HOCHSCHULE
MITTWEIDA
University of Applied Sciences

BACHELORARBEIT

Herr
Markus Popp

Eine abstrakte Meta-Penetration-Test-Methodik auf Basis von Threat-Modeling

2021

Fakultät **Computer- und Biowissenschaften**

BACHELORARBEIT

Eine abstrakte Meta-Penetration-Test-Methodik auf Basis von Threat-Modeling

Autor:
Markus Popp

Studiengang:
IT-Sicherheit

Seminargruppe:
IF18wI1-B

Erstprüfer:
Prof. Dipl.-Ing. (BA) Ronny Bodach

Zweitprüfer:
Diplom der Informatik Jan Girlich

Mittweida, 2021

Bibliografische Angaben

Popp, Markus: Eine abstrakte Meta-Penetration-Test-Methodik auf Basis von Threat-Modeling, 61 Seiten, 9 Abbildungen, Hochschule Mittweida, University of Applied Sciences, Fakultät Computer- und Biowissenschaften

BACHELORARBEIT, 2021

Referat

Seit nun mehr vielen Jahren ist der Einsatz von Software in allen Lebenslagen nicht mehr wegzudenken. Das Leben von fast allen Menschen wird täglich, bewusst oder unbewusst, von Software gesteuert, unterstützt oder beeinflusst. Da Softwareprodukte auch immer weitreichendere Eingriffe in persönliche Daten nehmen, sollte ein Hauptaugenmerk der Softwareentwicklung stets auf Sicherheit und Datenschutz gelegt werden. Umso wichtiger ist es daher, dass nicht nur Sicherheitsuntersuchungen durchgeführt werden, sondern dass diese auch möglichst umfassend und strukturiert realisiert werden.
Die vorliegende Arbeit beschäftigt sich daher mit der Entwicklung einer Methodik zur schrittweisen Überführung eines abstrakten Architekturmodells, wie beispielsweise einem Datenflussdiagramm, hin zu einem möglichst vollständigen Testplan zur Durchführung reproduzierbarer Penetrationstests, unter Einsatz von Hilfsmodellen zur Gefahrenklassifizierung. Hierbei sollen Konzepte, wie Threat-Modeling auf Basis des STRIDE-Modells und Finden von Sicherheitslücken mithilfe der *Common Vulnerability and Exposures*-Datenbank zum Einsatz kommen.

I. Inhaltsverzeichnis

II. Abbildungsverzeichnis

1 Einleitung und Motivation

Seitdem Software im kommerziellen und persönlichen Bereich Einsatz findet, wird versucht, Fehler in der Programmierung auszunutzen. Mögliche Motivationen für Angreifer decken hierbei ein breites Spektrum ab. Eines der Motive ist häufig die Bereicherung, welche über viele Wege erreicht werden kann. Ein prominentes Beispiel sind die in letzter Zeit immer häufiger auch medial auftretenden *Ransomware*-Angriffe. Bei dieser Art von Angriff werden Unternehmensdaten verschlüsselt und nur gegen Zahlung eines Lösegeldes (engl. „ransom“) wieder entschlüsselt.
Manchmal dienen diese Angriffe jedoch auch zu etwas Positivem, wie im Beispiel von „Alexey“, der im Jahr 2018 Schwachstellen in *MikroTik*-Routern ausnutzte, um Sicherheitspatches einzuspielen und die Benutzer vor weiterem Unglück zu bewahren.[1] Diese Fälle stellen jedoch die Minderheit aller Angriffe dar. Umso wichtiger ist es, bereits während der Entwicklung, beziehungsweise vor Veröffentlichung einer Software, Schritte zu unternehmen, um eine solche Gefährdungslage erst gar nicht entstehen zu lassen.

1.1 Stand der Technik

Aufgrund der fortwährenden Digitalisierung, sowie der steigenden Komplexität der eingesetzten Softwarelösung ist auch eine Vergrößerung der Angriffsfläche auf Software festzustellen. Die dadurch resultierende steigende Wichtigkeit von Sicherheitsexperten lässt sich unter anderem daran erkennen, dass allein zwischen 2014 und 2021 die Anzahl von offenen Stellen in der IT-Sicherheit weltweit von einer Million auf dreieinhalb Millionen gestiegen ist.[2] Da ein erfolgreicher Angriff auf Unternehmen mit geschätzten Durchschnittskosten von 3,86 Millionen Dollar[3] häufig einen großen finanziellen Schaden darstellt, wird versucht, die Angriffsfläche und das Auftreten möglicher Sicherheitslücken mithilfe verschiedener Maßnahmen zu minimieren. So hat sich in den letzten Jahren der Begriff des *Security-by-Design*-Konzepts etabliert, welcher beschreibt, dass Softwaresysteme bereits in der Planungsphase mit sicheren Schnittstellen und Kommunikationskanälen konzipiert werden. Neben dieser sorgfältigen Planung und kontinuierlichen Tests während der Entwicklung, werden bei sicherheitskritischen Komponenten häufig auch sogenannte *Penetration-Tests* durchgeführt. Dies bedeutet, dass das Softwareprodukt aus der Sicht eines potenziellen Angreifers betrachtet wird und dementsprechende Angriffstechniken zur Manipulation der Software eingesetzt werden. Somit können bereits vor der Veröffentlichung der Software mögliche Sicherheits- und Datenschutzmängel identifiziert und behandelt werden. Da Entwickler hierfür häufig nicht die nötige Spezialisierung zur Durchführung dieser Tests besitzen, wird oft auf einen externen Dienstleister ausgewichen.

1.2 Problemstellung

Allerdings existieren bei der heutigen Art und Weise der Durchführung eines Penetration-Tests mehrere Probleme. Eines der Hauptprobleme ist hierbei die Erstellung und Durchführung von reproduzierbaren Tests. Zwar ist die Dokumentation aller Tests ein Kernbestandteil der Analyse, doch besteht immer noch ein Unterschied im Vorgehen verschiedener Analysten, insbesondere in Hinblick auf unterschiedliche Wissensstände und Spezialisierungen. Sollte also ein Test zu einem späteren Zeitpunkt von einem anderen Tester durchgeführt werden, so sind andere *Findings* und Ergebnisse zu erwarten. *Re-Tests*, also die erneute Analyse von Komponenten nach beispielsweise dem Einspielen von Updates, sollten zu einem gewissen Maß reproduzierbar sein. Allerdings können auch hierbei Abweichungen durch die gewählte Methodik des Analysten auftreten.

Eine weitere Problematik ist das Finden aller möglichen Sicherheitslücken. Da es praktisch unmöglich ist, *alle* vorherrschenden Sicherheitslücken zu finden, sollte stets angestrebt werden, so viele wie möglich zu identifizieren, um eine Behebung der Mängel durchführen zu können. Hierbei spielt die Spezialisierung eines Analysten, in Hinblick auf Fertigkeiten zum Finden und Ausnutzen bestimmter Arten von Sicherheitslücken eine große Rolle. Besitzt ein Analyst beispielsweise verstärkte Fähigkeiten zur Ausnutzung von *Cross-Site-Scripting*-Fehlern (XSS), aber dafür hingegen lediglich Grundlagenverständnis im Umgang mit *SQL-Injections*, kann dies die Ergebnisse einer Analyse einer Applikation mit starkem Einsatz von Datenbanksystemen maßgeblich beeinflussen. Claudia Eckert beschreibt Cross-Site-Scripting als die Gefahr, dass durch nicht korrekt überprüfte Eingaben Schadsoftware eingeschleust werden kann.[4] Sie beschreibt außerdem eine SQL-Injection als Angriff der, ähnlich wie XSS, auf mangelhafter Eingabeprüfung basiert, bei dem allerdings Datenbankanfragen eingeschleust werden.[4]

Wird sich bei einem Penetration-Test an einer gegebenen Methodik oder Rahmen orientiert, so ist die Wahrscheinlichkeit etwas zu übersehen geringer. Um dies zu unterstützen, hat sich in den vergangenen Jahren eine Vielfalt an Konzepten und methodischen *Frameworks*, wie beispielsweise dem *Open Source Security Testing Methodology Manual*[5] oder dem *OWASP Web Security Testing Guide*[6], herausgebildet.

Des Weiteren werden Sicherheitsanalysen häufig von projektinternen Entwicklern durchgeführt. Dies kann von Vorteil sein, da das bereits vorhandene Wissen über den Aufbau des Projekts eine Einarbeitung vereinfacht und die Analyse in komplexeren Teilen der Softwarearchitektur ermöglicht. Allerdings kann es passieren, dass durch eine Beteiligung am Entwicklungsprozess dem Analysten eine unvoreingenommene Herangehensweise fehlt und Schwachstellen übersehen werden.

Die in dieser Arbeit beschriebene Methodik soll daher unter Einsatz von modernen Konzepten und Strategien eine möglichst umfangreiche und reproduzierbare Teststrategie ermöglichen. Insbesondere kann der Einsatz von *Threat-Modeling*-Techniken zur allgemeinen Reproduzierbarkeit der Analysen beitragen. Indem eine gegebene Softwarearchitektur nach einer Schrittfolge bearbeitet wird, wird einerseits sichergestellt, dass keine Komponenten übersehen werden, andererseits ermöglicht es ein geordnetes Abarbeiten der Analyseschritte. Dies wird besonders durch den hohen Detailgrad der Dokumentation der einzelnen Zwischenschritte verstärkt, da somit eine genaue Nachvollziehbarkeit gewährleistet ist. Außerdem kann der Softwareentwickler im Aufbau eines Verständnisses für das Auftreten von gefährdenden Programmkonstellationen unterstützt werden.

Ziel der Arbeit ist es demnach zu prüfen, ob der Entwurf einer Methodik zur Erstellung eines reproduzierbaren und möglichst vollständigen Testplans unter Einsatz bereits existierender Hilfsmodelle möglich ist.

1.3 Aufbau der Arbeit

Zunächst soll die entwickelte Methodik als abstrakter Algorithmus dargestellt werden. Die einzelnen Schritte werden im Anschluss präziser beschrieben. Danach folgt eine konkrete Beispielimplementierung des Algorithmus unter Einsatz von modernen Frameworks und Werkzeugen zur genaueren Beschreibung der Vorgehensweise. Um die beschriebenen Schritte zu veranschaulichen folgt ein minimalistisches akademisches Fallbeispiel. Zum Abschluss findet eine Diskussion mit Ausblick und einigen Anmerkungen zum Einsatz im realen Umfeld statt.

2 Algorithmus

In diesem Kapitel soll das theoretische Grundkonzept sowie eine mögliche Umsetzung unter Verwendung konkreter Bestandteile dargestellt werden. Da das zugrunde liegende abstrakte Konzept austauschbare Bestandteile besitzt, sollen abschließend außerdem einige Alternativen zu den in der Umsetzung eingesetzten Bausteinen beschrieben werden.

2.1 Begriffsdefinition

Im deutschen wie im englischen Sprachgebrauch werden einige Wörter, wie *Schwachstelle (Weakness)*, *Sicherheitslücke (Vulnerability)* oder *Gefahr (Threat)* häufig synonymisch verwendet. Im Kontext dieser Arbeit ist es jedoch relevant, zwischen den Begrifflichkeiten Abgrenzungen zu treffen. Folgend sind daher die eingesetzten Begriffe definiert.

Gefahr (Threat) Unter einer *Gefahr (Threat)*, oder auch *elementaren Gefährdung*[7] wird ein konkretes Bedrohungspotenzial, welches unter bestimmten Umständen eintreten kann, verstanden. Beispiele für Gefahren sind *Tampering*[8], also die Manipulation von Informationen, oder auch *Abhören*. Damit werden laut Bundesamt für Sicherheit in der Informationstechnik gezielte Angriffe auf Kommunikationsverbindungen, Gespräche oder IT-Systeme zur Informationssammlung beschrieben.[7] Bei Gefahren wird lediglich ein Bedrohungspotenzial durch äußere Einflüsse festgestellt, jedoch noch keine Möglichkeit der Ausnutzung.

Schwachstelle (Weakness) Als *Schwachstelle (Weakness)* bezeichnet man einen Implementations- oder Designfehler, welcher die Möglichkeit bietet, ausgenutzt zu werden, um eine Sicherheitslücke zu entwickeln oder zu entdecken. Beispiele für konkrete Schwachstellen sind das falsche Interpretieren von Eingabedaten (*Improper Input Validation*[9]) oder eine mangelhafte Identifikationsprüfung (*Improper Authentification*[10]).

Sicherheitslücke (Vulnerability) Unter einer *Sicherheitslücke (Vulnerability)* versteht man eine tatsächliche Ausnutzung eines Softwarefehlers eines bestimmten Produkts. Ein Beispiel für eine Sicherheitslücke ist *CVE-2020-6010*[11]. Zu beachten ist hierbei der Unterschied, dass eine Schwachstelle lediglich die *Möglichkeit* der Ausnutzung bietet, während eine Sicherheitslücke eine konkrete Ausnutzung der Schwachstelle darstellt.[12]

Exploit Ein *Exploit* ist ein entwickeltes Script oder Programm, welches bei der Ausnutzung einer Sicherheitslücke Anwendung findet. Der Exploit unterscheidet sich von der Sicherheitslücke selbst dadurch, dass ein Exploit eine konkrete Implementation einer Ausnutzung einer Sicherheitslücke ist. So kann ein Exploit beispielsweise in der Skriptsprache *Python* und ein anderer in der Sprache *C* geschrieben sein. Dies wären dann zwei unterschiedliche Exploits, welche aber beide für die selbe Sicherheitslücke genutzt werden können. Ein Exploit wird hierbei manchmal auch als *Schadroutine* bezeichnet. Ein Beispiel für einen Exploit ist das von *xynmaps* entwickelte Proof-of-Concept-Skript für die Ausnutzung einer *Remote Denial of Service*-Sicherheitslücke im Programm *vsftpd* in der Version 3.0.3.[13]

Dokumentation Die *Dokumentation* bildet das Protokoll über die gesamte Untersuchung und beinhaltet alle Informationen über durchgeführte Tests, analysierte Komponenten, Findings und Bemerkungen. Eine sorgfältige Dokumentation führt zu reproduzierbaren Tests und stellt Vorgehensweisen und Ergebnisse der Untersuchung dar. Es ist wichtig, auch Zwischenschritte zu dokumentieren, um neue Ansätze von jedem Zwischenstandpunkt aus zu ermöglichen.

Testplan Ein *Testplan* beschreibt eine Auflistung aller letztendlich durchzuführenden Tests an der Software. Der Detailgrad und Inhalt des Testplans ist variabel, und kann von einer Liste von zu testenden Exploits bis hin zu einer Ausarbeitung mehrerer Angriffsvektoren und Methodiken reichen. Im Gegensatz zur *Dokumentation*, welche hauptsächlich der Protokollierung und Nachvollziehbarkeit dient, besteht der Testplan hauptsächlich aus dem Endergebnis des in dieser Arbeit dargestellten Algorithmus und dient vordergründig einem Analysten zur systematischen Planung und Durchführung von Tests.

2.2 Methodik

Bei der Durchführung von Penetration-Tests wird grundsätzlich zwischen *Black*-, *Grey*- und *Whitebox*-Tests unterschieden. Claudia Eckert[4] beschreibt einen Whitebox-Ansatz als Test aus Sicht eines Innentäters, also einer Position mit Kenntnis über die inneren Abläufe und Zusammenhänge der Architektur. Im Gegensatz hierzu steht der Außentäter eines Blackbox-Tests, welcher nach Eckert lediglich über wenige und leicht zugängliche Informationen verfügt. Als Mischform existiert der Greybox-Test, bei dem der Analyst mit einigen, jedoch nicht allen Informationen im Vorhinein betraut wird. Aufgrund der Notwendigkeit detaillierter Informationen über den Aufbau der zu untersuchenden Softwarearchitektur ist zu beachten, dass sich die in dieser Arbeit dargestellten Methodik lediglich für Whitebox-Tests eignet.

In Abbildung 2.1 ist schematisch der Ablauf der einzelnen Phasen des entwickelten Algorithmus dargestellt. Folgend sind die einzelnen Schritte genauer definiert.

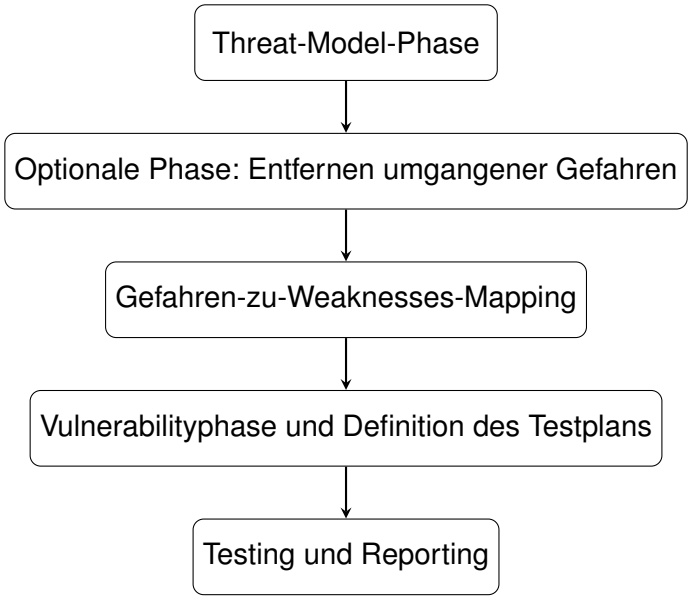


Abbildung 2.1: Schematische Schrittfolge

Die erste Phase dient zur Entwicklung des Threat-Models, auf dessen Grundlage im Anschluss weiter operiert werden kann. Unter einem *Threat-Model* ist keine explizite Umsetzung einer Strategie oder Prozedur zu verstehen. Victoria Drake beschreibt Threat-Modeling als jede strukturierte Repräsentation aller Informationen, die die Sicherheit einer Applikation betreffen.[14] Dementsprechend gibt es keine standardisierten Vorgaben zur Erstellung eines solchen Modells, lediglich einige Rahmenkonzepte. Die Anforderungen, die an ein Threat-Model gestellt sind, sind jedoch weitgehend gleichbleibend. Im *OWASP Web Security Testing Guide*[6] sind diese, angelehnt an den Standard *NIST 800-30*[15] für Risikoeinschätzung, beschrieben. Zusammengefasst muss jedes Threat-Model die nachfolgenden Kriterien erfüllen. So muss ein Threat-Model eine Zerlegung der Architektur bieten, sodass es möglich ist, ausgehend vom Modell, die Funktionswei-

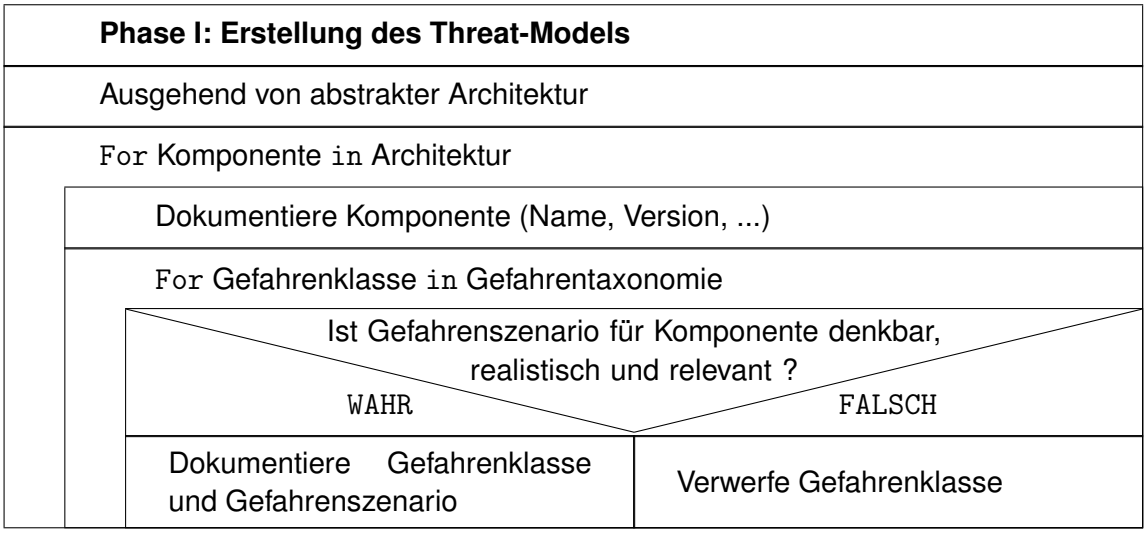


Abbildung 2.2: Phase I – Von abstrakter Architektur zum Threat-Model

se der Architektur nachvollziehen zu können. Außerdem muss es möglich sein, potenzielle Gefahren und Sicherheitslücken zu identifizieren und mithilfe des Modells Maßnahmen konzipieren zu können.

Ausgangspunkt für die Erstellung eines Threat-Models mithilfe der in Abbildung 2.2 dargestellten Schrittfolge bildet eine bereits existierende abstrakte Darstellung der zu untersuchenden Architektur. Ist diese noch nicht durch beispielsweise in der Planungsphase des Projekts entwickelte Schemata gegeben, oder erweisen sich diese als ungeeignet, so muss ein solches Modell für diesen Schritt erstellt werden. Hierbei kann es von Vorteil sein, im Modell alle Kommunikationsendpunkte mit erkennbaren Ein- und Ausgängen aufzuteilen. Je nach Funktionsweise kann eine Komponente auf dem Hin- und Rückkanal unterschiedliche Aufgaben erfüllen und somit unterschiedlichen Angriffsvektoren ausgesetzt sein. Zu bedenken ist außerdem, dass auch relevante menschliche Schnittstellen, also beispielsweise von einem Nutzer getätigte Eingaben, im Modell eingepflegt werden. Denn auch von der menschlichen Interaktion geht ein Gefahrenpotenzial aus, welches oft höher wiegt als angenommen. Dieses Potenzial reicht von simplen nicht bedachten Szenarien, wie der Benutzung eines einzelnen Eingabegeräts durch mehrere Nutzer, bis hin zu aktiven Angriffen, wie *Social-Engineering* oder *Phishing*. Claudia Eckert beschreibt *Social-Engineering*, auch *Social Hacking* genannt, als Angriff, bei dem versucht wird, einem Opfer sensible Informationen zu entlocken.[4] Häufig werden hierbei psychologische Tricks angewendet. So wird beispielsweise versucht einem Opfer Dringlichkeit zu suggerieren und das Opfer somit zur überstürzten Informationspreisgabe zu drängen. Der Begriff *Phishing* setzt sich aus den Begriffen „Passwort“ und „fishing“, zu Deutsch „Angeln“ zusammen.[4] Es beschreibt einen Angriff, bei dem ein Nutzer dazu verführt werden soll, auf einer gefälschten Seite Logindaten einzugeben und diese somit einem Angreifer zu überlassen.

Alle beteiligten Elemente, welche im folgenden Threat-Model Einzug finden, werden als *Komponenten* bezeichnet. Ist eine Komponente eine eingesetzte Version eines nicht innerhalb des Projekts entwickelten Programms oder einer Bibliothek, so wird diese als *externes Programm* oder *externe Bibliothek* bezeichnet. Diese Unterscheidung zu *intern* entwickelten Funktionen, Programmen oder Bibliotheken ist relevant, da es bei der Sicherheitsuntersuchung vordergründig darum geht, die innerhalb des Projekts entstandenen Komponenten zu prüfen. Auch die Einbindung von Schnittstellen externer Komponenten spielt bei der Analyse eine Rolle, jedoch ist es meist nicht zielführend die externen Komponenten selbst genauer zu untersuchen.

Ausgehend von dieser abstrakten Architektur kann nun das Threat-Model entwickelt werden. Hierbei werden nun alle Komponenten iterativ analysiert. Bei getrenntem Hin- und Rückkanal sind dabei mögliche Gefahren getrennt zu betrachten, da zum Beispiel eine Manipulation der Nachrichten beim Absenden auf einem Nutzergerät im Gegensatz zum Abhören der eintreffenden Nachrichten zwar am selben Endpunkt geschehen, aber zwei völlig unterschiedliche Gefahren sind.
Auf jede der Komponenten wird nun eine Gefahrentaxonomie projiziert. Das bedeutet, dass eine noch sehr grobe und abstrakte Gefahrenklassifizierung angewendet wird, welche dazu eingesetzt wird, um den zu untersuchenden Gefahrenbereich einzugrenzen, einzuteilen und bereits Gefahrenklassen zu eliminieren, die im gegebenen Modell nicht auftreten können. Die Ergebnisse können je nach angewendeter Taxonomie sehr unterschiedlich sein. Wird eine relativ feinkörnig definierte Taxonomie, wie der Gefahrenkatalog *elementare Gefährdungen* des *Bundesamts für Sicherheit in der Informationstechnik*[1] eingesetzt, so können viele Gefahren, wie „G 0.2 Ungünstige klimatische Bedingungen“[7] für viele Komponenten mit Hinweis auf Maßnahmen für Ausfallsicherheit oder sehr geringer Wahrscheinlichkeit ignoriert werden. Je abstrakter allerdings die Taxonomie, desto vielfältiger können mögliche Gefahrenszenarios sein. Ob eine Gefahr für eine Komponente zutreffend ist, kann mithilfe hypothetischer *Szenarien* geprüft werden. So kann für die Gefahr „G 0.1 Feuer“[7] überlegt werden:

> Kann ein Gerät Feuer fangen?
> Wie hoch ist die Eintrittswahrscheinlichkeit hierfür?
> Ist die Gefahr für das System relevant?

Beispielsweise könnte man hierbei argumentieren, dass ein Datenbankserver tatsächlich mit einer gewissen Wahrscheinlichkeit Feuer fangen könnte, und dies, wenn keine Sicherungskopien existieren, kritische Auswirkungen auf den Geschäftsablauf hätte. In diesem Fall sollten entsprechende Prüfungen durchgeführt und ein entsprechendes Szenario im Threat-Model aufgenommen werden. Betrachtet man hingegen ein Nutzerendgerät, so existiert auch hier die Gefahr einer Brandentwicklung, jedoch würden die auf einem Server gesicherten Nutzerdaten hiervon unberührt bleiben, und der temporäre Ausfall eines einzelnen Nutzers für gewöhnlich nicht den operativen Geschäftsbetrieb

[1] siehe Kapitel 2.4.1

kritisch einschränken.
In jedem Fall spielt in diesem Schritt die ebenfalls erfolgende Dokumentation eine wichtige Rolle. Es sollten stets alle Schritte, geprüfte Komponenten und erkannte Gefahrenklassen dokumentiert werden. Auch wenn nicht alle Gefahren oder Gefahrenklassen nach Anwendung des Algorithmus im Testplan auffindbar sind, so können diese Informationen *informative Findings* enthalten oder sowohl für einen Analysten, als auch den Auftraggeber neue Denkanstöße für Tests oder schützende Maßnahmen liefern. Ein *informatives Finding* ist ein während der Untersuchung identifizierter Mangel, der keinen direkten Einfluss auf die Sicherheit der Komponente besitzt. Allerdings kann ein solches Finding dazu beitragen, die Sicherheit weiter zu erhöhen und unnötige Risiken zu eliminieren. Ein Beispiel für ein informatives Finding ist die in Kapitel 2.3.1 beschriebene Preisgabe von Bannerinformationen.

Nachdem so die erste Phase abgeschlossen wurde, kann im Anschluss ein optionaler Schritt, dargestellt in Abbildung 2.3, durchgeführt werden. Somit soll ein erster Abgleich mit der realen Umsetzung durchgeführt werden. Dieser Abgleich sollte dazu dienen, Gefahren, gegen die bereits Maßnahmen eingeleitet wurden, aus der für die weiteren Schritte vorgesehenen Liste zu entfernen.
So könnte beispielsweise bereits eine Verschlüsselungstechnologie angewendet werden, um *Information Disclosure*, also die unbeabsichtigte Preisgabe von sensiblen Informationen, zu verhindern und somit diese Gefahr beim Versand von Nachrichten zu eliminieren. Allerdings sollte hierbei bedacht werden, dass das Eliminieren der Komponente nicht immer sinnvoll ist, da auch die Maßnahmen selbst Sicherheitsmängel aufweisen können. Im Gegenteil sollten eingesetzte Maßnahmen auch bei der Gefahrenevaluierung bedacht werden und sollten, je nach Situation gegebenenfalls als eigenständige Komponenten in das Threat-Model eingearbeitet werden. Wird beispielsweise ein Mailing-Gateway zur zusätzlichen automatisierten Verschlüsselung des Emailverkehrs eingesetzt, so bildet dies eine eigene Komponente mit Ein- und Ausgabeendpunkten und sollte auch gegen mögliche gefährdende Szenarien geprüft werden.

Außerdem kann in dieser Phase bereits eine Abschätzung durchgeführt werden, ob es sinnvoll ist, externe Komponenten für weitere Untersuchungen in Betracht zu ziehen. Werden externe Komponenten eingesetzt, welche in Bezug auf Sicherheit als etabliert gelten, so ist anzunehmen, dass diese bereits mehrfach und eingehend auf sicherheitskritische Fehler untersucht wurden. Eine Suche nach Sicherheitslücken muss demnach nicht ohne Erfolg sein, sich aber höchstwahrscheinlich als ineffizient herausstellen. Die für die Untersuchung benötigte Analysezeit kann in so einem Fall durch Entfernen der Komponente aus dem Testplan reduziert werden.

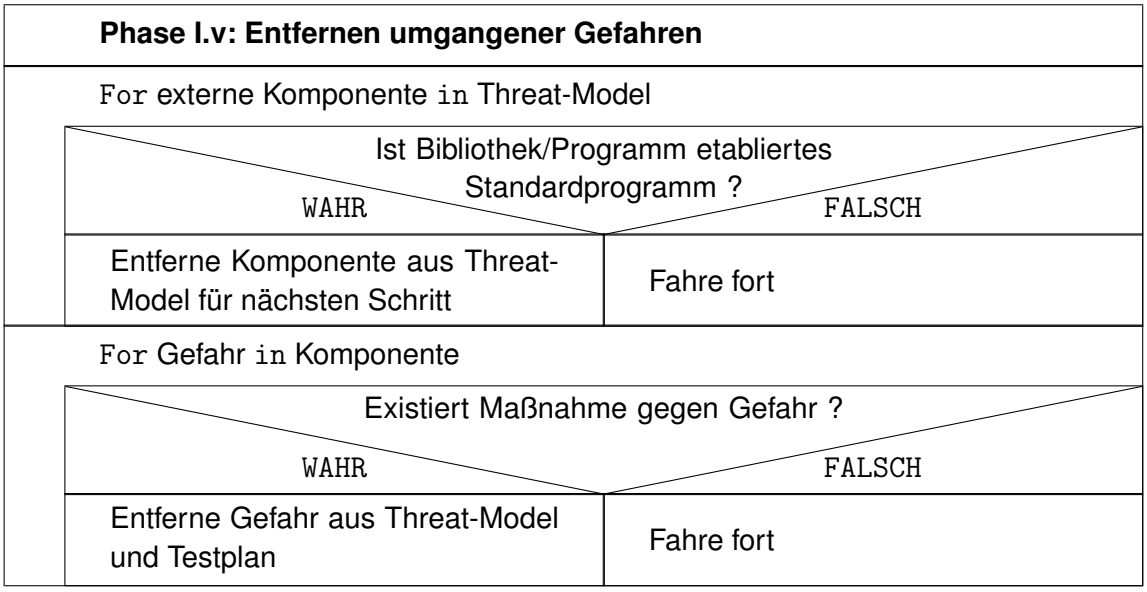


Abbildung 2.3: Phase I.v – Optionaler Zwischenschritt

In Phase II (siehe Abbildung 2.4) werden den in Phase I identifizierten Gefahren konkrete Schwachstellen zugeordnet. Hierbei wird die Schwachstelle möglichst konkret angegeben und gut dokumentiert. Es ist zu beachten, dass es möglich ist, mehrere Schwachstellen für eine Gefahrenklasse oder ein Gefahrenszenario einer Komponente zu finden. Außerdem kann die Granularität der Schwachstelleneinschätzung stark variieren und teilweise weitere Unterkategorisierungen aufweisen. Genaueres hierzu ist im Kapitel *2.3 Implementation* beschrieben.
Die Informationen zu konkreten Schwachstellen können sowohl für den Entwickler des Projekts, als auch für den Analysten zur späteren manuellen Analyse von großer Bedeutung sein. Dem Entwickler werden dabei Stichworte und weiterführende Informationen an die Hand gegeben. Hierzu können auch verwandte Probleme oder konkrete Fehlerbeispiele zählen. Dem Entwickler wird somit ein Einblick in den Hergang der Schwachstelle geboten und kann verhindern, diesen oder ähnliche Fehler erneut zu machen. Für einen anderen Analysten können diese Informationen zu einem besseren Verständnis der zu untersuchenden Architektur verhelfen, und somit den Untersuchungsaufwand minimieren. Daher ist es nicht nur aufgrund des Anspruchs auf Sorgfalt einer professionell durchgeführten Analyse wichtig, diesen Zwischenschritt vor Phase III durchzuführen.

In Phase III wird der eigentliche Testplan erstellt, also die konkrete Liste von zu untersuchenden Sicherheitslücken, beziehungsweise, falls die Sicherheitslücken für die Komponenten noch nicht explizit existieren, die Analyse der Schwachstellen. Relevant ist hierbei die in Abbildung 2.5 ersichtliche Aufteilung. Ist eine Sicherheitslücke mit bereits existierendem Exploit gefunden, so genügt es oft ein zugehöriges *Proof-of-Concept* durchzuführen, um die Anfälligkeit für die Sicherheitslücke zu prüfen. Ist ein solcher Exploit noch nicht entwickelt, beziehungsweise nicht auffindbar, so kann dieser im Rahmen

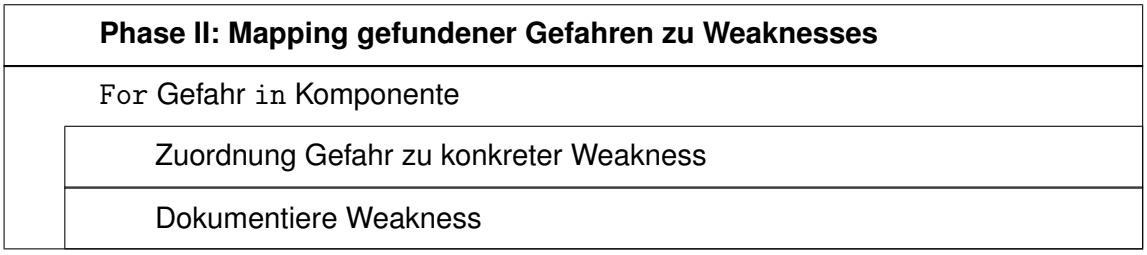


Abbildung 2.4: Phase II – Von Threat-Model zu Weaknesses

der Analyse entwickelt werden. Jedoch sollte bedacht werden, dass unter Realbedingungen eine vollständige Entwicklung eines Exploits nicht immer nötig ist. So genügt es oft, wenn die Sicherheitslücke aufgezeigt wird, um den Entwickler zur Einleitung von Gegenmaßnahmen zu bewegen. Demzufolge umfasst das Endergebnis dieser Phase die Erstellung eines strukturierten Testplans. Dieser besteht einerseits aus den für den Analysten erforderlichen Informationen zur Herleitung des Tests, also den zu untersuchenden Schwachstellen oder Sicherheitslücken, sowie dem zugehörigen zu testenden Gefahrenszenario. Andererseits sollte eine Testvorschrift entwickelt werden, welche die durchzuführenden Testmaßnahmen für die jeweilige Gefahr umschreibt.

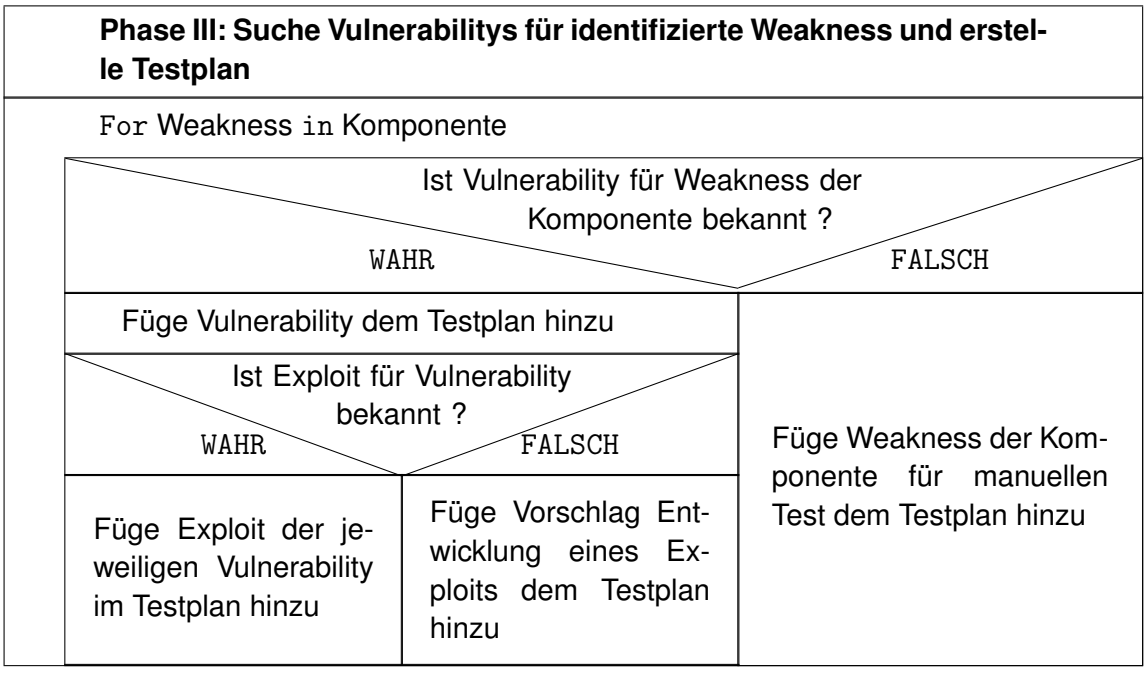


Abbildung 2.5: Phase III – Von Weakness zu Vulnerability

Im Anschluss an die in Phase III erfolgende Erstellung des Testplans folgt die tatsächliche Durchführung der Tests, sowie das Reporting der Ergebnisse. Dieser Abschnitt ist abhängig von individuellen Teststrategien und -werkzeugen eines Analysten, sowie dem vorgegebenen Reportingstils. Dementsprechend ist dieser Schritt in Abbildung 2.1 zur Komplettierung des Schemas vorgesehen, wird aber im Rahmen der in dieser Arbeit beschriebenen Methodik nicht genauer betrachtet.

2.3 Implementation

Grundsätzlich ist die entwickelte Methodik als abstraktes Modell konzipiert. Es ist daher vorgesehen, die eingesetzten Frameworks, Taxonomien und Katalogisierungen austauschbar zu halten. Näheres hierzu ist außerdem im Kapitel 4.3 beschrieben. Folgend ist in diesem Kapitel eine mögliche Implementation auf Grundlage von modernen Frameworks und Taxonomien, welche in dieser Form von Sicherheitsanalysten eingesetzt werden könnten, dargestellt.

Die in diesem Abschnitt dargestellten Beispiele sollen nur zur Veranschaulichung der einzelnen Schritte dienen. Der Zusammenhang wird in Kapitel 3 anhand eines akademischen Fallbeispiels verdeutlicht.

Bei jeder der Untersuchungsphasen ist es sinnvoll, eindeutige Identifikatoren (IDs), abhängig von Phase und Arbeitsschritt zuzuweisen. Folgend ist ein Vorschlag zur eindeutigen Nummerierung beschrieben, welche auch in Kapitel 3 bei der Untersuchung des Beispiels Anwendung findet.

- `K-I-<id>` für *Komponente intern*
- `K-E-<id>` für *Komponente extern*
- `G-<id>` für *Gefahr*, beziehungsweise *Gefahrenszenario*
- `SW-<id>` für *Schwachstelle*
- `SL-<id>` für *Sicherheitslücke*
- `E-<id>` für *Exploit*

Der Platzhalter `<id>` ist dafür vorgesehen mit einer unikalen Zahl ersetzt zu werden. Hierbei kann es allerdings sinnvoll sein, die Schrittweite zwischen zwei Elementen als größer als eins festzusetzen, um für eventuelle Zwischenschübe Platz zu reservieren. Die Vergabe von IDs für Schwachstellen, Sicherheitslücken und Exploits mag in Bezug auf die eingesetzten Datenbanken redundant erscheinen, jedoch ermöglichen sie aufgrund der aufeinander aufbauenden Struktur der Analysemethodik Rückschlüsse auf die jeweilige betroffene Komponente innerhalb der Analyse zu treffen.

2.3.1 Phase I

Die erste Phase beschreibt den Übergang von einem abstrakten Architekturmodell hin zu einem Threat-Model. Im Optimalfall sollte hierfür das Architekturmodell bereits in einer geeigneten Form vorhanden sein. Eine Möglichkeit ist der Einsatz eines *Datenflussdiagramms*, jedoch sind alle Modelle, welche die Kommunikationswege der Komponenten hinreichend darstellen, ausreichend. Sofern dies für die vorliegende Architektur sinnvoll ist, sollten auch die getrennten Hin- und Rückkanäle beachtet werden. Ist diese Darstellungsform in der Ausgangslage nicht gegeben, so sollte diese zunächst hergestellt werden.

Ist dieser Ausgangszustand hergstellt, kann wie in Abbildung 2.2 dargestellt, vorgegangen werden. Als Taxonomie wird hierbei das von Kohnfelder und Garg [16] für *Microsoft* entwickelte *STRIDE*-Modell eingesetzt. Diese Klassifizierung wird bereits in vielen Bereichen der IT-Sicherheit zur eingesetzt, unter anderem in dem von *Microsoft* entwickelten *Threat-Modeling Tool*[17]. Die relativ abstrakt gehaltene Definition des Modells ermöglicht eine hohe Flexibilität, weshalb es in einem breiten Spektrum von Anwendungsfällen eingesetzt werden kann. Das Akronym lässt sich in die möglichen Gefahrenklassen auflösen:

- **S** – Spoofing
- **T** – Tampering
- **R** – Repudiation
- **I** – Information Disclosure
- **D** – Denial of Service
- **E** – Elevation of Privilege

Spoofing Unter *Spoofing* versteht man hierbei die Vortäuschung einer falschen Identität. Bietet beispielsweise ein Programm die Möglichkeit Nachrichten unter einem frei wählbaren Namen zu versenden, so könnte sich ein böswilliger Angreifer als ein potenziell vertrauenswürdiger Kommunikationspartner ausgeben und somit andere Nutzer täuschen. Dies wird unter anderem häufig für sogenannte *Phishing*-Angriffe ausgenutzt.

Tampering *Tampering* ist die Gefahr der Manipulation von Daten. Werden Nachrichten beispielsweise ohne Mechanismen zur Manipulationssicherheit zwischen zwei Kommunikationsendpunkten versendet, so ist es einem Angreifer, welcher auf den Kommunikationskanal der Endpunkte zugreifen kann, möglich, Nachrichten zu verändern. Die Angreiferposition auf einem Kommunikationskanal zwischen zwei Endpunkten bezeichnet man auch als *Machine-in-the-Middle*. Ruby B. Lee beschreibt einen Machine-in-

the-Middle-Angriff (auch „Man-in-the-Middle“) als Situation, in der zwei Kommunikationsendpunkte denken, sie kommunizieren miteinander, während der Angreifer sich zwischen ihnen befindet und Kontrolle über die Kommunikation besitzt.[18]

Repudiation *Repudiation*, beziehungsweise die Gefahr der *non-repudiability*, beschreibt einen Zustand, wenn eine Nachricht keinem konkreten Author zugeschrieben werden kann. Claudia Eckert beschreibt ein System in dem Repudiation (dt. „Verbindlichkeit“ oder „Zuordenbarkeit“) herrscht, als ein solches, in dem es nicht möglich ist, dass ein Subjekt im Nachhinein die Durchführung einer Aktion abstreiten kann.[4] Insbesondere im finanziellen Sektor und im Bereich des Onlineshoppings und -handels spielt diese Gefahr eine große Rolle. Hierbei sollte es stets möglich sein, die Rechtskräftigkeit zu wahren und eine lückenlose beiderseitige Nachvollziehbarkeit und Authentizität zu gewähren.

Information Disclosure Unter *Information Disclosure*, zu deutsch auch oft als *Datenleck* oder *Datenreichtum* bezeichnet, versteht man die unbeabsichtigte Preisgabe von sensiblen Informationen. Ist es einem Angreifer beispielsweise ohne Prüfung einer Zugangsberechtigung möglich, Benutzerkontodaten einer Webseite mithilfe einer Abfrage an eine Programmierschnittstelle abzurufen, so ist dies als Datenleck zu bezeichnen. Häufig werden auch Machine-in-the-Middle-Angriffe dazu eingesetzt, um auf unverschlüsselten Kommunikationskanälen sensible Informationen mitschneiden zu können. Allerdings kann hierzu auch ein Verhalten, wie die Preisgabe eines sogenannten *Server-Banners*, also das Versehen einer Antwortnachricht mit Serverinformationen, wie Produktname und eingesetzter Versionsnummer, zählen.

Denial of Service Als *Denial of Service* (DoS) bezeichnet man die Gefahr der Blockade oder dem Ausfall eines Kommunikationskanals oder eines Dienstes.[4] Für gewöhnlich wird dies von Angreifern durch eine zu große Anzahl an Anfragen an einen Dienst hervorgerufen, sodass der Dienst überlastet ist und mit der Beantwortung der Anfragen blockiert ist. Allerdings kann eine solche Gefahr auch durch eine falsch verarbeitete Nachricht und dem daraus resultierenden Absturz einer Komponente eintreten. Ein Beispiel für einen solchen Fall ist ein Softwarefehler, welcher bei einem breiten Spektrum von Routern und Switches des Netzwerkherstellers *Cisco* im Jahr 2018 auftrat.[19] Hierbei führte ein fehlerhaftes Zusammenfügen von Datenpaketen zum Ausfall der Geräte.

Elevation of Privilege Unter *Elevation of Privilege* versteht man die unbefugte Ausweitung von Nutzerrechten. Zu verstehen ist hierunter beispielsweise die Ausführung einer Anfrage eines unprivilegierten Benutzers an einen Server, welcher die Anfrage so interpretiert, dass der unprivilegierte Benutzer unbefugter Weise in den Status eines Administrators erhoben wird.

Für jede Komponente, wobei eine wie in Kapitel 2.2 beschriebene Trennung von Hin- und Rückkanal sinnvoll sein kann, werden die eben definierten Gefahrenklassen appliziert. Das bedeutet, dass für jede Komponente alle Gefahrenklassen des STRIDE-Modells geprüft werden, und alle möglichen Arten und Weisen dokumentiert werden, durch welche der jeweilige Gefahrenzustand eintreten kann. Aufgrund der grobgliedrigen Kategorisierung des STRIDE-Modells sollte beachtet werden, dass pro Gefahrenklasse mehrere gefährdende Szenarien möglich sein können. Nach der vollständigen Durchführung der ersten Phase sollte demnach nun ein Threat-Model existieren und eine Strukturierung nach **Komponente – Gefahr – Gefahrenszenario** möglich sein.

Ein weiterer Bestandteil eines solchen Threat-Models sind sogenannte *Trust-Boundaries*. Das *CheatSheets Series Team* definiert Trust-Boundaries als jede Stelle, an der Daten ihr Vertrauensniveau verändern.[20] Hierunter ist zu verstehen, dass je nach Herkunft der Daten eingeschätzt wird, ob die Daten manipuliert oder verfälscht sein könnten. So sind beispielsweise Daten, welche von einem Nutzer stammen weniger vertrauenswürdig, als Daten, die aus der projektinternen Datenbank stammen. Eine theoretische Abgrenzung der Vertrauensstufen im Rahmen dieser Methodik ist nur bedingt durchführbar. Es wird daher eine rudimentäre Unterscheidung zwischen *vertrauenswürdigen* und *nicht-vertrauenswürdigen* Bereichen angenommen. Vertrauenswürdigen Komponenten, sowie ihren Unterteilungen und allen darin enthaltenen Daten wird die Glaubhaftigkeit zugesprochen. Erlaubt die Architektur den Ausschluss einer Manipulation der Daten in einem bestimmten Bereich, also wird die Vertrauenswürdigkeit der Daten angenommen, so kann hier eine Trust-Boundary gezogen werden und der Bereich von der Untersuchung ausgenommen werden.

2.3.2 Phase I.v

In Phase I.v erfolgt nun der Abgleich mit den eingesetzten Komponenten, um umgangene Gefahren aus dem Untersuchungsplan zu entfernen. Für diesen optionalen Schritt existieren jedoch keine konkreten Modelle oder Werkzeuge die eingesetzt werden können. Hier obliegt die Entscheidung der Einschätzung des Analysten, ob die Gefahr hinreichend umgangen wurde, beziehungsweise ob die eingesetzte externe Komponente nicht selbst Mängel aufweist. Wird zum Beispiel zur Sicherung eines Kommunikationskanals ein selbstentwickeltes kryptographisches System eingesetzt, so kann es sich als sinnvoll erweisen, dieses auf Sicherheit zu prüfen. Denn in der Vergangenheit hat es sich mehrfach gezeigt, dass eigenentwickelte Verschlüsselungssysteme häufig Desi-

gnfehler aufweisen. Diese Erfahrung wird auch von Zimmermann[21], dem Erfinder der *Pretty Good Privacy*-Verschlüsselung (PGP), geteilt. Auch eine veraltete oder seltene externe Komponente könnte für eine Analyse relevant sein. Wird hingegen eine aktuelle Version eines weit verbreiteten und etablierten externen Verschlüsselungsprotokolls eingesetzt, so sind die Chancen auf das Finden eines Mangels geringer. Eine Untersuchung dieser Komponente könnte demnach vermutlich nicht zwingend ohne Ergebnis sein, allerdings ist die Wahrscheinlichkeit für das Finden einer relevanten Sicherheitslücke gering.[2]

Nach sorgfältiger Abwägung kann nun entschieden werden, ob eine bestimmte Gefahr einer Komponente umgangen ist. Mögliche Faktoren die für externe Komponenten mit einbezogen werden können, sind beispielsweise, ob die Komponente in der Vergangenheit Sicherheitslücken aufgewiesen hat und ob bereits mehrere Untersuchungen der Komponente durchgeführt wurden. Auch der zeitliche Abstand zur jeweils zuletzt durchgeführten Analyse spielt hierbei eine Rolle. Wird nun die Gefahr als hinreichend behandelt eingeschätzt, so wird diese für den weiteren Verlauf der Untersuchung nicht beachtet, um den Untersuchungsaufwand zu minimieren und die Komplexität zu verringern.

2.3.3 Phase II

In der zweiten Phase soll ein *Mapping* einer Gefahr zu einer konkreten Schwachstelle durchgeführt werden. Hierzu wird über alle festgestellten Gefahren, beziehungsweise denkbaren Gefahrenszenarien, iteriert und diese zu konkreten Schwachstellen weiterentwickelt. Diese kann dann im Anschluss von einem Analysten für weitergehende Untersuchungen genutzt werden.

In der hier dargestellten Implementation wird die *Common Weakness Enumeration* (CWE) eingesetzt. Die CWE ist eine freie Schwachstellentaxonomie, geführt von der *MITRE Corporation*[22], unterstützt durch das *U.S. Department of Homeland Security*. Ziel des CWE-Projekts ist eine möglichst vollständige Katalogisierung aller Schwachstellenkategorien. Der Aufbau der CWE erfolgt hierbei nach einem hierarchischen Konzept.
Zunächst erfolgt eine Unterscheidung der zu untersuchenden Disziplin. Hierbei kann zwischen *Software Development*, *Hardware Design* und *Research Concepts* unterschieden werden. In den Disziplinen Software Development und Hardware Design folgt als nächste Präzisierung eine Aufteilung in *Categories*. Diese sind im Glossar des CWE-Projekts als Menge von Einträgen, die eine gemeinsame Charakteristik teilen, definiert.[23] Der Feinheitsgrad der Kategorien beschränkt sich auf Oberbezeichnungen, wie *CWE Category 275 – Permission Issues*[24] oder *CWE Category 310 – Cryptographic Issues*[25]. Jede der Kategorien besitzt dann Einträge, welche wiederum vier

[2] siehe Kapitel 4.1

Nature	Type	ID	Name
ChildOf	P	707	Improper Neutralization
ParentOf	C	75	Failure to Sanitize Special Elements into a Different Plane (Special Element Injection)
ParentOf	C	77	Improper Neutralization of Special Elements used in a Command ('Command Injection')
ParentOf	B	79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')
ParentOf	B	91	XML Injection (aka Blind XPath Injection)
ParentOf	B	93	Improper Neutralization of CRLF Sequences ('CRLF Injection')
ParentOf	B	94	Improper Control of Generation of Code ('Code Injection')
ParentOf	C	99	Improper Control of Resource Identifiers ('Resource Injection')
ParentOf	C	943	Improper Neutralization of Special Elements in Data Query Logic
ParentOf	B	1236	Improper Neutralization of Formula Elements in a CSV File
CanFollow	C	20	Improper Input Validation
CanFollow	C	116	Improper Encoding or Escaping of Output

Abbildung 2.6: Beispieldarstellung der hierarchischen Kategorisierung der CWE (Quelle: [26], abgerufen am 30.08.2021)

verschiedene Arten besitzen. Zum einen kann der Eintrag als *Base* bezeichnet sein, was eine von einer bestimmten Technologie unabhängige Schwachstelle beschreibt. Des Weiteren kann der Eintrag als *Variant* aufgeführt sein, was auf eine Schwachstelle in Verbindung mit einer bestimmten Sprache oder Technologie hinweist und ist spezifischer als ein Base-Eintrag. Außerdem können Einträge in Form einer *Class* auftreten, welche eine abstraktere Beschreibung als *Base* bieten.[23] Eine *Class* selbst kann auch weiter in *Base*- oder *Variant*-Einträge unterteilt werden. Ein *Pillar* stellt die abstrakteste Beschreibung dar, und wird häufig noch weiter unterteilt. In Abbildung 2.6 ist ein Beispiel dieser hierarchischen Kategorisierung für *CWE-74: Improper Neutralization of Special Elements in Output Used by a Downstream Component ('Injection')*[26] zu sehen. So ist *CWE-74* ein Kindeintrag der Pillar-Oberkategorie *CWE-707: Improper Neutralization*. Des weiteren kann die Gefahr selbst in weitere Class- und Baseeinträge unterteilt werden, gekennzeichnet durch ein grünes **C**, beziehungsweise ein blaues **B**. Außerdem sind mit *CWE-20* und *CWE-116* zwei weitere Schwachstellen aufgeführt, welche die beschriebene Schwachstelle *CWE-74* als Folge haben können.

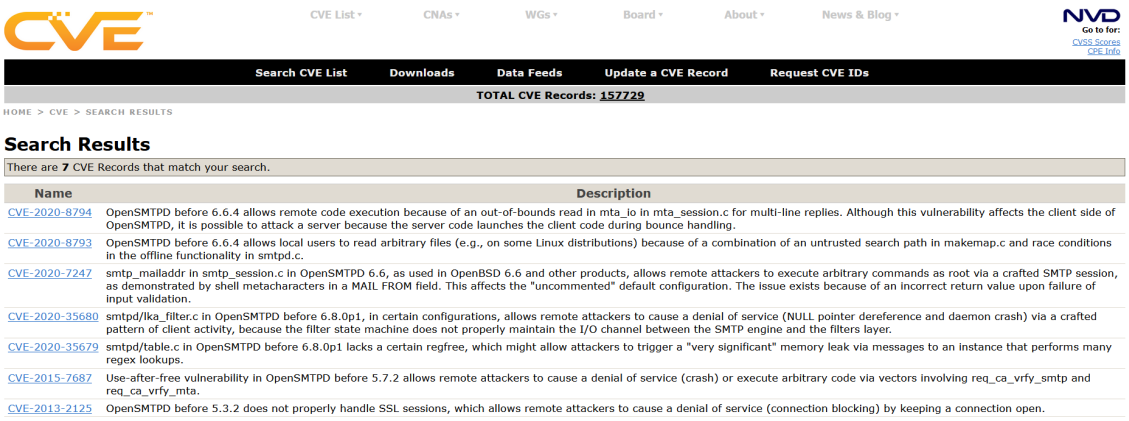
Bei Durchführung der zweiten Phase muss nun so vorgegangen werden, dass zu jedem Gefahrenszenario eine oder mehrere passende Schwachstelle zugeordnet werden. Der hierarchische Aufbau der CWE erlaubt es hierbei, sich zunächst an Oberkategorien zu orientieren und im Anschluss über Kind-Beziehungen der Schwachstellendefinitionen den Typus der Schwachstelle sehr genau einzugrenzen. Allerdings bietet es auch die Möglichkeit, bei weitreichenderen Szenarien eine gesamte Oberklasse anzugeben.
Die Informationen zur Beschreibung einer Schwachstelle, welche durch CWE angeboten werden, können vielfältig angewendet werden. So werden Entwickler mit Negativbeispielen zu fälschlichen Implementationen auf mögliche Fehler hingewiesen. Außerdem werden, sofern möglich, Weiterleitungen zu MITREs *Common Attack Pattern Enumeration and Classification* (CAPEC)[27] angeboten. Diese erweisen sich für Sicherheitsuntersuchungen von großem Wert, da hier mögliche Angriffsvektoren zu gefundenen Schwachstellen erklärt werden, und somit das Ausnutzen einer Sicherheitslücke vereinfacht wird.

2.3.4 Phase III

In der dritten Analysephase sollte versucht werden, ausgehend von den identifizierten Schwachstellen, anwendbare Sicherheitslücken zu finden. Hierfür wird die mit der CWE verwandte *Common Vulnerabilities and Exposures* (CVE)[28] eingesetzt, welche ebenfalls von der *MITRE Corporation* entwickelt wird. Ziel des CVE-Projekts ist die Erstellung einer einheitlichen, öffentlichen und globalen Datenbank für Sicherheitslücken. Hierfür wird jeder identifizierten Sicherheitslücke ein eindeutiger *CVE Identifier* (CVE ID) zugewiesen. Diese Zuweisung, und somit die Erstellung eines CVE-Eintrags, kann von einer *CVE Numbering Authority* (CNA) durchgeführt werden. Als CNA können sich Unternehmen, Organisationen und Projekte anmelden, um die Behandlung der betreffenden Sicherheitslücken selbst zu übernehmen und die Einträge für CVE zu kuratieren. Bei Meldung einer Sicherheitslücke zur Aufnahme in die CVE-Datenbank sollte daher stets die möglichst passende CNA-Stelle[29] benachrichtigt werden. Sollte ein betroffenes Projekt keine geeignete Anlaufstelle bieten, so ist es dennoch möglich die Sicherheitslücke an unabhängige sogenannte *CNAs of Last Resort* zu melden.

In der Dokumentation des CWE-Projekts existiert eine Schrittfolge zum Finden einer geeigneten CWE-Schwachstellenbeschreibung zu einer identifizierten CVE-Sicherheitslücke. Dies wird häufig eingesetzt, um ausgehend von einer gefundenen konkreten Sicherheitslücke in einem Softwareprojekt systematisch auf alle ähnlichen möglichen Mängel zu schließen. Es umfasst demnach lediglich das Verhalten *nach* dem Finden einer Sicherheitslücke. Für die in dieser Arbeit entwickelte Methodik ist daher das umgekehrte Verfahren, also das mögliche Finden einer passenden CVE-Sicherheitslücke zu einer identifizierten Schwachstelle, von Nöten. Zu beachten ist, dass dieser Schritt mithilfe der dargestellten Methode lediglich für öffentlich bekannte Komponenten funktioniert, da projektinterne Komponenten sich für gewöhnlich zum Zeitpunkt des Penetration-Tests noch in Entwicklung befinden.

Hierzu kann allerdings das als *Keyword-Search* umschriebene Verfahren in angepasster Weise umgekehrt angewendet werden.[30] Dementsprechend wird die Stichwortsuche auf die CVE-Datenbank angewendet. Hierbei relevante Informationen sind zunächst der in Phase I dokumentierte Name der Komponente, sowie Schlüsselworte zur in Phase II identifizierten Schwachstelle. Ein Beispiel sind die in Abbildung 2.7 dargestellten Ergebnis zur Suchanfrage mit den Stichworten `opensmtpd out-of-bounds`. Mit `opensmtpd` ist das konkrete Emailtransportprogramm und mit `out-of-bounds` ein Softwarefehler, der das Lesen oder Schreiben außerhalb des vorgesehenen Speicherbereichs erlaubt, gemeint. Für eine effizierentere und möglicherweise automatisierbare Methodik sei auf Kapitel 4.3 verwiesen. Für die in dieser Arbeit dargestellte Implementation fungiert die Keyword-Search als Form von „manuellem Suchen und Probieren“.


CVE List CNAs WGs Board About News & Blog NVD Go to for: CVSS Scores CPE Info

Search CVE List | Downloads | Data Feeds | Update a CVE Record | Request CVE IDs

TOTAL CVE Records: 157729

HOME > CVE > SEARCH RESULTS

Search Results

There are **7** CVE Records that match your search.

Name	Description
CVE-2020-8794	OpenSMTPD before 6.6.4 allows remote code execution because of an out-of-bounds read in mta_io in mta_session.c for multi-line replies. Although this vulnerability affects the client side of OpenSMTPD, it is possible to attack a server because the server code launches the client code during bounce handling.
CVE-2020-8793	OpenSMTPD before 6.6.4 allows local users to read arbitrary files (e.g., on some Linux distributions) because of a combination of an untrusted search path in makemap.c and race conditions in the offline functionality in smtpd.c.
CVE-2020-7247	smtp_mailaddr in smtp_session.c in OpenSMTPD 6.6, as used in OpenBSD 6.6 and other products, allows remote attackers to execute arbitrary commands as root via a crafted SMTP session, as demonstrated by shell metacharacters in a MAIL FROM field. This affects the "uncommented" default configuration. The issue exists because of an incorrect return value upon failure of input validation.
CVE-2020-35680	smtpd/lka_filter.c in OpenSMTPD before 6.8.0p1, in certain configurations, allows remote attackers to cause a denial of service (NULL pointer dereference and daemon crash) via a crafted pattern of client activity, because the filter state machine does not properly maintain the I/O channel between the SMTP engine and the filters layer.
CVE-2020-35679	smtpd/table.c in OpenSMTPD before 6.8.0p1 lacks a certain regfree, which might allow attackers to trigger a "very significant" memory leak via messages to an instance that performs many regex lookups.
CVE-2015-7687	Use-after-free vulnerability in OpenSMTPD before 5.7.2 allows remote attackers to cause a denial of service (crash) or execute arbitrary code via vectors involving req_ca_vrfy_smtp and req_ca_vrfy_mta.
CVE-2013-2125	OpenSMTPD before 5.3.2 does not properly handle SSL sessions, which allows remote attackers to cause a denial of service (connection blocking) by keeping a connection open.

Abbildung 2.7: Suchergebnisse der CVE (Quelle: [31], abgerufen am 28.07.2021)

Ist die Suche nach einer bereits bekannten Sicherheitslücke erfolgreich, so können weiterführenden Links gefolgt werden. Hierzu gehört für gewöhnlich ein Eintrag in der *National Vulnerability Database* (NVD)[32]. Genaueres hierzu ist in Kapitel 2.4.2 beschrieben. Der NVD-Eintrag bietet Informationen, wie einen *Common Vulnerability Scoring System*-Wert (CVSS), welcher dazu beitragen kann, eine Abschätzung nach Effektivität der Sicherheitslücke durch den Analysten durchzuführen. Damit ist eine Überlegung gemeint, welche Faktoren, wie die Schwere einer Sicherheitslücke und den Aufwand oder die Resourcen die zur Ausnutzung benötigt werden, gegeneinander abwägt. Außerdem kann es vorkommen, dass existierende Proof-of-Concepts im CVE-Eintrag referenziert werden. Ist dies der Fall, so sollte dieser Proof-of-Concept dem Testplan hinzugefügt werden.

Oft ist es jedoch möglich, dass kein öffentlich bekannter Eintrag in der CVE oder einer anderen Sammlung gefunden werden kann. Dies kann daran liegen, dass eine Sicherheitslücke noch nicht bekannt ist, oder noch nicht veröffentlicht wurde. Der Hauptgrund ist allerdings häufig der, dass in einem Softwareprojekt die Komponente noch in Entwicklung ist, und Mängel daher, wenn überhaupt, im internen Bugtracking-System des Entwicklers zu finden sind. In jedem Fall obliegt es dann dem Sicherheitsanalysten, eine Ausnutzung der Schwachstelle zu bewerkstelligen, beziehungsweise abzuschätzen, ob dies gewinnbringend ist und in absehbarer Zeit umgesetzt werden kann.

Zum Aufbau des Testplans in der Beispielimplementierung kann eine tabellarische Form genutzt werden. Diese sollte alle für den Test relevanten Elemente beinhalten. So ist beispielsweise eine Aufgliederung nach **Komponente – Gefahr – CWE – CVE – Exploit – Testbeschreibung** sinnvoll. Zu jeder zu testenden Gefahr sind somit alle benötigten Informaitionen ersichtlich. Ein Minimalbeispiel eines Testplans nach diesem Schema, ist in Tabelle A.1 zu sehen. Unter Angabe der CWE-Schwachstellen ist es möglich, eine Testbeschreibung, beziehungsweise ein Vorgehen zu beschreiben. Sollte, wie im Beispiel angegeben, eine Schwachstelle *CWE-319: Cleartext Transmission of Sensitive Information*[33] auftreten, so wäre als Testbeschreibung die Durchführung eines

Machine-in-the-Middle-Angriffs sinnvoll. Ist hingegen eine CVE-Sicherheitslücke oder bereits ein fertiger Exploit für eine Softwareversion bekannt, so sollte zunächst geprüft werden, ob diese ausgenutzt werden können. Der Grad der Konkretisierung der Testbeschreibung ist variabel und obliegt dem Analysten. Manche Tests können bereits im Vorfeld detailliert konzipiert werden, andere Tests unterliegen Umständen, welche erst während des Tests festgestellt werden können. Womöglich ist anschließend an die Analyse existierender Exploits ein Schritt, ähnlich der in Phase I.v durchgeführten Abschätzung, sinnvoll, um zu entscheiden, ob externe Komponenten weiteren Tests unterzogen werden sollen. Es sollte allerdings nicht davon ausgegangen werden, dass nur, weil für eine bestimmte Softwareversion eine CVE-Sicherheitslücke inklusive Exploit existiert, weitere Tests für diese Komponente ausgeschlossen werden.

2.4 Mögliche Alternativen

Das abstrakte Konzept der in dieser Arbeit entwickelten Methodik erlaubt es, die eingesetzten Taxonomien und Hilfsmodelle auszutauschen. In diesem Kapitel sollen daher nun einige mögliche Alternativen zu den in Kapitel 2.3 eingesetzten Werkzeugen dargestellt werden.

2.4.1 Alternative Gefahrenklassifikationen für Phase I

Neben der STRIDE-Klassifizierung existieren weitere Modelle, welche sich häufig in Bezug auf Einsatzgebiet und Abstraktionsgrad unterscheiden. Es sollte dementsprechend stets ein für den durchgeführten Test passendes Modell gewählt werden. Besitzt die zu untersuchende Struktur lediglich eine hardwarenahe Kommunikation, wie beispielsweise ein Treiberprogramm, so ist die Betrachtung von Gefahren für Webkomponenten nicht zielführend. Grundsätzlich kann jede Aufzählung oder Kategorisierung von Gefahren als Modell dienen, sofern diese für den Einsatzzweck geeignet sind und eine hinreichende Unterteilung der Gefahren bietet. Folgend sind einige Beispiele aufgeführt:

Grundschutzkompendium des Bundesamts für Sicherheit in der Informationstechnik Mit dem *Grundschutzkompendium* (GSK)[7] hat das deutsche *Bundesamt für Sicherheit in der Informationstechnik* (BSI) ein umfangreiches Rahmenwerk zur Gewährleistung eines hohen Sicherheitsstandards entwickelt. Bei erfolgreicher Umsetzung dieses Konzepts werden unter anderem auch alle Anforderungen an den internationalen Standard *ISO 27001*[34] erfüllt. Ein Bestandteil des GSK ist der Katalog „Elementare Gefährdungen“, welcher in Phase I eingesetzt werden kann. Die im GSK aufgeführten Gefährdungen sind als ganzheitliches System für Sicherheit im Unternehmensbereich ausgelegt und umfassen somit das gesamte Gefahrenspektrum, von physischem Zugriff und Naturkatastrophen bis hin zu Social Engineering und digitalen Eingriffen. Des

Weiteren bietet das Grundschutzkompendium zu jeder Gefährdung einen ausführlichen Beschreibungstext.

Aufgrund des hohen Detailgrads der Gefährdungsunterteilung, inklusive einiger Beispielszenarien in den Beschreibungen, kann sich das Grundschutzkompendium sehr gut eignen, um präzise Szenarien zu definieren und somit die Überführung zur Schwachstellendefinition in Phase II zu erleichtern.

Allerdings sind nicht immer alle Gefährdungen relevant. Gerade in Hinblick auf die moderne Landschaft der Webapplikationen sind Gefährdungen, wie „G 0.5 Naturkatastrophen“[7] oder „G 0.34 Anschlag“[7] vergleichsweise irrelevant. Je nach Umfang und Art des zur Untersuchung vorliegenden Projekts kann eine solch breite Aufstellung von Beispielgefahren allerdings von Vorteil sein. Ist beispielsweise Hardware oder physische Sicherheit betroffen, so wäre das STRIDE-Modell an vielen Stellen unzureichend. In einem solchen Fall sollte der Einsatz des Grundschutzkompendiums aufgrund der umfassenderen Abdeckung möglicher Gefahren anstelle des STRIDE-Modells in Betracht gezogen werden.

Open Threat Taxonomy Die von *Enclave Security* entwickelte *Open Threat Taxonomy*[35] bietet eine weitere alternative Gefahrentaxonomie. Der Aufbau ist hierbei hierarchisch nach verschiedenen Kategorien, wie *Physical Threats* oder *Tehcnical Threats* aufgebaut. Jede der Kategorien besitzt weiterhin eine Aufzählung möglicher Gefahren, inklusive eines *Threat Ratings*.

Die breite Fächerung der Kategorien erlaubt ein großes Spektrum an Einsatzmöglichkeiten. Auch die hierarchische Abstufung und das Ratingsystem erlauben es, die Taxonomie in jeder Abstraktionstiefe einzusetzen und optimal für eine Sicherheitsanalyse anzuwenden. Allerdings erschien die letzte Version bereits im Jahr 2015. Es ist somit mit immer größer werdendem Zeitabstand zur letzten Aktualisierung des Dokuments immer wahrscheinlicher, dass neue entwickelte Angriffstechniken und somit auch neu entstandene Gefahrenklassen nicht Bestandteil dieser Taxonomie sind. Auch ist das Projekt weitestgehend eigenständig und mangelt somit an Schnittstellen zu beispielsweise *MITREs* CWE- oder CVE-Projekt.

2.4.2 Alternative Sicherheitslückendatenbanken für Phase III

Im Bereich der Protokollierung von Sicherheitslücken wird MITREs CVE als globales System weitgehend anerkannt und von anderen Datenbanken als gemeinsamer Nenner eingesetzt. Häufig besitzen alternative Lösungen daher eine Referenz zur CVE.

National Vulnerability Database Eine Alternative ist die bereits in Kapitel 2.3.4 dargestellte *National Vulnerability Database* (NVD)[32] des *National Institue of Standards and Technology* (NIST). Diese Datenbank setzt zur eindeutigen Erkennung CVE-Identifikatoren ein, bietet jedoch ein breites Spektrum an weiterführenden Informationen. So ist jeder Sicherheitslücke ein *Common-Vulnerability-Scoring-System*-Wert (CVSS) zugeordnet. Dieser CVSS-Wert dient dazu, die Voraussetzungen und Auswirkungen einer Sicherheitslücke in standardisierter Form zusammenzufassen. In der aktuellsten Version 3.1 des Scoring-Systems werden hierfür verschiedene Basismetriken betrachtet.[36] Hierzu zählen:

- **Attack Vektor (AV)**: Der Angriffsvektor beschreibt, welche Art von Zugriff ein Angreifer benötigt (Netzwerk, Nahbereich, physikalisch, lokal)
- **Attack Complexity (AC)**: Komplexität des Angriffs – niedrig oder hoch.
- **Privileges Required (PR)**: Benötigte Berechtigungen – keine, niedrig oder hoch.
- **User Interaction (UI)**: Läuft der Angriff automatisiert ab oder sind Nutzerinteraktionen notwendig – benötigt oder nicht benötigt.
- **Scope (S)**: Beeinflusst der Angriff einen anderen Sicherheitskontext – ja oder nein.
- **Schutzziele Confidentiality (C), Integrity (I), Availability (A)**: Ist das jeweilige Schutzziel betroffen – nein, niedrig oder hoch.

Unter Berücksichtigung dieser Basismetriken wird anschließend ein CVSS-Basiswert ermittelt, welcher von 0 (harmlos) bis zu 10 (kritisch) reicht. Dieser kann anschließend mit weiteren Metriken unter Berücksichtigung von Zeit-, beziehungsweise verschiedener Umgebungsfaktoren weiter verfeinert werden.

Des Weiteren sind zugeordnete Schwachstellen der CWE aufgeführt, sowie weiterführende Referenzen. Für gewöhnlich zählen hierzu verfügbare Exploits, Proof-of-Concepts oder Links zu Informationsseiten des Herstellers. Abschließend sind mögliche betroffene Systemkonfigurationen im *Common Platform Enumeration*-Format (CPE)[37] angegeben. Diese Informationen können verwendet werden, um betroffene Systeme und Programmversionen schnell identifizieren zu können.

Die NIST NVD eignet sich demnach sehr gut als ergänzende Informationsquelle zu CVE, kann aber auch selbstständig eingesetzt werden. Der Einsatz der NIST NVD eignet sich allerdings nicht zur Analyse von Hardwarekomponenten, da hierfür keine Einträge in der Datenbank existieren.

Herstellerspezifische Datenbanken Manche Softwarehersteller besitzen eine eigene Datenbank zur besseren Verwaltung von Sicherheitslücken. Der Vorteil hierbei ist die Spezifizierung der Datenbank auf die Produktpalette des Herstellers und einer somit möglichen Anpassung an die Dokumentation vorhandener Sicherheitslücken. Außerdem kann so auch auf vorhandene Softwareupdates hingewiesen werden, um die Mängel zu beseitigen. Ein Beispiel hierfür sind die *Cisco Security Advisories*.[38] Diese werden genutzt, um produkt- und herstellerspezifische Informationen, wie betroffene Programmversionen, Patches oder Workarounds anzubieten. Häufig werden hierbei neben einem internen Identifikator auch Referenzen zu globalen Datenbanken, wie CVE oder CWE angeboten. Sofern eine solche herstellerspezifische Datenbank existiert, sollte diese nach Möglichkeit für genauere Informationen mit einbezogen werden. Anders als MITREs CVE oder der NIST NVD können diese Datenbanken Informationen zu betroffenen Hardwareprodukten bieten.

3 Fallbeispiel

In diesem Abschnitt soll die in Kapitel 2.3 dargestellte Methodik anhand eines Beispiels erläutert werden. Hierfür soll ein simples akademisches Beispiel dienen. Das Beispiel stellt den Aufbau einer Bibliotheksanwendung dar, welche dazu dient, dass Benutzer sich mit einem Benutzerkonto anmelden können, um dann auf ihr Konto Bücher ausleihen zu können.

3.1 Voraussetzungen

Das System besteht aus der Ausleihsoftware, welche als Web-Service betrieben wird und einem browserbasierten Nutzerclient. Zur Datenverwaltung kommuniziert die Webanwendung außerdem mit einem Datenbanksystem, welches sowohl die Nutzer-, als auch Bücherdaten verwaltet. Zur Annäherung des Beispiels an ein reelles Szenario sollen für die Komponenten folgende Spezifikationen angenommen werden: Die Webanwendung basiert auf der *PHP*-Skriptsprache[39]. Der eingesetzte Webserver ist ein *Apache* HTTP-Server[40] und unterstüzt für das gegebene Beispiel lediglich eine PHP-Version 5.3.3. Der Grund für die Auswahl dieser nach heutigen Standards gesehen veralteten Version ist die bessere Darstellung des Verhaltens der Testmethodik, aufgrund einer in dieser Version existierenden Sicherheitslücke. Die Buch- und Kundendatenbank ist mithilfe einer *MariaDB*-Datenbank[41] realisiert. MariaDB ist eine freie Abspaltung des *MySQL*-Projekts, welche hohen Wert auf Stabilität und Geschwindigkeit legt. Für das vorliegende Beispiel soll die aktuelle Programmversion 10.6 angenommen werden. Die Kommunikation zwischen dem Client eines Benutzers und dem Webserver erfolgt über eine *Transport-Layer-Security* (TLS) Verbindung, unter Einsatz der Protokollversion 1.3. Die Auswahl der Programme und Versionen ist arbiträr und dient lediglich der Veranschaulichung.

Das Szenario bestehend aus einer Webapplikation auf Grundlage von PHP, sowie einer SQL-Datenbank bildet ein typisches Schema für heutige Softwareentwicklung ab. Da die Komplexität der Analyse einer Architektur mit der bestehenden Methodik schnell anwächst, soll das Minimalbeispiel genügen, um alle notwendigen Schritte darzustellen.

3.2 Phase I: Erstellung des Threat-Models

Ein Überblick über das Zusammenspiel der Komponenten ist in Abbildung 3.1 dargestellt. Der Zugriff des Benutzers auf die Login- und Ausleihkomponenten erfolgt jeweils mithilfe des Webservers. Dies soll durch die perforierte Verbindungslinie dargestellt sein. Der Aufbau der Datenbanktabellen kann im weiteren Verlauf der Analyse mit Rücksicht

auf mögliche SQL-Injections relevant sein. In den ersten Analysephasen bilden dieser jedoch lediglich Datenstrukturen ab, welche für sich allein gesehen keine sicherheitstechnischen Auswirkungen bieten.

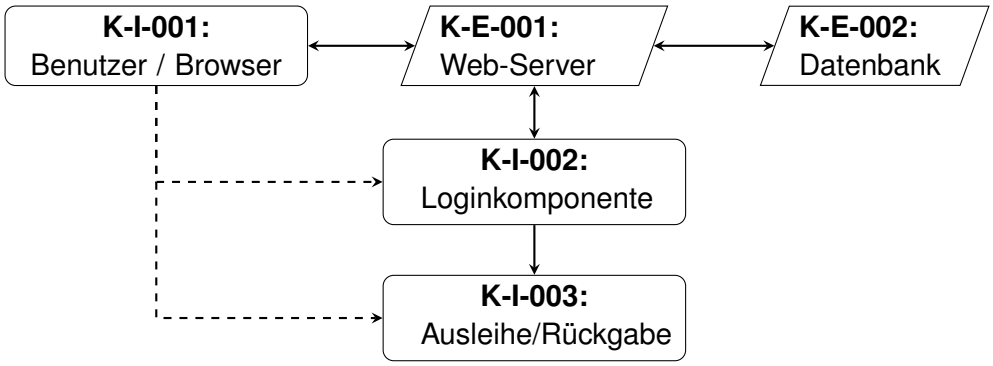


Abbildung 3.1: Darstellung der Komponenten im Fallbeispiel

Im ersten Schritt erfolgt die Erstellung des Threat-Models, inklusive der eindeutigen Nummerierung der Komponenten. Abbildung 3.1 kann hierbei als vollständiges Architekturmodell angesehen werden. Dementsprechend ergeben sich folgende Komponenten:

- `K-I-001` User inklusive Browserclient
- `K-I-002` Loginfunktion
- `K-I-003` Ausleihe/Rückgabe
- `K-E-001` Apache Webserver
- `K-E-002` MariaDB Datenbanksystem

Die Erstellung des Threat-Models zu den gegebenen Komponenten stellt sich folgendermaßen dar:

Kommunikation `K-I-001` $\rightarrow$ `K-I-002` **via** `K-E-001`

- Spoofing
  - `G-001` Ein Clientgerät wird von mehreren Benutzern verwendet
  - `G-002` Social-Engineering oder Phishing-Angriffe zum Erlangen von Logindaten
  - `G-003` Diebstahl eines Geräts mit gespeicherten Logindaten
  - `G-004` Ein Benutzer greift auf Daten anderer Benutzer zu

- Tampering
  - G-005 Datenmanipulation durch präparierte Nutzereingaben
- Information Disclosure
  - G-006 MITM: Mitschneiden von Logindaten
- Denial of Service
  - G-007 Überlasten der Loginkomponente
- Elevation of Privilege
  - G-008 Unbefugtes Anmelden als Administrator
  - G-009 Erlangen von Remote-Code-Execution auf K-E-001
  - G-010 Erweitern der Berechtigungen eines Nutzerkontexts durch Codeinjektion

Kommunikation K-I-002 → K-I-001 **via** K-E-001

- Spoofing
  - G-011 MITM: Angreifer kann sich als Authentifizierungsseite ausgeben
- Information Disclosure
  - G-012 MITM: Mitschneiden von Sitzungsidentifikation
  - G-013 Preisgabe von Informationen durch Debug- und Fehlernachrichten
  - G-014 Erlangen von Versionsinformationen von K-E-001

Kommunikation K-I-001 → K-I-003 **via** K-E-001

- Spoofing
  - G-015 Ausleihen als anderer Benutzer
  - G-016 Rückgabe als anderer Benutzer
- Tampering
  - G-017 Eingabefelder mit präparierten Nachrichten angreifen
- Repudiation
  - G-018 Ausleihe ohne eindeutig zuzuordnenden Nutzer durch Entfernen einer Nutzerzuordnung

- Information Disclosure
  - G-019 MITM: Mitschneiden von Informationen
  - G-020 Extrahieren von sensiblen Informationen aus der Datenbank mithilfe manipulierter Anfragen
- Denial of Service
  - G-021 Überladen der Ausleihkomponente mit Anfragen
  - G-022 Blockieren aller Bücher durch Ausleihe
- Elevation of Privilege
  - G-023 Nutzen von Administratorenschnittstellen als normaler Nutzer
  - G-024 Erlangen von Remote-Code-Execution auf K-E-001

Kommunikation K-I-003 $\rightarrow$ K-I-001 **via** K-E-001

- Tampering
  - G-025 MITM: Manipulation von Buchinformationen aus der Datenbank
- Information Disclosure
  - G-026 MITM: Mitschneiden von Buch- und Anwenderinformationen
  - G-027 MITM: Mitschneiden von Sitzungsidentifikation
  - G-028 Preisgabe von Informationen durch Debug- und Fehlernachrichten
  - G-029 Erlangen von Versionsinformationen von K-E-001
- Elevation of Privilege
  - G-030 Erweitern der Berechtigungen eines Nutzerkontexts durch Codeinjektion

Da nicht für jede STRIDE-Gefahrenklasse aller Beispiele ein Gefahrenszenario möglich ist, sind nicht immer alle Gefahrenklassen im Threat-Model vorhanden. Insbesondere die Gefahr *Repudiation*, beziehungsweise der *non-repudiability*, tritt vergleichsweise selten auf, da derartige Gefahren häufig in Verbindung mit *Spoofing* auftreten. Dementsprechend werden diese Gefahren dann nur einer der Kategorien zugeordnet. In G-009 und G-024 ist die Gefahr der *Remote-Code-Execution* genannt. Oriyano und Shimonski bezeichnen dies als eine Sicherheitslücke, bei der es einem Angreifer möglich ist, eigene Programme auf einem Server auszuführen.[42] Die mit „MITM“ gekennzeichneten Gefahren stellen jeweils Angriffe auf Grundlage einer Machine-in-the-Middle-Position dar.

3.3 Phase I.v: Entfernen umgangener Gefahren

Zur Demonstration des in Phase I.v stattfindenden Realitätsabgleichs werden die eingesetzten Programme und Versionen in Betracht gezogen. Der Einsatz einer verschlüsselten TLS-Kommunikation hat zur Folge, dass Mitschnitte höchstens an den Kommunikationsendpunkten selbst, nicht jedoch auf dem Verbindungskanal stattfinden können. Da die Serverinfrastruktur als vertrauenswürdig eingeschätzt wird, werden die als Machine-in-the-Middle gekennzeichneten Szenarien als unwahrscheinlich klassifiziert. Dementsprechend werden diese Gefahren im optionalen Schritt für die weitere Untersuchung ausgenommen.
Außerdem findet eine direkte Kommunikation des Benutzers mit der Datenbank nicht statt. Da für dieses Beispiel ein übliches Datenbanksystem, wie die angegebene aktuelle Version des MariaDB-Datenbankservers angenommen wird, wäre eine Untersuchung dieser externen Software selbst, mit Blick auf Verhlältnismäßigkeit der Untersuchung[3], nicht sinnvoll. Somit wird auch die Komponente `K-E-002` von weiteren Untersuchungen ausgenommen. Lediglich durch eine Erweiterung des Zugriffs auf `K-E-001` könnte eine direkte Kommunikation mit dem Datenbanksystem möglich sein. In diesem Fall wäre allerdings eine Reevaluierung des Threat-Models unter Einbezug eines Zugriffs des Angreifers aus der Position des Webservers nötig. Hierbei sollten demnach auch Überlegungen in Bezug auf bestehende *Trust-Boundaries*, wie in Kapitel 2.3.1 beschrieben, angestellt werden. Zu beachten ist jedoch, dass beispielsweise die Gefahr „`G-018` Extrahieren von sensiblen Informationen aus der Datenbank mithilfe manipulierter Anfragen“ immer noch realistisch ist, da dies auf mögliche Logikfehler in `K-I-003` und nicht auf Programmfehler in der Datenbank zurückzuführen wäre.

Folgend sind die nach Abschluss der optionalen Evaluierungsphase I.v zu dokumentierenden Komponenten sowie zugehörige Gefahrenszenarien aufgeführt.

`K-I-001`

- `G-001` Ein Clientgerät wird von mehreren Benutzern verwendet
- `G-002` Social-Engineering oder Phishing-Angriffe zum Erlangen von Logindaten
- `G-003` Diebstahl eines Geräts mit gespeicherten Logindaten
- `G-004` Ein Benutzer greift auf Daten anderer Benutzer zu
- `G-005` Datenmanipulation durch präparierte Nutzereingaben

[3] siehe Kapitel 4.1

K-I-002

- G-007 Überlasten der Loginkomponente
- G-008 Unbefugtes Anmelden als Administrator

K-I-003

- G-015 Ausleihen als anderer Benutzer
- G-016 Rückgabe als anderer Benutzer
- G-017 Eingabefelder mit präparierten Nachrichten angreifen
- G-018 Ausleihe ohne eindeutig zuzuordnenden Nutzer durch Entfernen einer Nutzerzuordnung
- G-020 Extrahieren von sensiblen Informationen aus der Datenbank mithilfe manipulierter Anfragen
- G-021 Überladen der Ausleihkomponente mit Anfragen
- G-022 Blockieren aller Bücher durch Ausleihe
- G-023 Nutzen von Administratorenschnittstellen als normaler Nutzer

K-E-001

- G-009 / G-024 Erlangen von Remote-Code-Execution
- G-010 / G-030 Erweitern der Berechtigungen eines Nutzerkontexts durch Codeinjektion
- G-013 / G-028 Preisgabe von Informationen durch Debug- und Fehlernachrichten
- G-014 / G-029 Erlangen von Versionsinformationen von K-E-001

3.4 Phase II: Mapping gefundener Gefahren zu Weaknesses

Den in Phase I entwickelten Gefahrenszenarios wird nun eine passende Schwachstellenkategorie nach CWE zugewiesen. Sofern dies für ein Szenario zutreffend ist, kann es möglich sein, dass mehrere CWE-Beschreibungen nötig sind, um die Schwachstelle zu beschreiben. Außerdem ist zu bedenken, dass aufgrund des hierarchischen Aufbaus der CWE-Datenbank viele verwandte Einträge für eine Gefahr passend sein kön-

nen. Häufig ist es in solchen Fällen möglich, über Kind- und Elternbeziehungen des CWE-Eintrags weitere passende Schwachstellen zu identifizieren. Zum Beispiel besteht für Gefahr „G-021 Überladen der Ausleihkomponente mit Anfragen“ die Schwachstelle abstrakt betrachtet daraus, dass je nach Konfiguration, unendlich Anfragen an die Komponente gestellt werden könnten. Eine solche Situation passt auf die Schwachstelle *CWE-410: Insufficient Resource Pool*[43], welche beschreibt, dass der Resourcenpool, in diesem Falle Rechenzeit und Kommunikationsbandbreite, keine ausreichende Größe besitzt. Dies würde dann eine klassische Denial-of-Service-Situation nach sich ziehen. Diese Schwachstelle ist direkt verwandt mit *CWE-400: Uncontrolled Resource Consumption*[44], welche eine Oberkategorie für verschiedene Mechanismen zur Resourcenkontrolle darstellt. Weitere verwandte Einträge zu dieser Schwachstelle wären beispielsweise *CWE-664: Improper Control of a Resource Through its Lifetime*[45] oder *CWE-770: Allocation of Resources Without Limits or Throttling*[46]. Prinzipiell wäre es möglich, alle verwandten Schwachstellen aufzuführen, was häufig jedoch nicht nötig ist. Im Fallbeispiel wurde jeweils versucht, mindestens eine abstraktere Oberkategorie, sowie eine speziellere Schwachstelle zuzuordnen. Besteht die Gefahr aus zwei nicht direkt verwandten Schwachstellenarten, so wurde versucht, für jeden Aspekt mindestens eine Schwachstellenbeschreibung zu finden.

Die so entstehenden Überlegungen zu Schwachstellen und damit verbundenen möglichen Angriffsvektoren bilden die Grundlage für den Testplan.

K-I-001

- G-001 Ein Clientgerät wird von mehreren Benutzern verwendet
  - CWE-284: Improper Access Control
  - CWE-283: Unverified Ownership
- G-002 Social-Engineering oder Phishing-Angriffe zum Erlangen von Logindaten
  - CWE-308: Use of Single-factor Authentication
  - CWE-654: Reliance on a Single Factor in a Security Decision
- G-003 Diebstahl eines Geräts mit gespeicherten Logindaten
  - CWE-922: Insecure Storage of Sensitive Information
  - CWE-312: Cleartext Storage of Sensitive Information

K-I-002

- G-004 Ein Benutzer greift auf Daten anderer Benutzer zu
  - CWE-200: Exposure of Sensitive Information to an Unauthorized Actor
  - CWE-284: Improper Access Control
- G-005 Datenmanipulation durch präparierte Nutzereingaben
  - CWE-20: Improper Input Validation
  - CWE-74: Improper Neutralization of Special Elements in Output Used by a Downstream Component ('Injection')
- G-007 Überlasten der Loginkomponente
  - CWE-400: Uncontrolled Resource Consumption
  - CWE-410: Insufficient Resource Pool
- G-008 Unbefugtes Anmelden als Administrator
  - CWE-287: Improper Authentication
  - CWE-266: Incorrect Privilege Assignment

K-I-003

- G-015 Ausleihen als anderer Benutzer
  - CWE-285: Improper Authorization
  - CWE-290: Authentication Bypass by Spoofing
- G-016 Rückgabe als anderer Benutzer
  - CWE-285: Improper Authorization
  - CWE-290: Authentication Bypass by Spoofing
- G-017 Eingabefelder mit präparierten Nachrichten angreifen
  - CWE-20: Improper Input Validation
  - CWE-74: Improper Neutralization of Special Elements in Output Used by a Downstream Component ('Injection')
- G-018 Ausleihe ohne eindeutig zuzuordnenden Nutzer durch Entfernen einer Nutzerzuordnung
  - CWE-285: Improper Authorization
  - CWE-862: Missing Authorization
- G-020 Extrahieren von sensiblen Informationen aus der Datenbank mithilfe manipulierter Anfragen

        - CWE-74: Improper Neutralization of Special Elements in Output Used by a Downstream Component (’Injection’)
        - CWE-89: Improper Neutralization of Special Elements used in an SQL Command (’SQL Injection’)
    - G-021 Überladen der Ausleihkomponente mit Anfragen
        - CWE-400: Uncontrolled Resource Consumption
        - CWE-410: Insufficient Resource Pool
    - G-022 Blockieren aller Bücher durch Ausleihe
        - CWE-770: Allocation of Resources Without Limits or Throttling
    - G-023 Nutzen von Administratorenschnittstellen als normaler Nutzer
        - CWE-266: Incorrect Privilege Assignment
        - CWE-269: Improper Privilege Management
        - CWE-274: Improper Handling of Insufficient Privileges

K-E-001

    - G-009 / G-024 Erlangen von Remote-Code-Execution
        - CWE-20: Improper Input Validation
        - CWE-438: Behavioral Problems
    - G-010 / G-030 Erweitern der Berechtigungen eines Nutzerkontexts durch Codeinjektion
        - CWE-250: Execution with Unnecessary Privileges
        - CWE-74: Improper Neutralization of Special Elements in Output Used by a Downstream Component (’Injection’)
    - G-013 / G-028 Preisgabe von Informationen durch Debug- und Fehlernachrichten
        - CWE-209: Generation of Error Message Containing Sensitive Information
        - CWE-1295: Debug Messages Revealing Unnecessary Information
    - G-014 / G-029 Erlangen von Versionsinformationen
        - CWE-200: Exposure of Sensitive Information to an Unauthorized Actor

3.5 Phase III: Vulnerabilitys und Testplan

Letztlich wird die dritte Phase zum Prüfen aller externen Komponenten auf bekannte Vulnerabilitys durchgeführt und entweder diese gefundenen Sicherheitslücken inklusive eventueller Exploits oder mögliche Tests für die übrigen Schwachstellen im Testplan festgehalten. Dieser kann dann methodisch abgearbeitet werden und die Ergebnisse in einem abschließenden Bericht für einen Auftraggeber festgehalten werden. Allerdings können auch bereits die in Phase II gesammelten Informationen zu möglichen Schwachstellen der Komponenten für Entwickler von Nutzen sein, um gefährdendes Verhalten zu verhindern. Ein möglicher Testplan nach Schritt III für das Beispielszenario ist im Anhang als Tabelle A.2 dargestellt. Der Bereich „Testbeschreibung“ dient hierbei als Zusammenfassung von zu unternehmenden Schritten, um die Software auf Anfälligkeit gegen die entdeckten Schwachstellen zu prüfen. Für manche Schwachstellen ist das Vorgehen offensichtlich. So ist für „`G-003` Diebstahl eines Geräts mit gespeicherten Logindaten“ ein logischer Schritt zu testen, ob und in welcher Form sensible Daten auf dem Gerät hinterlegt sind. Testszenarien, wie „`G-007` Unbefugtes Anmelden als Administrator“ sind hingegen komplexerer Natur. Zur Durchführung eines solchen Tests sind Werkzeuge, wie das von *PortSwigger Ltd.* entwickelte *Burp*[47] nötig. Burp kann dazu genutzt werden, Web-Anfragen abzufangen und zu verändern. Somit kann beispielsweise versucht werden, die bei der Anmeldung zugewiesene Rolle zu manipulieren und dem Benutzer erweiterte Berechtigungen zuzuweisen. Dies kann vorkommen, wenn die Berechtigungsprüfung lediglich clientseitig, beziehungsweise vom Server fehlerhaft durchgeführt wird. Da auch alternative Werkzeuge zu Burp eine derartige Manipulation durchführen können, sollte die Testbeschreibung stets unabhängig von konkreten Werkzeugen beschrieben werden. Dementsprechend ergibt sich die in Tabelle A.2 dargestellte Beschreibung, dass clientseitig Flaggen zur unbefugten Erweiterung der Nutzerrechte übergeben werden können.
Nach einem ähnlichen Vorgehen wurde für die anderen Testbeschreibungen des Testplans verfahren.

Die Entwicklung der Testbeschreibung zur Prüfung der Schwachstellen ist demnach dem Analysten überlassen. Je nach persönlicher Vorgehensweise sollte für die Gefahr, beziehungsweise Testen der Schwachstelle, ein mögliches Vorgehen beschrieben werden. Mithilfe dieser Beschreibung sollte es möglich sein, unabhängig von Testvorgehen und Werkzeug eines Analysten die Schwachstelle hinreichend zu testen.

In einem reellen Untersuchungsumfeld wäre es allerdings denkbar, neben den in Tabelle A.2 dargestellten Feldern noch weitere Beschreibungsfelder zum Testplan hinzuzufügen. So könnten Informationen über Erfolg oder Misserfolg eines Tests in Hinblick auf Re-Tests sinnvoll sein.

3.6 Evaluation

Der Testplan dient als Endprodukt der Methodik. Der so entstandene Plan umfasst für den Analysten alle relevanten Informationen in einem Überblick. Für den Analysten ist somit ersichtlich, welche Gefahr mit welchen Schwachstellen zusammenhängt und welche möglichen Testschritte einzuleiten sind. Zusammen mit dem Architekturmodell des Threat-Models ist es möglich, das Zusammenspiel der Komponenten zu erkennen und einen geordneten Test durchzuführen. Grundsätzlich kann man somit sagen, dass ein ordnungsgemäßer Test anhand der entwickelten Modelle und Daten durchgeführt werden kann.

Am Beispiel ist jedoch zu sehen, dass bei sorgfältiger Durchführung der Methodik, selbst bei einem Minimalbeispiel die Komplexität und der Dokumentationsaufwand rasch anwächst. Nimmt man an, dass im Schnitt ein Gefahrenszenario pro STRIDE-Gefahrenklasse ersichtlich ist und eine durchschnittliche Durchführungszeit pro Test von etwa zwei Stunden eingeplant wird, so würde dies für die vier Komponenten (ohne Datenbanksystem) im Fallbeispiel bereits eine reine Testzeit von 48 Arbeitsstunden ergeben. Hierzu müssten außerdem die Zeit zum Erstellen des Testplans, sowie das abschließende Reporting addiert werden.

Des weiteren ist zu beachten, dass die Testbeschreibungen nach der jeweils präferierten Methodik des Analysten entwickelt sind. Um den Schritt der Entwicklung der einzelnen Testbeschreibungen zu vereinheitlichen, wäre der Einsatz einer einheitlichen Form für Angriffs-, beziehungsweise Teststrategien denkbar. Die in Kapitel 2.3.3 erwähnte CAPEC-Datenbank[27] könnte hierzu eingesetzt werden.

4 Fazit

In diesem abschließenden Kapitel sollen einige Überlegungen zum Einsatz der Methodik unter Realbedingungen dargestellt werden. Hierbei soll auch eine Zusammenfassung und Auswertung stattfinden und die Forschungsfrage beantwortet werden.

4.1 Verhältnismäßigkeit und Abschätzung

Grundsätzlich kann angenommen werden, dass so gut wie jedes Softwareprojekt Schwachstellen aufweist, die es lediglich auszunutzen gilt.[48] Zur Risikoeinschätzung werden daher häufig zwei Kriterien einbezogen: Der Aufwand eines Angriffs, sowie die verfügbaren Mittel um einen solchen durchzuführen. So ist es zum Beispiel möglich mithilfe eines *Brute-Forcing*-Angriffs jedes Passwort zu erraten, vorausgesetzt es wurden keine weiteren Sicherheitsmaßnahmen implementiert. Mihailescu und Nita[49] beschreiben einen Brute-Forcing-Angriff als den Versuch viele Passwörter oder Passphrasen auszuprobieren, mit dem Ziel das richtige zu erraten. Jedoch stellt sich bei hinreichender Länge des Passworts die Frage, ob dies in einem relevanten Zeitraum umsetzbar ist.

Da Penetration-Tests und Sicherheitsanalysen für gewöhnlich auf einen Zeitraum begrenzt sind, oft lediglich auf einen Raum von wenigen Wochen, obliegt der analysierenden Partei eine notwendige Aufwandsabschätzung. Der in dieser Arbeit dargestellte Algorithmus dient dazu, mithilfe eines iterativen Prozesses ein möglichst breites Spektrum an Schwachstellen und Sicherheitslücken ausfindig zu machen. Auch wird für eine theoretische Vollständigkeit verlangt, für jede Schwachstelle und Sicherheitslücke einen Exploit zu entwickeln. In der Realität ist dies oft nicht umsetzbar, da viele der Prozesse eine nicht unerhebliche Entwicklungszeit in Anspruch nehmen.

Es obliegt daher dem Analysten, eine Abschätzung hinsichtlich mehrerer Parameter zu treffen. So sollte zunächst, sofern möglich, eine Abstufung nach Wichtigkeit, also Schwere der Schwachstelle, getroffen werden, um möglichst lohnenswerte Lücken zu entdecken und zu schließen. Doch auch dies sollte gegen Entwicklungs-/Durchführungszeit abgewogen werden. Beispielsweise wäre es zwar sehr lohnenswert einen Administratorenzugang mittels Brute-Forcing zu ermitteln, jedoch kann dies bei Passphrasen von zwei Worten, inklusive Sonderzeichen und Zahlen bereits mehrere Jahre dauern. Eine solche Abstufung findet in dieser Arbeit nicht statt, da das Konzept den theoretischen Optimalfall betrachtet, in dem alle Schritte für alle Komponenten vollständig durchgeführt werden, und daher eine Selektion nicht stattfindet. Zur Unterstützung dieser Abwägung kann eine *Priorisierungsmatrix*, angelehnt an eine klassische *Risikomatrix*[50], eingesetzt werden. Diese ist in Abbildung 4.1 zu sehen. Dabei werden die beiden Faktoren *Aufwand* und *Nutzen* gegenüber gestellt, um somit eine sinnvolle

Priorisierung zu ermöglichen. Ist ein Test schnell und einfach durchzuführen und birgt zudem ein großes Potenzial eine kritische Sicherheitslücke zu offenbaren, so sollte diesem Test eine hohe Priorität zugemessen werden. Benötigt der Test jedoch mehrere Voraussetzungen, möglicherweise lange Laufzeiten für Analysewerkzeuge und bietet zudem nur einen geringen Nutzen, so sollte dieser Test niedrig priorisiert werden. Sind alle höher priorisierten Tests abgeschlossen, und ist noch Zeit für die Durchführung der niedriger priorisierten Tests übrig, können diese im Anschluss durchgeführt werden. Grundsätzlich ist demnach zu sagen, je weiter sich ein Test in die linke obere Ecke der Matrix einordnen lässt, desto wichtiger ist die Durchführung.

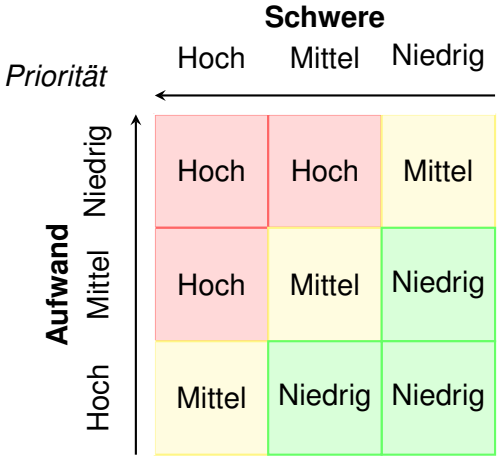


Abbildung 4.1: Priorisierungsmatrix

Die in der Arbeit dargestellte Methodik sollte als Modell und Konzepthilfestellung zur reproduzierbaren Testplanung eingesetzt werden, jedoch muss im Realfall oft die Verhältnismäßigkeit der Analyse der einzelnen Komponenten geprüft werden.

4.2 Diskussion

Die in dieser Arbeit entwickelte Methodik konnte bisher in der Praxis noch keine, beziehungsweise lediglich unter Laborbedingungen, Anwendung finden. Ob das Konzept mit den ausgewählten Hilfsmodellen demnach im Realfall einen entscheidenen Vorteil bringt, ist daher noch unklar. Gerade auch in Hinblick auf die in Kapitel 4.1 genannten Beschränkungen des Realfalls bleibt es abzusehen, ob ein Einsatz in der Praxis von Wert sein wird.

Allerdings gibt das dargelegte Vorgehen einen ganzheitlichen Rahmen, welcher auf Sicherheitsuntersuchungen angewendet werden kann. Die Forschungsfrage, ob die Erstellung eines solchen Modells auf Grundlage eines Threat-Modelling-Ansatzes und

existierender Hilfsmodelle möglich ist, ist demnach zu bejahen. Sollten alle eingesetzten Hilfsmodelle und Werkzeuge gut miteinander interagieren können und eine Vollständigkeit der Ergebnisse in den einzelnen Zwischenphasen ermöglichen, so erfüllt das Konzept tatsächlich die Ansprüche der Reproduzierbarkeit, möglichst hohe Vollständigkeit und Planbarkeit für Penetration-Tests.

Eines der Hauptprobleme der Methodik bietet jedoch der hohe Variationsgrad im realen Anwendungsfall. Im angegebenen akademischen Minimalbeispiel war die Umsetzung der Methodik aufgrund der einfachen Zusammenhänge der verschiedenen Komponenten gut anwendbar. Allerdings ist zu erwarten, dass unter Realbedingungen eine praktisch unendlich große Variabilität der Kommunikationsschnittstellen anzutreffen ist. Es ist daher sehr wahrscheinlich, dass für viele Anwendungsfälle mindestens eine Anpassung der eingesetzten Frameworks (STRIDE, CWE, CVE, ...), beziehungsweise eine Überführung in ein kompatibles Architekturmodell nötig ist.
Des Weiteren ist ein sehr schnell steigender Komplexitäts- und Protokollierungsaufwand bei der Untersuchung größerer Systeme zu beachten. In solchen Fällen sollten Überlegungen zur Vereinfachung von Teilschritten angestellt werden, um die entstehende Datenmenge handhabbar zu halten und gleichzeitig der Verhältnismäßigkeit gerecht zu werden.

4.3 Ausblick

Das in der Arbeit entwickelte Konzept bietet an vielen Stellen die Möglichkeit zur Vertiefung und Erweiterung durch weitere Konzepte. So ist beispielsweise eine problemlose Abbildung zwischen STRIDE und CWE, bzw. CWE und CVE zum Zeitpunkt der Arbeit nicht möglich. Zwar existieren Ansätze und Versuche ein einheitliches Verfahren für die Abbildung zwischen STRIDE und CWE zu bewerkstelligen, allerdings basieren existierende Konzepte mindestens auf weiteren Kompatibilitätsschichten oder sind laut Anne Honkaranta et al. für einen Einsteiger unzureichend, um ein eine effektive Abbildung durchzuführen.[51] Denkbare weitere Forschungsschritte könnten demnach die Entwicklung eines sinnvolleren Mappings, beziehungsweise das Finden oder Entwickeln von kompatibleren Taxonomien sein. Obgleich ein Mapping grundsätzlich möglich ist, so verlangt es nach dem derzeitigen Stand stets menschliche Interaktion um ein für den Sachverhalt zutreffendes Mapping durchzuführen. Um das finale Ziel einer vollautomatischen vollständigen Durchführung von Penetration-Tests mithilfe der in dieser Arbeit entwickelten Methodik zu erreichen, ist die Lösung dieser Problemstellung somit notwendig.

Sollte während der Durchführung die Vergabe und Arbeit mit IDs sorgfältig durchgeführt werden, so wäre auch die Entwicklung einer einheitlichen Datenstruktur, sowie die Überführung in eine Datenbankform denkbar. Durch eine ordnungsgemäße Aufteilung der verschiedenen ID-Arten kann so ein strukturiertes Datenformat entwickelt werden, welches bei Re-Tests zur einfachen Nachvollziehbarkeit genutzt werden kann. Auch bei einer potentiellen visuellen Darstellung wäre dieser Schritt hilfreich. Eventuell wäre sogar eine globale Vereinheitlichung (ähnlich CVE) von Tests für gegebene gleiche Komponenten oder bestimmte Architekturkonstellationen denkbar.

Schlussendlich könnte die konkrete Definition weiterer Phasen, wie eine genauere Beschreibung des Aufbaus des Testplans oder der Dokumentation, zu einem erweiterten ganzheitlichen Konzept führen.

Anhang A: Tabellen

Komponente	Gefahr	CWEs	CVEs	Exploit	Testbeschreibung
K-E-001					
	G-001 Erlangen von Remote-Code-Execution	CWE-125: Out-of-bounds Read	CVE-2020-8794	NVD-NIST	Prüfen der Softwareversion auf Anfälligkeit für gegebene oder weitere Sicherheitslücken
[...]					
K-I-004					
	G-007 Mitschneiden von Logininformationen	CWE-319: Cleartext Transmission of Sensitive Information	-	-	Machine-in-the-Middle-Position testen

Tabelle A.1: Minimalbeispiel Testplan

Komponente	Gefahr	CWEs	CVEs	Exploit	Testbeschreibung
K-I-001					
	G-001 Ein Clientgerät wird von mehreren Benutzern verwendet	CWE-284: Improper Access Control, CWE-283: Unverified Ownership	-	-	Prüfen, wie sich der Client bei mehreren Bentzern verhält. Ist ein wechseln des Benutzers möglich?
	G–002 Social-Engineering oder Phishing-Angriffe zum Erlangen von Logindaten	CWE-308: Use of Single-factor Authentication, CWE-654: Reliance on a Single Factor in a Security Decision	-	-	Existiert eine 2-Faktor-Authentifizierung, um Diebstahl von Logindaten vorzubeugen? Ist ein ändern der Daten möglich?
	G–003 Diebstahl eines Geräts mit gespeicherten Logindaten	CWE-922: Insecure Storage of Sensitive Information, CWE-312: Cleartext Storage of Sensitive Information	-	-	Prüfen, ob und wie sensible Informationen auf Clientgerät gespeichert werden.
K–I–002					
	G–004 Ein Benutzer greift auf Daten anderer Benutzer zu	CWE-200: Exposure of Sensitive Information to an Unauthorized Actor, CWE-284: Improper Access Control	-	-	Abfragen von Daten anderer Benutzer mit Berechtigungen des eigenen Benutzerkontos durchführen.

	G–005 Datenmanipulation durch präparierte Nutzereingaben	CWE-20: Improper Input Validation, CWE-74: Improper Neutralization of Special Elements in Output Used by a Downstream Component (’Injection’)	-	-	Prüfen auf mögliche XSS oder andere Injection Angriffe.
	G–007 Überlasten der Loginkomponente	CWE-400: Uncontrolled Resource Consumption, CWE-410: Insufficient Resource Pool	-	-	Prüfen von möglichen vorhandenen Anti-DoS-Maßnahmen.
	G–008 Unbefugtes Anmelden als Administrator	CWE-287: Improper Authentication, CWE-266: Incorrect Privilege Assignment	-	-	Prüfen, ob beim Loginvorgang clientseitige Flags zur unbefugten Authentifizierung als Administrator übergeben werden können.
K-I-003					
	G–015 Ausleihen als anderer Benutzer	CWE-285: Improper Authorization, CWE-290: Authentication Bypass by Spoofing	-	-	Prüfen, ob es möglich ist, Ausleihanfragen zu manipulieren und somit als ein anderer Benutzer Ausleihen zu tätigen.

G–016 Rückgabe als anderer Benutzer	CWE-285: Improper Authorization, CWE-290: Authentication Bypass by Spoofing	-	-	Prüfen, ob es möglich ist, Rückgaben über einen anderen Benutzer zu erledigen, als die Ausleihe getätigt wurde.
G–017 Eingabefelder mit präparierten Nachrichten angreifen	CWE-20: Improper Input Validation, CWE-74: Improper Neutralization of Special Elements in Output Used by a Downstream Component (’Injection’)	-	-	Prüfen auf mögliche XSS oder andere Injection Angriffe.
G–018 Ausleihe ohne eindeutig zuzuordnenden Nutzer durch Entfernen einer Nutzerzuordnung	CWE-285: Improper Authorization, CWE-862: Missing Authorization	-	-	Prüfen, ob eine Ausleihe ohne angegebenen Nutzer möglich ist.
G–020 Extrahieren von sensiblen Informationen aus der Datenbank mithilfe manipulierter Anfragen	CWE-74: Improper Neutralization of Special Elements in Output Used by a Downstream Component (’Injection’), CWE-89: Improper Neutralization of Special Elements used in an SQL Command (’SQL Injection’)	-	-	Prüfen auf mögliche SQL-Injections oder Ähnliches um sensible Informationen aus der Datenbank zu erhalten.

	G–021 Überladen der Ausleihkomponente mit Anfragen	CWE-400: Uncontrolled Resource Consumption, CWE-410: Insufficient Resource Pool	-	-	Prüfen von möglichen vorhandenen Anti-DoS-Maßnahmen.
	G–022 Blockieren aller Bücher durch Ausleihe	CWE-770: Allocation of Resources Without Limits or Throttling	-	-	Prüfen von Ausleihlimits eines Benutzers.
	G–023 Nutzen von Administratorenschnittstellen als normaler Nutzer	CWE-266: Incorrect Privilege Assignment, CWE-269: Improper Privilege Management , CWE-274: Improper Handling of Insufficient Privileges	-	-	Prüfen, ob ein unberechtigter Nutzer Zugriff auf Administratorenschnittstellen besitzt.
K-E-001					
	G–009 / G–024 Erlangen von Remote-Code-Execution	CWE-20: Improper Input Validation, CWE-438: Behavioral Problems	CVE-2012-1823	ExploitDB 29290	Prüfen der Softwareversion auf gegebene oder mögliche weitere Sicherheitslücken.

G–010 / G–030 Erweitern der Berechtigungen eines Nutzerkontexts durch Codeinjektion	CWE-250: Execution with Unnecessary Privileges, CWE-74: Improper Neutralization of Special Elements in Output Used by a Downstream Component (’Injection’)			Prüfen auf mögliche Skriptinjektionen zur Erweiterung der Nutzerberechtigungen.
G–013 / G–028 Preisgabe von Informationen durch Debug- und Fehlernachrichten	CWE-209: Generation of Error Message Containing Sensitive Information , CWE-1295: Debug Messages Revealing Unnecessary Information	-	-	Prüfen der Webserverkonfiguration, ob Diagnose- oder sensible Fehlernachrichten an den Nutzer ausgegeben werden.
G–014 / G–029 Erlangen von Versionsinformationen	CWE-200: Exposure of Sensitive Information to an Unauthorized Actor	-	-	Prüfen, ob vom Server ausgesendete Nachrichten Informationen über die eingesetzte Programmversion enthalten.

Tabelle A.2: Testplan für Fallbeispiel

Anhang B: Literatur

Literatur

[1] Daniel Markuson.
„Vigilante hacker breaks into routers to boost their security“.
In: *NordVPN Blog* (2018).
URL: https://nordvpn.com/de/blog/vigilante-hacker-mikrotik-routers/ (Abgerufen am 24. 08. 2021).

[2] Steve Morgan.
„Cybersecurity Talent Crunch To Create 3.5 Million Unfilled Jobs Globally By 2021“.
In: *Cybercrime Magazine* (2019).
URL: https://cybersecurityventures.com/jobs/ (Abgerufen am 20. 08. 2021).

[3] IBM Security.
Cost of a Data Breach Report.
Techn. Ber.
IBM Corporation, 2020.
URL: https://www.capita.com/sites/g/files/nginej291/files/2020-08/Ponemon-Global-Cost-of-Data-Breach-Study-2020.pdf.

[4] Claudia Eckert.
IT-Sicherheit : Konzepte - Verfahren - Protokolle.
Berlin Boston: De Gruyter, 2018,
S. 12, 19, 23, 25, 165, 169, 197.
ISBN: 9783110563900.

[5] Pete Herzog.
The Open Source Security Testing Methodology Manual.
ISECOM. 2010.
URL: https://www.isecom.org/OSSTMM.3.pdf.

[6] Elie Saad und Rick Mitchell.
Web Security Testing Guide, Version 4.2.
The OWASP Foundation. 2020,
S. 18.
URL: https://owasp.org/www-project-web-security-testing-guide/stable/.

[7] Bundesamt für Sicherheit in der Informationstechnik.
IT-Grundschutz-Kompendium.
Techn. Ber.
2021.
URL: https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Grundschutz/Kompendium/IT_Grundschutz_Kompendium_Edition2021.pdf.

[8] Microsoft Corporation.
„The STRIDE Threat Model“.
In: *Microsoft Technical Documentation* (2009).
URL: https://docs.microsoft.com/en-us/previous-versions/commerce-server/ee823878(v=cs.20) (Abgerufen am 14. 07. 2021).

[9] The MITRE Corporation.
CWE-20: Improper Input Validation.
2006.
URL: https://cwe.mitre.org/data/definitions/20.html (Abgerufen am 14. 07. 2021).

[10] The MITRE Corporation.
CWE-287: Improper Authentication.
2006.
URL: https://cwe.mitre.org/data/definitions/287.html (Abgerufen am 14. 07. 2021).

[11] National Institute of Standards und Technology.
CVE-2020-6010 Detail.
2020.
URL: https://nvd.nist.gov/vuln/detail/CVE-2020-6010 (Abgerufen am 14. 07. 2021).

[12] The MITRE Corporation.
About CWE. Frequently Asked Questions.
2020.
URL: https://cwe.mitre.org/about/faq.html (Abgerufen am 15. 07. 2021).

[13] xynmaps.
vsftpd 3.0.3 - Remote Denial of Service.
2021.
URL: https://www.exploit-db.com/exploits/49719 (Abgerufen am 16. 07. 2021).

[14] Victoria Drake.
Threat Modeling.
The OWASP Foundation, 2020.
URL: https://owasp.org/www-community/Threat_Modeling (Abgerufen am 19. 07. 2021).

[15] *Guide for conducting risk assessments*.
Techn. Ber.
2012.
DOI: 10.6028/nist.sp.800-30r1.
URL: https://doi.org/10.6028/nist.sp.800-30r1.

[16] Loren Kohnfelder und Praerit Garg.
„The threats to our products“.

In: (1999).
URL: https://adam.shostack.org/microsoft/The-Threats-To-Our-Products.docx (Abgerufen am 21. 07. 2021).

[17] Microsoft Corporation.
„Microsoft Threat Modeling Tool“.
In: (2017).
URL: https://docs.microsoft.com/en-us/azure/security/develop/threat-modeling-tool (Abgerufen am 21. 07. 2021).

[18] Ruby B. Lee.
„Security Basics for Computer Architects“.
In: *Synthesis Lectures on Computer Architecture* 8.4 (Sep. 2013), S. 1–111, 73.
DOI: 10.2200/s00512ed1v01y201305cac025.
URL: https://doi.org/10.2200/s00512ed1v01y201305cac025.

[19] Cisco Security Advisory.
Linux Kernel IP Fragment Reassembly Denial of Service Vulnerability Affecting Cisco Products.
Techn. Ber.
Cisco Systems, Inc., 2018.
URL: https://tools.cisco.com/security/center/content/CiscoSecurityAdvisory/cisco-sa-20180824-linux-ip-fragment (Abgerufen am 22. 07. 2021).

[20] CheatSheets Series Team.
Threat Modeling Cheat Sheet.
2021.
URL: https://cheatsheetseries.owasp.org/cheatsheets/Threat_Modeling_Cheat_Sheet.html (Abgerufen am 27. 08. 2021).

[21] Phil Zimmermann.
„Phil Zimmermann on PGP“.
In:
An Introduction to Cryptography.
PGP Corporation, 2005,
S. 54.

[22] The MITRE Corporation.
Solving Problems for a Better World.
2021.
URL: https://www.mitre.org/ (Abgerufen am 23. 07. 2021).

[23] The MITRE Corporation.
About CWE. CWE Glossary.
2018.
URL: https://cwe.mitre.org/documents/glossary/index.html (Abgerufen am 23. 07. 2021).

[24] The MITRE Corporation.

CWE-CATEGORY: Permission Issues.
2006.
URL: https://cwe.mitre.org/data/definitions/275.html (Abgerufen am 23. 07. 2021).

[25] The MITRE Corporation.
CWE-CATEGORY: Cryptographic Issues.
2006.
URL: https://cwe.mitre.org/data/definitions/310.html (Abgerufen am 23. 07. 2021).

[26] The MITRE Corporation.
CWE-74: Improper Neutralization of Special Elements in Output Used by a Downstream Component, Injection.
2006.
URL: https://cwe.mitre.org/data/definitions/74.html (Abgerufen am 02. 09. 2021).

[27] The MITRE Corporation.
Common Attack Pattern Enumeration and Classification.
2021.
URL: https://capec.mitre.org/ (Abgerufen am 24. 07. 2021).

[28] The MITRE Corporation.
Common Vulnerabilities and Exposures.
2021.
URL: https://cve.mitre.org/ (Abgerufen am 24. 07. 2021).

[29] The MITRE Corporation.
Request CVE IDs.
2021.
URL: https://cve.mitre.org/cve/request_id.html (Abgerufen am 23. 08. 2021).

[30] The MITRE Corporation.
CVE → CWE Mapping Guidance.
2021.
URL: https://cwe.mitre.org/documents/cwe_usage/guidance.html (Abgerufen am 26. 07. 2021).

[31] The MITRE Corporation.
Search Results.
2021.
URL: https://cve.mitre.org/cgi-bin/cvekey.cgi?keyword=opensmtpd+%5C%27out-of-bounds%5C%27 (Abgerufen am 28. 07. 2021).

[32] National Institue of Standards und Technology.
National Vulnerability Database.
2021.

URL: https://nvd.nist.gov/ (Abgerufen am 26. 07. 2021).

[33] The MITRE Corporation.
CWE-319: Cleartext Transmission of Sensitive Information.
2006.
URL: https://cwe.mitre.org/data/definitions/319.html (Abgerufen am 02. 09. 2021).

[34] International Organization for Standardization.
ISO/IEC 27001 Information Security Management.
2021.
URL: https://www.iso.org/isoiec-27001-information-security.html (Abgerufen am 27. 07. 2021).

[35] James Tarala und Kelli K. Tarala.
Open Threat Taxonomy, Version 1.1.
Enclave Security. Venice, United States, 2015.
URL: https://www.auditscripts.com/free-resources/open-threat-taxonomy/.

[36] Inc. FIRST.Org.
Common Vulnerability Scoring System version 3.1: Specification Document, Revision 1.
Techn. Ber.
2019.
URL: https://www.first.org/cvss/v3.1/specification-document (Abgerufen am 09. 09. 2021).

[37] National Institue of Standards und Technology.
Official Common Platform Enumeration (CPE) Dictionary.
2021.
URL: https://nvd.nist.gov/products/cpe (Abgerufen am 29. 07. 2021).

[38] Inc Cisco Systems.
Cisco Security Advisory.
2021.
URL: https://tools.cisco.com/security/center/publicationListing.x (Abgerufen am 29. 07. 2021).

[39] The PHP Group.
php.
2021.
URL: https://www.php.net/ (Abgerufen am 24. 08. 2021).

[40] The Apache Software Foundation.
Apache HTTP Server Project.
2021.
URL: https://httpd.apache.org/ (Abgerufen am 24. 08. 2021).

[41] MariaDB Foundation.
MariaDB Server: The open source relational database.
2021.
URL: https://mariadb.org/ (Abgerufen am 24. 08. 2021).

[42] Sean-Philip Oriyano und Robert Shimonski.
„Web Application Attacks“.
In: *Client-Side Attacks and Defense*.
Elsevier, 2012,
S. 195–221.
DOI: 10.1016/b978-1-59-749590-5.00008-0.
URL: https://doi.org/10.1016/b978-1-59-749590-5.00008-0.

[43] The MITRE Corporation.
CWE-410: Insufficient Resource Pool.
2006.
URL: https://cwe.mitre.org/data/definitions/410.html (Abgerufen am 02. 09. 2021).

[44] The MITRE Corporation.
CWE-400: Uncontrolled Resource Consumption.
2006.
URL: https://cwe.mitre.org/data/definitions/400.html (Abgerufen am 02. 09. 2021).

[45] The MITRE Corporation.
CWE-664: Improper Control of a Resource Through its Lifetime.
2008.
URL: https://cwe.mitre.org/data/definitions/664.html (Abgerufen am 02. 09. 2021).

[46] The MITRE Corporation.
CWE-770: Allocation of Resources Without Limits or Throttling.
2009.
URL: https://cwe.mitre.org/data/definitions/770.html (Abgerufen am 02. 09. 2021).

[47] PortSwigger Ltd.
Burp Suite.
2021.
URL: https://portswigger.net/burp (Abgerufen am 08. 09. 2021).

[48] Justin Hutchings.
„Introducing new ways to keep your code secure“.
In: *The GitHub Blog* (2019).
URL: https://github.blog/2019-05-23-introducing-new-ways-to-keep-your-code-secure/ (Abgerufen am 08. 09. 2021).

[49] Marius Iulian Mihailescu und Stefania Loredana Nita.

Pro Cryptography and Cryptanalysis with C++20.
Apress, 2021.
DOI: 10.1007/978-1-4842-6586-4.
URL: https://doi.org/10.1007/978-1-4842-6586-4.

[50] Ed Dante.
„Cybersecurity Risk Assessment – A Better Way“.
In: *Fractional CISO* (2018).
URL: https://fractionalciso.com/cybersecurity-risk-assessment/
(Abgerufen am 08. 09. 2021).

[51] Anne Honkaranta, Tiina Leppanen und Andrei Costin.
„Towards Practical Cybersecurity Mapping of STRIDE and CWE — a Multi-perspective Approach“.
In: *2021 29th Conference of Open Innovations Association (FRUCT)*.
IEEE, Mai 2021.
DOI: 10.23919/fruct52173.2021.9435453.
URL: https://doi.org/10.23919/fruct52173.2021.9435453.

Eidesstattliche Versicherung

Hiermit versichere ich an Eides statt, dass ich meine Arbeit selbstständig verfasst, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt und die Arbeit noch nicht anderweitig für Prüfungszwecke vorgelegt habe.

Stellen, die wörtlich oder sinngemäß aus Quellen entnommen wurden, sind als solche kenntlich gemacht.

Mittweida, 16. September 2021