/linux`

CVSS-Score von mindestens fünf bis mindestens neun, ist ein Wendepunkt um das Jahr 2014 herum erkennbar. Ab diesem Zeitpunkt wird der Anstieg dieser CVE-Kategorien stärker.



(a) CVE Linux Kernel

(b) CVE Linux Kernel kumulativ

Abbildung 4.24: Entwicklung CVE Linux Kernel

Um die Art der Entwicklung der CVE-Anzahl in Linux Kernel zu untersuchen, wurde ein Log-Plot über die Differenzen der kumulierten CVE für jedes Jahr erstellt. Der Log-Plot ist in Abbildung 4.25 dargestellt. Die Punkte bilden mit etwas Streuung eine Gerade mit positivem Anstieg. Dies kann als technischer Indikator gewertet werden, dass die Entwicklung der CVE im Linux Kernel mit hinreichender Wahrscheinlichkeit einen exponentiellen Trend verfolgt.

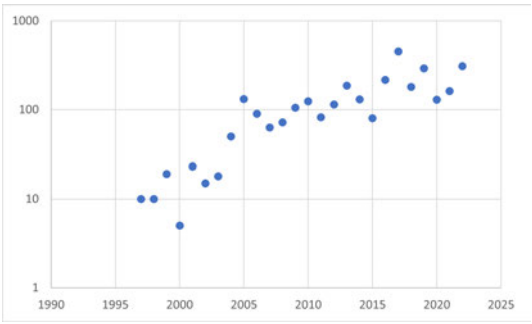


Abbildung 4.25: Log-Plot CVE Linux Kernel

LoC Entwicklung Linux Kernel Abbildung 4.26 zeigt die Entwicklung der LoC der Versionen von Linux Kernel. Die erste auf GitHUb öffentlich verfügbare Version stammt von 2005. Die LoC von Linux Kernel werden dahingehend von 2005 bis 2022 betrachtet. Die LoC-Entwicklung weist einen starken stetigen Anstieg auf.

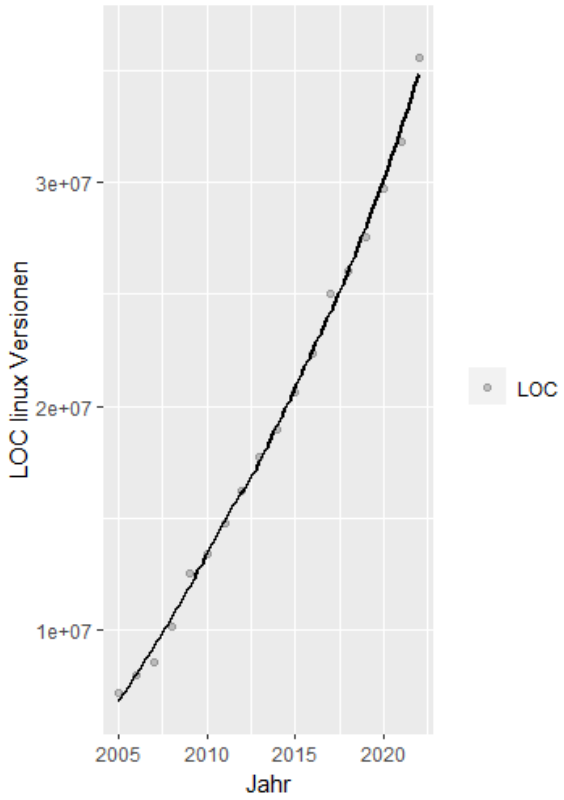


Abbildung 4.26: Entwicklung LoC Linux Kernel

Um die Art des Wachstums der LoC von Linux Kernel zu untersuchen, wurde ein Log-Plot über die Differenzen der letzten verzeichneten LoC jedes Jahres erstellt. Der Log-Plot ist in Abbildung 4.27 abgebildet. Die aufgezeigten Punkte bilden dabei mit etwas Streuung eine Gerade mit positivem Anstieg. Dies kann als technischer Indikator gewertet werden, dass die LoC-Entwicklung in Linux Kernel mit hinreichender Wahrscheinlichkeit einen exponentiellen Trend verfolgt.

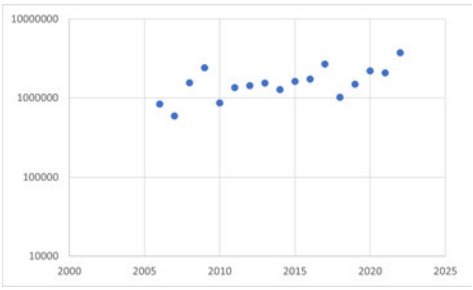


Abbildung 4.27: Log-Plot LoC Linux Kernel

Gegenüberstellung von CVE und LoC von Linux Kernel In Abbildung 4.28 wird die Entwicklung der LoC von Linux Kernel der Entwicklung der CVE (kumuliert) gegenübergestellt. Dabei fällt eine deutliche optische Ähnlichkeit der Graphen der LoC- und CVE-Entwicklung auf. Zudem ist die Anzahl der CVE mit CVSS-Score von mindestens neun so gering, dass sie in der Abbildung nicht dargestellt werden können. Auch die Anzahl der CVE mit CVSS-Score von mindestens fünf bzw. mindestens sieben können erst ab 2007 bzw. 2009 sinnvoll abgebildet werden.

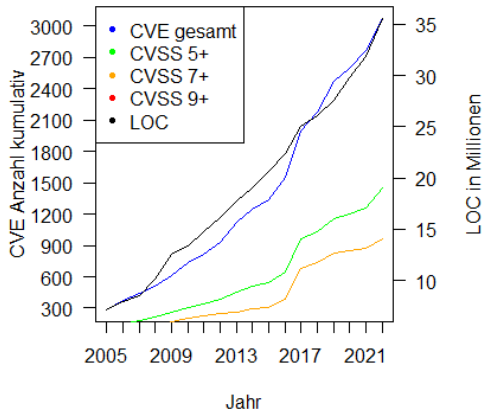
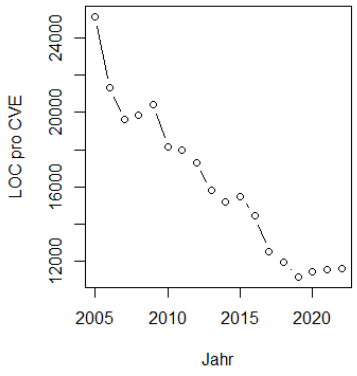


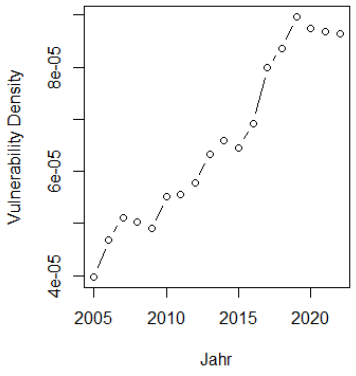
Abbildung 4.28: Vergleich der CVE- und LoC-Entwicklung von Linux Kernel

Um die optische Ähnlichkeit der Entwicklung von LoC und CVE in Linux Kernel zu untersuchen, wird ein Korrelationstest nach der Methode „Kendall“ durchgeführt. Das Ergebnis dieses Tests für τ ist 1. Somit existiert nachweislich eine Korrelation zwischen der Entwicklung der LoC und der Entwicklung der CVE.

Normalisierung von LoC und CVE In Abbildung 4.29 werden die LoC- und CVE-Zahlen von 2005 bis 2022 normalisiert dargestellt. Abbildung 4.29a zeigt die Entwicklung der LoC, die im Code von Linux Kernel auf eine CVE kommen. Der Trend der LoC pro CVE scheint seit 2005 bis auf kleine lokale Maxima 2009 und 2015 von annähernd 25000 LoC pro CVE auf rund 12000 LoC pro CVE 2020 zu sinken. Seit 2020 bleibt der normalisierte Wert annähernd gleich. Der sinkende Trend von LoC pro CVE deuetet auf einen höher werdenden Anteil an CVE im Code von Linux Kernel hin. Der Trend zeigt sich auch in der *Vulnerability Density*-Entwicklung des Codes von Linux Kernel. Die *Vulnerability Density* steigt bs 2020 konstant. Ab 2020 scheint sie dagegen leicht abzusinken. Somit zeigt sich, dass der Anteil der CVE im Code von Linux Kernel über die Jahre hinweg zugenommen hat, seit 2020 allerdings erstmals leicht zu sinken scheint.



(a) Normalisierung LoC/CVE



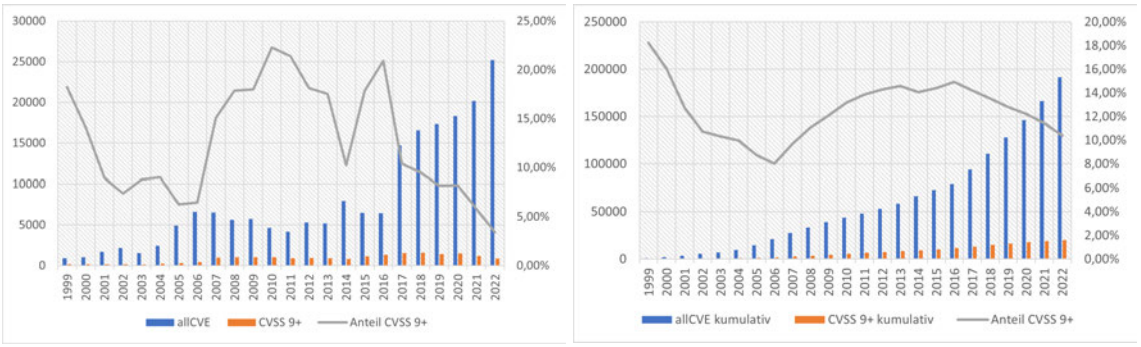
(b) Vulnerability Density (CVE/LoC)

Abbildung 4.29: Normalisierter Vergleich Linux Kernel

4.3 Verhalten der CVE mit kritischem Schweregrad

Für einen weiterführenden Einblick in die Entwicklung der CVE wird die Entwicklung der kritisch eingestuften CVE (CVSS-Score $\geq$9) der Entwicklung aller entdeckten CVE gegenübergestellt. Dieser Vergleich wird mit Hilfe von Excel visualisiert. Es ist anzumerken, dass noch nicht allen CVE aus 2022 CVSS-Scores zugeordnet wurden.

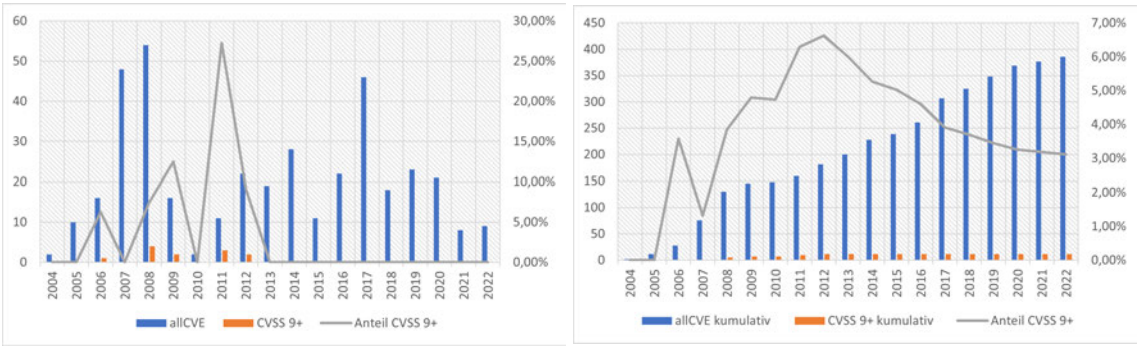
.. in Bezug auf die Gesamt-CVE Abbildung 4.30 zeigt den Anteil der CVE mit CVSS-Score von mindestens neun im Vergleich zu der Entwicklung der Gesamt-CVE in Prozent von 1999 bis 2022. In Abbildung 4.30a, welche auf den nicht kumulierten CVE-Entdeckungszahlen basiert, sinkt der Anteil erst von 1999 bis 2005 und anschließend erneut von 2016 bis 2022. 2022 erreicht der Anteil dabei mit rund 4% seinen gegenwärtig tiefsten Punkt. Von 2006 bis 2017 werden hohe Schwankungen und vermehrt höhere Anteile verzeichnet, wobei 2010 mit rund 22% das globale Maximum und 2014 mit annähernd 12% ein lokales Minimum erreicht wird. 2016 wird vor dem erneuten Sinken des Anteils mit rund 20% ein weiteres lokales Maximum erreicht. Abbildung 4.30b zeigt einen ähnlichen Trend. Der Anteil der CVE mit kritischem Schweregrad ist 1999 am höchsten und sinkt anschließend bis 2006 ab. 2006 bis 2016 steigt der Anteil erneut auf ein lokales Maximum von rund 15.5%. Nachdem dieser Hochpunkt erreicht ist, beginnt der Anteil ein weiteres Mal abzusinken. Der Verlauf ist dabei stetig und 2022 liegt der Anteil der kritisch eingestuften CVE nur noch bei rund 10%.



(a) Anteil der CVE mit CVSS 9+ **(b)** Anteil der kumulierten CVE mit CVSS 9+

Abbildung 4.30: CVSS 9+ Anteil

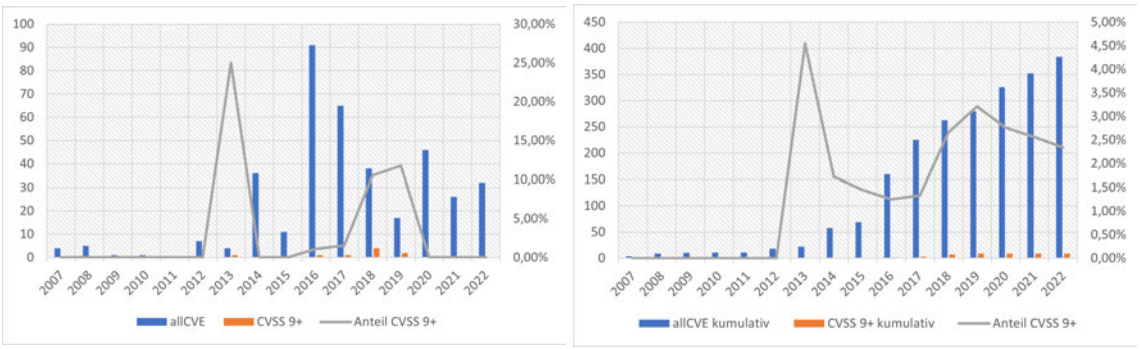
.. in Bezug auf die WordPress-CVE Abbildung 4.31 visualisiert den prozentualen Anteil der in WordPress entdeckten CVE mit CVSS-Score von mindestens neun von 2004 bis 2022. 4.31a zeigt dabei den Verlauf des Anteils auf den nicht kumulierten Daten, wohingegen 4.31b den Verlauf des Anteils auf den kumulierten Daten der CVE-Entwicklung darlegt. In Abbildung 4.31a wird, nach zwei Ausschlägen des Anteils in 2006 und 2009, 2011 der größte Anteil der kritisch eingestuften CVE mit rund 27% verzeichnet. Nach diesem globalen Maximum sinkt der Anteil rasant. Ab 2013 beträgt der Anteil der CVE mit CVSS-Score von mindestens neun in WordPress 0%. In 2013 und den darauf folgenden Jahren wurden somit keine CVE mit CVSS-Score von mindestens neun verzeichnet. Dieser Trend bildet sich auch in Abbildung 4.31b ab. Auch in dieser Abbildung zeigen sich 2006 und 2009 Ausschläge. Der Höhepunkt mit knapp 7% der Gesamt-CVE wird 2012 erreicht. Ab diesem Zeitpunkt sinkt der Anteil der kritisch eingestuften CVE in WordPress. 2022 beträgt der Anteil rund 3%.



(a) Anteil der CVE mit CVSS 9+ **(b)** Anteil der kumulierten CVE mit CVSS 9+

Abbildung 4.31: WordPress CVSS 9+ Anteil

.. in Bezug auf die QEMU-CVE Abbildung 4.32 stellt die Entwicklung des prozentualen Anteils der CVE mit CVSS-Score von mindestens neun in QEMU von 2007 bis 2022 dar. 4.32a zeigt den Anteil unter Betrachtung der nicht kumulierten Daten, während 4.32b den Anteil unter Betrachtung der kumulierten Daten darlegt. In Abbildung 4.32a ist zu beobachten, dass der Anteil der CVE mit CVSS-Score von mindestens neun über viele Jahre hinweg bei 0% bleibt. Ausschläge gibt es in den Jahren 2013, 2016, 2017, 2018 und 2019. Das globale Maximum des Funktionsgraphen wird 2013 mit rund 25% erreicht. Die in 2018 und 2019 folgenden Ausschläge liegen zwischen 10 und 15%. Ab 2020 wird erneut ein Anteil von 0% verzeichnet. In Abbildung 4.32b werden die oben genannten Ausschläge erneut deutlich. Nach dem globalen Maximum 2013 ist ein starker Abfall des Anteils bis zum lokalen Maximum 2016 zu beobachten. Nach seinem letzten Ausschlag 2019 fällt der Anteil bis zum letzten betrachteten Jahr 2022 stetig ab.

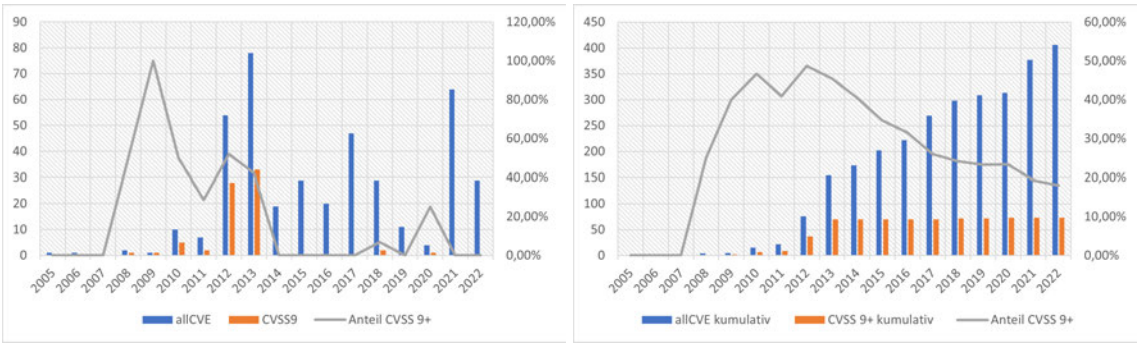


(a) Anteil der CVE mit CVSS 9+ **(b)** Anteil der kumulierten CVE mit CVSS 9+

Abbildung 4.32: QEMU CVSS 9+ Anteil

.. in Bezug auf die FFmpeg-CVE Abbildung 4.33 zeigt den prozentualen Anteil der CVE mit CVSS-Score von mindestens neun, die seit 2005 in FFmpeg entdeckt wurden. 4.33a zeigt den Anteil der CVE mit CVSS-Score von mindestens neun für die nicht kumulierten CVE-Daten von FFmpeg. 4.33b zeigt den Anteil bezogen auf die kumulierten CVE-Daten. In Abbildung 4.33a liegt der Anteil der kritisch eingestuften CVE von 2005 bis 2007 bei 0%. Ab 2008 steigt der Anteil rasant an und trifft 2009 mit rund 95% Anteil auf sein globales Maximum. Auf den Höhepunkt folgt anschließend ein starker Abfall des Anteils. Ausschläge des Abstiegs mit rund 50% Anteil sind 2012 und 2013. 2014 bis 2017 liegt der Anteil erneut bei 0%. Nachfolgend gab es 2018 und 2020 zwei lokale Maxima. Die restlichen Jahre bleibt der Anteil bei 0%. In Abbildung 4.33b ist der starke Anstieg

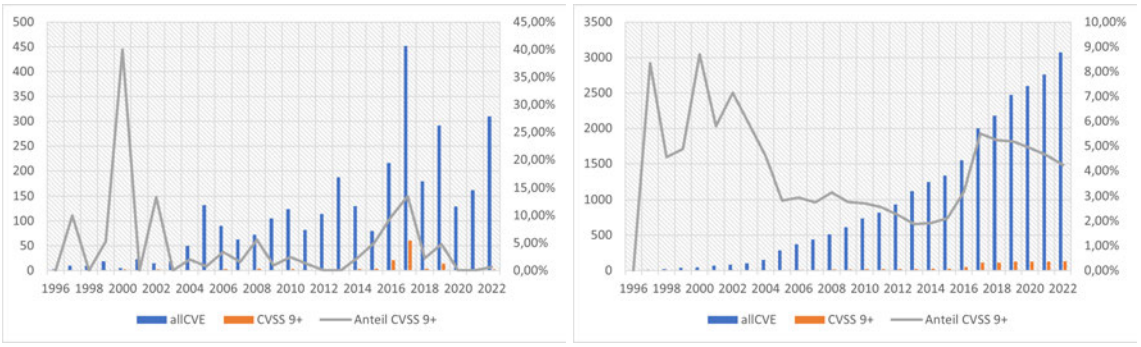
des Anteils ab 2008 wiederzufinden. 2012 wird dabei der Höhepunkt bei einem Anteil von rund 50% aller CVE erreicht. Während die kumulierte Anzahl der CVE mit CVSS-Score von mindestens neun ab 2013 jedes Jahr annähernd gleich bleibt, sinkt dabei ihr Anteil in den Gesamt-CVE stetig. 2021 sind nur noch rund 20% aller CVE kritisch eingestuft.



(a) Anteil der CVE mit CVSS 9+ **(b)** Anteil der kumulierten CVE mit CVSS 9+

Abbildung 4.33: FFmpeg CVSS 9+ Anteil

.. in Bezug auf Linux Kernel CVE Abbildung 4.34 zeigt den prozentualen Anteil der in Linux Kernel entdeckten CVE mit CVSS-Score von mindestens neun im Vergleich zu allen CVE des Kernels von 1996 bis 2022. Abbildung 4.34a zeigt den Anteil der kritisch eingestuften CVE auf den nicht kumulierten Entdeckungszahlen pro Jahr. Abbildung 4.34b zeigt den Anteil auf den kumulierten CVE-Daten. In Abbildung 4.34a wird im Jahr 2000 mit rund 40% das globale Maximum des Anteils erreicht. Danach scheint sich der Anteil, bis auf zwei vereinzelte lokale Maxima 2002 und 2017 mit 15%, auf 0 bis 6% einzupegeln. Seit 2020 liegt der Anteil der kritisch eingestuften CVE bei 0 bis 1%. In Abbildung 4.34b wird der beschriebene Trend in ähnlicher Weise abgebildet. Auch hier wird im Jahr 2000 das globale Maximum erreicht. Der Anteil liegt zu diesem Zeitpunkt bei rund 8.7%. 1997 wird dabei mit rund 8.3% das erste lokale Maximum aufgezeichnet. Nach einem Abfall des Anteils 2001, steigt der Anteil 2002 wieder kurzzeitig auf annähernd 7% an. Nachfolgend ist ein abfallender Trend im Anteil zu erkennen. Erst 2016 steigt der Anteil erneut merklich. Dieser Anstieg hat im Folgejahr 2017 mit rund 5.5% sein lokales Maxima erreicht. Seit diesem Hochpunkt sinkt der Anteil stetig. 2022 wurde ein Anteil von annähernd 4% verzeichnet.



(a) Anteil der CVE mit CVSS 9+ **(b)** Anteil der kumulierten CVE mit CVSS 9+

Abbildung 4.34: Linux CVSS 9+ Anteil

5 Diskussion

Nachfolgend werden die Ergebnisse ausgewertet und umfassend diskutiert. Dabei werden Problemfelder beschrieben, die während der Bearbeitung aufgekommen sind und in zukünftigen Forschungsarbeiten weiter betrachtet werden können.

5.1 CVE-Entwicklung

In Abschnitt 4.1 wurde die Entwicklung der CVE von 1999 bis 2022 betrachtet. Im Fokus lag dabei, neben den konkreten Zahlen, die Art der Entwicklung der Anzahl der CVE über die Jahre hinweg. Betrachtet wurde dabei einerseits die Entwicklung der Anzahl der CVE-Entdeckungen pro Jahr, andererseits die Entwicklung der kumulierten Anzahl existenter CVE bis Ende jedes Jahres.

Entdeckungen der einzelnen Jahre Bei Betrachtung der Anzahl der Entdeckungen von CVE jedes Jahr, wie sie in Abbildung 4.1 dargestellt sind, fällt auf, dass die Entdeckungen bis 2007 steigen, dann abfallen und anschließend ab 2013 stetig stark zunehmen. Dabei folgt die Entwicklung der Anzahl der Entdeckungen von CVE, basierend auf dem Log-Plot ihrer Differenzen, annähernd einem exponentiellen Trend. Auch das CISQ beobachtet, dass die Cybersicherheits-Vorfälle seit „einem Jahrzehnt steigen“ [14]. Gleichsam steigen auch die Verluste aufgrund ausgenutzter Sicherheitslücken. Von 2020 auf 2021 wurde ein Verlust-Anstieg von 64% verzeichnet [14].

Es stellt sich die Frage, was den Abfall der Entdeckungen von 2007 bis 2011 bzw. das rasante Steigen der Entdeckungen der CVE seit 2013 beeinflusst hat.

Die Entwicklung der Entdeckungen von CVE spiegelt tendenziell eher einen gesellschaftlichen Prozess als ein naturwissenschaftliches Phänomen wieder. Bei der Beantwortung der Frage sollte somit der Fokus auf den Zeitpunkten liegen, die die Gesellschaft und folglich den Verlauf der Sicherheitslücken-Entwicklung beeinflusst haben.

Eine mögliche Erklärung für den Abfall der Zahlen ab 2007 ist die beginnende Finanzkrise. Finanzturbulenzen am US-Hypothekenmarkt Mitte 2007 entwickelten sich 2008-2009 zu einer globalen Finanzkrise, welche in den darauffolgenden Jahren in einer Schuldenkrise überlief [56]. Der Zeitraum der Finanzkrise und ihrer direkten Auswirkungen überschneidet sich somit mit dem Zeitraum, indem die Anzahl der Entdeckungen von CVE sinkt und kommt damit als möglicher Einfluss in Frage. Auch die größten gemessenen Differenzen in den Anzahl an Entdeckungen zwischen den Jahren 2005, 2014, 2017 und 2022 könnten durch gesellschaftliche Ereignisse beeinflusst worden sein.

Die Relation der Entwicklung der CVE zu politischen und gesellschaftlich relevanten Ereignissen sollte in weiteren Forschungsvorhaben eingehender betrachtet werden.

Die vermehrte Nutzung von Open Source Komponenten in Software könnte einen weiteren Einflussfaktor für den starken Anstieg an entdeckten CVE in den letzten Jahren darstellen. Wie in Abschnitt 2.6.1 beschrieben, nutzen 96% der für den Bericht untersuchten Code-Basen Open Source Komponenten [39]. Sicherheits-Vorfälle aufgrund von Sicherheitslücken in den Software-Systemen zugrundeliegenden Open Source-Komponenten von Drittanbietern sind dabei von 2020 auf 2021 erheblich um 650% gestiegen [14]. Da Open Source Komponenten weitreichend verwendet werden, hat ihre Existenz einen besonders drastischen Einfluss auf die Sicherheit von Software.

Neben den Open Source-Komponenten sollte auch die gegenwärtige Software-Qualität in ihrer Gesamtheit im Fokus stehen. Laut des CISQ ist die Qualität von Software derzeitig schlecht und stellt dadurch ein Problem mit hohem Ausmaß dar [14]. Seit 2018 analysiert das CISQ aller zwei Jahre die Software bezüglich ihrer Qualität in den USA. Sie stellten dabei fest, dass sich die Qualität von Software seit 2020 nicht verbessert hat [14]. Obwohl im Vergleich zu den vorherigen Jahren immer mehr Anwendungssicherheitstests durchgeführt werden [14, nach VERACODE „State of Software Security Volume 12“], deute sich sogar eine Verschlechterung ab.

Das CISQ sieht einen Zusammenhang zwischen Zunahme der Anzahl, Größe und Komplexität von Software-Systemen und dem Anstieg an Sicherheitslücken [14]. Somit ist ein anderer möglicher Faktor, der den Trend der CVE-Entwicklung (mit) beeinflusst haben könnte, die Entwicklung der LoC von Software. Der Zusammenhang zwischen CVE- und LoC-Entwicklung wird im Abschnitt 5.2 weiterführend diskutiert.

Kumulierte Entwicklung Werden die CVE für jedes Jahr mit den Entdeckungen der vorherigen Jahre, wie in Abbildung 4.4 dargestellt, kumuliert betrachtet, ergibt sich die Entwicklung der vorhandenen CVE jedes Jahres von 1999 bis 2022. Die Kurve, die aus dieser Entwicklung über den Zeitraum entsteht, zeigt einen rapiden Anstieg. Der auf den Differenzen der kumulierten Daten basierende Log-Plot kann als Indikator gesehen werden, dass dieser rapide Anstieg mit hinreichender Wahrscheinlichkeit einen exponentiellen Trend verfolgt.

These I Die Anzahl der IT-Sicherheitsücken steigt exponentiell. Das ist die erste Hypothese, welche diese Arbeit betrachtet. Nachfolgend wird diskutiert, ob diese Hypothese anhand der dargestellten Ergebnisse angenommen oder abgelehnt wird.

Die Hypothese basiert auf der Entwicklung der Sicherheitslücken über die Jahre bis in die Gegenwart. Da es keine absoluten Zahlen gibt, wie viele Sicherheitslücken insgesamt existieren, wird auf die CVE-Datenbank zugegriffen. In dieser sind die Sicherheitslücken aufgeführt, die entdeckt, gemeldet und anschließend als CVE veröffentlicht wurden.

Für die Auswertung der Entwicklung der Sicherheitslücken wurde, wie oben aufgeführt, zwischen jährlich entdeckten und insgesamt existenten CVE unterschieden. Die Entwicklung der Anzahl der Entdeckungen von CVE jedes Jahres ist annähernd exponentiell. Zur Betrachtung der existenten CVE wird die Anzahl der in einem Jahr entdeckten CVE mit den in den vorherigen Jahren entdeckten CVE kumuliert. Die Entwicklung der in jedem Jahr existierenden CVE verfolgt einen exponentiellen Trend.

Daraus folgt, dass die Hypothese von der Definition der „Anzahl der IT-Sicherheitslücken“ abhängig ist. Die Menge der kumulierten CVE beinhaltet alle CVE, die bis zu den jeweiligen Zeitpunkten entdeckt wurden. Es werden somit alle CVE betrachtet, deren Existenz zum jeweiligen Zeitpunkt bekannt ist. Werden die CVE nicht kumuliert betrachtet, wird die Anzahl der jedes Jahr entdeckten CVE erfasst. Es wird somit ein Überblick über die Entwicklung der Entdeckungsraten ermöglicht.

Abschnitt 2.2.2 beschreibt, dass die Betrachtung von Sicherheitslücken endet, wenn ein Patch existiert und installiert ist [12]. Die Veröffentlichung eines Patches ist dabei nicht zwangsläufig im Jahr der Entdeckung einer Sicherheitslücke, weshalb die Sicherheitslücke auch in der darauf folgenden Zeit eine Rolle spielt [19]. In [57] wurden diesbezüglich Bibliotheken betrachtet, welche Sicherheitslücken aufweisen. 30% der Sicherheitslücken dieser Bibliotheken wurden innerhalb von zwei Jahren nicht behoben. Abschnitt 2.3.2 legt darüber hinaus dar, dass die Existenz eines Patches nicht zwangsläufig seine Installation auf allen betroffenen Systemen zur Folge hat, wodurch die Gefahr der Sicherheitslücke für ungepatchte Systeme vorhanden bleibt [8]. 91% der in [39] untersuchten Code-Basen enthielten veraltete Software-Komponenten, für welche Updates oder Patches existieren, diese aber nicht angewendet werden. Sicherheitslücken können somit auch nach der Veröffentlichung ihres Patches ein Risiko darstellen. CVE nur in ihrem Entdeckungsjahr zu betrachten und in den folgenden Jahren außer Acht zu lassen, ist dementsprechend nicht zielführend. Werden die CVE hingegen über die Jahre kumuliert, werden alle existenten CVE auch nach ihrem Entdeckungsjahr betrachtet.

In dieser Arbeit wird die Anzahl der kumulierten CVE somit für die Entwicklung der „Anzahl der IT-Sicherheitslücken“ genutzt, um alle existenten Sicherheitslücken und somit Risiken einzuschließen. Da die Entwicklung der kumulierten Anzahl an CVE einen exponentiellem Trend folgt, wird Hypothese I angenommen.

In einem weiteren Forschungsvorhaben wäre dessen ungeachtet die Betrachtung von CVEs ausschließlich in der „Lebensdauer ihrer Code-Basis“ [19, S. 2207] interessant, um festzustellen, inwieweit dies die in dieser Arbeit dargestellte CVE-Entwicklung potenziell verändert. Die „Lebensdauer der Code-Basis“ beschreibt dabei die Zeitspanne von der Entdeckung einer Sicherheitslücke bis hin zu ihrem Patch. Bei Ubuntu Kernel CVE beträgt diese Lebensdauer beispielsweise durchschnittlich fünf Jahre [19]. Eine umfängliche Grundlage für die Analyse von CVE und ihren Patches wird in [19] dargelegt.

Limitation Der Nachweis des Trends, den die Entwicklung der CVE verfolgt, ist für die Annahme der Hypothese maßgeblich. Als Methode für den Nachweis der Entwicklungsart wurde die Erstellung von Log-Plots basierend auf der Ableitung der CVE-Entwicklung gewählt. Durch die Log-Plots ist eine optische Einschätzung des Trends möglich. Die aus der Auswertung der Log-Plots abgeleiteten Ergebnisse können allerdings nur als technische Indikatoren betrachtet werden und stellen keinen absoluten Nachweis dar. Da gegenwärtig keine Methode einen absoluten Beweis erbringt, ist die Nutzung solcher Indikatoren allerdings weit verbreitet. In weiteren Forschungsvorhaben könnten die Ergebnisse, die von den Log-Plots abgeleitet wurden, mit weiteren Methoden gegengeprüft werden.

5.2 CVE- und LoC-Entwicklung in Software

In Abschnitt 4.2 wurde, im Zuge der Untersuchung eines möglichen Zusammenhanges zwischen CVE- und LoC-Entwicklung, die CVE- und LoC-Entwicklung der Software-Systeme WordPress, QEMU, FFmpeg und Linux Kernel betrachtet.

5.2.1 CVE-Entwicklung der Software-Systeme

Die erste für den Vergleich durchgeführte Analyse ergab, dass sich die Entwicklung der CVE in WordPress (Abb. 4.6a) und QEMU (Abb. 4.12a) in dem betrachteten Zeitraum einem einem linearen Trend annähert. Der Anstieg der in FFmpeg existenten CVE deutet hingegen eine anfangende Stagnation an (Abb. 4.18a). Die Entwicklung der Anzahl der CVE in Linux scheint einen exponentiellen Trend zu verfolgen (Abb. 4.24a).

Die in den einzelnen Software-Systemen existenten CVE verfolgen somit im Gegensatz zur Entwicklung aller CVE über die vergangenen Jahre in der Mehrheit keinen exponentiellen Trend in der Entwicklung.

Software-Systeme Werden die Unterschiede der CVE-Entwicklung der einzelnen Systeme zur Gesamtheit betrachtet, scheint das exponentielle Wachstum der gesamt existenten CVE nicht ausschließlich durch den Anstieg der CVE in einzelnen Software-Systemen begründbar. Da aber auch die Zunahme der Anzahl an Software-Systemen den Anstieg von Sicherheitslücken-Aufkommen beeinflussen kann [14], ist es möglich, dass der Anstieg der Entdeckungen damit zu tun hat, dass es einen überdurchschnittlichen Anstieg an Software-Systemen gab, welcher zu größerer Analysegrundlage und somit mehr entdeckten CVE führte. Da die CVE der einzelnen Software-Systeme großteils nicht exponentiell ansteigen, müsste, der Überlegung folgend, die Anzahl der Software-Systeme exponentiell steigen, um einen exponentiellen Anstieg der insgesamt entdeckten CVE zu erklären.

Limitation Um diese These abschließend diskutieren zu können, fehlt es an Daten zur Veröffentlichungsanzahl von Software-Systemen pro Jahr. Zur Bewertung wird ein Datensatz benötigt, welcher die jährlichen Veröffentlichungen von Software-Systemen erfasst. Auf Basis solch eines Datensatzes wäre die Betrachtung der These „Die Anzahl der Sicherheitslücken steigt exponentiell, weil die Anzahl der Software-Systeme exponentiell steigt“ von großer Relevanz und könnte als Ausgangspunkt für weitere Forschungsvorhaben dienen.

In der zugrundeliegenden Arbeit wird davon ausgegangen, dass die vier betrachteten Software-Systeme repräsentativ für alle Software-Systeme sind. Sollten sie einen anderen Trend abbilden, als die Mehrheit aller Software-Systeme, müssten die Thesen neu bewertet werden.

5.2.2 LOC-Entwicklung der Software-Systeme

Die zweite Analyse ergab, dass die Entwicklung der LoC von WordPress (Abb. 4.8) und QEMU (Abb. 4.14) einem linearen Trend folgt. Bei FFmpeg (Abb. 4.20) scheint der Anstieg hingegen langsam zu stagnieren. Die Entwicklung der CVE in Linux Kernel (Abb. 4.26) verfolgt einen exponentiellen Trend.

These II Die Anzahl der IT-Sicherheitslücken steigt exponentiell, weil die Code-Länge exponentiell wächst.

Um diese zweite aufgestellte These annehmen oder ablehnen zu können, muss sie als erstes granuliert betrachtet werden. Die erste Voraussetzung der These ist das exponentielle Wachstum der CVE-Anzahl. Die zweite Voraussetzung ist das exponentielle Wachstum der Code-Länge. Die dritte Voraussetzung ist die Kausalität zwischen den beiden exponentiellen Trends. Diese Kausalität wird zusätzlich in These III der Arbeit abgebildet.

In Abschnitt 5.1 wurde These I angenommen, welche aussagt, dass die Anzahl der IT-Sicherheitslücken exponentiell steigt. Somit ist die erste Voraussetzung der These erfüllt.

Um die zweite Voraussetzung zu überprüfen, wurde, wie oben beschrieben, die LoC-Entwicklung analysiert. Die LoC-Entwicklungen der betrachteten Software-Systeme verfolgt dabei in Mehrheit einen linearen Trend. Zudem wird ein exponentieller Trend sowie ein Trend ähnlich einer Sättigungsfunktion verzeichnet.

Der bei Linux Kernel nachgewiesene exponentielle Trend kann aufgrund der Mehrheit von maximal linearem Wachstum nicht als repräsentativ angesehen werden. Daraus ist zu schließen, dass die Code-Länge der gewählten Software-Systeme zum großen Teil nicht exponentiell wächst. Die zweite Voraussetzung der These ist damit nicht erfüllt und die These, dass die Anzahl der IT-Sicherheitslücken exponentiell wächst, weil die Code-Länge exponentiell wächst, wird abgelehnt.

Zwischen der LoC-Entwicklung von Software-Systemen und der Entwicklung der existenten CVE konnte somit kein Zusammenhang bezüglich des exponentiellen Wachstums nachgewiesen werden. Die Ergebnisse der LoC-Entwicklung korrelieren jedoch mit der CVE-Entwicklung des jeweiligen Software-Systems. So verfolgt beispielsweise sowohl die CVE-Entwicklung als auch die LoC-Entwicklung von WordPress einen linearen Trend. Folglich scheint zwischen CVE-Entwicklung und LoC-Entwicklung eines Software-Systemes ein Zusammenhang zu bestehen. Dieser Zusammenhang von CVE und LoC innerhalb eines Software-Systems wird im folgenden Abschnitt 5.2.3 genauer betrachtet.

5.2.3 Vergleich von CVE und LoC Entwicklung

Um einen möglichen Zusammenhang zwischen CVE- und LoC-Entwicklung von Software-Systemen zu überprüfen, wurden beide Entwicklungen jedes betrachteten Software-Systems einander vergleichend gegenübergestellt.

5.2.3.1 Korrelation

Die Entwicklungen der CVE und LoC wiesen bei ihrer Gegenüberstellung für jedes Software-System optisch eine Ähnlichkeit auf. Um diese Ähnlichkeit zu ergründen, wurde ein Korrelationtest über Kendalls τ durchgeführt.

Je näher τ an 1 ist, desto wahrscheinlicher ist die Korrelation der Stichproben. Der Korrelationstest ergab für WordPress, FFmpeg und Linux jeweils einen τ-Wert von 1. Bei QEMU lag τ bei 0.9944903.

These III Wenn die Entwicklung der Code-Länge und das Auftreten von Sicherheitslücken von Software über die Zeit korreliert, dann ist ein bestimmender Faktor für das Wachstum der Sicherheitslücken das Wachstum des Codes einer Software.

Diese dritte aufgestellte These beruht auf der Korrelation von Code-Länge und Sicherheitslücken. Die Code-Länge wird in dieser Arbeit mittels der LoC-Entwicklung repräsentiert. Die Sicherheitslücken werden durch die Entwicklung der existenten CVE vertreten. Um die These zu überprüfen, muss somit die Korrelation von CVE- und LoC-Entwicklung geprüft werden.

Im Zuge der Arbeit wurde, wie oben beschreiben, die Korrelation beider Entwicklungen getestet. Bei WordPress, QEMU und Linux wurde jeweils ein Korrelationskoeffizient von $\tau = 1$ berechnet werden. Bei FFmpeg liegt der τ-Wert mit 0.9944903 sehr nah an 1. Da eine Korrelation wahrscheinlicher ist, je näher der τ-Wert an 1 liegt, kann für jedes betrachtete Software-System eine Korrelation von CVE- und LoC-Entwicklung über die Zeit nachgewiesen werden. These III, welche besagt, dass die Entwicklung von Code-Länge und das Auftreten von Sicherheitslücken über die Zeit korreliert, wird somit angenommen. Die Entwicklung der Code-Länge kann folglich als ein bestimmender Faktor für die Entwicklung der Sicherheitslücken in einem Software-System betrachtet werden.

Dies bestätigt die Aussage des CISQ, dass es einen Zusammenhang zwischen Größe bzw. Komplexität eines Software-Systems und Aufkommen an Sicherheitslücken gibt [14].

5.2.3.2 Normalisierung

Um die Entwicklung von CVE und LoC unabhängig von der Korrelation gegenüberzustellen, wurden die Werte der betrachteten Jahre normalisiert. Dabei wurde zum einen die durchschnittliche Anzahl an LoC, die im Code auf eine CVE kommen, und zum anderen die *Vulnerability Density* der verschiedenen Software-Systeme betrachtet.

Die Normalisierung ergab, dass der Anteil an CVE im Code von WordPress sinkt (Abb. 4.11). Bei QEMU nimmt der Anteil erst seit seinem Höhepunkt 2019 ab (Abb. 4.17). Bei FFmpeg (Abb. 4.23) und Linux (Abb. 4.29) steigt der Anteil hingegen stetig.

Alle Software-Systeme wurden über einen ähnlichen Zeitraum betrachtet. Trotzdem weist die Entwicklung ihres CVE-Anteils unterschiedliche Trends auf.

Betrachtung der Hersteller In Abschnitt 2.6.1 wurde erfasst, dass die Sicherheit von Software in großen Teilen davon abhängig ist, wie viel unternommen wird, um die Software sicherer zu gestalten [34]. Dies könnte sich auf die oben genannten Ergebnisse insoweit übertragen lassen, dass die Trends der CVE-Anteile in kausalem Zusammenhang mit dem Sicherheitsbemühen der jeweiligen Hersteller stehen. Dem Gedanken folgend, müssten die Entwickler von WordPress und QEMU mehr auf die anfallenden Sicherheitsrisiken achten, als die Entwickler von FFmpeg und Linux. Da es sich allerdings bei allen betrachteten Software-Systemen um Open Source Projekte handelt, können alle Nutzer helfen, die Software sicherer zu machen [34]. Der Unterschied in den CVE-Anteilen kann somit nicht ausschließlich von den Maßnahmen der einzelnen Produktverantwortlichen stammen. WordPress und QEMU müssten, um die Unterschiede der Entwicklung der Anteile zwischen den Software-Systemen zu erklären, eine aktivere Community als FFmpeg und Linux haben.

„Insights“ auf Github gibt dahingehend einen Einblick in die Aktivität der Community von WordPress[35], QEMU[36], FFmpeg[37] und Linux Kernel[38].

Bei Betrachtung des GitHubs der Software-Systeme in der Zeitspanne eines Monats, konkret vom 31. Juli 2023 bis 31. August 2023, wurden folgende Aktivitäten verzeichnet (vgl. Tabelle 5.1). Bei WordPress haben 25 verschiedene Autoren insgesamt 172 Commits veröffentlicht und 61001 Änderungen durchgeführt. Änderungen beziehen sich auf das Hinzufügen oder Löschen von Code. Commits beschreiben Durchführungen von Codeänderungen beziehungsweise Fehlerbehebungen [58]. Bei QEMU haben 68 Autoren 409 Commits veröffentlicht und 11004 Änderungen durchgeführt. Bei FFmpeg haben 36 Autoren 211 Commits und insgesamt 13202 Änderungen veröffentlicht. Bei Linux Kernel haben 1053 Autoren 5729 Commits veröffentlicht und 483318 Änderungen durchgeführt.

Tabelle 5.1: GitHub-Aktivitäten in einem Monat

Community	Autoren	Commits	Änderungen
WordPress	25	172	61001
QEMU	68	409	11004
FFmpeg	36	211	13102
Linux Kernel	1053	5729	483318

Linux Kernel liegt somit in allen Aktivitätsbereichen weit vor den anderen GitHub-Communities. Davon abgesehen, hat QEMU mehr Autoren und Commits als FFmpeg und FFmpeg mehr als WordPress. Wird die Anzahl der Änderungen betrachtet, wurden in WordPress mehr Änderungen als in FFmpeg durchgeführt und in FFmpeg mehr Änderungen als in QEMU.

[35] `https://github.com/WordPress/WordPress/pulse/monthly`, Zugriff 31.08.2023
[36] `https://github.com/qemu/qemu/pulse/monthly`, Zugriff 31.08.2023
[37] `https://github.com/FFmpeg/FFmpeg/pulse/monthly`, Zugriff 31.08.2023
[38] `https://github.com/torvalds/linux/pulse/monthly`, Zugriff 31.08.2023

Laut [58] weisen „regelmäßig auftretende Commits [...] auf eine aktive Community-Unterstützung hin“ [58, S. 82]. Gemessen an der Anzahl der Commits, hat Linux die aktivste und WordPress die inaktivste Community. Dies spiegelt nicht den oben aufgestellten Zusammenhang zwischen Software-Sicherheit und Aktivität der Community wider.

Es scheint allerdings möglich, dass die Anzahl der Commits in großen Teilen von der Anzahl der Autoren abhängt. Weniger Autoren führen zu weniger Commits. Da WordPress von allen betrachteten Software-Systemen im letzten Monat die wenigsten Autoren hatte, kann sich daraus die geringere Anzahl an Commits erklären lassen. Wenige Autoren sprechen für eine kleinere Programmiercommunity. Dies muss aber im Umkehrschluss nicht weniger Aktivität bedeuten. Dies zeigt sich beim Vergleich der Anzahl der Änderungen. WordPress, mit den wenigsten Autoren in diesem Monat, hat die meisten Änderungen verzeichnet, wenn Linux Kernel außer Acht gelassen wird. Dies spricht dafür, dass die Community von WordPress zwar kleiner als die anderen, dafür aber aktiver ist. Dies könnte ein möglicher Einfluss auf den stetig sinkenden CVE-Anteil in WordPress im Verlauf des betrachteten Zeitraums sein. Die hohe Aktivität der Linux-Community erklärt hingegen den stetigen Anstieg des CVE-Anteils in Linux Kernel nicht. Auch der steigende CVE-Anteil von FFmpeg, welches bei allen Aktivitätsvergleichen mittig aufgestellt ist, lässt sich durch die Community-Aktivität nicht erklären.

An dieser Stelle ist auch der Gedanke zu beleuchten, dass eine hohe Autorenanzahl mehr Chance auf Manipulation lässt. Dem steht allerdings entgegen, dass auch bei Open Source Projekten Maßnahmen zur Qualitätssicherung existieren [34]. Nichtsdestotrotz steigt die Anzahl der CVE des Linux Kernel Projektes mit einer hohen Autorenanzahl, im Gegensatz zu der CVE-Anzahl von WordPress mit einer geringen Autorenanzahl. Weitere Studien bieten sich an.

Abschließend scheint die Aktivität der Community zur Sicherheit beitragen zu können, eine große sehr aktive Community verspricht im Umkehrschluss allerdings keine sichere Software.

Veröffentlichungsfrequenz Das CISQ beschreibt das vorzeitige Ausliefern als Grund der Verschlechterung von Software [14]. Dabei veröffentlichen Unternehmen um wettbewerbsfähig zu bleiben Versionen schneller, ohne die Fehlerquote so weit wie möglich reduziert zu haben. Um diesen möglichen Zusammenhang mit den CVE-Anteilen zu überprüfen, wird die Frequenz der Versionsveröffentlichung der einzelnen Software-Systeme sowie der Trend ihres CVE-Anteils betrachtet.

Der Anteil von CVE im Code von WordPress ist über die letzten Jahre gesunken. WordPress hat in dem betrachteten Zeitraum in den meisten Jahren mehrere Versionen veröffentlicht[39]. Zwischen den großen Versionswechseln (beispielsweise von Version 2.0 zu Version 3.0) liegen allerdings im Schnitt vier Jahre. Die lange Zeit zwischen den Versionswechseln könnte zu einer besseren Fehlerqouten-Reduzierung beigetragen haben. Die längere Zeit zwischen den großen Versionen kann somit möglicherweise mit dem Trend

[39] https://wordpress.org/documentation/article/learn-about-wordpress-and-version-history/

des CVE-Anteils in Zusammenhang gebracht werden.
QEMU veröffentlicht im Schnitt drei Versionen pro Jahr[40]. Zwischen den großen Versionswechseln liegen seit 2018 nur jeweils rund ein Jahr. Die Analyse des CVE-Anteils ergab ein Sinken des Anteils seit 2019. Bei QEMU scheint die kurze Zeit zwischen den großen Versionen somit keinen negativen Einfluss auf den CVE-Anteil zu haben. Es scheint somit bei QEMU kein Zusammenhang erkennbar.
FFmpeg veröffentlicht ebenfalls mehrere Versionen pro Jahr[41]. Zwischen den großen Versionswechseln liegen dabei meist zwei bis drei Jahre. Der Anteil der CVE im Code von QEMU nimmt allerdings seit Veröffentlichung zu. Die längere Zeit zwischen Versionen im Hinblicke auf das stete Steigen des CVE-Anteils spricht für keinen Zusammenhang zwischen langen Zeiträumen zwischen Versionswechseln und besserer Software-Qualität durch geringere Fehlerquote.
Für Linux Kernel werden von den betrachteten Software-Systemen mit fünf bis sechs die meisten Versionen pro Jahr veröffentlicht[42]. Während sich Version 2 für zehn Jahre gehalten hat vor einem Wechsel, passiert ein Versionenswechseln gegenwärtig aller zwei bis vier Jahre. Die Tendenz ist dabei sinkend. Die kurze Zeit zwischen den Versionen könnte das Steigen des Anteils von CVE in Code über den Betrachtungszeitraum hinweg erklären.

Aufgrund der unterschiedlichen Ergebnisse der Betrachtung des Zusammenhangs von Veröffentlichungsfrequenz von Versionen und Trend der CVE-Anteile, scheint kein direkter kausaler Zusammenhang zu bestehen. Eine Beeinflussung der CVE-Anteilsentwicklung durch die Veröffentlichungsfrequenz von Versionen ist trotzdem nicht auszuschließen und sollte weiterführend betrachtet werden.

Betrachtung der Software-Komponenten [39] stellt dar, dass viele Sicherheitslücken in Software von veralteten Komponenten kommen. 91% der in der Studie betrachteten Code-Basen enthielten dabei veraltete Komponenten. Es wird erklärt, dass diese hohe Anzahl daher kommt, dass die Entwickler nicht von der Sicherheitslücke in der Komponente bzw. der Existenz eines Updates oder Patches wissen. Es scheint eine Diskrepanz zwischen Risikobewusstsein der Entwickler und tatsächlichem Risiko, welches von der Nutzung der Drittanbieter-Komponenten ausgeht, zu geben [14]. Die Nutzung solcher Komponenten könnte zur Entwicklung der CVE-Anteile der Software-Systeme beitragen. In einem weiteren Forschungsvorhaben sollte überprüft werden, wie viele veraltete Komponenten in den betrachteten Software-Systemen sind und inwieweit ihr Einsatz die Trends der Entwicklung der CVE-Anteile beeinflusst.

Ein weiterer Gedanke zu der unterschiedlichen Entwicklung der CVE-Anteile ist die Herangehensweise der Entwickler. Könnte die Einstellung „Kann man eh nichts machen“ dazu führen, dass die Entwickler tatsächlich nichts machen? Ohne eine Studie mit einer repräsentativen Anzahl an Teilnehmer, lässt sich dieser Gedanke nicht verfolgen, wäre für die Zukunft allerdings ein denkbarer Ansatzpunkt für weiterführende Forschung.

[40] https://download.qemu.org/
[41] https://ffmpeg.org/index.html#news
[42] https://en.wikipedia.org/wiki/Linux_kernel_version_history

Betrachtung der Software-Größe In einem Bericht von CAST wird kein kausaler Zusammenhang zwischen Größe einer Software-Anwendung (im Vergleich zu anderen Anwendungen) und ihrer strukturellen Qualität entdeckt [14, nach CASTs „CRASH Report 2020“][43].

Die neuste Version von WordPress hat eine LoC-Anzahl von rund 1.5 Millionen. Die neuste Version von QEMU besteht aus rund 3.3 Millionen LoC. Die neuste Version von FFmpeg besteht aus rund 1.9 Millionen LoC. Linux hat mit annähernd 36.5 Millionen LoC in der neusten Version mit Abstand den meisten Code. QEMU ist am zweitgrößten, darauf folgt FFmpeg und an letzter Stelle ist WordPress. Der Trend des CVE-Anteils ist dahingegen bei WordPress am vorteilhaftesten. Danach folgt QEMU, während Linux und FFmpeg momentan den schlechtesten Trend bezüglich CVE-Anteilsentwicklung verfolgen.

Die Größe der betrachteten Software-Systeme steht damit in keinem Zusammenhang ihrer Qualität, was die Aussage aus dem oben genannten Bericht unterlegt. Die allgemeine Größe einer Software kann somit nicht als Anhaltspunkt für ihre Sicherheit und somit auch nicht als ein Grund für den Trend der CVE-Anteile genommen werden.

5.2.4 Limitation: Problemfeld Daten

Die verwendeten Daten bilden die Grundlage für getroffene Aussagen über die Entwicklung von Sicherheit und Sicherheitslücken.

Dabei gilt im Vorhinein, dass nicht zwangsweise alle Sicherheitslücken gefunden werden [13]. Keine Sicherheitslücken-Datenbank ist somit als vollständig zu betrachten [19]. Es gibt immer eine unbekannte Anzahl nicht entdeckter Sicherheitslücken, die folglich keine CVE-Kennung haben und auch in keiner Datenbank erscheinen. Abschnitt 2.3.2 beschreibt, dass Sicherheitslücken erst ein Risiko darstellen, wenn sie entdeckt sind, wodurch noch unbekannte Sicherheitslücken bis zu ihrer Entdeckung keinen Einfluss spielen. Es gibt keine Zahlen für nicht entdeckte Sicherheitslücken. Sie wurden in der Auswertung der CVE-Daten somit nicht mit betrachtet.

Bezüglich CVE In der verwendeten Sicherheitslücken-Datenbank finden sich die Daten zu allen gefundenen und gemeldeten Sicherheitslücken aufgezeichnet ab 1999 bis zum heutigen Zeitpunkt. Da 2023 gegenwärtig noch nicht abgeschlossen ist, liegen keine vollständigen Daten für das Jahr vor. Auf Grundlage dieser lückenhaften Daten, wurde das Jahr 2023 bei der Analyse der Entwicklung der CVE ausgelassen. Der Grund dafür ist eine potenzielle optische „Verfälschung“ der Statistik, da nur die Anzahl der bisher entdeckten und gemeldeten CVE aus der ersten Jahreshälfte von 2023 betrachtet und verglichen werden kann. Die Daten der zweiten Jahreshälfte fehlen. Ein anderer Ansatz hätte sein können, dass die unvollständigen Daten von 2023 mitbetrachtet werden und auf Grundlage dessen sowie der vorherigen Entwicklung Vorhersagen getroffen werden können, die Ende des Jahres überprüfbar wären. Dies ändert allerdings nichts an dem Problem der optisch dargestellten Entwicklung der CVE, die bis 2022 auf den jeweils gesamten entdeckten CVE eines Jahres beruhen.

[43] https://learn.castsoftware.com/crash-report_cast-research-on-application-software-health

Neben den unvollständigen Daten für 2023 ist zu beachtenden, dass für den Großteil der CVE von 2022 und 2023 keine CVSS-Scores vorliegen und den dabei betroffenen CVE als Platzhalter ein CVSS-Score von 0.0 zugeordnet wurde. Dies entwertet die Ergebnisse der Entwicklung der CVSS-Scores aus beiden Jahren und kann somit die Analyse der Entwicklung der Schweregrade von Sicherheitslücken beeinflussen. Da 2023 nicht betrachtet wird, kann das Jahr auch bei dieser Überlegung übergangen werden. Die CVE-Daten von 2022 wurden hingegen trotzdem betrachtet, da sie ansonsten vollständig bezüglich Gesamtanzahl der entdeckten CVEs sind und auch schon einige CVSS-Scores vorliegen. Es wäre interessant diese Daten nochmal aufzugreifen, wenn sie vollständig vorliegen und dabei auch die durch ihr Vorhandensein hervorgerufenen Veränderungen der Entwicklungen der Schweregrade zu betrachten.

Bezüglich LoC Da auf GitHub nicht alle Versionen der betrachteten Software-Systeme zur Verfügung stehen, konnte die LoC sowie der Vergleich zwischen Entwicklung LoC und Auftreten von CVE erst ab dem Veröffentlichungsdatum der ersten auf GitHub verfügbaren Version starten. Dies wurde so geregelt, da sich auf frei zugängliche Quellen beschränkt wurde. Hätten die anderen Versionen, wobei es sich hauptsächlich um die allerersten Versionen handelt, zur Verfügung gestanden, hätten diese ebenfalls analysiert werden können. Dadurch wäre ein weiter differenziertes Bild der Entwicklung der LoC entstanden.

5.2.5 Limitation: Software-Auswahl

Die vier betrachteten Software-Systeme wurden aufgrund ihres öffentlich verfügbaren Codes ausgewählt, welcher Analysen zulässt. Bei der Software handelt es sich somit ausschließlich um Open Source-Software. Die Ergebnisse sind somit nicht auf proprietäre Software übertragbar.

In Abschnitt 2.6.1 wurde dargelegt, dass der Code von proprietärer Software nicht öffentlich ist. Dies verhindert einer Nutzung solcher Software für Analysen, wie sie im Zuge dieser Arbeit durchgeführt werden. Da allerdings kein Software-Entwicklungsansatz dem anderen im Bereich der Sicherheit grundlegend über- bzw. unterlegen zu sein scheint [34], könnten durchaus Schlussfolgerungen für proprietäre Software aus den Analysen der betrachteten Open Source-Software gezogen werden, auch wenn sie bei der Analyse nicht berücksichtigt wurde.

Die Aussagekraft der Ergebnisse der Analysen der repräsentativ gewählten Software-Systeme sollte in weiteren Forschungsvorhaben unter Betrachtung und Analyse von anderen Software-Systemen geprüft werden.

5.3 Entwicklung der CVE mit hohem Schweregrad

Mittels Gegenüberstellung der Entwicklung der gesamten entdeckten CVE-Anzahl und der Entwicklung der Anzahl der CVE mit CVSS-Score von mindestens neun, wurde das Verhalten der CVE mit kritischem Schweregrad analysiert.

Bei allen betrachteten Software-Systemen sowie bei der Betrachtung der gesamten CVE-Entdeckungen ist der Trend zu beobachten, dass der Anteil der CVE mit kritischem Schweregrad im Vergleich zu der Anzahl aller entdeckten CVE über die Zeit sinkt. Die erreichten Höhepunkte der Anteile bei den Software-Systemen sind zu verschiedenen Zeitpunken. In den letzten Jahren wurden kaum bis keine kritisch eingestuften CVE innerhalb der vier betrachteten Software-Systeme verzeichnet.

Es ist aus den analysierten Daten zu schließen, dass der Anteil kritisch eingestufter CVE abnimmt. Zu einem ähnlichen Ergebnis kam auch [39], welche auf Seite 7 des Berichtes den prozentualen Anteil von hoch-Risiko Sicherheitslücken von 2018 bis 2022 betrachtet. Der Anteil der hoch-risikobehaftete Sicherheitslücken in den untersuchten Code-Basen stieg ab 2018 und erreichte 2020 mit 60% seinen Höhepunkt. 2021 sank der Anteil auf 50%. 2022 fiel der Anteil weiter auf 48%. Hoch-risikobehaftete Sicherheitslücken umfassen dabei Sicherheitslücken, die „bereits aktiv ausgenutzt wurden, für die es bereits dokumentierte Proof-of-Concept-Exploits gibt oder die als Sicherheitslücken für die Remotecodeausführung eingestuft sind“ [39].

Da die Höhepunkte der Anteile kritischer CVE zu unterschiedlichen Zeitpunkten sind, scheint diesbezüglich kein Zusammenhang erkennbar. Dies wirft wiederum die Frage auf, warum der Anteil der CVE mit kritischem Schweregrad über die letzten Jahre geschlossen sinkt.

a) besserer Code Ein Grund für die die Abnahme des Anteils der CVE mit kritischem Schweregrad ist die Zunahme der Sicherheit von Software und Software-Code.

Die Qualität von gegenwärtiger Software wurde in [14] untersucht und scheint sich über die letzten Jahre tendenziell verschlechtert zu haben. Zusätzlich wurde in [39] neben dem sinkenden Trend der kritischen Sicherheitslücken festgestellt, dass mit 48% Anteil immer noch zu viele höher eingestufter Sicherheitslücken existieren, wodurch von keinem besseren Code ausgegangen werden kann.

b) Problemfeld Meldebereitschaft Eine weitere Überlegung bezüglich möglicher Gründe für die sinkenden Zahlen der Entdeckungen von kritisch eingestuften CVE ist die Meldebereitschaft für entdeckte Sicherheitslücken. Sicherheitslücken-Datenbanken berufen sich auf die „power of the crowd“ [59, S. 919] und beruhen hauptsächlich auf der Basis freiwilliger Meldung entdeckter Sicherheitslücken. Wie in Kapitel 2.2.2 aufgeführt, werden allerdings nicht alle entdeckten Sicherheitslücken gemeldet, auf Grund von unter anderem Eigennutz oder Möglichkeit des weiteren Profitgewinns. Exakte Daten oder Literatur, die sich damit beschäftigen, wie viele Sicherheitslücken gefunden, aber nicht gemeldet werden, gibt es nicht.

In [19] wird aufgeführt, dass Forschende nicht alle Sicherheitslücken betrachten können und sich somit nach Möglichkeit Sicherheitslücken mit hohem Schweregrad widmen und diese, im Gegenzug zu Sicherheitslücken mit niedrigerem Schweregrad, vermehrt melden. Dies steht allerdings konträr zu dem beobachteten Abfall des CVE-Anteils mit kritischem Schweregrad.

So scheint es auf die gebrachten Ergebnisse bezogen eher möglich, dass vor allem möglichst schwer eingestufte Sicherheitslücken weniger gemeldet werden. Ein Grund könnte dabei beispielsweise sein, dass aus ihnen potenziell der meiste Profit gewonnen werden kann. Dieser Überlegung folgend, würde sich die allgemein geringe Anzahl der CVE mit CVSS-Score von mindestens neun erklären lassen. Dabei können hohe Ausschläge daraus erfolgen, dass zusätzlich zu neuen Sicherheitslücken kritische Sicherheitslücken, die über die vorherigen Jahre entdeckt und genutzt wurden, entdeckt und gemeldet wurden.

Es ist zu beachten, dass das Veröffentlichungsdatum einer CVE nicht ihrem Entdeckungsdatum entspricht [60].

c) mehr „leichtere“ Sicherheitslücken Da die CVE-Anzahl in den letzten Jahren, wie in Abschnitt 4.1 dargestellt, stark zunimmt, gleichzeitig aber der Anteil kritisch eingestufter CVE abnimmt, bedeutet dies, dass die Anzahl leichter eingestufter CVE steigt.

Um die möglichen Gründe für die Abnahme der CVE mit kritischem Schweregrad weiterführend einzugrenzen, würde sich weitere Forschung in dem Bereich als sinnvoll erweisen.

6 Fazit

Die vorliegende Bachelorarbeit hat sich mit der Fragestellung beschäftigt, ob und warum die Anzahl von IT-Sicherheitslücken exponentiell steigt. Zur Beantwortung dieser Frage wurden drei Thesen aufgestellt, welche mit Hilfe der Auswertung von CVE- und LoC-Daten belegt beziehungsweise widerlegt wurden. Nachfolgend werden die durch die Untersuchung erlangten Erkenntnisse kurz und bündig erläutert.

Die Entwicklung der Anzahl der existenten CVE verfolgt mit hinreichender Wahrscheinlichkeit einen exponentiellen Trend. Die Entwicklung der LoC von Software-Systemen scheint ein verstärkender Faktor für die Entwicklung der CVE zu sein. Es konnte allerdings nicht nachgewiesen werden, dass die LoC-Entwicklung die CVE-Entwicklung alleinig bestimmt. Auch die Qualität von Software hat sich als ein weiterer wichtiger Einflussfaktor für die Entwicklung der Entdeckungszahlen von CVE erwiesen.

Auf Grundlage der vorangegangenen Ausführungen ist die Entwicklung der Sicherheitslücken und ihrer Meldung ein Zusammenspiel verschiedener verstärkender Einflüsse, wobei neben technischen Faktoren vor allem auch gesellschaftliche und politische Ereignisse nicht außer Betrachtung gelassen werden sollten. Mehr Forschung in dem Bereich bietet sich an.

Ausblick Neben der Verfeinerung der aufgeführten Methoden, wäre in zukünftigen Forschungsvorhaben die Betrachtung von mehr Software-Systemen und der weiterführende Vergleich der Entwicklung ihrer LoC und CVE interessant. Durch diese Forschung wäre es zudem möglich, zu überprüfen, ob die betrachteten Software-Systeme passend für einen repräsentativen Überblick gewählt waren oder einen anderen Trend zeigen, als die Mehrheit der restlichen Systeme.
Für eine weiter granulierte Betrachtung bietet es sich an, den Fokus auf die konkreten Veröffentlichungsdaten von CVE zu legen. Durch diesen Fokus würden sich mögliche Trends für die CVE-Entdeckungen innerhalb eines Jahres untersuchen und einsehen lassen.
Mit den Daten der betrachteten Software-Systeme könnten Trendvorhersagen getroffen werden, welche nachfolgend überprüft werden können. Dazu zählt beispielsweise die Entwicklung der LoC und CVE zum Ende von 2023.
Auch das Ergründen weiterer Einflussfaktoren der CVE-Entwicklung kann einen Ausgangspunkt für weitere Forschung bilden.

Zusammenfassend gibt es noch viel, was in dem Bereich der Entwicklung von IT-Sicherheitslücken erforscht werden kann. Die Ergründung der Einflussfaktoren der CVE-Entwicklung könnten eine große Rolle bei der Umsetzung von Software-Sicherheits spielen.

Anhang A: Anlagen Grundlagen

A.1 Gefährdungsablauf groß

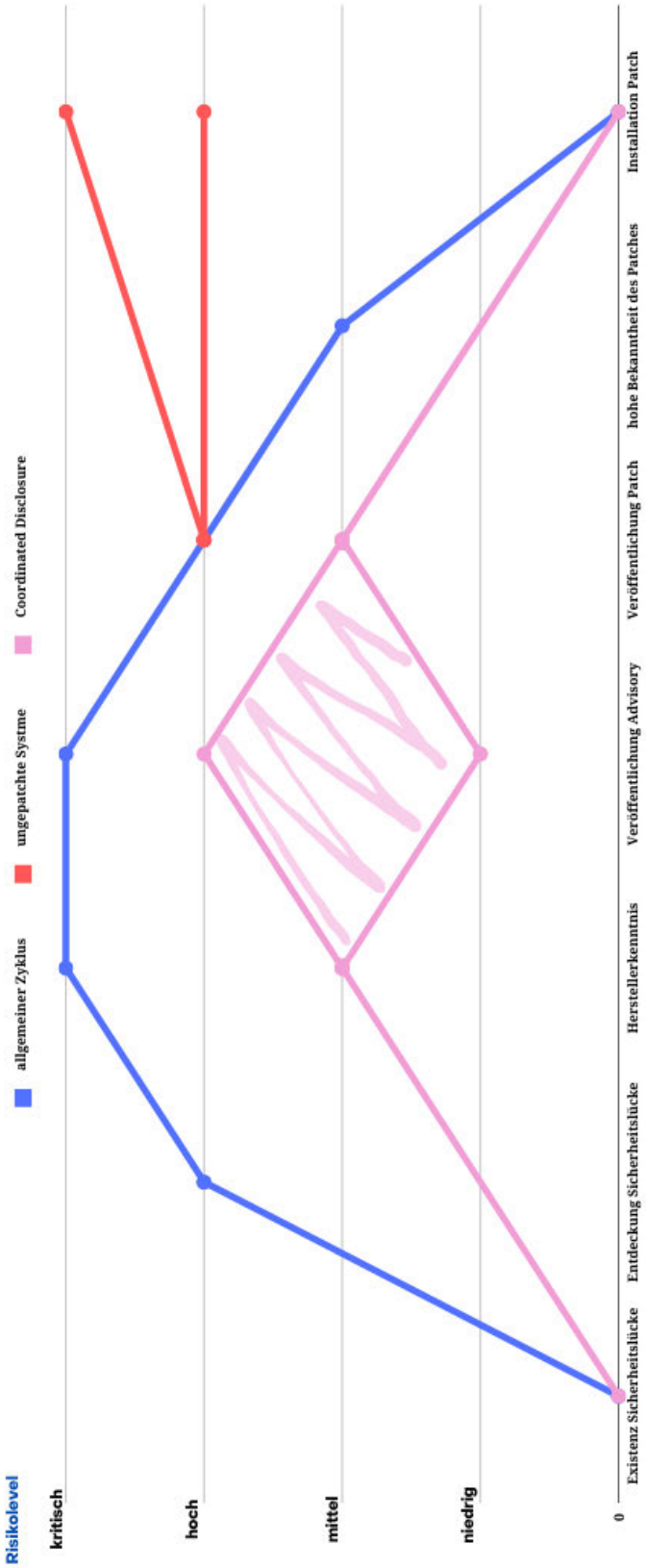


Abbildung A.1: Gefährdungsverlauf einer Sicherheitslücke siehe Abschnitt 2.3.2

Anhang B: Anlagen Methoden

B.1 Datengrundlage der Ableitungen

Die lineare Funktion wird durch die Datenpunkte 5, 10, 15, 20, 25, 30, 35, 40, 45 und 50 repräsentiert.

Die Datenpunkte der exponentiellen Funktion wurden in R über den Befehl

```
qexp(c(.10, .20, .30, .40, .50, .60, .70, .80), 2)
```

erstellt. Folgende Datenpunkte wurden zurückgegeben und weiterführend genutzt.

- 0,05268026
- 0,11157178
- 0,17833747
- 0,25541281
- 0,34657359
- 0,45814537
- 0,6019864
- 0,80471896

B.2 Ergebnisse Crawler

Abbildung B.1 zeigt einen Auszug aus den noch nicht bereinigten Ausgaben des CVE-Crawlers.

Name	Type	Value
DATA	list [25]	List of length 25
[[1]]	list [4]	List of length 4
PageID	double [1]	1
CVE	character [1]	'CVE-2023-32269'
CVSS	character [1]	'\n\t\t\t\t\t\t\t\t6.7\t\t\t\t\t\t\t\t\n\t\t\t\t\t\t\t'
Datum	character [1]	'\n\t\t\t\tPublished\n\t\t\t\t2023-05-05 17:15:10\t\t\t'
[[2]]	list [4]	List of length 4
PageID	double [1]	2
CVE	character [1]	'CVE-2023-32258'
CVSS	character [1]	'\n\t\t\t\t\t\t\t\t8.1\t\t\t\t\t\t\t\t\n\t\t\t\t\t\t\t'
Datum	character [1]	'\n\t\t\t\tPublished\n\t\t\t\t2023-07-24 16:15:12\t\t\t'
[[3]]	list [4]	List of length 4
PageID	double [1]	3
CVE	character [1]	'CVE-2023-32257'
CVSS	character [1]	'\n\t\t\t\t\t\t\t\t8.1\t\t\t\t\t\t\t\t\n\t\t\t\t\t\t\t'
Datum	character [1]	'\n\t\t\t\tPublished\n\t\t\t\t2023-07-24 16:15:12\t\t\t'
[[4]]	list [4]	List of length 4
PageID	double [1]	4
CVE	character [1]	'CVE-2023-32254'
CVSS	character [1]	'\n\t\t\t\t\t\t\t\t8.1\t\t\t\t\t\t\t\t\n\t\t\t\t\t\t\t'
Datum	character [1]	'\n\t\t\t\tPublished\n\t\t\t\t2023-07-10 16:15:52\t\t\t'
[[5]]	list [4]	List of length 4
PageID	double [1]	5
CVE	character [1]	'CVE-2023-32252'
CVSS	character [1]	'\n\t\t\t\t\t\t\t\t7.5\t\t\t\t\t\t\t\t\n\t\t\t\t\t\t\t'
Datum	character [1]	'\n\t\t\t\tPublished\n\t\t\t\t2023-07-24 16:15:12\t\t\t'
[[6]]	list [4]	List of length 4
PageID	double [1]	6
CVE	character [1]	'CVE-2023-32250'
CVSS	character [1]	'\n\t\t\t\t\t\t\t\t8.1\t\t\t\t\t\t\t\t\n\t\t\t\t\t\t\t'
Datum	character [1]	'\n\t\t\t\tPublished\n\t\t\t\t2023-07-10 16:15:52\t\t\t'
[[7]]	list [4]	List of length 4

Abbildung B.1: Auszug Crawler-Ausgabe

Anhang C: Anlagen Ergebnisse

C.1 Differenzen

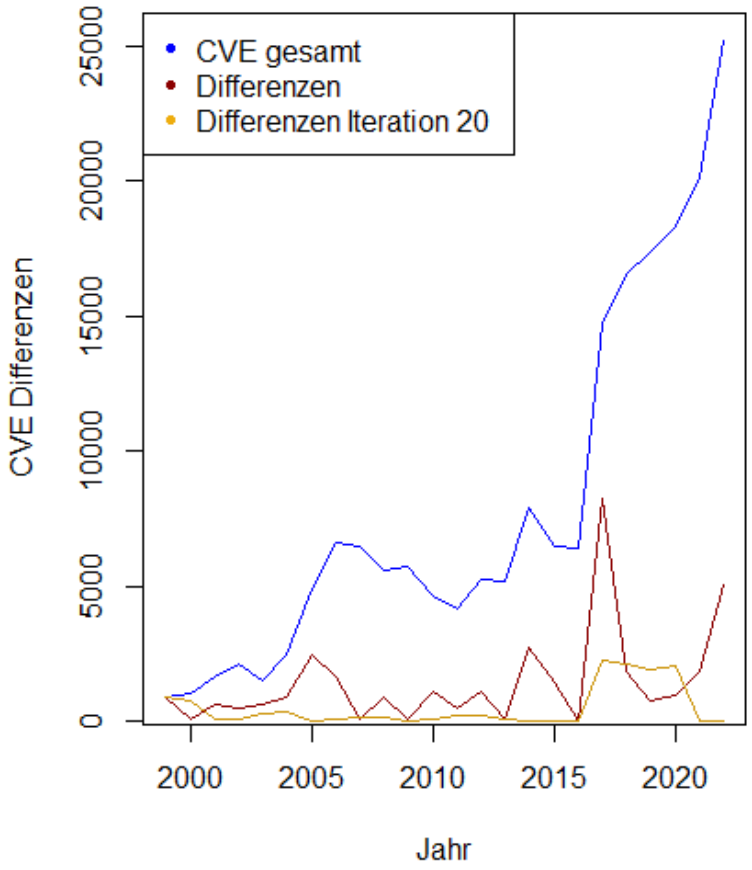


Abbildung C.1: Differenzen in der CVE-Entwicklung

Literaturverzeichnis

[1] C. Stoll, *The cuckoo's egg: tracking a spy through the maze of computer espionage*. New York: Pocket Books, 2005, OCLC: ocm61690806, ISBN: 978-1-4165-0778-9.

[2] M. Wagner und O. Vettermann, „Verantwortungsbewusster Umgang mit IT- Sicherheitslücken: Problemlagen und Optimierungsoptionen für ein effizientes Zusammenwirken zwischen IT-Sicherheitsforschung und IT-Verantwortlichen", de, 2023. DOI: 10.25353/UBTR-XXXX-8597-6CB4. Adresse: https://digitalrecht-oe.uni-trier.de/index.php/droe/catalog/book/9 (besucht am 25. 04. 2023).

[3] Cyberagentur, *Die Agentur - QA*, deutsch. Adresse: https://www.cyberagentur.de/agency/ (besucht am 25. 04. 2023).

[4] P. K. Kapur, V. S. S. Yadavali und A. K. Shrivastava, „A comparative study of vulnerability discovery modeling and software reliability growth modeling", in *2015 International Conference on Futuristic Trends on Computational Analysis and Knowledge Management (ABLAZE)*, Greater Noida, India: IEEE, Feb. 2015, S. 246–251, ISBN: 978-1-4799-8433-6. DOI: 10.1109/ABLAZE.2015.7155000. Adresse: http://ieeexplore.ieee.org/document/7155000/ (besucht am 02. 05. 2023).

[5] W. Jimenez, A. Mammar und A. Cavalli, *Software Vulnerabilities, Prevention and Detection Methods: A Review*, Juli 2010. Adresse: https://www.researchgate.net/publication/253704494_Software_Vulnerabilities_Prevention_and_Detection_Methods_A_Review_1 (besucht am 26. 05. 2023).

[6] Y. Su, M. Li, C. Tang und R. Shen, „An Overview of Software Vulnerability Detection", en, *International Journal of Computer Science and technology*, Jg. 7, Nr. 3, S. 72–76, Sep. 2016, ISSN: ISSN : 0976-8491 (Online) | ISSN : 2229-4333 (Print). Adresse: https://www.semanticscholar.org/paper/An-Overview-of-Software-Vulnerability-Detection-Su-Li/15694e11c27f32c703fc6a48a582b08290da2779 (besucht am 27. 04. 2023).

[7] O. Alhazmi und Y. Malaiya, „Quantitative vulnerability assessment of systems software", in *Annual Reliability and Maintainability Symposium, 2005. Proceedings.*, Alexandria, VA, USA: IEEE, 2005, S. 615–620, ISBN: 978-0-7803-8824-6. DOI: 10.1109/RAMS.2005.1408432. Adresse: http://ieeexplore.ieee.org/document/1408432/ (besucht am 04. 05. 2023).

[8] BSI, *Lebenszyklus einer Schwachstelle*, deutsch, Juli 2018. Adresse: https://www.allianz-fuer-cybersicherheit.de/SharedDocs/Downloads/Webs/ACS/DE/BSI-CS/BSI-CS_027.pdf?__blob=publicationFile&v=1 (besucht am 28. 04. 2023).

[9] p. kumar pawan und y. singh yudhvir, „An Overview on Software Vulnerabilities", *International Journal of Emerging Technologies and Innovative Research*, Jg. 5, Nr. 6, S. 16–21, Juni 2018. Adresse: https://www.jetir.org/view?paper=JETIR1806503.

[10] J. E. E. Schultz, D. S. Brown und T. A. Longstaff, „Responding to computer security incidents: Guidelines for incident handling", en, Lawrence Livermore National Lab., CA (USA), United States, Techn. Ber., Juli 1990. Adresse: https://www.osti.gov/biblio/6919646.

[11] C. P. Pfleeger und S. L. Pfleeger, *Security in computing*, 3rd ed. Upper Saddle River, N.J: Prentice Hall PTR, 2003, ISBN: 978-0-13-035548-5.

[12] J. T. Chambers und J. W. Thompson, „Vulnerability Disclosure Framework“, National Infrastructure Advisory Council, Techn. Ber., Jan. 2004. Adresse: https://www.cisa.gov/sites/default/files/publications/niac-vulnerability-framework-final-report-01-13-04-508.pdf (besucht am 04. 06. 2023).

[13] O. Alhazmi, Y. Malaiya und I. Ray, „Measuring, analyzing and predicting security vulnerabilities in software systems“, en, *Computers & Security*, Jg. 26, Nr. 3, S. 219–228, Mai 2007, ISSN: 01674048. DOI: 10.1016/j.cose.2006.10.002. Adresse: https://linkinghub.elsevier.com/retrieve/pii/S0167404806001520 (besucht am 25. 04. 2023).

[14] H. Krasner, „The Cost of Poor Software Quality in the US: A 2022 report“, Techn. Ber., Dez. 2022. Adresse: https://www.synopsys.com/content/dam/synopsys/sig-assets/reports/cpsq-report-nov-22-1.pdf (besucht am 16. 08. 2023).

[15] J.-B. Gao, B.-W. Zhang und X.-H. Chen, „Using CVSS to quantitatively analyze risks to software caused by vulnerabilities“, *MATEC Web of Conferences*, Jg. 31, D. Mingxing und X. Guosheng, Hrsg., S. 16 004, 2015, ISSN: 2261-236X. DOI: 10.1051/matecconf/20153116004. Adresse: http://www.matec-conferences.org/10.1051/matecconf/20153116004 (besucht am 03. 06. 2023).

[16] M. R. Lyu, Hrsg., *Handbook of software reliability engineering*. Los Alamitos, Calif. : New York: IEEE Computer Society Press ; McGraw Hill, 1996, ISBN: 978-0-07-039400-1.

[17] G. Jabeen u. a., „Machine learning techniques for software vulnerability prediction: a comparative study“, en, *Applied Intelligence*, Jg. 52, Nr. 15, S. 17 614–17 635, Dez. 2022, ISSN: 0924-669X, 1573-7497. DOI: 10.1007/s10489-022-03350-5. Adresse: https://link.springer.com/10.1007/s10489-022-03350-5 (besucht am 03. 06. 2023).

[18] V. H. Nguyen und F. Massacci, „An Independent Validation of Vulnerability Discovery Models“, 2012. DOI: 10.48550/ARXIV.1203.5830. Adresse: https://arxiv.org/abs/1203.5830 (besucht am 03. 06. 2023).

[19] F. Li und V. Paxson, „A Large-Scale Empirical Study of Security Patches“, en, in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, Dallas Texas USA: ACM, Okt. 2017, S. 2201–2215, ISBN: 978-1-4503-4946-8. DOI: 10.1145/3133956.3134072. Adresse: https://dl.acm.org/doi/10.1145/3133956.3134072 (besucht am 14. 08. 2023).

[20] Bundesministerium des Inneren, für Bau und Heimat, Hrsg., *Cybersicherheitsstrategie für Deutschland 2021*, Aug. 2021. Adresse: https://www.bmi.bund.de/SharedDocs/downloads/DE/veroeffentlichungen/2021/09/cybersicherheitsstrategie-2021.pdf;jsessionid=C0ECB7394AA6309695CE8AA10260B56D.2_cid287?__blob=publicationFile&v=2 (besucht am 07. 06. 2023).

[21] J. T. Chambers und J. W. Thompson, „Common Vulnerability Scoring System“, National Infrastructure Advisory Council, Techn. Ber., Okt. 2004. Adresse: https://www.cisa.gov/sites/default/files/publications/niac-common-vulnerability-scoring-final-report-10-12-04-508.pdf (besucht am 03. 06. 2023).

[22] P. Mell, K. Scarfone und S. Romanosky, „The common vulnerability scoring system (CVSS) and its applicability to federal agency systems“, en, National Institute of Standards und Technology, Gaithersburg, MD, Techn. Ber. NIST IR 7435, 2007, Edition: 0, NIST IR 7435. DOI: 10.6028/NIST.IR.7435. Adresse: https://nvlpubs.nist.gov/nistpubs/Legacy/IR/nistir7435.pdf (besucht am 03. 06. 2023).

[23] H. Holm und K. K. Afridi, „An expert-based investigation of the Common Vulnerability Scoring System“, en, *Computers & Security*, Jg. 53, S. 18–30, Sep. 2015, ISSN: 01674048. DOI: 10.1016/j.cose.2015.04.012. Adresse: https://linkinghub.elsevier.com/retrieve/pii/S0167404815000620 (besucht am 03. 06. 2023).

[24] Forum for Incident Response and Security Teams, „Common Vulnerability Scoring System version 3.1“, FIRST, Techn. Ber. Adresse: https://www.first.org/cvss/v3-1/cvss-v31-specification_r1.pdf (besucht am 04. 06. 2023).

[25] National Institute of Standards and Technology, *NVD - Vulnerability Metrics*. Adresse: https://nvd.nist.gov/vuln-metrics/cvss (besucht am 04. 06. 2023).

[26] J. A. Wang, H. Wang, M. Guo und M. Xia, „Security metrics for software systems“, en, in *Proceedings of the 47th Annual Southeast Regional Conference*, Clemson South Carolina: ACM, März 2009, S. 1–6, ISBN: 978-1-60558-421-8. DOI: 10.1145/1566445.1566509. Adresse: https://dl.acm.org/doi/10.1145/1566445.1566509 (besucht am 06. 06. 2023).

[27] R. Schneider, *Podiumsdiskussion: Digital sicher in eine nachhaltige Zukunft*, Mai 2023.

[28] B. Kiss, N. Kosmatov, D. Pariente und A. Puccetti, „Combining Static and Dynamic Analyses for Vulnerability Detection: Illustration on Heartbleed“, in *Hardware and Software: Verification and Testing*, N. Piterman, Hrsg., Bd. 9434, Series Title: Lecture Notes in Computer Science, Cham: Springer International Publishing, 2015, S. 39–50, ISBN: 978-3-319-26286-4 978-3-319-26287-1. DOI: 10.1007/978-3-319-26287-1_3. Adresse: http://link.springer.com/10.1007/978-3-319-26287-1_3 (besucht am 26. 05. 2023).

[29] X. Li, S. Moreschini, Z. Zhang, F. Palomba und D. Taibi, „The Anatomy of a Vulnerability Database: A Systematic Mapping Study“, en, *SSRN Electronic Journal*, 2022, ISSN: 1556-5068. DOI: 10.2139/ssrn.4211033. Adresse: https://www.ssrn.com/abstract=4211033 (besucht am 08. 06. 2023).

[30] MITRE, *CVE ID Syntax Change (Archived)*. Adresse: https://cve.mitre.org/cve/identifiers/syntaxchange.html (besucht am 07. 06. 2023).

[31] R. Byers, D. Waltermire und C. Turner, „Collaborative Vulnerability Metadata Acceptance Process (CVMAP) for CVE Numbering Authorities (CNAs) and Authorized Data Publishers“, National Institute of Standards und Technology, Techn. Ber., Dez. 2020. DOI: 10.6028/NIST.IR.8246. Adresse: https://nvlpubs.nist.gov/nistpubs/ir/2020/NIST.IR.8246.pdf (besucht am 07. 06. 2023).

[32] V. Dimitrov, „CVE (NVD) Ontology“, Sofia, Bulgarien, Mai 2022, S. 220–227. Adresse: https://ceur-ws.org/Vol-3191/paper20.pdf (besucht am 07. 06. 2023).

[33] J. Ruohonen, „A look at the time delays in CVSS vulnerability scoring“, en, *Applied Computing and Informatics*, Jg. 15, Nr. 2, S. 129–135, Juli 2019, ISSN: 22108327. DOI: 10.1016/j.aci.2017.12.002. Adresse: https://linkinghub.elsevier.com/retrieve/pii/S2210832717302995 (besucht am 03. 06. 2023).

[34] D. M. Ohm, *Studie zum Vergleich der Sicherheit von Open-Source-Software und Proprietärer Software*, Juni 2023. Adresse: https://osb-alliance.de/wp-content/uploads/2023/01/Studie-zum-Vergleich-der-Sicherheit-von-Open-Source-Software-und-proprietaerer-Software.pdf (besucht am 02. 08. 2023).

[35] M. Souppaya, K. Scarfone und D. Dodson, „Secure Software Development Framework (SSDF) version 1.1 :: recommendations for mitigating the risk of software vulnerabilities“, National Institute of Standards und Technology (U.S.), Gaithersburg, MD, Techn. Ber. NIST SP 800-218, Feb. 2022, NIST SP 800–218. DOI: 10.6028/NIST.SP.800-218. Adresse: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-218.pdf (besucht am 07. 08. 2023).

[36] L. Gentemann, D. F. Termer und D. A. Weber, „Open-Source-Monitor: Studienbericht 2021“, Techn. Ber., 2021. Adresse: https://www.bitkom.org/sites/main/files/2021-12/211207-bitkom-studie-openmonitor-2021.pdf (besucht am 03. 08. 2023).

[37] OpenUK, „State of Open: The UK in 2021 Phase Three The Values of Open“, Techn. Ber., Okt. 2021. Adresse: https://openuk.uk/wp-content/uploads/2021/10/openuk-state-of-open_final-version.pdf?ref=thestack.technology (besucht am 03. 08. 2023).

[38] „Open Source Studie 2021“, Techn. Ber., Juni 2021. Adresse: https://www.oss-studie.ch/ (besucht am 03. 08. 2023).

[39] SYNOPSYS, „Open Source Security and Risk Analysis Report“, Techn. Ber., 2023. Adresse: https://www.synopsys.com/software-integrity/resources/analyst-reports/open-source-security-risk-analysis.html (besucht am 03. 08. 2023).

[40] BSI, *Open Source Software und Vorabversionen von Betriebssystemen: Fragen & Antworten zu Software-Sicherheit*, 2021. Adresse: https://www.bsi.bund.de/DE/Themen/Verbraucherinnen-und-Verbraucher/Informationen-und-Empfehlungen/Cyber-Sicherheitsempfehlungen/Updates-Browser-Open-Source-Software/Open-Source-Vorabversionen-von-Betriebssystemen/open-source-vorabversionen-von-betriebssystemen.html (besucht am 03. 08. 2023).

[41] C. Olston und M. Najork, „Web Crawling“, en, *Foundations and Trends® in Information Retrieval*, Jg. 4, Nr. 3, S. 175–246, 2010, ISSN: 1554-0669, 1554-0677. DOI: 10.1561/1500000017. Adresse: http://www.nowpublishers.com/article/Details/INR-017 (besucht am 01. 06. 2023).

[42] B. Liu, *Web Data Mining: Exploring Hyperlinks, Contents, and Usage Data*, en. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, ISBN: 978-3-642-19459-7 978-3-642-19460-3. DOI: 10.1007/978-3-642-19460-3. Adresse: https://link.springer.com/10.1007/978-3-642-19460-3 (besucht am 01. 06. 2023).

[43] S. Khalil und M. Fakir, „RCrawler: An R package for parallel web crawling and scraping“, en, *SoftwareX*, Jg. 6, S. 98–106, 2017, ISSN: 23527110. DOI: 10.1016/j.softx.2017.04.004. Adresse: https://linkinghub.elsevier.com/retrieve/pii/S2352711017300110 (besucht am 12. 05. 2023).

[44] E. Ferrara, P. De Meo, G. Fiumara und R. Baumgartner, „Web data extraction, applications and techniques: A survey“, en, *Knowledge-Based Systems*, Jg. 70, S. 301–323, Nov. 2014, ISSN: 09507051. DOI: 10.1016/j.knosys.2014.07.007. Adresse: https://linkinghub.elsevier.com/retrieve/pii/S0950705114002640 (besucht am 01. 06. 2023).

[45] P. Srinivas Sajja und R. Akerkar, *Intelligent technologies for web applications* (Chapman & Hall/CRC data mining and knowledge discovery series). Boca Raton: CRC Press, 2012, OCLC: ocn748331540, ISBN: 978-1-4398-7162-1. Adresse: https://books.google.de/books?hl=de&lr=&id=HqXxoWK7tucC&oi=fnd&pg=PP1&ots=XvfXOGT18Z&sig=ZI67cNH--Z_PbKLvDmU-CXoRfEc&redir_esc=y#v=onepage&q&f=false.

[46] The R Core Team, *R: A Language and Environment for Statistical Computing*, 2014. Adresse: http://lib.stat.cmu.edu/R/CRAN/doc/manuals/r-devel/fullrefman.pdf (besucht am 03. 06. 2023).

[47] Y. Zhao, *R and data mining: examples and case studies*, First edition. Amsterdam ; Boston: Academic Press, an imprint of Elsevier, 2013, ISBN: 978-0-12-396963-7.

[48] J. F. Hair, G. T. M. Hult, C. M. Ringle, M. Sarstedt, N. P. Danks und S. Ray, „Overview of R and RStudio“, en, in *Partial Least Squares Structural Equation Modeling (PLS-SEM) Using R*, Series Title: Classroom Companion: Business, Cham: Springer International Publishing, 2021, S. 31–47, ISBN: 978-3-030-80518-0 978-3-030-80519-7. DOI: 10.1007/978-3-030-80519-7_2. Adresse: https://link.springer.com/10.1007/978-3-030-80519-7_2 (besucht am 06. 06. 2023).

[49] RDocumentation, *cor.test*. Adresse: https://www.rdocumentation.org/packages/stats/versions/3.6.2/topics/cor.test (besucht am 12. 07. 2023).

[50] C. Hummert, „Analyse und Interpretation der Varianz von Genexpressionsdaten“, Diss., Friedrich-Schiller-Universität Jena, Jena, 2014. Adresse: https://www.db-thueringen.de/servlets/MCRFileNodeServlet/dbt_derivate_00031054/Diss/Dissertation.pdf (besucht am 24. 07. 2023).

[51] W3Techs, *Usage statistics and market share of WordPress*. Adresse: https://w3techs.com/technologies/details/cm-wordpress (besucht am 17. 07. 2023).

[52] *About WordPress*. Adresse: https://wordpress.org/about/ (besucht am 17. 07. 2023).

[53] *QEMU*. Adresse: https://www.qemu.org/ (besucht am 17. 07. 2023).

[54] *FFmpeg*. Adresse: https://ffmpeg.org/ (besucht am 18. 07. 2023).

[55] Linux Kernel Organization, *About Linux Kernel*. Adresse: https://www.kernel.org/linux.html (besucht am 23. 08. 2023).

[56] F. Illing, *Deutschland in der Finanzkrise: Chronologie der deutschen Wirtschaftspolitik 2007 - 2012*, de. Wiesbaden: Springer Fachmedien Wiesbaden, 2013, ISBN: 978-3-531-19824-8 978-3-531-19825-5. DOI: 10.1007/978-3-531-19825-5. Adresse: https://link.springer.com/10.1007/978-3-531-19825-5 (besucht am 10. 08. 2023).

[57] VERACODE, „The State of Software Security Regional Snapshot: Europe“, Techn. Ber. 12, 2021. Adresse: https://www.veracode.com/sites/default/files/pdf/resources/sossreports/state-of-software-security-v12-european.pdf (besucht am 21. 08. 2023).

[58] O. S. Gräupner, „Grundlagen auditierbarer KI-Systeme“, Magisterarb., 2023. Adresse: https://monami.hs-mittweida.de/frontdoor/deliver/index/docId/14380/file/MA_47120_Olivia-Graeupner_geschwaerzt.pdf (besucht am 31. 08. 2023).

[59] D. Mu u. a., „Understanding the Reproducibility of Crowd-reported Security Vulnerabilities“, in *27th USENIX Security Symposium (USENIX Security 18)*, USENIX Association, Hrsg., Baltimore, MD, Aug. 2018, S. 919–936. Adresse: https://www.usenix.org/conference/usenixsecurity18/presentation/mu (besucht am 27. 07. 2023).

[60] National Institute of Standards and Technology, *NVD Glossary*. Adresse: https://ncp.nist.gov/glossary (besucht am 14. 08. 2023).

Eidesstattliche Erklärung

Hiermit versichere ich – Joline Wochnik – an Eides statt, dass ich die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe.

Sämtliche Stellen der Arbeit, die im Wortlaut oder dem Sinn nach Publikationen oder Vorträgen anderer Autoren entnommen sind, habe ich als solche kenntlich gemacht.

Diese Arbeit wurde in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegt oder anderweitig veröffentlicht.

Mittweida, 08.09.2023

Ort, Datum