Modellierung von Angriffen für quantitative Sicherheitsanalysen

Masterarbeit von

Matthias Lüthy

An der KIT-Fakultät für Informatik
KASTEL – Institut für Informationssicherheit und Verlässlichkeit

1. Prüfer/Prüferin:	Prof. Dr. Ralf Reussner
2. Prüfer/Prüferin:	Prof. Dr. Bernhard Beckert
1. Betreuer/Betreuerin:	Frederik Reiche, M.Sc.
2. Betreuer/Betreuerin:	Florian Lanzinger, M.Sc.
3. Betreuer/Betreuerin:	Nils Niehues, M.Sc.

01. August 2024 – 31. Januar 2025

Karlsruher Institut für Technologie
Fakultät für Informatik
Postfach 6980
76128 Karlsruhe

Modellierung von Angriffen für quantitative Sicherheitsanalysen (Masterarbeit)

Ich versichere wahrheitsgemäß, die Arbeit selbstständig verfasst, alle benutzten Quellen und Hilfsmittel vollständig und genau angegeben und alles kenntlich gemacht zu haben, was aus Arbeiten anderer unverändert oder mit Abänderungen entnommen wurde sowie die Satzung des KIT zur Sicherung guter wissenschaftlicher Praxis in der jeweils gültigen Fassung beachtet zu haben.

Karlsruhe, 31. Januar 2025

. . . . . .

(Matthias Lüthy)

Zusammenfassung

Die vorliegende Masterarbeit beschäftigt sich mit der Modellierung von Angreifern und Angriffen auf ein komponentenbasiertes Softwaresystem, das zuvor bereits mittels eines bestehenden Ansatzes zur Quantifizierung von Architektur und Code modelliert und formal verifiziert wurde. Während die vorausgehende formale Verifikation einen wohlwollenden, ehrlichen Nutzer voraussetzt, erweitert diese Masterarbeit das Modell um einen Angreifer, der gezielt Randfälle und Implementierungsfehler ausnutzt. Der Unterschied zu bisherigen Arbeiten, die sich mit Angriffen auf Softwaresysteme beschäftigen, besteht darin, dass Modellierung und Analyse nicht auf der Architekturebene, sondern auf der Quellcode-Ebene erfolgen. Konkret wird ein Metamodell für Schwachstellen, Angriffsbäume und Angreifer vorgestellt und ein Analysetool implementiert, um eine quantitative Aussage über die Angriffssicherheit des Softwaresystems treffen zu können. Zur Evaluation wird eine Fallstudie mit zehn verschiedenen Angriffsbäumen durchgeführt, die alle verschiedenen, denkbaren Baumkonstruktionen repräsentieren. Zur Skalierbarkeitsuntersuchung wird für einen Angriffsbaum die Anzahl an parallelen Knoten sowie die Anzahl von Angreifern variiert. In der Auswertung ergeben sich eine Präzision und Sensitivität von 100 % sowie eine rein auf den neuen Beitrag dieser Arbeit bezogene sehr gute Skalierbarkeit der Analyse.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1. Einleitung

Kaum ein Tag vergeht, ohne dass auf gängigen Portalen zu IT-Nachrichten wie beispielsweise *heise online* über gefundene oder geschlossene, teils hochriskante Sicherheitslücken in Software und Hardware berichtet wird (ein paar Beispiele der zum Zeitpunkt des Schreibens letzten sieben Tage: [37], [20], [17], [38], [18]). Auch die Politik hat das Problem der oft mangelnden Sicherheit von Softwaresystemen längst erkannt. Erst Mitte Januar verkündete der amtierende Bundesgesundheitsminister Lauterbach in Bezug auf die neu eingeführte elektronische Patientenakte: „Sicherheit hat oberste Priorität“ [19]. Der Sicherheitsbegriff ist in dieser plakativen Überschrift recht weit gefasst und umfasst viele verschiedene Sicherheitsaspekte. Beispiele hierfür sind die Datensicherheit, die Zuverlässigkeit, die Verfügbarkeit oder die Angriffssicherheit. Viele dieser Aspekte sind auch für einen Softwareentwickler sehr relevant und müssen zwingend berücksichtigt werden. Viega und McGraw sprechen in diesem Zusammenhang von Sicherheit als Qualitätsanforderung, die möglichst früh im Software-Entwicklungsprozess berücksichtigt werden sollte [42, S. 13ff.]. Im Folgenden werden zwei konkrete Aspekte genauer betrachtet: die funktionale Korrektheit und die Angriffssicherheit. Zunächst stellt sich für diese beiden Aspekte die Frage, wie sie im Software-Entwicklungsprozess adäquat berücksichtigt werden können.

Eine naive Lösung hierfür ist die Durchführung von verschiedenen Tests im Laufe des Entwicklungsprozesses, also beispielsweise Unittests, Integrationstests oder Systemtests. Die Testziele dieser verschiedenen Tests sind je nach Softwaresystem individuell festgelegt. In der Regel dienen sie aber dazu, die Erfüllung der Anforderungen an das Softwaresystem beziehungsweise an die zu testende Komponente nachzuweisen [41, S. 14f.]. Durch die verschiedenen Teststufen werden vermutlich sicherheitsrelevante Fehler gefunden und behoben. Es gilt jedoch auch hier die Aussage von Dijkstra: „Programmtests können dazu dienen, das Vorhandensein von Fehlern nachzuweisen, aber niemals, um deren Abwesenheit zu beweisen!“ [9, S. 6] Eine quantitative Aussage, ob das Softwaresystem korrekt arbeitet und eine Sicherheit bezüglich Angriffen gegeben ist, ist durch die reine Durchführung von Tests mit anschließender Fehlerbehebung nicht möglich.

1.1. Ziel und Methodik

Als Ziel dieser Arbeit stellt sich folgende Forschungsfrage: Ist es möglich, ein komponentenbasiertes Softwaresystem hinsichtlich seiner Angriffssicherheit auf Grundlage einer formalen Analyse des Quellcodes quantitativ zu bewerten?

In einem bereits existierenden Ansatz namens „Quantifying Architecture and Code“ (QuAC) [25] wird zunächst die funktionale Korrektheit einzelner Komponenten eines Softwaresystems formal bewiesen. Im Anschluss daran wird mittels verschiedener weiterer Modellierungen ein quantitatives Maß für die Korrektheit der Ausführung in Bezug auf das Modell eines ehrlichen, wohlwollenden Nutzers bestimmt. Allerdings kann aktuell keine Aussage über die Angriffssicherheit des Softwaresystems getroffen werden. Hierfür müssten Nutzer berücksichtigt werden, die nicht wohlwollend sind, sondern gezielt versuchen, fehlerhafte Eingaben zu machen sowie Randfälle und Implementierungsfehler auszunutzen. Diese Lücke der fehlenden Berücksichtigung von Angreifern wird in der hier vorliegenden Arbeit gefüllt: Basierend auf dem formalen Korrektheitsbeweis aus QuAC werden gezielt Fälle, deren Korrektheit nicht bewiesen wurde, ausgenutzt. Dafür werden zunächst Angreifer und Angriffe, die auf genau diese Fälle abzielen, mittels Angriffsbäumen modelliert. Der Angreifer versucht also, einen inkorrekten Zustand zu provozieren. Diese Angriffsbäume enthalten verschiedene Möglichkeiten, einen Angriff durchzuführen, wobei die Knoten mit verschiedenen hierfür relevanten Parametern annotiert sind. Abgedeckt hiervon sind derzeit ausschließlich Angriffe, die sich aus der Analyse des Quellcodes und aus daraus resultierenden Fehlerzuständen ergeben. Noch nicht abgedeckt, aber für eine Erweiterung vorgesehen, sind Angriffe auf allgemein bekannte Sicherheitslücken, wie sie beispielsweise in den „Common Vulnerabilities and Exposures“ (CVE) [13] beschrieben sind.

Nach erfolgter Modellierung wird das Ergebnis analysiert und ein Maß für die Angriffssicherheit des Systems berechnet. Hierfür werden die modellierten Angreifer mit ihren Angriffsbäumen und Schwachstellen analysiert und zwei verschiedene Angreiferverhalten berücksichtigt: ein possibilistischer Angreifer und ein probabilistischer Angreifer. Der possibilistische Angreifer zeichnet sich dadurch aus, dass er die für ihn beste Handlungsoption, also den für ihn am erfolgversprechendsten Angriff, wählt. Der probabilistische Angreifer hingegen wählt einen zufälligen Angriff. Um hier vergleichbare Werte zu erhalten, wird im Folgenden für diesen Angreifertyp keine zufällige Angriffsoption ausgewählt, sondern es wird das arithmetische Mittel aus allen Angriffsoptionen gebildet und für die Berechnung verwendet. Durch die Betrachtung dieser beiden Angreifertypen ist eine differenzierte Betrachtung von Angriffen möglich: Die probabilistische Angriffsauswertung bietet aufgrund der Berechnung des durchschnittlich zu erwartenden Ergebnisses eine gute Orientierung für die Gesamtsicherheit bezüglich eines einzelnen Angriffs, wohingegen die possibilistische Auswertung die Einschätzung des maximalen Angriffsrisikos erleichtert, da dieses maximale Risiko mit der besten Handlungsoption des Angreifers gleichzusetzen ist. Durch die zusätzliche Ausgabe des entsprechenden Angriffspfads für den possibilistischen Angreifer können gezielt die Angriffe mit dem höchsten Risiko, der höchsten Wahrscheinlichkeit oder auch den größten Auswirkungen betrachtet und Schutzmaßnahmen dafür getroffen werden.

Der konkrete Beitrag dieser Arbeit liegt also in der Möglichkeit, Angreifer und Angriffsbäume zu modellieren sowie auf dieser Grundlage eine Auswertung mit einer Sicherheitsanalyse durchzuführen. Somit kann eine quantitative Aussage über die Angriffssicherheit eines komponentenbasierten Softwaresystems getroffen werden. Mittels dieser quantitativen Aussage wäre es beispielsweise auch für unseren Bundesgesundheitsminister im anfangs

erwähnten Beispiel möglich, eine konkrete Einschätzung über die Angriffssicherheit der elektronischen Patientenakte zu treffen.

1.2. Aufbau der Arbeit

In Kapitel 2 werden zunächst alle für das weitere Verständnis erforderlichen Grundlagen, wie beispielsweise Angriffsbäume und QuAC, erklärt. Außerdem wird hier auf verschiedene Kategorisierungen von Schwachstellen und deren Bewertungskriterien eingegangen. Im folgenden Kapitel 3 wird ein laufendes Beispiel eingeführt, das im übrigen Verlauf der Arbeit mehrfach aufgegriffen wird. Daran anschließend folgt mit Kapitel 4 die Vorstellung der Angreifermodellierung. Zunächst wird die Architektur des neu erstellten Metamodells vorgestellt, bevor dann alle erklärten Modellierungskonzepte zu einem vollständigen Angriff zusammengesetzt werden. Kapitel 5 beschäftigt sich mit der Analyse der Modellierung und der Berechnung des Angriffssicherheitsmaßes. Die Evaluation der Arbeit erfolgt in Kapitel 6, bevor Kapitel 7 einen Überblick über die verwandten Arbeiten gibt. Abschließend werden in Kapitel 8 die Forschungsergebnisse zusammengefasst und die oben gestellte Forschungsfrage beantwortet. Auch auf Beschränkungen sowie mögliche zukünftige Arbeiten wird an dieser Stelle eingegangen.

2. Grundlagen

Dieses Kapitel führt die grundlegenden Begriffe ein, die für das Verständnis im weiteren Verlauf der Arbeit relevant sind. Zunächst wird das Konzept der Angriffsbäume erklärt. Daran anschließend folgt das Palladio Komponentenmodell sowie eine Erweiterung dessen durch QuAC. QuAC stellt gleichzeitig die Grundlage beziehungsweise den Ausgangspunkt für diese Arbeit dar. Abschließend folgt ein Unterkapitel zu Schwachstellen, in welchem auch konkrete Bewertungsmethoden beziehungsweise Sammlungen von Schwachstellen vorgestellt werden.

2.1. Angriffsbäume und Angriffsgraphen

Sehr viele verschiedene Arbeiten beschäftigen sich auf verschiedenen Ebenen mit Angriffen auf Softwaresysteme. Einen Überblick darüber bieten beispielsweise Konsta u. a. [21] und Kordy u. a. [22] in ihren Arbeiten. Grundsätzlich stellt sich bei der Modellierung von Angriffen die Frage nach der geeigneten Repräsentation. Wie von Konsta u. a. [21] vorgestellt, gibt es hierfür im Wesentlichen zwei verbreitete Möglichkeiten: Angriffsbäume und Angriffsgraphen.

Angriffsbäume Ein Angriffsbaum dient dazu, verschiedene Möglichkeiten aufzuzeigen, wie ein definiertes Ziel erreicht werden kann. Verschiedene Möglichkeiten können zusammenhängen und es kann auch erforderlich sein, einzelne Aktionen in einer bestimmten Reihenfolge auszuführen. Der Aufbau entspricht dabei einer Baumstruktur. Ursprünglich wurde dieses Konzept des Angriffsbaums von Schneier [39] definiert. Als anschauliches Beispiel für die weiteren Erklärungen dient ein Beispiel von Mantel und Probst [27], das sich auf die ursprüngliche Definition von Schneier bezieht. Der Angriffsbaum für dieses Beispiel ist in Abbildung 2.1 zu sehen.

Das Ziel des hier dargestellten Angriffs ist es, Zugang zu Geld zu erlangen. Dieses Ziel ist im sogenannten Wurzelknoten definiert. Von diesem Wurzelknoten gehen mehrere Kanten aus. Jeder Knoten am Ende einer solchen Kante entspricht einer Möglichkeit, das Gesamtziel „Zugang zu Geld erlangen“ zu erreichen. Jede dieser vier Möglichkeiten ist unabhängig von den jeweils anderen Optionen erreichbar.

Beim ganz linken und ganz rechten Knoten, also „Get ATM“ und „Replace Control Software“, handelt es sich um zwei der Blätter des Baums. Sie verfügen also über keine ausgehende Kante, sondern bilden das Ende des jeweiligen Pfads im Baum. Bei allen Blättern handelt es

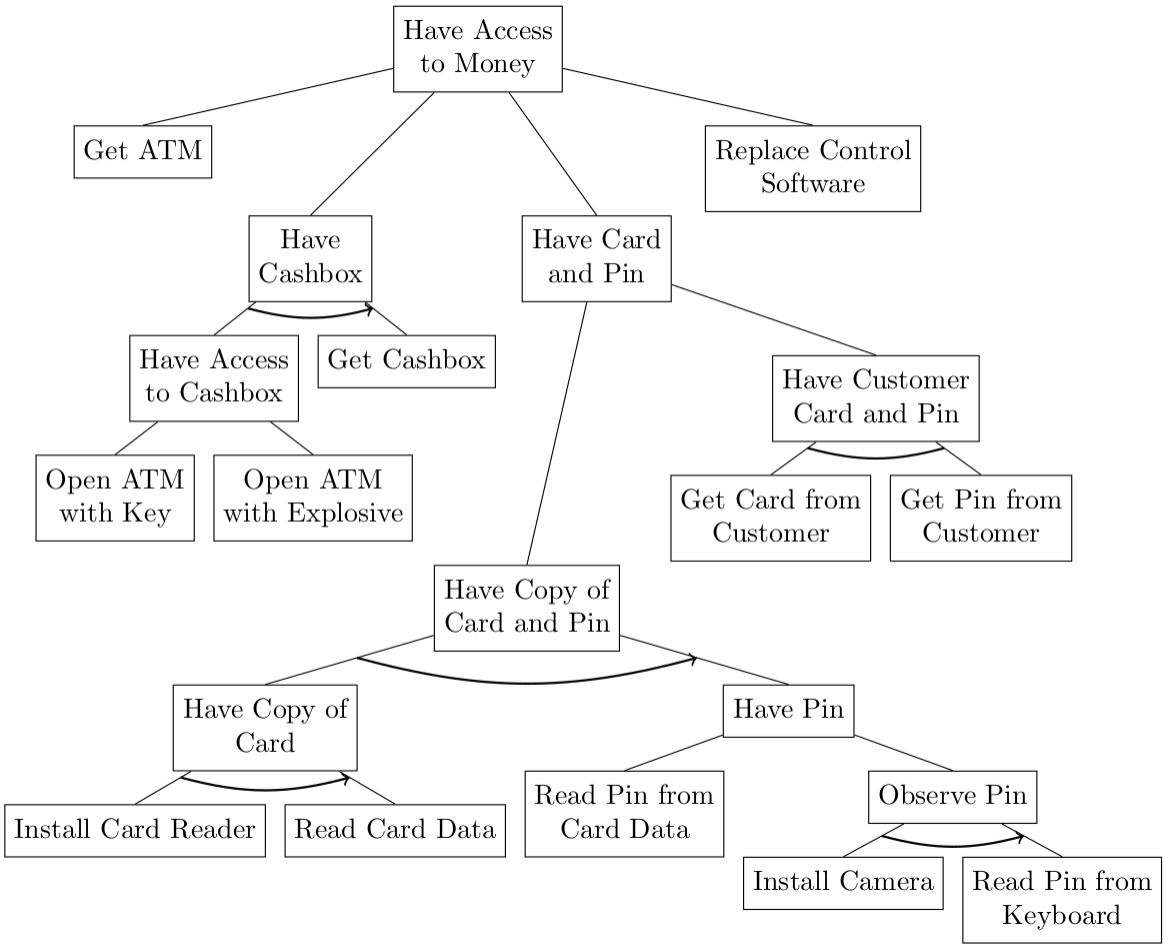

Abbildung 2.1.: Ein beispielhafter Angriffsbaum für das Szenario „Have Access to Money“, der alle möglichen Elemente eines Angriffsbaums enthält [27].

sich um eine konkrete Aktion, die zum Erreichen des Ziels ausgeführt werden kann beziehungsweise muss. Im Fall des ganz linken Blattes ist die Aktion also beispielsweise, einen Geldautomaten zu erhalten. Eine andere Aktion aus dem Baum wäre aber beispielsweise auch „Install Card Reader“. Die Aktionen dienen grundsätzlich immer dazu, das im direkt übergeordneten Knoten definierte Ziel zu erreichen [27].

Beim eben angesprochenen Beispiel „Install Card Reader“, welches ganz unten links im Baum zu sehen ist, gibt es eine Besonderheit: das Ziel des direkt übergeordneten Knotens „Have Copy of Card“, also eine Kopie einer Karte zu haben, benötigt zwingend zwei Aktionen, die nacheinander ausgeführt werden müssen. Dies ist an der Linie mit eingezeichneter Pfeilspitze, die die beiden Kanten zu den Blättern „Install Card Reader“ und „Read Card Data“ verbindet, zu erkennen. Durch die Verbindungslinie zwischen den beiden Kanten wird hier definiert, dass beide Aktionen zum Erreichen des Ziels ausgeführt werden müssen. Die Pfeilspitze sagt aus, dass zusätzlich auch die Ausführungsreihenfolge der beiden Aktionen wichtig ist. In diesem Fall muss also zuerst der Kartenleser installiert werden, bevor die

Kartendaten ausgelesen werden können. Im weiteren Verlauf wird dies als „sequentielle Konjunktion“ (SAND) bezeichnet. Ohne die Pfeilspitze, wie es beispielsweise bei „Have Customer Card and Pin“ der Fall ist, ist die Ausführungsreihenfolge irrelevant. Es handelt sich dann um eine „parallele Konjunktion“ (AND) [27]. Im weiteren Verlauf wird nur die parallele Konjunktion verwendet, da sich diese für den Umfang dieser Arbeit nicht von der sequentiellen unterscheidet.

Alle Aktionen oder auch Knoten, die über keine solche Verbindungslinie verfügen, sind disjunktiv verbunden und werden im weiteren Verlauf auch „OR-Knoten“ genannt. Sie sind unabhängig voneinander ausführbar. Zur Erreichung des übergeordneten Ziels ist die erfolgreiche Ausführung eines der Knoten ausreichend. In der hier vorgestellten Formalisierung von Angriffsbäumen können diese drei Typen von Knoten nicht gemischt werden. Für jeden Knoten wird beim Entwurf des Angriffsbaums einer der drei Typen festgelegt. Der Typ des nachfolgenden Kindknotens kann sich wiederum von dem des Elternknotens unterscheiden [27].

Angriffsgraphen Bei Angriffsgraphen gilt nicht mehr das Prinzip eines Baums, dass jeder Kindknoten genau einen direkten Vorgängerknoten besitzt, sondern es sind mehrere Vorgängerknoten für einen Kindknoten oder beispielsweise auch Verbindungen zwischen verschiedenen Knoten auf einer Ebene möglich. Verschiedene Ausprägungen von Angriffsgraphen schränken die möglichen Verbindungen weiter ein. Da im weiteren Verlauf der Arbeit Angriffsbäume verwendet werden, wird hier nicht weiter auf die Definition von verschiedenen Angriffsgraphen eingegangen [21].

2.2. Palladio Komponentenmodell

Mithilfe des von Reussner u. a. [35] entwickelten Palladio Komponentenmodells (PCM) ist es möglich, eine Softwarearchitektur zu modellieren und die Auswirkungen von verschiedenen Entwurfsentscheidungen zu simulieren. Die Architektur besteht dabei aus verschiedenen, unabhängigen Komponenten, die jeweils Dienste bereitstellen und Dienste benötigen können. Stellt eine Komponente einen Dienst bereit, den eine andere Komponente benötigt, sind diese beiden Komponenten miteinander verbunden ([35], [25]).

Für jede Komponente wird eine Verhaltensspezifikation definiert, die sogenannte „Service Effect Specification“ (SEFF). In dieser ist das Verhalten der jeweiligen Komponente zwischen den Schnittstellen für die benötigten und die bereitgestellten Dienste dargestellt. Der Abstraktionsgrad ist individuell und hängt vom konkreten Anwendungsfall ab. Beispielsweise ist es möglich, verschiedene Verzweigungen oder Parameter-Einflüsse zu spezifizieren. Optimierungen und Berechnungen, die die grundlegende Funktionsweise des Dienstes nicht maßgeblich beeinflussen, sollten in der SEFF wegabstrahiert werden, da das Abstraktionslevel der SEFF allgemein sehr hoch sein sollte ([35], [25]).

Außerdem bietet das Palladio Komponentenmodell die Möglichkeit, das Verhalten des Nutzers als „Usage Profile“ zu beschreiben. Das zu diesem Nutzungsprofil gehörende Nutzungsmodell besteht aus einem Flussdiagramm sowie probabilistischen Werten zur Modellierung der Nutzereingaben. Es beinhaltet beispielsweise Wahrscheinlichkeitswerte für die Verzweigungen innerhalb des Flussdiagramms [35].

2.3. QuAC

QuAC stellt die Grundlage für diese Arbeit dar. Die Abkürzung steht, wie bereits in Abschnitt 1.1 eingeführt, für „Quantifying Architecture and Code“ und bezeichnet einen modular aufgebauten Ansatz, mit dem die Korrektheit von Service-orientierten Softwaresystemen quantifiziert werden kann. Dieser Ansatz wurde von Lanzinger u. a. [25] entworfen und wird im folgenden Abschnitt auf Grundlage des genannten Papiers vorgestellt.

Der Ansatz baut darauf auf, dass die Aussagen von formalen Methoden, die oft auf binäre Entscheidungen beschränkt sind, der Komplexität von Softwaresystemen nicht gerecht werden. Einige andere, nicht-binäre Messungen der Software-Zuverlässigkeit beziehen sich oft auf die Testabdeckung. Jedoch kann hiermit keine Korrektheit für konkrete Anwendungsszenarien garantiert werden. Konkret beantworten Lanzinger u. a. die Frage, mit welcher Wahrscheinlichkeit eine typische Nutzung des Systems nicht zu einem Fehler führt. Ein Fehler wird dabei als ein Programmzustand definiert, der nicht dem Vertrag einer Methode entspricht. Hierfür werden „Coverage Regions“ definiert. Es handelt sich dabei um eine Formel, die die Eingabeparameter und Komponentenzustände beschreibt, für die sich die aktuelle Methode bewiesenermaßen korrekt verhält, das heißt, für die sie ihre Methodenverträge einhält. Ein Methodenvertrag bezeichnet hierbei das von Meyer [28] beschriebene Konzept, wonach Methoden über eine oder mehrere Vor- und Nachbedingungen, die in einer „behavioral specification language“ wie beispielsweise JML [26] geschrieben sind, verfügen sollten. Vorbedingungen beschreiben dabei Voraussetzungen, die vor dem Ausführen der Methode erfüllt sein müssen, während Nachbedingungen eine Aussage darüber treffen, welche Eigenschaften nach erfolgreicher Ausführung der Methode gegeben sein müssen [28]. Für die Nutzung des Ansatzes wird auf Quellcode-Ebene vorausgesetzt, dass es sich um ein objektorientiertes Programm handelt, das aus Klassen besteht, die wiederum Methoden bereitstellen. Auf architektonischer Ebene muss es sich um eine Service-orientierte, komponentenbasierte Architektur handeln. Wird der Zusammenhang zwischen dem abstrahierten Komponentenmodell und der konkreten Quellcode-Ebene hergestellt, lässt sich feststellen, dass jeder Service einer Methode und jede Komponente einer Klasse entspricht. Jeder Service und somit auch jede Methode hat dabei einen Vertrag, der, wie oben beschrieben, eine oder mehrere Vor- und Nachbedingungen besitzt [25].

Die gesamte Modellierung von QuAC wird mittels des in Abschnitt 2.2 vorgestellten Palladio Komponentenmodells durchgeführt. Hierfür wird die SEFF des PCM um Zustandsvariablen und Coverage Regions erweitert: Der für QuAC relevante Zustand einer Komponente wird mittels Zustandsvariablen beschrieben. Diese enthalten einen Namen, einen Typ sowie eine Wahrscheinlichkeitsverteilung, die die möglichen Initialwerte der Variable beschreibt.

Außerdem werden in dieser SEFF-Erweiterung Coverage Regions in Form von „Direct Error Conditions“ definiert. Hier können verschiedene Variablenbelegungen festgelegt werden, die zu einem vom Methodenvertrag abweichenden Zustand führen würden. Diese Variablenbelegungen werden in frühen Stadien des Softwareprojekts durch die Entwickler bestimmt. In späteren Projektstadien können diese dann durch Test- und Verifikationsergebnisse ersetzt werden [25].

Die in diesem erweiterten Modell des PCMs erfolgte Modellierung wird im QuAC-Ansatz anschließend in ein probabilistisches Programm übersetzt. Mittels eines Model Checkers beziehungsweise Model Counters werden abschließend die Wahrscheinlichkeiten bestimmt. Hierbei wird zwischen verschiedenen Wahrscheinlichkeiten unterschieden: direkte Fehler, kein Fehler und „Conditional Calls“. Unter einem direkten Fehler wird die Wahrscheinlichkeit verstanden, dass eine Methode selbst diesen Fehler auslöst. Conditional Calls können hingegen im Ergebnis verschachtelt sein und verschiedene Unterelemente enthalten: erneute Conditional Calls sowie direkte Fehler und kein Fehler. Jedes dieser Unterelemente ist dabei mit einer eigenen Wahrscheinlichkeit versehen. Diese Conditional Calls decken genau die Fälle ab, in denen ein Methodenaufruf aus einer anderen Methode heraus zu einem Fehler führt. Bei der Wahrscheinlichkeit, dass ein Conditional Call eintritt, handelt es sich also um die Wahrscheinlichkeit, dass der durch diesen Conditional Call definierte Zustand eingenommen wird [25].

Insgesamt ist somit mittels QuAC eine quantitative Aussage über die Zuverlässigkeit von größeren Softwaresystemen möglich. Durch die Nutzung und Erweiterung der SEFFs und die Nutzung des Usage Models können hierbei Implementierungsdetails explizit berücksichtigt werden [25].

2.4. Schwachstellen

Zunächst stellt sich die Frage nach dem Unterschied zwischen einer Schwachstelle („Weakness“) und einer Verwundbarkeit („Vulnerability“). Bei Schwachstellen handelt es sich um Probleme oder Fehler in einer Hardware oder Software, wie zum Beispiel Buffer Overflows oder Fehler bei der Zugriffskontrolle auf geteilte Ressourcen. Verwundbarkeiten bezeichnen Schwachstellen, die durch einen böswilligen Angreifer konkret ausnutzbar sind oder sogar schon ausgenutzt werden. Schwachstellen sind somit die Vorstufe zu einer Verwundbarkeit [7].

Für dieses Thema existieren neben der etwas komplexen Definition beziehungsweise Abgrenzung der beiden Begriffe auch verschiedene Kataloge, die bereits bekannte Schwachstellen und Verwundbarkeiten enthalten. Zwei davon werden in Unterabschnitt 2.4.1 vorgestellt. Anschließend folgen in Unterabschnitt 2.4.2 zwei verschiedene Methoden, um Schwachstellen zu bewerten.

2.4.1. Schwachstellen- und Verwundbarkeitskataloge

CWE steht für „Common Weakness Enumeration“ [7]. In der Liste der CWEs sind verschiedenste, häufig anzutreffende Software- und Hardware-Schwachstellen zu finden. Dabei handelt es sich um Schwachstellen und nicht um konkrete Verwundbarkeiten. Die Sammlung zielt darauf ab, ein Bewusstsein für häufig auftauchende Schwachstellen zu schaffen und die weitere Implementierung solcher Schwachstellen zu verhindern. Hierdurch soll implizit auch die Ausnutzung der aus den Schwachstellen entstehenden Verwundbarkeiten reduziert werden [7].

In den „Common Vulnerabilities and Exposures“ (CVE) [13] hingegen werden konkrete Verwundbarkeiten, also ausgenutzte Sicherheitslücken, gelistet. Das Ziel dieser Liste ist es, gefundene Verwundbarkeiten zu klassifizieren, zentral zu speichern und zu veröffentlichen. Über die vergebene, eindeutige CVE-Nummer ist ein eindeutiger Bezug zur jeweiligen Verwundbarkeit möglich [30].

2.4.2. Schwachstellen-Bewertung

Im Folgenden werden zwei verschiedene, konkrete Systeme für die Bewertung von Schwachstellen vorgestellt.

2.4.2.1. CVSS

Das „Common Vulnerability Scoring System“ (CVSS) [6] dient dazu, Software-Verwundbarkeiten anhand ihrer Charakteristiken und der Schwere der Verwundbarkeiten zu klassifizieren. Es ordnet jeder klassifizierten Verwundbarkeit einen Wert zwischen null und zehn zu. Je nach Version des recht komplexen Berechnungsalgorithmus unterscheidet sich das konkrete Ergebnis.

2.4.2.2. OWASP

Das „Open Worldwide Application Security Project“ (OWASP) [1] stellt eine alternative Risiko-Bewertungsmethode zur Verfügung. Die Berechnung des Risikos, das von einer Schwachstelle ausgeht, wird im Gegensatz zu einer Vielzahl von Faktoren, die das CVSS berücksichtigt, hier lediglich von zwei Faktoren beeinflusst: Impact und Likelihood. Für beide Werte wird für die OWASP-Methode eine Berechnungshilfe zur Verfügung gestellt. Beide zu bestimmenden Werte liegen im Wertebereich von null bis neun. Sie werden in eine der drei Risikostufen gemäß Tabelle 2.1 eingeordnet. Anschließend werden beide Ergebnis-Risikostufen mittels Tabelle 2.2 miteinander kombiniert, um einen Wert für die Gesamtschwere des von der Schwachstelle ausgehenden Risikos zu erhalten [31]. Diese Bewertungsmethode stellt im Vergleich zum CVSS eine vergleichsweise einfache Möglichkeit der Klassifizierung des Risikos dar und wird deshalb im Verlauf der Arbeit weiter verwendet.

Wertebereich	Risikostufe
0 bis < 3	**LOW**
3 bis < 6	**MEDIUM**
6 bis 9	**HIGH**

Tabelle 2.1.: Mittels dieser Skala werden die berechneten Werte für Impact und Likelihood einer Risikostufe zugeordnet [31].

Gesamtschwere des Risikos				
Impact	HIGH	Medium	High	Critical
	MEDIUM	Low	Medium	High
	LOW	Note	Low	Medium
		LOW	MEDIUM	HIGH
	Likelihood			

Tabelle 2.2.: Bewertungstabelle gemäß OWASP, um aus den beiden berechneten Risikostufen für Impact und Likelihood die Bewertung der Gesamtschwere des Risikos zu bestimmen [31].

Likelihood Likelihood bezeichnet die Wahrscheinlichkeit, dass eine Schwachstelle von einem Angreifer gefunden und ausgenutzt wird. Der Wertebereich der Likelihood liegt zwischen null (niedrig) und neun (hoch). OWASP sieht für die Berechnung des Ergebniswertes die Bildung des arithmetischen Mittels aus den einzelnen Teilergebnissen der folgenden, in Angreiferfaktoren und Schwachstellen-Faktoren unterteilten Aufzählung vor [31]. Diese Aufzählung wurde vollständig aus [31] übernommen und ins Deutsche übersetzt. Anschließend erfolgt die Klassifizierung in eine Risikostufe gemäß Tabelle 2.1.

Angreiferfaktoren Die Gruppe der Angreifer kann an dieser Stelle auch aus nur einem Angreifer bestehen. Die genaue Gruppengröße spielt für das Ergebnis keine Rolle.

- Fähigkeiten: Wie technisch geschickt ist die Gruppe der Angreifer?
  - (1) keine technischen Fähigkeiten
  - (3) einige technische Fähigkeiten
  - (5) fortgeschrittener Computer-Nutzer
  - (6) Netzwerk- und Programmierkenntnisse
  - (9) Kenntnisse zur Security-Durchdringung

- Motiv: Wie motiviert ist die Gruppe der Angreifer, die Schwachstelle zu finden und auszunutzen?
  - (1) geringe oder keine Belohnung
  - (4) mögliche Belohnung
  - (9) hohe Belohnung
- Möglichkeiten: Welche Ressourcen und Möglichkeiten sind für die Gruppe von Angreifern erforderlich, um die Schwachstelle zu finden und auszunutzen?
  - (0) Vollzugriff oder sehr teure Ressourcen benötigt
  - (4) spezieller Zugriff oder spezielle Ressourcen erforderlich
  - (7) einige Ressourcen oder bestimmter Zugriff erforderlich
  - (9) kein Zugriff und keine Ressourcen erforderlich
- Größe: Wie groß ist die Gruppe der Angreifer?
  - (2) Entwickler
  - (2) Systemadministratoren
  - (4) Intranet-Nutzer
  - (5) Partner
  - (6) authentifizierte Nutzer
  - (9) anonyme Internetnutzer

Schwachstellen-Faktoren

- Entdeckbarkeit: Wie leicht ist die Schwachstelle für die Gruppe der Angreifer zu finden?
  - (1) nahezu unmöglich
  - (3) schwer
  - (5) leicht
  - (9) automatisierte Tools verfügbar
- Ausnutzbarkeit: Wie leicht ist die tatsächliche Ausnutzung der Schwachstelle für die Gruppe der Angreifer?
  - (1) theoretisch
  - (3) schwer
  - (5) leicht

  - (9) automatisierte Tools verfügbar
- Bekanntheit: Wie gut ist die Schwachstelle bei der Gruppe der Angreifer bekannt?
  - (1) unbekannt
  - (4) versteckt
  - (6) offensichtlich
  - (9) öffentliches Wissen
- Angriffserkennung: Wie wahrscheinlich ist es, dass der Angriff bemerkt wird?
  - (1) aktive Erkennung in der Anwendung
  - (3) protokolliert und überprüft
  - (8) protokolliert ohne Überprüfung
  - (9) nicht protokolliert

Impact Unter Impact verstehen die OWASP-Autoren die Auswirkungen eines erfolgreichen Angriffs. Sie unterscheiden dabei zwischen den technischen Auswirkungen wie zum Beispiel Datenverluste und den geschäftlichen Auswirkungen wie zum Beispiel finanzieller Schaden. Die geschäftlichen Auswirkungen werden dabei als wesentlich kritischer gesehen als die technischen Auswirkungen. Allerdings sind die geschäftlichen Auswirkungen oft deutlich schwerer bezifferbar. Dementsprechend obliegt es dem Nutzer, der die Bewertung des Impacts vornimmt, ob und wie er die beiden Faktorengruppen gewichtet. Von den OWASP-Autoren gibt es den Vorschlag, entweder nur die geschäftlichen Auswirkungen zu berücksichtigen, sofern diese adäquat bestimmbar sind, oder analog zur Likelihood den Durchschnitt aus allen Werten zu bilden [31].

In der folgenden Aufzählung werden zunächst alle technischen Impact-Faktoren und anschließend alle geschäftlichen Impact-Faktoren vorgestellt. Sofern einzelne Werte nicht sinnvoll bestimmbar sind, ist es möglich, anhand der verbleibenden Werte einen Gesamt-Impact-Wert zu bestimmen. Dieser wird daraufhin wieder einer Risikostufe gemäß Tabelle 2.1 zugeordnet. Auch hier wurde die Aufzählung vollständig aus [31] übernommen und ins Deutsche übersetzt.

Technische Faktoren

- Vertraulichkeitsverlust: Wie viele Daten könnten offengelegt werden und wie sensibel sind sie?
  - (2) sehr wenige nicht sensible Daten offengelegt
  - (6) sehr wenige kritische Daten offengelegt
  - (6) sehr viele nicht sensible Daten offengelegt
  - (7) sehr viele kritische Daten offengelegt

  - (9) alle Daten offengelegt
- Integritätsverlust: Wie viele Daten könnten beschädigt werden und wie stark werden sie beschädigt?
  - (1) sehr wenige leicht beschädigte Daten
  - (3) sehr wenige schwer beschädigte Daten
  - (5) viele leicht beschädigte Daten
  - (7) viele schwer beschädigte Daten
  - (9) alle Daten vollständig beschädigt
- Verfügbarkeitsverlust: Wie viele Dienste könnten verloren werden und wie wichtig sind diese?
  - (1) minimale Unterbrechung von Sekundärdiensten
  - (5) minimale Unterbrechung von Primärdiensten
  - (5) umfangreiche Unterbrechung von Sekundärdiensten
  - (7) umfangreiche Unterbrechung von Primärdiensten
  - (9) vollständiger Verlust aller Dienste
- Verlust der Verantwortlichkeit: Können die Angreiferaktionen auf eine konkrete Person zurückgeführt werden?
  - (1) vollständig rückverfolgbar
  - (7) möglicherweise rückverfolgbar
  - (9) vollständig anonym

Geschäftliche Faktoren

- Finanzieller Schaden: Wie hoch ist der finanzielle Schaden, der durch eine Ausnutzung entsteht?
  - (1) niedriger als die Kosten zur Behebung der Schwachstelle
  - (3) geringfügige Auswirkungen auf den Jahresgewinn
  - (7) erhebliche Auswirkungen auf den Jahresgewinn
  - (9) Konkurs
- Rufschaden: Würde eine Ausnutzung zu einer Rufschädigung führen, die dem Unternehmen schaden würde?
  - (1) geringer Schaden
  - (4) Verlust von Großkunden

  - (5) Verlust des Firmenwerts
  - (9) Markenschaden
- Nichteinhaltung: Wie hoch ist das Risiko bei Nichteinhaltung?
  - (2) geringfügiger Verstoß
  - (5) klarer Verstoß
  - (7) schwerwiegender Verstoß
- Verletzung der Privatsphäre: Wie viele personenbezogene Daten könnten offengelegt werden?
  - (3) von einer einzigen Person
  - (5) von hunderten Personen
  - (7) von tausenden von Personen
  - (9) von Millionen von Personen

3. Laufendes Beispiel

Als anschauliches Beispiel für die Erklärungen in dieser Arbeit sowie auch für die Evaluation dient ein beispielhaftes Energienetz, das in Abbildung 3.1 zu sehen ist. Das Netzwerk besteht aus einem Energieverbraucher sowie mehreren Energieproduzenten und einem Umweltsensor. Ursprünglich stammt das gesamte Szenario aus der Fallstudie zu QuAC [25]. Dieses wird hier weiter verwendet und erweitert.

Die beiden Stromproduzenten `Photovoltaics` und `WindTurbine` können bis zu ihrem jeweiligen Maximalwert Strom produzieren und diesen in das verbundene Netzwerk einspeisen (`addCharge`). Je nach Umgebungsbedingung kann die Stromproduktion auch eingeschränkt sein, wenn der ebenfalls verbundene `EnvironmentSensor` Grenzwert-Überschreitungen feststellt. Die entsprechenden Grenzwerte hierfür sind in den einzelnen SEFFs definiert. Mit `EnergyConsumer` gibt es einen Verbraucher für den produzierten Strom. Dieser ist ebenfalls

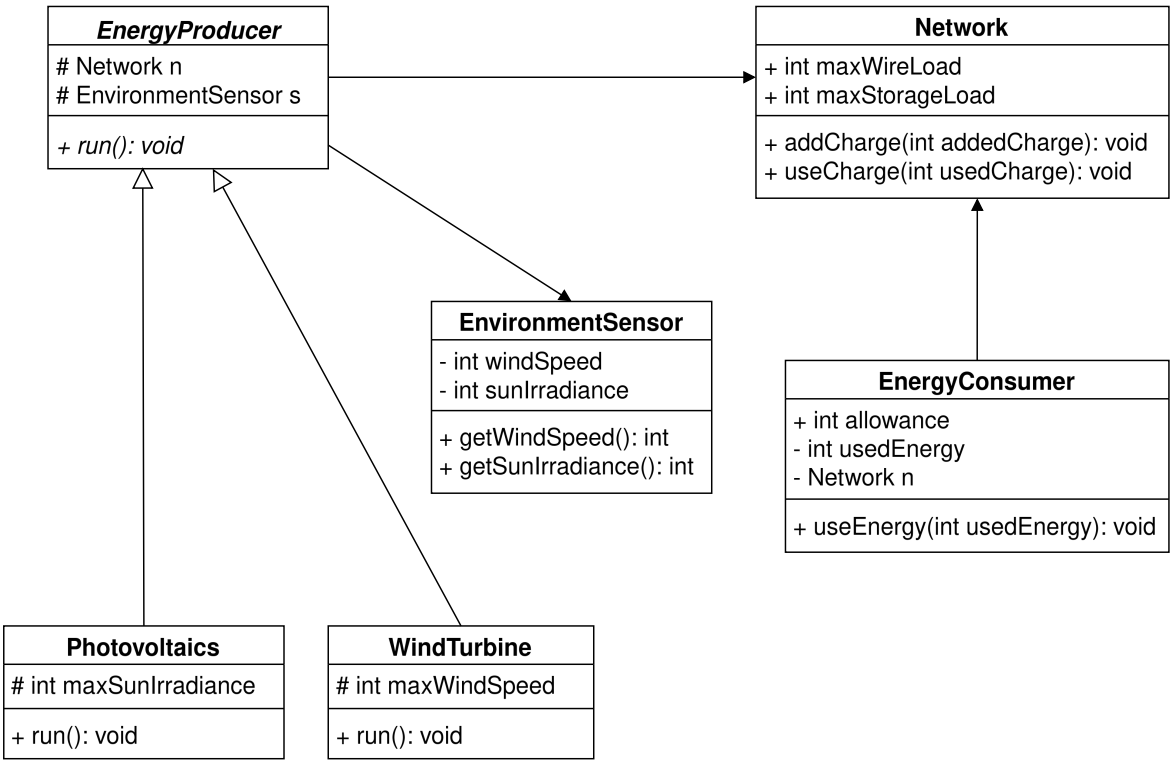

Abbildung 3.1.: Klassendiagramm des Energienetz-Beispielszenarios, auf dem die Beispielangriffe beruhen.

mit dem Netzwerk verbunden und kann über `useCharge` Strom abrufen beziehungsweise nutzen. Die SEFFs für die beiden Netzwerk-Funktionen `addCharge` und `useCharge` sind in den Abbildungen 3.2 und 3.3 zu sehen. Dabei sind jeweils zusätzlich zum eigentlichen Verhalten der Funktion die Coverage Region sowie die benötigten Komponenten und die Zustandsvariablen definiert. Sofern Startwerte für die Zustandsvariablen festgelegt sind, sind auch diese in der SEFF enthalten. Gleiches gilt für die beiden SEFFs der Energieproduzenten `Photovoltaics::run` und `WindTurbine::run` sowie für `EnergyConsumer::useEnergy`. Diese sind den Abbildungen 3.4, 3.5 und 3.6 zu entnehmen.

Passend zu den SEFFs aus QuAC stehen auch die von QuAC berechneten Direct Error Conditions zur Verfügung. Basierend auf diesen werden von QuAC die Fehlerwahrscheinlichkeiten berechnet. Die Direct Error Conditions für die Funktion `Network::useCharge` sind in Listing 3.1 zu sehen. Hierbei stellt jeder Absatz eine eigene Bedingung dar. Insgesamt werden also mehrere verschiedene mögliche Bedingungen abgedeckt. Für die anderen angesprochenen Funktionen sind die zugehörigen Direct Error Conditions in Abschnitt A.1 zu finden.

Ein beispielhafter Angriff auf das Beispiel-Energienetz unter Berücksichtigung beziehungsweise Ausnutzung der Coverage Regions wird in Abbildung 3.7 als Angriffsbaum dargestellt. Das Ziel dieses Angriffsbaums ist es, eine der oben genannten Direct Error Conditions, die im QuAC-Energienetz-Szenario modelliert wurden, auszunutzen. Dementsprechend bezieht sich jedes einzelne Blatt auf eine entsprechend modellierte Schwachstelle, die einen Bezug zu einer Direct Error Condition herstellt.

Ein zweiter Beispielangriff soll etwas realitätsnaher sein und nicht allein darauf abzielen, lediglich eine Direct Error Condition auszunutzen. Hierzu dient eine deutlich erweiterte Modellierung des Angriffsbaums, welche in Abbildung 3.8 zu sehen ist. Als Ziel wurde für diesen zweiten Angriffsbaum „Netz schwächen“ festgelegt. Dieses Ziel ist frei konstruiert, da es für das gegebene Energienetz-Szenario passend erscheint. Während im ersten Beispiel jedes Blatt des Baums aus einer konkreten Coverage Region-Schwachstelle, also im Prinzip einer Direct Error Condition, besteht, existieren im zweiten Beispiel teils mehrere Zwischenschritte. Diese Zwischenschritte, die im weiteren Verlauf als Attack Step Leaf bezeichnet werden, dienen dazu, einen Angriffsschritt darzustellen, der nicht als konkrete Schwachstelle modelliert ist, weil es sich beispielsweise um eine vorbereitende Maßnahme zur Ausnutzung einer Schwachstelle handelt. Bei allen mit „CRVuln“ beschrifteten Blättern handelt es sich um Coverage Region Vulnerabilities, also die konkrete Ausnutzung einer modellierten Schwachstelle. Alle anderen Blätter stellen Attack Step Leafs dar. Für das Teilziel „WindSpeed-Werte manipulieren“ und das im Angriffsbaum als Kindknoten darauf folgende nächste Teilziel „Maximalwert manipulieren (verkleinern)“ existieren beispielsweise drei Attack Step Leafs: Zunächst muss der Sensorwert für die aktuelle Windgeschwindigkeit ausgelesen werden. Anschließend muss Zugriff auf den Code der Windturbine erlangt und die maximal erlaubte Windgeschwindigkeit im Code verringert werden. Sind all diese vorbereitenden Schritte erfolgreich, so sind alle Voraussetzungen dafür gegeben, die konkret modellierte Schwachstelle „WindTurbine::run“ auszunutzen. Eine Beispielbelegung für die Variablen des ersten Beispielangriffs wird in Abschnitt 5.2 beschrieben.

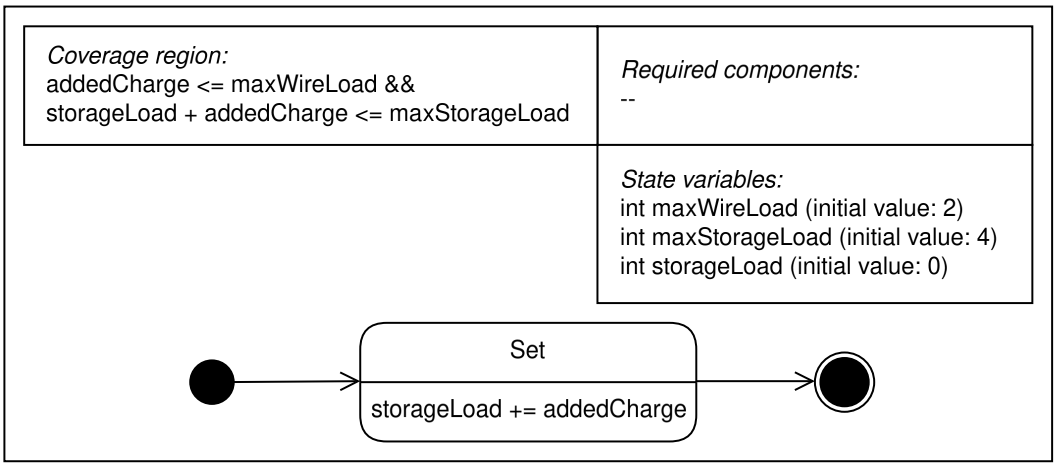

Abbildung 3.2.: SEFF für Network::addCharge aus dem Energienetz-Beispielszenario mit der zugehörigen Coverage Region sowie den Zustandsvariablen und deren Initialwerten.

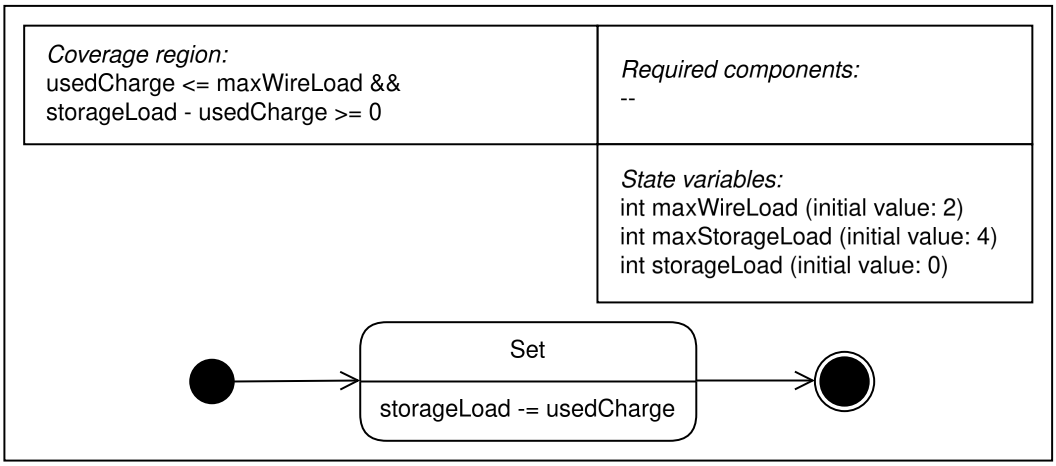

Abbildung 3.3.: SEFF für Network::useCharge aus dem Energienetz-Beispielszenario mit der zugehörigen Coverage Region sowie den Zustandsvariablen und deren Initialwerten.

Aufgrund der Kompaktheit des ersten Beispielangriffs wird vor allem dieser in den folgenden Kapiteln erneut aufgegriffen. Alle Beispiele lassen sich jedoch auch auf den zweiten Angriffsbaum übertragen. Konkret werden die einzelnen modellierten Schwachstellen mit individuellen Likelihood- und Impact-Parametern versehen und es wird anhand der modellierten Parameter eine Einschätzung berechnet, wie groß das Risiko ist, dass der modellierte Angriff erfolgreich ist.

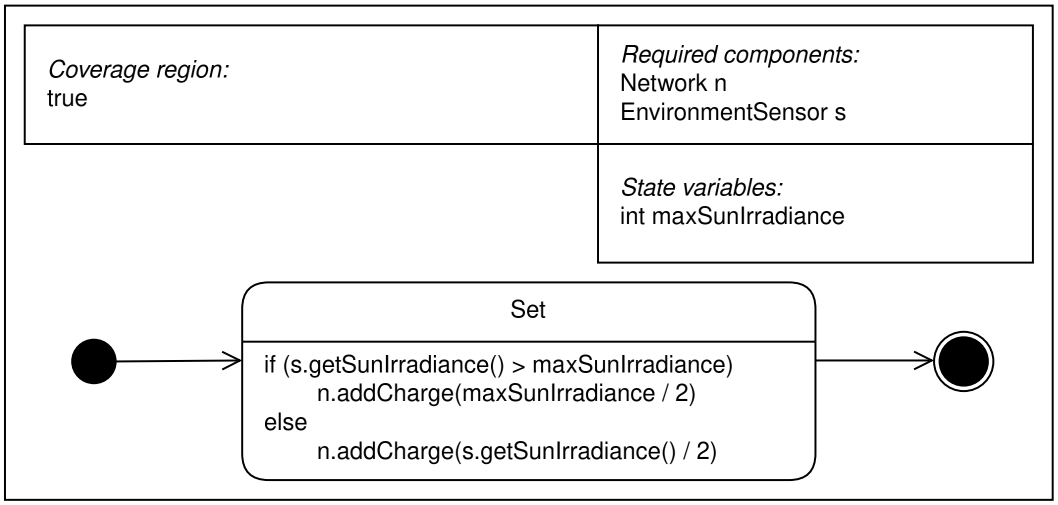

Abbildung 3.4.: SEFF für Photovoltaics::run aus dem Energienetz-Beispielszenario mit der zugehörigen Coverage Region, den benötigten Komponenten sowie den Zustandsvariablen und deren Initialwerten.

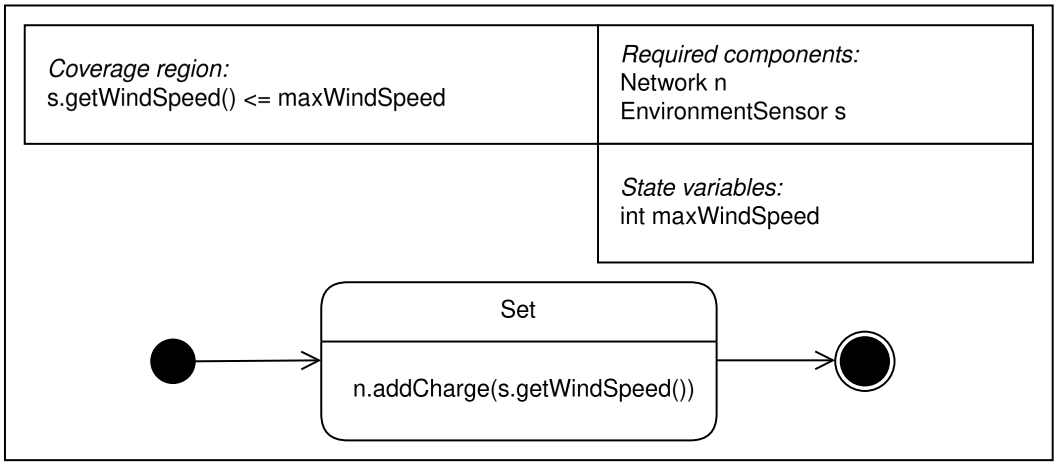

Abbildung 3.5.: SEFF für WindTurbine::run aus dem Energienetz-Beispielszenario mit der zugehörigen Coverage Region, den benötigten Komponenten sowie den Zustandsvariablen und deren Initialwerten.

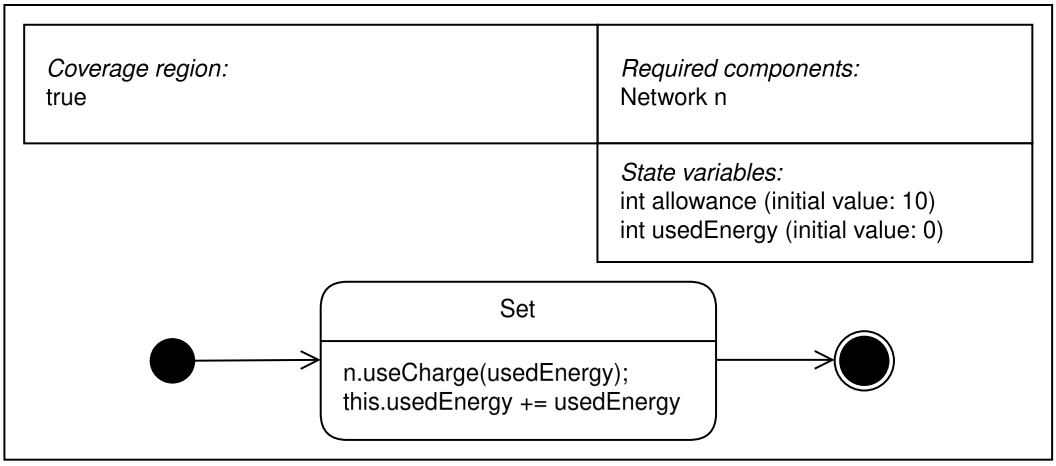

Abbildung 3.6.: SEFF für EnergyConsumer::useEnergy aus dem Energienetz-Beispielszenario mit der zugehörigen Coverage Region, den benötigten Komponenten sowie den Zustandsvariablen und deren Initialwerten.

```
1  this_maxWireLoad.VALUE <= -1 + usedCharge.VALUE
2  AND this_maxStorageLoad.VALUE >= 0
3  AND this_storageLoad.VALUE >= 0
4  AND this_storageLoad.VALUE <= this_maxStorageLoad.VALUE
5
6  this_storageLoad.VALUE <= -1 + usedCharge.VALUE
7  AND this_maxWireLoad.VALUE >= usedCharge.VALUE
8  AND this_maxStorageLoad.VALUE >= 0
9  AND usedCharge.VALUE >= 1
10 AND this_storageLoad.VALUE >= 0
11 AND this_storageLoad.VALUE <= this_maxStorageLoad.VALUE
12
13 usedCharge.VALUE <= -1
14 AND this_storageLoad.VALUE >= 1 + usedCharge.VALUE
15     + this_maxStorageLoad.VALUE
16 AND this_maxStorageLoad.VALUE >= usedCharge.VALUE
17 AND this_storageLoad.VALUE >= usedCharge.VALUE
18 AND this_maxWireLoad.VALUE >= usedCharge.VALUE
19 AND this_maxStorageLoad.VALUE >= 0
20 AND this_storageLoad.VALUE >= 0
21 AND this_storageLoad.VALUE <= this_maxStorageLoad.VALUE
```

Listing 3.1: Im Rahmen der QuAC-Fallstudie berechnete Direct Error Conditions für Network::useCharge [25].

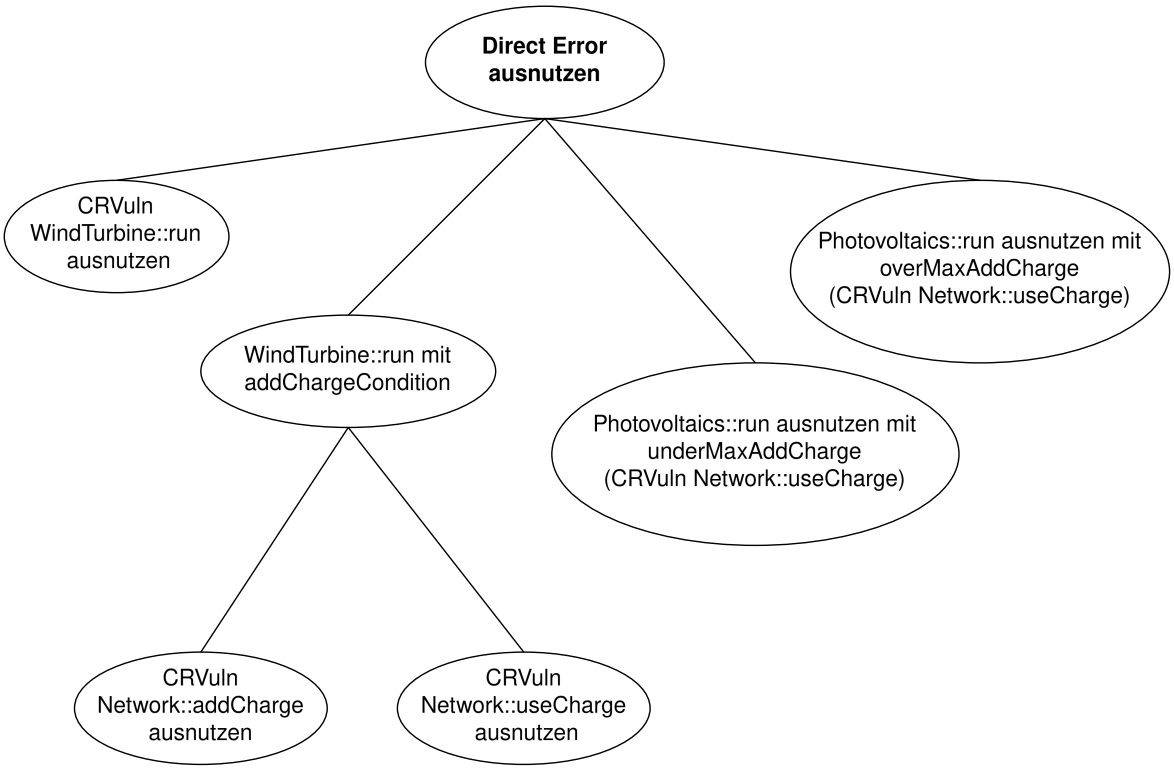

Abbildung 3.7.: Angriffsbaum für das Ziel „Direct Error Condition ausnutzen“ im Energienetz-Beispielszenario.

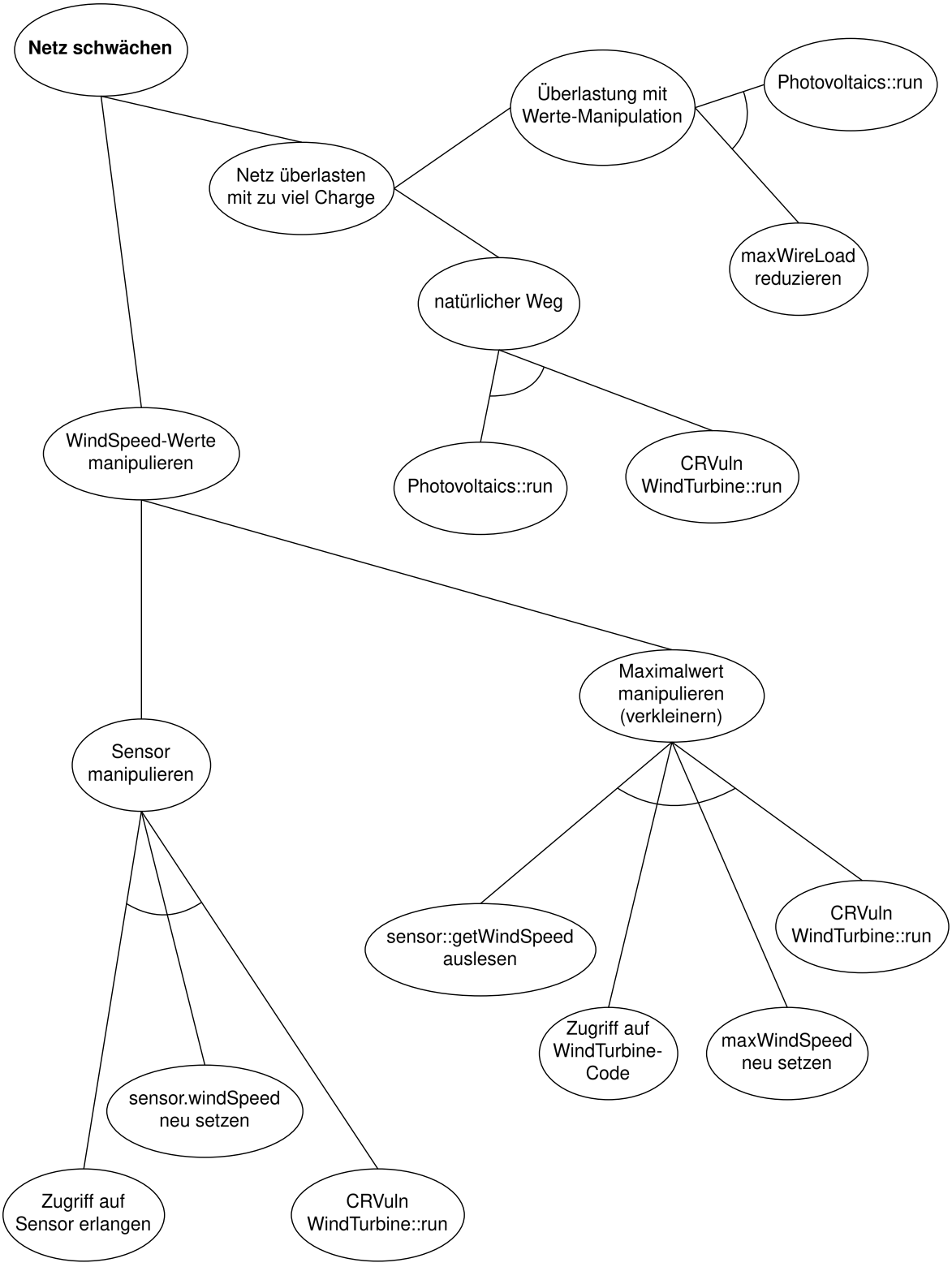

Abbildung 3.8.: Angriffsbaum für das Ziel „Netz schwächen“ im Energienetz-Beispielszenario.

4. Angreifermodellierung

Die vorliegende Arbeit erweitert QuAC um Sicherheitseigenschaften, indem Angriffe auf Coverage Regions betrachtet werden. Hierfür wird eine Modellierung für Angreifer und Angriffe definiert. Voraussetzung für diese Modellierung ist eine bereits erfolgte Modellierung des zu analysierenden komponentenbasierten Softwaresystems mittels des QuAC-Ansatzes.

Ein Angriff besteht aus mindestens einer Schwachstelle, die ausgenutzt wird, sowie einem Angriffsbaum und einem Angreifer, der mittels des Angriffsbaums die Schwachstelle ausnutzt und den Angriff durchführt. Die Nutzung von Angriffsbäumen bietet in diesem Zusammenhang den Vorteil, dass sie einerseits „eine pragmatische Notation“ [27] darstellen, andererseits aber auch inhaltlich alle relevanten Angriffsschritte, Angriffsstrategien und die zugehörigen Parameter adäquat abbilden können [39]. Zunächst wird in Abschnitt 4.1 die Architektur vorgestellt und es werden die getroffenen Architekturentscheidungen dargelegt und begründet. Anschließend wird in Abschnitt 4.2 dargestellt, wie mittels der architektonisch vorgestellten Modellierung ein konkreter Angriff mit all seinen zugehörigen Komponenten modelliert werden kann. Auch die Berechnungsvorschriften für die im theoretischen Abschnitt eingeführten Parameter werden hier weiter ausgeführt.

4.1. Architektur des Metamodells

In Abbildung 4.1 ist das Klassendiagramm des entwickelten Metamodells zu sehen. Es besteht aus einer Oberklasse `Identifiable` sowie mehreren davon abgeleiteten Unterklassen. Mittels der Vererbungsbeziehung zu `Identifiable` ist aufgrund des Attributs `id` sichergestellt, dass alle Objekte eindeutig identifizierbar sind. Auf ein zusätzliches Namensattribut wurde verzichtet, da dieses nicht für alle Klassen erforderlich ist. Wo benötigt, ist ein solches Attribut in den Unterklassen zusätzlich modelliert.

Die Unterklassen lassen sich in drei Bereiche aufteilen: Schwachstellen-Modellierung, Angriffsbaum-Modellierung und Angreifer-Modellierung. In den folgenden drei Unterabschnitten wird jeder Teilbereich separat genauer vorgestellt. In Unterabschnitt 4.1.4 folgt daran anschließend die Klasse `AttackModellingContainer`. Da die Klasse für die Architektur selbst nicht relevant ist, wurde sie der Übersichtlichkeit halber im Klassendiagramm nicht abgebildet. Für die konkrete Modellierung ist sie im weiteren Verlauf als Basisklasse erforderlich.

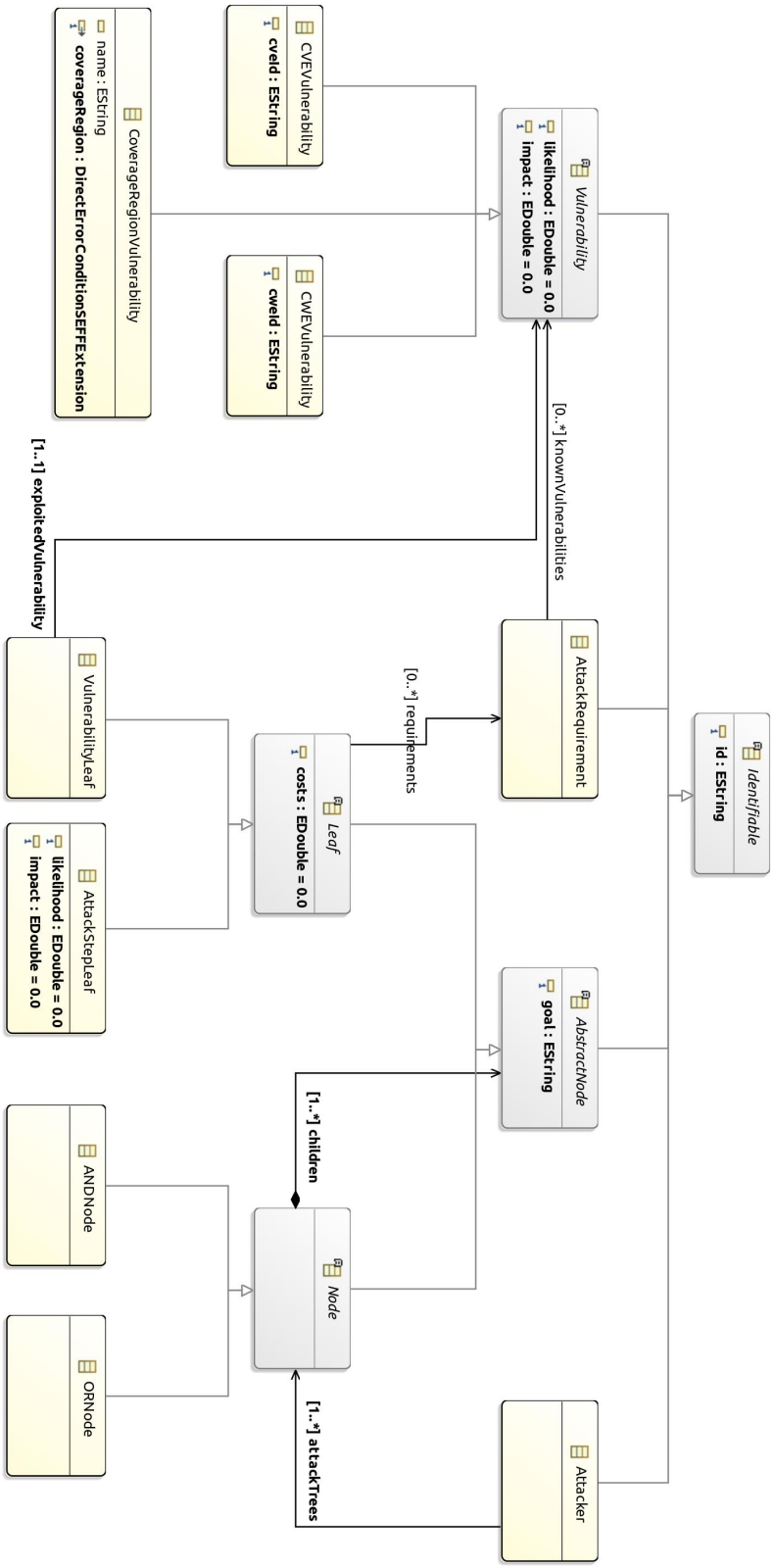

Abbildung 4.1.: Klassendiagramm des Metamodells der Angreifermodellierung, ohne die zusätzliche Klasse AttackModellingContainer.

4.1.1. Schwachstellen

Aufgrund der Zielsetzung der Arbeit, QuAC um Angreifer und Angriffsbäume zu erweitern, und den Bezug zu den dort gefundenen und modellierten Coverage Regions herzustellen, erfolgt an dieser Stelle die Modellierung von Schwachstellen. Dafür ist zunächst eine abstrakte Klasse `Vulnerability` definiert. Zwecks der eindeutigen Identifizierbarkeit und Zuordnungsmöglichkeit erbt `Vulnerability` von der abstrakten Oberklasse `Identifiable`. Als Unterklassen stehen mehrere verschiedene, konkrete Ausprägungen einer solchen Schwachstelle zur Verfügung: `CoverageRegionVulnerability`, `CVEVulnerability` und `CWEVulnerability`. Die `CoverageRegionVulnerability` dient dazu, Schwachstellen zu modellieren, die sich aus den Coverage Regions ergeben. Mittels der beiden anderen Klassen, `CVEVulnerability` und `CWEVulnerability`, soll es möglich sein, Schwachstellen zu modellieren, die nach den in Unterabschnitt 2.4.1 vorgestellten CVE- oder CWE-Systemen klassifiziert sind.

Die konkrete Modellierung enthält für die Oberklasse `Vulnerability` zwei Attribute: `likelihood` und `impact`. Diese sind für die vorgesehene Berechnung des Angriffssicherheitsmaßes sowie insbesondere auch für die daraus folgende Berechnung der OWASP-Bewertung relevant. Deshalb sind die beiden Faktoren bereits in der Oberklasse modelliert und müssen dementsprechend von allen Kindklassen berücksichtigt werden. Genauere Informationen und Vorgaben zur Berechnung der beiden Werte sowie zur Nutzung der OWASP-Bewertung folgen in Unterabschnitt 4.2.1.

Die Modellierung der `CoverageRegionVulnerability` enthält zwei Attribute. Das Attribut `name` dient der Benennung der Schwachstelle zwecks einer leichteren Identifikation in der Ergebnisausgabe. Das Attribut `coverageRegion` stellt eine Referenz auf ein Objekt vom Typ `DirectErrorConditionSEFFExtension` aus der Modellierung von QuAC dar. In diesem Objekt definiert QuAC die in Abschnitt 2.3 beschriebenen Direct Error Conditions. Über die Referenz zu diesem Objekt besteht die Möglichkeit, weitere Analysen zu implementieren, die über den Umfang dieser Arbeit hinausgehen. Insbesondere ist es möglich, die verschiedenen Variablenbelegungen, die zu einem Fehler führen, auszulesen. Damit sind veränderte Analysen oder auch eigene Wahrscheinlichkeitsberechnungen auf Grundlage dieser Variablenbelegungen möglich. Für den Umfang dieser Arbeit dient der Bezug zunächst lediglich der Vollständigkeit und der Referenz zu QuAC.

Die beiden Klassen `CVEVulnerability` und `CWEVulnerability` sind in der Modellierung als Grundlage für eine mögliche Erweiterung vorgesehen. Sie beziehen sich nicht auf eine sich aus einer Coverage Region ergebende Schwachstelle, sondern auf bereits öffentlich bekannte Schwachstellen und Sicherheitslücken. Beide Klassen verfügen über ein ID-Attribut (`cveId` beziehungsweise `cweId`), um die eindeutige Zuordnung zu ermöglichen. Eine ähnliche Modellierung, die hier auch als Vorlage gedient hat, wurde bereits durch Walter u. a. vorgestellt [43]. Auf weitere Implementierungsdetails, beispielsweise die Bewertung der Schwachstellen mittels eines Referenzsystems, wurde zunächst verzichtet, um keine Festlegungen zu treffen, die die Erweiterbarkeit stärker als notwendig einschränken.

4.1.2. Angriffsbäume

Angriffsbäume bestehen, wie bereits in Abschnitt 2.1 dargestellt, aus Knoten und Blättern sowie aus Kanten, die die einzelnen Knoten und Blätter miteinander verbinden. Die Blätter stellen eine konkrete Angriffsaktion dar, während die Knoten ein Teilziel repräsentieren und zusätzlich die Information darüber enthalten, ob die verschiedenen Kindknoten mittels eines logischen ANDs oder eines ORs betrachtet werden müssen [27].

In der Modellierung stellt die abstrakte Klasse `AbstractNode` die Oberklasse für alle Baumelemente dar. Kindklassen hiervon sind die beiden ebenfalls abstrakten Klassen `Leaf` und `Node`. Mittels dieser drei Klassen ist es im weiteren Verlauf möglich, einen Angriffsbaum rekursiv aufzubauen. Von `Node` erben die beiden konkreten Klassen `ANDNode` und `ORNode`, während von `Leaf` die beiden konkreten Klassen `VulnerabilityLeaf` und `AttackStepLeaf` erben.

Alle Knoten haben gemeinsam, dass sie ein Ziel besitzen. Blattknoten stellen einzelne ausführbare Aktionen dar. Das Ziel, das diese Aktion erreichen soll, stellt dabei das Ziel des Blattknotens dar. Alle anderen Knoten verfügen über ein Ziel, das sie mittels der Aktionen oder Teilziele, die in den von ihnen ausgehenden Kindknoten definiert sind, erreichen können. Deshalb ist dieses Ziel bereits als Attribut `goal` in der abstrakten Oberklasse `AbstractNode` modelliert. In der späteren Ergebnisausgabe erleichtert eine korrekte Zielangabe die Lesbarkeit sowie die Verständlichkeit. Zur einfacheren Identifikation im weiteren Verlauf erweitert `AbstractNode` daher die abstrakte Oberklasse `Identifiable`.

Die beiden von `AbstractNode` abgeleiteten Klassen `Leaf` und `Node` dienen primär der verbesserten Struktur der Modellierung sowie der Verständlichkeit. Mit `children` besitzt jeder `Node` eine Containment-Beziehung [11] zu einem oder mehreren Kindknoten und/oder Blättern. Diese folgen dem Knoten im Baum direkt, sie sind also mittels einer Kante miteinander verbunden. Aufgrund der Variabilität ist diese Containment-Relation als `AbstractNode` definiert. Somit können zunächst weitere `Nodes` als `children` definiert werden, bevor abschließend ein `Leaf` folgt. An dieser Stelle ist keine einfache Referenz möglich, sondern zwingend eine Containment-Referenz erforderlich, um zu verhindern, dass einzelne Knoten mehrfach innerhalb der gleichen Rekursion auftauchen und somit zu einer Endlosschleife führen. Die Containment-Referenz bewirkt die direkte Speicherung des jeweiligen `Node` an der entsprechenden Stelle [11]. Der Wurzelknoten eines jeden Angriffsbaums wird implizit durch den Angreifer per Referenz auf einen `Node` festgelegt. Die Details hierzu werden in Unterabschnitt 4.1.3 beschrieben.

Eine explizite Modellierung der Kanten des Angriffsbaums muss die Unterscheidung zwischen AND-Kanten und OR-Kanten enthalten. Die Speicherung der Kindknoten anstelle der ausgehenden Kanten in den Elternknoten führt zum gleichen Ergebnis. Zudem legt die Modellierung von Angriffsbäumen in der Literatur, wie zum Beispiel von Mantel und Probst [27], nahe, dass die Unterscheidung zwischen AND- und OR-Kanten durch die Knoten erfolgen sollte. Daher wurde in dieser Arbeit auf die explizite Modellierung von Kanten verzichtet. Stattdessen stehen hierfür zwei konkrete Implementierungen von `Node` zur Verfügung: `ANDNode` und `ORNode`. Diese verfügen über keine eigenen Attribute oder Referenzen,

werden aber in der Analyse des Angriffsbaums je nach Auswertestrategie unterschiedlich behandelt. Die Modellierung von SAND-Knoten, die von Mantel und Probst [27] zusätzlich zu AND- und OR-Knoten verwendet werden, wurde hier nicht separat implementiert, da sich diese in der Analyse nicht von AND-Knoten unterscheiden. Die Unterscheidung zwischen AND-Knoten und OR-Knoten kann in Abschnitt 2.1 nachgelesen werden.

Endpunkt eines jeden Asts beziehungsweise eines jeden Pfades des Angriffsbaums ist ein Blattknoten, hier modelliert als `Leaf`. Jedes Blatt steht, wie bereits in Abschnitt 2.1 beschrieben, für eine konkrete Aktion. Um möglichst viel Freiraum für mögliche zukünftige Erweiterungen der Modellierung zu lassen, ist im abstrakten `Leaf` lediglich ein Attribut `costs` für die Kosten der Aktion enthalten. Mit diesem sollen die Kosten der konkreten Angriffsaktion definiert werden, also ohne Bezug zu eventuell zuvor erforderlichen Angriffsschritten. Weitere Informationen, insbesondere zum Wertebereich der Kosten, sind Unterabschnitt 4.2.1 zu entnehmen.

`Leaf` verfügt zusätzlich über eine optionale Referenz zu einer oder mehreren Angriffsvoraussetzungen, die in der Modellierung `AttackRequirement` genannt werden. Die Idee von `AttackRequirements` ist, für einzelne Angriffsschritte definieren zu können, dass bestimmte Schwachstellen vorliegen müssen, damit der Angriffsschritt überhaupt ausgeführt werden kann. Hierfür referenziert ein `AttackRequirement` beliebig viele Schwachstellen, die als `Vulnerability` modelliert sind. Zusätzlich wäre es beispielsweise in einer Erweiterung auch möglich, `AttackRequirements` im `Attacker` zu definieren und so festzulegen, welche Fähigkeiten ein Angreifer hat und dies dann in jedem Angriffsschritt abzugleichen: Wenn der Angreifer eine Fähigkeit nicht hat, die für den Angriffsschritt erforderlich ist, wird die Erfolgswahrscheinlichkeit automatisch zu null. Aufgrund der vielfältigen, teils auch etwas komplexen Einsatzmöglichkeiten und des beschränkten Umfangs dieser Abschlussarbeit wurde die Klasse `AttackRequirement` definiert, um verschiedene Optionen aufzuzeigen und eine leichte Implementierung einiger Ansätze zu ermöglichen. Eine konkrete Implementierung und Berücksichtigung in der Analyse konnte aus Zeitgründen in diesem Rahmen jedoch nicht erfolgen. Zusätzlich zu den drei Hauptkomponenten `Attacker`, `AbstractNode` und `Vulnerability` erbt auch `AttackRequirement` von `Identifiable`, um eine eindeutige Identifikation zu ermöglichen.

Als konkrete Implementierungen von `Leaf` sind zwei verschiedene Klassen modelliert: `AttackStepLeaf` und `VulnerabilityLeaf`. Diese unterscheiden sich in ihren Attributen und in ihrem eigentlichen Sinn.

Ein `AttackStepLeaf` modelliert die Vorstufe zur Ausnutzung einer Schwachstelle. Es handelt sich dabei also um eine konkrete Aktion oder eine vorbereitende Maßnahme zur Ausnutzung einer modellierten Schwachstelle. Diese Aktion selbst ist jedoch nicht als separate Schwachstelle modelliert. Eine solche Vorstufe könnte beispielsweise der Kauf von erforderlicher Hardware oder Software sein, die für die Ausnutzung der konkreten Schwachstelle zwingend erforderlich ist. Mit dieser Klasse sind somit Teilschritte eines Angriffs modellierbar, die Teil eines AND-Elternknotens sind, an dessen Ende die Ausnutzung einer Schwachstelle steht. Grundsätzlich ist es auch möglich, AND-Ketten zu erstellen, die nur aus `AttackStepLeafs` bestehen. Da jedoch das eigentliche Ziel dieser Arbeit und auch der Modellierung darin besteht, Angriffe auf Coverage Regions zu betrachten, sollte ein Verzicht

auf ein `VulnerabilityLeaf` in einer AND-Kette gründlich abgewogen werden. Ist dies der Fall, empfiehlt es sich, die Modellierung und insbesondere deren Granularität sowie die einzelnen `AttackStepLeafs` nochmals zu prüfen. Für die Analyse des Angriffsbaums stehen hier bereits die Attribute `likelihood` und `impact` zur Verfügung. Diese müssen modelliert werden, um eine adäquate Berechnung der Analyse zu ermöglichen. Details zur konkreten Berechnung der beiden Faktoren sind Unterabschnitt 4.2.1 zu entnehmen.

Im Gegensatz zum `AttackStepLeaf` bezieht sich ein Blatt, das als `VulnerabilityLeaf` modelliert ist, immer auf eine konkrete Schwachstelle. Es wird also in diesem Blatt genau eine modellierte Schwachstelle ausgenutzt. Diese wird vom Blatt als `exploitedVulnerability` referenziert. Da eine Schwachstelle, wie bereits aufgezeigt, auch in anderen Zusammenhängen genutzt werden kann, und um Missverständnisse zu vermeiden, enthält die Klasse `VulnerabilityLeaf` die beiden Parameter `likelihood` und `impact` nicht selbst als Parameter. Stattdessen nutzt sie die in `Vulnerability` bereits definierten Parameter für diese Zwecke.

4.1.3. Angreifer

Der `Attacker` ist der Startpunkt eines jeden Angriffs. Jeder Angreifer verfügt über einen oder mehrere Angriffsbäume, um seine Ziele zu erreichen. Diese werden in der Modellierung als `attackTrees` referenziert und sind vom Typ `Node`. Diese Referenz zeigt also immer auf den Wurzelknoten des jeweiligen Baums. Wie bereits im vorausgehenden Abschnitt beschrieben, ist durch diese Referenz ein rekursiver Aufbau des Angriffsbaums möglich und es ist sichergestellt, dass es sich beim hier referenzierten Baumelement um einen Knoten und nicht um ein Blatt handelt.

Mehrere verschiedene Angreifer können den gleichen Angriffsbaum referenzieren, also den gleichen Angriffsbaum für einen ihrer Angriffe nutzen. Die Angriffsbäume sind somit wiederverwendbar. An den Ergebnissen ändert sich durch die mehrfache Nutzung der Angriffsbäume nichts. Die mehrfache Nutzung dient derzeit lediglich dem Zweck, eine Modellierung zu ermöglichen, in der mehrere Angreifer das gleiche Ziel besitzen.

Gleichzeitig kann jeder Angreifer über mehrere verschiedene Angriffsbäume und somit auch über mehrere Ziele verfügen. Das bedeutet, dass für einen Angreifer mehrere verschiedene Angriffsbäume definiert werden können, die dann einzeln nacheinander ausgeführt werden. Wahlweise ist es auch möglich, mehrere Angriffsbäume zu definieren und dem Angreifer für die Auswertung nur einen Baum hinzuzufügen. Daraus würde am Ende der Analyse nur die Auswertung für den einen Angriffsbaum resultieren, obwohl mehrere Bäume definiert wurden. In der Modellierung ist es hierdurch außerdem möglich, mehrere Angriffsbäume in einem Angreifer zu kapseln, der dann ausgewertet wird. Die explizite Modellierung des Angreifers sorgt also für zusätzliche Struktur und Übersichtlichkeit in der Modellierung sowie im weiteren Verlauf auch in der Auswertung. Auch für den Angreifer sind verschiedene Erweiterungen denkbar, beispielsweise die Modellierung von Verteidigungsstrategien oder bestimmten Angreiferfähigkeiten, die dann die Berechnung der einzelnen Angriffsbäume beeinflussen.

4.1.4. Container

Zusätzlich zu den in Abbildung 4.1 zu sehenden Klassen existiert im Metamodell eine weitere Klasse namens `AttackModellingContainer`. Diese enthält Containment-Referenzen zu mindestens einem `Attacker`, mindestens einem `AbstractNode`, mindestens einer `Vulnerability` sowie beliebig vielen Referenzen auf die Klasse `AttackRequirement`. Diese Klasse mit den vier Containment-Beziehungen ist erforderlich, um die entsprechenden Elemente für die konkrete Modellierung verfügbar zu machen. Sie dient dort als Basisklasse. Weitere Details hierzu folgen in Unterabschnitt 4.2.3. Im Klassendiagramm in Abbildung 4.1 wurde aus Gründen der Übersichtlichkeit auf die Darstellung dieser Klasse verzichtet.

4.2. Praktische Umsetzung der Modellierung von Angriffen

Nachdem die Architektur der Modellierung erklärt wurde, stellt sich die Frage, wie ein Angriff konkret erstellt werden kann. Damit verbunden ist die Frage, was der Nutzer konkret tun muss, um selbst einen Angriff zu modellieren. Diese Fragen sollen im folgenden Abschnitt beantwortet werden.

4.2.1. Berechnung der erforderlichen Parameter

Für eine präzise und vergleichbare Auswertung ist eine passende Bestimmung der zu modellierenden Parameter erforderlich. Hierfür werden im Folgenden verschiedene Berechnungsvorschriften vorgestellt und festgelegt. Grundsätzlich orientiert sich die gesamte Analyse und Angriffssicherheits-Berechnung am OWASP-System, das in Unterabschnitt 2.4.2 bereits vorgestellt wurde. Die Voraussetzung für die Wahl eines Bewertungssystems für die Angriffssicherheit besteht darin, ein bereits existierendes System zu verwenden, um die Vergleichbarkeit sicherzustellen. Grundsätzlich erfüllen die beiden in Unterabschnitt 2.4.2 vorgestellten Systeme CVSS und OWASP diese Voraussetzung. Beim CVSS gestaltet sich jedoch die Berechnung der einzelnen Parameter und auch des gesamten Ergebniswertes recht kompliziert. Aufgrund der vielen Einflussfaktoren ist die Ergebnisberechnung darüber hinaus etwas unübersichtlich [6]. Für das OWASP-System hingegen gibt es für die beiden Einflussfaktoren `Impact` und `Likelihood` eine klar definierte Berechnungsvorschrift, die auf wenigen, jeweils klar definierten Parametern beruht. Für den Umfang und die Anforderungen dieser Masterarbeit wurde sich daher für die Verwendung des OWASP-Systems entschieden.

4.2.1.1. Likelihood

Likelihood bezeichnet die Erfolgswahrscheinlichkeit des Angriffs beziehungsweise des Angriffsschritts. Diese wird gemäß der OWASP-Vorgaben aus mehreren Faktoren berechnet, welche in der Auflistung in Unterabschnitt 2.4.2 aufgeführt werden. Der Wertebereich für

die Likelihood liegt zwischen null (niedrig) und neun (hoch). Grundsätzlich ist es für die Modellierung in dieser Arbeit auch möglich, die Faktoren individuell und losgelöst von den Vorgaben zu berechnen. Allerdings ist dadurch keine Vergleichbarkeit gegeben und die Berechnung des OWASP-Ratings wird verfälscht. Darüber hinaus ist es möglich, einzelne der OWASP-Faktoren nicht zu berücksichtigen, wenn diese nicht relevant sind oder für den jeweiligen Kontext nicht sinnvoll anwendbar sind. Auch dabei ist jedoch der Einfluss auf die Vergleichbarkeit zu beachten.

4.2.1.2. Impact

Beim Impact liegt der Wertebereich, analog zur Likelihood, zwischen null (niedrig) und neun (hoch). Die zugehörige Berechnung gemäß der OWASP-Faktoren wird in Unterabschnitt 2.4.2 dargestellt. Es erfolgt eine Aufteilung der Faktoren in technische Faktoren und geschäftliche Faktoren. Die Gewichtung und die Entscheidung, ob beide Faktorengruppen berücksichtigt werden sollen, ist dem Anwender überlassen. Grundsätzlich wird von OWASP jedoch empfohlen, sich auf die geschäftlichen Faktoren zu fokussieren, da diese oft die deutlich größere Relevanz haben.

Vergleichbar mit den Vorgaben für die Likelihood ist es in dieser Arbeit auch für den Impact vorgesehen, das arithmetische Mittel aus allen Werten, die für das konkrete Szenario relevant sind, zu bilden. Es kann auf einzelne Faktoren verzichtet oder auch eine vollständig eigene Bewertung verwendet werden. Dabei ist jedoch wie oben zu beachten, dass die Vergleichbarkeit von solchen Lösungen beeinträchtigt wird.

4.2.1.3. Costs

Zusätzlich zu den beiden ausführlich erklärten Faktoren Likelihood und Impact besteht die Möglichkeit, Kosten zu definieren. Auch hier liegt der Wertebereich zwischen null (niedrig) und neun (hoch). Jedoch ist an dieser Stelle ausdrücklich vorgesehen, dass die Kosten zur individuellen Berücksichtigung von möglicherweise zusätzlich hinzukommenden, in Impact und Likelihood nicht ausreichend beachteten Faktoren existieren. Dementsprechend muss jeder Anwender selbst einen geeigneten Rahmen beziehungsweise eine Vergleichbarkeit für die verschiedenen Kosten festlegen, die den Bedürfnissen des jeweiligen Systems entsprechen. Die Berücksichtigung der Kosten ist optional und erfolgt in der Berechnung nicht standardmäßig. Sofern die Kosten nicht angegeben sind, die Berechnung aber mit Kosten ausgeführt wird, werden Kosten von null angenommen. Werden die Kosten berücksichtigt, so wird das arithmetische Mittel aus den Kosten und dem eigentlichen Likelihood-Wert gebildet. Der sich so ergebende Wert wird für alle Berechnungen als Likelihood-Wert verwendet.

4.2.2. Erstellung von Schwachstellen und Angriffsbäumen

Die bereits im vorherigen Abschnitt berechneten Parameter sind nicht ausreichend für die Analyse und Bestimmung des Angriffssicherheitsmaßes. Es müssen zusätzlich Schwachstellen und Angriffsbäume erstellt werden. Darauf wird im Folgenden eingegangen.

Im Rahmen dieser Arbeit existiert eine Implementierung für Schwachstellen: die `CoverageRegionVulnerability`. Diese stellt den Bezug zu einer Coverage Region aus QuAC dar. Dementsprechend ist für die Definition einer Schwachstelle zunächst einmal eine Coverage Region erforderlich. In der Implementierung heißt diese `DirectErrorConditionSEFFExtension`. Der Zusammenhang zwischen einer Coverage Region und einer Direct Error Condition ist in Abschnitt 2.3 erklärt. Die Unterschiede zwischen den beiden Werten sind im weiteren Verlauf für das Verständnis zu beachten.

Steht eine solche `DirectErrorConditionSEFFExtension` zur Verfügung, kann eine `CoverageRegionVulnerability` zum `AttackModellingContainer` hinzugefügt werden. Durch das Hinzufügen wird der neu erstellten Schwachstelle eine eindeutige ID zugewiesen. In den zugehörigen Eigenschaften können die Attribute `impact` und `likelihood` gemäß der oben vorgestellten Berechnungsvorschriften definiert werden. Im Feld `coverageRegion` muss der Bezug zur genannten `DirectErrorConditionSEFFExtension` hergestellt werden. Zuletzt sollte die Schwachstelle benannt werden. Die Benennung erleichtert im weiteren Verlauf die einfachere Identifikation der Schwachstelle. Auch in der Ausgabe des Angriffspfads als Ergebnis der Analyse wird dieser Name verwendet.

Sobald mindestens eine Schwachstelle modelliert ist, kann ein passender Angriffsbaum erstellt werden. Dieser besteht aus einem Wurzelknoten sowie mehreren Kindknoten. Je nachdem, ob es sich beim Wurzelknoten und auch bei allen weiteren, folgenden Knoten um AND- oder OR-Knoten handeln soll, muss die Wahl entsprechend auf einen `ANDNode` oder einen `ORNode` fallen. Über die Containment-Referenz `children` muss mindestens ein Kindknoten beziehungsweise Blatt definiert werden. Auch hier kann die Wahl zwischen den verschiedenen Varianten von `Node` und `Leaf` frei getroffen werden. Ein `Leaf` ist zur Modellierung eines Blatts zu wählen. Ein `AttackStepLeaf` ist, wie bereits in Unterabschnitt 4.1.2 erklärt, dann zu wählen, wenn für den konkreten Angriffsschritt keine Schwachstelle ausgenutzt wird, sondern es sich um einen kleineren, vorbereitenden Schritt handelt. Andernfalls ist ein `VulnerabilityLeaf` zu wählen. Dieses benötigt eine Referenz auf genau eine konkret implementierte Schwachstelle, im Rahmen dieser Arbeit also auf eine Coverage Region Vulnerability. Den Attributen der Blätter sind in diesem Schritt die individuell gemäß der Vorgaben in Unterabschnitt 4.2.1 berechneten Werte zuzuweisen. Im Falle eines `VulnerabilityLeafs` gilt dies auch für die zugehörige `Vulnerability`. `AttackRequirements` sind in den Blättern in der derzeitigen Version nicht zu beachten, da es sich hierbei bisher lediglich um eine vorgesehene Erweiterung der Arbeit handelt, die nicht implementiert ist.

Insgesamt kann auf diese Weise ein rekursiver Angriffsbaum, bestehend aus beliebig vielen Knoten und Blättern, konstruiert und modelliert werden.

4.2.3. Zusammenfügen zu vollständigem Angreifer-System

Um nun die Angriffsmodellierung zu vervollständigen und eine Analyse starten zu können, muss mindestens ein Angreifer erstellt werden. Dieser wird als direktes Kind des `AttackModellingContainers` erstellt. Je nach Szenario können an dieser Stelle auch mehrere Angreifer erstellt werden. Jedem Angreifer muss mindestens ein Angriffsbaum in Form eines `Node` zugeordnet werden. Dabei sind auch mehrere Angriffsbäume pro Angreifer sowie mehrfach die gleichen Angriffsbäume für verschiedene Angreifer möglich.

Für den Beispiel-Angriffsbaum, der in Abbildung 3.7 zu sehen ist, bietet es sich an, zunächst die Schwachstellen zu modellieren, die ausgenutzt werden sollen. Hierzu werden als direkte Kinder des `AttackModellingContainers` drei `CoverageRegionVulnerabilities` erstellt. Diesen werden jeweils die entsprechenden Attribute und Referenzen zugeordnet. Anschließend sollte ein `ORNode` als Wurzelknoten des Angriffsbaums erstellt werden. Dieser erhält das im vorigen Kapitel bereits erwähnte Ziel „Direct Error ausnutzen“ als Goal-Attribut. Nun können für diesen Wurzelknoten mehrere Kindknoten erstellt werden: ein weiterer `ORNode` sowie drei `VulnerabilityLeafs`. Für den neu erstellten `ORNode` sind zwei weitere Kinder als `VulnerabilityLeaf` erforderlich. Den fünf erstellten `VulnerabilityLeafs` können nun die entsprechenden Attribute und Referenzen zugewiesen werden. Dabei ist insbesondere die Referenz zur jeweiligen `CoverageRegionVulnerability` wichtig, da die Parameter `impact` und `likelihood` für die Analyse der zugehörigen `Vulnerability` entnommen werden. Zum Schluss muss für eine erfolgreiche Auswertung der zugehörige Angreifer erstellt werden. Auch dieser wird als direktes Kind des `AttackModellingContainers` erstellt. Im vorliegenden Beispiel erhält er lediglich einen Angriffsbaum, welcher als Referenz auf den oben erstellten Wurzelknoten definiert wird.

Analog dazu erfolgt die Modellierung für das zweite, umfangreichere Beispiel mit dem Angriffsbaum, der in Abbildung 3.8 zu sehen ist. Der einzige Unterschied besteht darin, dass im zweiten Baum zusätzlich `ANDNodes` sowie `AttackStepLeafs` enthalten sind und dass der Baum grundsätzlich etwas größer ist.

Die gesamte, jetzt erfolgte Modellierung wird in einer XML-Datei mit der Dateiendung `.attackmodelling` gespeichert. Die in Kapitel 5 vorgestellte Analyse kann die Modellierung aus dieser Datei auslesen und weiterverarbeiten.

5. Analyse und Berechnung des Angriffssicherheitsmaßes

Für die Auswertung und Analyse der modellierten Angreifer und Angriffsbäume wurde ein separates Tool entwickelt. Damit kann am Ende eine Aussage über die Angriffssicherheit eines zuvor mittels der Vorgaben aus Kapitel 4 modellierten komponentenbasierten Softwaresystems getroffen werden. Konkret wird dies in diesem Kapitel anhand des in Abbildung 3.7 abgebildeten Angriffsbaums, der sich auf das Energienetz-Beispiel aus Abbildung 3.1 bezieht, dargestellt.

Im Folgenden wird zunächst die Architektur des Analysetools vorgestellt und es wird auf verschiedenste Implementierungsdetails eingegangen. Daran anschließend folgt der Bezug zum eben angesprochenen beispielhaften Energienetz-Angriffsbaum aus Abbildung 3.7, indem die Analyse für dieses Beispiel konkret durchgeführt wird.

5.1. Architektur des Analysetools

Das Analysetool ist als Eclipse-Plugin [10] geschrieben. Hierdurch ist die Kompatibilität mit der QuAC-Modellierung sichergestellt. Grundsätzlich sind hier mehrere Aspekte zu berücksichtigen: Die Modellierung muss ausgelesen, die Analyse der Modellierung unter Berücksichtigung der Nutzereingaben beziehungsweise Nutzerwünsche durchgeführt und das Ergebnis auf eine für den Nutzer leicht verständliche Art und Weise ausgegeben werden. Diese Teile werden in den folgenden Unterkapiteln einzeln vorgestellt.

5.1.1. Software-Architektur

Grundsätzlich besteht das Analysetool aus zwei Paketen: `Model` und `View` (voll ausgeschrieben `edu.kit.kastel.sdq.quac.attackmodelling.evaluation.model` beziehungsweise `.view`). Im `Model` sind die verschiedenen Berechnungsstrategien sowie einige Hilfsklassen hierfür gekapselt, während sich in `View` die Klassen für die Nutzerinteraktion und Ergebnisausgabe befinden.

Model Das `Model`-Package stellt verschiedene Schnittstellen und Hilfsklassen zur Verfügung. Um die Schnittstellen erklären zu können, werden zunächst die kleineren Klassen und Hilfsklassen vorgestellt, bevor anschließend die größeren Schnittstellen folgen.

OwaspRating und OwaspValues `OwaspRating` ist eine `Enumeration` und enthält alle fünf Kategorien, die in der OWASP-Bewertung zur Verfügung stehen, als String: `NOTE`, `LOW`, `MEDIUM`, `HIGH` und `CRITICAL`. Die Klasse `OwaspValues` bietet die Möglichkeit, die beiden Einflussfaktoren für die OWASP-Bewertung, Impact und Likelihood, gemeinsam zu speichern. Dies ist für einzelne Algorithmus-Implementierungen von Vorteil. Mittels der OWASP-Abstufungen, die in Tabelle 2.1 zu sehen sind, würde sich für verschiedene Risikowerte möglicherweise die gleiche OWASP-Bewertung ergeben. Durch die explizite Speicherung der Parameter ist es möglich, dennoch den Pfad mit dem höchsten absoluten Risikowert zu bestimmen und für die Bewertung zu nutzen.

EvaluationResult Die Schnittstelle `EvaluationResult` dient der leichten Darstellung des Ergebnisses der Auswertungen. Sie bietet der `View` die Möglichkeit, eine textuelle Repräsentation des Evaluationsergebnisses mittels der Funktion `getEvaluationResult()` zu erhalten. Die Funktionen, die innerhalb des `Model`-Packages für die Erstellung des Ergebnisses relevant sind, werden bewusst nicht in dieser Schnittstelle modelliert, da diese außerhalb des Packages irrelevant sind. Auf eine mögliche Package-interne Erweiterung der Schnittstelle wird zur Einhaltung des sogenannten YAGNI-Prinzips [12] verzichtet. Stattdessen steht eine Schnittstellen-Implementierung bereit, die alle benötigten Funktionen enthält.

AttackPath Die Klasse `AttackPath` stellt Funktionalität zur Speicherung einer Liste von Knoten zur Verfügung. Sie dient dazu, berechnete Angriffspfade zu speichern und diese textuell anzuzeigen. Für die textuelle Ausgabe steht die `toString()`-Methode zur Verfügung. Diese listet alle im Pfad enthaltenen Knoten mittels ID und Name beziehungsweise Ziel auf.

PcmAttackModel Das `PcmAttackModel` stellt die Funktionalität zum Einlesen der Angreifer- und Angriffsmodellierung zur Verfügung, welche in Unterabschnitt 5.1.2 genauer vorgestellt wird.

Configuration Die `Configuration`-Schnittstelle ermöglicht es, alle für die Evaluation notwendigen Parameter zentral zu speichern und auszulesen. Darunter fallen die Evaluationsoptionen `impact`, `likelihood`, `risk` und `rating`, welche jeweils per booleschem Wert gesetzt werden können und dementsprechend später von der Evaluation ausgewertet werden oder nicht. Zusätzlich wird in der `Configuration` gespeichert, ob der `costs`-Faktor berücksichtigt werden soll. Neben diesen Werten, die als boolesche Variablen gesetzt werden können, können der `Configuration` zusätzlich beliebig viele `EvaluationStrategies` und `Attacker` hinzugefügt werden. Für alle Angriffsbäume der hinzugefügten `Attacker` werden in der Evaluation dann alle hinzugefügten `EvaluationStrategies` angewandt.

EvaluationStrategy Die letzte und gleichzeitig wichtigste Schnittstelle ist die `EvaluationStrategy`. Sie stellt alle benötigten Methoden zur Verfügung, um die Auswertung der Modellierung durchzuführen. An dieser Stelle wurde das Strategie-Entwurfsmuster verwendet, um die Verwendung von verschiedenen Auswertestrategien sehr leicht zu ermöglichen. Die Schnittstelle selbst definiert die drei Maximalwerte `MAX_COSTS`, `MAX_IMPACT` und `MAX_LIKELIHOOD` jeweils als Integer mit dem Wert neun, da dies, wie in Unterabschnitt 2.4.2 dargestellt, dem maximal möglichen Wert gemäß der OWASP-Bewertungen entspricht. Die zur Verfügung gestellten Funktionen werden im Folgenden genauer definiert und erklärt:

- `calculateImpact(AbstractNode node): double` - Diese Funktion nimmt den Wurzelknoten des Angriffsbaums entgegen, berechnet den Impact-Wert des Angriffsbaums und gibt diesen als double im Wertebereich $[0, \text{MAX\_IMPACT}]$ zurück.
- `calculateLikelihood(AbstractNode node, boolean withCosts): double` - Diese Funktion benötigt als Parameter den Wurzelknoten des Angriffsbaums, für den die Likelihood berechnet werden soll. Der zweite Parameter `withCosts` dient dazu, der Funktion mitzuteilen, ob der optionale `costs`-Parameter für die Likelihood-Berechnung berücksichtigt werden soll oder nicht. Zur Berücksichtigung muss `withCosts` `true` sein. Als Ergebnis wird der berechnete Likelihood-Wert als `double` im Wertebereich $[0, \text{MAX\_LIKELIHOOD}]$ zurückgegeben.
- `calculateOwasp(AbstractNode node, boolean withCosts): OwaspRating` - Die beiden benötigten Parameter der Funktion entsprechen den gleichen wie bei `calculateLikelihood`: der Wurzelknoten des Angriffsbaum und die Angabe, ob eine Berücksichtigung der zusätzlich modellierten Kosten erfolgen soll. Hier wird als Ergebnis das entsprechende `OwaspRating` zurückgegeben.
- `calculateRisk(AbstractNode node, boolean withCosts): double` - Die Risk-Berechnung erfolgt ebenfalls auf Basis des als Parameter übergebenen Wurzelknotens sowie der Information aus dem Parameter `withCosts`, ob die zusätzlich modellierten Kosten berücksichtigt werden sollen. Die Methode gibt einen `double`-Wert für das Risk zurück, der einem Prozentwert entspricht und in $[0, 1]$ liegt.
- `evaluate(Configuration config): Map<Attacker, List<EvaluationResult>>` - Mit der Verwendung dieser Methode ist es nicht mehr erforderlich, alle zuvor vorgestellten Funktionen einzeln aufzurufen. Stattdessen können alle gewünschten Parameter in der als `config`-Parameter übergebenen `Configuration` festgelegt werden. Der Funktionsaufruf berechnet diese Parameter dann für alle in der `config` enthaltenen Angreifer und Angriffsbäume. Als Ergebnis liefert die Methode für jeden Angreifer eine Liste der Ergebnisse der Auswertung der einzelnen Angriffsbäume als Objekt von Typ `EvaluationResult` zurück.
- `getName(): String` - Jede Auswertestrategie verfügt über eine Bezeichnung, die von dieser Methode als `String` zurückgegeben wird.

Als Implementierung der Schnittstelle dient die abstrakte Klasse `EvaluationStrategyImpl`. Diese kapselt die gemeinsam genutzten Funktionen der beiden folgenden, konkreten Implementierungsstrategien wie beispielsweise die Berechnung von Likelihood oder Impact für

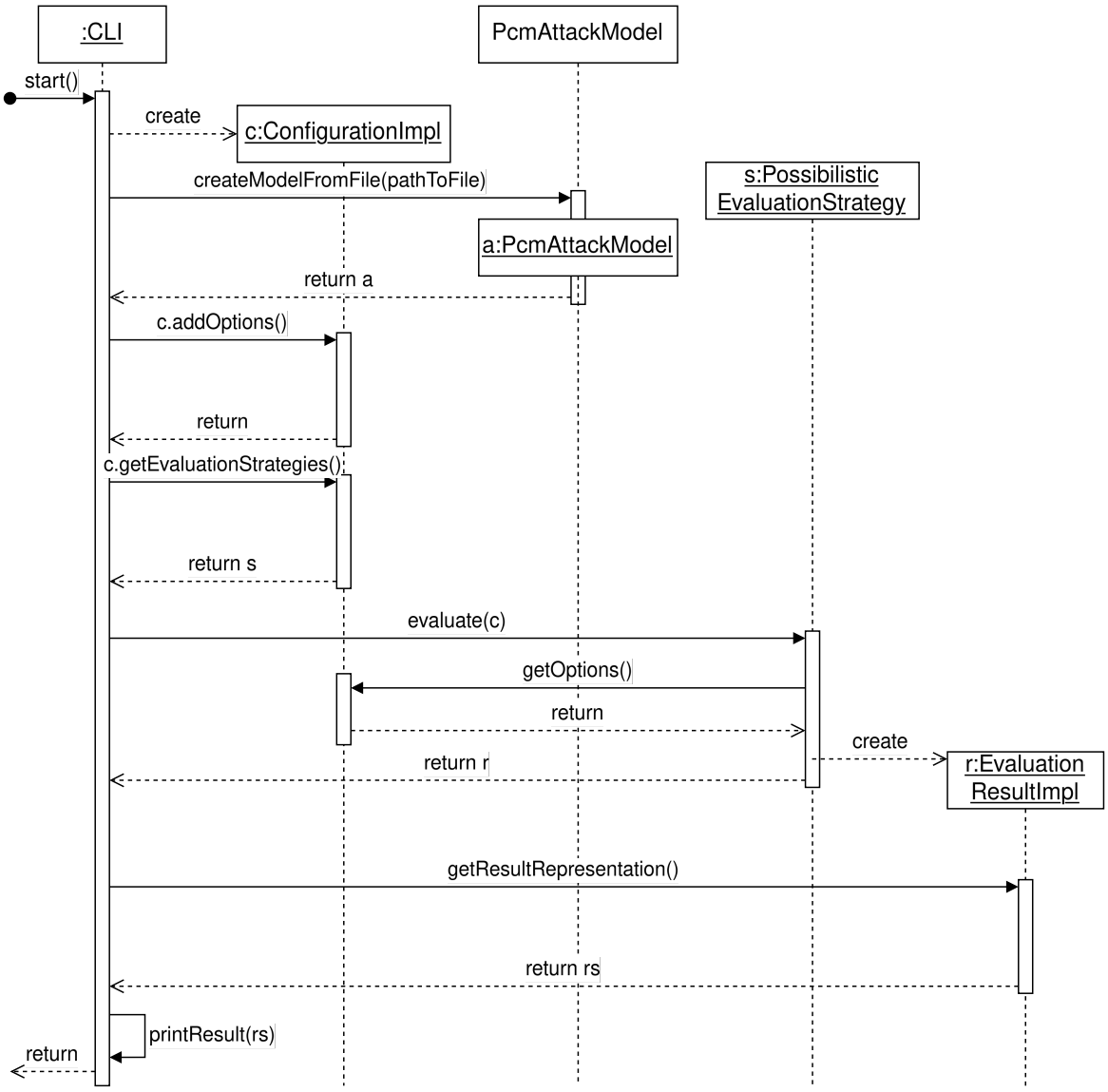

Abbildung 5.1.: Sequenzdiagramm für den beispielhaften Ablauf einer Analyse, vom Start der Analyse über die Auswertung bis hin zur Ergebnisausgabe.

ein `Leaf`. Als konkrete Implementierungen stehen dann die beiden Klassen `PossibilisticEvaluation` und `ProbabilisticEvaluation` zur Verfügung. Die genaue Funktionsweise der in diesen beiden Klassen implementierten Berechnungsalgorithmen wird in Unterabschnitt 5.1.3 vorgestellt.

View Das `View`-Package dient der Interaktion mit dem Nutzer. Hier können Nutzereingaben entgegengenommen und die Evaluationsergebnisse grafisch dargestellt werden. Dazu wird eine Kommandozeilen-Schnittstelle („Command Line Interface“, CLI) genutzt. Diese implementiert die `IApplication`-Schnittstelle [14], wodurch sie innerhalb der Eclipse-Umgebung

als Anwendung ausführbar ist. Weitere Details hierzu sind Unterabschnitt 5.1.4 zu entnehmen.

Die `start()`-Methode wird von Eclipse bei der Programmausführung aufgerufen und ist für die gesamte Abarbeitung des Programmablaufs verantwortlich. Diese Abarbeitung ist beispielhaft und teilweise etwas vereinfacht in Abbildung 5.1 als Sequenzdiagramm dargestellt: Die Angreifermodellierung wird, wie in Unterabschnitt 5.1.2 erläutert, mithilfe der `PcmAttackModel`-Klasse eingelesen, die gewünschte Konfiguration des Nutzers gespeichert und anschließend die Analyse ausgeführt. Über die Methoden `addOptions()` und `getOptions()` werden, wie im Abschnitt zur `Configuration`-Schnittstelle bereits beschrieben, die einzelnen Kommandozeilen-Parameter verwaltet. In der konkreten Implementierung steht eine separate Funktion für jede Option zur Verfügung. Diese wurden zwecks der Anschaulichkeit im Diagramm zu den beiden genannten Funktionen zusammengefasst. Abschließend wird das berechnete Ergebnis durch die `printResult()`-Methode in textueller Form über die Konsole ausgegeben. Hierfür wird die Schnittstelle `EvaluationResult` genutzt, die eine textuelle Repräsentation der Ergebniswerte zur Verfügung stellt. Die `stop()`-Methode gehört zur Implementierung der `IApplication`-Schnittstelle, verfügt jedoch über keine weitere Funktionalität. Details zum Programmstart und den zur Verfügung stehenden Parametern werden in Unterabschnitt 5.1.4 erklärt.

Im `View`-Package existiert des Weiteren eine Implementierung der `Configuration`-Schnittstelle aus dem `Model`-Package. Hier sind standardmäßig alle zur Verfügung stehenden Auswerteoptionen auf `false` gesetzt. Diese Implementierung wird von der `CLI` genutzt, um alle per Kommandozeilen-Argument übergebenen Nutzerparameter sowie die aus der Modellierung ausgelesenen Angreiferdaten zu sammeln und diese gekapselt an die `EvaluationStrategy` zu übergeben.

5.1.2. Auslesen der Modellierung

Um die Analyse starten zu können, muss zunächst die zuvor erstellte Angreifermodellierung aus der `.attackmodelling`-Datei ausgelesen werden. Dies erfolgt in der speziell dafür ausgelegten Klasse `PcmAttackModel`. Die statische Methode `createModelFromFiles` benötigt als Parameter den Pfad zur eben genannten `.attackmodelling`-Datei. Aus diesem Pfad wird ein `URI`-Objekt erstellt. Mittels weiterer Funktionen erfolgt dann das Auslesen des Dateiinhalts, was schlussendlich zu einem `AttackModellingContainer`-Objekt führt. Das Rückgabeobjekt der Methode besteht nun aus einem `PcmAttackModel`-Objekt mit dem soeben ausgelesenen `AttackModellingContainer` als Attribut.

Das so erhaltene Objekt vom Typ `AttackModellingContainer` enthält die gesamte zuvor in der inneren Eclipse-Instanz erstellte Angreifer- und Angriffsmodellierung. Für alle Klassen der Angreifermodellierung, die in Kapitel 4 vorgestellt wird, stehen Getter-Methoden für deren Attribute und Referenzen zur Verfügung. Somit kann für die Analyse auf alle Elemente des Angreifermodells zugegriffen werden.

5.1.3. Berechnungsalgorithmen

Als Ergebnis der Analyse sollen die Wahrscheinlichkeit eines erfolgreichen Angriffs („Likelihood“) sowie die Auswirkungen des Angriffs („Impact“) berechnet werden. Diese beiden Werte werden als Zahl im Wertebereich $[0, 9]$ ausgegeben, wobei null niedrig und neun hoch ist. Somit sind für die folgenden Algorithmen die Werte für `MAX_IMPACT` und `MAX_LIKELIHOOD` auf neun festgelegt. Für die Likelihood wird zudem ein prozentualer Wahrscheinlichkeitswert berechnet und ausgegeben. Zusätzlich zu diesen beiden Werten soll gemäß der Konzeption der Arbeit eine Aussage über die Angriffssicherheit des Systems möglich sein. Hierzu wird die in Abschnitt 2.4 vorgestellte Risiko-Bewertungsmethode OWASP genutzt. Zum einen wird das Risiko jedes Angriffs als $(impact \cdot likelihood)/(\text{MAX\_IMPACT} \cdot \text{MAX\_LIKELIHOOD})$ berechnet. Diese Berechnung ergibt einen prozentualen Wert. Zum anderen wird der Schweregrad des Risikos mittels der durch das OWASP-System vorgegebenen Berechnungsmethode ausgegeben.

All diese Werte können sowohl mittels einer probabilistischen als auch einer possibilistischen Auswertestrategie berechnet werden. Die genaue Definition der beiden Auswertestrategien kann Abschnitt 1.1 entnommen werden. Im Folgenden werden die konkreten Berechnungsalgorithmen für die vier genannten Werte für beide Auswertestrategien detailliert vorgestellt.

Mit der possibilistischen Auswertestrategie ist es zudem möglich, den zugehörigen Pfad innerhalb des Angriffsbaums auszugeben, der für das jeweilige Ergebnis verwendet wurde. Dementsprechend wird der Pfad hier für alle Parameter zusätzlich als Ausgabe dargestellt. Kommen mehrere Pfade innerhalb eines Angriffsbaums infrage, da sie identische Ergebniswerte besitzen, wird grundsätzlich immer der zuerst besuchte Pfad für das Ergebnis verwendet. Dies entspricht im Baum dem linkesten Pfad. Aus Gründen der Übersichtlichkeit wurde in den folgenden Algorithmen auf den Abdruck des Codes für die Pfadausgabe verzichtet. Diese stellt aber lediglich eine Erweiterung der Algorithmen dar und verändert diese inhaltlich nicht.

Impact Algorithmus 1 zeigt die possibilistische Impact-Berechnung. Ausgangspunkt ist der Wurzelknoten, der als `node` übergeben wird. Ausgehend von diesem wird rekursiv für alle Kindknoten der Impact berechnet. Hierbei wird vorausgesetzt, dass pro Pfad nur ein einziger Impact-Wert existiert, da ausschließlich Blattknoten über einen Impact-Wert verfügen. Dementsprechend speichert der Algorithmus lediglich den höchsten Impact-Wert und gibt diesen als Ergebnis zurück. Es erfolgt keine Unterscheidung zwischen AND- und OR-Knoten, da auch für Werte unterhalb eines AND-Knotens ausschließlich der höchste Impact-Wert berücksichtigt werden würde. In einer Kette von mehreren zwingend aufeinanderfolgenden Aktionen wird dieser höchste Impact-Wert auf jeden Fall auftreten. Die Verrechnung mit den übrigen Werten innerhalb der Konjunktion würde folglich zu einem falsch-niedrigen Impact-Wert führen.

Die probabilistische Impact-Berechnung, die in Algorithmus 2 zu sehen ist, berücksichtigt grundsätzlich den Impact-Wert jedes Blattknotens und führt eine rekursive Berechnung durch. Handelt es sich beim aktuell betrachteten Knoten um einen OR-Knoten, werden die

Algorithmus 1 Possibilistische Impact-Berechnung

```
1:  function calculateImpact(node)
2:      if node is instance of Node then
3:          highestImpact ← 0
4:          for all n ∈ node.children do
5:              impact ← calculateImpact(n)
6:              if impact > highestImpact then
7:                  highestImpact ← impact
8:              end if
9:          end for
10:         return highestImpact
11:     end if
12:     if node is instance of Leaf then
13:         return node.impact
14:     end if
15:     return 0.0
16: end function
```

Impact-Werte aller Kindknoten aufsummiert und durch die Anzahl dieser geteilt. Es wird also das arithmetische Mittel gebildet. Dieser Mittelwert bezieht sich auf die direkt nachfolgenden Kindknoten, welche wiederum den Mittelwert ihrer Kindknoten berechnen. Ein sehr weit oben im Baum liegendes Blatt hat somit tendenziell ein deutlich größeres Gewicht im Impact-Wert als ein sehr tief im Baum liegendes, durch mehrere Knoten verschachteltes Blatt. Grundsätzlich wird der Mittelwert für OR-Knoten verwendet, da hier alle Kindknoten als Kandidaten für den Angriff unabhängig voneinander in Frage kommen und mit der probabilistischen Auswertestrategie das durchschnittlich zu erwartende Ergebnis berechnet werden soll.

Bei AND-Knoten verhält sich dies, wie im possibilistischen Fall bereits beschrieben, etwas anders: Es wird nur der höchste Impact-Wert unter den Kindknoten berücksichtigt, da dieser Wert aufgrund der AND-Kette auf jeden Fall auftreten wird.

Likelihood Als Likelihood werden der Ergebniswert und auch die jeweiligen in der Angreifermodellierung berechneten Likelihood-Werte gemäß der Vorgaben aus Unterabschnitt 4.2.1 im Wertebereich $[0, 9]$ verstanden. In der Ergebnisausgabe ist ein weiterer Wert enthalten, der als „Probability“ bezeichnet wird. Dieser Wert entspricht im Grunde dem gleichen Ergebnis. Jedoch handelt es sich hierbei um einen prozentualen Wert. Die Umrechnung vom Likelihood- zum Probability-Wert erfolgt mittels einer Division des Likelihood-Wertes durch `MAX_LIKELIHOOD`.

Im Gegensatz zur possibilistischen Impact-Berechnung unterscheidet die possibilistische Likelihood-Berechnung zwischen den verschiedenen Knoten-Typen. Für OR-Knoten wird mittels der Hilfsfunktion `selectBestLikelihoodChild` der beste Kindknoten bestimmt und dessen Likelihood-Wert zurückgegeben. Die genannte Hilfsfunktion, die in Algorithmus 6 zu sehen ist, berechnet rekursiv für alle Kinder des aktuellen Knotens die Likelihood, speichert

Algorithmus 2 Probabilistische Impact-Berechnung

```
1:  function calculateImpact(node)
2:      if node is instance of Node then
3:          children ← node.children
4:          if node is instance of ORNode then
5:              sum ← 0
6:              for all n ∈ children do
7:                  sum ← sum + calculateImpact(n)
8:              end for
9:              return sum / children.size
10:         end if
11:         if node is instance of ANDNode then
12:             highestImpact ← 0
13:             for all n ∈ children do
14:                 impact ← calculateImpact(n)
15:                 if impact > highestImpact then
16:                     highestImpact ← impact
17:                 end if
18:             end for
19:             return highestImpact
20:         end if
21:     end if
22:     if node is instance of Leaf then
23:         return node.impact
24:     end if
25:     return 0.0
26: end function
```

alle Ergebnisse zwischen und gibt den Knoten mit dem besten Ergebnis zurück. Zwecks einer Laufzeitverbesserung im Falle eines mehrfachen Vorkommens des gleichen Knotens wird das Ergebnis dieser Berechnung zusätzlich separat gespeichert. Wird die Funktion dann mit dem gleichen Knoten erneut aufgerufen, muss keine rekursive Berechnung für alle Kinder mehr erfolgen. Stattdessen wird der zuvor bereits ermittelte beste Kindknoten zurückgegeben.

Handelt es sich beim vorliegenden Knoten um einen AND-Knoten, so werden die Likelihood-Werte aller Kindknoten als Prozentwerte miteinander multipliziert, da es sich um eine zwingend aufeinanderfolgende Kette von Wahrscheinlichkeiten handelt. Anschließend wird der Ergebniswert erneut auf den Wertebereich $[0, 9]$ hochskaliert.

Die Likelihood-Berechnung für Blätter ist in Algorithmus 5 zu sehen. Hier werden die Likelihood- und Kosten-Werte aus dem entsprechenden Blatt oder der zugehörigen Schwachstelle ausgelesen. Sofern die zusätzlichen Kosten mit berücksichtigt werden sollen, werden diese anschließend mit dem ausgelesenen Likelihood-Wert multipliziert und durch zwei geteilt. Daraus lässt sich schließen, dass die Kosten lediglich eine weitere Einschränkung

der Likelihood zulassen und der Maximalwert mit dem eigentlichen Likelihood-Wert gleichbedeutend ist. Sollen die Kosten nicht berücksichtigt werden, wird an dieser Stelle nur der ausgelesene, zuvor modellierte Likelihood-Wert zurückgegeben.

Die probabilistische Likelihood-Berechnung aus Algorithmus 4 unterscheidet sich im Falle eines OR-Knotens nicht von der entsprechenden Impact-Berechnung: Es wird der arithmetische Mittelwert über alle Kindknoten gebildet. Im Falle eines AND-Knotens werden, analog zur possibilistischen Berechnung, alle Likelihood-Werte der Kindknoten als Prozentwert miteinander multipliziert. Das Ergebnis wird wiederum auf den Wertebereich $[0, 9]$ skaliert. Die Likelihood-Berechnung der Blattknoten entspricht dem oben bereits vorgestellten Algorithmus `calculateLeafLikelihood`.

Algorithmus 3 Possibilistische Likelihood-Berechnung

```
1:  function calculateLikelihood(node, withCosts)
2:      if node is instance of Node then
3:          children ← node.children
4:          if node is instance of ORNode then
5:              nextNode ← selectBestLikelihoodChild(node, children, withCosts)
6:              return calculateLikelihood(nextNode, withCosts)
7:          end if
8:          if node is instance of ANDNode then
9:              probability ← 1
10:             for all n ∈ children do
11:                 probability ← probability · calculateLikelihood(n,withCosts) / MAX_LIKELIHOOD
12:             end for
13:             return probability · MAX_LIKELIHOOD
14:         end if
15:     end if
16:     if node is instance of Leaf then
17:         return calculateLeafLikelihood(node, withCosts)
18:     end if
19:     return 0.0
20: end function
```

Risk Grundsätzlich berechnet sich das Risiko, hier Risk genannt, als Produkt aus Likelihood und Impact und wird als Prozentwert angegeben. Der Algorithmus für die possibilistische Risk-Berechnung entspricht genau dem Algorithmus der possibilistischen Likelihood-Berechnung, weshalb an dieser Stelle auf den expliziten Abdruck verzichtet wurde. Die im Likelihood-Algorithmus notwendige Skalierung auf den Wertebereich $[0, 1]$ ist hier nicht erforderlich, da dieser Wertebereich standardmäßig eingehalten wird. Auch hier steht die Hilfsfunktion `selectBestRiskChild` zur Verfügung, die in Algorithmus 7 zu sehen ist. Die Hilfsfunktion `calculateLeafRisk` liest lediglich die Werte aus dem entsprechenden Blatt beziehungsweise der zugehörigen Schwachstelle aus und ruft damit die Funktion `calculateSingleRisk` auf, die in Algorithmus 8 zu sehen ist. Diese berechnet den Risk-Wert entsprechend der oben bereits genannten Formel aus Produkt von Impact und Likelihood.

Algorithmus 4 Probabilistische Likelihood-Berechnung

```
1:  function calculateLikelihood(node, withCosts)
2:      finalResult ← 1
3:      if node is instance of Node then
4:          children ← node.children
5:          if node is instance of ORNode then
6:              sum ← 0
7:              for all n ∈ children do
8:                  sum ← sum + calculateLikelihood(n, withCosts)
9:              end for
10:             return sum / children.size
11:         end if
12:         if node is instance of ANDNode then
13:             probability ← 1
14:             for all n ∈ children do
15:                 probability ← probability · calculateLikelihood(n,withCosts) / MAX_LIKELIHOOD
16:             end for
17:             return probability · MAX_LIKELIHOOD
18:         end if
19:     end if
20:     if node is instance of Leaf then
21:         return calculateLeafLikelihood(node, withCosts)
22:     end if
23:     return 0.0;
24: end function
```

Um den Wertebereich $[0, 1]$ einzuhalten, muss dieser Wert anschließend durch das Produkt aus `MAX_IMPACT` und `MAX_LIKELIHOOD` geteilt werden.

Auch die probabilistische Risk-Berechnung entspricht genau dem probabilistischen Likelihood-Algorithmus mit den angesprochenen, ausgetauschten Funktionen und der ausbleibenden Umrechnung des Wertebereichs. Deshalb wird dieser Algorithmus ebenfalls nicht explizit dargestellt.

OWASP Die Berechnung der OWASP-Bewertung gestaltet sich für den probabilistischen Fall sehr leicht: Es wird für beide Einflussfaktoren, also Impact und Likelihood, lediglich die jeweilige Berechnungsfunktion aufgerufen: `calculateImpact` und `calculateLikelihood`. Mit den beiden Ergebniswerten wird dann der Ergebniswert gemäß den in Abschnitt 2.4 vorgestellten Vorgaben berechnet und zurückgegeben.

Die possibilistische OWASP-Berechnung stellt sich dafür etwas komplizierter dar, da für jeden Knoten Likelihood und Impact gemeinsam betrachtet werden müssen und jeweils der insgesamt höchste Wert, also der höchste Risk-Wert, verwendet werden muss. Da als Rückgabewert von der `EvaluationStrategy`-Schnittstelle ein `OwaspRating` erwartet wird,

Algorithmus 5 Likelihood-Berechnung für ein einzelnes Blatt

```
1:  function calculateLeafLikelihood(node, withCosts)
2:      if node is instance of AttackStepLeaf then
3:          likelihood ← node.likelihood
4:          costs ← node.costs
5:          return calculateSingleLikelihood(costs, likelihood, withCosts)
6:      else if node is instance of VulnerabilityLeaf then
7:          likelihood ← node.exploitedVulnerability.likelihood
8:          costs ← node.costs
9:          return calculateSingleLikelihood(costs, likelihood, withCosts)
10:     else
11:         return 0.0
12:     end if
13: end function
14:
15: function calculateSingleLikelihood(costs, likelihood, withCosts)
16:     if withCosts then
17:         return (costs+likelihood)/2
18:     else
19:         return likelihood
20:     end if
21: end function
```

Algorithmus 6 Possibilistische Auswahl des besten Likelihood-Kindknotens

```
1:  bestLikelihoodChilds ← {}
2:  function selectBestLikelihoodChild(currentNode, children, withCosts)
3:      if bestLikelihoodChilds.containsKey(currentNode) then
4:          return bestLikelihoodChilds[currentNode]
5:      end if
6:      results ← {}
7:      for all n ∈ children do
8:          results[n] ← calculateLikelihood(n, withCosts)
9:      end for
10:     bestNode ← results.firstKey
11:     bestProb ← 0
12:     for all (node, prob) ∈ results do
13:         if prob > bestProb then
14:             bestProb ← prob
15:             bestNode ← node
16:         end if
17:     end for
18:     bestLikelihoodChilds[currentNode] ← bestNode
19:     return bestNode
20: end function
```

Algorithmus 7 Possibilistische Auswahl des besten Risk-Kindknotens

```
1:  bestRiskChilds ← {}
2:  function selectBestRiskChild(currentNode, children, withCosts)
3:      if bestRiskChilds.containsKey(currentNode) then
4:          return bestRiskChilds[currentNode]
5:      end if
6:      results ← {}
7:      for all n ∈ children do
8:          results[n] ← calculateRisk(n, withCosts)
9:      end for
10:     bestNode ← results.firstKey
11:     highestRisk ← 0.0
12:     for all (node, risk) ∈ results do
13:         if risk > highestRisk then
14:             highestRisk ← risk
15:             bestNode ← node
16:         end if
17:     end for
18:     bestRiskChilds[currentNode] ← bestNode
19:     return bestNode
20: end function
```

Algorithmus 8 Risk-Berechnung für ein einzelnes Blatt

```
1:  function calculateSingleRisk(impact, probability)
2:      return (impact·probability) / (MAX_LIKELIHOOD·MAX_IMPACT)
3:  end function
```

die Berechnung jedoch rekursiv funktioniert und auf `OwaspValues` beruht, ist die Funktion in Algorithmus 9 und Algorithmus 10 aufgeteilt.

Zum Algorithmus selbst: Für einen OR-Knoten ist das Verhalten ähnlich zu den bereits beschriebenen Implementierungen der anderen Werte: Es wird mittels der Hilfsfunktion `selectBestOwaspChild` der Kindknoten mit dem besten Ergebnis berechnet und zurückgegeben. Diese Hilfsfunktion ist in Algorithmus 11 zu sehen. Es wird also für alle Kinder das Risk berechnet und das Kind mit dem höchsten Risk zurückgegeben.

Im Falle eines AND-Knotens muss zwischen Impact und Likelihood unterschieden werden. Die Likelihood-Werte aller Kindknoten werden mit dem Faktor 1/MAX_LIKELIHOOD miteinander multipliziert. Der Skalierungsfaktor ist erforderlich, um Wahrscheinlichkeiten im Wertebereich $[0, 1]$ miteinander multiplizieren zu können. Das Ergebnis wird abschließend wieder mit `MAX_LIKELIHOOD` multipliziert, um ein Ergebnis im ursprünglichen Wertebereich $[0, 9]$ zu erhalten. Aus den Impact-Werten wird der höchste Wert bestimmt und als Impact für den AND-Knoten verwendet. Die Kombination dieser beiden Werte ergibt die OWASP-Werte für den AND-Knoten.

Die hier nicht abgedruckte Funktion `calculateLeafOwasp` liest, wie bereits ihre Vorgängerfunktionen für Impact, Likelihood und Risk, die Faktoren Impact, Likelihood und Costs aus dem Blatt beziehungsweise der zugehörigen Schwachstelle aus. Für den Fall, dass die zusätzlichen Kosten berücksichtigt werden sollen, berechnet sie den neuen Likelihood-Wert beziehungsweise ruft den oben hierfür vorgestellten Algorithmus `calculateSingleLikelihood` auf. Anschließend gibt sie die Ergebniswerte als Tupel `(impact, likelihood)` zurück, was einem `OwaspValues`-Objekt entspricht.

Algorithmus 9 Possibilistische Berechnung der OWASP-Bewertung

```
1: function calculateOwasp(node, withCosts)
2:     value ← calculateOwaspSteps(node, withCosts)
3:     return calculateOwaspCategory(value.impact, value.likelihood)
4: end function
```

Algorithmus 10 Possibilistische Berechnung der OWASP-Faktoren

```
1: function calculateOwaspSteps(node, withCosts)
2:     if node is instance of Node then
3:         children ← node.children
4:         if node is instance of ORNode then
5:             nextNode ← selectBestOwaspChild(node, children, withCosts)
6:             return calculateOwaspSteps(nextNode, withCosts)
7:         end if
8:         if node is instance of ANDNode then
9:             currentImpact ← 0
10:            currentLikelihood ← 1
11:            for all n ∈ children do
12:                value ← calculateOwaspSteps(n, withCosts)
13:                if value.impact > currentImpact then
14:                    currentImpact ← value.impact
15:                end if
16:                currentLikelihood ← currentLikelihood · value.likelihood / MAX_LIKELIHOOD
17:            end for
18:            currentLikelihood ← currentLikelihood · MAX_LIKELIHOOD
19:            return (currentImpact, currentLikelihood)
20:        end if
21:    end if
22:    if node is instance of Leaf then
23:        return calculateLeafOwasp(node, withCosts)
24:    end if
25:    return (0, 0)
26: end function
```

Algorithmus 11 Possibilistische Auswahl des besten OWASP-Kindknotens

```
1:  function selectBestOwaspChild(node, children, withCosts)
2:      results ← {}
3:      for all n ∈ children do
4:          results[n] ← calculateOwaspSteps(n, withCosts)
5:      end for
6:      bestNode ← results.firstKey()
7:      highestRisk ← 0.0
8:      for all (node, value) ∈ results do
9:          if value.impact · value.likelihood > highestRisk then
10:             highestRisk ← value.impact · value.likelihood
11:             bestNode ← node
12:         end if
13:     end for
14:     return bestNode
15: end function
```

5.1.4. Kommandozeilen-Schnittstelle

Grundsätzlich ist für die Ausführung der Analyse eine Kommandozeilen-Schnittstelle („CLI“) vorgesehen. Für den Umfang dieser Arbeit beschränkt sich diese jedoch darauf, dass die CLI-Parameter der Anwendungsausführung innerhalb der Eclipse-Umgebung als Argumente übergeben werden. Die Ausgabe der Berechnungsergebnisse erfolgt über die Eclipse-Konsole. Eine eigenständige Ausführung der Analyse über die Kommandozeile ist nicht möglich.

Im Folgenden werden die verschiedenen Kommandozeilen-Parameter einzeln vorgestellt und erklärt. Jeder Parameter hat eine Abkürzung, die nach dem Komma angegeben wird.

- `-model`, `-m`: Diese Option ist für die Analyse zwingend erforderlich. Angehängt an die Option muss der vollständige Pfad zur `.attackmodelling`-Datei, deren Modellierung ausgewertet werden soll, übergeben werden. Es erfolgt die Auswertung für alle in dieser Datei modellierten Angreifer und somit auch für alle Angriffsbäume eines jeden Angreifers.
- `-impact`, `-i`: Für die Angriffsbäume soll der jeweilige Impact, also die Auswirkungen beziehungsweise die Konsequenzen des Angriffs, berechnet und ausgegeben werden.
- `-likelihood`, `-l`: Für die Angriffsbäume soll die jeweilige Likelihood, also die Wahrscheinlichkeit eines erfolgreichen Angriffs, berechnet und ausgegeben werden.
- `-owasp`, `-o`: Für die Angriffsbäume soll die jeweilige OWASP-Bewertung, also die Einschätzung des Risikos gemäß der OWASP-Kriterien, unterteilt in `NOTE`, `LOW`, `MEDIUM`, `HIGH` und `CRITICAL`, berechnet und ausgegeben werden.
- `-risk`, `-r`: Für die Angriffsbäume soll das jeweilige Risiko berechnet und ausgegeben werden. Dieses setzt sich aus Likelihood und Impact zusammen.

- `-possibilistic`, `poss`: Für die gewählten Parameter soll die Analyse und Ergebnisberechnung mit einer possibilistischen Auswertestrategie genutzt werden. Es soll also stets die Option mit der besten Wahrscheinlichkeit, dem höchsten Schaden etc. (je nach Parameter) gewählt werden. Bei der Wahl dieser Option wird zusätzlich der berechnete Angriffspfad im jeweiligen Angriffsbaum ausgegeben. Zudem ist die Option auch mit weiteren Auswertestrategien kombinierbar. Diese werden dann nacheinander durchgeführt.
- `-probabilistic`, `-prob`: Für die gewählten Parameter soll die Analyse und Ergebnisberechnung mit einer probabilistischen Auswertestrategie genutzt werden. Es sollen also stets die durchschnittlich zu erwarteten Werte wie zum Beispiel die durchschnittliche Wahrscheinlichkeit genutzt werden. Aufgrund der Berechnung der arithmetischen Mittelwerte ist für diese Option keine Ausgabe des gewählten Angriffspfads im Angriffsbaum möglich. Die Option ist kombinierbar mit weiteren Auswertestrategien. Sind mehrere Strategien gewünscht und als Parameter gewählt, werden diese nacheinander ausgeführt.
- `-costs`, `-c`: Als zusätzlicher Faktor für die Likelihood soll der modellierte Kostenfaktor berücksichtigt werden. Sämtliche Likelihood-Werte ergeben sich mit dieser Option aus dem arithmetischen Mittel aus Likelihood und Costs.
- `-help`, `-h`: Hilfe-Option. Es werden alle verfügbaren Parameter angezeigt, inklusive einer kurzen Erklärung. Eine Analyse und Ergebnisberechnung ist bei Wahl der Hilfe-Option nicht möglich. Für die Nutzung der Hilfe-Option ist die `-model`-Option nicht erforderlich.

5.2. Durchführung einer Analyse

Für die konkrete Durchführung und Auswertung der Analyse wird wieder Bezug zum Beispiel-Angriffsbaum aus Abbildung 3.7 genommen. Die Modellierung enthält die folgenden konkreten Werte:

- Vulnerability Leaf „CR Network::addCharge“:
  - Costs: 3,0
  - Goal: Network::addCharge ausnutzen
  - Exploited Vulnerability: Network::addCharge
- Vulnerability Leaf „CR Network::useCharge“:
  - Costs: 5,0
  - Goal: Network::useCharge ausnutzen
  - Exploited Vulnerability: Network::useCharge
- Vulnerability Leaf „CR WindTurbine::run“:

  - Costs: 2,0
  - Goal: WindTurbine::run ausnutzen
  - Exploited Vulnerability: WindTurbine::run
- Vulnerability Leaf „Photovoltaics::run mit underMaxAddCharge“:
  - Costs: 1,0
  - Goal: Photovoltaics::run ausnutzen mit underMaxAddCharge
  - Exploited Vulnerability: Network::useCharge
- Vulnerability Leaf „Photovoltaics::run mit overMaxAddCharge“:
  - Costs: 4,0
  - Goal: Photovoltaics::run ausnutzen mit overMaxAddCharge
  - Exploited Vulnerability: Network::useCharge
- Coverage Region Vulnerability „WindTurbine::run“:
  - Impact: 5,0
  - Likelihood: 6,0
- Coverage Region Vulnerability „Network::addCharge“:
  - Impact: 4,0
  - Likelihood: 2,0
- Coverage Region Vulnerability „Network::useCharge“:
  - Impact: 6,0
  - Likelihood: 2,0

Diese Werte sind aufgrund der gesamten Konstruktion des Energienetz-Beispiels fiktiv gewählt und nicht gemäß der Berechnungsvorschriften für die einzelnen Parameter bestimmt. Für die folgende, beispielhafte Auswertung spielt dies jedoch keine Rolle.

Um diese Modellierung auswerten zu können, muss zunächst der entsprechende Kommandozeilen-Befehl konstruiert werden. Im vorliegenden Beispiel sollen alle zur Verfügung stehenden Parameter genutzt werden. Zusammengesetzt zu einem vollständigen Befehl ergibt sich Folgendes: `-m /path/to/file/QuAC_Energy_V1.attackmodelling -i -l -o -r -poss -prob -c`

Aufgrund der oben angesprochenen Nutzung der Eclipse-Ausführungsumgebung ist der Anwendungsname `quacattack` nicht erforderlich. Stattdessen kann der Befehl in der Run Configuration als Argument an die bestehenden „Program arguments“ angehängt werden. Die Ausführung des Programms mit diesen Parametern führt zur in Listing 5.1 zu sehenden Ausgabe. Die Ausgabe ist aufgeteilt in zwei Teile. Im oberen Teil befindet sich zunächst das Ergebnis für die probabilistische Auswertestrategie. Diese enthält, wie oben bereits

ausgeführt, keine Pfade des Angriffsbaums, sondern lediglich alle berechneten Werte. Im unteren Teil folgt die Ausgabe für die possibilistische Auswertestrategie. Hier ist jeweils auch der entsprechende Pfad des Baums enthalten, der für die Berechnung des Parameters genutzt wurde und somit der erfolgversprechendste ist.

```
**************************************

Used Evaluation Strategy: Probabilistic

Attacker ID: _kCGhcLMLEe-9sf5E-LZqVw

Attack Tree ID: _kcNDhbMLEe-9sf5E-LZqVw
Attack Tree Goal: Direct Error ausnutzen
Likelihood: 2,88
Probability: 31,94%
Impact: 5,50
Risk: 19,29%
OWASP-Rating: LOW

**************************************

Used Evaluation Strategy: Possibilistic

Attacker ID: _kCGhcLMLEe-9sf5E-LZqVw

Attack Tree ID: _kcNDhbMLEe-9sf5E-LZqVw
Attack Tree Goal: Direct Error ausnutzen
Likelihood: 4,00
Probability: 44,44%
Attack Tree-Path for Likelihood:
ORNodeImpl: Direct Error ausnutzen; ID _kcNDhbMLEe-9sf5E-LZqVw
VulnerabilityLeafImpl: WindTurbine::run ausnutzen; ID _93BRBrMLEe-9sf5E-LZqVw
exploited Vulnerability: WindTurbine::run (ID: _CQoVkbMMEe-9sf5E-LZqVw)

Impact: 6,00
Attack Tree-Path for Impact:
ORNodeImpl: Direct Error ausnutzen; ID _kcNDhbMLEe-9sf5E-LZqVw
ORNodeImpl: WindTurbine.run mit addChargeCondition; ID _zuDc8bMLEe-9sf5E-LZqVw

VulnerabilityLeafImpl: Network::useCharge ausnutzen; ID _GU6DUrMMEe-9sf5E-
    LZqVw
exploited Vulnerability: Network::useCharge (ID: _bZkI4bMMEe-9sf5E-LZqVw)

Risk: 25,93%
```

```
Attack Tree-Path for Risk:
ORNodeImpl: Direct Error ausnutzen; ID _kcNDhbMLEe-9sf5E-LZqVw
ORNodeImpl: WindTurbine.run mit addChargeCondition; ID _zuDc8bMLEe-9sf5E-LZqVw

VulnerabilityLeafImpl: Network::useCharge ausnutzen; ID _GU6DUrMMEe-9sf5E-
    LZqVw
exploited Vulnerability: Network::useCharge (ID: _bZkI4bMMEe-9sf5E-LZqVw)

OWASP-Rating: HIGH
Attack Tree-Path for OWASP Rating:
ORNodeImpl: Direct Error ausnutzen; ID _kcNDhbMLEe-9sf5E-LZqVw
ORNodeImpl: WindTurbine.run mit addChargeCondition; ID _zuDc8bMLEe-9sf5E-LZqVw

VulnerabilityLeafImpl: Network::useCharge ausnutzen; ID _GU6DUrMMEe-9sf5E-
    LZqVw
exploited Vulnerability: Network::useCharge (ID: _bZkI4bMMEe-9sf5E-LZqVw)


**************************************
```

Listing 5.1: Ausgabe des Analyseprogramms für den Energienetz-Angriffsbaum aus Abbildung 3.7.

Um die Korrektheit der Auswertung zu überprüfen, wurden sämtliche hier ausgegebenen Werte manuell überprüft. Um eine Nachvollziehbarkeit der Berechnungen sicherzustellen, werden diese manuellen Überprüfungen im Folgenden dargestellt. Beim Vergleichen der Ergebniswerte der Analyse mit den manuell berechneten Werten kann festgestellt werden, dass diese vollständig übereinstimmen.

Der Einfachheit halber gelten für diesen Abschnitt die folgenden Abkürzungen für die Vulnerability Leafs:

- VL_1 = CR Network::addCharge
- VL_2 = CR Network::useCharge
- VL_3 = CR WindTurbine::run
- VL_4 = Photovoltaics::run mit underMaxAddCharge
- VL_5 = Photovoltaics::run mit overMaxAddCharge

Possibilistische Auswertung

Impact $impact = max(i \in allImpactValues) = i(VL2) = 6.0$

Likelihood Die Berechnung der einzelnen Werte erfolgt mittels der Formel $likelihood = (likelihood + costs)/2$.

- $l(VL_1) = \frac{2+3}{2} = 2,5$
- $l(VL_2) = \frac{2+5}{2} = 3,5$
- $l(VL_3) = \frac{6+2}{2} = 4$
- $l(VL_4) = \frac{2+1}{2} = 1,5$
- $l(VL_5) = \frac{2+4}{2} = 3$
- $likelihood = max(l \in allLikelihoodValues) = l(VL_3) = 4$
- $probability = \frac{likelihood}{\text{MAX\_LIKELIHOOD}} = \frac{4}{9} = 44,44\%$

Risk Die Berechnung der einzelnen Werte erfolgt mittels der Formel $risk = (impact \cdot likelihood)/(\text{MAX\_IMPACT} \cdot \text{MAX\_LIKELIHOOD})$.

- $r(VL_1) = \frac{2,5 \cdot 4}{81} = 12,35\%$
- $r(VL_2) = \frac{3,5 \cdot 6}{81} = 25,93\%$
- $r(VL_3) = \frac{4 \cdot 5}{81} = 24,69\%$
- $r(VL_4) = \frac{1,5 \cdot 6}{81} = 11,11\%$
- $r(VL_5) = \frac{3 \cdot 6}{81} = 22,22\%$
- $risk = max(r \in allRiskValues) = r(VL_2) = 25,93\%$

OWASP Die Bestimmung der einzelnen Ergebnisse erfolgt anhand der OWASP-Tabelle, die in Unterabschnitt 2.4.2 vorgestellt wird.

- $o(VL_1) = l(VL_1) \times i(VL_1) = \text{LOW} \times \text{MEDIUM} = \text{LOW}$
- $o(VL_2) = l(VL_2) \times i(VL_2) = \text{MEDIUM} \times \text{HIGH} = \text{HIGH}$
- $o(VL_3) = l(VL_3) \times i(VL_3) = \text{MEDIUM} \times \text{MEDIUM} = \text{MEDIUM}$
- $o(VL_4) = l(VL_4) \times i(VL_4) = \text{LOW} \times \text{HIGH} = \text{MEDIUM}$
- $o(VL_5) = l(VL_5) \times i(VL_5) = \text{MEDIUM} \times \text{HIGH} = \text{HIGH}$
- $owasp = max(o \in allOwaspValues) = l(VL_2) = \text{HIGH}$

Probabilistische Auswertung Als Vereinfachung wird für diesen Abschnitt der OR-Knoten „WindTurbine::run mit addChargeCondition“ mit „OR_2“ abgekürzt. Der Wurzelknoten würde mit „OR_1“ abgekürzt werden, wird allerdings für die folgenden Ausführungen nicht benötigt.

Impact

- $i(OR_2) = \frac{i(VL_1)+i(VL_2)}{2} = \frac{4+6}{2} = 5$
- $impact = \frac{i(OR_2)+i(VL_3)+i(VL_4)+i(VL_5)}{4} = \frac{5+5+6+6}{4} = 5,5$

Likelihood Die hier eingesetzten Likelihood-Werte entsprechen den im Unterabschnitt zur possibilistischen Auswertung berechneten Werten. Da es sich dabei um die Werte für einzelne Blätter handelt, unterscheiden diese sich nicht zwischen der possibilistischen und der probabilistischen Auswertestrategie.

- $l(OR_2) = \frac{l(VL_1)+l(VL_2)}{2} = \frac{2,5+3,5}{2} = 3$
- $likelihood = \frac{l(OR_2)+l(VL_3)+l(VL_4)+l(VL_5)}{4} = \frac{3+4+1,5+3}{4} = 2,88$
- $probability = \frac{likelihood}{\text{MAX\_LIKELIHOOD}} = \frac{2,875}{9} = 31,94\%$

Risk Auch hier entsprechen die eingesetzten Risk-Werte den für die possibilistische Evaluation berechneten Werten, da es sich hier ebenfalls um Blätter handelt und somit kein Unterschied zwischen den Evaluationsstrategien besteht.

- $r(OR_2) = \frac{r(VL_1)+r(VL_2)}{2} = \frac{12,35\%+25,93\%}{2} = 19,14\%$
- $risk = \frac{r(OR_2)+r(VL_3)+r(VL_4)+r(VL_5)}{4} = \frac{19,14\%+24,69\%+11,11\%+22,22\%}{4} = 19,29\%$

OWASP Für die Berechnung des OWASP-Scores werden in der probabilistischen Auswertung keine separaten Werte berechnet. Stattdessen werden hier die beiden bereits berechneten Werte für Likelihood und Impact verwendet und gemäß der OWASP-Tabelle der Score bestimmt:

$owasp = likelihood \times impact = 2,88 \times 5,5 \equiv \text{LOW} \times \text{MEDIUM} = \text{LOW}$

6. Evaluation

Die Evaluation der vorgestellten Arbeit wird mittels der von Basili und Weiss [2] eingeführten „Goal Question Metric"-Methode (GQM) durchgeführt. Diese Methode steht für ein gestaffeltes Vorgehen: Für verschiedene Evaluationsziele („Goal") werden zunächst konkrete Fragen („Question") definiert, die in der Evaluation beantwortet werden sollen, sowie Metriken für die Beantwortung dieser Fragen („Metric"). Anschließend wird der Aufbau der Fallstudie vorgestellt, mittels derer die Evaluation anhand des definierten GQM-Plans nachfolgend durchgeführt wird. Für die hier vorliegende Arbeit werden die Aspekte Skalierbarkeit, Machbarkeit und Genauigkeit untersucht. In Abschnitt 6.4 werden abschließend mögliche Bedrohungen der Validität der Fallstudie diskutiert.

6.1. Evaluationsziele und Metriken

Die drei Evaluationsziele, die in den folgenden Abschnitten untersucht werden, sind die Machbarkeit, die Genauigkeit und die Skalierbarkeit des Ansatzes.

Machbarkeit Die Machbarkeit stellt Ziel *G1* dar. Zur Untersuchung dieses Ziels dient Frage *Q1*: Können durch den Ansatz Angriffe für alle Fälle berechnet werden, die laut QuAC zu einem Fehler führen? Ist also für alle Fälle, für die QuAC eine Fehlerwahrscheinlichkeit liefert, die Berechnung eines Angriffs möglich? Diese Frage ist relevant, um grundsätzlich zu evaluieren, ob der Ansatz alle Problemklassen, die zu erwarten sind, abdeckt und diese auswerten kann. Als Metrik *M1* für die Beantwortung von Q1 dient die Sensitivität *S*. Diese lässt sich mittels der Formel $S = a_b/(a_b + a_n)$ berechnen. Der Faktor a_b bezeichnet dabei alle Fälle, für die mittels des in dieser Arbeit entwickelten Ansatzes ein Angriff berechnet werden kann, der auf einer zuvor von QuAC berechneten Fehlerwahrscheinlichkeit beruht. a_n bezeichnet alle Fälle, für die von QuAC eine Fehlerwahrscheinlichkeit berechnet wurde, für die aber mit dem neu entwickelten Angreifermodell kein Angriff berechnet werden kann [34].

Genauigkeit Ziel *G2* untersucht die Genauigkeit des Ansatzes. Die Genauigkeit dient als weiteres Maß, um später beurteilen zu können, ob die Ergebnisse der Analyse korrekt sind. Die damit verbundene Fragestellung *Q2* lautet: Entsprechen die Ergebnisse der Analyse den erwarteten Ergebnissen? Für die Beantwortung dieser Frage wird die Präzision $P = a_t/(a_t + a_f)$ als Metrik *M2* verwendet. Der Faktor a_t entspricht hierbei der Anzahl der Angriffe, für die die korrekten Ergebniswerte berechnet wurden, während Faktor a_f die Anzahl an Angriffen darstellt, für die falsche Ergebniswerte berechnet wurden [34].

Skalierbarkeit Mit Ziel *G3* soll die Skalierbarkeit des implementierten Ansatzes untersucht werden. Die Untersuchung dieses Ziels ist insbesondere im Hinblick auf QuAC [25] relevant, welches die Grundlage dieser Arbeit bildet. Für die Berechnung des Angriffssicherheitsmaßes wird vorausgesetzt, dass zuvor bereits eine Modellierung des Softwaresystems mit QuAC erfolgt ist. Die Ergebnisberechnung von QuAC stellt zwar keine zwingende Voraussetzung dar, allerdings sind ohne diese Ergebnisse keine Korrektheitswahrscheinlichkeiten bekannt. Dementsprechend erscheint es sinnvoll, die Auswertung von QuAC vor der Analyse des hier implementierten Ansatzes durchzuführen. Die derzeitige Implementierung von QuAC verfügt jedoch bereits für kleine Modelle eine schlechte, exponentielle Laufzeit. Dementsprechend sollte der hier vorliegende Ansatz mindestens für zu erwartende Modellgrößen gut skalieren, um die Gesamtskalierbarkeit nicht zusätzlich negativ zu beeinflussen. Des Weiteren ist auch unabhängig von QuAC eine sinnvolle Nutzung des Ansatzes nur dann möglich, wenn eine gute Skalierbarkeit gegeben ist.

Folgende beide Fragen sollen in diesem Zusammenhang beantwortet werden: *Q3.1*: Wie verhält sich die Skalierbarkeit bei einer steigenden Anzahl von Knoten? *Q3.2*: Wie verhält sich die Skalierbarkeit bei einer steigenden Anzahl von Angreifern? Die Anzahl an parallelen Knoten sowie die Anzahl an Angreifern stellen die beiden wesentlichen Faktoren dar, um die Skalierbarkeit zu bewerten. Jeder zusätzliche Angreifer sorgt für zusätzlichen Rechenaufwand, da die Analyse auf den Angriffsbäumen für jeden Angreifer separat ausgeführt wird. Die Anzahl paralleler Knoten ist relevant, da für alle Knoten, die auf einer Ebene des Angriffsbaums nebeneinander liegen, alle Parameter einzeln berechnet werden müssen und sich daraus möglicherweise ein Skalierbarkeitsproblem ergeben könnte. Die Anzahl an Schwachstellen spielt hingegen keine maßgebliche Rolle in der Skalierbarkeit, da sie die Größe eines Angriffsbaums nicht direkt beeinflusst. Der einzige Einfluss, den Schwachstellen haben, sind die unterschiedlichen Parameterwerte, die sie liefern. Indirekt kann eine größere Anzahl an vorhandenen Schwachstellen auch zu größeren Angriffsbäumen führen. Da jedoch zu erwarten ist, dass nur wenige Schwachstellen für das Erreichen eines Ziels infrage kommen, sorgt eine höhere Zahl von Schwachstellen tendenziell für eine größere Zahl von Angriffsbäumen beziehungsweise von Angreifern. Die Unterscheidung zwischen mehreren Angriffsbäumen pro Angreifer und mehreren Angreifern führt zum gleichen Ergebnis, da die Analyse sequentiell erfolgt. Dementsprechend ist es ausreichend, die Evaluation auf die Skalierbarkeit bezüglich der Anzahl von Angreifern zu beschränken.

Als Metrik *M3* für die Beurteilung der Skalierbarkeit dient der Vergleich der Ausführungszeiten für verschiedene Skalierungsstufen in Sekunden. Dabei gelten im Rahmen dieser Fallstudie alle Ausführungszeiten von unter fünf Sekunden als sehr gut, unter einer Minute als gut und alle Werte unter zehn Minuten als ausreichend. Diese Grenzwerte wurden gewählt, um eine gute Vergleichbarkeit der Ergebnisse innerhalb dieser Studie zu schaffen. Die Wahl des Grenzwerts von fünf Sekunden orientiert sich an der Aussage von Benyon, wonach „alles, was länger als fünf Sekunden dauert, [...] zu Frustration und Verwirrung [führt]“ [3, S. 34]. Für die Wahl der Grenzwerte von einer Minute und zehn Minuten wurden zum einen eigene Einschätzungen berücksichtigt. Zum anderen wurde auch die Tatsache, dass es sich hierbei um eine Fallstudie sowie beim realen System um eine intensiv vorzubereitende Analyse handelt, berücksichtigt. Weitere Faktoren wie die CPU- oder RAM-Auslastung wurden nicht separat evaluiert. Die Evaluation der CPU-Auslastung erscheint

nicht sinnvoll, da keine parallele Ausführung der Analyse erfolgt, sondern diese sequentiell durchgeführt wird. Dementsprechend sind hier im Vergleich zur Ausführungszeit keine neuen Erkenntnisse zu erwarten. Die RAM-Auslastung ist aufgrund der Ausführung innerhalb von Eclipse nur schwer messbar. Stichprobenhafte Kontrollen der Auslastung während der laufenden Analyse haben keine signifikant erhöhte RAM-Auslastung ergeben, weshalb auf die Berücksichtigung dieses Faktors ebenfalls verzichtet wurde.

6.2. Aufbau der Fallstudie

Für die Evaluation wurde eine Fallstudie entworfen. Diese besteht aus zehn verschiedenen Angriffsbäumen, die in Abbildung 6.1 zu sehen sind. Die ersten beiden Angriffsbäume sind die kleinsten möglichen Varianten und bestehen aus einem Wurzelknoten, einem `AttackStepLeaf` (ASL) und einem `VulnerabilityLeaf` (VL). Beim Wurzelknoten handelt es sich bei Baum 1 um einen OR-Knoten und bei Baum 2 um einen AND-Knoten. Die nachfolgenden Bäume 3 und 4 fassen die eben beschriebenen Bäume 1 und 2 mit einem neuen Wurzelknoten als OR- (Baum 3) beziehungsweise als AND-Knoten (Baum 4) zusammen. Ausgehend von Baum 3 werden die übrigen sechs Angriffsbäume definiert, indem jeweils ein oder zwei der Blattknoten durch die Bäume 1 oder 2 ersetzt werden. Es wird also ein weiterer Teilknoten eingefügt und die Blätter um eine Ebene nach unten verschoben.

Mit dieser Konstruktion der Angriffsbäume ist es möglich, alle Fälle abzudecken, die für die Faktoren Genauigkeit und Machbarkeit relevant sind: eine Kombination aus `AttackStepLeaf` und `VulnerabilityLeaf` mittels AND- und OR-Knoten, eine Aneinanderreihung beider Knotentypen sowie alle sich daraus ergebenden unterschiedlichen Ergebniswerte für die Berechnungen.

Als Grundlage für die Auswertung der zehn gezeigten Angriffsbäume dienen folgende einheitlichen Parameter für alle Blätter des jeweiligen Typs:

- `AttackStepLeaf` (ASL): `impact`: 4.0, `likelihood`: 8.0
- `VulnerabilityLeaf` (VL) beziehungsweise zugehörige `CoverageRegionVulnerability`: `impact`: 7.0, `likelihood`: 3.0

Für alle der in Abbildung 6.1 aufgeführten Angriffsbäume wurden die Ergebniswerte mit dieser Parametrisierung zunächst manuell berechnet. Der Faktor `costs` wurde in den Berechnungen nicht berücksichtigt, da dieser optional ist und in der Analyse lediglich den Likelihood-Wert als Faktor beeinflusst. Der Likelihood-Wert beeinflusst zwar die berechneten Risk-Werte und die OWASP-Bewertung. Diese Einflussnahmen sind aber alle auf die einzelne Berechnung des Likelihood-Wertes zurückzuführen. Es führt also zum gleichen Ergebnis, wenn $(likelihood + costs)/2$ ohne separate Berücksichtigung des `costs`-Faktors als Likelihood-Wert gesetzt wird. Dementsprechend wurde auf eine gesonderte Betrachtung des `costs`-Faktors in der Evaluation verzichtet. In der grundlegenden Überprüfung der Funktionalität des Codes mittels Unit-Tests wurde der Wert jedoch selbstverständlich berücksichtigt.

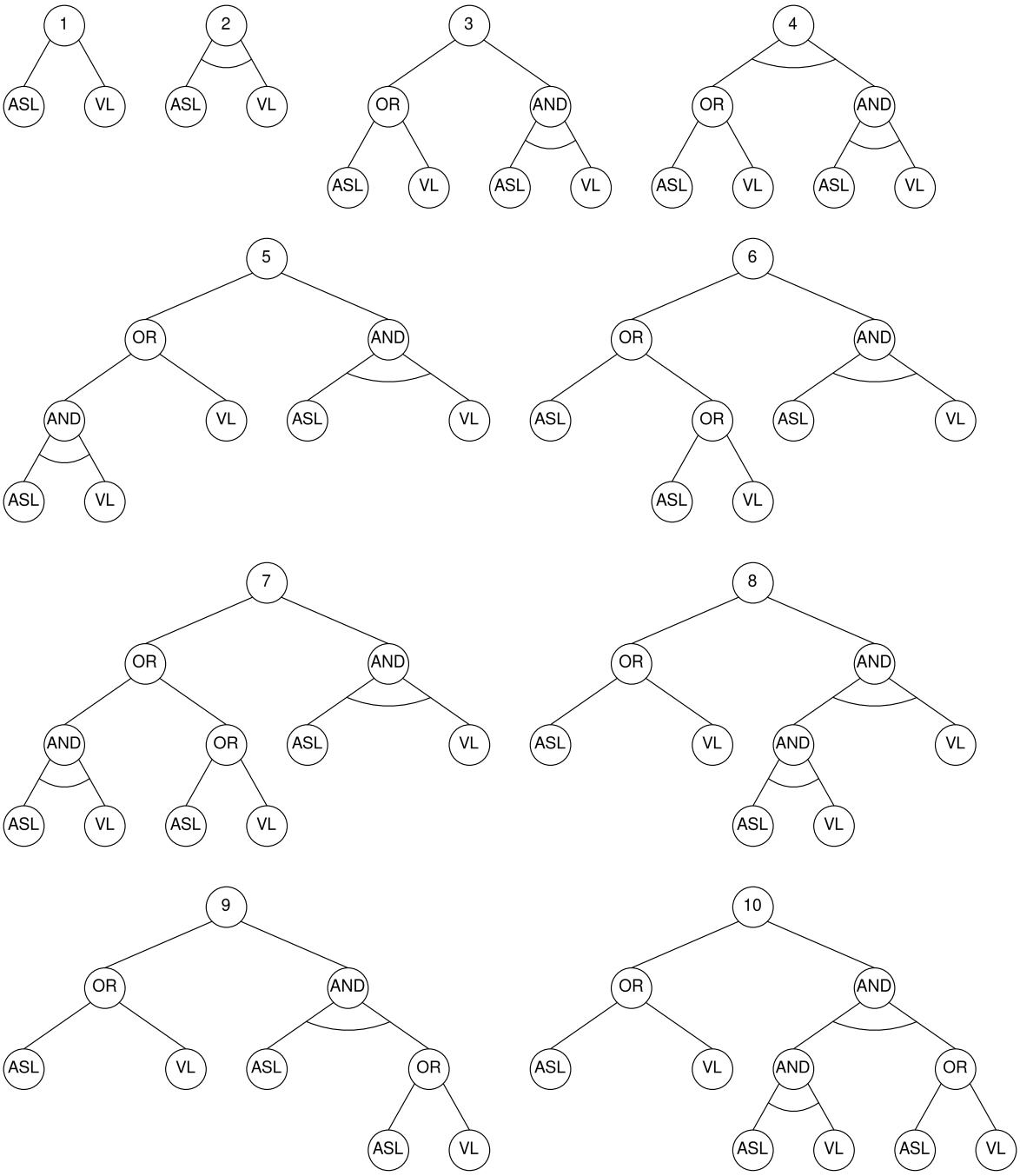

Abbildung 6.1.: Die Angriffsbäume, die für die Fallstudie der Evaluation genutzt werden.

Für die Durchführung der Analyse mittels des implementierten Analysetools wurde folgende Parametrisierung genutzt:

```
-m /path/to/file/QuAC_Energy_EVAL.attackmodelling -i -l -o -r -poss -prob
```

Da die vorgestellten Angriffsbäume für die Evaluation der Skalierbarkeit nicht ausreichend groß sind, wurde für dieses Evaluationsziel Baum 3 als Grundlage verwendet und erweitert:

Der rechte Teilbaum mit dem AND-Knoten verfügt statt nur einem AND-Knoten über insgesamt zehn rekursive AND-Knoten, die sich immer anstelle des ASL befinden. Eine größere Zahl von rekursiven Knoten erscheint unwahrscheinlich. Gleichzeitig stellt diese Zahl im Vergleich zum ursprünglichen Baum aber auch keinen ganz trivialen Fall dar. Der Teilbaum mit den zehn rekursiven AND-Knoten wird für das Evaluationsszenario nun in mehreren Stufen um den Faktor zehn skaliert: aus nur einem AND-Teilbaum parallel zum bestehenden OR-Teilbaum werden in Schritten erst zehn Teilbäume, dann 100, 1000 und 10000. Die Wahl dieser Skalierungsstufen erschien sinnvoll, um eine systematische Steigerung durchführen zu können. Das obere Ende der Skalierungsstufen wurde nach ersten Tests mit entsprechend hohen Skalierungen festgelegt. Gleichzeitig dazu wurde für alle Skalierungsstufen der AND-Teilbäume auch die Anzahl der Angreifer skaliert. Das Basisszenario besteht aus einem Angreifer, der über einen Angriffsbaum verfügt, wobei die Größe des Baums von der Skalierungsstufe der AND-Knoten abhängt. Die Anzahl der Angriffsbäume bleibt aus den im vorherigen Abschnitt genannten Gründen konstant. Die Anzahl der Angreifer wird jedoch ebenfalls skaliert auf zehn, 100 und 1000. Diese Skalierungsstufen orientieren sich an den Skalierungsstufen der Anzahl der parallelen AND-Knoten.

Die Wahl der Parameter für die Blattknoten unterscheidet sich in diesem Evaluationsszenario. Jedoch spielen die Parameterwerte für die Evaluation der Skalierbarkeit keine Rolle, da diese generell nicht von konkreten Variablenbelegungen abhängt. Der auszuführende Befehl für die Analyse bleibt grundsätzlich gleich im Vergleich zum Befehl für die Analyse aller Angriffsbäume aus Abbildung 6.1. Einzig der Pfad zur `.attackmodelling`-Datei muss auf die für dieses Szenario abgestimmte Modellierung angepasst werden. Die Messung der Ausführungszeit beginnt mit dem Aufruf der `start()`-Methode und endet mit dem Abschluss der Ergebnisausgabe in `printResult()`.

Sämtliche Analysen für die Evaluation wurden auf einem ThinkPad T460p mit Ubuntu 22.04 als Betriebssystem sowie einem 2,60 GHz Intel Core i7-6700HQ Prozessor mit 32 GiB Arbeitsspeicher ausgeführt.

6.3. Ergebnisse und Diskussion

Im folgenden Abschnitt werden zunächst die verschiedenen Ergebnisse vorgestellt, bevor diese im zweiten Unterabschnitt diskutiert werden.

Ergebnisse Die Ergebnisse der Evaluation für die Untersuchungsziele G1 und G2 sind in Tabelle 6.1 für den possibilistischen Fall und in Tabelle 6.2 für den probabilistischen Fall zu sehen. Beide Tabellen zeigen für alle Angriffsbäume aus der Fallstudie einen Ergebniswert für die Faktoren Impact, Likelihood, Risk und OWASP. Es sind somit für alle Angriffsbäume alle Werte berechenbar. Für Angriffsbaum 1 ergibt sich beispielsweise im possibilistischen Fall ein Risiko von 39,51 % mit einer OWASP-Bewertung von *HIGH*, wohingegen der gleiche Baum im probabilistischen Fall auf ein Risiko von nur 32,72 % und eine OWASP-Bewertung von *MEDIUM* kommt. Als weiteres Beispiel zeigt Baum 10, dass die Werte für Impact und

Likelihood sehr stark variieren können und dennoch die gleiche OWASP-Bewertung daraus resultiert: im possibilistischen Fall ergibt sich ein Impact von 7,0, eine Likelihood von 8,0 und ein Risiko von 39,51 %. Die OWASP-Bewertung lautet *HIGH*. Für den probabilistischen Fall sind die berechneten Werte niedriger: der Impact liegt bei 6,25, die Likelihood bei 3,56 und das Risiko bei 18,03 %. Die Berechnung der OWASP-Bewertung ergibt jedoch auch in diesem Fall *HIGH*. Alle manuell berechneten und somit zu erwartenden Ergebniswerte entsprechen genau den Ergebniswerten, die in den beiden Tabellen zu sehen sind. Werden mit diesen Ergebnissen nun die Metriken berechnet, folgen daraus folgende Ergebnisse: Die Sensitivität aus M1 ergibt sich aus den Faktoren $a_b = 10$ und $a_n = 0$ zu $S = a_b/(a_b + a_n) = 10/10 = 1$ und beträgt somit 100 %. Die Präzision (M2) berücksichtigt alle korrekten und inkorrekten Berechnungen für die zehn Angriffsbäume. Durch die in allen Fällen erfolgreichen Berechnungen folgt daraus die Parameterbelegung $a_t = 10$ und $a_f = 0$, woraus sich dann das Ergebnis $P = a_t/(a_t + a_f) = 10/10 = 1$ ergibt. Somit beträgt auch die Präzision des Ansatzes 100 %.

Die Ergebnisse der Skalierbarkeitsuntersuchung und somit von Ziel G3 sind im Diagramm in Abbildung 6.2 dargestellt. Das Diagramm zeigt auf der x-Achse die Anzahl der parallelen AND-Knoten und auf der y-Achse die durchschnittliche Ausführungszeit in Sekunden über die pro Angreifer- und AND-Knoten-Kombination jeweils fünf Mal ausgeführten Analysen. Die verschiedenfarbigen Säulen zeigen die Variation der Angreiferzahl zwischen eins und 1000. Für das Basisszenario von einem parallelen AND-Knoten ergibt sich für alle vier Angreiferzahlen eine Ausführungszeit zwischen 1 und 3 Sekunden. Auch beim Szenario mit zehn parallelen AND-Knoten liegt die maximale durchschnittliche Ausführungszeit für die höchste Säule mit 1000 Angreifern nur knapp über 3 Sekunden. Im weiteren Verlauf ist ein exponentieller Anstieg auf unterschiedlichen Niveaus zu beobachten: Für niedrige Angreiferzahlen von 1 und 10 Angreifern bleibt die durchschnittliche Ausführungszeit auch bei bis zu 10000 parallelen AND-Knoten unter 6 Sekunden. Die größeren Angreiferzahlen erreichen im Maximalfall von 10000 parallelen AND-Knoten hingegen eine durchschnittliche Ausführungszeit von circa 25 Sekunden für 100 Angreifer und knapp unter 230 Sekunden für 1000 Angreifer. Bezieht man die vorgestellten Ausführungszeiten auf Metrik M3, lässt sich sagen, dass die Ausführungszeit für alle Analysen mit einem einzigen Angreifer, für bis zu 1000 parallele AND-Knoten mit 10 und 100 Angreifern sowie für bis zu zehn parallele AND-Knoten für 1000 Angreifer im sehr guten Bereich liegt. Bis auf den Wert für 1000 Angreifer und 10000 parallele AND-Knoten, der im ausreichenden Bereich liegt, befinden sich alle übrigen Werte bezüglich der durchschnittlichen Ausführungszeit im guten Bereich.

Die vollständigen Analyseergebnisse inklusive der Rohdaten können in Tabelle A.1 nachgelesen werden.

Diskussion Für das Untersuchungsziel der Machbarkeit (G1) stellt sich die Frage, ob mittels der evaluierten Angriffsbäume ein ausreichender Beweis geliefert ist, dass für alle Fälle Angriffe berechnet werden können, für die die vorherige Ausführung von QuAC zu einer Fehlerwahrscheinlichkeit führt. Hierfür werden die Angriffsbäume aus Abbildung 6.1 nochmals detaillierter betrachtet. Als Blattknoten kommen entweder `AttackStepLeafs` oder `VulnerabilityLeafs` infrage. Beide Typen sind in allen zehn Angriffsbäumen enthalten. Als

Angriffsbaum Nr.	Impact	Likelihood	Risk	OWASP
1	7,00	8,00	39,51 %	HIGH
2	7,00	2,67	10,24 %	MEDIUM
3	7,00	8,00	39,51 %	HIGH
4	7,00	2,37	4,05 %	MEDIUM
5	7,00	3,00	25,93 %	HIGH
6	7,00	8,00	39,51 %	HIGH
7	7,00	8,00	39,51 %	HIGH
8	7,00	8,00	39,51 %	HIGH
9	7,00	8,00	39,51 %	HIGH
10	7,00	8,00	39,51 %	HIGH

Tabelle 6.1.: Ergebnisse der possibilistischen Auswertung der Fallstudie.

Angriffsbaum Nr.	Impact	Likelihood	Risk	OWASP
1	5,50	5,50	32,72 %	MEDIUM
2	7,00	2,67	10,24 %	MEDIUM
3	6,25	4,08	21,48 %	HIGH
4	7,00	1,63	3,35 %	MEDIUM
5	7,00	2,75	14,16 %	MEDIUM
6	5,88	4,71	23,18 %	MEDIUM
7	6,63	3,38	15,86 %	HIGH
8	6,25	3,19	17,69 %	HIGH
9	5,50	5,19	22,82 %	MEDIUM
10	6,25	3,56	18,03 %	HIGH

Tabelle 6.2.: Ergebnisse der probabilistischen Auswertung der Fallstudie.

Knoten kommen `ANDNodes` und `ORNodes` infrage. Auch diese sind in allen größeren Angriffsbäumen beide enthalten, konkret in den Bäumen 3 bis 10. Baum 4 evaluiert zusätzlich die Hintereinanderreihung von zwei `ANDNodes` und gleichzeitig den Fall, dass der Wurzelknoten ein `ANDNode` ist. Die folgenden, von den ersten vier „Basisbäumen“ abgeleiteten Bäume re-

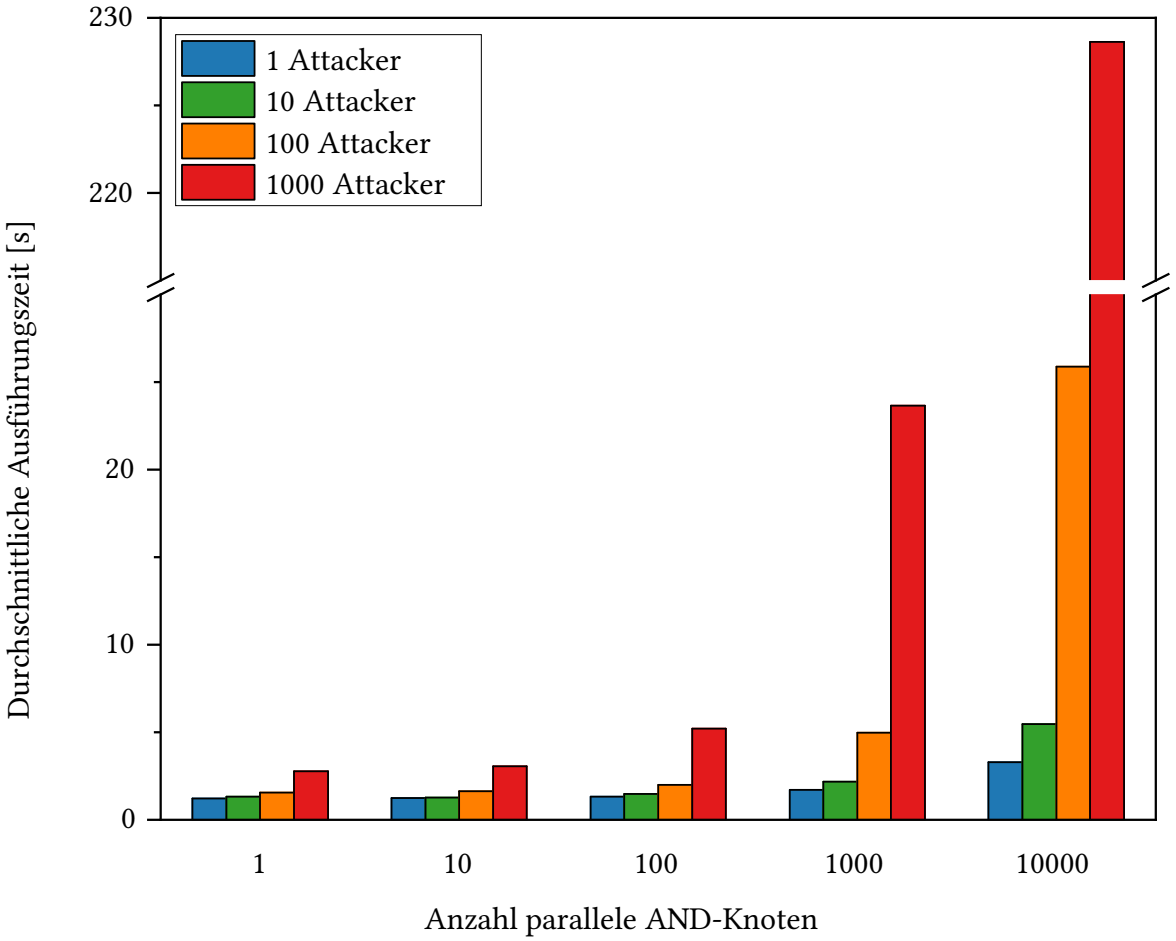

Abbildung 6.2.: Durchschnittliche Ausführungszeiten der Skalierbarkeitsanalyse.

präsentieren alle möglichen Kombinationen von AND- und OR-Knoten als Kindknoten der ersten AND- und OR-Knoten. Alle weiteren möglichen Erweiterungen dieser Bäume stellen eine Kombination oder Erweiterung der hier dargestellten Fälle dar. Daraus kann gefolgert werden, dass alle beliebigen Kombinationen von `ANDNodes`, `ORNodes`, `AttackStepLeafs` und `VulnerabilityLeafs` mittels dieser zehn Angriffsbäume abgedeckt sind. Es war nicht möglich, einen Fall zu konstruieren, der mittels dieser Definition eines Angriffsbaums nicht umsetzbar war. Werden nun die Ergebnisse von QuAC betrachtet, so resultieren daraus am Ende, wie bereits in Abschnitt 2.3 vorgestellt, grundsätzlich zwei Typen von Wahrscheinlichkeiten: direkte Fehlerwahrscheinlichkeiten und Korrektheitswahrscheinlichkeiten. Q1 bezieht sich explizit auf Fälle, die zu einem Fehler führen. Dementsprechend sind hier die Fehlerwahrscheinlichkeiten relevant. Bei der Betrachtung dieser kann festgestellt werden, dass sie sich alle auf eine der modellierten `DirectErrorConditionSEFFExtensions` beziehen. Dies wiederum sind genau die Objekte, die von einer `CoverageRegionVulnerability` referenziert werden können. Somit lässt sich abschließend die in Q1 gestellte Frage, ob Angriffe für alle Fälle berechnet werden können, die laut QuAC zu einem Fehler führen, mit ja beantworten. Aus der Kombination des Aufbaus der Angriffsbäume und der Definition

einer `CoverageRegionVulnerability` folgt, dass die Berechnungsgrundlage der Metrik M1 korrekt ist und die Sensitivität tatsächlich bei 100 % liegt.

Die Berechnung der Präzision, also die Evaluation von Ziel G2, liefert ein Ergebnis von 100 %. Hier kann hinterfragt werden, warum die Präzision nur für diese zehn beziehungsweise in Kombination beider Tabellen 20 Ergebnisse berechnet wurde. Wie bereits im vorherigen Absatz dargelegt, bilden die ausgewählten zehn Angriffsbäume die Grundlage, um alle denkbaren Variationen von Angriffsbäumen zu konstruieren. Die Fallstudie wurde deshalb auf diese zehn Bäume beschränkt. Beim Studienentwurf wurde darauf geachtet, für die Parameter `impact` und `likelihood` Werte zu definieren, die in verschiedene OWASP-Einstufungen fallen. Die Ergebnisse decken die OWASP-Einstufungen *NOTE*, *LOW* und *CRITICAL* nicht ab. Gleichzeitig ist aus der zugrunde liegenden Bewertungstabelle aus Tabelle 2.2 ersichtlich, dass die beiden fehlenden, niedrigen OWASP-Einstufungen voraussetzen, dass mindestens einer der beiden Einflussfaktoren ebenfalls in die Stufe *LOW* fällt. Dies ist mit den gewählten Zahlen nicht der Fall, weshalb diese Bewertungen nicht zustande kommen können. Indirekt wird somit die Korrektheit des Ergebnisses zusätzlich bestätigt. Gleiches gilt auch für die Bewertungsstufe *CRITICAL*, wofür beide Parameter in die Kategorie *HIGH* hätten fallen müssen.

Die im Ergebnisabschnitt angesprochenen Werte für Angriffsbaum 10 müssen nochmals genauer betrachtet werden. Im possibilistischen Fall ergibt sich ein Impact von 7,0 und eine Likelihood von 8,0. Gemäß der OWASP-Bewertungstabelle in Tabelle 2.2 müssten diese beiden Werte zu einer OWASP-Einstufung von *CRITICAL* führen. Dies ist hier jedoch nicht der Fall, stattdessen bleibt die Einstufung bei *HIGH*. Die Ursache hierfür liegt darin, dass für den possibilistischen Fall nicht einfach die Werte der beiden Faktoren berücksichtigt werden dürfen, sondern der Pfad betrachtet werden muss, der für die Kombination aus beiden Faktoren, also Impact und Likelihood, das höchste Risiko und damit die höchste OWASP-Einstufung erreicht. Am Ende dieses Pfades muss ein einziger Blattknoten stehen, während sich die beiden errechneten Blattknoten für Impact und Likelihood voneinander unterscheiden dürfen. Diese Berechnung bestätigt also, dass die OWASP-Berechnung korrekt erfolgt und keine reine Berechnung anhand der beiden berechneten Werte für Impact und Likelihood stattfindet.

Aus der Betrachtung der Ergebnisse für Evaluationsziel G3, der Skalierbarkeit, folgt für zehn Angreifer mit bis zu 1000 parallelen AND-Knoten und für 100 Angreifer mit bis zu 100 parallelen AND-Knoten eine sehr gute Skalierbarkeit. Für größere Zahlen von parallelen AND-Knoten und/oder Angreifern erscheint es unrealistisch, dass Entwickler solche Angriffe nicht in mehrere Teile aufteilen, die dann wiederum im Bereich einer sehr guten Skalierbarkeit liegen. Aufgrund der zugehörigen Entwicklungsumgebung und der dortigen Rahmenbedingungen der Darstellung von Angriffsbäumen ist davon auszugehen, dass zu große Bäume hier sehr unübersichtlich werden. Gleiches gilt für eine zu große Anzahl von Angreifern. Nutzt ein möglicher Entwickler stattdessen die Option, verschiedene Angriffsbäume in mehreren verschiedenen Dateien zu modellieren, verringert sich zusätzlich die Gefahr, dass die Modellierung sehr groß wird. Somit ist auch hier wieder zu erwarten, dass die Skalierbarkeit im sehr guten Bereich liegt.

Grundsätzlich wurde in der Ergebnisdarstellung der arithmetische Mittelwert über die jeweils berechneten fünf Werte gebildet. Der Mittelwert wurde dem Median vorgezogen, da die Varianz für die sehr guten Werte im Bereich von unter einer Sekunde liegt und der Ergebniswert durch die Berechnung des Mittelwerts nicht erheblich in eine Richtung verfälscht wird. Auch bei den in der Fallstudie hohen Ausführungszeiten von über 3 Minuten liegt die Varianz mit 6,5 Sekunden in einem im Verhältnis zur Ausführungszeit akzeptablen Bereich. Da hier zusätzlich zwei Werte an der unteren Grenze und zwei Werte an der oberen Grenze, insgesamt also vier der fünf Messwerte die jeweiligen Extreme darstellen, erscheint insbesondere an dieser Stelle der Mittelwert sogar als das geeignetere Maß für die Darstellung des Ergebnisses.

Zusätzlich zur Ausführungszeit hinzu kommen einige Sekunden, die die Analyseumgebung für den Start benötigt. Da diese jedoch nur zu Beginn und nicht während der Analyse erforderlich ist und sehr stark von der konkreten Hardware und den freien Ressourcen abhängt, wurde diese Startzeit von einigen Sekunden für die Analyse nicht berücksichtigt.

Darüber hinaus muss beachtet werden, dass, sofern neben der Angriffssicherheit auch Aussagen über die Korrektheits- und Fehlerwahrscheinlichkeiten gewünscht sind, zusätzlich zur Auswertung des hier vorliegenden Ansatzes eine Ausführung der Analyse von QuAC erforderlich ist. Die derzeitige Implementierung von QuAC weist erhebliche Probleme bezüglich der Skalierbarkeit auf, welche in diesem Zusammenhang nicht vernachlässigt werden sollten [25].

6.4. Bedrohungen der Validität

In diesem Abschnitt wird die Validität der Evaluation anhand von vier verschiedenen Kriterien untersucht. Diese vier Kriterien, die speziell auf die Evaluation der Validität von Fallstudien ausgerichtet sind, wurden von Runeson und Höst [36] definiert.

Konstruktvalidität Die Konstruktvalidität überprüft, ob die für die Evaluation verwendeten Metriken geeignet sind, um die zur jeweiligen Metrik gehörenden Fragen zu beantworten. Die Sensitivität wurde für die Evaluation der Machbarkeit (G1) als Metrik genutzt. Da die Sensitivität überprüft, ob alle relevanten Angriffe auch berechenbar sind und hierfür ein Ergebnismaß liefert, erscheint die Metrik zur Beantwortung von Frage Q1 als angemessen.

Q2 stellt die Frage nach der Genauigkeit des Ergebnisses, also, ob die berechneten Werte korrekt sind und somit den erwarteten, manuell geprüften Ergebnissen entsprechen. Die Präzision, die hierfür als Metrik genutzt wird, berechnet genau den prozentualen Anteil der korrekten Berechnungen von der Gesamtmenge aller Berechnungen. Dementsprechend ist die Präzision als Metrik für G2 passend und es kann keine Bedrohung der Validität gesehen werden.

Die Ausführungszeit als Metrik für die Untersuchung der Skalierbarkeit (G3) erscheint ebenfalls als geeignet, da zu erwarten ist, dass sich eine schlechte Skalierbarkeit in einer höheren Ausführungszeit bemerkbar machen würde. In diesem Fall wären deutlich mehr

Berechnungen oder mehr zeitintensive Berechnungen erforderlich, die dazu führen, dass die Ausführungszeit ansteigt. Andere Faktoren wie beispielsweise die Prozessor- oder Arbeitsspeicherauslastung können jedoch ebenso eine erhebliche Rolle in der Skalierbarkeit spielen. Eine zu hohe Arbeitsspeicherauslastung könnte im schlimmsten Fall zum Absturz des Prozesses beziehungsweise des ganzen Systems führen und somit implizit auch dazu, dass die Analyse nicht zu Ende geführt werden kann. Da diese beiden Faktoren nicht Teil der Metrik sind, kann hierin eine Gefährdung der Validität gesehen werden. Um diese Gefährdung zu minimieren, wurde bereits im Abschnitt zur Definition der Metrik M3 dargelegt, weshalb diese zusätzlichen Faktoren zu vernachlässigen sind. Die stichprobenhaften Kontrollen der Arbeitsspeicher- und Prozessorauslastung erfüllen nicht die Anforderungen an eine vollständige Evaluation. Jedoch dienen sie dazu, das Risiko einzuschätzen, das aus der Nichtbeachtung der beiden Faktoren in der Metrik entsteht. Da dieses Risiko auf Grundlage der Beobachtung als gering einzuschätzen ist, ist auch die Gefährdung der Validität durch die fehlenden zusätzlichen Faktoren für die Beurteilung der Skalierbarkeit als gering einzuschätzen.

Interne Validität Die interne Validität soll sicherstellen, dass die Ergebnisse der Evaluation ausschließlich von Faktoren beeinflusst werden, die klar definiert sind und deren Einfluss zu erwarten ist. Für die Evaluation der Skalierbarkeit ist ein gewisses Risiko nicht auszuschließen, dass die konkret genutzte Hardware die gemessenen Ergebnisse erheblich beeinflusst. Da dieser Aspekt jedoch im Versuchsaufbau durch die Angabe der genutzten Hardware berücksichtigt wird, besteht hier keine konkrete Gefährdung der internen Validität.

Für die Evaluation der Genauigkeit wurden sämtliche Ergebnisse manuell berechnet und anschließend mit der Ausgabe der Analyse verglichen. Durch diese doppelte Berechnung ist für die untersuchten Angriffsbäume sichergestellt, dass die Ergebnisse von keinen weiteren, nicht erwarteten Faktoren beeinflusst werden. In der Ergebnisdiskussion wurde dargelegt, warum diese Ergebnisse verallgemeinert werden können. Dennoch bleibt hier ein Restrisiko bestehen, dass Fälle auftreten könnten, die nicht evaluiert wurden und für die die interne Validität gefährdet ist. Eine weitere Gefährdung der internen Validität besteht in diesem Zusammenhang darin, dass die manuelle Berechnung der Ergebnisse Kenntnisse über die konkrete Implementierung beziehungsweise über die Funktionsweise der einzelnen Berechnungsalgorithmen voraussetzt. Da diese Algorithmen jedoch ausführlich in Unterabschnitt 5.1.3 vorgestellt werden, sind die erforderlichen Berechnungsschritte nachvollziehbar und die Gefährdung der Validität durch diesen Faktor kann minimiert werden.

Externe Validität Im Rahmen der Betrachtung der externen Validität soll geprüft werden, ob Ergebnisse der durchgeführten Fallstudie, die möglicherweise verallgemeinert wurden, auch tatsächlich verallgemeinerbar sind. Gemäß Runeson und Höst [36] stellen Fallstudien grundsätzlich nur spezifische Aspekte und Stichproben dar. Da aus den Ergebnissen der Fallstudie jedoch allgemeine Aussagen bezüglich der drei Untersuchungsziele getroffen werden, liegt hier grundsätzlich eine Bedrohung der Validität vor. Aus einer genaueren Betrachtung des Studienentwurfs folgt, dass die gewählten Angriffsbäume in der Summe eine repräsentative Stichprobe ergeben. Dies wurde bereits in den beiden vorhergehenden Unterkapiteln zum Studienaufbau (siehe Abschnitt 6.2) und der Ergebnisdiskussion (siehe

Abschnitt 6.3) hinreichend dargelegt. Damit kann die Gefährdung der externen Validität für alle Untersuchungsziele, insbesondere aber für G1 und G2, reduziert werden. Die Bedrohung der externen Validität im Hinblick auf Untersuchungsziel G3 kann ebenfalls als minimal angesehen werden, da derart große Angriffsbäume und Angreiferzahlen, wie sie für die Evaluationsstudie verwendet wurden, in der Realität äußerst unwahrscheinlich sind. Dies wurde ebenfalls bereits in der Diskussion genauer dargelegt. Die aus den Ergebnissen gezogene Verallgemeinerung, dass für realistische Angreifer- und Angriffsbaum-Größen eine gute bis sehr gute Skalierbarkeit des Ansatzes besteht, ist damit als korrekt anzusehen und stellt keine Bedrohung der Validität dar.

Verlässlichkeit Das Kriterium der Verlässlichkeit prüft, ob die Analyse durch andere Forscher reproduzierbar ist. Da der Quellcode für die Analyse durch andere Forscher nicht verändert werden muss und die gesamte Konstruktion der Angriffsbäume inklusive der Parametrisierung in den beiden Abschnitten zu den Evaluationsszenarien beschrieben ist, ist keine Gefährdung der Verlässlichkeit zu erwarten. Die Skalierbarkeits-Ergebnisse könnten je nach verwendeter Hardware gegebenenfalls etwas von den in dieser Arbeit präsentierten Ergebnissen abweichen. Um hier eine Vergleichbarkeit herzustellen, wurden die konkreten Hardware-Daten im Studienaufbau bereits aufgeführt. Des Weiteren sollten sich leicht veränderte Skalierbarkeits-Eigenschaften nicht wesentlich auf das aus dem Ergebnis gezogene Fazit auswirken, da insgesamt nur kleinere Abweichungen zu erwarten sind.

7. Verwandte Arbeiten

Im Folgenden werden verwandte Arbeiten, sortiert nach den drei Themenbereichen Angriffsbäume und Angriffsgraphen, Schwachstellenmodellierung sowie konkrete Parameterberechnung, vorgestellt. Einige der genannten Arbeiten stellen gleichzeitig eine Erweiterungsoption dieser Masterarbeit dar.

Angriffsbäume und Angriffsgraphen Wie in Abschnitt 2.1 vorgestellt, existieren hauptsächlich zwei verbreitete Möglichkeiten zur Modellierung von Angriffen: Angriffsbäume und Angriffsgraphen. Johnson u. a. [15] nutzen in ihrer „Meta Attack Language“ beispielsweise Angriffsgraphen zur Modellierung von Angriffen. Gleiches gilt auch für die Ansätze von Polatidis u. a. ([32], [33]) und Walter u. a. [44]. Dem gegenüber stehen Arbeiten wie die von Kumar u. a. [24], Naik u. a. [29] sowie Kumar und Stoelinga [23], die allesamt Angriffsbäume nutzen. Für den Rahmen dieser Masterarbeit werden die letztgenannten Angriffsbäume verwendet. Insbesondere der Ansatz von Naik u. a. [29] ist sehr ähnlich zum hier vorliegenden Ansatz. Jedoch ergeben sich speziell bei der Grundlage, auf der die Angriffe berechnet werden, Unterschiede. Bei Naik u. a. bildet die Organisationsebene eines Unternehmens diese Grundlage. Hier sind beispielsweise einzelne Mitarbeiter oder Besucher genauso als Schwachstelle denkbar wie Fehler in IT-Anwendungen [29]. Hingegen basiert diese Masterarbeit auf konkret gefundenen Fehlern und Schwachstellen in der Quellcode-Ebene.

Im Ansatz von Johnson u. a. [15] wird lediglich die oben genannte „Meta Attack Language“ definiert. Die Arbeit von Polatidis u. a. [32] generiert Angriffspfade innerhalb von Angriffsgraphen basierend auf festgelegten Regeln. Der Fokus ihrer Arbeit liegt auf dem Finden dieser Angriffspfade und nicht auf einer weiteren Auswertung. Auch Walter u. a. [44] präsentieren die Generierung von Angriffsgraphen sowie eine Filterung dieser Graphen. In allen drei Arbeiten findet jedoch im Gegensatz zu dieser Arbeit keine weitergehende Analyse und Bestimmung von konkreten Parametern wie der Angriffswahrscheinlichkeit oder dem Angriffsrisiko statt.

In [33] sagen Polatidis u. a. Angriffsschritte mittels eines Angriffsgraphen vorher und nutzen hierfür das CVE- und CWE-System. Sie orientieren sich im Gegensatz zu dieser Arbeit nicht an konkreten, in der Implementierung gefundenen Fehlern oder Schwachstellen.

Die Arbeit von Kumar und Stoelinga [23] basiert auf einer stochastischen Analyse und arbeitet nicht auf Quellcode-Ebene. In [24] präsentieren Kumar u. a. ein Tool, das zuvor modellierte Angriffsbäume nach einer Transformation in zeitgesteuerte Automaten mittels eines Model Checkers analysiert und die Ergebnisse anschließend in den ursprünglich modellierten Angriffsbaum zurück transformiert. Der Unterschied zu dieser Arbeit liegt darin,

dass Kumar u. a. kein explizites Angriffssicherheitsmaß berechnen, sondern lediglich die Option der Berechnung verschiedener einzelner Eigenschaften wie beispielsweise der Kosten eines Angriffs bieten. Außerdem besteht kein direkter Bezug zu konkreten Schwachstellen, sondern es handelt sich um einen generischen Ansatz, der Erweiterungsmöglichkeiten in viele verschiedene Richtungen bietet [24].

Angreiferfähigkeiten Die Fähigkeiten, über die Angreifer verfügen, sind ebenfalls schon Teil von bestehenden Arbeiten. Polatidis u. a. [32] generieren Angriffspfade gemäß festgelegter Regeln, zu denen unter anderem Angreiferfähigkeiten oder auch der Standort des Angreifers zählen. Im Ansatz von Walter u. a. [43] basieren die berücksichtigten Angreiferfähigkeiten auf modellierten Schwachstellen und Verwundbarkeiten, welche sich wiederum auf bereits bekannte Schwachstellen aus den CVE-/CWE-Systemen beziehen. Eine solche Erweiterung um Angreiferfähigkeiten bringt eine erhebliche Variabilität in die Angriffsmöglichkeiten. Gegebenenfalls ist ein Angreifer dadurch nicht mehr in der Lage, alle Angriffe durchzuführen, könnte aber beispielsweise von anderen Angreifern mit ihren Fähigkeiten unterstützt werden. Eine direkte Implementierung von Angreiferfähigkeiten war im Rahmen dieser Masterarbeit aus Zeitgründen nicht möglich. Jedoch sind insbesondere für die Methode von Walter u. a. die Voraussetzungen für eine Erweiterung gegeben, da sich die Schwachstellen-Modellierung, wie auch in Abschnitt 4.1 bereits ausgeführt, an dieser Arbeit orientiert.

Schwachstellen-Modellierung Wie bereits im vorherigen Abschnitt angesprochen, modellieren Walter u. a. [43] in ihrem Ansatz Schwachstellen und Sicherheitslücken mittels der CVE-/CWE-Systeme. Auch Deloglos u. a. [8] nutzen unter anderem CVEs für die in ihrem Ansatz enthaltene Datenbank, die die möglichen Angriffsaktionen enthält. Polatidis u. a. [33] verwenden ebenfalls die CVE-/CWE-Systeme für ihre Angriffsoptionen. Die Modellierung von CVEs und CWEs ist als Erweiterung dieses Ansatzes im Metamodell ebenfalls vorgesehen, allerdings noch ohne konkreten Einfluss auf die Analyse. Vielmehr werden derzeit Schwachstellen betrachtet, die sich auf konkrete Fehler in der Implementierung beziehen, die im Rahmen der formalen Verifikation mittels QuAC in Form von Fehlerwahrscheinlichkeiten gefunden wurden. Es handelt sich also um eine andere Ebene, auf der die Schwachstellen betrachtet werden. CVEs und CWEs lassen sich zwar auch auf Implementierungsfehler zurückführen, setzen aber an einem anderen Punkt an: bei bekannten Schwachstellen und Verwundbarkeiten.

In einem auf der von Johnson u. a. [15] eingeführten „Meta Attack Language“ basierenden Ansatz modellieren Katsikeas u. a. [16] IT-Infrastrukturen. Sie bieten dabei die Möglichkeit, sowohl bekannte als auch unbekannte Schwachstellen zu modellieren. Allerdings betrachten sie IT-Infrastrukturen, wohingegen sich diese Arbeit mit komponentenbasierten Softwaresystemen auf Implementierungsebene beschäftigt.

Konkrete Parameterberechnung Es existieren viele verschiedene Ansätze, die sich mit der Berechnung von Angriffswahrscheinlichkeiten oder Angriffsrisiken beschäftigen. Sommestad u. a. [40] bestimmen die Wahrscheinlichkeit, dass ein Angriff innerhalb einer Woche durch Penetrationstester erfolgreich ist. Es handelt sich dabei jedoch zum einen um ein

Enterprise-Netzwerk, zum anderen kann nur eine Auswahl von bereits bekannten Sicherheitslücken berücksichtigt werden. Kumar u. a. [24] berücksichtigen in ihrer automatisierten Analyse von Angriffsbäumen Faktoren wie beispielsweise die Erreichbarkeit oder die Wahrscheinlichkeit eines erfolgreichen Angriffs. Canbolat u. a. [5] und Bernsmed u. a. [4] verwenden Bewertungsskalen zur Bestimmung der Werte für Likelihood und Impact. Diese Skalen sind ähnlich zur hier verwendeten OWASP-Skala. Sie bestehen aus mehreren Stufen und enthalten für jede Stufe eine passende Beschreibung. Auch Naik u. a. [29] berechnen diese Parameter, allerdings nicht anhand einer einfachen Zuordnungsskala, sondern mittels verschiedener Formeln, die unter anderem die Kosten des Angriffs, die Angriffshäufigkeit und die technische Schwere der Angriffsausführung berücksichtigen.

All diese Berechnungsansätze beziehen sich jedoch auf die Architekturebene eines Softwaresystems, Naik u. a. gar noch eine Ebene höher auf die Organisationsebene eines Unternehmens. Im Gegensatz dazu bezieht sich der Ansatz dieser Masterarbeit auf die Quellcodeebene und analysiert auf dieser mögliche Angriffe auf bereits modellierte, komponentenbasierte Softwaresysteme.

Insgesamt lässt sich sagen, dass viele zu dieser Arbeit verwandte Arbeiten existieren, die Angriffsbäume verwenden und/oder eine konkrete Parameterberechnung durchführen. Es konnte jedoch keine Arbeit gefunden werden, die ein Angriffssicherheitsmaß auf Grundlage von Angriffsbäumen berechnet und dabei auf der Ebene des Quellcodes arbeitet. Diese Arbeit hebt sich insofern von den anderen Arbeiten ab und schließt die Lücke der fehlenden Angriffssicherheits-Analyse mittels Angriffsbäumen auf Grundlage einer formalen Verifikation des Quellcodes.

8. Fazit und Ausblick

In diesem Kapitel werden zunächst die Ergebnisse und der Beitrag der Masterarbeit zusammengefasst, bevor in Abschnitt 8.2 auf die Beschränkungen des Ansatzes eingegangen wird. Außerdem wird ein Ausblick auf mögliche zukünftige Arbeiten in diesem Themenbereich gegeben.

8.1. Fazit

In der vorliegenden Masterarbeit wurde der existierende QuAC-Ansatz um die Möglichkeit erweitert, Angreifer zu modellieren, die gezielt fehlerhafte Eingaben machen sowie gezielt Randfälle und Implementierungsfehler ausnutzen. Mittels eines neu entwickelten Metamodells ist eine kombinierte Modellierung von Angreifern, Angriffsbäumen und Schwachstellen im Kontext des QuAC-Ansatzes möglich. Die modellierten Schwachstellen ermöglichen dabei den direkten Bezug zu den von QuAC berechneten Fällen, für die die Korrektheit der Ausführung nicht formal verifiziert werden konnte. Dank einer individuellen Annotation der Blätter des Angriffsbaums sowie der Schwachstellen ermöglicht eine ebenfalls in dieser Arbeit entwickelte Analyse, die Modellierung auszuwerten und die Parameter Likelihood, Impact und Risk zu berechnen. Likelihood bezeichnet dabei die Wahrscheinlichkeit, dass eine Schwachstelle von einem Angreifer gefunden und ausgenutzt wird. Die Auswirkungen eines erfolgreichen Angriffs werden vom Impact dargestellt, während Risk für das Risiko steht, das aus dem Angriff hervorgeht. Außerdem wird die Einordnung des Angriffs im OWASP-Bewertungssystem bestimmt. Mittels dieser Parameter kann eine Aussage über die Angriffssicherheit des zugrundeliegenden komponentenbasierten Softwaresystems getroffen werden. Beispielsweise könnte am Ende ein Angriff als „kritisch“ für die Angriffssicherheit eingestuft werden.

In der Evaluation in Kapitel 6 wurden mittels einer Fallstudie, bestehend aus zehn verschiedenen Angriffsbäumen, die Faktoren Machbarkeit, Genauigkeit und Skalierbarkeit analysiert. Die Ergebnisse der Evaluation lieferten im Rahmen der Machbarkeitsuntersuchung eine Sensitivität von 100 % und für die Genauigkeit eine Präzision von 100 %. Die Skalierbarkeit wurde für bis zu 100000 parallele Knoten und 1000 Angreifer untersucht. Bis zur Zahl von 100 parallelen Knoten und 100 Angreifern lag die Ausführungszeit dabei bei unter 5 Sekunden. Auch für die maximalen Zahlen wurde eine Ausführungszeit von unter 4 Minuten erreicht. Somit konnte eine sehr gute Skalierbarkeit nachgewiesen werden.

Mittels der genannten Parameter Likelihood, Impact und Risk sowie der OWASP-Einstufung ist eine quantitative Bewertung eines komponentenbasierten Softwaresystems hinsichtlich

seiner Angriffssicherheit möglich. Die Evaluation ergab durchweg positive Ergebnisse in Form der Sensitivität, Präzision und Skalierbarkeit. Somit kann auch die eingangs gestellte Forschungsfrage, ob es möglich ist, ein komponentenbasiertes Softwaresystem hinsichtlich seiner Angriffssicherheit auf Grundlage einer formalen Analyse des Quellcodes quantitativ zu bewerten, positiv beantwortet werden.

Betrachtet man den aktuellen Forschungsstand zu diesem Thema, der in Kapitel 7 vorgestellt wurde, so lassen sich verschiedene Forschungsansätze finden, die sich mit Angriffsbäumen und Angreifermodellen beschäftigen. Auch zur Einschätzung des Sicherheitsrisikos, das durch Schwachstellen und Sicherheitslücken entsteht, existieren bereits verschiedene Ansätze. Der Beitrag dieser Masterarbeit zur aktuellen Forschung besteht jedoch darin, dass die Angriffssicherheits-Einschätzung auf dem Quellcode der konkreten Softwarearchitektur aufbaut und nicht, wie in vielen anderen Arbeiten, auf der reinen Architekturebene.

8.2. Beschränkungen und Ausblick

Im Folgenden werden zunächst die festgestellten Beschränkungen dargelegt, bevor der Ausblick auf mögliche zukünftige Forschung im Themenbereich dieser Arbeit erfolgt.

Beschränkungen Im Verlauf der Evaluation hat sich gezeigt, dass die probabilistische Berechnung eine wesentliche Einschränkung besitzt: Die Berechnung der Ergebnisse berücksichtigt nicht jedes Blatt zum gleichen Anteil. Je tiefer ein Blatt im Angriffsbaum liegt, das heißt, je mehr Knoten vom Wurzelknoten bis zum Blatt passiert werden müssen, desto geringer wird der Einfluss des einzelnen Blattes. Für den Fall, dass alle Blätter die exakt identische Anzahl an Elternknoten besitzen, spielt diese Beschränkung keine Rolle. Existieren aber zwei Pfade, von denen der eine sehr viele Verzweigungen besitzt und der andere direkt in einem Blatt endet, sorgt die Implementierung für den probabilistischen Fall dafür, dass die Werte des Blattes ohne viele Elternknoten ein viel höheres Gewicht im Ergebnis haben als die Werte der anderen, tiefer verschachtelten Blätter. Begründet liegt diese Einschränkung darin, dass der Algorithmus rekursiv für jeden Knoten die einzelnen Parameter berechnet und aus den Werten aller Kindknoten den arithmetischen Mittelwert bildet, unabhängig davon, ob es sich um den Impact-, Likelihood- oder Risk-Wert handelt. Er beachtet also nicht, wie viele Blätter sich unter einem Kindknoten verbergen. Diese Beschränkung kann derzeit nicht beeinflusst werden. Möglich wäre die Implementierung von anderen Auswertestrategien oder die Anpassung der existierenden Auswertestrategie, sodass die Tiefe eines Blatts innerhalb des Angriffsbaums bei der Berechnung berücksichtigt wird.

Auch für die OWASP-Berechnung ist diese Einschränkung implizit gegeben, da für den probabilistischen Fall für die Bestimmung der OWASP-Einstufung lediglich die resultierenden Impact- und Likelihood-Werte für den Wurzelknoten und somit für den gesamten Angriffsbaum betrachtet werden. Die possibilistischen Berechnungen sind von dieser Einschränkung nicht betroffen, da sich in diesem Fall stets das Blatt mit den besten Parameterwerten durchsetzt und keine Berechnung eines Durchschnitts erforderlich ist.

Eine weitere Beschränkung des Ansatzes liegt darin, dass aktuell nur Schwachstellen darstellbar sind, die auf einer Coverage Region aufbauen. Andere Schwachstellen, die beispielsweise eine bereits allgemein bekannte, nicht direkt auf das Softwaresystem bezogene Sicherheitslücke ausnutzen, können nicht modelliert werden. Außerdem wird durch die Coverage Regions vorausgesetzt, dass vor der Modellierung mit dem hier neu entwickelten Ansatz bereits eine Modellierung mittels QuAC erfolgt ist. Sollen zusätzlich die Korrektheitswahrscheinlichkeiten und Fehlerwahrscheinlichkeiten, also die Ergebnisse der Auswertung der Modellierung in QuAC, vorliegen, muss die QuAC-Modellierung nicht nur erstellt, sondern auch mittels eines Model Checkers ausgewertet werden. Da die derzeitige Implementierung von QuAC erhebliche Probleme bezüglich der Skalierbarkeit aufweist, ergibt sich hieraus ebenfalls eine Beschränkung in der Modellgröße des neuen Ansatzes.

Ausblick Im Verlauf der Arbeit sowie insbesondere in Kapitel 7 wurden verschiedene Anknüpfungspunkte für mögliche zukünftige Erweiterungen dieser Arbeit aufgezeigt. Für eine konkrete Erweiterung sind im Metamodell die Klassen `CVEVulnerability` und `CWEVulnerability` vorgesehen. Mittels dieser beiden Klassen kann der Ansatz erweitert werden, sodass auch bereits bekannte Schwachstellen und Sicherheitslücken modelliert und basierend darauf die Angriffssicherheit analysiert werden kann.

Kordy u. a. [22] geben in ihrer Arbeit einen Überblick über verschiedenste Arbeiten rund um Angriffsbäume. Dabei berücksichtigen sie nicht nur die Berechnung und Analyse von Angriffen, sondern auch Verteidigungsstrategien. Solche Verteidigungsstrategien könnten einzeln oder in Kombination mit verschiedenen Fähigkeiten von Angreifern modelliert werden. Somit wäre dann beispielsweise nicht jeder Angreifer in der Lage, alle Schwachstellen auszunutzen. Nur diejenigen Angreifer, die die für die jeweilige Verteidigungsstrategie passende Fähigkeit besitzen, wären erfolgreich. Um eine Erweiterung der Arbeit um solche Verteidigungsstrategien zu ermöglichen, existiert bereits eine explizite Modellierung des Angreifers. Diesem könnten beispielsweise Fähigkeiten in Form von einzelnen `Vulnerabilities` zugeordnet werden, die er in der Lage ist, anzugreifen. Über die ebenfalls bereits vorgesehene Erweiterung mittels `AttackRequirements` könnten beispielsweise die Verteidigungsstrategien der Schwachstellen beziehungsweise die für die erfolgreiche Durchführung eines Angriffs erforderlichen Voraussetzungen, also Fähigkeiten des Angreifers, modelliert werden.

Ein weiterer großer Verbesserungspunkt stellt die Verbesserung von QuAC selbst dar. Die Modellierung des Softwaresystems in QuAC bildet eine Voraussetzung für die Nutzung des Ansatzes. Um am Ende nicht nur über das hier berechnete Angriffssicherheitsmaß zu verfügen, sondern zusätzlich auch über die von QuAC berechneten Fehler- und Korrektheitswahrscheinlichkeiten, muss die QuAC-Auswertung mittels eines Model Checkers durchgeführt werden. Hier bestehen in der derzeitigen Implementierung erhebliche Skalierbarkeitsprobleme, die in einer zukünftigen Arbeit untersucht und verbessert werden könnten.

Literatur

[1] *About the OWASP Foundation | OWASP Foundation*. URL: https://owasp.org/about/ (besucht am 25. 01. 2025).

[2] Victor R. Basili und David M. Weiss. „A Methodology for Collecting Valid Software Engineering Data“. In: *IEEE Transactions on Software Engineering* SE-10.6 (Nov. 1984). Conference Name: IEEE Transactions on Software Engineering, S. 728–738. ISSN: 1939-3520. DOI: 10.1109/TSE.1984.5010301. URL: https://ieeexplore.ieee.org/document/5010301 (besucht am 17. 01. 2025).

[3] David Benyon. *Designing interactive systems: a comprehensive guide to HCI and interaction design*. Third edition. Harlow, England: Pearson, 2014. ISBN: 978-1-292-01384-8.

[4] Karin Bernsmed, Martin Gilje Jaatun und Christian Frøystad. „Is a Smarter Grid Also Riskier?“ In: *Security and Trust Management*. Hrsg. von Sjouke Mauw und Mauro Conti. Cham: Springer International Publishing, 2019, S. 36–52. ISBN: 978-3-030-31511-5. DOI: 10.1007/978-3-030-31511-5_3.

[5] Sine Canbolat, Ghada Elbez und Veit Hagenmeyer. „A new hybrid risk assessment process for cyber security design of smart grids using fuzzy analytic hierarchy processes“. In: *at - Automatisierungstechnik* 71.9 (1. Sep. 2023). Publisher: De Gruyter (O), S. 779–788. ISSN: 2196-677X. DOI: 10.1515/auto-2023-0089. URL: https://www.degruyter.com/document/doi/10.1515/auto-2023-0089/html (besucht am 22. 04. 2024).

[6] *CVSS v4.0 Specification Document*. FIRST — Forum of Incident Response and Security Teams. URL: https://www.first.org/cvss/v4.0/specification-document (besucht am 09. 10. 2024).

[7] *CWE - Frequently Asked Questions (FAQ)*. URL: https://cwe.mitre.org/about/faq.html (besucht am 27. 12. 2024).

[8] Christopher Deloglos, Carl Elks und Ashraf Tantawy. „An Attacker Modeling Framework for the Assessment of Cyber-Physical Systems Security“. In: *Computer Safety, Reliability, and Security*. Hrsg. von António Casimiro u. a. Cham: Springer International Publishing, 2020, S. 150–163. ISBN: 978-3-030-54549-9. DOI: 10.1007/978-3-030-54549-9_10.

[9] Edsger W. Dijkstra. „Notes on structured programming“. In: *Structured programming*. GBR: Academic Press Ltd., 1. Jan. 1972, S. 1–82. ISBN: 978-0-12-200550-3. (Besucht am 20. 01. 2025).

[10] *Eclipse IDE | The Eclipse Foundation*. eclipseide.org. URL: https://eclipseide.org/ (besucht am 28. 01. 2025).

[11] *EReference (EMF Documentation)*. URL: https://download.eclipse.org/modeling/emf/emf/javadoc/2.9.0/org/eclipse/emf/ecore/EReference.html#isContainment() (besucht am 28. 01. 2025).

[12] Martin Fowler. *Yagni*. martinfowler.com. 26. Mai 2015. URL: https://martinfowler.com/bliki/Yagni.html (besucht am 28. 01. 2025).

[13] *Glossary | CVE*. URL: https://www.cve.org/ResourcesSupport/Glossary (besucht am 27. 12. 2024).

[14] *Interface IApplication*. Help - Eclipse Platform. URL: https://help.eclipse.org/latest/topic/org.eclipse.platform.doc.isv/reference/api/org/eclipse/equinox/app/IApplication.html (besucht am 28. 01. 2025).

[15] Pontus Johnson, Lagerström Robert und Mathias Ekstedt. „A Meta Language for Threat Modeling and Attack Simulations“. In: 27. Aug. 2018, S. 1–8. DOI: 10.1145/3230833.3232799.

[16] Sotirios Katsikeas u. a. „An Attack Simulation Language for the IT Domain“. In: *Graphical Models for Security*. Hrsg. von Harley Eades III und Olga Gadyatskaya. Cham: Springer International Publishing, 2020, S. 67–86. ISBN: 978-3-030-62230-5. DOI: 10.1007/978-3-030-62230-5_4.

[17] Dirk Knop. *Codefinger-Ransomware verschlüsselt Amazon-S3-Buckets*. heise online. 16. Jan. 2025. URL: https://www.heise.de/news/Codefinger-Ransomware-verschluesselt-Amazon-S3-Buckets-10244874.html (besucht am 20. 01. 2025).

[18] Dirk Knop. *Probleme mit Malware-Schutz in macOS: Bitdefender und Docker betroffen*. heise online. ISSN: 1024-2104. 14. Jan. 2025. URL: https://www.heise.de/news/Probleme-mit-Malware-Schutz-in-macOS-Bitdefender-und-Docker-betroffen-10242104.html (besucht am 20. 01. 2025).

[19] Marie-Claire Koch. *Lauterbach zur elektronischen Patientenakte: SSicherheit hat oberste Priorität"*. heise online. 15. Jan. 2025. URL: https://www.heise.de/news/Lauterbach-zur-ePA-Patientenakte-macht-den-Patienten-zum-Herr-seiner-Daten-10243519.html (besucht am 20. 01. 2025).

[20] Marie-Claire Koch. *Vertrauensdiensteanbieter D-Trust informiert über Datenschutzvorfall*. heise online. 17. Jan. 2025. URL: https://www.heise.de/news/Vertrauensdiensteanbieter-D-Trust-informiert-ueber-Datenschutzvorfall-10246338.html (besucht am 20. 01. 2025).

[21] Alyzia-Maria Konsta u. a. „Survey: Automatic generation of attack trees and attack graphs“. In: *Computers & Security* 137 (1. Feb. 2024), S. 103602. ISSN: 0167-4048. DOI: 10.1016/j.cose.2023.103602. URL: https://www.sciencedirect.com/science/article/pii/S0167404823005126 (besucht am 26. 01. 2025).

[22] Barbara Kordy, Ludovic Piètre-Cambacédès und Patrick Schweitzer. „DAG-based attack and defense modeling: Don’t miss the forest for the attack trees“. In: *Computer Science Review* 13-14 (1. Nov. 2014), S. 1–38. ISSN: 1574-0137. DOI: 10.1016/j.cosrev.2014.07.001. URL: https://www.sciencedirect.com/science/article/pii/S1574013714000100 (besucht am 26. 08. 2024).

[23] Rajesh Kumar und Mariëlle Stoelinga. „Quantitative Security and Safety Analysis with Attack-Fault Trees“. In: *2017 IEEE 18th International Symposium on High Assurance Systems Engineering (HASE)*. 2017 IEEE 18th International Symposium on High Assurance Systems Engineering (HASE). ISSN: 1530-2059. Jan. 2017, S. 25–32. DOI: 10.1109/HASE.2017.12. URL: https://ieeexplore.ieee.org/abstract/document/7911867 (besucht am 28. 08. 2024).

[24] Rajesh Kumar u. a. „Effective Analysis of Attack Trees: A Model-Driven Approach“. In: *Fundamental Approaches to Software Engineering*. Hrsg. von Alessandra Russo und Andy Schürr. Cham: Springer International Publishing, 2018, S. 56–73. ISBN: 978-3-319-89363-1. DOI: 10.1007/978-3-319-89363-1_4.

[25] Florian Lanzinger u. a. *Quantifying Software Correctness by Combining Architecture Modeling and Formal Program Analysis*. 25. Jan. 2024. DOI: 10.1145/3605098.3636008. arXiv: 2401.14320[cs]. URL: http://arxiv.org/abs/2401.14320 (besucht am 22. 04. 2024).

[26] Gary T. Leavens u. a. *JML Reference Manual*. Revision 2344. 2013. URL: http://www.eecs.ucf.edu/~leavens/JML//refman/jmlrefman.pdf (besucht am 20. 01. 2025).

[27] Heiko Mantel und Christian W. Probst. „On the Meaning and Purpose of Attack Trees“. In: *2019 IEEE 32nd Computer Security Foundations Symposium (CSF)*. 2019 IEEE 32nd Computer Security Foundations Symposium (CSF). Hoboken, NJ, USA: IEEE, Juni 2019, S. 184–18415. ISBN: 978-1-7281-1407-1. DOI: 10.1109/CSF.2019.00020. URL: https://ieeexplore.ieee.org/document/8823696/ (besucht am 22. 04. 2024).

[28] Bertrand Meyer. „Applying 'design by contract'“. In: *Computer* 25.10 (Okt. 1992), S. 40–51. ISSN: 0018-9162. DOI: 10.1109/2.161279. URL: http://ieeexplore.ieee.org/document/161279/ (besucht am 18. 01. 2025).

[29] Nitin Naik, Paul Jenkins und Paul Grace. „Cyberattack Analysis Based on Attack Tree with Weighted Average Probability and Risk of Attack“. In: *Advances in Computational Intelligence Systems*. Hrsg. von George Panoutsos, Mahdi Mahfouf und Lyudmila S. Mihaylova. Cham: Springer Nature Switzerland, 2024, S. 324–333. ISBN: 978-3-031-55568-8. DOI: 10.1007/978-3-031-55568-8_27.

[30] *Overview | CVE*. URL: https://www.cve.org/About/Overview (besucht am 27. 12. 2024).

[31] *OWASP Risk Rating Methodology | OWASP Foundation*. URL: https://owasp.org/www-community/OWASP_Risk_Rating_Methodology (besucht am 28. 11. 2024).

[32] Nikolaos Polatidis, Michalis Pavlidis und Haralambos Mouratidis. „Cyber-attack path discovery in a dynamic supply chain maritime risk management system“. In: *Computer Standards & Interfaces* 56 (1. Feb. 2018), S. 74–82. ISSN: 0920-5489. DOI: 10.1016/j.csi.2017.09.006. URL: https://www.sciencedirect.com/science/article/pii/S0920548917302301 (besucht am 26. 08. 2024).

[33] Nikolaos Polatidis u. a. „From product recommendation to cyber-attack prediction: generating attack graphs and predicting future attacks“. In: *Evolving Systems* 11.3 (1. Sep. 2020), S. 479–490. ISSN: 1868-6486. DOI: 10.1007/s12530-018-9234-z. URL: https://doi.org/10.1007/s12530-018-9234-z (besucht am 26. 08. 2024).

[34] *Precision and recall*. In: *Wikipedia*. Page Version ID: 1268160106. 8. Jan. 2025. URL: https://en.wikipedia.org/w/index.php?title=Precision_and_recall&oldid=1268160106 (besucht am 21. 01. 2025).

[35] Ralf Reussner u. a. *Modeling and Simulating Software Architectures: The Palladio Approach*. Cambridge, Massachusetts: MIT Press, 2016. 377 S. ISBN: 978-0-262-03476-0. URL: https://mitpress.mit.edu/modeling.

[36] Per Runeson und Martin Höst. „Guidelines for conducting and reporting case study research in software engineering". In: *Empirical Software Engineering* 14.2 (1. Apr. 2009), S. 131–164. ISSN: 1573-7616. DOI: 10.1007/s10664-008-9102-8. URL: https://doi.org/10.1007/s10664-008-9102-8 (besucht am 13. 01. 2025).

[37] Dennis Schirrmacher. *Sicherheitspatch: Unbefugte Zugriffe auf bestimmte Switches von Moxa möglich*. heise online. 20. Jan. 2025. URL: https://www.heise.de/news/Sicherheitspatch-Unbefugte-Zugriffe-auf-bestimmte-Switches-von-Moxa-moeglich-10249285.html (besucht am 20. 01. 2025).

[38] Jürgen Schmidt. *Kopierdienst rsync mit kritischer Lücke*. Security. 15. Jan. 2025. URL: https://www.heise.de/news/Sechs-Bugs-in-rsync-Server-gefixt-einer-davon-kritisch-10244157.html (besucht am 20. 01. 2025).

[39] Bruce Schneier. *Attack Trees*. Schneier on Security. URL: https://www.schneier.com/academic/archives/1999/12/attack_trees.html (besucht am 26. 11. 2024).

[40] Teodor Sommestad, Mathias Ekstedt und Hannes Holm. „The Cyber Security Modeling Language: A Tool for Assessing the Vulnerability of Enterprise System Architectures". In: *IEEE Systems Journal* 7.3 (Sep. 2013). Conference Name: IEEE Systems Journal, S. 363–373. ISSN: 1937-9234. DOI: 10.1109/JSYST.2012.2221853. URL: https://ieeexplore.ieee.org/abstract/document/6378394 (besucht am 08. 05. 2024).

[41] Andreas Spillner und Tilo Linz. *Basiswissen Softwaretest: Aus- und Weiterbildung zum Certified Tester - Foundation Level nach ISTQB-Standard*. 7. Aufl. Heidelberg: dpunkt.verlag, 2024. ISBN: 978-3-98889-005-4.

[42] John Viega und Gary McGraw. *Building secure software: How to Avoid Security Problems the Right Way*. Addison-Wesley professional computing series. Boston, Mass.: Addison-Wesley Professional Computing Series, 2002.

[43] Maximilian Walter, Robert Heinrich und Ralf Reussner. „Architectural Attack Propagation Analysis for Identifying Confidentiality Issues". In: *2022 IEEE 19th International Conference on Software Architecture (ICSA)*. 2022 IEEE 19th International Conference on Software Architecture (ICSA). März 2022, S. 1–12. DOI: 10.1109/ICSA53651.2022.00009. URL: https://ieeexplore.ieee.org/document/9779702 (besucht am 08. 05. 2024).

[44] Maximilian Walter, Robert Heinrich und Ralf Reussner. „Architecture-Based Attack Path Analysis for Identifying Potential Security Incidents". In: *Software Architecture*. Hrsg. von Bedir Tekinerdogan u. a. Cham: Springer Nature Switzerland, 2023, S. 37–53. ISBN: 978-3-031-42592-9. DOI: 10.1007/978-3-031-42592-9_3.

A. Anhang

A.1. Coverage Regions für Energienetz-Beispielszenario

```
1  this_maxWireLoad.VALUE <= -1 + addedCharge.VALUE
2  AND this_maxStorageLoad.VALUE >= 0
3  AND this_storageLoad.VALUE >= 0
4  AND this_storageLoad.VALUE <= this_maxStorageLoad.VALUE
5
6  addedCharge.VALUE >= 1
7  AND this_storageLoad.VALUE >= 1 + addedCharge.VALUE * -1
8      + this_maxStorageLoad.VALUE
9  AND this_maxWireLoad.VALUE >= addedCharge.VALUE
10 AND this_maxStorageLoad.VALUE >= 0
11 AND this_storageLoad.VALUE >= 0
12 AND this_storageLoad.VALUE <= this_maxStorageLoad.VALUE
13
14 this_storageLoad.VALUE <= -1 + addedCharge.VALUE * -1
15 AND this_storageLoad.VALUE <= addedCharge.VALUE * -1
16     + this_maxStorageLoad.VALUE
17 AND this_maxWireLoad.VALUE >= addedCharge.VALUE
18 AND this_maxStorageLoad.VALUE >= 0
19 AND this_maxStorageLoad.VALUE >= addedCharge.VALUE
20 AND addedCharge.VALUE <= -1
21 AND this_storageLoad.VALUE >= 0
22 AND this_storageLoad.VALUE <= this_maxStorageLoad.VALUE
```

Listing A.1: Im Rahmen der QuAC-Fallstudie berechnete Direct Error Conditions für Network::addCharge [25].

```
this_maxWindSpeed.VALUE <= -1 + sensor_windSpeed.VALUE
AND sensor_windSpeed.VALUE >= 1
AND this_maxWindSpeed.VALUE >= 0
AND this_maxWindSpeed.VALUE <= 4
AND sensor_windSpeed.VALUE >= 0
AND network_storageLoad.VALUE >= 0
AND network_storageLoad.VALUE <= sensor_windSpeed.VALUE
```

Listing A.2: Im Rahmen der QuAC-Fallstudie berechnete Direct Error Conditions für WindTurbine::run [25].

```
usedEnergy.VALUE >= 1
AND this_totalUsedEnergy.VALUE >= 1+ usedEnergy.VALUE * -1
    + this_allowance.VALUE
AND this_allowance.VALUE >= usedEnergy.VALUE * -1
AND this_totalUsedEnergy.VALUE >= usedEnergy.VALUE * -1
AND network_maxStorageLoad.VALUE >= 0
AND network_maxStorageLoad.VALUE >= usedEnergy.VALUE * -1
AND network_storageLoad.VALUE >= 0
AND network_storageLoad.VALUE <= network_maxStorageLoad.VALUE
AND this_allowance.VALUE >= 0
AND this_totalUsedEnergy.VALUE >= 0
AND this_totalUsedEnergy.VALUE <= this_allowance.VALUE
AND network_maxStorageLoad.VALUE >= usedEnergy.VALUE
AND network_storageLoad.VALUE >= usedEnergy.VALUE
AND network_storageLoad.VALUE <= usedEnergy.VALUE
    + network_maxStorageLoad.VALUE
```

Listing A.3: Im Rahmen der QuAC-Fallstudie berechnete Direct Error Conditions für EnergyConsumer::useEnergy [25].

A.2. Ergebnisse der Skalierbarkeitsanalyse

A.2.1. Durchschnittliche Ausführungszeiten

		Anzahl parallele AND-Knoten				
		1	**10**	**100**	**1000**	**10000**
Anzahl Attacker	**1**	1,222 s	1,235 s	1,311 s	1,714 s	3,282 s
	10	1,331 s	1,260 s	1,466 s	2,177 s	5,475 s
	100	1,553 s	1,622 s	2,006 s	4,982 s	25,878 s
	1000	2,770 s	3,059 s	5,213 s	23,658 s	228,624 s

Tabelle A.1.: Durchschnittliche Ausführungszeiten [s] der Skalierbarkeitsanalyse der Evaluation.

A.2.2. Rohdaten der Ausführungszeiten

AND parallel	Zeit 1	Zeit 2	Zeit 3	Zeit 4	Zeit 5
1	00:01,262	00:01,178	00:01,110	00:01,261	00:01,300
10	00:01,234	00:01,149	00:01,261	00:01,216	00:01,313
100	00:01,211	00:01,271	00:01,512	00:01,277	00:01,285
1000	00:01,480	00:01,787	00:01,775	00:01,827	00:01,699
10000	00:03,431	00:03,230	00:03,007	00:03,405	00:03,339

Tabelle A.2.: Ausführungszeiten der Skalierbarkeitsanalyse der Evaluation für einen Attacker in Minuten.

AND parallel	Zeit 1	Zeit 2	Zeit 3	Zeit 4	Zeit 5
1	00:01,310	00:01,226	00:01,332	00:01,506	00:01,281
10	00:01,270	00:01,145	00:01,263	00:01,463	00:01,158
100	00:01,297	00:01,597	00:01,504	00:01,465	00:01,468
1000	00:02,035	00:02,149	00:02,193	00:02,314	00:02,195
10000	00:05,339	00:05,387	00:05,567	00:05,528	00:05,552

Tabelle A.3.: Ausführungszeiten der Skalierbarkeitsanalyse der Evaluation für 10 Attacker in Minuten.

AND parallel	Zeit 1	Zeit 2	Zeit 3	Zeit 4	Zeit 5
1	00:01,560	00:01,520	00:01,688	00:01,502	00:01,495
10	00:01,698	00:01,545	00:01,638	00:01,603	00:01,624
100	00:01,902	00:02,071	00:02,051	00:02,069	00:01,935
1000	00:04,580	00:05,074	00:05,060	00:05,141	00:05,057
10000	00:25,565	00:25,746	00:25,758	00:25,984	00:26,338

Tabelle A.4.: Ausführungszeiten der Skalierbarkeitsanalyse der Evaluation für 100 Attacker in Minuten.

AND parallel	Zeit 1	Zeit 2	Zeit 3	Zeit 4	Zeit 5
1	00:02,567	00:02,848	00:02,736	00:03,055	00:02,652
10	00:02,960	00:03,043	00:02,854	00:03,254	00:03,184
100	00:05,179	00:05,004	00:04,995	00:05,511	00:05,376
1000	00:23,378	00:24,695	00:22,985	00:22,951	00:24,279
10000	03:51,709	03:51,189	03:45,207	03:49,209	03:45,807

Tabelle A.5.: Ausführungszeiten der Skalierbarkeitsanalyse der Evaluation für 1000 Attacker in Minuten.