universität wien

MASTERARBEIT / MASTER'S THESIS

Titel der Masterarbeit / Title of the Master's Thesis

Sicherheit von Datenbanken – Ein Framework zur Überprüfung und Bewertung der Sicherheit von relationalen Datenbanken

verfasst von / submitted by

Dominik Dinulovic, BSc

angestrebter akademischer Grad / in partial fulfilment of the requirements for the degree of

Diplom-Ingenieur (Dipl.-Ing.)

Wien, 2017 / Vienna 2017

Studienkennzahl lt. Studienblatt / degree programme code as it appears on the student record sheet:	A 066 926
Studienrichtung lt. Studienblatt / degree programme as it appears on the student record sheet:	Masterstudium Wirtschaftsinformatik
Betreut von:	Univ.-Prof. Dipl.-Ing. Dr. Dr. Gerald Quirchmayr

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides Statt, dass ich die vorliegende Arbeit selbstständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

Wien, März 2017 Dominik Dinulovic

Zusammenfassung

Die Sicherheit von Datenbanken ist gegenwärtig von hoher Relevanz. Schon ein kleiner Fehler bei den Policies oder bei der Konfiguration kann zu einer instabilen Struktur und zu erhöhten Kosten führen. Weiterhin sind Datenbanken bei unzureichender Sicherung potenziellen Risiken ausgesetzt. Oft kann es aufgrund dessen zum Missbrauch durch autorisierte Nutzer, Hacker sowie Schadprogramme kommen, wodurch Daten gestohlen oder beschädigt werden. In dieser Arbeit werden die verbreitetsten Datenbankbedrohungen auf einer relationalen Datenbank simuliert, mit dem Ziel, Schwachstellen zu ermitteln und anschließend den Schaden des Angriffs mit Hilfe der Security Metrics zu bewerten. Die Aufstellung der zu schützenden Sicherheitsziele, wie etwa die Vertraulichkeit, Integrität oder Verfügbarkeit sind dabei von entscheidender Bedeutung. In diesem Zusammenhang ist es notwendig das potenzielle Schadensausmaß der Bedrohungen in Bezug auf die Sicherheitsziele zu identifizieren. Darauf basierend können weitere Sicherheitsziele einer Datenbank analysiert und folglich adaptiert oder verstärkt werden. Die Aufstellung verschiedener Security Metrics bietet beispielsweise eine große Unterstützung zur Wahrnehmung der Schwachstellen in der Softwarearchitektur, den Sicherheitsoperationen und den Managemententscheidungen. Um effektive Ergebnisse zu erzielen, müssen alle durchgeführten Aktionen gemessen und bewertet werden. Hierzu ist die Einbindung einer Skala sehr wirkungsvoll, die wie folgt definiert ist: schlecht, durchschnittlich, gut oder keine Angabe. Weiterhin werden die Security Metrics anhand bestimmter Prinzipien zu den bereits genannten Sicherheitszielen aufgestellt. Im Verlauf der Arbeit wird ebenfalls ein Framework implementiert, das dem Benutzer die Möglichkeit gibt, Bedrohungen auf einer Oracle Datenbank zu simulieren, die daraufhin ausgewertet und anhand der Security Metrics bewertet werden. Abschließend hat der Benutzer die Gelegenheit weitere Details wie den Zeitpunkt, Benutzernamen oder die IP-Adresse zu den simulierten Bedrohungen aufzurufen. Dadurch soll ein größerer Überblick über das Schadensausmaß einer Bedrohung ermöglicht werden. Letztendlich wird im Verlauf der Arbeit ein Fokus auf Sicherheit, Security Metrics, SQL und relationale Datenbanken gelegt.

Abstract

The security of databases is today accepted as an important factor for securing IT environments. Even a small mistake, based on policy or configuration errors, can lead to unstable structures and higher costs. Hackers or even authorized users may abuse information of databases. Moreover, there are many threats with intentions to steal or manipulate passwords and data. The topic of this work is to simulate common database threats on a relational database, with the goal of identifying vulnerabilities. Furthermore, security metrics are used to assess the damage of the executed threats. Confidentiality, integrity and availability are the traditional security goal examples, which are also used in this work. It is necessary to figure out which threat is afflicting damage to which security goal. In this sense it is possible to analyze security goals and if required to adapt or enhance them. Security metrics for example support identifying software architecture, security operation, and management vulnerabilities. To achieve effective results, all realized activities need to be measured and evaluated. Therefore, a rating scale can be very effective. The result of this might be weak, average, good or not specified. Furthermore, it is important to mention that security metrics are based on security goals. Throughout the progress of this work, a framework will be implemented. With this framework, the user is able to simulate threats on an Oracle database and afterwards to make an assessment, based on the security metrics. The user has the opportunity to invoke detailed information about the threat that contains the Timestamp, Username or IP-Address. This feature enables a better understanding about the measure of damages caused by a threat. IT security, security metrics, SQL security and relational databases are the major topics addressed in this work.

Inhaltsverzeichnis

1 Einleitung ... 8
2 Problembeschreibung ... 10
3 Erwartete Ergebnisse ... 11
4 Methoden und Vorgehen ... 12
5 State of the Art ... 13
5.1 Datenbankmodelle ... 13
5.2 Datenbankmanagementsysteme ... 22
5.3 Allgemeine Sicherheitsanforderungen in Datenbanken ... 26
5.4 Bedrohungen im Datenbankbereich ... 30
5.4.1 Excessive Privilege Abuse ... 30
5.4.2 Legitimate Privilege Abuse ... 31
5.4.3 SQL-Injection ... 31
5.4.4 Denial of Service (DoS) ... 33
5.4.5 Weak Audit Trail ... 35
5.5 Security Metrics ... 38
6 Literatur und Praxis ... 45
6.1 Security Metrics und Messungen im Datenbankbereich ... 45
6.1 Einsätze der Security Metrics ... 48
6.2 Vor- und Nachteile ... 51
6.3 Verfahren zur Erstellung von Security Metrics ... 52
6.3.1 Threat and Vulnerability Analysis ... 53
6.3.2 Dekomposition Methode ... 54
6.3.3 Goal-Question-Metric (GQM) Methode ... 54
6.4 Unterschiede zwischen Literatur und Praxis ... 55
7 Gap Analysis ... 57
8 Framework Entwicklung ... 61
8.1 Modell ... 62
8.2 Metamodell ... 68
8.3 Sprache & Toolunterstützung ... 69
8.4 Implementierung ... 70
9 Case Study ... 76

9.1 Auswertung der Case Study 76
9.1.1 Brute-Force Simulation 78
9.1.2 SQL-Injection Simulation 85
9.1.3 Ping of Death Simulation 94
9.1.4 SYN-Flooding Simulation 98
9.2 Vergleiche und Bewertungen der Ergebnisse 106
10 Erwartete Ergebnisse vs. Case Study 107
11 Conclusio 109
Tabellenverzeichnis 110
Abbildungsverzeichnis 111
References 113

1 Einleitung

Datenbanken sind ein wesentlicher Bestandteil jedes Unternehmens. Dadurch das eine große und sensible Menge an Daten gespeichert werden müssen, muss die Datenbank dementsprechend geschützt werden. Hilfreich ist die Auswertung und Bewertung des Schadens bei Angriffen, um zukünftig besser auf potenzielle Bedrohungen vorbereitet zu sein. Das Ziel dieser Arbeit ist, die Sicherheit einer Datenbank anhand simulierter Bedrohungen und aufgestellter Security Metrics auszuwerten sowie zu bewerten. Dazu bedarf es einer detaillierten Literaturrecherche, um zu ermitteln welche Datenbankbedrohungen allgegenwärtig sind. Ein großer Fokus liegt dabei auf Brute-Force, SQL-Injection, Ping of Death und SYN Flooding Attacken. Ein wichtiger Punkt dieser Arbeit basiert auf Audit Trails [2]. Durch eine schlechte Überwachung entstehen Fehler, wodurch Datenbanktabellen manipuliert werden können. Die Erstellung der Security Metrics ist notwendig, um zukünftige Gefahren zu vermeiden und Datenbankbedrohungen zu bewerten. Ein wichtiger Bestandteil ist den Unterschied zwischen Literatur und Praxis darzustellen. Durch die Entwicklung eines Tools sollen relevante Ergebnisse geliefert werden. Der Benutzer hat somit die Möglichkeit eine Bedrohung zu simulieren und anhand der Security Metrics zu bewerten. Allerdings befindet sich dieses Thema noch in einem frühen Stadium, in dem erst seit kurzer Zeit Ressourcen investiert werden [3]. In der Theorie sind viele Informationen zu Security Metrics verfügbar, aber in der Praxis wurden bisher unbrauchbare Resultate erzielt [4][5]. Somit ist die theoretische Aufbereitung des Wissens von großer Bedeutung, um praktisch positive Ergebnisse erreichen zu können.

In den Anfangsjahren des Internets wurde die Sicherheit in IT-Systemen unterschätzt, wodurch der Datenbank-, Netzwerk- oder Software Architekturbereich anfällig für potenzielle Bedrohungen wurde. Die Folge waren Angriffe durch Hacker oder Datenmanipulationen, aufgrund entstehender Vulnerabilitäten innerhalb der Datenbanksysteme, wobei die bisher größte Gefahr von dem Datenbanknutzer selbst ausgeht [7]. Sobald ein Benutzer alle Rechte innerhalb der Datenbank besitzt, besteht ein größeres Risiko manipulierte oder gestohlene Daten aufzufinden. Dies könnte beispielsweise durch Zuteilung von Rollen und Rechten unter Kontrolle gehalten werden. Gleichermaßen gefährlich sind SQL-Injections sowie Denial of Services, auf die im Verlauf der Arbeit detaillierter eingegangen wird. Obwohl bereits viele Sicherheitsmaßnahmen getroffen wurden, muss identifiziert werden, wie Bedrohungen entstehen und welchen Schaden sie anrichten können.

Seit einigen Jahren ist man nun mehr darauf fokussiert bestimmte Richtlinien und Maßnahmen zu erstellen, um eine entsprechende Sicherheit ermöglichen zu können. Deshalb arbeiten viele internationale Organisationen, wie die International Telecommuncation Unit (ITU) oder die European Network and Information Security Agency (ENISA) daran, Sicherheitsrichtlinien und Security Metrics zu definieren sowie aufzustellen [3][5]. Der Begriff Security Metrics ist mittlerweile zu einer standardisierten Bezeichnung herangewachsen und dient als Synonym für Security Performance, Security Indicators und Security Strength [3]. Ein allgemeiner Zusammenhang basiert auf der Kombination der Messungen und Security Metrics, wobei der folgende Unterschied betrachtet werden muss: Messungen beziehen sich auf

einen bestimmten Datensatz hinsichtlich spezifisch formulierter Faktoren, wobei Security Metrics die abgeleiteten Daten der Messungen beschreiben. Security Metrics sind dadurch zweckdienlich für Optimierungen im Management-, Monitoring- oder Performancebereich [3]. Daraufhin müssen Sicherheitsziele identifiziert werden. Die am häufigsten in Verwendung tretenden Ziele sind die Vertraulichkeit, Verfügbarkeit und Integrität. Das Ziel der Vertraulichkeit ist nur autorisierten Benutzern einen Zugang zu ermöglichen. Weiters ist die Integrität für die Genauigkeit und Vollständigkeit der Daten zuständig. Mit Hilfe der Verfügbarkeit wird nur berechtigten Benutzern ein Zugriff auf notwendigen Daten zur Verfügung gestellt [3]. Daraufhin können, durch eine Zerteilung in Low-Level Ebenen, Security Metrics aufgestellt werden. Die ITU hat in diesem Zuge weitere Sicherheitsziele, wie zum Beispiel die Authentifizierung oder Kommunikationssicherheit identifiziert [3]. Für die Umsetzung dieser Arbeit wurden deshalb die bedeutendsten Sicherheitsziele für Security Metrics ausgewählt. Diese sind die Vertraulichkeit, Verfügbarkeit, Integrität, Authentifizierung und Verschlüsselung.

Die weiteren Schritte zu dieser Arbeit beinhalten eine gründliche Problembeschreibung sowie die Beschreibung der erwarteten Ergebnisse und die dazu notwendigen Vorgehensweisen. Daraufhin erfolgt eine Offenlegung der erforderlichen Literaturrecherche in Bezug auf den State of the Art der Datenbanken. Hier werden Datenbankmodelle, Datenbankmanagementsysteme, allgemeine Sicherheitsanforderungen sowie Bedrohungen thematisiert. Infolgedessen muss ein Unterschied zwischen der Literatur und Praxis festgestellt werden. Dazu wird näher auf Security Metrics, Messungen und deren Vor- und Nachteile eingegangen. Anschließend folgt eine detaillierte Beschreibung der entwickelten Anwendung mit den dazu relevanten Modellen und Metamodellen. Letztlich werden alle ausgewerteten Testdurchgänge dokumentiert und deren Ergebnisse diskutiert.

2 Problembeschreibung

Der Umgang mit großen Datenmengen erfordert eine präzise Planung der Strategie und Sicherheit. Deshalb ist die qualitativ hochwertige Sicherung sensibler Daten eine Voraussetzung, die in den meisten Fällen vernachlässigt oder unzureichend gewährleistet wird. Dadurch ist das Angriffspotenzial von Datenbanken erhöht, weswegen die Ermittlung von Schwachstellen und Risiken proportional dazu an Bedeutung gewinnt. Hierbei sind die Kompetenzen und Programmierkenntnisse von Angriffen Außenstehender ein signifikanter Faktor, der Berücksichtigt werden muss. Folglich können Hackerangriffe oder Bedrohungen wie SQL-Injections und Denial-of-Services bei unzureichender Sicherung potenziell großen Schaden anrichten. Angesichts des soeben geschilderten Bedrohungsszenario von außen, besteht ein hohes Missbrauchsrisiko seitens der Datenbanknutzer. Trotz unterschiedlich vorhandener Lösungsansätze werden im Sicherheitsbereich laufend Lücken entdeckt. Um spezifische Probleme zu lösen, müssen die betreffenden Vorgänge zuerst simuliert werden. Demnach entsteht ein Basiswissen für zukünftige Bedrohungen. Durch unterschiedliche Testphasen in wechselnden Umgebungen kann das Ausmaß einer Bedrohung analysiert und ausgewertet werden. Dies soll zu einem besseren Verständnis der potenziellen Gefahren beitragen. Dabei sind Security Metrics von signifikanter Bedeutung. In diesem Zusammenhang ist die Identifikation von Sicherheitszielen notwendig, die somit definieren welche Sicherheitsbereiche abgedeckt werden müssen.

Folglich soll eine bestehende Datenbank in einem lokalen Netzwerk unterschiedlichen Gefahren ausgesetzt werden. Dadurch sollen Bedrohungen anschließend gemessen und bewertet werden. Mittels Messkriterien soll die Relevanz der Security Metrics, sowie die Nutzbarkeit der daraus erhobenen Ergebnisse eruiert werden. Die Simulation von Bedrohungen ist ein notwendiger Faktor, um Daten zu sammeln und effektive Ergebnisse zu erzielen. Basierend auf diesem Fundament ist zudem die Erarbeitung des Status Quo von Audit Trails erforderlich, damit überprüft werden kann, inwiefern sich diese in der Praxis als hilfreich erweisen. Hierfür bedarf es der Selektion einer adäquaten Testumgebung in der die Eignung eines lokalen Netzwerks definiert wird, um den größenspezifischen Unterschieden von Netzwerken in Bezug auf das Gefährdungspotenzial oder die Performance gerecht zu werden. Letztlich liegt die Motivation darin die bestehende Sicherheit einer Oracle-Datenbank in einem lokalen Netzwerk anhand Security Metrics auszuwerten und zu bewerten.

3 Erwartete Ergebnisse

Oracle bietet in vielerlei Hinsicht ein gutes Sicherheitskonzept. Dadurch können Überwachungen und Monitoring gewährleistet, aber auch Datenanalysen erstellt werden. Wie aber auch jede weitere Software ist Oracle nicht unbezwingbar. Nach Angaben wissenschaftlicher Publikationen bestehen große Gefahren durch Denial-of-Services, Trojaner oder Malware-Attacken [20]. Durch die eben genannten Beispiele ist es gelungen, große Datenmengen von bekannten Unternehmen zu stehlen oder zu manipulieren. Laut einer Studie der IDC (International Data Corporation) besteht die größte potenzielle Gefahr bei den Mitarbeitern eines Unternehmens selbst [21]. Dies kann zu manipulierten Informationen sowie Enthüllung der Daten führen.

Die zu erzielenden Resultate sollen durch simulierte Bedrohungen auf einer Oracle Datenbank, installiert auf einem lokalen Rechner, durchgeführt und dokumentiert werden. Dabei ist zu erwarten, dass SQL-Injections einen Schaden auf der Datenbank verursachen können. Zu diesem Thema sind viele Onlineressourcen vorhanden, die allerdings nicht auf Oracle beschränkt sind. Das Ziel ist, eine potenzielle Gefahr durch SQL-Injection erzeugen zu können, um beispielsweise Tabellen zu manipulieren oder auch als Benutzer in die Datenbank einzudringen. Die Umsetzung einer Brute-Force Attacke ist hingegeben schwierig abzuschätzen, weil diese Bedrohung in einer Oracle Datenbank einer Identifikation bedarf. Trotz eines elaborierten Literaturfundus zu dieser Problematik, ist eine Durchführung von praktischen Tests nicht ersetzbar [62]. Durch die hervorgehenden Testergebnisse können weitere Charakteristiken der Bedrohung analysiert werden. Nach einigen Einschätzungen und Recherchen wird angenommen, dass Brute-Force in der Praxis keine Auswirkung auf einer Oracle Datenbank ausüben kann.

Denial-of-Service Attacken sind derzeitig noch immer sehr präsent und können beim Datenbanklistener einen großen Schaden anrichten [42]. Sowohl bei der Ping of Death als auch der SYN-Flooding Simulation wird von einem erfolgreichen Angriff ausgegangen. Dadurch soll der Datenbanklistener, der in der Datenbank für den Kommunikationsaustausch zuständig ist, vollständig abstürzen und keine weitere Verbindungsanfrage aufnehmen können. Letztlich wird der simulierte Schaden mit Hilfe der Security Metrics bewertet.

4 Methoden und Vorgehen

Die von Hevner definierten Design-Science Research Richtlinien sind für die positiv zu erzielenden Resultate dieser Arbeit ausschlaggebend [66]. Folglich werden die identifizierten Richtlinien näher erläutert.

Design as an artifact: Das Design-Science Research muss ein brauchbares Artefakt in Form eines Konstruktes, eines Modells, eines Verfahrens oder einer Instanziierung liefern. Dabei liegt der Fokus auf dem zu entwickelnden Framework mit dem dazugehörigen Klassenmodell sowie Metamodell und der Softwarearchitektur.

Problem relevance: Das Ziel der Design-Science Research ist die Entwicklung einer technologischen Lösung bezogen auf relevante Businessprobleme. Hier liegt das Augenmerk auf den aufzufindenden Schwachstellen in Oracle Datenbanken anhand der zu entwickelnden Bedrohungen, die mit Hilfe des Frameworks realisiert werden.

Design evaluation: Die Nützlichkeit, Qualität und Effizienz eines Designartefakts muss anhand einer Auswertungsmethode exakt demonstriert werden. Mit Hilfe der Security Metrics wird schließlich die Effektivität und Korrektheit der Datenbank bewertet. Dazu gehört eine detaillierte Auswertung der auszuführenden Bedrohungen.

Research contributions: Ein effektives Design-Science Research muss klare und nachweisbare Beiträge in den Bereichen des Designartefakts, der Designgrundlage und Designmethodologie anbieten. Anhand der aufgestellten Sicherheitsziele und den darauf basierenden Security Metrics wird die Zielvorgabe eindeutig definiert. Durch die Bedrohungen entstehen nachweisbare Beiträge darüber, ob die Zielvorgabe eingehalten und wie sie gegebenenfalls verbessert oder adaptiert werden kann.

Research rigor: Das Design-Science Research beruht auf strikten Applikationsverfahren, sowohl in der Konstruktion, als auch Evaluation eines Designartefakts. Durch das MVC-Pattern (Model-View-Controller) wird ein stabiles Konstrukt des Frameworks ermöglicht. Darüber hinaus ist es notwendig, Verfahren zur Erstellung der Security Metrics aufzufinden und umzusetzen, um eine Grundlage für das Framework zu schaffen.

Design as a search process: Die Suche nach einem effektiven Artefakt erfordert die Verwendung verfügbarer Mittel, um gewünschte Ziele zu erreichen und die Regelungen der Problemumgebung einzuhalten. Durch die Zuhilfenahme unterschiedlicher Tools wie JavaFX, Virtual Box oder Oracle wird eine Entwicklungsumgebung geschaffen, in der Testphasen und Auswertungen ermöglicht werden.

Communication of research: Das Design-Science Research muss technologie-orientierten und management-orientierten Zielgruppen effektiv vorgetragen werden.

5 State of the Art

Das kommende Kapitel dient als fundamentale Grundlage für den Verlauf der Arbeit. Dafür war es notwendig, nach qualitativer Literatur zu recherchieren. Dabei diente die verfasste Publikation “Database Management Systems“ von Ramakrishnan als grundlegende Basis [1]. Weiterhin relevant war die Ausgabe von Watt [17].

In diesem Kapitel werden die verschiedenen Arten der Datenbankmodelle mit deren Vor- und Nachteilen definiert. Außerdem erfolgt eine ausführliche Beschreibung des Datenbankmanagementsystems, sowie der allgemeinen Sicherheitsanforderungen und den Datenbankbedrohungen. Alle Beispiele innerhalb der Unterkapitel 5.1 und 5.2 basieren auf den eben erwähnten Publikationen von Ramakrishnan und Watt [1][17]. Für die Aufstellung der unterschiedlichen Datenbankmodelle waren die Werke von Compendio und Pöschek bedeutend [18][25].

5.1 Datenbankmodelle

Die Grundlage des relationalen Modells basiert auf Tabellen, die als Relationen definiert sind, in denen Datensätze abgespeichert werden. Dementsprechend ist die Tabelle in Tupel und Spalten aufgeteilt. Wobei jedes Tupel einen Datensatz und jede Spalte einen Attributwert repräsentiert. Somit legt ein Relationsschema die Anzahl und den Typ eines Attributes fest.

Hierbei wird von folgend exemplarischen Szenario ausgegangen: Ein Unternehmen mit einer großen Anzahl an Mitarbeitern und Projekten soll gewartet werden. Damit ein Mitarbeiter eindeutig identifiziert werden kann, ergibt sich eine Relation nach folgendem Schema.

Mitarbeiter (Mitarbeiter-ID: INTEGER, Nachname: VARCHAR(30),
Vorname: VARCHAR(30), Geburtstag: DATE)

Folglich definiert das Schema eine Relation *Mitarbeiter* mit vier Spalten sowie Spaltennamen und Typen. Damit jeder Datensatz eindeutig identifizierbar ist, muss ein Schlüssel definiert werden, der unverändert bleiben muss. In diesem Fall ist die Spalte *Mitarbeiter-ID* der Schlüssel der Relation. Dazu ein praktisches Beispiel.

Mitarbeiter-ID	Nachname	Vorname	Geburtstag
124563	Müller	Thomas	11.10.1990
143874	Bauer	Martin	02.05.1992

Tabelle 1: Instanzen der Relation Mitarbeiter

Ebenso können Verknüpfungen erstellt werden, die Beziehungen zwischen mehreren Tabellen darstellen. Beispielsweise hat die Relation *Mitarbeiter* für jeden Mitarbeiter einen Eintrag. Im folgenden Szenario arbeitet ein Mitarbeiter an einem Projekt des Unternehmens. Das *Projekt* besteht ebenfalls als Relation mit den Spalten *Projekt-ID,*

Name und *Kunde*. Um festlegen zu können welcher Mitarbeiter bei welchem Projekt eingesetzt wird, benötigt man eine dritte Relation *Department*. Hier wird angegeben, welcher Mitarbeiter bei welchem Projekt tätig ist. Aus diesem Grund müssen aus den Relationen *Mitarbeiter* und *Projekt* bestimmte Schlüssel definiert werden. In diesem Fall sind die Spalten *Mitarbeiter-ID* und *Projekt-ID* eindeutig. Dementsprechend beinhaltet die Relation *Department* die zwei zuletzt genannten Spalten. Dadurch kann eine Mitarbeiter ID einer Projekt ID zugeordnet werden. Zeitgleich sind die zwei Spalten der Relation *Department* mit einem Eintrag der Relation *Mitarbeiter* und *Projekt* verbunden. Dieser Begriff wird auch als Fremdschlüssel bezeichnet.

Zur Erstellung von Relationen wird in der Praxis die Datenbanksprache Structured Query Language (SQL) verwendet. Diese hat eine einfache Struktur und wird zum Bearbeiten, Einfügen, Verändern, Löschen und Abfragen von Datensätzen benutzt. Weiterhin lässt sich SQL in drei Kategorien unterteilen. Folglich sind das die Data Manipulation Language (DML), Data Definition Language (DDL) und die Data Control Language (DCL). Die DML Befehle werden großflächig zur Datenmanipulation zur Verfügung gestellt. Hier wird das Ändern, Einfügen und Löschen von Datensätzen befähigt. In der DDL werden Befehle zur Definition des Datenbankschemas und in der DCL jene, für die Rechteverwaltung und Transaktionskontrolle definiert. Abfragen können durch verschiedene Statements mittels SQL durchgeführt werden. Um eine Tabelle zu erstellen, wird das Statement *CREATE TABLE* verwendet. Somit könnte die Erstellung der Tabelle *Mitarbeiter* folgenderweise aussehen.

```
CREATE TABLE Mitarbeiter (Mitarbeiter-ID INTEGER,
                          Nachname VARCHAR(30),
                          Vorname VARCHAR(30),
                          Geburtstag DATE)
```

Letztlich gleicht das Statement dem vorher definierten Schema. Damit die Tabelle mit Informationen befüllt wird, muss das folgende Statement benutzt werden.

```
INSERT
INTO Mitarbeiter (Mitarbeiter-ID, Nachname, Vorname, Geburtstag)
Values (124563,'Müller','Thomas',11.10.1990)
```

Hier wird mittels *INSERT INTO* auf die Tabelle hingewiesen, bei der die angegebenen Spalten befüllt werden sollen. Schließlich werden die Informationen durch das Schlüsselwort *Values* gespeichert. Ebenfalls besteht die Möglichkeit; nur bestimmte notwendige Spalten anzugeben. Dementsprechend werden Informationen nur in den angeführten Spalten der Tabelle aufbewahrt. Mit Hilfe des *DELETE* Statements können Datensätze aus den Tupeln gelöscht werden.

```
DELETE
FROM Mitarbeiter
WHERE Mitarbeiter.Name = ‘Müller‘
```

Durch das *WHERE*-Kommando wird auf ein bestimmtes Tupel hingewiesen. Somit werden alle Tupel mit dem Namen Müller aus der Spalte Name der Tabelle *Mitarbeiter* gelöscht. Ebenfalls kann der Wert einer Spalte durch das Statement *UPDATE* aktualisiert beziehungsweise verändert werden.

```
UPDATE Mitarbeiter
SET Mitarbeiter.Name = ‘Meier‘
WHERE Mitarbeiter.Mitarbeiter -ID = 124563
```

Hier erfolgt eine Namensänderung des Mitarbeiters mit der angegebenen ID. Mit Hilfe von *SET* wird der Name von Müller auf Meier geändert und durch *WHERE* die ID des Mitarbeiters angegeben. Somit ist ersichtlich, bei welchem Mitarbeiter die Änderung vorgenommen werden soll. Hier wird die Stärke von SQL zweifellos ersichtlich. Zugleich ist die Abfrageerstellung ein weiterer Vorteil, wobei durch simple Statements effiziente Ergebnisse erzielt werden. Mit der folgenden Abfrage werden alle Mitarbeiter ausgegeben die bei einem Department arbeiten, deren Nummer kleiner 30 ist.

```
SELECT *
FROM Mitarbeiter M
WHERE M.Department-ID < ‘30‘
```

Mittels *SELECT* wird die Auswahl der Spalten einer Tabelle angegeben. Durch das Stern-Symbol wird ausgedrückt, dass alle Spalten der folgenden Tabelle *Mitarbeiter* ausgegeben werden sollen. Demnach sollen nur die Zeilen ausgewählt werden, deren Department-ID kleiner 30 ist. Gleichermaßen gibt es viele Möglichkeiten um bestimmte Daten abzufragen. Der folgende Befehl enthält die wichtigsten Elemente einer SQL-Abfrage [24].

```
SELECT [DISTINCT | ALL]
        <Spaltenliste> | *
FROM    <Tabellenliste>
[WHERE          <Bedingungsliste> ]
[GROUP BY       <Spaltenliste>    ]
[HAVING         <Bedingungsliste> ]
[UNION          <Select-Ausdruck>]
[ORDER BY       <Spaltenliste>    ]
;
```

Die ersten drei Zeilen sind im SQL-Standard vorgegeben. Die restlichen, unter Klammern stehenden Zeilen sind je nach Abfrage optional anzugeben. Wie bisher

erwähnt, wird die *WHERE*-Klausel dazu genutzt um Einschränkungen fest zu setzen. Durch *GROUP BY* können Daten gruppiert werden. Die *HAVING*-Klausel gleicht von der Funktionsweise der *WHERE*-Klausel, wobei *WHERE* im Zusammenhang mit aggregierten Funktionen nicht verwendet werden kann. Da diese Klausel sehr umfangreich ist, tritt sie nicht oft in Verwendung. Mit *UNION* können mehrere Abfragen verbunden werden. Letztlich werden durch *ORDER BY* Sortierungen ausgeführt.

Vorteile

Relationale Datenbankmodelle ermöglichen eine redundanzfreie und konsistente Implementierung sowie eine mengenorientierte Datenverarbeitung. Es ist ein einfaches Datenmodell, das eine garantierte Datenkonsistenz bietet. Im Vergleich zu hierarchischen und netzwerkorientierten Datenbankmodellen müssen keine bekannten Einstiegspunkte definiert werden. Eine weitere Stärke ist die Verknüpfung von Inhalten. Objekte werden in Einzelteile zerlegt und mengenorientiert verarbeitet. Demzufolge sind komplexe Abfragen und Operationen auf große Datenmengen möglich. Ebenso ist ein hohes Maß an Datenunabhängigkeit vorhanden. Somit ist keine Navigation in der Datenbank durch den Benutzer erforderlich. Die Sicht auf die Daten ist abstrakter und es sind weniger Kenntnisse über die physische Datenorganisation erforderlich. Durch die einheitlich standardisierte Datenbanksprache SQL können einfache Abfragen erstellt werden.

Nachteile

Angesichts der vielen Vorteile sind zugleich auch einige Schwächen auffindbar. Beispielsweise werden Beziehungstypen wie Entity-Typen in Relationen angelegt. Diese Vereinfachung bedeutet, dass semantische Beziehungen zwischen Entitäten im relationalen Modell nicht explizit ausgedrückt werden. Die semantische Lücke ist die dementsprechende Definition dazu. Generalisierungsbeziehungen lassen sich ebenfalls nicht ausdrücken, da es nicht möglich ist den Tupel wiederum Sub-Tupel zuzuordnen. Gleichermaßen problematisch ist die inhaltliche Zerlegung der Daten und die Performanceoptimierung bei der Suche nach komplexen Zusammenhängen. Folglich müssen gespeicherte Daten, die zusammen benutzt werden sollen, getrennt voneinander auf der Anwendungsebene in Form von Joins wieder zusammengeführt werden.

Im Vergleich zum relationalen Datenbankmodell bestehen weitere alternative Konzepte, wobei sich diese Modelle nicht durchsetzen konnten. Trotzdem werden sie für bestimmte Applikationen verwendet. Das hierarchische Datenbankmodell ist beispielsweise eines der ältesten Modelle. Hier wird die reale Welt in einer hierarchischen Baumstruktur abgebildet. Somit hat jeder Datensatz genau einen Vorgänger bis auf die Wurzel der Baumstruktur. Der darauf basierende Gedanke wird auch als Eltern-Kind-Beziehung definiert. Dementsprechend werden Datensätze in einer Reihe gespeichert und mit verschiedenen Feldern verknüpft. Der Nachteil ist, dass Verknüpfungen nur in einer Baumstruktur abgebildet werden können. Deshalb sind Verbindungen zwischen mehreren Bäumen oder über mehrere Ebenen eines Baumes nicht möglich. Innerhalb des Datenbankmodells sind zwei Strukturelemente vorhanden mit denen minimalistische Bedingungen ausgeführt werden können. Ein Datensatztyp muss das Wurzelelement darstellen, wobei jeder andere genau einmal als Kind auftreten kann. Weiterhin wird ein Datensatztyp, der nicht als Elternteil auftritt auch als Blatt bezeichnet. Angesichts dessen lassen sich durch diese Baumstruktur nur 1:1 und 1:n-Beziehungen darstellen. Man nehme das vorher gebrachte Beispiel mit den Tabellen *Mitarbeiter* und *Projekt*. Die Tabelle *Mitarbeiter* besteht weiterhin aus den Spalten *Mitarbeiter-Id, Name, Geburtstag und Department-ID.* Der Unterschied in diesem Beispiel ist nun die Repräsentation der Tabellen. Die folgenden Grafiken definieren nochmals die Tabelle *Mitarbeiter* und die Tabelle *Projekt*.

Mitarbeiter-ID	Nachname	Vorname	Geburtstag
124563	Müller	Thomas	11.10.1990
143874	Bauer	Martin	02.05.1992

Tabelle 2: Instanzen der Tabelle Mitarbeiter

Projekt-ID	Name	Kunde
25634	Projekt X	IBM
41568	Projekt Y	Microsoft

Tabelle 3: Instanzen der Tabelle Projekt

Die folgende Abbildung, basiert auf der veröffentlichten Publikation der Compendio Bildungsmedien [25].

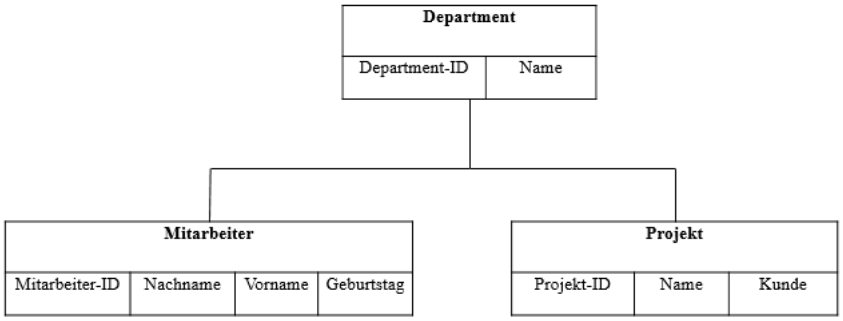


Fig. 1: Hierarchisches Datenbankmodell [25]

In diesem Beispiel ist die Eltern-Kind-Beziehung eines hierarchischen Modells abgebildet. Dabei beinhaltet das Department alle Mitarbeiter die darin arbeiten und alle Projekte die von diesem Department abgewickelt werden. Letztlich ist das hierarchische Datenbankmodell weitgehend von anderen Modellen abgelöst. Dennoch wird es in Applikationen wie zum Beispiel in IBM's IMS Datenbankmanagementsystem eingesetzt [1].

Das Netzwerkdatenbankmodell ist im Gegensatz zum hierarchischen Datenbankmodell verallgemeinert. Es fordert keine strenge Hierarchie und ermöglicht deshalb auch m:n-Beziehungen. Dieses Konzept besteht aus Datensätzen, die aus verschiedenen Feldern gebildet werden, die ebenfalls einen Namen und einen Wert beinhalten. Ein Datensatz kann beliebig eine Person, ein Ereignis oder ein Objekt beschreiben, welches auch als Event bezeichnet wird. Daraus folgen meistens unterschiedliche Suchwege, um zu einem bestimmten Datensatz zu gelangen. Der Nachteil daran ist, wenn der Entwickler nur einen Lösungsweg benutzen will. Genauso problematisch ist die verringerte Übersicht die durch das ständig wachsende Modell entsteht. Zu beachten ist, dass ein Datensatz eine innere Struktur beinhaltet. Beispielweise können Felder zu Gruppenfeldern zusammengefasst werden. Wobei Gruppenfelder nicht nur aus Einzelfeldern, sondern ebenfalls aus Gruppenfeldern bestehen. Anhand des folgenden Beispiels wird ein Netzwerkdatenbankmodell dargestellt.

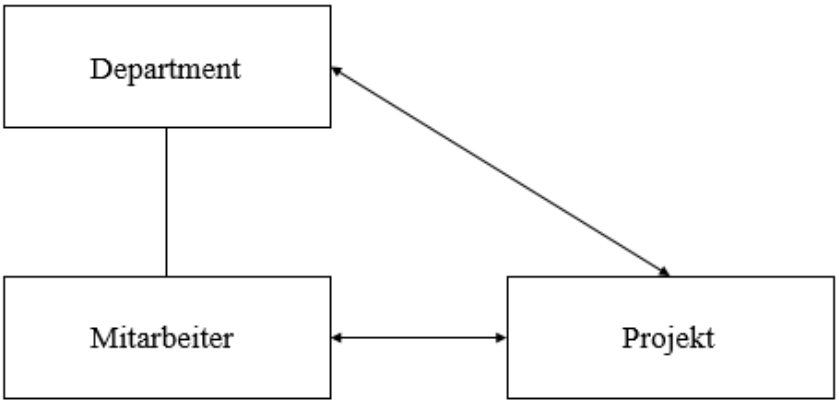


Fig. 2: Beispielhafte Netzwerkdatenbank [25]

In diesem Szenario wird die Beziehung zwischen einem Mitarbeiter, dem Projekt und der Abteilung dargestellt. Hier werden die m:n-Beziehungen zwischen dem Projekt und dem Mitarbeiter als auch der Abteilung abgebildet. Obwohl das Netzwerkdatenbankmodell nicht weit verbreitet ist, wird es dennoch in Applikationen wie zum Beispiel im IDMS (Integrated Database Management System) verwendet [1].

In objektorientierten Datenbanken werden die Daten im Gegensatz zur relationalen Datenbank als Objekte verwaltet. Davon ist jedes eine eigenständige Einheit, die Daten und Verhalten beinhaltet. Weiters werden die Daten als Attributwerte und das Verhalten als Methode eines Objektes referenziert. Deshalb werden Objekte mit identischer Struktur und gleichem Verhalten zu einer Klasse zusammengefügt. Innerhalb eines Objektes kann nur durch definierte Methoden auf Daten zugegriffen werden. Dementsprechend ist ein direkter Zugriff auf Daten von außerhalb nicht möglich. Ein objektorientiertes Datenbanksystem beinhaltet acht Regeln der Objektorientierung und fünf Grundsätze der Datenhaltung und -nutzung. Nach einer umfangreichen Literaturrecherche wurden die folgenden Kriterien für objektorientierte Datenbanksysteme aufgestellt [18].

1. **Komplexe Objekte**: Ein wichtiger Faktor der Objektorientierten Datenbanksysteme (OODBS) ist die Unterstützung komplexer Objekte, deren Attribute wiederum Objekte sein können. Attribute können einfache Datentypen wie zum Beispiel Integer, String oder Charakter repräsentieren, aber auch Listen, Mengen oder ganze Objekte.

2. **Objektidentität**: Unabhängig von den Werten der Attribute, muss jedes Objekt eine eigene Identität beinhalten. Dadurch muss die Identifikation eines Objektes eindeutig und unveränderbar sein.

3. **Einkapselung**: Die Datenkapselung setzt voraus, dass Implementierungsdetails nach außen nicht repräsentiert werden. Deshalb kann auf die Werte der Attribute nur durch Methoden zugegriffen werden.

4. **Typen und Klassen**: Objekte, die eine äquivalente Struktur und ein gleiches Verhalten aufweisen, werden zu Klassen zusammengefasst.

5. **Typen und Klassenhierarchie**: Sowohl Typen als auch Klassen können hierarchisch oder im Netzwerk aufgebaut sein.

6. **Overriding**: Definiert das Überladen von Methoden.

7. **Datenbank Programmiersprache**

8. **Erweiterbarkeit:** Es besteht die Möglichkeit einer Erweiterung von neuen Typen und Klassen eines Systems. Außerdem kann aus der außenstehenden Sicht keine Unterscheidung zwischen systemeigenen und benutzerdefinierten Klassen erkannt werden. Somit können auf allen Ebenen Erweiterungen von Typen, Funktionen und Speicherstruktur ermöglicht werden.

9. **Mehrfach-Vererbung**: Hier kann eine Subklasse in einer Klassenhierarchie Eigenschaften von übergeordneten Klassen erben. Weiterhin darf die Unterklasse eigene Eigenschaften hinzufügen.

10. **Polymorphismus**: Methodennamen können auf Objekte unterschiedlicher Klassen vielseitig eingesetzt werden. Außerdem können Subklassen beispielsweise gleichnamige Methoden der Oberklassen überschreiben.

11. **Vollständigkeit**: Eine bestimmte Operation kann durch die Sprache in der Datenbank ausgedrückt werden.

12. **Persistenz**: Ein Datenzustand muss über die Laufzeit eines Programmes fortbestehen. Dabei erfolgt die Übertragung auf den externen Speicher ohne explizite Kommandos.

13. **Ad-hoc-Abfragemöglichkeit**: Das OODBS stellt eine effektive Abfragesprache zur Auswertung von Daten zur Verfügung.

Das folgende Beispiel veranschaulicht ein objektorientiertes Datenbankmodell. Dazu wird ein Objekt *Mitarbeiter* erstellt, das den folgenden schematischen Aufbau beinhaltet:

[Mitarbeiter-ID]
124563
[Nachname]
Müller
[Vorname]
Thomas
[Geburtstag]
11.10.1990

In diesem Objekt befinden sich Zugriffsmethoden wie *getID*, *getNachname*, *getVorname* und *getGeburtsdatum*. Die Ausführung der Methoden anhand einer Objektsprache liefert folgende Werte.

Mitarbeiter → getID() →124563
Mitarbeiter → getNachname() → Müller
Mitarbeiter → getVorname() → Thomas
Mitarbeiter → getGeburtstdatum() → 11.10.1990

Vorteile

Die Objektdatenbanken ermöglichen eine Kooperation mit relationalen Datenbanken. Beispielsweise können entwickelte objektorientierte Anwendungen mit relationalen Datenbanken zusammenarbeiten, obwohl sie sich in einigen Konzepten widersprechen. Als Lösung werden objektrelationale Abbildungen verwendet, die als Softwarekomponenten zwischen einer relationalen Datenbank und objektorientierter Software vermitteln. Jedoch wird durch den Einsatz einer objektorientierten Datenbank diese Vermittlung überflüssig, weil die Anwendung somit direkt mit der Datenbank kommunizieren kann. Weiterhin wird das Problem der Objektidentität und der Schlüsselvergabe auf unterschiedliche Art und Weise gelöst. In relationalen Datenbanken wird ein Schlüssel vergeben der einen Datensatz identifiziert. Im Gegensatz dazu wird in objektorientierten Datenbanken automatisch eine Objekt-ID erzeugt, wobei die Verwaltung vom System übernommen wird.

Nachteile

Mittlerweile ist die Verbreitung des Modells gering, weshalb nur begrenzt unterstützende Tools vorhanden sind. Manipulationsoperationen von Objekten können somit zu massiven Performanceproblemen führen.

5.2 Datenbankmanagementsysteme

Mittlerweile steigt die Ansammlung von Daten rapide. Dementsprechend wird ein Hilfsmittel benötigt, das innerhalb einer bestimmten Laufzeit relevante Informationen filtert und gleichzeitig verarbeiten kann. In Erwägung gezogen wird nur ein Datenbankmanagementsystem, abgekürzt DBMS, welches Daten aufrecht zu erhalten vermag und dadurch für den Benutzer von Vorteil ist. Obwohl ein File System als Speicher verwendet werden kann, ist es durchaus problematisch, wenn eine Datenmenge von über 500 GB konsistent aufrechterhalten werden muss. Bei Ausführungen von Abfragen muss ebenfalls mit einer großen Laufzeit gerechnet werden, da beispielsweise mit 32-bit Rechnern auf maximal 4GB des Hauptspeichers zugegriffen werden kann. Weiterhin muss der Speicher Schutz bieten und persistent sein. Passwörter sind für den Schutz von File-Systemen meistens unzureichend, weshalb mittels dem DBMS eine Software erstellt wurde, die alle folgenden Kriterien erfüllt. Es bietet viele Vorteile, um eine große Datensammlung robust und effizient zu managen [1].

Data Independence: Das DBMS beinhaltet eine abstrakte Sicht der Daten, das bestimmte Details ausblendet.

Efficient Data Access: Zum Speichern und Abrufen der Daten werden komplexe Techniken angewendet, die eine schnelle Laufzeit ermöglichen.

Data Integrity and Security: Bei vielen Datenzugriffen können Constraints erstellt werden, die als Einschränkung dienen. Weiterhin sind Access Controls eine Möglichkeit, einen Benutzer auf bestimmte Daten einzuschränken. Letztendlich liegt hier der Fokus auf Benutzereinschränkungen.

Data Administration: Viele Benutzer greifen in einer Vielzahl an Situationen auf dieselben Daten zu. Jedoch ist die Nutzung sehr unterschiedlich. Mit Hilfe der Administration wird eine bessere Übersicht der Daten ermöglicht. Somit ist eine bessere Organisation und Repräsentation möglich.

Concurrent Access and Crash Recovery: Das DBMS erstellt einen Zeitplan für mehrmalige Zugriffe, wobei dem Benutzer vermittelt wird, dass zeitgleich nur einer auf einen Datensatz zugreift. Ebenfalls werden viele Maßnahmen ergriffen, um den Benutzer vor Fehlern zu schützen.

Reduced Application Development Time: Den unterstützten Funktionsumfang weisen andere Applikationen in ähnlicher Form auf. Nach den aufgezählten Vorteilen stellt sich die Frage, ob es Gründe gegen die Verwendung eines DBMS gibt. Das DBMS ist trotz seines komplexen Funktionsumfangs zur Optimierung von Auslastungen und Prozessen der Verarbeitung, für Applikationen mit strikten Echtzeit Einschränkungen und kritischen Operationen nicht geeignet. Daher muss für diesen Zweck ein effizienter Code entwickelt werden, der dies gewährleisten kann.

Gegenwärtig wird eine erhebliche Menge an Daten verarbeitet. Somit können verschiedene Einzelheiten wie zum Beispiel Entitäten oder auch Beziehungen ausgedrückt werden. Das Data Model ist ein High-level Konstrukt mit einer Ansammlung von Eigenschaften, das mehrere Low-level Speicher beinhaltet. Wobei zu beachten ist, dass in dem zuletzt genannten Speicher die Details ausgeblendet werden. Der Benutzer kann demnach seine Daten nach bestimmten Konditionen in das Data Model einpflegen. Nichtsdestotrotz ist der Einblick auf das Konstrukt für außenstehende Benutzer als kritisch zu bewerten. Das semantische Data Model ermöglicht es, durch eine abstrakte Art und Weise einen besseren Überblick der Daten und des Aufbaus zu verschaffen. Gleichermaßen vorteilhaft ist die Darstellung von realen Szenarien. Somit wird eine Darstellung über die Wechselverhältnisse der Daten ermöglicht, obwohl eine direkte Unterstützung von DBMS nicht gewährleistet ist. Folglich ist das semantische Data Model der Ausgangspunkt, für die Beschreibung realer Szenarien. Infolgedessen wird es zu einem Data Model umgewandelt, das auch von DBMS unterstützt wird. Eines der am weitverbreitetsten High-level Modelle ist das Entity-Relationship-Modell (ER-Modell). Es ist im Stande die Entitäten, Attribute und deren Beziehungen zueinander grafisch darzustellen. Dementsprechend ermöglicht es einem Benutzer die Umwandlung einer formlosen Beschreibung, was man sich von einer Datenbank erwartet, zu einer Beschreibung die im DBMS implementiert werden kann. Das Entity-Relationship-Modell findet häufig im Zusammenhang mit dem Design der Datenbank Verwendung, weil es die Basis der Implementierung bildet. Die Umsetzung der realen Daten in ein Datenbankschema erfolgt in mehreren Schritten. Zunächst werden durch Abstraktion Entitäten in Entitätstypen zusammengefasst. Beispielsweise werden die Personen Müller und Bauer als Entitäten in den Entitätstypen Mitarbeiter zusammengefügt. Infolgedessen werden die Beziehungen zwischen zwei Objekten zu einem Beziehungstyp zusammengefasst. Die Grundlage liegt darin, dass die Mitarbeiter Müller und Bauer in derselben Abteilung arbeiten. Somit ergibt sich der Beziehungstyp Arbeitsort. Im nächsten Schritt werden die Kardinalitäten bestimmt. Diese sagen aus, dass eine Abteilung mehrere Mitarbeiter hat, wobei ein Mitarbeiter aber nur in einer Abteilung beherbergt werden kann. Weiters werden detaillierte Beschreibungen der Attribute bestimmt. Danach muss ein Schlüsselattribut eines Entitätstypen definiert werden. Im weiteren Verlauf werden notwendige Beziehungstypen wie Pflichtbeziehungen und Fremdschlüssel festgelegt. Darauf folgt die Generierung des Schemas einer relationalen Datenbank mit allen Tabellen und zugehörigen Definitionen.

Für die Unterstützung der Datenbankentwicklung ist eine Folge von Schritten erforderlich, um eine Menge von Anforderungen in ein logisches und physikalisches Schema zu transformieren. Dadurch erfolgt eine Aufteilung auf drei Abstraktionsebenen. Sie umfassen externe, konzeptuelle und physikalische Schemas. Die Motivation der Aufteilung ist, die Isolation von Details zu ermöglichen und Änderung tieferer Ebenen unabhängig voneinander zu gestalten. Das Ziel für die Nutzung dieser Form der Abstraktion ist das Sicherstellen einer Datenunabhängigkeit [1][17].

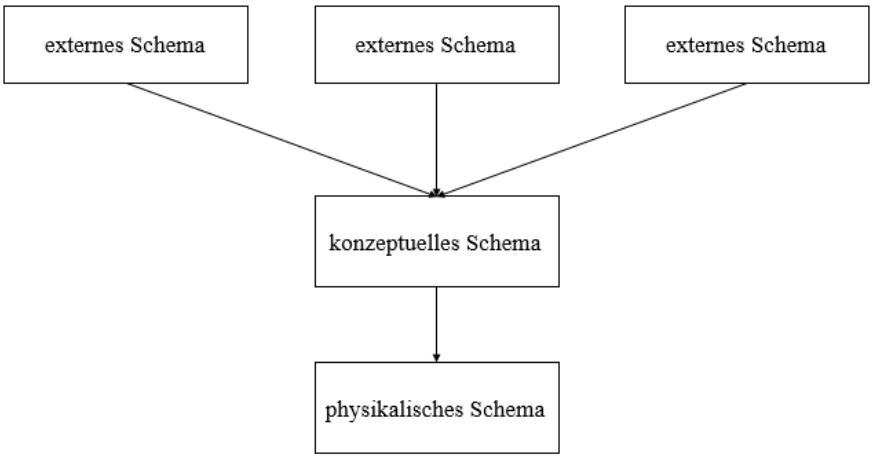


Fig. 3: Abstraktionsebenen einer DBMS [17]

Das externe Schema definiert die Sicht der Benutzer auf die Datenbank. Weiterhin beinhaltet es eine Anzahl an externen Views und basiert auf der wahrgenommenen realen Welt des Benutzers.

Das konzeptuelle Schema beschreibt alle vorhandenen Daten sowie Beziehungen die in einer Datenbank gespeichert sind. Somit wird eine Beschreibung zwischen den Beziehungen der Datenbankstruktur auf höherer Ebene sichergestellt, wobei das Entity-Relationship-Modell hierfür hilfreich ist.

Letztlich ist das physikalische Schema eine physische Repräsentation der Datenbank mit den dazugehörigen Abstraktions-Leveln. Es beschreibt anhand der Run-Time Performance, Speicherauslastung, Datenorganisation und Verschlüsselung wie die Daten gespeichert sind. Außerdem beinhaltet es Konzepte, die den Speicherablauf von Daten auf einem Computer bis ins Detail beschreiben.

Das Entity-Relationship-Modell zeigt seine Stärken in der Darstellung der Semantik und Struktur. Im bisherigen Verlauf des Kapitels wurden Details über Objekte, Beziehungen und Kardinalitäten erläutert. Dementsprechend werden nun praktische Ansätze analysiert. Die Grundlage der Entity-Relationship-Modelle ist die Klassifikation von Objekten, deren Beziehungen und ihrer Attribute. In diesem Modell können die Entitäten Mitarbeiter und Projekt als Raute modelliert werden, die dadurch ein eindeutig identifizierbares und reales Objekt repräsentieren. Außerdem legt die Raute die Beziehung zwischen den Entitäten fest. Gleichermaßen werden die Kardinalitäten vergeben. Somit kann zum Beispiel ein einziger Mitarbeiter mehrere Projekte leiten, wobei ein Projekt genau von einem Mitarbeiter geleitet werden kann. Ebenso werden die Attribute anhand einer Ellipse abgebildet. Hier werden Informationen über eine Entität dargestellt. Beispielsweise kann der Mitarbeiter einen Namen und eine ID besitzen und gleichermaßen das Projekt.

Weiterhin können mittels Entity-Relationship-Modelle Generalisierungen ausgedrückt werden. Die Spezialisierung beschreibt einen Entitätstypen als Teilmenge eines anderen Entitätstypen, wobei je zwei Entitäten mit den Namen interner und externer Mitarbeiter belegt sind. Diese können zu einem Entitätstypen Mitarbeiter zusammengefasst werden, weil die bestehenden Attribute der zwei Entitäten identisch sind. Somit beinhaltet die übergeordnete Entität die identifizierbaren Attribute, während die untergeordneten weitere zusätzliche Informationen aufweisen. Ebenso wichtig in der Praxis sind Aggregationen. Darunter versteht man die Zusammenfassung von mehreren Objekten zu einem Objekt. Der übergeordnete Entitätstyp wird als Aggregat und der untergeordnete als Komponente bezeichnet. Dadurch wird eine Unterstützung für komplex strukturierte Objekte ermöglicht.

Die Beschreibungen und Darstellungen besonderer Sachverhalte in Entity-Relationship-Modellen sind sehr illustrativ. Entitätstypen können als stark oder schwach definiert werden. Zur Identifikation eines starken Typen muss ein eindeutiges Attribut der gleichen Type vorhanden sein, während ein schwacher Typ durch Attribute einer starken Type definiert wird. Die Kardinalität bestimmt für jeden beteiligten Entitätstypen, an wie vielen Beziehungen er beteiligt sein muss oder kann. Außerdem ermöglicht die reflexive Beziehung, eine Beziehung innerhalb des gleichen Entitätstypen. Eine beispielhafte Beziehung dafür wäre, wenn eine Organisationseinheit aus anderen Organisationseinheiten gegliedert ist. Weiterhin kann die Komplexität eines Beziehungstyps bestimmt und die Anzahl der Entitätstypen definiert werden, die an einem Beziehungstyp beteiligt sind. Im Allgemeinen wird der zweite Grad, oder auch binärer Beziehungstyp genannt, verwendet. Der dritte Grad oder auch ein höherer tritt selten in Verwendung. Zusätzlich können Beziehungstypen ebenfalls Attribute beinhalten.

5.3 Allgemeine Sicherheitsanforderungen in Datenbanken

Dadurch, dass viele kritische Informationen in Datenbanken gespeichert sind, müssen Schutzmaßnahmen getroffen werden. Wichtige Sicherheitsziele die beispielsweise geschützt werden müssen sind die Vertraulichkeit, Integrität und Verfügbarkeit.

Bezogen auf dieses Unterkapitel wurde eine große Anzahl an Informationen aus dem IT-Grundschutzkatalog des Bundesamtes für Sicherheit in der Informationstechnik entnommen [26]. Weiterhin ist hinzuzufügen, dass alle beschriebenen Informationen aus den gekennzeichneten Publikationen interpretiert wurden.

Datenmissbrauch durch Schadprogramme ist ein häufiges Risiko. Weitere Schwachstellen auf die geachtet werden muss sind Überlastungs-, Leistungseinschränkungs- und Kapazitätsprobleme. Genauso kritisch sind physische Schäden, Konstruktionsschwachstellen sowie Datenverfälschungen. Zur Minimierung der Risiken, müssen Kontrollmechanismen verwendet werden. Diese Mechanismen sind die Zugriffskontrolle, Authentifizierung, Verschlüsselung, Konsistenzkontrolle und die Datensicherung.

Mit Hilfe der Zugriffskontrolle kann der Zugriff und die Steuerung von Ressourcen überwacht werden. Wobei die Kontrolle wiederum in drei Unterkategorien aufgeteilt wird. Die administrative Kontrolle beschreibt den Umgang bei Zugriffen auf Informationen. Sie umfasst die Sicherheitsvorschiften, die Prozeduren sowie die Kontrollstruktur einer Organisation. Weiters werden alle physischen Maßnahmen durch die physische Kontrolle geleitet. Dies betrifft zum Beispiel den Aufbau und die Architektur von Gebäuden oder die Beschaffung von Schutzdiensten. Die dritte Unterkategorie ist ausschließlich für die technische Kontrolle zuständig. Hier erfolgt die Einschränkung der Zugriffe auf Software und Hardware, zum Beispiel auf Komponenten eines Betriebssystems, Applikationen oder Netzwerkgeräten. Die Grundmodelle dieses Konzeptes sind beispielsweise das Discretionary Access Control (DAC) oder auch das Role Based Access Control (RBAC) Modell [28][29]. Das DAC regelt den Zugriff auf die Ressourcen. Dementsprechend werden die Zugriffsrechte für jeden Benutzer bestimmt. Im Gegensatz dazu werden im RBAC Modell Benutzer in Gruppen aufgeteilt und mit unterschiedlichen Rollen versehen. Dadurch wird das Konzept des DAC Modells geschwächt, weil nicht jeder Benutzer eine bestimmte Rolle trägt. Der Vorteil des RBAC Modell ist, dass einem Benutzer mehrere Rollen zugeteilt werden können. Der Einsatz erfolgt in Systemen wie im Microsoft Active Directory, Microsoft SQL Server oder auch in Solaris oder Oracle RDBMS. Die wohl am stärksten verbreitete Methode sind die Access Control Lists (ACL) [30]. Durch die Aufbietung des Verfahrens werden Zugriffe auf Daten und Funktionen eingegrenzt. Im Gegensatz zu einfachen Zugriffsrechten werden die Einstellungen in ACLs detaillierter definiert. Demnach können bei unterschiedlichen Systemen mehrere Benutzer und Gruppen unterschiedliche Rechte für eine Datei erhalten.

Die Authentifizierung ist bei Anfragen von sensiblen Daten ein wichtiges Kriterium. Bei Benutzeranfragen muss zunächst eine Authentisierung am Server erfolgen. Hier wird überprüft, ob eine Zugangsberechtigung vorhanden ist. Danach erfolgt als

abschließende Bestätigung eine Autorisierung. Letztlich wird das Verfahren als Triple-A-System definiert [63]. Die verbreitetsten Standardprotokolle sind unter anderem das TACACS+ oder das RADIUS Protokoll. Das Terminal Access Controller Access Control System (TACACS+) agiert als Client-Server-Kommunikation zwischen AAA-Servern und einem Network Access Server (NAS) [33]. Somit ermöglicht es einem Benutzer durch eine Authentifizierung die Verbindungsherstellung mit einem NAS über das Internet oder dem Intranet. Es wird oft in Netzwerkumgebungen mit Routern oder Switches verwendet. Weiterhin dient es dem zentralen Benutzermanagement der Netzwerkadministratoren. Im Vergleich zum UDP-basierten RADIUS Protokoll verwendet TACACS+ das verbindungsorientierte TCP Transportprotokoll. Außerdem wird die ganze Kommunikation verschlüsselt durchgeführt. Das Remote Authentication Dial-In User Service (RADIUS) ist gleichermaßen ein weit verbreiteter Standard und wird bei zentralen Authentifizierungen im Modem, ISDN, VPN oder DSL eingesetzt [31][32]. Es arbeitet nach dem Client-Server Prinzip und besteht somit aus einem Authentifizierungsserver an den sich Services von Clients wenden. Der Server übernimmt die Authentifizierungen der Services, die beispielsweise den Benutzernamen und das Passwort überprüfen. Dazu werden weitere Parameter für die Verbindung bereitgestellt. Der Vorteil des RADIUS Protokolls liegt in der einmaligen Registrierung des Benutzers der in Netzwerken jederzeit verfügbar ist.

Bei der Übertragung von Datensätzen ist die Verschlüsselung ausschlaggebend. Hierbei existieren unterschiedliche Verschlüsselungsarten, die ein System vor Risiken mehr oder weniger schützen können. Gegenwärtige Verschlüsselungsarten sind die symmetrischen und asymmetrischen Verfahren. Der bekannteste symmetrische Verschlüsselungsalgorithmus ist der Data Encryption Standard (DES) [34]. Bei diesem Algorithmus wird zur Ver- und Entschlüsselung der gleiche Schlüssel verwendet. Weiters funktioniert DES als Blockchiffre, wobei jeder Block mit Hilfe des Schlüssels einzeln chiffriert wird. Zeitgleich werden die Daten in 16 Iterationen durch Substitutionen und Transpositionen nach dem Feistel-Schema vertauscht. Somit wird ein 64 Bit Block Klartext in einem Chiffretext umgewandelt. Ebenso besitzt die Schlüssellänge 64 Bit, wobei 8 Bit für einen Paritäts-Check zur Verfügung gestellt werden. Letztlich erfolgt die Entschlüsselung nach demselben Verschlüsselungsverfahren. Durch die schwache beziehungsweise halbschwache Schlüssellänge ist das DES in Kritik geraten. Beispielsweise gelingt es einem Brute-Force-Angriff systematisch alle Schlüssel zu testen, obwohl 72 Billiarden mögliche Schlüssel vorhanden sind [64].

Das Advanced Encryption Standard (AES) wurde zur Ablöse des Data Encryption Standard (DES) entwickelt [35]. Die Arbeitsweise des AES bezieht sich ebenfalls auf das Substitutions- und Permutationsverfahren die bei Blockchiffren angewendet werden. Jedoch können die Block- und Schlüssellängen voneinander unabhängige Werte besitzen. Der Block besitzt allerdings eine festgesetzte Größe von 128 Bit, während die Länge des Schlüssels entweder 128, 192 oder 256 Bit beinhalten kann. Weiters wird jeder Block in eine zweidimensionale Tabelle mit vier Zellen eingetragen, wobei jede Zelle ein Byte groß ist und die Anzahl der Spalten je nach Blockgrößen variiert. Danach wird jeder Block einer bestimmten Transformation unterzogen. Hier werden verschiedene Teile des Schlüssels nacheinander auf den Klartextblock

angewendet. Dabei ist die Anzahl der Runden von der Schlüssellänge und Blockgröße abhängig und kann dementsprechend abweichen. Gleichermaßen wird die Entschlüsselung rückwärts abgearbeitet. Obwohl der AES-Algorithmus den schwächeren DES-Algorithmus ersetzen soll, traten im Laufe der Jahre einige Kritikpunkte zum Vorschein. Zum Beispiel gelang es im Jahr 2011 einer Gruppe von Kryptologen den ersten erfolgreichen Angriff zu entwickeln. Der Biclique-Angriff ist für die praktische Sicherheit nicht relevant, zeigt aber das der AES-Algorithmus Schwächen vorweisen kann [36].

Angesichts dieser Problematik ist die Unzugänglichkeit des Schlüssels eine Stärke des asymmetrischen Verfahrens, weil hier die Schlüssel nicht übertragen werden müssen. Ein Benutzer erzeugt ein Schlüsselpaar mit Hilfe eines geheimen, privaten und eines öffentlichen Schlüssels. Der bekannteste Algorithmus ist das RSA-Verfahren. Anhand eines kurzen Beispiels wird das Verfahren näher erläutert [37][38]. Die Sicherheit des Verfahrens basiert auf dem Problem, eine große ganze Zahl in ihre Primfaktoren zu zerlegen. Dementsprechend darf die Schlüssellänge nicht zu klein gewählt werden. Beide Schlüssel werden erzeugt, indem zufällig zwei verschieden große Primzahlen p und q gewählt werden. Darauf basierend wird das Produkt berechnet. Danach wird ein zufälliger Wert e ermittelt, der kleiner m und teilerfremd zu $(p-1) * (q-1)$ ist. Im nächsten Schritt muss die modulare Inverse d berechnet werden. Dabei wird ein Wert e gesucht, welcher kleiner m ist, sodass der größte gemeinsame Teiler von e und $(p-1)*(q-1)$ gleich 1 ist. Letztlich gilt als öffentlicher Schlüssel die Kombination von e und m und als privater Schlüssel die Vereinigung von d und m. Die Stärke des Verfahrens liegt in der Erstellung des Schlüssels. Das RSA Verfahren verwendet bei Ver- und Entschlüsselung nur Zahlen, wodurch der Klartext mit einem bekannten Alphabet in eine Zahlenfolge übersetzt werden muss. Da das RSA-Verfahren im Vergleich zu AES langsamer ist, wird in der Praxis der RSA nur zum Austausch eines Schlüssels benutzt, während die Verschlüsselung der Daten durch den Einsatz des symmetrischen Verfahrens erfolgt. Somit sind ein einfacher Schlüsselaustausch und eine effiziente Verschlüsselung möglich.

Die Konsistenzkontrolle übernimmt die Garantie der Korrektheit von Datenbankinhalten und der korrekten Ausführung von Änderungen [39]. Dabei können inkonsistente Datenbanken Fehler erzeugen, wenn die darüber liegende Anwendungsschicht auftretende Probleme nicht identifiziert. In relationalen Datenbanken wird ein korrekter Zustand durch benutzerdefinierte Integritätsbedingungen definiert. Infolgedessen werden die Anwendungen von Systemen während der Laufzeit kontrolliert. Weiters sind verschiedene Arten der Integritätsbestimmungen vorhanden [39].

Entity integrity: Hier wird das Konzept des Primärschlüssels verwendet. Bei dieser Integritätsbedingung muss jede Tabelle einen Primärschlüssel besitzen. Außerdem sollen die ausgewählten Primary Key Spalten eindeutig und nicht Null sein.

Referential integrity: In diesem Fall erfolgt der Einsatz des Fremdschlüsselkonzeptes. Somit muss ein Objekt mit einem Fremdschlüssel existieren oder der Wert muss Null sein.

Domain integrity: Hier muss der Wert jedes Attributes in einem bestimmten Wertebereich liegen.

Ein grundlegender Faktor in Datenbanken ist die Sicherung in einem konsistenten Zustand, wobei zwischen einem Hot Backup oder einem Cold Backup differenziert wird [40]. Bei einem Hot Backup verläuft die Sicherung eines Systems während des laufenden Betriebes. Dementsprechend können die Daten aktuell gehalten werden. Trotzdem müssen kontinuierliche Kontrollen durchgeführt werden, um Daten und Verknüpfungen nicht zu verlieren. Der Vorteil dieser Sicherung ist, dass bei einem Ausfall durch ein Online-Datensicherungssystem in kürzester Zeit der normale Servicebetrieb wiederhergestellt werden kann. Im Gegenzug ist das Cold Backup die sichere Variante. Dazu wird der Server heruntergefahren, was üblicherweise außerhalb der Arbeitszeiten stattfindet. Bei dieser Variante wird der komplette Datenstand gesichert. So bietet ein Hot Backup lediglich einen Export der Daten und keine dauerhafte Sicherung. Das Cold Backup hingegen sichert die Daten in einem konsistenten Zustand.

5.4 Bedrohungen im Datenbankbereich

Das Datenbanken sensible Informationen beinhalten ist allgemein bekannt. Dementsprechend werden verschiedene Maßnahmen getroffen, um diese Daten zu schützen. Dennoch existieren Bedrohungen, die Datenbanken gefährden. Folglich verursachen diese Angriffe eine Manipulation in Bezug auf die Zugriffskontrolle, Integrität, Vertraulichkeit oder Verfügbarkeit. Somit sind die Daten großen Risiken ausgesetzt. Basierend auf Recherchen in [2][41][42][43], werden die verbreitetsten Bedrohungen aufgelistet und erläutert.

5.4.1 Excessive Privilege Abuse

Auf der Vergabe von Privilegien sowie Zugriffsrechten liegt in der Datenbanksicherheit ein wesentlicher Fokus. Dementsprechend sollte ein Benutzer nur die Rechte besitzen, die er für die Ausübung seiner Funktion benötigt. Dennoch werden einem Benutzer häufig alle Privilegien zugewiesen, was für missbräuchliche Zwecke genutzt werden kann. Beispielsweise hat ein Benutzer in einem Unternehmen die Aufgabe alle Mitarbeiterinformationen aktuell zu halten. Da er jedoch alle Rechte besitzt, könnte er dies zu seinem Vorteil nutzen und zum Beispiel die Mitarbeiterbezüge beliebig verändern. Dieses Problem entsteht dadurch, dass die Datenbankadministratoren nicht genügend Zeit investieren können um jedem Benutzer die Zugriffsrechte und Kontrollmechanismen anzugeben. Deshalb werden allen Benutzern die gleichen Rechte zugewiesen, was verheerend für ein Unternehmen sein kann.

Um Excessive Privilege zu unterbinden, kann nach Shulmans Publikation das Query-Level Access Control eingesetzt werden [2]. Rechte werden auf ein Minimum reduziert und ermöglichen nur eine begrenzte Anzahl an SQL-Operationen. Maßgeblich dabei ist die Granularität der Zugriffskontrolle. So wird dem Benutzer die Ausübung seiner Funktion gestattet, löst aber einen Alarm aus sobald er beispielsweise die Mitarbeiterbezüge ändert. Die Rechteverteilung für alle User ist nach dem Query-Level Access Control für Datenbankadministratoren überaus zeitintensiv. Eine bessere Alternative bietet das SecureSphere Dynamic Profiling [2]. Dies ist eine automatisierte Version des Query-Level Access Controls und vereinfacht die Rechteverteilung durch automatisch lernende Algorithmen. Dadurch werden allen Benutzern die Zugriffsrechte automatisch verteilt. Außerdem wird bei einer nicht genehmigten Aktion ein Alarm ausgelöst und diese Aktion sofort unterbunden.

5.4.2 Legitimate Privilege Abuse

Diese Bedrohung weist Parallelen zum zuvor beschriebenen Excessive Privilege Abuse auf. Jedoch besteht hier die Gefahr, dass ein Benutzer mit zugewiesenen Rechten, Daten für unautorisierte Zwecke ausnutzt. Ein Beispiel hierfür wäre die Abfrage aller Informationen von Angestellten eines Unternehmens durch einen Mitarbeiter mit legitimen Zugriff. Die Struktur der Applikation limitiert die Ausgabe der Mitarbeiterhistorie. Dennoch besteht die Möglichkeit sich mit Hilfe eines MS-Excel Clients mit der Datenbank, über Umwege, zu verbinden. Da der Benutzer einen legitimen Zugriff hat, kann er alle Datensätze abspeichern. Diesbezüglich sind zwei Arten von Risiken zu beachten. Im ersten Fall übergibt der Mitarbeiter die Daten für Geld an Dritte. In der zweiten Variante ist das Problem, dass die Datensätze lokal abgespeichert werden. Dadurch sind die Informationen unzähligen Viren, wie Trojanern oder Malware ausgeliefert.

Das Risiko der Bedrohung kann reduziert werden, indem die Zugriffskontrolle verwendet wird [2]. Jedoch liegt hier nicht der Fokus auf das Minimieren der SQL-Operationen, sondern auf dem Kontext hinter dem Datenbankzugriff. Dabei sind besonders die Client Applikationen, Zeit oder Standorte zu beachten, mit denen Benutzer identifiziert werden können. Durch die zusätzliche Hilfestellung von SecureSphere Dynamic Profiling wird automatisch ein Modell erstellt, das sich alle Kontextinformationen zurechtlegt. Sobald eine Verbindung zu einer Datenbank hergestellt wird, welche nicht den Kontextinformationen entspricht, wird ein Alarm ausgelöst. Wenn sich beispielsweise ein Mitarbeiter, mit Hilfe des MS-Excel Clients mit der Datenbank verbindet, wird dieser sofort aufgespürt. Zusätzlich können weitere Details wie die IP-Adresse oder das verbrauchte Datenvolumen eingesehen werden.

5.4.3 SQL-Injection

Diese Art der Bedrohung ist weit verbreitet und eine große Herausforderung für die Sicherheit von Datenbanken. Hier ist die Gefahr, ein unautorisiertes Statement in eine SQL Datenschnittstelle einzuschleusen. Das Ziel von SQL-Injection ist, die SQL-Syntax auszunutzen und Schwachstellen zu identifizieren, um einen Datenbankzugriff zu ermöglichen. Die Entstehung dieses Risikos gründet auf schlechten Eingabeüberprüfungen sowie ungewollter Interpretation von Metacharakteren der Datenbank. Die Motivation ist auf vorhandene Prozeduren sowie Input Parameter zugreifen zu wollen und daraus einen unautorisierten und uneingeschränkten Zugriff zu erhalten. Dementsprechend kann ein Angreifer dadurch die Inhalte einer Datenbank stehlen, zerstören oder manipulieren. Die Konsequenzen in den Bereichen Vertraulichkeit, Authentifikation, Autorisierung und Integrität sind enorm.

Der folgende Absatz beschreibt nach Halfond [41] mehrere mögliche, bisher identifizierte SQL-Injection Typen.

Tautologien - Bei dem Tautologie-basierten Angriff ist das primäre Ziel ein Statement dementsprechend zu manipulieren, dass die Statement-Bedingung immer True ergibt. Dabei sind die Konsequenzen dieser Attacke von den resultierenden Ergebnissen des Statements abhängig. Ein weit verbreitetes Szenario ist das Umgehen der Authentifikation und die darauffolgende Einsicht der Daten. Dieser Injection-Typ verwendet zur Manipulation die WHERE-Bedingung eines Statements, in dem die Bedingung anhand einer Tautologie ersetzt wird und somit alle Daten einer Tabelle ausgeliefert werden. Ein Angreifer könnte beispielsweise, laut Halfond beschrieben [41], für das Login über eine Applikation die folgende Zeichenkombination in einer WHERE-Klausel angeben " ' or 1=1 --". Durch die Kombination der Zeichen wird die WHERE-Klausel in eine Tautologie verwandelt. Somit wird durch das Statement jede Zeile als True gekennzeichnet und liefert alle Werte einer Tabelle.

Illegal/Logically Incorrect Queries – Diese Art von Attacke ist darauf ausgerichtet alle wichtigen Informationen über den Typ und die Struktur der Back-End Datenbank einer Web Applikation zu sammeln. Dementsprechend ist das Ziel, Informationen für weitere Angriffe zu sammeln. Die Gefahr liegt hier besonders bei den von einem Applikationsserver retournierten Fehlermeldungen, weil die Server in den meisten Fällen eine große Menge an Informationen beinhalten. Zweifellos dienen die detaillierten Meldungen den Entwicklern der Applikation als Hilfestellung. Genauso hilft es auch den Angreifern, die dadurch viele Informationen über das Back-End der Applikation sammeln können. Dementsprechend versuchen Angreifer Statements einzufügen, die Fehler in der Syntax, Typ-Umwandlung oder Logik einer Datenbank hervorbringen.

Union Query – Das Ziel dieses Injection-Typs ist es einen verwundbaren Parameter auszubeuten, sodass dadurch die retournierten Datensätze einer Abfrage manipuliert werden. Durch diese Technik ist es dem Angreifer möglich die Applikation zu täuschen und Datensätze einer anderen Tabelle anstatt der beabsichtigten aufzurufen. Der Publikation von Halfond [41] zufolge könnte das Statement folgenderweise ausgeführt werden: *UNION SELECT <rest of injected query>*. Da der Angreifer den gesamten zweiten Teil der Abfrage kontrolliert, kann diese dazu genutzt werden auf eine spezifische Tabelle zuzugreifen. Die Datenbank liefert als Ergebnis einen Datensatz, welcher auf einer Vereinigung von den Resultaten der originalen ersten und der zweiten eingefügten Abfrage basiert.

Piggy-Backed Queries – In dieser Attacke liegt der Fokus des Angreifers darauf, mehrere zusätzliche Abfragen in einer ursprünglichen Abfrage einzufügen [41]. Diese Art von Injection Attacke unterscheidet sich von den restlichen, weil es nicht darum geht die ursprüngliche Abfrage zu modifizieren, sondern gezielt versucht wird neue Abfragen einzuschleusen. Dadurch erhält die Datenbank eine bestimmte zusätzliche Anzahl von abzuarbeitenden Abfragen. Dieser Angriff kann großen Schaden anrichten, wenn Angreifer das beschriebene Szenario durchführen können. Die potentielle Gefahr dieses Angriffs ist von der Datenbankkonfiguration abhängig, ob aufeinander folgende Statements innerhalb eines gesamten String ausgeführt werden dürfen.

Dennoch sind laut Shulman verschiedene Techniken vorhanden um SQL-Injection zu verhindern [2]. Beispielsweise identifiziert Intrusion Prevention anfällig gespeicherte Prozeduren oder SQL-Injection Strings. Mit zusätzlicher Hilfe von SecureSphere, einer Kombination aus Dynamic Profiling, Intrusion Prevention und Correlated Attack Validation Technologien, besteht die Möglichkeit SQL-Injections auf schnelle Art und Weise zu identifizieren. Durch Dynamic Profiling werden zuverlässige Zugriffskontrollen und Query Patterns angefertigt, indem automatische Benutzerprofile erstellt werden. Somit kann jede ausgeführte Abfrage anhand der Patterns und Benutzer verglichen werden. Dementsprechend können alle unpassenden Abfragen rechtzeitig identifiziert und Maßnahmen eingeleitet werden. Correlated Attack Validation filtert mit Hilfe der SecureSphere Layer alle eingehenden Sicherheitsverstöße. Deshalb kann SecureSphere allfällige SQL-Injections aufspüren und unterbinden.

5.4.4 Denial of Service (DoS)

Das Denial of Service ist eine immense Bedrohung für Organisationen und Unternehmen [43]. Das Ziel dieser Attacke ist, einen Computer oder ein Netzwerk anzugreifen und für den Benutzer unzugänglich zu machen. Dabei kann es auf unterschiedliche Layer, wie zum Beispiel auf die Netzwerk-, Applikations- oder Datenbankebene agieren. Im Datenbankbereich kann es einen großen Schaden verursachen. Unter anderem werden Services und Daten manipuliert oder Ressourcen ausgenutzt. Um Denial of Service zu verhindern muss ein Schutz auf mehreren Ebenen vorhanden sein. Grundlegend sind die Kontrolle der Verbindungen und Zugriffe sowie die Reaktionszeiten. Dadurch können zum Beispiel die Verbindungsbandbreite limitiert oder die Reaktionszeiten von Server gemessen werden. Weiterhin unterscheidet Mittner die Distributed Denial of Service Attacken (DDoS) basierend auf der Anzahl der Angriffsrechner [42]. Somit ist, ab einer ungefähren Anzahl von 50 Angriffsrechnern, von einer DDoS Attacke die Rede. Der folgende Abschnitt behandelt unterschiedliche DoS Attacken [42].

Broadcast Storms – Diese Art von DoS Attacken stammen aus einer älteren Generation, können aber vor allem in lokalen Netzwerken einen großen Schaden anrichten. Die Problematik liegt hier in der mangelhaften Wartung der Netzwerktopologien. Das Ziel des Angreifers ist, jeden Rechner in einem Netzwerk mit einer großen Menge an IP-Paketen, die mit nicht vorhandenen Zielen adressiert sind, zu überfluten. Wenn die Rechner diesen gewaltigen Datenstrom aufrechterhalten, wird das gesamte Netzwerk überlastet, weil die falsch adressierten Pakete über die Gateways andauernd in andere Subnetze verschoben werden. Es bedarf einer durchdachten und effektiven Planung des Netzwerkes, um diese Art von Gefahr zu vermeiden.

Smurf – Die Arbeitsweise einer Smurf-Attacke ähnelt der Broadcast Storm Bedrohungen, die sich aber in der Ausführung unterscheiden. Das Ziel ist, an der Broadcast Adresse eines Netzwerkes eine große Anzahl von ICMP-Paketen zu senden, wobei die Pakete mit der Adresse des Angriffsziels getarnt sind. Dadurch erhält jeder Rechner innerhalb eines Netzwerkes die versendeten Pakete, wobei sich die ICMP-Anfragen durch die Anzahl der Rechner vervielfachen. Eine wichtige Rolle spielt die Netzwerkkonfiguration. Im Fall einer korrekt aufgestellten Konfiguration werden die

ICMP-Antworten am Router des Netzwerkes aufgehalten und an das Angriffsziel nicht weitergeleitet. Sobald ICMP-Broadcasts zulässig sind, werden die ICMP-Antworten an das Angriffsziel weitergeleitet und der Angreifer kann mit einer geringen Leistungskapazität das Ziel mit ICMP-Paketen überfluten. Weiterhin werden durch die ICMP-Antworten die gesamte Leistungskapazität und Datenkommunikation unterbrochen. Im schlimmsten Fall besteht die Möglichkeit von Serverausfällen, die darauffolgend vom Netz getrennt werden müssen. Die Attacke ist eine große Gefahr, weil die Angreifer, durch die Tarnung als Angriffsziel, schwer zu identifizieren sind. Deshalb ist eine präzise Netzwerkadministration erforderlich, um dieser Bedrohung Stand zu halten.

SYN-Flooding – Sobald ein Client eine TCP-Verbindung zu einem Server aufbaut muss ein Three-Way-Handshake ausgeführt werden. Dabei sendet der Absender SYN-Pakete an dem Empfänger, womit eine Verbindung angekündigt wird. Daraufhin erfolgt die Antwort des Empfängers anhand eines ACK-Paketes. Folglich muss der Absender das SYN/ACK-Paket wiederum mit einem ACK-Packet bestätigen, um somit den Verbindungsaufbau zu vervollständigen. Sobald der Absender eine falsche IP-Adresse benutzt, wird vom Angriffsziel angenommen, dass das SYN/ACK-Paket verloren gegangen ist und versucht innerhalb eines bestimmten Intervalls die Quittung erneut zu senden. Der Verbindungsaufbau wird zwar nach einer bestimmten Zeit abgebrochen, allerdings gehen in der Zwischenzeit eine große Menge an Speicherplatz und Rechenleistung verloren. Wenn ein Absender eine SYN-Flooding Attacke ausführt, versendet er an das Angriffsziel eine Unzahl an SYN-Paketen, woraufhin der Empfänger alle SYN-Pakete quittieren muss. Vorausgesetzt der Angreifer verwendet eine nicht vorhandene IP-Adresse, sendet der Router zusätzlich ein ICMP-Unreachable-Paket zurück, dass einen unerreichbaren Host definiert. Folglich wird ein großer Datenverkehr erzeugt und das Opfer ist anschließend nicht mehr erreichbar.

Ping of Death – Die Ausführung von Ping of Death Attacken sind einfach und effektiv. Anhand der Spezifikation des IP-Protokolls darf ein Paket nicht mehr als 65,536 Bytes beinhalten. Folglich wird für eine einfachere Übertragung das Paket in kleinere Teile zerlegt. Sobald alle einzeln vorliegen, muss der Empfänger alle Pakete wieder zusammenfügen. Wenn die ankommenden Pakete mehr als die vorher definierte Größe beinhalten, kann es dazu führen das der interne Speicher überläuft und im schlimmsten Fall der Rechner abstürzt. Dennoch sind die meisten aktuellen Netzwerke gegen diese Gefahr ausgerüstet [43].

5.4.5 Weak Audit Trail

Das Audit Trail ist ein wichtiger Bestandteil für die Überprüfung und Aufnahme von sensiblen Daten oder ungewöhnlichen Datenbanktransaktionen. Außerdem ist es ein bedeutendes Fundament für die Datenbankentwicklung. Bei einer schwachen Audit Policy können, laut Shulman [2] Risiken auf mehreren Ebenen einer Organisation entstehen.

Regulatory Risk – Organisationen mit schwachen oder kaum vorhandenen Audit Mechanismen müssen zunehmend feststellen, dass sie im Widerspruch zu den staatlichen regulatorischen Anforderungen stehen. Dabei definiert Shulman die Sarbanes-Oxley (SOX) oder die Healthcare Information Portability and Accountability Act (HIPAA) als zwei staatlich regulierte Beispiele mit einer klaren Audit Anforderung [2].

Deterrence – Wie Videokameras sollen die Audit Mechanismen zur Abschreckung der Angreifer dienen indem klargestellt wird, dass Audit Tracking eingesetzt wird.

Detection and Recovery – Audit Mechanismen repräsentieren die letzte Defensivlinie einer Datenbank. Sobald ein Angreifer es schafft die vorhandenen Verteidigungslinien zu umgehen, kann er durch die gesammelten Audit Daten identifiziert werden. Dabei kann eine Verknüpfung zu unterschiedlichen Benutzern oder Systemen erstellt werden.

Lack of User Accountability – Sobald sich Benutzer über Web-Applikationen wie SAP, Oracle E-Business Suite oder PeopleSoft an der Datenbank anmelden, haben die ursprünglichen Audit Mechanismen keine spezifische Möglichkeit den Benutzer zu identifizieren. In diesem Fall werden alle Benutzeraktivitäten mit den Accountnamen der Web Applikation assoziiert. Somit können keine Verbindungen zwischen den ursprünglichen Audit Logs einer betrügerischen Transaktion und dem dazu verantwortlichen Benutzer erstellt werden.

Performance Degradation – Ursprüngliche Datenbank Audit Mechanismen sind für den Verbrauch der CPU und Disk Ressourcen bekannt. Deshalb zwingt der durch die Aktivierung von Audit Features entstehende Leistungsabfall viele Organisationen dazu, die Audit Mechanismen ganz oder teilweise zurückzuziehen.

Separation of Duties – Benutzer mit legitimen oder nicht legitimen administrativem Zugang können innerhalb der Datenbank die Audit Mechanismen ausschalten, um beispielsweise betrügerische Aktivitäten zu verschleiern. Deshalb müssen die Aufgaben der Audit Mechanismen auf Datenbankadministratoren und Datenbank Serverplattformen aufgeteilt werden.

Limited Granularity – Bei vielen ursprünglichen Audit Mechanismen werden keine bis wenige Details aufgezeichnet. Dabei ist dies dringend notwendig, um beispielsweise Angriffe aufzudecken oder Daten wiederherzustellen. So werden zum Beispiel die benutzte Datenbank Clientapplikation, IP-Adresse, abgefragten Attribute oder gescheiterte Abfragen nicht dokumentiert.

Proprietary – In jeder Datenbank Serverplattform sind die Audit Mechanismen einzigartig. Dementsprechend gibt es zum Beispiel große Unterschiede zwischen Oracle und MS-SQL Logs. Für Organisationen mit unterschiedlichen Datenbankumgebungen besteht somit kaum die Möglichkeit, eine einheitliche Implementierung skalierbarer Audit Prozesse zu schaffen.

Weiterhin definiert Shulman, dass Datenbank Softwareplattformen die grundlegenden Audit Möglichkeiten integrieren, diese aber durch multiple Schwächen in der Entwicklung eingegrenzt werden [2]. Folglich werden durch qualitative Netzwerk-basierte Audit Instrumente viele Schwächen identifiziert und mit den ursprünglichen Audit Tools assoziiert. Die Kombination der folgenden beschriebenen Attribute kann die Datenbank und die administrativen Kosten reduzieren [2].

High Performance – Netzwerk-basierte Audit Instrumente können auf einer hohen Geschwindigkeit operieren, ohne die Datenbankperformance zu beeinflussen. Sobald die Audit Prozesse durch die Netzwerkinstrumente übernommen wurden, kann man von einer höheren Datenbankperformance ausgehen.

Sepeartion of Duties – Der ausschlaggebende Punkt ist das unabhängige Agieren der Netzwerkinstrumente, in dem die Audit Aufgaben von den administrativen Aufgaben getrennt werden.

Cross-Platform Auditing – Weiterhin können durch den Einsatz der Netzwerkinstrumente die unterschiedlichen Datenbankumgebungen unterstützt werden. Dadurch können einheitliche Standards und Audit Operationen in heterogenen Datenbankumgebungen ausgeführt werden.

Universal User Tracking – Mit diesem Hilfsmittel können Benutzer, unabhängig davon ob der Zugang zur Datenbank über eine Web Applikation vollzogen wurde, mit deren Aktivitäten identifiziert werden. Durch SecureSphere Interfaces werden beispielsweise Login Informationen, Benutzersessions und deren Transaktionen dokumentiert.

Weiterhin definiert Dean, welche Audit Mechanismen in Oracle Datenbanken notwendig sind und wie die Einstellungen durchgeführt werden müssen [44]. In jeder Datenbank sind Audits verpflichtend und automatisch angelegt. Dokumentiert werden Start und Stopp der Datenbank sowie die Logins von SYSDBA oder SYSOPER Privilegien. Alle Daten werden in einem Ordner hinterlegt, welcher anhand des AUDIT_FILE_DEST Parameters definiert wird. Sobald der Parameter ein unzulässiges Verzeichnis beinhaltet, kann die Datenbank nicht gestartet werden. Weiters muss für die erstellten Daten genügend Speicher zur Verfügung stehen. Dazu sollten periodische Reviews und Säuberungen der Daten durchgeführt werden. Folglich ist zu beachten, dass der administrative SYS Benutzer von allen Audit Policies ausgeschlossen ist und dies zu einer Lücke führen kann. Diese Einstellung kann geändert werden, indem der Parameter AUDIT_SYS_OPERATIONS auf TRUE gesetzt wird. Darauf basierend

werden in Oracle alle Statements des SYS Benutzers unter dem angegebenen Pfad AUDIT_FILE_DEST gespeichert.

Ein Standard Audit ist dann aktiviert, wenn der AUDIT_TRAIL Parameter mit den AUDIT/NOAUDIT Konfigurationskommandos eingeschalten wird. Dabei sind unterschiedliche Verwaltungsmethoden der Audit Daten vorhanden. Die folgenden Parameter definieren die möglichen Speichermethoden der Daten [44].

NONE - deaktiviert das Standard Audit
OS - speichert Audit Datensätze in ein OS File
DB – speichert Audit Datensätze in die SYS.AUD$ Tabelle der Datenbank
DB_EXTENDED – speichert Audit Datensätze in die SYS.AUD$ Tabelle der Datenbank inklusive aller ausgeführten SQL Statements und allen SQL-Bind Werten.
XML – speichert Audit Datensätze in ein OS File anhand eines XML Formates.
XML_EXTENDED – speichert Audit Datensätze in ein OS File anhand eines XML Formates inklusive aller ausgeführten SQL Statements und allen SQL-Bind Werten.

Letztlich ist es notwendig zu definieren, welche Interaktionen dem Audit Trail unterzogen werden sollen. Generell müssen alle Veränderungen in der Datenbank beobachtet werden. Dies wären beispielsweise die System Privilegien, erfolglose Operationen sowie das Login und Logoff von Benutzern [44]. Genauer betrachtet muss der Fokus auf die sensiblen und wichtigen Daten gelegt werden.

5.5 Security Metrics

Security Metrics haben im Computer Security Bereich erst spät einen bedeutenden Zuwachs an Aufmerksamkeit erlangt. Jansen definiert es nicht als neuen Themenbereich, aber als einen der sporadisches Interesse geweckt hat [5]. Weiters ist laut Jansen das Ziel der bisherigen Publikationen die Erstellung von Definitionen, weil in der Praxis geringe Security Metrics nachweisbar waren [5].

Savola definiert den Fachausdruck Security Metrics gegenwärtig als einen wichtigen Bestandteil in Bezug auf die Sicherheitslevel, Sicherheitsperformance sowie die Indikatoren und Stärken der vorhandenen Sicherheit [3]. Dennoch wird der Ausdruck im Kontext der Informationstechnologie oft missverstanden und unbeachtet. In der Praxis werden die Messungen gleichbedeutend mit den Security Metrics dargestellt. Folglich beziehen sich Messungen einen bestimmten Datensatz in Hinsicht auf spezifisch formulierten Faktoren, wobei Security Metrics die abgeleiteten Daten der Messungen zuständig sind. Dadurch werden mittels Security Metrics Verbesserungen im Management-, Monitoring- oder Performancebereich erzielt [3].

In der bisherigen Arbeit war der Fokus auf die Sicherheit der Vertraulichkeit, Integrität und Verfügbarkeit gerichtet. Dennoch beinhaltet dieser Ansatz einige Einschränkungen, da nur drei wesentliche Ziele bevorzugt werden. Dementsprechend müssen weitere Vorsätze hinzugefügt werden, um zusätzliche Sicherheitsziele zu berücksichtigen. Die Internationale Telecommunication Union (ITU) hat dazu einen wichtigen Beitrag geleistet und das Ausmaß erweitert [3]. Access Control, Authentication, Non-Repudiation, Data Confidentiality, Communication Security, Data Integrity oder Availability und Privacy sind weitere wichtige Faktoren.

In großen Systemen ist die Abschätzung der Sicherheit komplexer, weil Komponenten ermittelt werden müssen, die möglicherweise einen höheren Schutz erfordern. Eine Möglichkeit hierfür ist die Zerlegung des Systems in kleine Bereiche [45][46]. Deshalb ist ein Modell erforderlich, das an den Attributen der Low-level Ebene orientiert ist. Die Dekomposition Methode startet mit den High-level Sicherheitszielen eines Systems und arbeitet sich in die Low-level Ebene zu den Komponenten und Interaktionen. Im folgenden Beispiel von Wang und Wulf werden die Sicherheit und das Konstrukt eines Hauses analysiert [46]. Hier ist das Ziel die Privatsphäre zu unterstützen und unautorisierten Zugang zu vermeiden. Bei diesem Ereignis besteht die Annahme, dass ein perfektes Konstrukt vorhanden ist, welches jeder Attacke standhalten kann. Das Haus hat in dem Fall ein Fenster und eine Tür, die dementsprechend geschützt werden müssen. Wie in Figur 4 ersichtlich, werden die beiden Faktoren auf eine weitere Ebene aufgeteilt. Somit wurde die Dekomposition Methode auf das Haussystem angewendet.

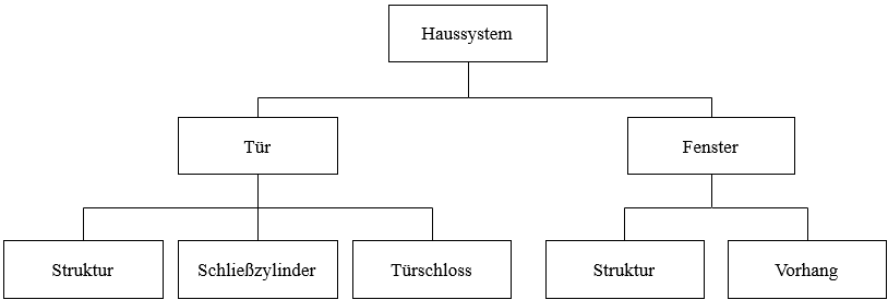


Fig. 4: Dekomposition Methode anhand eines Haussystems [46]

Letztlich kann das Ergebnis so interpretiert werden, dass die Attribute Struktur, Schließzylinder und Türschloss der vorhandenen Tür einen unautorisierten Zugang verhindern sollen. Weiters ist durch die Struktur und den Vorhang des Fensters der Schutzer der Privatsphäre möglich. Die Interpretation des Systems ist ausschlaggebend dafür, ob eine Entscheidung vorteilhafter ist als die andere. Relevant sind die Sicherheitsanforderungen an das System und die zu schützenden Komponenten [46].

Zweifellos kann die Dekomposition Methode für jede High-level Ebene verwendet werden. Im weiteren Verlauf des Kapitels werden basierend auf dem vorhin beschriebenen Beispiel, die ausgearbeiteten Sicherheitsziele, auf mehreren Ebenen unterteilt. In Figur 5 liegt der Fokus zunächst auf der Authentifizierung. Dabei sind die Mechanismen und Autorisierung wichtige Faktoren. Die untergeordneten Knoten wurden darauf basierend identifiziert.

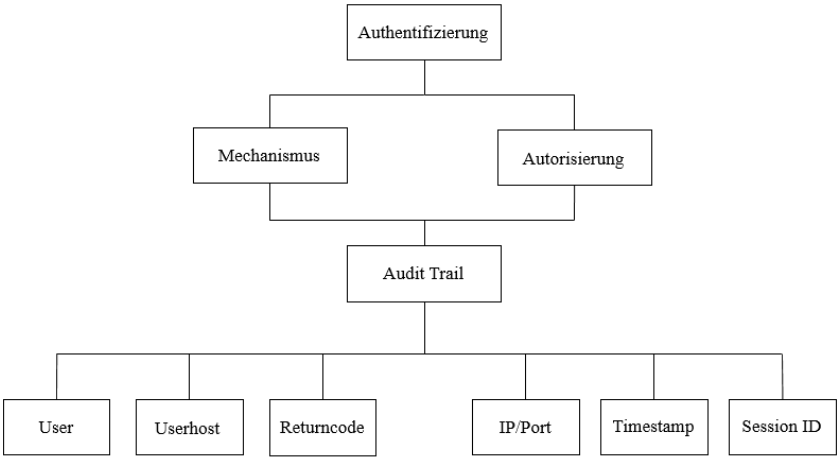


Fig. 5: Dekomposition Methode zur Authentifizierung

Die Authentifizierung bildet einen wichtigen Bestandteil der ausgearbeiteten Sicherheitsziele. Dementsprechend muss man Schlussfolgerungen ziehen, welche Faktoren für die Unterteilung relevant sind. Der Mechanismus und die Autorisierung spielen dabei eine wichtige Rolle, weil sie das Fundament für eine strukturierte Authentifizierung bilden. Ungeachtet dessen ob es Standard Mechanismen oder erweiterte sind, können alle Zugriffe anhand der Audit Trails erfasst werden. Die unterste Ebene beinhaltet schließlich die messbaren Attribute, die aus den Audit Datensätzen identifiziert werden. Durch Attribute wie Userhost, Returncode oder IP/Port-Adresse können alle möglichen Zugriffe ausgewertet und bemessen werden.

Die Verschlüsselung ist ein weiteres wichtiges Sicherheitsziel, das Unbefugten den Zugriff auf Informationen erschwert und dementsprechend schützt. Dabei ist die Höhe der Sicherheit von der Auswahl unterschiedlicher Algorithmen abhängig. Wichtige Faktoren könnten hier zum Beispiel ein passender Algorithmus mit einer hohen Schlüssellänge und einem großen Speicher sein. Weiters ist die Performance Rate ein wichtiges Instrument, wodurch eine größere Sichtweise auf die Schnelligkeit und das Verhalten des Algorithmus ermöglicht wird. Figur 6 definiert das Dekomposition Modell anhand der Verschlüsselung.

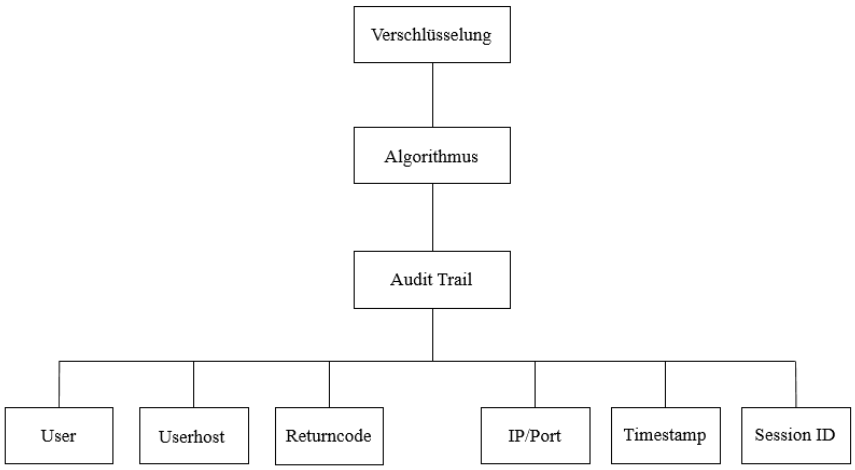


Fig. 6: Dekomposition Methode anhand der Verschlüsselung

Die Verschlüsselung steht als Sicherheitsziel an oberster Stelle. Daraufhin erfolgt eine weitere Unterteilung in Faktoren, welche einen effektiven Verschlüsselungsalgorithmus beinhaltet. Unabhängig davon ob ein Standard oder erweiterter Algorithmus zum Einsatz kommt, kann das Modell beliebig getestet werden. Die unterste Ebene definiert die Attribute, die dementsprechend gemessen und ausgewertet werden können.

Integrität ist auf verschiedenen Arten und Weisen definiert. Die Integrität herrscht zum Beispiel bei Abbildungen korrekter Inhalte oder unmodifizierter Zustände. Genauso bedeutungsvoll ist die Erkennung von Modifikationen und temporalen Korrektheiten. Sie ist gleichermaßen bedeutsam, wie alle bisher definierten Sicherheitsbereiche. Folglich, kam man basierend auf dem wissenschaftlichen Beitrag von Wang und Wulf zum nachfolgenden Ergebnis [46].

Die Integrität als Sicherheitsziel steht in dem exemplifizierten Fall auf der obersten High-level Ebene. Darauf beruhend wurden alle relevanten untergeordneten Faktoren ermittelt. Die Verschlüsselung und Authentifizierung erwiesen sich in diesem Vorgang als wichtige Unterkategorie, weil sie mit ihren Funktionalitäten die vorhin beschriebenen Eigenschaften der Integrität aufrechterhalten. Weiters sind der Algorithmus, der Mechanismus sowie die Autorisierung wichtige Kriterien für die übergeordneten Faktoren. Die wichtigsten Elemente befinden sich in der untersten Low-level Ebene. Anhand der definierten Attribute kann die Sicherstellung der Integrität bewertet werden. Dabei ist es notwendig, messbare Daten in den Attributen zu erlangen.

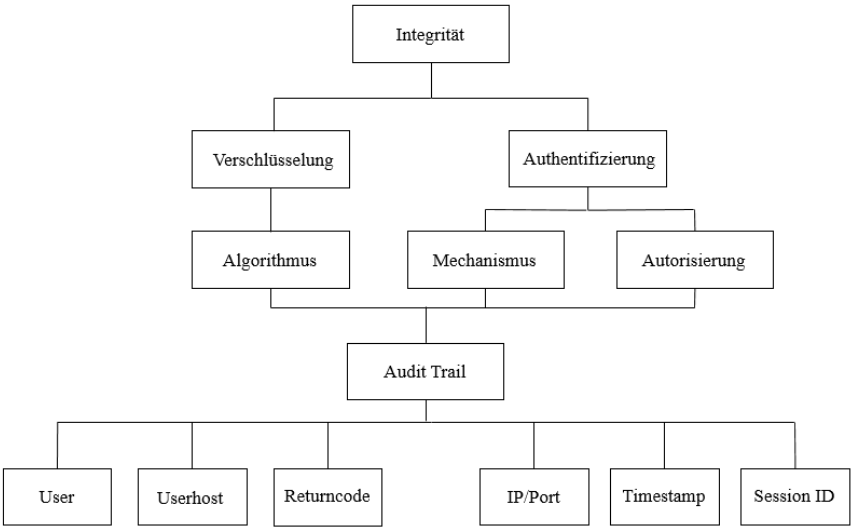


Fig. 7: Dekomposition Methode anhand der Integrität

Folgend identifiziertes Sicherheitsziel bezieht sich auf die Vertraulichkeit. Das Bundesamt für Sicherheit in der Informationstechnik definiert die Vertraulichkeit als Schutz vor unbefugter Preisgabe von Informationen [26]. Um dieses Sicherheitsziel aufrecht zu erhalten, sind die Verschlüsselung und die Authentifizierung ausschlaggebend. Mit Hilfe der Verschlüsselung sollen bestimmte Daten vor Angriffen oder Manipulation geschützt werden. Dabei ist die Auswahl des Algorithmus entscheidend, weil die Performance oder der Schlüsselaustausch eine hohe Priorität aufweist. Ein zusätzlich hilfreicher Faktor ist die Authentifizierung. Durch den Einsatz von Mechanismen und Autorisierungen dürfen nur zuständigen Benutzern die Daten zur Verfügung stehen. Dennoch könnten ohne des Einsatzes der Audit Trails, keine relevanten Auswertungen erstellt werden. Durch die definierten Attribute der untersten Low-level Ebene können nach ausgeführten Bedrohungen Daten erlangt werden. Folglich das Sicherheitsziel anhand im Hinblick auf Bedrohungen entsprechend der gewonnenen Daten ausgewertet und darauf basierend bewertet werden.

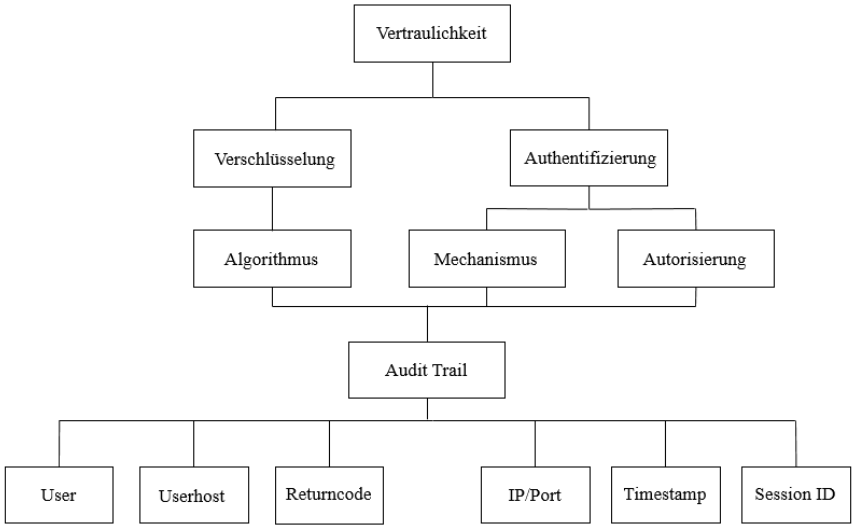


Fig. 8: Dekomposition Methode anhand der Vertraulichkeit

Die Verfügbarkeit ist das letzte identifizierte Sicherheitsziel, das ausschlaggebend für diese Arbeit ist. Dabei gilt es sowohl Systemausfälle zu verhindern, als auch den Zugriff auf Daten innerhalb eines vereinbarten Zeitrahmens zu gewährleisten [65]. Die Low-level Hierarchie entspricht dem Aufbau der Integrität und Vertraulichkeit, weil hier die Verschlüsselung und Authentifizierung schützende Faktoren für eine Systemverfügbarkeit sind. Dementsprechend gleicht das Dekomposition Modell, dem der Vertraulichkeit und Integrität.

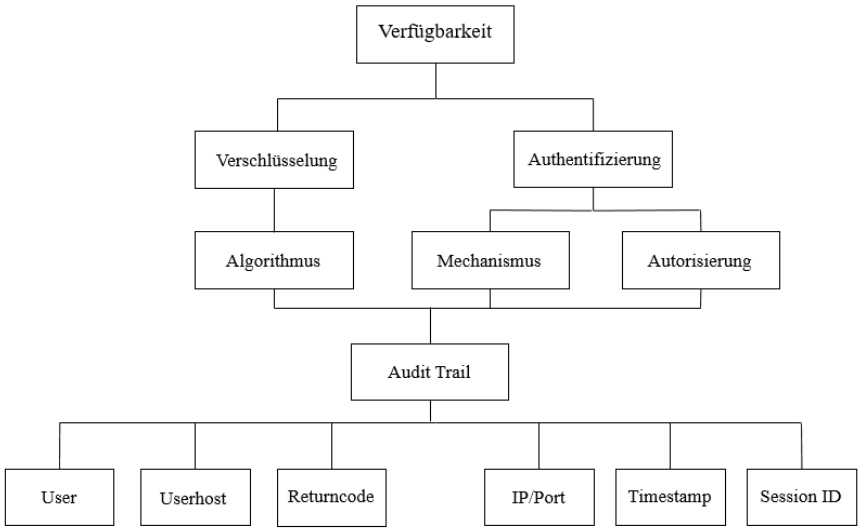


Fig. 9: Dekomposition Modell anhand der Verfügbarkeit

Ein wichtiger Bestandteil für die Aufbereitung der Informationen zu diesem Kapitel basiert auf der Publikation von Wang und Wulf [46]. Während der Umsetzung entstanden einige Probleme, weil mittels dieser Theorie keine bedeutenden Resultate in der Praxis erzielt werden konnten. Daher wurde die Top-Down Methode mit Hilfe des Bottom-Up Verfahrens zur Anwendung gebracht. Der Schwerpunkt lag nach weiteren Literaturrecherchen auf Datenbankprotokollen, Log-Dateien und Audit Trails, weil mit den zuletzt genannten Audit Trails konnten relevante Ergebnisse in der Praxis erzielt werden konnten. Aufgrund der Komplexität verschiedener Attribute, wie etwa der IP-Adresse, dem Benutzername oder des Userhosts, konnten Daten gesammelt werden. Diese Daten mussten folglich ausgewertet und dementsprechend bewertet werden.

.

6 Literatur und Praxis

In diesem Kapitel werden die theoretischen Betrachtungen aus der Literatur den praktischen Anwendungen gegenübergestellt. Dabei ist es wichtig zu definieren, wie weit die Literatur und Praxis von einander liegen. Weiters werden die Einsatzbereiche sowie Vor- und Nachteile von Security Metrics näher erläutert. Außerdem folgen ausführliche Beschreibungen bezüglich der Einsätze in Datenbankbereichen sowie der möglichen Erstellungsverfahren.

6.1 Security Metrics und Messungen im Datenbankbereich

Security Metrics und Messungen sind eine bedeutende Konstante, wenn etwa Entscheidungen bezüglich des Designs der Sicherheitsarchitektur oder der Sicherheitsoperationen getroffen werden müssen [3][5]. Nach Jansen bieten Security Metrics eine quantitative, als auch objektive Basis für die Sicherheit eines Systems [5]. Dabei werden die Hauptgründe der Nutzung wie folgt definiert:

- **Strategic Support** – Die Bewertung von Sicherheitsattributen dient der Definition unterschiedlicher Entscheidungsmöglichkeiten. Dadurch sind Beschreibungen der Programmplanungen sowie Produkt- oder Serviceauswahlen möglich.

- **Quality Assurance** – Security Metrics können während der Softwareentwicklungsphase eingesetzt werden um Schwachstellen im Code oder auch bei der Performance zu verbessern.

- **Tactical oversight** – Monitoring und Reporting sind wichtige Bestandteile zur Identifizierung des Sicherheitsstatus eines IT-Systems. Damit können Sicherheitsanforderungen und Risiken gehandhabt werden.

Das US National Institute of Standards and Technology (NIST) definiert Security Metrics wie folgt [8]: Security Metrics sind Tools die erstellt werden, um Entscheidungen und Verbesserungen in Bereichen der Performance sowie Verantwortlichkeit, durch das Sammeln, Analysieren und Reporting von relevanten Daten, zu ermöglichen. Das Ziel der Performancemessung ist, den Status von gemessenen Aktivitäten zu überwachen und Verbesserungen bei den erwähnten Aktivitäten durch Einsatz der fehlerbehebenden Aktionen einzubinden.

Weiters argumentiert Vaughn, dass Messungen oft als Security Metrics missverstanden werden [9]. Er definiert Security Metrics als individuelle Messungen in zusammenhängenden Begriffen oder Frameworks. Wie in den Kapiteln zuvor bereits erwähnt, zieht Savola eine klare Trennlinie zwischen den Begriffen Messergebnisse und Security Metrics indem er Messergebnisse als “Single-Point-in-Time“ Daten, gemessen anhand spezifischer Faktoren, und Security Metrics als Beschreibungen der abgeleiteten Daten der Messungen definiert [3].

Jansen zufolge wurden einige Versuche unernommen um die Sicherheit in Systemen durch Kriterien wie der Trusted Computer System Evaluation Criteria, Information Technology Security Evaluation Criteria oder Systems Security Engineering Capability Maturity Model zu messen. Dennoch waren alle Kriterien mit geringem Erfolg eingestuft [5]. Im weiteren Verlauf werden nun wichtige Faktoren und Kriterien diskutiert, mit denen positive Resultate bezüglich Security Metrics erreicht werden können [5].

Correctness und Effectiveness: Die Sicherheit eines IT-Systems beinhaltet zwei voneinander abhängige Aspekte, nämlich die Korrektheit und Effektivität. Der zuerst genannte Aspekt stellt sicher, dass die sicherheitsdurchführenden Mechanismen korrekt implementiert wurden. Weiters beschreibt die Effektivität, ob die Mechanismen die definierten Sicherheitsziele abdecken. In vielen praktischen Anwendungen wird die Korrektheit vollständig implementiert, aber die Effektivität nicht beachtet.

Leading Versus Lagging Indicators: Security Metrics können führende, übereinstimmende oder Spätindikatoren einer bestehenden Sicherheit eines Systems sein. Übereinstimmende Indikatoren reflektieren Sicherheitsbedingungen als gleichzeitig ablaufende Bedingungen, während führende oder Spätindikatoren die bestehenden Sicherheitsbedingungen vor jeglichen Umstellungen definieren. Viele Security Metrics werden als übereinstimmende Indikatoren betrachtet. Dadurch ist die Verbesserung eines Sicherheitsstatus, sowie die Minimierung von Risiken möglich. Letztlich kann ein System und deren Schwächen nur erkannt werden, wenn Attacken erfolgreich absolviert wurden.

Organizational Security Objectievs: Organisation sind in unterschiedlichen Bereichen, mit verschiedenen Tätigkeiten verortet und daher von diversen potenziellen Bedrohungen gefährdet. Durch die eben aufgezählten Merkmale ist es von hoher Priorität, signifikante Sicherheitsziele durchzusetzen. Diese sind für die Ziele und Bestrebungen einer Organisation von immenser Bedeutung, weswegen die Security Metrics anhand der Sicherheitsziele definiert werden müssen.

Qualitative and Quantitative Properties: Die Messung von Softwareeigenschaften ist generell schwierig in der Umsetzung. Relevante Eigenschaften wie Komplexität, Benutzerfreundlichkeit und Skalierbarkeit sind prinzipiell bezeichnende Qualitäten, die in der Praxis umfangreich zu erfassen sind. Der Differenz zwischen qualitativen und quantitativen Security Metrics kann schwer definiert werden. Qualitative Aufgaben können beispielsweise dazu verwendet werden, um quantitative Messungen von Sicherheitseigenschaften zu repräsentieren.

Measurements of the Large versus the Small: Messungen in kleineren Aufgaben haben einen größeren Erfolgsfaktor im Gegensatz zu großen und Komplexen Zielvorgaben. Das liegt an der Größe und Komplexität der Funktionalitäten. Sobald die Anzahl der Komponenten eines Systems steigt, wächst die Anzahl der Interaktionen zwischen den Komponenten. Composability spielt dabei eine wichtige Rolle. Dadurch können Interaktionen von Komponenten in Betracht gezogen und in verschiedenen Kombinationen selektiert werden. Es gilt einen Weg zu finden, Messungen von kleinen Systemen innerhalb der großen Systeme einzubinden.

Alle in diesem Kapitel beschriebenen Kriterien spielen für diese Arbeit eine wichtige Rolle. Obwohl im Datenbankbereich kaum vertreten, sind Security Metrics in viele Einsatzbereichen relevant. Literaturrecherchen führten zu einer Arbeit, die sich mit Security Metrics und Messungen in Oracle Datenbanken beschäftigt [10]. Der theoretische Teil von Vladoiu entspricht den Definitionen und Beschreibungen der Security Metrics und Messungen dieser Arbeit. Allerdings ist der praktische Teil nicht aussagekräftig, weil nicht beschrieben wird welche Daten und Attribute für die Messungen in Betracht gezogen werden und wie die Security Metrics dies bewerten können.

Letztlich waren für die Erzielung positiver Resultate die beschriebenen Kriterien eine große Hilfestellung. Dabei war es erforderlich, korrekte Mechanismen zu entwickeln und effizient auszuwerten. Die Auswahl der Spätindikatoren ermöglicht es nach umgestellten Bedrohungen, die Security Metrics je nach Anpassung auszuwerten. Weiters ist die Auswahl der Sicherheitsziele eines Systems von großer Bedeutung. Dadurch können die zu messenden Bereiche eingegrenzt werden. Die Auswahl der Attribute kann wie bereits erwähnt schwer definiert werden, weil sie sowohl qualitativ als auch quantitativ sein kein.

6.1 Einsätze der Security Metrics

Security Metrics dienen der Bewertung von Sicherheitszielen eines Systems. Dabei ist es irrelevant, in welchem Unternehmensbereich Security Metrics zum Einsatz kommen, weil Security Metrics ein relativ junger Themenbereich sind, indem bisherige praktische Informationen keine bedeutenden Ergebnisse liefern konnten [7]. Dennoch gibt es Ansätze von Frameworks und Taxonomien, die in weiterer Folge näher erläutert werden. Das von Nichols und Peterson entwickelte Framework, das unterschiedliche Messungen auf einer Open Web Application mit den gefährlichsten Bedrohungen zulässt, sei an dieser Stelle exemplarisch [6]. Das Ziel war es, mittels einer Security Scorecard, Messergebnisse in einer qualitativen Ampelstruktur darzustellen. Dadurch sollen Resultate anhand der Farben besser nachvollziehbar sein. Weiters definiert Savola einige Einsatzmöglichkeiten der Security Metrics wie zum Beispiel im Business Management [7].

Businessziele sind darauf ausgelegt die Sicherheit und das Trust Management zu steuern. Deswegen müssen die Security Metrics entsprechend der Ziele eines Unternehmens ausgerichtet sein. Zur Erstellung der Security Metrics ist es notwendig mit den definierten Businesszielen zu starten und einen Abgleich der Lower-level Security Managementziele mit dem Kontext darzustellen. Dennoch muss beachtet werden, dass beispielsweise in staatlichen Organisationen, Businessziele durch organisationsspezifischer Ziele ersetzt werden können. Die folgende Grafik illustriert die von Savola dargestellten Business-level Security Metrics [7].

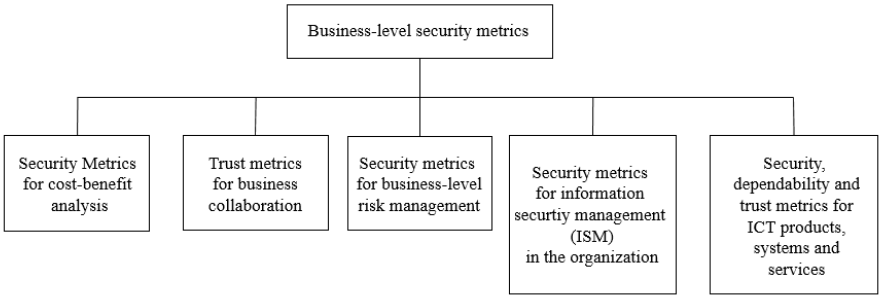


Fig. 10: Business-level Security Metrics [7]

In diesem Zusammenhang verweist Savola auf das von Blakley definierte Security ROI (Return on Investment), weil es effektive Security Metrics darstellt [7]. Security ROI sind demnach so definiert, dass die Menge der alljährlichen Vorteile gegenüber den Kosten dargestellt werden. Letztlich wird das Trust Management als ein relativ junges Forschungsfeld beschrieben mit der Zielsetzung, die Darstellung des Controllings und die Modelle vom Trust Management zu verbessern.

Gleichermaßen definiert Savola die Security Metrics für das Information Security Management in Organisationen anhand der aufgestellten Taxonomie von Swanson [7][11]. Die folgende Grafik zeigt die dargestellten Ebenen der Security Metrics für das Information Security Management [7].

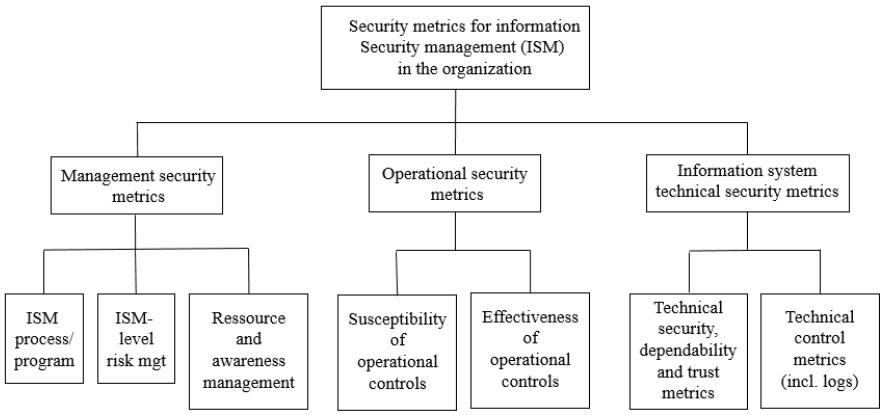


Fig. 11: Security Metrics für das Information Security Management [7]

Security Metrics für das Information Security Management dienen der Evaluierung von Sicherheitskontrollen, Planungen, Policies sowie der Zertifikationen von zugelassenen Aktivitäten. Weiterhin basiert die Human Resource Veranlagung auf Training und Auswahl der Security sowie der Evaluierung von Human Resource Aufgaben. Operationale Security Metrics beziehen sich auf die Anfälligkeit und Effektivität von operationellen Sicherheitsmethoden. Dabei liegt ein großer Fokus auf der vorfallenden Reaktion der archivierten und aufrechterhaltenden Prozesse der Software und Hardware. Außerdem werden die Dokumentation der Security sowie die Datenintegrität und Planungskontingenz bewertet. Die technischen SDT (Security, Dependability and Trust) Metrics sind eine Teilmenge oder Instanz der SDT Metrics für das Product Life Cycle Management [7]. Swanson definiert eine Anzahl von Richtlinien der Security Self-Assessment für Informationssysteme, die auf dem Framework der U.S. Federal IT Security basieren [11]. Weiters erwähnt Savola die Publikation der NIST [12], die Bewertungsmethoden und Prozeduren anhand minimaler Level für Organisationen definiert und die Security Elemente in ihrem Informationssystem bewerten. Eine weitere Ausgabe der NIST [8] beinhaltet eine Anleitung wie die verwendeten Security Metrics bei Organisationen die Zulänglichkeit der Security Elemente, Policies und Prozeduren identifizieren. Effektive und effiziente Security Metrics werden dazu verwendet, die Ergebnisse der Security Implementierung von Elementen zu überwachen. Das Federal Information Processing Standard (FIPS) [13] hat laut Savola Security Kategorien sowohl für Informationen und Informationssysteme veröffentlicht. Dabei kann die potenzielle Auswirkung klassifiziert werden als gering, mittel oder hoch [13]. Nach Lennons Definition, kann das Thema Security Metrics letztlich wie folgt zusammengefasst

werden [14]: Die Anzahl der möglichen Security Metrics die auf existierenden Policies und Prozeduren basieren sind gewaltig. Dementsprechend ist die Priorisierung der möglichen Security Metrics unumgänglich.

Einige der herausforderndsten Kategorien sind die Sicherheits-, Vertrauens- und Zuverlässigkeitsmetriken für die Bereiche Produkte, Systeme und Services [7]. Die folgende Grafik bezeichnet Savola als Basiskonzept für die eben erwähnten Kategorien [7].

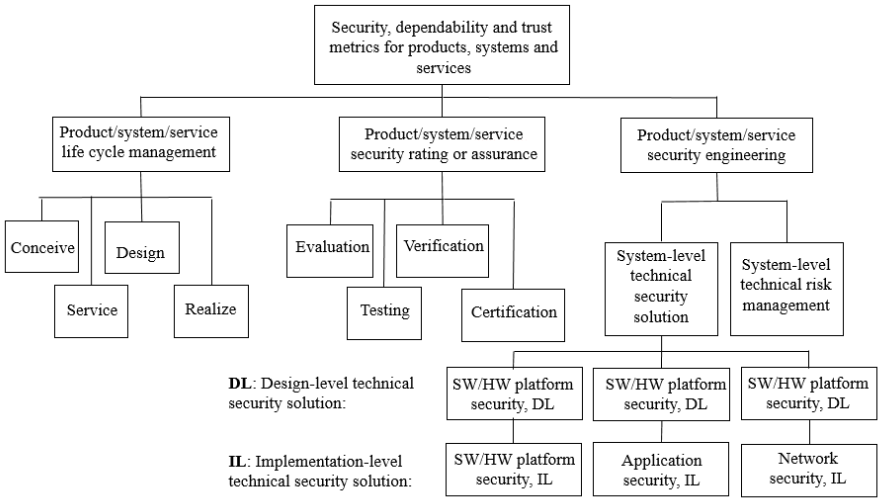


Fig. 12: Security, Dependability und Trust Metrics für Produkte, Systeme und Services [7]

Während der Entwurfsphase sind die Definitionen der Sicherheitsanforderungen die Basis zur Messung der Sicherheit [7]. Die Designphase vereinigt die Aktivitäten zu einem architektonischen und Lower-level Design. Weiters sind die Durchführungen von Tests, Analysen und Überprüfungen von Bedeutung. Daraufhin nennt Savola das von der ISO/IEC publizierte Systems Security Engineering Capability Maturity Model (SSE-CMM) [15] als Beispiel eines Produkt Life-Cycle Security Metrics. Dieses beinhaltet Security Metrics für fällige Bewertungen der Sicherheitslevel von Security Engeeniring Processes und deren Ergebnissen. Die resultierenden Standards können als Basis für die Evaluierung, ausgeführt von neutralen Dritt-Parteien, neben den Herstellern und Vermittlern definiert werden. Eine der bekanntesten Bemühungen ist das Common Criteria (CC) der ISO/IEC [16][7]. Das CC Standard ist eine Kombination aus unterschiedlichen existierenden Standards wie zum Beispiel das TCSEC (Trusted Computer System Evaluation Criteria), CTCPEC (Canadian Trusted Computer Product Evuluation Criteria) und FC (Federal Criteria for Information Technology Security). Dabei erwähnt Savola, dass der Vorteil der Reports in der Qualität der Produkte liegt und nicht ausschließlich in der Security [7].

SDT Metrics sind technische Security Lösungen in der Software/Hardware Platform Security, Application Security oder Network Security die auf mehrere design-level metrics unterteilt werden können. Design-level security engineering metrics können demnach als detaillierte implementation-level metrics definiert werden, die hauptsächlich vulnerability metrics repräsentieren. Basierend auf dem CVSS (Common Vulnerability Scoring System) können die Schwachstellen als bug, flaw, behaviour, output, outcome oder event in einer Applikation, einem System oder Gerät definiert werden [7]. In diesem Zusammenhang nennt Savola einige aufgestellte Security Metrics wie zum Beispiel das von der NIST publizierte Software Assurance Metrics and Tool Evaluation (SAMATE) project. Mit diesem Tool können unterschiedliche Fragen im Bereich Software assurance, tools und security metrics beantwortet werden. SAMATE fokussiert sich dabei ausschließlich auf Security Metrics und Messungen für Softwares und SSA (Software Security Assurance). MITRE bietet zum Beispiel eine standardisierte Sprache in Bezug auf einer sorgfältigen Kommunikation für den Informationsaustausch mit Benutzern, indem Repositories entwickelt werden [7].

6.2 Vor- und Nachteile

Security Metrics sind in vielerlei Hinsicht vorteilhaft, weil damit Managemententscheidungen optimiert, Kosten eingespart und Systemmanagern oder Riskmanagern Informationen zu deren Systemen geboten werden können [3]. Weiterhin definiert Savola die Security Metrics als hilfreich in folgenden Bereichen [3]: Risk Management, Vergleich von Security-enforcing Mechanismen, Software Security Assurance, Security Testing, Security Performance, Security Monitoring, Intrusion detection und prevention.

Tashi und Ghernaouti nennen die Security Metrics als wichtigen Faktor in organisatorischen und finanziellen Angelegenheiten eines Unternehmens, weil sie zu einer besseren Verantwortung in Bezug auf die Informationssicherheit führt [48]. Durch die erzielten Ergebnisse kann das Organisationsmanagement die technisch oder operativ implementierten Maßnahmen identifizieren. Diese Resultate ermöglichen es, die Probleme aufzuspüren und zu bereinigen.

Die Problematik liegt in der Unterscheidung von brauchbaren und unbrauchbaren Security Metrics. Vijayan definiert das Problem in Bezug auf das Auffinden und Erfassen der relevanten Security Metrics und deren Zusammenhang mit den Businesszielen eines Unternehmens [49]. In den meisten Fällen ist ein großer Fokus auf die Compliance Richtlinien gerichtet, anstatt zu identifizieren, wie effektiv Risiken anhand der Sicherheitsprogramme ermittelt werden. Außerdem wird die einfache Gestaltung der Ergebnisse den Details vorgezogen. Die Anzeige des Sicherheitsstatus in einer einfachen Ampel-basierten Art ist vorteilhaft, bietet aber wenig Details, sodass die Risiken oder die Bedrohungen innerhalb eines Unternehmens kaum nachvollziehbar sind. Problematisch ist, dass Security Metrics als exakte Wissenschaft betrachtet werden [49]. Sie definieren zwar wie viele Bedrohungen aufgehalten wurden, können allerdings nicht eindeutig vorhersagen wie viele weitere Bedrohungen gestoppt oder auch nicht beachtet wurden.

6.3 Verfahren zur Erstellung von Security Metrics

Es gibt mehrere Faktoren, die bei der Erstellung von Security Metrics berücksichtigt werden müssen. Relevante Bestandteile sind die Analyse von messbaren Komponenten oder potenziellen Bedrohungen und Schwachstellen. Weiters können beispielsweise durch den Einsatz von Top-Down oder Bottom-Up Methoden unterschiedliche Resultate entstehen. Savola beschreibt in seiner Publikation die Erstellung von Security Metrics anhand folgender Prozesse [45][51].

1. Zunächst müssen die Bedrohungen und Schwachstellen analysiert werden. Daraufhin erfolgt eine Ermittlung und Ausarbeitung der Bedrohungen innerhalb des Systems und der Einsatzumgebung. Sobald genügend Informationen vorliegen, können die bekannten und verdächtigen Schwachstellen identifiziert werden.

2. Folglich müssen die Sicherheitsanforderungen anhand der in Punkt 1 analysierten Bedrohungen und Schwachstellen definiert und priorisiert werden. Hier ist es von großer Wichtigkeit, den relevantesten Sicherheitsanforderungen die meiste Aufmerksamkeit zu widmen.

3. Weiters folgt die Identifizierung von messbaren Komponenten anhand der Dekomposition Methode, beginnend mit der High-level Ebene der Sicherheitsanforderungen.

4. Erstellung messbarer Architekturen und Mechanismen zur Beweissicherung für Security Metrics.

5. Auswahl der messbaren Komponenten, die basierend auf deren Durchführbarkeit, als Ausgangspunkt für die detaillierten Security Metrics, verwendet werden.

6. Letztlich erfolgt die Definition und Bestätigung der Security Metrics sowie der Funktionalitäten und Prozesse in denen sie in Verwendung treten.

Weiters definiert Savola, dass die Schritte iterativ sind und die Reihenfolge ebenso variieren kann [45]. Die Schritte 1 und 2 sollten innerhalb des Forschungs- und Entwicklungszyklus so früh wie möglich starten, wobei die Schritte 3 und 4 parallel zu einander ausgeführt werden können.

6.3.1 Threat and Vulnerability Analysis

Die Bedrohungsanalysen sind ein wichtiger Prozessbestandteil zur Untersuchung von relevanten Bedrohungen in einer SUI (System under Investigation). Die Resultate der Analyse bilden eine priorisierende Beschreibung der Bedrohungssituationen. In der Praxis sind mehrere Analysemöglichkeiten vorhanden, von der einfachen Aufzählung bis hin zur gründlichen Modellierung. Savola definiert die folgenden Analyseprozessmöglichkeiten basierend auf den threat risk modeling process vom Microsoft [45]:

1. Identifizierung von Sicherheitszielen
2. Untersuchung der SUI Architektur
3. Anwendung der Dekomposition Methode an der SUI Architektur um die Funktionen und Entitäten zu identifizieren, die einen Einfluss auf die Sicherheit ausüben.
4. Identifizierung von Bedrohungen
5. Identifizierung von Schwachstellen

Weiters können die Sicherheitsziele in mehrere Kategorien wie zum Beispiel in eine finanzielle, private oder regulatorische aufgeteilt werden. Außerdem sind unterschiedliche Quellen vorhanden, die als Hilfestellung zur Entwicklung der Sicherheitsziele dienen. Einige nennenswerte Beispiele sind gesetzliche, regulatorische oder standard security policies. Sobald die Sicherheitsziele definiert wurden, ist es notwendig die SUI Architektur zu analysieren und unterschiedliche Komponenten zu identifizieren. Die Funktionen und Entitäten der Sicherheitsziele werden ermittelt, indem eine Aufteilung der Architektur durchgeführt wird. Folglich definiert Savola, dass der Angreifer das Ziel verfolgt, soviel Gewinn wie möglich aus der Bedrohung zu generieren. Zur Identifizierung von Bedrohungen, ist die Auseinandersetzung mit folgenden Fragen obligatorisch [45]: Wie kann der Widersacher die Informationen benutzen oder manipulieren, um das System zu modifizieren oder kontrollieren? Besteht die Möglichkeit, dass der Widersacher Zugang zu den Informationen erhält ohne aufgespürt zu werden?

Die Bedrohungen können zum Beispiel anhand eines passenden Modells wie STRIDE (Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, Elevation of Privilege) klassifiziert werden [45]. Das DREAD (Damage Potential, Reproducibility, Exploitability, Affected Users, Discoverability) ist ein gleichermaßen geeignetes Klassifikationsschema das zur Quantifizierung, zum Vergleich und zur Priorisierung des Risikopotenzials einer Bedrohung zweckdienlich ist [45].

6.3.2 Dekomposition Methode

Im Kapitel 5.5 wurde die Dekomposition Methode anhand des Beispiels von Wang und Wulf ausführlich erklärt [46]. Diese Methode basiert auf dem Top-Down Verfahren, wobei aus einer definierten High-level Ebene mehrere Unterteilungen angestrebt werden. Weiters folgt eine kurze Zusammenfassung über den Prozess dieser Methode, der anhand folgender Schritte bestimmt werden kann [45][46].

1. Identifikation von Sicherheitszielen eines Systems zur Erstellung einer Analyse.

2. Identifikation von sukzessiven Komponenten, die einen Beitrag zur erfolgreichen Abwicklung der Ziele bewirken.

3. Analyse der untergeordneten Knoten um zu ermitteln, ob eine weitere Dekomposition erforderlich ist. Wenn dies zutrifft wird der Prozess wiederholt, indem die untergeordneten Knoten als aktuelles Ziel definiert werden.

4. Der Prozess wird dann beendet, wenn keine weiteren Knoten aufgeteilt werden können oder eine Analyse der Komponenten nicht mehr notwendig ist. Wenn alle Aufteilungen beendet sind, müssen alle untergeordneten Knoten messbare Komponenten sein.

6.3.3 Goal-Question-Metric (GQM) Methode

Das Goal-Question-Metric (GQM) Verfahren ist basierend auf einem Top-Down Ansatz, eine systematische Vorgehensweise zur Planung von Messmethoden [52]. Ein wichtiger Bestandteil dieser Methode ist zunächst die Definition des Zieles. Daraufhin werden spezifische Fragestellungen formuliert, die anschließend zu Security Metrics verfeinert werden. Somit entsteht ein Entscheidungsmodell, das mit Hilfe von Messmethoden die Ziele widerspiegelt [52]. Dabei werden nach Basili drei wichtige Kriterien genannt [52][53]:

- Fokussierung auf konkrete Ziele
- Integration aller Produkte, Zwischenprodukte, Prozesse und Ressourcen
- Darstellung im Kontext der Organisation, Umwelt und Ziele.

Ziele stehen immer in Bezug zu einem Objekt [52]. Deshalb ist es notwendig zu definieren, welche Ziele durch Messungen erreicht werden sollen. Nach Basili werden messbare Objekte wie folgt definiert [52][53]:

- **Produkte**: sind Artefakte, Leistungen sowie Dokumente wie zum Beispiel Spezifikationen, Entwürfe oder Protokolle.
– **Prozesse**: sind zusammenhängende Arbeitsschritte um etwas innerhalb einer Organisation zu erstellen. Dies betrifft beispielsweise Interviews, Spezifikationen, Tests oder Planungen.
– **Ressourcen**: werden durch die Nutzung der Prozesse ermöglicht. Beispielsweise betrifft das Personal, die Hardware, Software oder Büros.

Die eben erwähnten Objekte werden in unterschiedlichen Projekten anderen Anforderungen unterzogen. Außerdem können Ziele zu einander in Konflikten geraten, weil die Nutzung der Objekte sich überschneidet.

Weiters werden zu jedem Ziel mehrere Fragen gebildet. Dabei ist es wichtig, die Ziele nachvollziehbar zu definieren. Folglich steht jede Frage für eine Hypothese, wodurch die Verfolgung des Ziels konkretisiert wird. Dadurch werden Entscheidungen im Kontext der Fragestellung sichtbar. Habenicht definiert Teilkriterien, die zur Lösung der notwendigen Fragestellung erforderlich sind [52]: Sichtweise, Anwendungsbereich, Kontext.

Schließlich werden die gemessenen Daten mit Hilfe der gebildeten Fragen, Hypothesen und Ziele interpretiert. Dementsprechend können die Daten objektiv oder subjektiv betrachtet werden. Letztlich werden die qualifiziertesten Security Metrics ausgewählt, die in der Lage sind die definierten Eigenschaften zu beschreiben.

6.4 Unterschiede zwischen Literatur und Praxis

Unterschiedliche Organisationen wie zum Beispiel die NIST oder ENISA legen einen großen Fokus auf Security Metrics [3][5]. Weiters gibt es verschiedene Publikationen, die einen wichtigen theoretischen Bestandteil dieses Themas bilden. Basierend auf der Publikation von Yasasin [54] beinhaltet folgende Tabelle die Definitionen der Security Metrics aufgelistet nach den veröffentlichten Referenzen.

Referenz	Definition der Security Metrics
Savola (2007) [7]	Security Metrics bieten eine quantitative und objektive Basis zur Absicherung der Sicherheit. Weiterhin erleichtern sie die Business- und Entwicklungsentscheidungen hinsichtlich der Informationssicherheit.
Kaur und Jones (2008) [55]	Security Metrics bieten ein Framework anhand kommerziell erhältlicher Produkte und Dienstleistungen zur Bewertung der Sicherheit.
Masera und Fovino (2010) [54]	Security Metrics sind Indikatoren und keine Sicherheitsmessungen. Sie sind stark vom Kontext der Messungen abhängig und sollten nicht als absolute Werte bezüglich einer externen Skala betrachtet werden.
Rudolph und Schwarz (2012) [54]	Security Metrics sind Sicherheitsmaßnahmen mit einer definierten Anzahl an Regeln für die Interpretation der Messdaten.

Preschern (2015) [56]	Security Metrics beschreiben den Sicherheitsstatus eines Systems qualtitativ oder quantitativ.
Ouchani und Debbabi (2015) [54]	Security Metrics sind quantitative Maßnahmenindikatoren die angeben, inwiefern die betrachtete Entität die Eigenschaft besitzt sicher zu sein.

Tabelle 4: Definitionen der Security Metrics [7][54][55][56]

Die eben beschriebenen Definitionen zeigen, dass die theoretischen Ansichten der Verfasser einander in großem Maße ähneln. Dennoch beschreibt Yasasin, dass sich viele Forscher weiterhin mit diesem Thema beschäftigen [54]. Vaughn erklärt dazu, dass bezüglich der Charakterisierung oder Interpretation von Messungen und Security Metrics in vielen Fällen Verwechslungen aufscheinen [9]. Dementsprechend definiert er, dass die Brauchbarkeit der Messungen stark von den eben erwähnten Interpretationen abhängt. Bei direktem Vergleich mit anderen Messergebnissen kann bestimmt werden, inwiefern ein Resultat wünschenswert ist. Gleichermaßen beschreiben Wang und Wulf die Wichtigkeit der Interpretation des Systems [46]. Allerdings definieren sie, dass eine Entscheidung nicht relevanter ist als eine andere. Savola erläutert die Auswahl der Sicherheitsziele als einen außerordentlich wichtigen Bestandteil der Security Metrics [3]. Je größer die Auswahl der Sicherheitsziele, desto mehr Bereiche können abgesichert werden.

Wie in den Kapiteln zuvor erwähnt, gab es bisher einige praktische Ansätze zur Umsetzung von Security Metrics. Ein angemessenes Beispiel ist das von Nichols und Peterson entwickelte Open Web Application Security Project (OWASP) [6]. In diesem Projekt sind viele theoretische Ansätze zum Thema Security Metrics demonstriert. So definieren sie beispielsweise die notwendigen Sicherheitsziele und bilden die Resultate anhand einer Ampel-Skala ab. Das von der ISO/IEC entwickelte Common Criteria basiert ebenfalls auf der Auswahl von Bewertungszielen [16]. Dementsprechend inkludieren die Ziele beispielsweise die Softwareapplikation, das Betriebssystem oder das Netzwerk mit allen Terminals und Servern. Außerdem definieren sie zusätzlich die Zielgruppen, welche aus Konsumenten, Entwicklern und Gutachtern bestehen. Folglich werden Bedrohungen und Schwachstellen identifiziert, die auf die Sicherheitsziele ausgerichtet sind. Das Forum of Incident Response and Security Team (FIRST) hat das Framework Common Vulnerability Scoring System (CVSS) zur Charakterisierung und Darstellung von Schwachstellen in Softwares entwickelt [57]. Dabei definieren sie, dass es keinen expliziten Zeitpunkt zum Ausführen der CVSS erfordert und die Security Metrics dementsprechend variieren können. Weiterhin sind die Umgebung und die Interpretation ein wesentlicher Bestandteil des Frameworks. Schließlich basiert die Bewertungsmethode auf einer Skala, mit der die Schwachstellen mit Angaben wie hoch, mittel oder niedrig bewertet werden.

Letztlich kann zusammengefasst werden, dass sowohl im praktischen als auch theoretischen Ansatz die Auswahl der Sicherheitsziele sowie die Ausgabe und Interpretation der Bewertung eine große Rolle spielen. Dementsprechend bildet die Theorie ein essenzielles Fundament zur praktischen Entwicklung von Security Metrics. Dennoch beschreiben Wang und Wulf, dass ein Ansatz oder eine Interpretation nicht

vorteilhafter ist als eine andere [46]. Gleichermaßen ist es ein Fehler anzunehmen, dass Security Metrics eine exakte Wissenschaft bieten [49]. Deshalb ist für eine dauerhafte praktische Verbesserung, die einzusetzende Theorie von großer Wichtigkeit.

7 Gap Analysis

Die Gap Analyse ist ein hilfreicher Ansatz bei der Identifizierung notwendiger Schritte, zur Erreichung der Projektziele. Dementsprechend ist es von großer Bedeutung, die Lücke zwischen dem aktuellen und zukünftigen Projektstatus sowie die erforderlichen Prozesse zur Schließung der Lücke festzustellen. Folglich wird die Gap Analyse anhand der folgenden drei Schritte durchgeführt [58].

– Identifizierung des zukünftigen Status
– Analyse des aktuellen Status
– Identifizierung der Prozesse um die Lücke zu schließen

Zukünftiger Status	Aktueller Status	Nächster Prozess
Problembeschreibung	Beschreibung und Motivation des Themas	1. Identifizierung von Problemen und Schwachstellen in Datenbanken. 2. Weiters muss definiert werden, wie eine potenzielle Lösung der Thematik aussehen könnte.
Erwartete Ergebnisse	Problembeschreibung	1. Nach einer Literaturrecherche soll ersichtlich sein, welche Bedrohungen einen Schaden anrichten und ob sie für diese Arbeit relevant sind.
Methoden und Vorgehen	Erwartete Ergebnisse	1. Die Definition der zu treffenden Maßnahmen ist notwendig, um die zukünftigen Ziele erreichen zu können. 2. Dementsprechend wichtig ist die Auswahl der vorhandenen Tools sowie der Referenzen.
State of the Art	Methoden und Vorgehen	1. Bedeutsam ist die Identifizierung der

		unterschiedlichen Datenbankmodelle und deren Vor- und Nachteil. 2. Ein notwendiger Bestandteil sind das DBMS und SQL.
Allgemeine Sicherheitsanforderungen in Datenbanken	State of the Art	1. Es sollen mögliche Risiken und Schwachstellen in Datenbanken analysiert werden. Daraufhin kann anhand der Sicherheitsziele definiert werden, welche Bereiche geschützt werden müssen. 2. Die Ausarbeitung der aktuellen Tools oder Methoden kann mehr Klarheit verschaffen. 3. Aufzählung der unterschiedlichen Tools mit deren Vor- und Nachteilen.
Bedrohungen im Datenbankbereich	Allgemeine Sicherheitsanforderungen in Datenbanken	1. Einen Überblick der vorhandenen Bedrohungen und Attacken aufzeigen. 2. Weiterhin muss recherchiert werden, welche Bedrohungen für die Entwicklung in Frage kommen und wie realistisch die Erzielung erfolgreicher Angriffe ist. 3. Folglich ist zu klären, welche Bedrohungen sich auf welche Sicherheitsziele auswirken.
Security Metrics	Bedrohungen im Datenbankbereich	1. Definition der Security Metrics recherchieren und

		deren Zusammenhänge vergleichen. 2. Verfahren und Methoden zur Erstellung von Security Metrics identifizieren. 3. Die wichtigsten Bestandteile sowie die bisher erreichten praktischen Ergebnisse festhalten. 4. Wichtig ist die Identifikation der Bedrohungen und wie sie in der Praxis anhand der Security Metrics bewertet werden können.
Framework Entwicklung	Security Metrics	1. Entwicklung eines Klassendiagramms und Metamodells. 2. Einrichtung einer virtuellen Maschine mit einer Oracle Datenbank in einem lokalen Netzwerk. 3. Literaturrecherche über vorhandene APIs zur Entwicklung der folgenden Bedrohungen: Brute-Force, SQL-Injection, Ping of Death, SYN-Flooding 4. Aktivierung und Einstellungen der Audit Trails in der Oracle Datenbank. 5. Testdurchführungen und Verbesserungen der Bedrohungen durch den Einsatz der Audit Trails

		6. Entwicklung der Security Metrics anhand der identifizierten Sicherheitsziele. Dabei ist es wichtig, einen Zugriff auf die Audit Trail Daten zu ermöglichen, um messbare Ergebnisse zu erhalten. 7. Erstellung einer grafischen Oberfläche zur besseren Handhabung der Bedrohungen und Security Metrics. 8. Beschreibung und Auswertung der erzielten Resultate.
Literatur und Praxis	Framework Entwicklung	1 Wie sahen die bisher veröffentlichten Ergebnisse aus und welchen Einfluss hatte die Literatur? 2 Aufstellung der bisher recherchierten Methoden und Verfahren zur Erstellung der Security Metrics sowie Abweichungen in der Praxis. 3 Definition der recherchierten Einsatzmöglichkeiten der Security Metrics und deren Vor- und Nachteile.
Diskussion der Ergebnisse	Literatur und Praxis	1 Beschreibung der vorhandenen Resultate. 2 Vergleich zwischen den erwarteten Ergebnissen und Testdurchführungen dokumentieren.

Tabelle 5: Gap Analyse

8 Framework Entwicklung

Die Planung, Dokumentation sowie Erstellung von Modellen ist ein wichtiger Bestandteil einer erfolgreichen Framework Entwicklung. Für die Realisierung dieses Tools war die Berücksichtigung der Literatur erforderlich, damit ermitteln werden konnte, mit welchen Entwicklungssprachen das Problem am effektivsten gelöst werden kann. Weiters war die Auswahl der umzusetzenden Datenbankbedrohungen ein wichtiger Faktor. Diese mussten mit den ausgeforschten Security Metrics harmonieren, um auch praktisch Ergebnisse erzielen zu können. Zur praktischen Umsetzung der Funktionalitäten wurde Java in Kombination mit JavaFX verwendet. Mit Hilfe des JavaFX Scene Builders wird ein Framework zur Verfügung gestellt, das eine einfache Gestaltung der Anwendungsoberflächen ermöglicht [60]. Ein weiterer Vorteil ist die einfache Handhabung des Scene Builders mit der Entwicklungsoberfläche NetBeans, wodurch eine übersichtliche Gestaltung des Tools ermöglicht wird. Vor der Entwicklung mussten weitere Einflussfaktoren berücksichtigt werden. Wichtige Aspekte waren die Planung und Aufstellung des Software Architektur Patterns sowie des Klassendiagramms und Metamodells. Daher war die Nutzung des Model-View-Controller (MVC) Patterns essenziel für die Zielerreichung, weil durch die Zuhilfenahme des MVC Patterns ein anpassungsfähiger Programmentwurf entstand, indem Erweiterungen als auch Änderungen leicht vorgenommen werden können. Der Vorteil dieses Patterns ist die Trennung der Komponenten für die Durchführung unterschiedlicher Umsetzungen [61]. Die Model Komponente enthält die darstellenden Daten und ist von der View und von dem Controller unabhängig. Der Controller dient zur Steuerung der Views und der Ausführung von Benutzerinteraktionen. Im weiteren Verlauf werden die Interaktionen geprüft und die Daten aus dem Modell entnommen und wiederum zur View weitergeleitet. Folglich erfolgt die Darstellung der Daten in der View Komponente. Die folgende Abbildung verkörpert die Interaktion innerhalb eines MVC-Patterns [61].

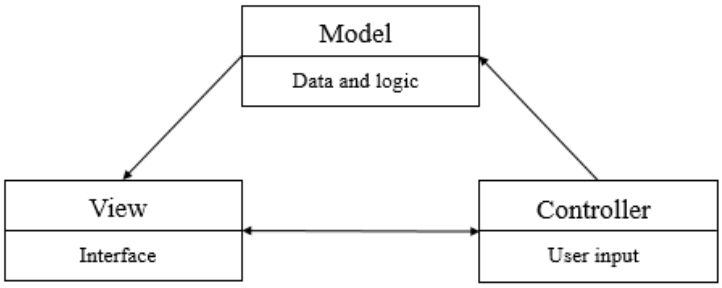


Fig. 13: MVC Pattern basierend auf [61]

Die Aufstellung eines Klassendiagrammes und Metamodells für die Framework Entwicklung ist insofern bedeutsam, als dadurch ein besseres Verständnis für die

Handhabung der Anwendung ermöglicht wird. Im weiteren Verlauf des Kapitels werden die eben erwähnten Modelle abgebildet und definiert. Als zusätzliche Hilfestellung werden alle entwickelten Klassen und deren Assoziationen detailliert beschrieben. Schließlich folgt die Definition der Implementierungen mit Code-Beispielen und einer umfassenden Beschreibung. Die folgende Abbildung ist eine Übersicht aller Klassen, sowie den Beziehungen des entwickelten Frameworks.

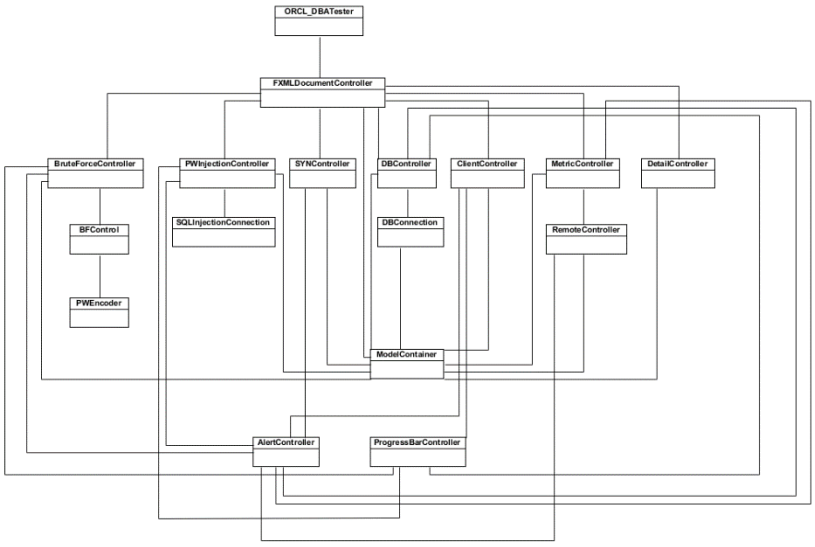


Fig. 14: Übersichtsdiagramm der Klassen und deren Beziehungen

8.1 Modell

Durch den Einsatz des Klassendiagramms wird eine grafische Darstellung von Klassen und deren Beziehungen der entwickelten Anwendung ermöglicht. Zuerst folgt eine detaillierte Beschreibung zu den Funktionalitäten der Klassen. Dadurch soll ein besseres Verständnis ermöglicht werden.

ORCL_DBATester

Hier findet der Start der Applikation statt. Die View wird geladen und bei einer Interaktion an die FXMLDocumentController Klasse weitergeleitet.

FXMLDocumentController

Diese Klasse ist der Hauptcontroller der Applikation. Jede Interaktion einer View wird innerhalb der Klasse aufgerufen sowie überprüft und an einen zuständigen Controller weitergeleitet. Somit dient der FXMLDocumentController als Verteiler an alle weiteren Controller.

BruteForceController

An dieser Stelle übernimmt der BruteForceController die Anforderung von dem FXMLDocumentController. Sobald die View für die Brute-Force Bedrohung ersichtlich ist, wartet die Klasse auf eine Interaktion seitens des Benutzers. Daraufhin werden die eingegebenen Daten zur Bearbeitung an die BFControl Klasse übergeben.

BFControl

Dies ist eine der zwei Klassen, welche die grundlegende Arbeit für die Brute-Force Bedrohung verrichtet. Nachdem der 60 Zeichen lange SHA1 Hashwert übernommen wurde, erfolgt eine Aufteilung in einen Hash- und Saltwert, wobei die ersten 40 Zeichen den Hash und die übrigen 20 dem Salt zugeordnet werden. Daraufhin generiert die Klasse alle möglichen Zeichenkombinationen, die an die PWEncoder Klasse inklusive des Hashwertes weitergeleitet werden. Folglich wartet die BFControl Klasse auf den neu erzeugten Hashwert vom PWEncoder, woraufhin der neu erzeugte und bereits vorhandene überprüft werden. Dabei wird das Szenario solange wiederholt, bis die zwei Hashwerte übereinstimmen und das Passwort als Klartext ausgegeben wird.

PWEncoder

Hier wird von der BFControl Klasse der Salt und die generierte Zeichenkombination übernommen, woraufhin jede Stelle des Salts und der Zeichenkombination in Bytes umgewandelt werden. Danach erfolgt eine Zusammenführung der erstellten Bytes, wobei an erster Stelle die Bytes der Zeichenkombination und danach die des Salts gestellt werden. Daraufhin entsteht mit Hilfe des SHA1-Algorithmus ein verschlüsselter Hashwert. Schließlich erfolgt die Übergabe an der Klasse BFControl, in der der neu erzeugte Hashwert mit den bereits vorhandenen überprüft wird. Dies wird solange wiederholt bis die zwei Hashwerte einander gleichen.

DBController

Diese Klasse ist für die Datenbankverbindung ausschlaggebend. Nachdem ein Benutzer alle relevanten Daten in der View eingegeben hat, übernimmt der DBController die Informationen und überprüft zunächst, ob alle benötigten Daten vorhanden sind. Trifft das zu wird die Klasse DBConnection aufgerufen, andernfalls erscheint eine Fehlermeldung in der man aufgefordert wird, seine Eingabe zu überprüfen. Sowohl bei einer erfolgreichen als auch nicht erfolgreichen Anmeldung werden alle Informationen in der ModelContainer Klasse aktualisiert.

DBConnection

In dieser Klasse erfolgt die Verbindungsherstellung mit der Datenbank. Es werden alle verbindungsrelevanten Daten aus dem DBController entgegengenommen und ein Verbindungsaufbau wird gestartet. Anschließend wird das Ergebnis, unabhängig ob erfolgreich oder nicht erfolgreich, an den DBController retourniert.

ClientController

An dieser Stelle der Anwendung besteht die Möglichkeit eine Verbindungsüberprüfung durch einen Ping durchzuführen oder einen Ping of Death zu starten, wobei ein einfacher Ping zu jeder Zeit ausgeführt werden kann. Nachdem alle benötigten Daten wie die IP-Adresse, der Port, die Pinganzahl und simulierte Clientanzahl eingegeben wurden, startet der ClientController den Ping of Death. Daraufhin wird mit Hilfe des Executors ein fixierter Threadpool erzeugt, um einen Ping of Death simulieren zu können, weil damit alle Threads synchron gestartet werden. Anschließend verschickt jeder Thread die eingegebene Anzahl von Pings an eine Zieladresse. Dieses Szenario wird so lange ausgeführt, bis die Anzahl der Durchläufe beendet ist oder der Executor gestoppt wird. Solange der Executor im Einsatz ist, kann ein Verbindungsaufbau mit der Datenbank getestet werden. Zu diesem Zweck wird die Klasse DBController aufgerufen. Da nach dem Start des Ping of Deaths Performanceprobleme auftreten, wurde die Funktionalität der Anwendung in ein eigenes Projekt ausgelagert um eine schnellere Verbindungsüberprüfung zu ermöglichen.

DetailController

Der DetailController wird nur bei Ausführung des MetricController gestartet. Die Klasse dient als zusätzliche Informationsquelle zu den primären Informationen des MetricControllers. Hier werden der Benutzername, Userhost, die Verbindungszeit und weitere Daten gebündelt, sodass die Intensität der Bedrohung ersichtlich wird. Weiters kann bei einer SQL-Injection Simulation jedes ausgeführte Statement ausgewertet werden. Mit Hilfe einer Combobox kann zwischen den Einträgen navigiert werden.

MetricController

Für die Erstellung der Auswertungen ist der MetricController entscheidend. Die Klasse kann nur gestartet werden, wenn eine Bedrohung simuliert wurde. Dementsprechend wird sonst eine Fehlermeldung ausgegeben. Sobald eine Bedrohung und der Verbindungsaufbau ausgeführt sind, kann die View des MetricControllers gestartet werden. Danach erfolgt eine Überprüfung, um welche Bedrohung es sich handelt und ob der Verbindungsaufbau erfolgreich oder nicht erfolgreich verlaufen ist. Daraufhin muss man den Benutzernamen, das Passwort und den Pfad des Ordners in dem die Audit Dateien liegen eingeben. Währenddessen erfolgt in der Klasse RemoteController eine Kontrolle der eingegebenen Daten. Schließlich werden alle auffindbaren Audit Einträge vom RemoteController übernommen. In weitere Folge kommt es zur Untersuchung jedes Audit Eintrages, indem überprüft wird ob der Benutzer, Zeitstempel sowie Verbindungsstatus der vorhandenen Daten aus dem ModelContainer übereinstimmen. Sobald alle Faktoren einander gleichen, wird der Audit Eintrag gespeichert und kann im weiteren Verlauf im DetailController aufgerufen werden. Letztlich erfolgt je nach Bedrohung die Einstufung der Vertraulichkeit, Verfügbarkeit, Authentifizierung, Verschlüsselung und Integrität. Die Ergebnisse kommen dabei folgenderweise zustande. Für ein “gut“ werden zwei Punkte, ein “durchschnittlich“ ein Punkt und ein “schlecht“ null Punkte vergeben. Das Ergebnis wird in Prozent der maximalen Punktezahl, hier sind es zehn, angegeben. Letztlich wird ein Ergebnis unter

40% als schlecht, zwischen 40% und 70% als durchschnittlich sowie größer 70% als gut bewertet.

RemoteController

Die Aufgabe des RemoteControllers liegt darin, die Audit Dateien aus dem angegebenen Pfad zu durchsuchen und jeden relevanten Audit Eintrag zu speichern. Um die Auswahl einzuschränken werden nur die 10 aktuellsten Audit Dateien in Betracht gezogen. Daraufhin erfolgt eine Überprüfung, ob der Zeitstempel einer Audit Datei mit dem des getesteten Verbindungsaufbaus übereinstimmt. Deshalb werden nur jene Einträge als relevant eingestuft, die entweder eine Minute vor oder nach dem Zeitstempel des Verbindungsaufbaus erstellt wurden. Sobald dies übereinstimmt, wird der Eintrag nach Attributen wie Benutzer, Host, Returncode, Zeitstempel, SQL-Text oder der Session-ID durchsucht und gespeichert. Dieses Verfahren wird solange wiederholt, bis alle Audit Einträge kontrolliert wurden. Letztlich werden alle Audit Einträge an den MetricController weitergeleitet.

PWInjectionController

Diese Klasse dient ausschließlich zur Simulation einer SQL-Injection Bedrohung. Sobald ein Benutzer die Anwendung auswählt, leitet der FXMLDocumentController die Anforderung an den PWInjectionController mit der dazugehörigen View weiter. Danach muss zuerst eine Anmeldung durchgeführt werden um eine SQL-Abfrage zu starten. Ist die Anmeldung erfolgreich, können unterschiedliche SQL-Statements abgefragt werden.

SQLInjectionConnection

Der Aufbau dieser Klasse ähnelt der DBConnection Klasse. Allerdings liegt der Unterschied darin, dass die Session einer Verbindung solange geöffnet bleibt bis sich ein Benutzer abmeldet. Diese Klasse wird von dem PWInjectionController aufgerufen und übernimmt alle anmelderelevanten Informationen. Schließlich wird das Ergebnis wieder an die PWInjectionController Klasse retourniert.

ProgressBarController

Der ProgressBarController ist verantwortlich für das Beladen jeder vorhandenen ProgressBar in der Anwendung. Die ProgressBar ist in den folgenden Klassen auffindbar: PWInjectionController, DBController, RemoteController, BruteForceController und im ClientController. Sobald die Methode loadProgress aufgerufen wird, startet das Laden der ProgressBar. Dabei wird in einem Thread ein counter solange erhöht, bis er eine maximale Anzahl erreicht hat. Mit der Methode stopProgress kann definiert werden, wann die ProgressBar nicht mehr weiter laden soll.

AlertController

Innerhalb dieser Klasse sind mehrere Methoden mit unterschiedlichen Fehlermeldungen auffindbar. Je nach Fehler, kann eine Klasse aus der Anwendung eine beliebige Methode aus dem AlertController aufrufen. Diese Klasse ist für Hinweise, Warnungen und Fehlermeldungen ausschlaggebend.

SYNController

Die Funktion dieser Klasse bezieht sich auf die Simulierung der SYN-Flooding Bedrohung. Sobald der FXMLDocumentController die Klasse und dazugehörige View aufruft, können die IP-Adresse und der Port des Zielhosts angegeben werden. Daraufhin wird der SYNController gestartet, der wiederum die innerhalb befindliche Klasse start aufruft. Die Aufgabe hier ist es unzählige Threads, in der ebenfalls innerhalb befindlichen Klasse ClientThread, zu starten. Zeitgleich werden wahllos viele Verbindungen geöffnet mit dem Ziel, den Datenbanklistener zu überlasten und somit keine Verbindungen mehr zuzulassen. Allerdings leidet darunter die Performance der Anwendung sehr, weshalb eine externe Anwendung zur Auswertung dieser Bedrohung erstellt wurde. Das Beenden der Threads kann durch eine so enorme Anzahl auch etwas Zeit in Anspruch nehmen.

ModelContainer

Die ModelContainer Klasse beinhaltet alle Attribute die innerhalb der Anwendung benötigt werden. Sie besteht ausschließlich aus Getter und Setter Methoden und dient zur Verwaltung und Aktualisierung der Daten wie Benutzernamen, Userhost, Verbindungsstatus oder Returncode. Diese Eigenschaften werden nahezu in jeder Klasse der Anwendung verwendet und sind für einen reibungslosen Ablauf entscheidend.

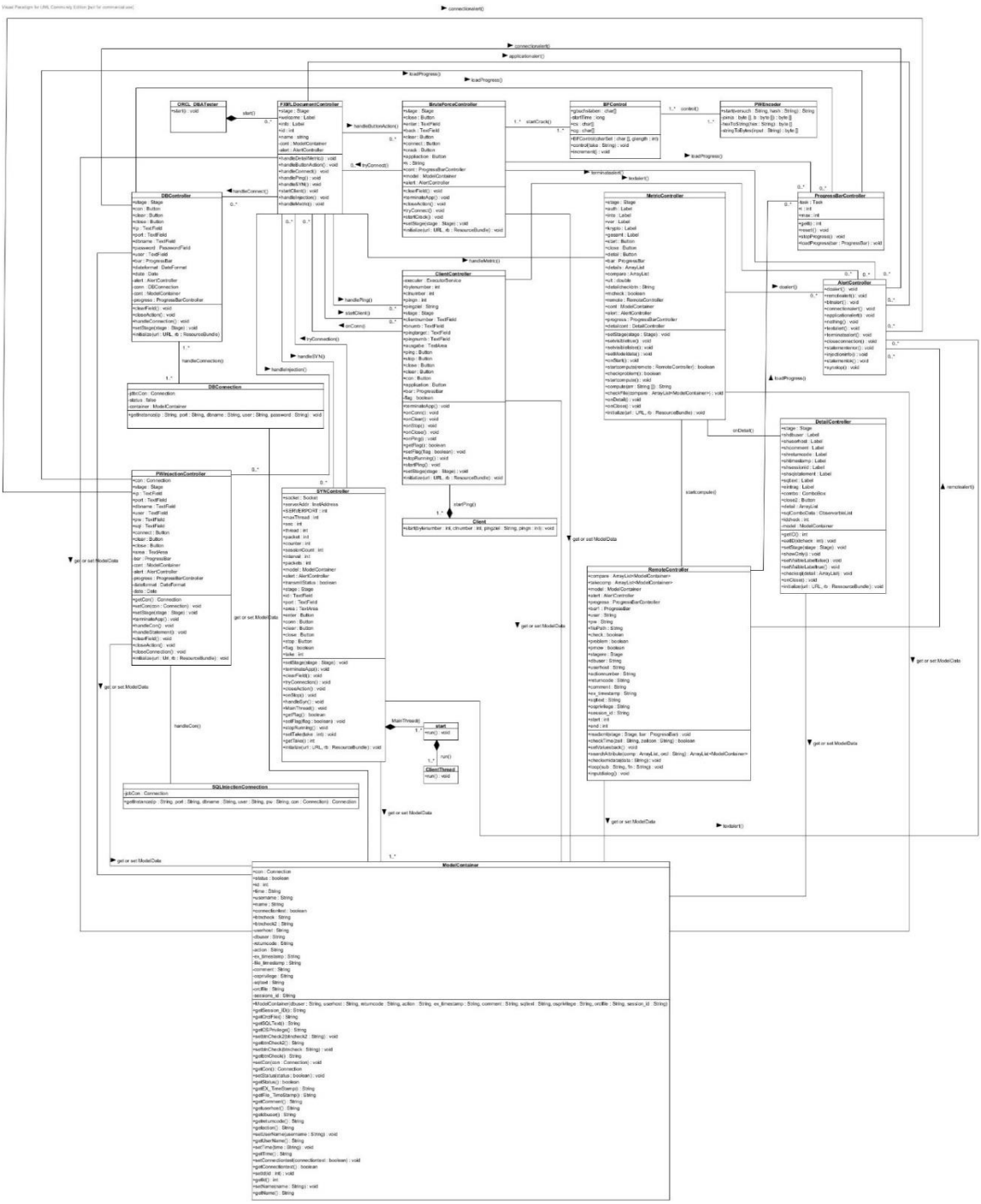


Fig. 15: Klassendiagramm des erstellten Frameworks

8.2 Metamodell

Modellierungsmethoden bestehen aus der Modellierungstechnik und Mechanismen. Weiters wird die Modellierungstechnik auf eine Modellierungssprache und Modellierungsprozedur aufgeteilt. Durch die Modellierungssprache werden die Mechanismen definiert. Folglich wird die Modellierungssprache durch die Syntax, Semantik und Notation beschrieben. Um Ergebnisse erreichen zu können, definiert die Modellierungsprozedur alle angewendeten Schritte einer Modellierungssprache [19].

Ein Metamodell kann als Modell einer Modellierungssprache definiert werden. Weiters wird zur erfolgreichen Erstellung eines Metamodells eine Metamodellierungssprache benötigt. Durch den Einsatz eines Metamodells kommt es zu einer Hierarchie von Metasprachen. Somit wird das Modell von einem Metamodell definiert, das Metamodell von einem Meta-Metamodell und weiter fortlaufend.

Zur Modellierung der Anwendungsfunktionalitäten wurde, wie vorher beschrieben das Klassendiagramm verwendet. Zusätzlich zu dieser Arbeit ist ein entwickeltes Metamodell vorhanden. Somit legt es die Struktur des Modells fest, die sich beispielsweise auf die Klasse, das Attribut, den Typen oder die Methode beziehen. Letztlich sind alle bestehenden Komponenten des Klassendiagramms Instanzen des aufgestellten Metamodells.

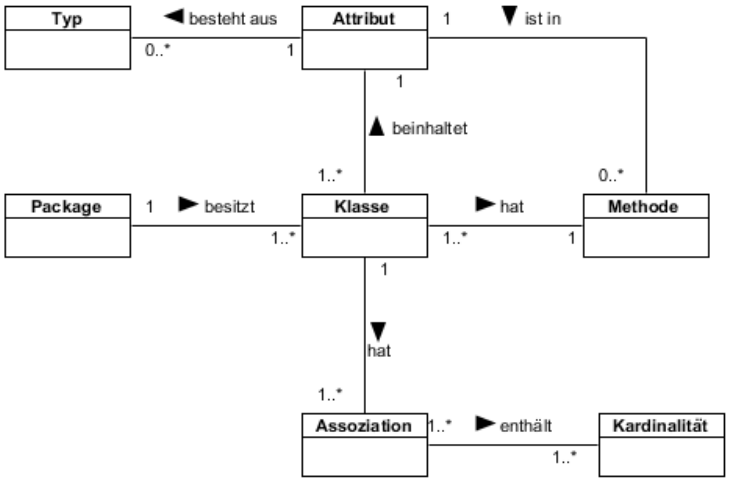


Fig. 16: Metamodell des erstellten Frameworks

8.3 Sprache & Toolunterstützung

Zur Umsetzung der bestehenden Funktionalitäten wurde Java in Kombination mit JavaFX verwendet. Der Einsatz von Java ist insofern vorteilhaft, als das es beispielsweise plattformunabhängig ist und eine Menge Application Programming Interfaces (APIs) zur Umsetzung von Netzwerkkommunikationen oder Remote Services unterstützt. Die Verwendung von JavaFX ermöglicht die Gestaltung einer modernen Anwendungsoberfläche und ersetzt somit das veraltete Java Swing. Mit Hilfe des JavaFX Scene Builder Tools können Oberflächen einfach per drag and drop erstellt und durch die Vergabe von IDs in der Entwicklungsoberfläche NetBeans verwendet werden. Somit konnten, wie in der abgebildeten Grafik, unterschiedliche Elemente reibungslos eingebunden und genutzt werden.

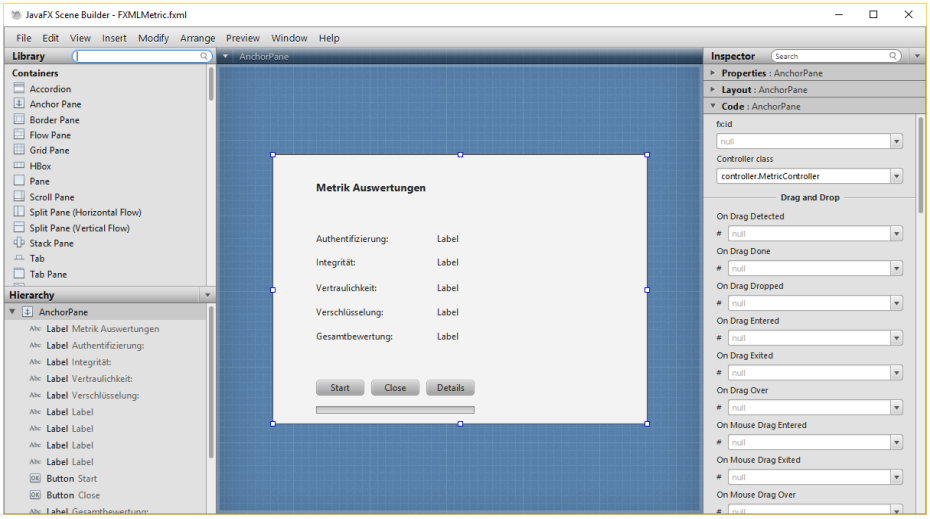


Fig. 17: JavaFX Scene Builder

Vor der Implementierung musste eine Testumgebung generiert werden, sodass mit Hilfe der Oracle VM Virtual Box und dem Betriebssystem Windows 7 eine virtuelle Maschine erstellt wurde. Daraufhin wurde die Oracle 11g Datenbank installiert und konfiguriert. Oracle bietet weit verbreitete relationale Datenbanken mit einer Menge an Informationen an. Damit innerhalb eines Netzwerks ein Zugriff auf die Datenbank erfolgen kann, muss der Datenbanklistener eine IP-Adresse und einen Port beinhalten. Oracle stellt hierfür die Anwendung NetManager bereit, in der IP- und Port-Einstellungen überprüft und adaptiert werden können.

Weiters erwies sich die Software Visual Paradgim für die Erstellung der Modelle als hilfreich, weil der Vorteil dieses Tools die einfache Handhabung sowie die Unterstützung einer großen Anzahl von Diagrammen ist.

8.4 Implementierung

Durch die Entwicklung eines Frameworks hat der Benutzer die Möglichkeit Datenbankbedrohungen zu simulieren und im Anschluss anhand der Security Metrics auszuwerten und zu bewerten. Im Verlauf der Implementierung mussten einige Problemstellungen bewältigt werden, die eine Reihe an notwendigen Recherchen erforderten, um positive Resultate zu erzielen. Dabei war es notwendig zu identifizieren, wie Simulationen und Messungen der bestehenden Bedrohungen ausgeführt werden können. Die folgende Literaturrecherche [44] führte zu dem in Oracle definierten Audit Trail. Mit Hilfe der Audit Trails konnten alle Aktionen eines Benutzers dokumentiert werden. Jeder Audit Eintrag wird in einer XML-Datei in einem definierten Ordner gespeichert. Daraufhin konnte auf die Dateien im Ordner innerhalb des Netzwerkes zugegriffen werden. Die folgende Abbildung ist ein Auszug des Code-Ausschnitts für die Erstellung einer Verbindung zu einem Ordner innerhalb eines Netzwerkes. Dazu ist die Eingabe des Benutzernamens, sowie des Passworts, als auch die Angabe des Ordnerpfads erforderlich. Ein weiterer Vorteil von Oracle ist die Verfügbarkeit eines Rootdokuments, in dem die Dateinamen der XML-Dateien nach Aktualität abgespeichert sind. In Figur 17 wird in der Schleife nach dem Rootdokument gesucht, in dem die aktuellsten Einträge gespeichert sind.

```
74              UserAuthenticator auth=new StaticUserAuthenticator("",user,pw);
75              FileSystemOptions opts=new FileSystemOptions();
76              DefaultFileSystemConfigBuilder.getInstance().setUserAuthenticator(opts, auth);
77              //"\\\\ADMIN-PC\\adump"
78              FileObject fo = VFS.getManager().resolveFile(filePath,opts);
79              FileObject k = null;
80              ////////////////////
81              for (FileObject children : fo.getChildren()) {
82                  String name = children.getName().getBaseName();
83                  if (name.equals("adx_orcl.txt")) {
84                      k = children;
85                  }
86              }
```

Fig. 18: Code Auszug aus NetBeans zur Verbindung mit der virtuellen Maschine

Für die Umsetzung der Brute-Force Bedrohung waren weitere Recherchen erforderlich. Der Algorithmus ist weit verbreitet und bietet praktische Lösungsansätze. Die Problematik liegt hier, einen Zusammenhang zwischen den Passwörtern der Benutzer und dem Algorithmus zu entwickeln. In [59] wird eine SQL-Abfrage beschrieben, die den Benutzer und sowie das dazugehörige Passwort in Oracles Standardverschlüsselungsalgorithmus SHA1 ausgibt. Die Verschlüsselungen des Algorithmus bestehen aus einem 60 Zeichen langen Hash. Durch die Zuhilfenahme eines Saltwertes soll die Stärke der Verschlüsselung erhöht werden. In diesem Fall erweist sich der Saltwert als Nachteil, weil in jedem verschlüsselten Passwort die letzten 20 Zeichen des Hashwertes dafür reserviert sind. Sobald der richtige generierte Klartext mit dem Salt durch den SHA1-Algorithmus verschlüsselt wird, ergibt es den gleichen ursprünglich verschlüsselten Hashwert des Passwortes. Somit können mehrere Iterationen durchgeführt werden, bis die richtig generierte Zeichenkombination das Klartext Passwort ergibt.

Die folgende Abbildung repräsentiert die Klasse PWEncoder, die zunächst die generierte Zeichenkombination und den angegebenen Hashwert übernimmt. Weiterhin werden die letzten 20 Zeichen des Hashwertes gespeichert, die den Salt darstellen. Anschließend werden der Saltwert und die Passworteingabe in Bytes umgewandelt sowie zusammengelegt. Letztlich werden die Bytes mittels SHA-1 verschlüsselt und mit dem ursprünglichen Hashwert auf eine Übereinstimmung überprüft.

```
15  public class PWEncoder {
16
17      public  String start(String versuch, String hash) throws NoSuchAlgorithmException{
18              String hashString = hash.substring(40, 60).toLowerCase();
19
20              byte[] key = hashToString(hashString);
21              String some = versuch;
22              MessageDigest md = MessageDigest.getInstance("SHA1");
23              byte[] input = combine(stringToBytes(some), key);
24              String checksum = new BigInteger(1, md.digest(input)).toString(16);
25
26              return checksum;
27          }
```

Fig. 19: Code Auszug der Klasse PWEncoder aus NetBeans

Im weiteren Verlauf lag der Fokus auf einer effektiven SQL-Injection Simulation. Die Funktionalitäten wurden so implementiert, dass zuerst eine Datenbankverbindung erforderlich ist, um anschließend SQL-Abfragen tätigen zu können. Die Entwicklung der Verbindungsüberprüfung verlief nach mehreren Testphasen positiv, während sich die Umsetzungen der SQL-Abfragen als problematisch erwiesen. In diesem Fall mussten Statement Unterschiede zwischen select, insert, update, create oder delete definiert werden. Folglich war die Einbeziehung von Metadaten unumgänglich, weil dadurch die Spaltenanzahl der Tabellen dynamisch ausgewertet werden konnten. Letztlich konnten alle Spalten und Zeilen, die zur Ausgabe notwendig sind, in einer Map gespeichert werden.

```
    @FXML
    public void handleStatement(ActionEvent event){
        Connection conn = getCon();
        String correct = sql.getText().toLowerCase();
        if ((conn != null)) {
            if (sql.getText().equals("")) {
                alert.textalert();
            } else if (correct.startsWith("create") || correct.startsWith("insert") || correct.startsWith("update")
                    || correct.startsWith("delete")) {
                progress.loadProgress(bar);
                Statement s;
                try {
                    s = conn.createStatement();
                    s.executeUpdate(sql.getText());
                    progress.stopProgress();
                    alert.statementok();
                } catch (SQLException ex) {
                    progress.stopProgress();
                    alert.statementerror();
                    Logger.getLogger(PWInjectionController.class.getName()).log(Level.SEVERE, null, ex);
                }
            } else {
                progress.loadProgress(bar);
                cont.setbtnCheck2("clicked");
                List<Map<String, Object>> rows;
                Statement s;
                try {
                    s = conn.createStatement();
                    ResultSet result = s.executeQuery(sql.getText());
                    rows = new ArrayList<>();
                    ResultSetMetaData meta = result.getMetaData();
                    int column = meta.getColumnCount();
                    while (result.next()) {
                        Map<String, Object> columns = new LinkedHashMap<>();

                        for (int i = 1; i <= column; i++) {
                            columns.put(meta.getColumnLabel(i), result.getObject(i));
                        }
                        rows.add(columns);
                    }

                    Platform.runLater(() -> {
                    area.clear();
                    rows.stream().forEach((map) -> {
                        map.entrySet().stream().forEach((entry) -> {
                            if (entry.getKey() == null || entry.getValue() == null) {
                                String k = entry.getKey();
                                String v = "Kein Eintrag!";
                                area.appendText(k + " " + v + "\n");
                            } else {
                                String k = entry.getKey();
                                String v = entry.getValue().toString();
                                area.appendText(k + " " + v + "\n");
                            }
                        });
                    });
                    });
                    progress.stopProgress();
                    } catch (SQLException ex) {
                        progress.stopProgress();
                        alert.statementerror();
                        Logger.getLogger(PWInjectionController.class.getName()).log(Level.SEVERE, null, ex);
                    }
                }
            } else {
                alert.btnalert();
            }
        }

```

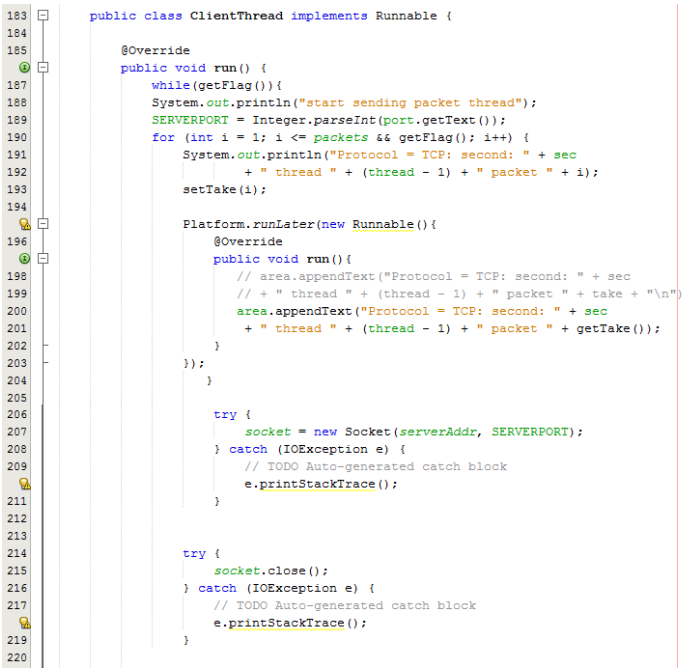
Fig. 20: Code Auszug zur SQL-Injection Simulation aus NetBeans

Die Ping of Death Simulation erforderte ähnliche Recherchen wie die bisherigen Bedrohungen. Zu diesem Thema existieren reichlich theoretische Ressourcen mit geringen praktischen Beispielen. Deshalb waren für die Erreichung angemessener Resultate mehrere Testphasen erforderlich. Mit Hilfe von Executors war die Simulation einer bestimmten Anzahl von Clients möglich. Durch Executors wird ein Threadpool erstellt, der anschließend eine definierte Pinganzahl versendet. Nach einer Anzahl von Testdurchläufen erwies sich die Beendigung einer hohen Threadanzahl als problematisch, weil der Abbruch von Threads Zeit in Anspruch nimmt.

```
    public void startPing(){
       executor = Executors.newFixedThreadPool(clnumber);
        //Client client = new Client();
        for(int i = 0; i < clnumber && getFlag(); i++){
            executor.execute(new Runnable(){
                @Override
                public void run(){
                    try {
                        new Client().start(bytenumber,clnumber,pingziel,pingn);
                    } catch (IOException ex) {
                        Logger.getLogger(ClientController.class.getName()).log(Level.SEVERE, null, ex);
                    }
                }
            });
        }
    }

public class Client{

public void start(int bytenumber, int clnumber, String pingziel, int pingn) throws IOException {

             for(int i = 0; i < pingn && getFlag(); i++){
              final IcmpPingRequest request = IcmpPingUtil.createIcmpPingRequest();
              request.setHost(pingziel);
              request.setPacketSize(bytenumber);
              final IcmpPingResponse response = IcmpPingUtil.executePingRequest(request);
              final String formatResponse = IcmpPingUtil.formatResponse(response);
              Platform.runLater(new Runnable(){
                @Override
                public void run(){
                        cont.stopProgress();
                        ausgabe.appendText(formatResponse + "\n");
                }
              });
              System.out.println(formatResponse);
             }
  }
}
```

Fig. 21: Code Auszug der Ping of Death Simulation aus NetBeans

Für die Umsetzung der SYN-Flooding Simulation, ist ebenso die Arbeit mit Threads erforderlich. Durch die Erstellung einer hohen Anzahl an Threads, konnte eine potenzielle Gefahr simuliert werden. Zu diesem Zweck wird in der Klasse SYNConroller die Klasse start mit einer großen Anhäufung an Threads gestartet. Daraufhin wird eine weitere Klasse Client Thread gestartet, die ebenfalls eine enorme Menge an Threads erstellt und mit Hilfe der Sockets massenhaft Verbindungen öffnet. So wird es sichergestellt, dass der Datenbanklistener innerhalb kurzer Zeit eine große Menge an Verbindungsanfragen erhält. Folglich kann der Listener nicht alle Verbindungen verarbeiten, wodurch ein Zusammenbruch verursacht wird. Dadurch entstehen enorme Performanceeinbrüche, weil sehr viele Ressourcen beansprucht werden. In dieser Simulation ist das Beenden der Threads problematisch, weil eine dementsprechend große Anzahl nicht unverzüglich abgebrochen werden kann. Die folgende Abbildung repräsentiert den relevanten Code-Teil für die Eröffnung der Verbindungen.



```
183     public class ClientThread implements Runnable {
184
185         @Override
187         public void run() {
188             while(getFlag()){
189             System.out.println("start sending packet thread");
190             SERVERPORT = Integer.parseInt(port.getText());
191             for (int i = 1; i <= packets && getFlag(); i++) {
192                 System.out.println("Protocol = TCP: second: " + sec
193                         + " thread " + (thread - 1) + " packet " + i);
194                 setTake(i);
196
198                 Platform.runLater(new Runnable(){
199                     @Override
200                     public void run(){
201                        // area.appendText("Protocol = TCP: second: " + sec
202                        // + " thread " + (thread - 1) + " packet " + take + "\n");
203                        area.appendText("Protocol = TCP: second: " + sec
204                         + " thread " + (thread - 1) + " packet " + getTake());
205                     }
206                 });
207                    }
208
209                     try {
211                         socket = new Socket(serverAddr, SERVERPORT);
212                     } catch (IOException e) {
213                         // TODO Auto-generated catch block
214                         e.printStackTrace();
215                     }
216
217
219                 try {
220                     socket.close();
                 } catch (IOException e) {
                     // TODO Auto-generated catch block
                     e.printStackTrace();
                 }
```

Fig. 22: Code Auszug der SYN-Flooding Simulation aus NetBeans

Nachdem alle Bedrohungen getestet wurden, mussten im weiteren Verlauf die Auswertung und Bewertung geplant werden. Dabei war die Einbindung von Flags erforderlich, um jede Bedrohung adäquat bewerten zu können. Dadurch konnte genau identifiziert werden, welche Bedrohung zu welchem Zeitpunkt ausgeführt wurde. Ebenso relevant war die Analyse der Zeitstempel, die dazu dienen, die notwendigen Audit Einträge auszuwerten. Hierfür wird überprüft, ob die erstellte Datei in einem einminütigen Abstand vor oder nach dem Zeitstempel erstellt wurde. Der folgende Code Teil repräsentiert das eben beschriebenen Szenario.

```
180     public boolean checkTime(String zeit, String zeitcon) throws ParseException{
181
182             Date d = new SimpleDateFormat("HH:mm").parse(zeit);
183             Calendar calafter = Calendar.getInstance();
184             calafter.setTime(d);
185             calafter.add(Calendar.MINUTE, 1);
186             String k = calafter.getTime().toString().substring(11, 16);
187             Date after = new SimpleDateFormat("HH:mm").parse(k);
188
189             Date d2 = new SimpleDateFormat("HH:mm").parse(zeit);
190             Calendar calbefore = Calendar.getInstance();
191             calbefore.setTime(d2);
192             calbefore.add(Calendar.MINUTE, -1);
193             calbefore.add(Calendar.DATE, 1);
194             String k1 = calbefore.getTime().toString().substring(11, 16);
195             Date before = new SimpleDateFormat("HH:mm").parse(k1);
196
197             Date d3 = new SimpleDateFormat("HH:mm").parse(zeit);
198             Date gleich = new SimpleDateFormat("HH:mm").parse(zeitcon);
199             boolean checkdate = false;
200
201
202             if((before.getTime() <= gleich.getTime() && after.getTime() >= gleich.getTime())
203                || gleich.getTime() == d3.getTime()){
204                 checkdate = true;
205             }
206             return checkdate;
207     }
```

Fig. 23: Code Auszug des Zeitvergleiches aus NetBeans

Sobald der Zeitstempel, des erstellten Audit Eintrages mit dem Verbindungszeitpunkt übereinstimmt, wird der Eintrag näher betrachtet. In diesem Zusammenhang war die Änderung der Zeitzone problematisch, weil eine Lösung entwickelt werden musste, um die UTC Zone der Audit Einträge in die UTC+1 Zone umzuwandeln. Somit musste zur UTC Zone zwei Stunden dazu gerechnet werden, um eine Überprüfung zu ermöglichen. Anschließend werden in der MetricController Klasse alle gespeicherten Attribute überprüft. Die Bewertung und Auswertung wird gestartet, wenn die eben erwähnten Attribute mit den ursprünglichen Attributen der Verbindungsüberprüfung übereinstimmen. Letztlich besteht die Möglichkeit alle Informationen im Detail aufzurufen. Dabei wurde zusätzlich die Detailansicht an der SQL-Injection Simulation mit Comboboxen angepasst, sodass jedes ausgeführte Statement angezeigt werden kann.

```
142     @Override
143     public void initialize(URL url, ResourceBundle rb) {
144         combo.setItems(sqlcomboData);
145         //combo.getItems().addAll(sqlcomboData);
146         combo.setOnAction( e ->{
147             setVisibleLabeltrue();
148             int count = 0;
149             for(ModelContainer detail1: detail){
150                 count++;
151                 String c = String.valueOf(count);
152                 if(combo.getValue().equals(c)){
153                 shdbuser.setText(detail1.getdbuser());
154                 shuserhost.setText(detail1.getuserhost());
155                 shreturncode.setText(detail1.getreturncode());
156                 if(!detail1.getComment().equals("")){
157                 shcomment.setText(detail1.getComment().substring(0, 26) + "\n" + detail1.getComment().substring(28));
158             // + "\n" + comment.substring(53, 65)
159                 }else{
160                 shcomment.setText("");
161                 }
162                 //shcomment.setText(detail1.getComment());
163                 shtimestamp.setText(detail1.getEX_TimeStamp());
164                 shsessionid.setText(detail1.getSession_ID());
165                 shsqlstatement.setText(detail1.getSQLText());
166                 }
167             }
168                 }
169                 );
170     }
```

Fig. 24: Code Auszug zur detaillierten Auswertung der Security Metrics aus Net Beans

9 Case Study

9.1 Auswertung der Case Study

Die Funktionalitäten und Oberflächen der Anwendung wurden mit Hilfe von Java und JavaFX erstellt. Durch die Kombination der Entwicklungssprachen wird eine einfache Benutzerinteraktion realisiert, die das Simulieren und Auswerten von Datenbankbedrohungen ermöglicht. Nach dem Anwendungsstart erscheint eine benutzerfreundliche Oberfläche mit einer Menüleiste und vier Menüpunkten. Unter dem ersten Punkt “Action“ stehen 4 Unterkategorien zur Auswahl, die in Figur 26 ersichtlich sind. Hier kann eine beliebige Datenbankbedrohung gestartet werden. Im zweiten Menüpunkt “Verbindung“ besteht die Möglichkeit eine Verbindungsüberprüfung mittels Ping durchzuführen. Zu diesem Zweck muss die Zieladresse angegeben werden. Infolgedessen kommt es durch den Aufruf des dritten Menüpunktes “Security Metrics“ zu einer Auswertung und Bewertung einer Bedrohungssimulation. Allerdings kann dieser Menüpunkt erst gestartet werden, wenn eine Datenbankbedrohung simuliert wurde. Schließlich können unter dem Menüpunkt “Help“ weitere Informationen zur der Anwendung entnommen werden.

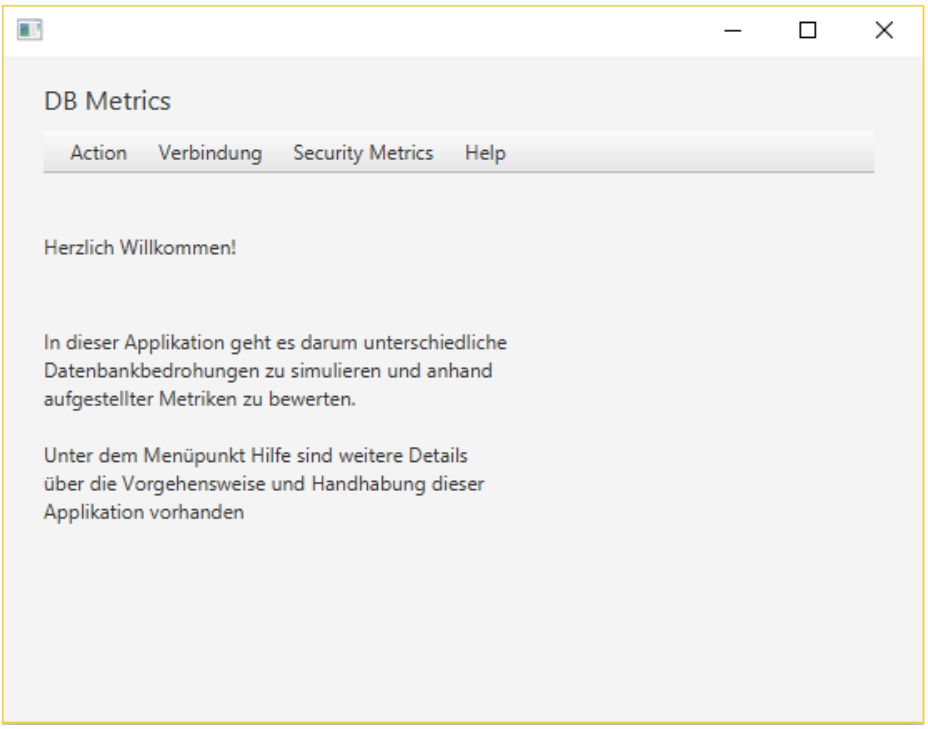


Fig. 25: Startoberfläche der entwickelten Anwendung

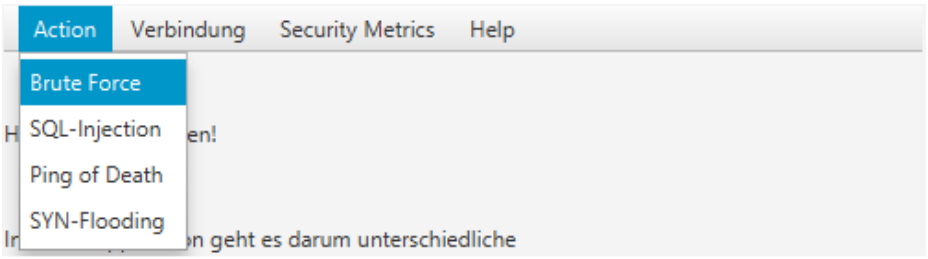


Fig. 26: Auswahl der möglichen Bedrohungen

Der zweite Menüpunkt "Verbindung" ist bis auf eine eingeschränkte Eingabemöglichkeit ident mit der Oberfläche für die Ping of Death Simulation. Für den Zweck der Verbindungsüberprüfung kann nur die Zieladresse eingegeben werden. Nach dem Starten beginnt die Ausführung des Pings.

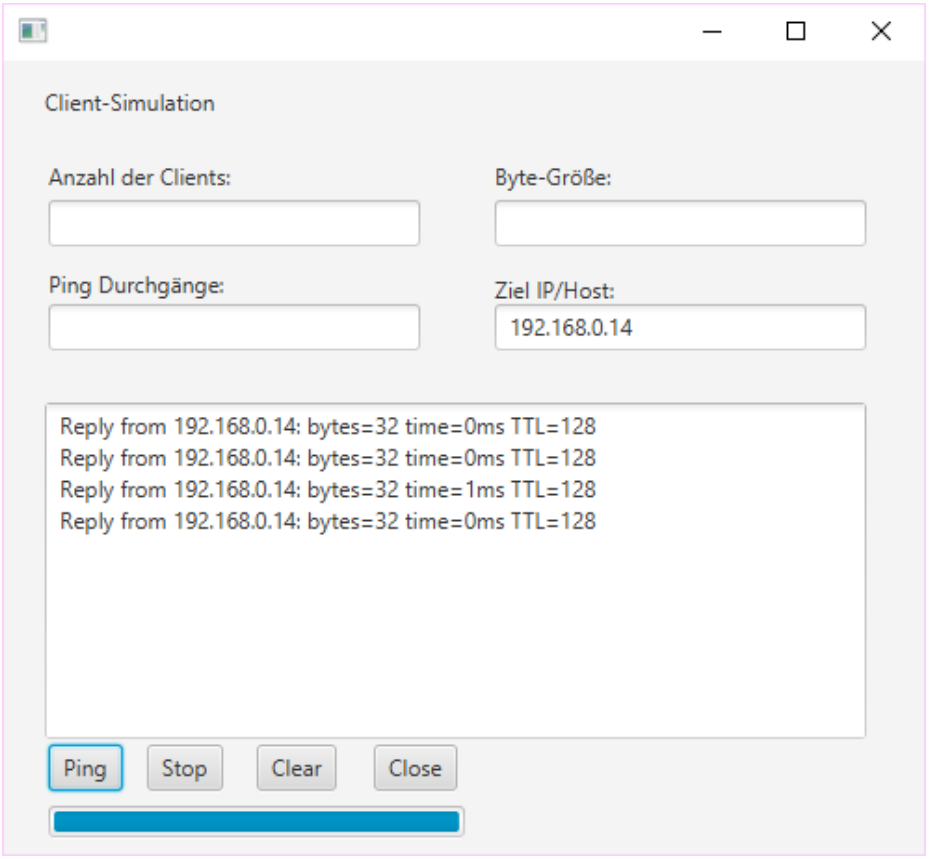


Fig. 27: Verbindungsüberprüfung anhand eines Pings

9.1.1 Brute-Force Simulation

Sobald unter dem Menüpunkt “Action“ die Brute-Force Simulation ausgewählt wird, erscheint, wie in Figur 28 ersichtlich die Anwendungsoberfläche. Für den Start der Simulation, ist die Eingabe eines 60 Zeichen langer Hashwert erforderlich. Andernfalls erscheint eine Fehlermeldung. Die Simulation ist dann beendet, wenn der Hashwert in einen Klartext entschlüsselt wurde. Daraufhin kann ein kann ein Anmeldetest durchgeführt werden. Weiters besteht die Möglichkeit, die Anwendung vorzeitig zu verlassen und eine andere Bedrohung zu starten.

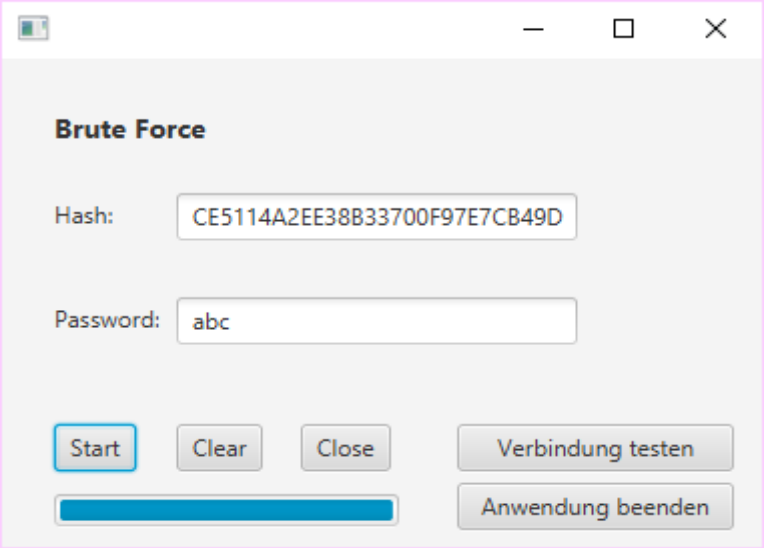


Fig. 28: Brute-Force Simulation

Nachdem das Passwort entschlüsselt und der Username bekannt ist, kann ein Anmeldetest durchgeführt werden. Dazu müssen die IP-Adresse des Datenbanklisteners, der Port und der Datenbankname angeführt sein.

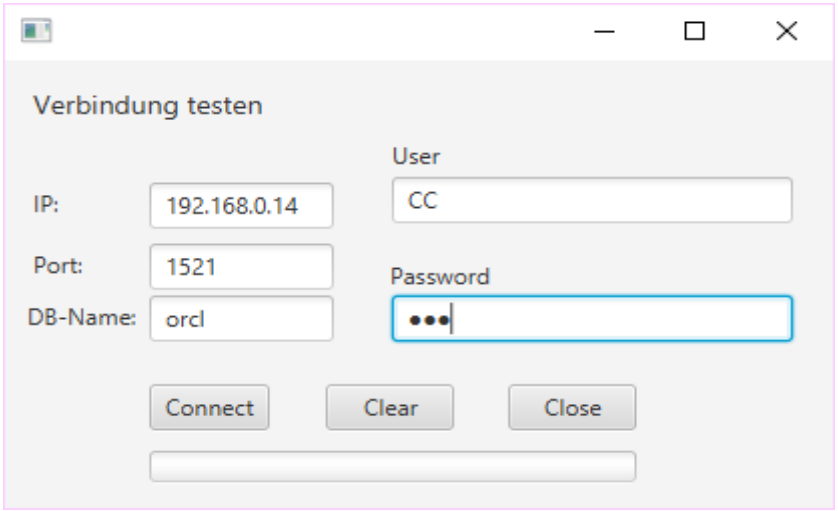


Fig. 29: Anmeldeversuch nach der Brute-Force Simulation

Sowohl bei einer erfolgreichen als auch nicht erfolgreichen Anmeldung erscheint eine Anmeldeinformation. Nach der in Figur 29 abgebildeten Eingabe, wird in Figur 30 die Anmeldebestätigung einer erfolgreichen Verbindung veranschaulicht.

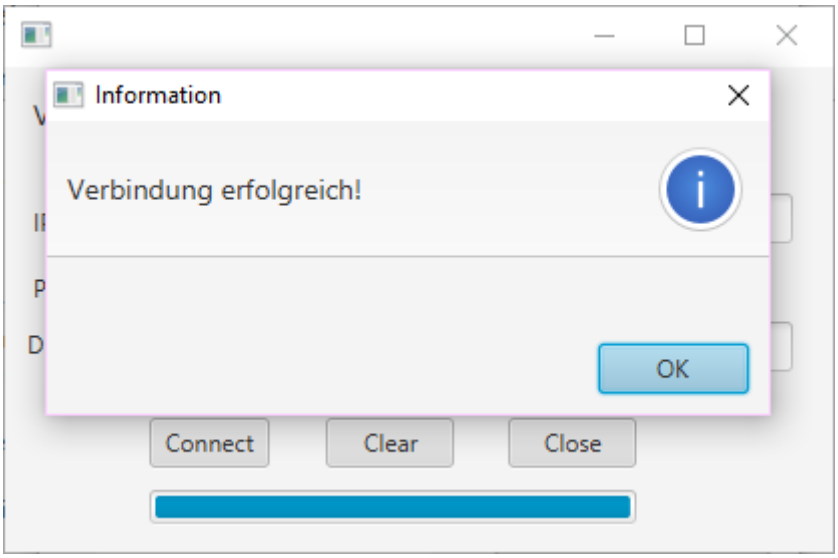


Fig. 30: Positive Anmeldebestätigung nach der Brute-Force Simulation

Unter dem Menüpunkt “Metrics“ erfolgt die Auswertung und Bewertung der Bedrohungssimulation. Daraufhin erscheint, wie in Figur 31 ersichtlich, die Anwendungsoberfläche mit den Sicherheitszielen Verfügbarkeit, Authentifizierung, Integrität, Vertraulichkeit und Verschlüsselung. Für die Verbindung mit der virtuellen Maschine ist, wie in Figur 32 abgebildet, eine Anmeldung notwendig. Dafür ist die Angabe des Benutzers, Passwortes und Ordnerpfades erforderlich. Sobald die Anmeldung positiv verläuft, starten die Funktionalitäten des Programmes, indem auf die 10 aktuellsten Audit Einträge zugegriffen und folglich die Attribute ausgewertet werden.

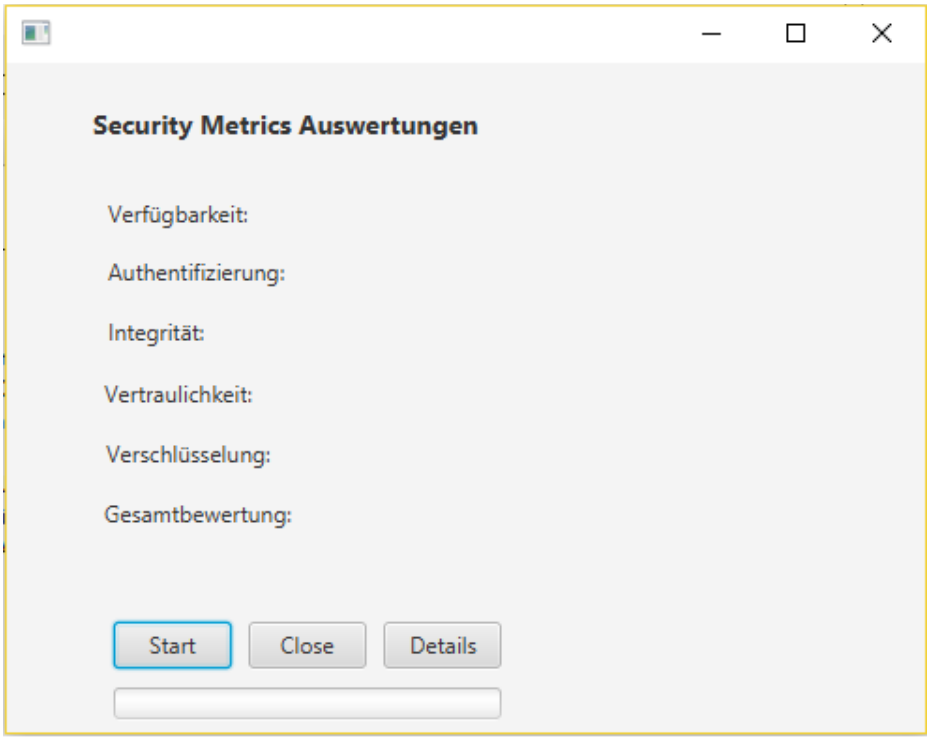


Fig. 31: Anwendungsoberfläche der Security Metrics

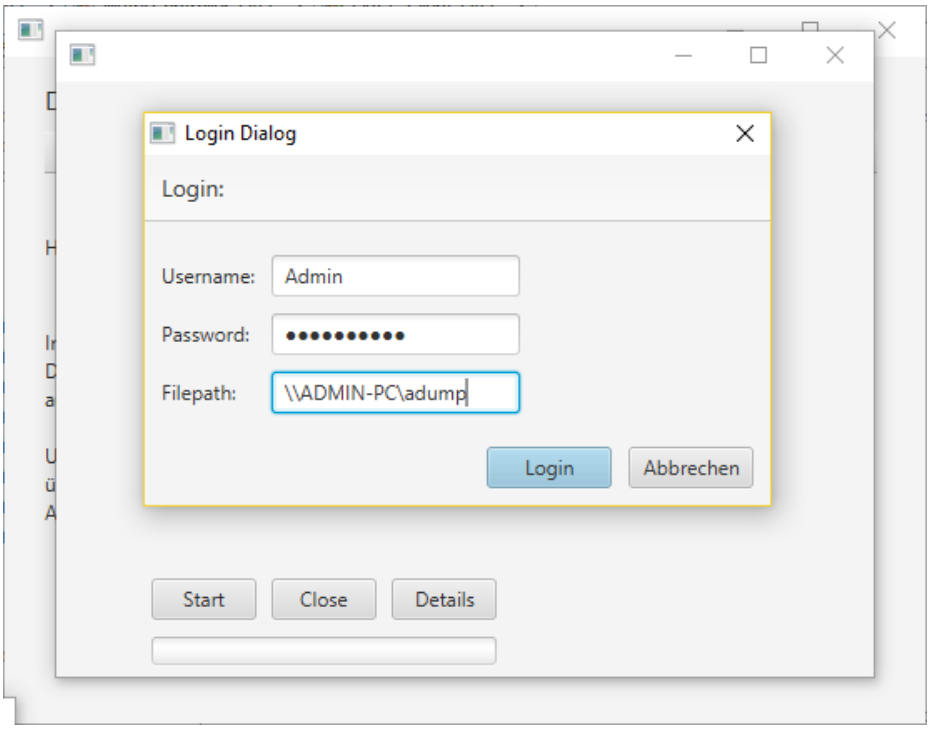


Fig. 32: Anmeldung für den Zugriff auf die virtuelle Maschine durch die Eingabe der notwendigen Attribute

Die Brute-Force Simulation verlief in diesem Szenario erfolgreich. Durch eine Anmeldeüberprüfung wurde demonstriert, dass der angegebene Hashwert das korrekt entschlüsselte Passwort zum Benutzer enthielt. Dadurch werden die Sicherheitsziele folgendermaßen bewertet. Die Authentifizierung und Verfügbarkeit sind in diesem Fall gut, weil die Anmeldung durch die Eingabe des Benutzers und Passwortes richtig verlief sowie der Systemzugriff nicht beeinflusst wurde. Weiters wird die Vertraulichkeit als durchschnittlich eingestuft, weil ein Datenbankbenutzer mit SYSTEM-Rechten die benötigten Hashwerte für die Simulation aufspüren und somit alle Passwörter entschlüsseln könnte. Ebenfalls durchschnittlich eingestuft ist die Integrität, weil die Gefahr zur Manipulation und zum Diebstahl der Daten besteht und folglich keine konsistenten Datenzustände abgerufen werden können. Die Verschlüsselung ist der Kern dieser Simulation. Die Brute-Force Attacke konnte erfolgreich durchgeführt werden, wodurch die Verschlüsselung als schlecht bewertet wird. Letztlich ergibt die Gesamtbewertung 60 %.

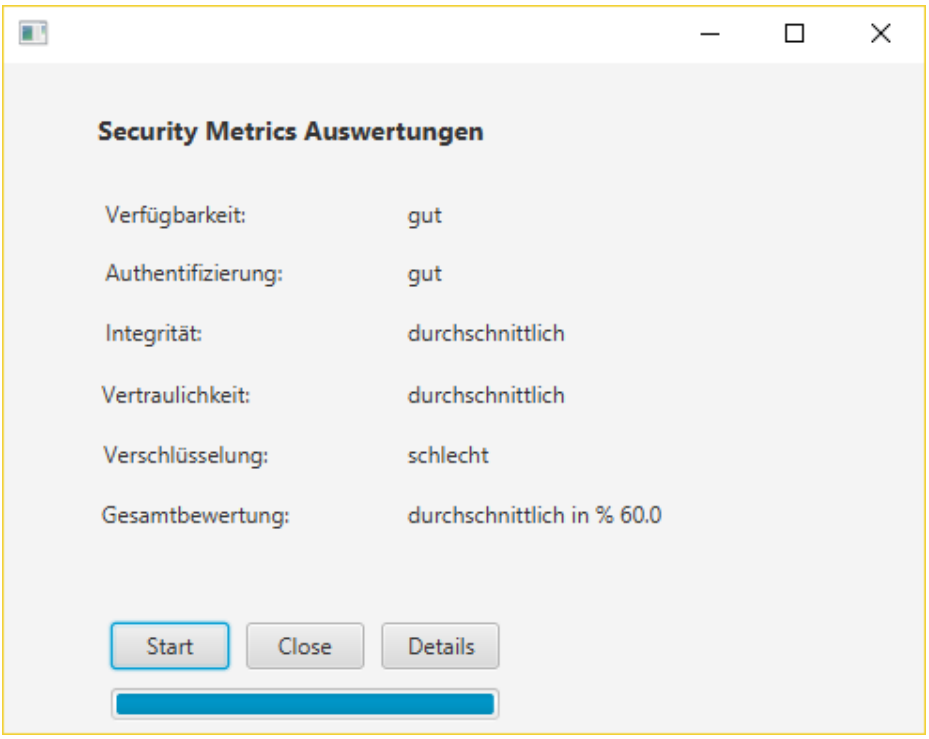


Fig. 33: Auswertung und Bewertung der Sicherheitsziele nach der Brute-Force Simulation

In weiterer Folge können detaillierte Informationen zur Auswertung abgerufen werden. Diese beinhalten den Usernamen, Userhost, Returncode sowie die Quell-IP-Adresse und den Zeitstempel. Die Attribute sollen einer besseren Interpretation der Simulationsresultate dienen, um zukünftige Bedrohungen effektiv zu ermitteln.

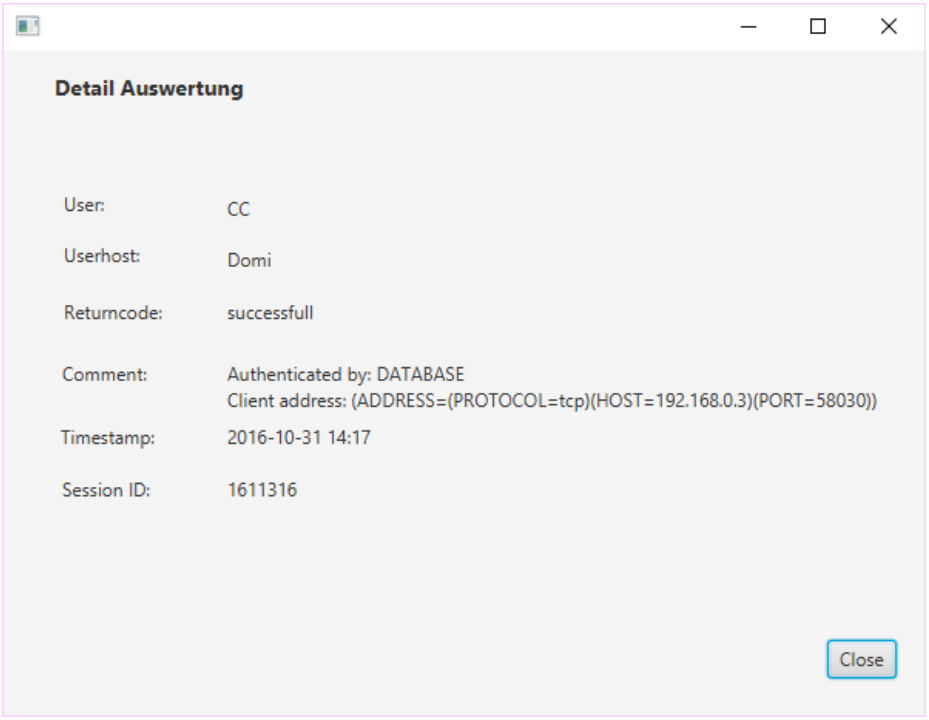


Fig. 34: Detailansicht der ausgewerteten Brute-Force Simulation

Die Simulation zu dieser Bedrohung kann als erfolgreich bewertet werden. Damit ein Datensatz eindeutig klassifiziert werden kann, darf der Zeitstempel des Audit Eintrags nur innerhalb eines einminütigen Intervalls liegen, weil nur so die Eindeutigkeit des Datensatzes sichergestellt werden kann. In manchen Fällen werden die Audit Einträge verzögert angelegt, wodurch keine Auswertungen durchgeführt werden. Somit können Kollisionen mit anderen Datensätzen vermieden werden.

9.1.2 SQL-Injection Simulation

Die SQL-Injection Bedrohung wird unter dem Menüpunkt “Action“ gestartet. Daraufhin erscheint die Anwendungsoberfläche mit der Aufforderung, sich mit der Datenbank zu verbinden. Hier besteht die Möglichkeit unterschiedliche SQL-Injection Anmeldevarianten, wie die folgenden Zeichenkombinationen ‘ OR 1=1‘ oder ‘“ OR “=‘, zu testen. Sobald die Anmeldung erfolgreich ist, können SQL-Abfragen sowie SQL-Injection Statements durchgeführt werden. Für eine erfolgreiche Anmeldung müssen die IP-Adresse, der Port, Datenbankname, Benutzername und das Passwort richtig angegeben werden.

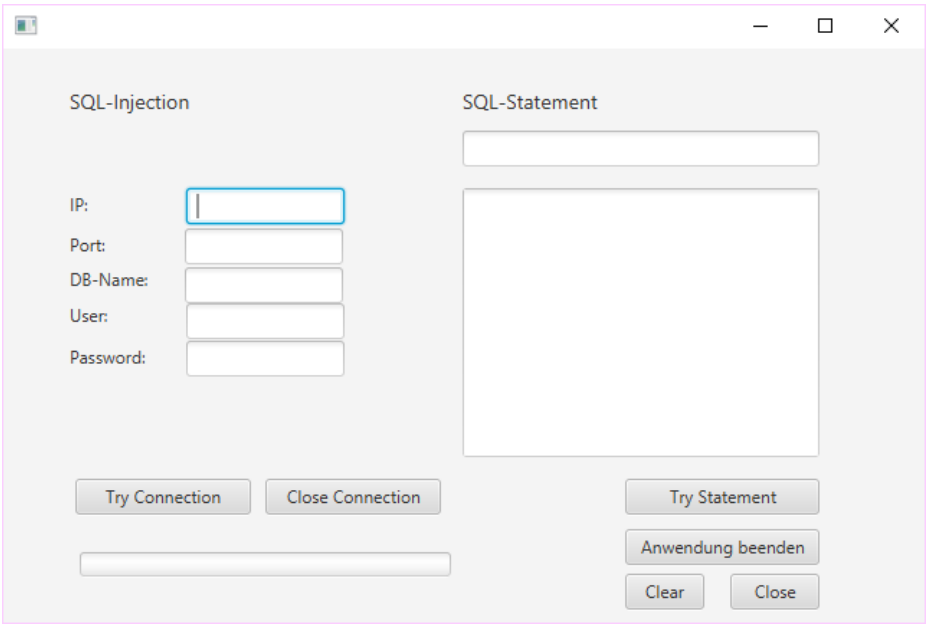


Fig. 35: Anwendungsoberfläche zur SQL-Injection Simulation

Bei der folgenden Simulation wird davon ausgegangen, dass die Eingabe des Passworts und Benutzers korrekt war. Somit erfolgt eine positive Anmeldung, in der folglich SQL-Injection Statements ausgeführt werden können. Dieses Testszenario soll zunächst die Ergebnisse anhand einer positiven Anmeldung und Statement Eingabe demonstrieren. Im anschließenden Fall werden die Versuchsresultate der Passwortmanipulationen präsentiert. Die Aufstellung der Testversuche soll ein besseres Verständnis für die Anwendung sowie Ergebnisse ermöglichen.

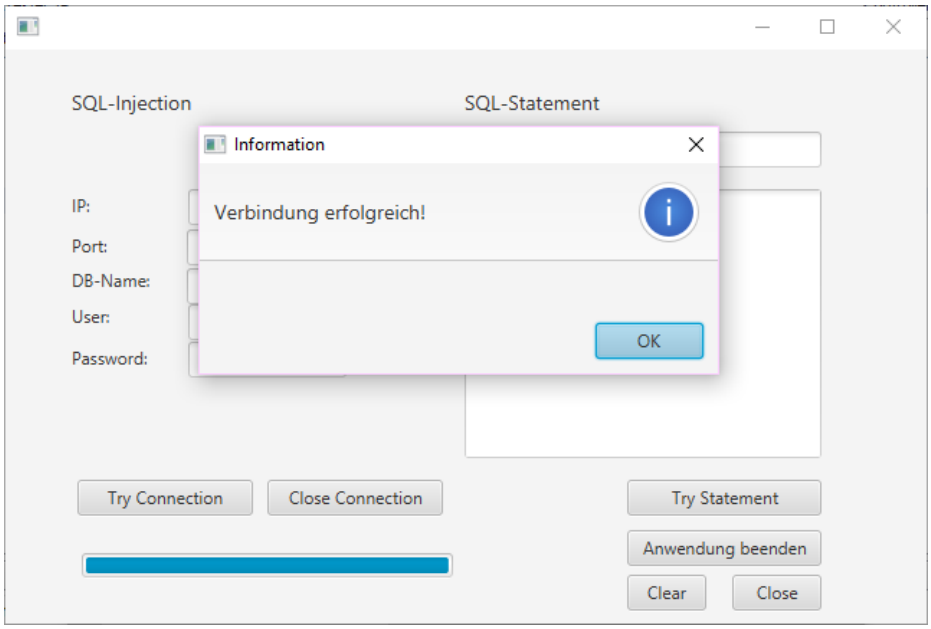


Fig. 36: Positives SQL-Injection Anmeldeverfahren

Nach einer erfolgreichen Anmeldung, können unterschiedliche Statements ausgeführt werden. In Figur 37 werden vom Benutzer CC mit Hilfe des Statements “select * from mp“ alle Datensätze der Tabelle mp aufgerufen. Das Resultat ist in diesem Fall positiv, weil der Benutzer auf diese Tabelle zugreifen darf.

In Figur 38 wird folgendes Statement ausgeführt: “select * from mp; delete table suppliers“. Dieses Statement ist auch als Piggy-Backed Query bekannt, wobei gezielt versucht wird, zum Ursprünglichen Statement ein Zusätzliches einzufügen [41]. In diesem Fall ist das Ziel, die Tabelle suppliers zu löschen. Dieser Versuch brachte keine positiven Resultate, weil das Application Progamming Interface (API) diese Art von Statements nicht zulässt, wodurch eine Fehlermeldung erscheint.

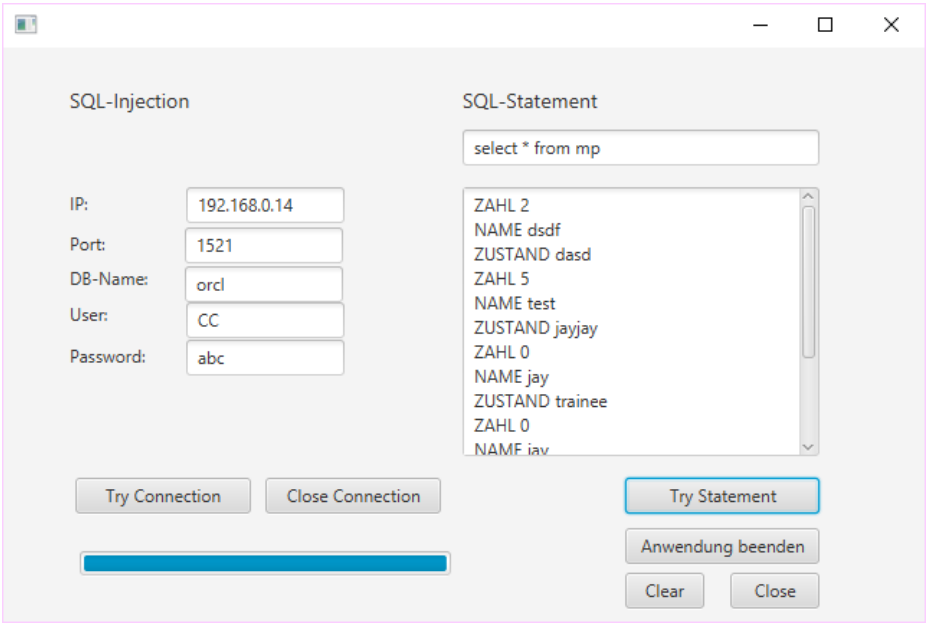


Fig. 37: Erfolgreiches SQL-Statement

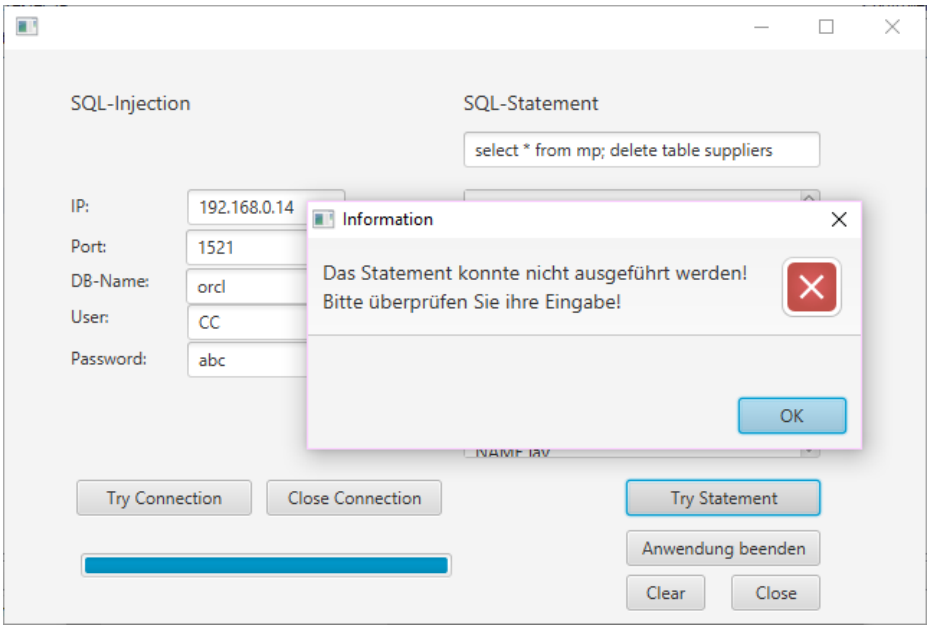


Fig. 38: Erfolgloses Piggy-Backed Query

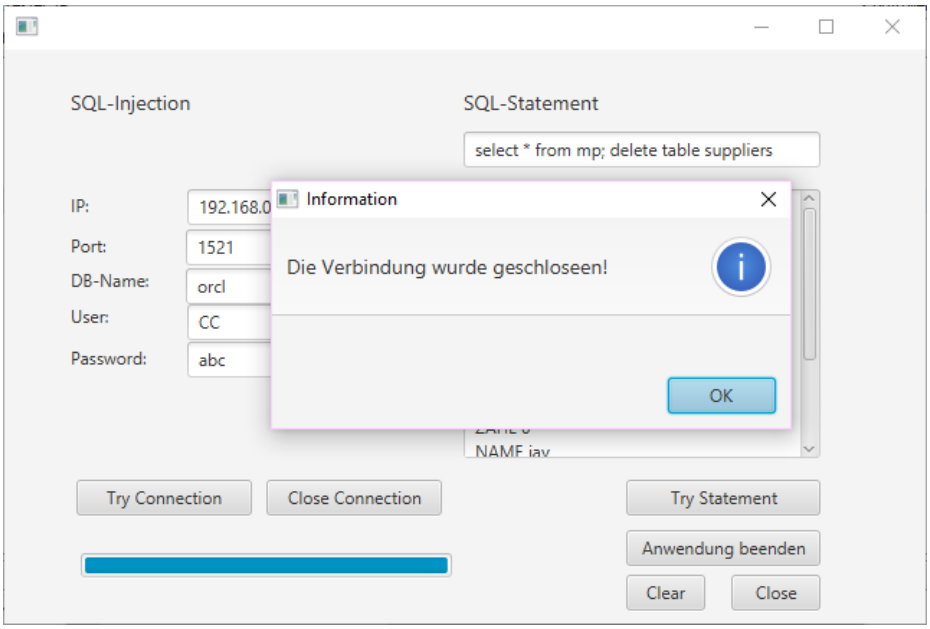


Fig. 39: Schließen der Verbindung nach der SQL-Injection Simulation

Folglich muss, wie in Figur 39 dargestellt, die Verbindung geschlossen werden, woraufhin die Simulation anhand der Security Metrics ausgewertet und bewertet werden kann. In diesem Testdurchgang konnte eine erfolgreiche Verbindung hergestellt werden, wodurch die Authentifizierung als schlecht bewertet wird. Weiters sind die Integrität und Vertraulichkeit als durchschnittlich eingestuft, weil die Konsistenz und Vertraulichkeit der Daten nicht gewährleistet werden kann. Die Verschlüsselung und Verfügbarkeit sind in dieser Simulation gut, weil keine Gefahr auf das Sicherheitsziel projiziert wird und der Systemzugriff garantiert ist. Letztlich ergibt das eine durchschnittliche Gesamtbewertung von 60%, weil der Benutzer Zugang zur Datenbank hatte, aber keine Daten manipulieren konnte.

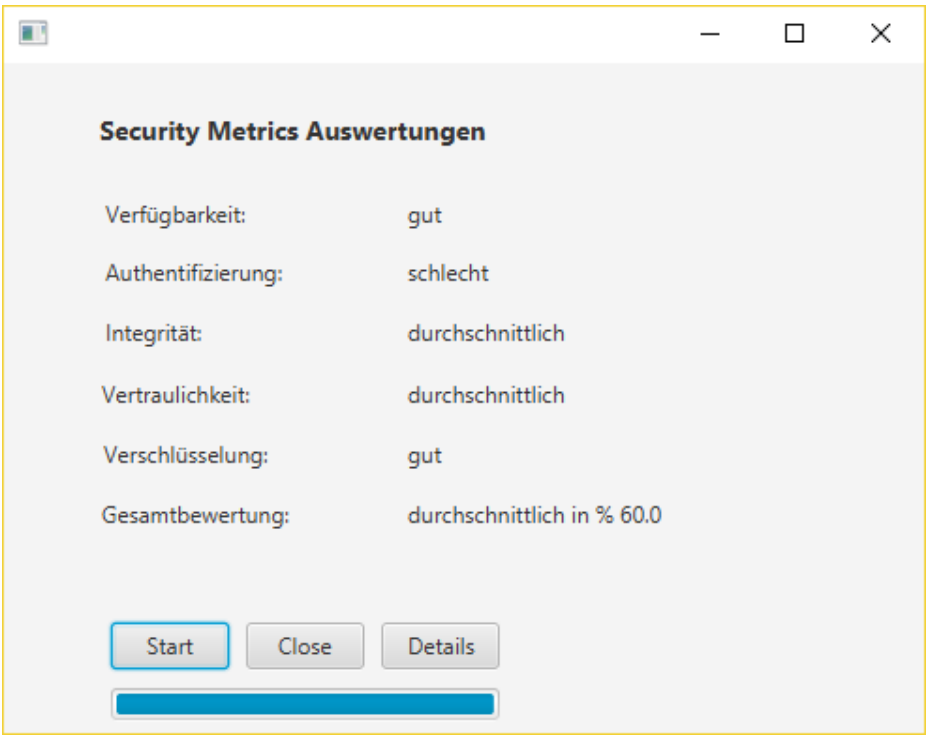


Fig. 40: Auswertung und Bewertung der SQL-Injection Simulation

Letztlich können weitere Detailinformationen der Simulationen aufgerufen werden. Hier besteht die Möglichkeit, zwischen den Einträgen zu wechseln und somit einen Überblick aller ausgeführten Statements zu erhalten. Dadurch werden alle ausgeführten Abfragen mit deren zusammenhängenden Informationen zur Verfügung gestellt. In Figur 41 ist beispielsweise das ausgeführte Statement “select * from mp“ mit dem Ausführungszeitraum, Benutzer sowie dem Returncode abgebildet.

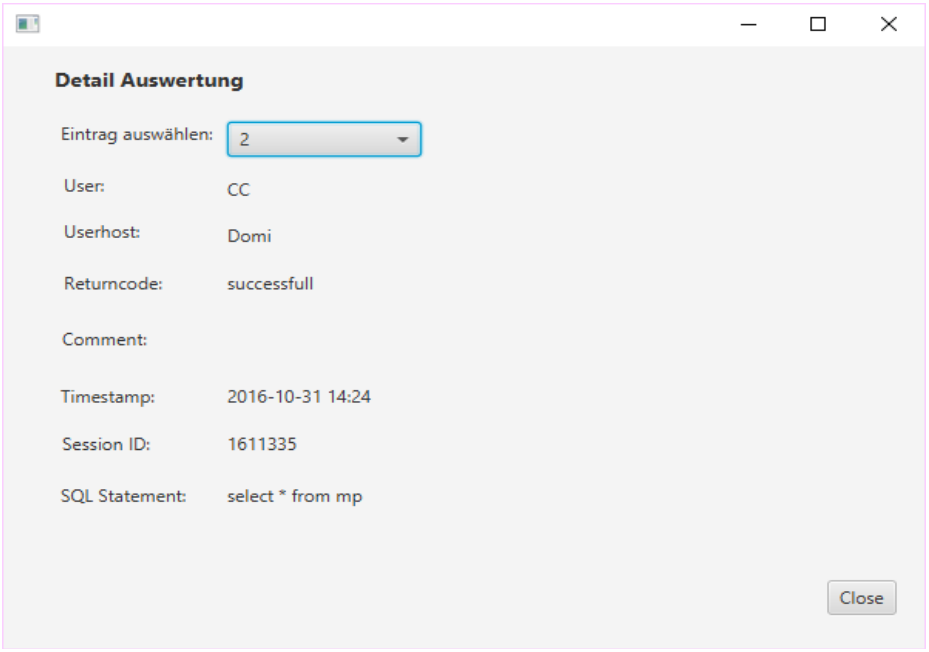


Fig. 41: Detailansicht der Informationen zur SQL-Injection Simulation

Die folgende Simulation stellt die Funktionalität der Anwendung auf die Probe, bei der anstatt des Passwortes der Manipulationsversuch ‘“ OR “=‘ getestet wird. Alle verbindungsnotwendigen Attribute bleiben, wie in Figur 42 abgebildet, erhalten.

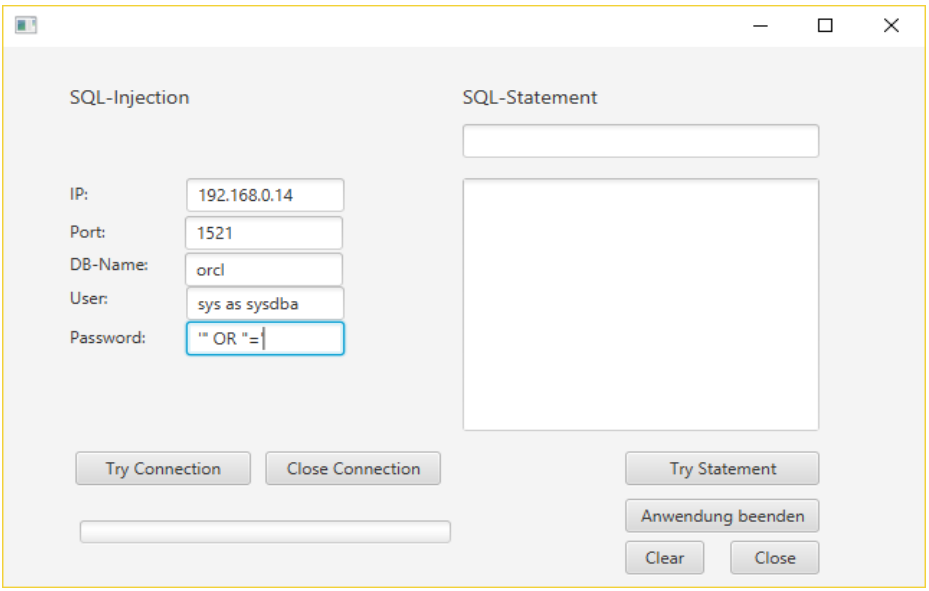


Fig. 42: Tautologie-basiertes SQL-Injection

Wie in Figur 43 dargestellt, hat die SQL-Injection Simulation nicht funktioniert, wodurch eine Fehlermeldung erscheint.

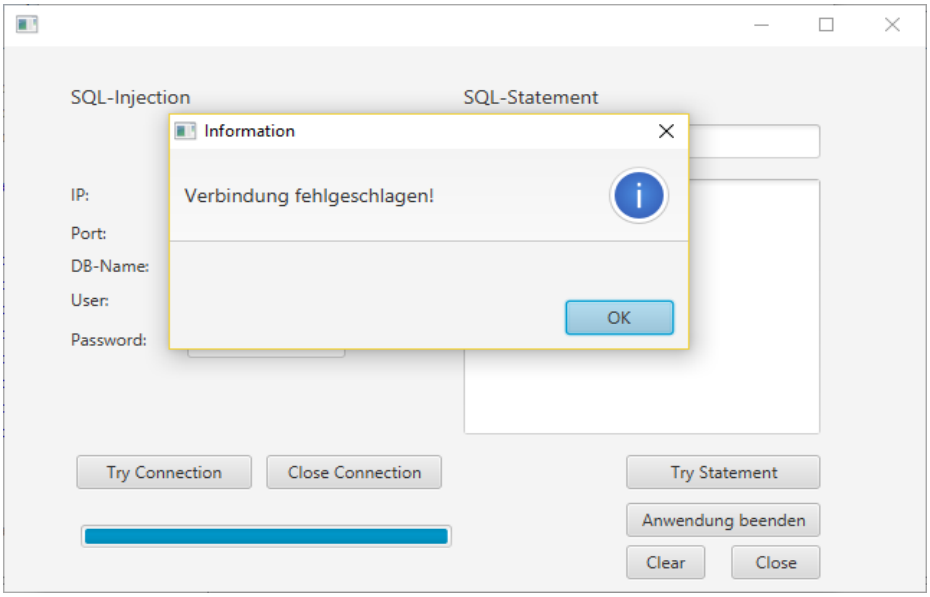


Fig. 43: Erfolglose SQL-Injection Simulation

Nachdem die Verbindung fehlgeschlagen ist und das SQL-Injection keine Auswirkungen auf die Datenbank hatte, werden alle Sicherheitsziele als gut bewertet.

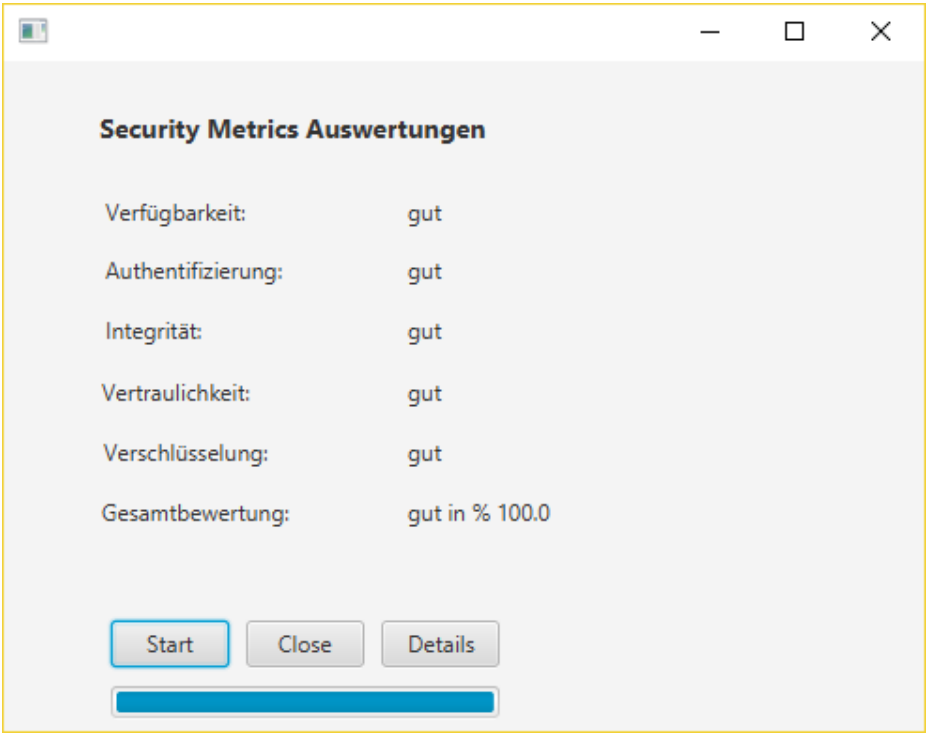


Fig. 44: Auswertung und Bewertung der SQL-Injection

Letztlich können Detailinformationen aufgerufen werden in denen klar ersichtlich ist, dass der Benutzer sys keine Verbindung zur Datenbank erstellen konnte.

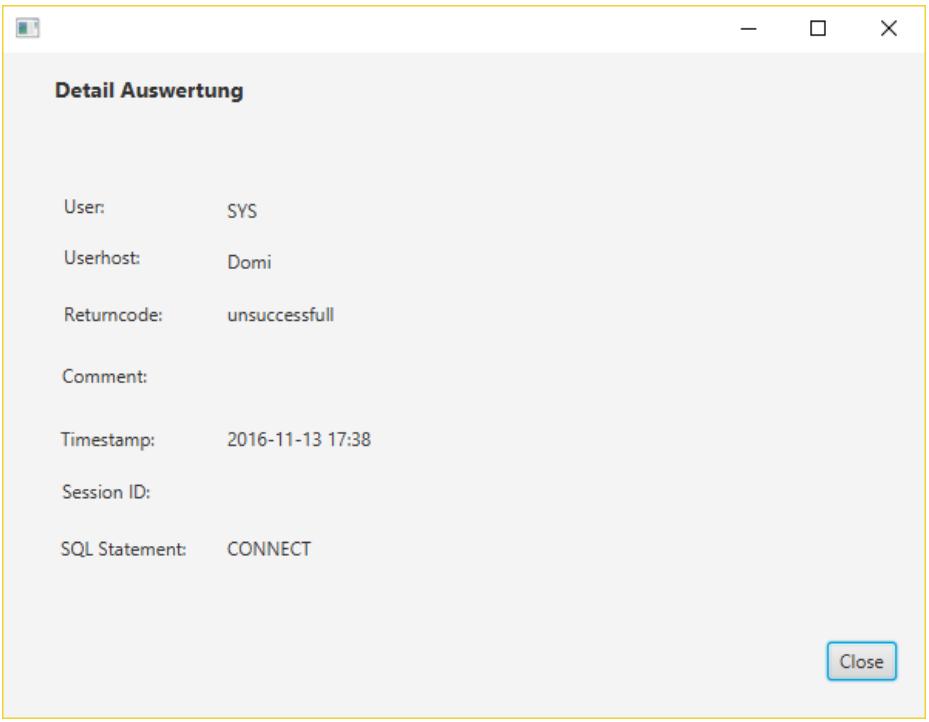


Fig. 45: Detailansicht der SQL-Injection Auswertung

9.1.3 Ping of Death Simulation

Zur Ausführung dieser Bedrohungssimulation müssen bestimmte Attribute eingegeben werden. Benötigt wird die Anzahl der simulierten Clients, die Byte Größe sowie die Anzahl der Ping-Durchgänge und die Ziel IP-Adresse. Sobald alle aufgezählten Informationen eingegeben sind, kann die Simulation gestartet werden. Für das folgende Beispiel wurden 5 Clients, eine Byte Größe von 65500 Bytes, 1000 Ping-Durchgänge und die Ziel IP-Adresse des Datenbanklisteners eingegeben. Das Ziel hierbei ist, die Verbindungskommunikation zum Datenbanklistener zu stören und keine Anmeldungen zuzulassen.

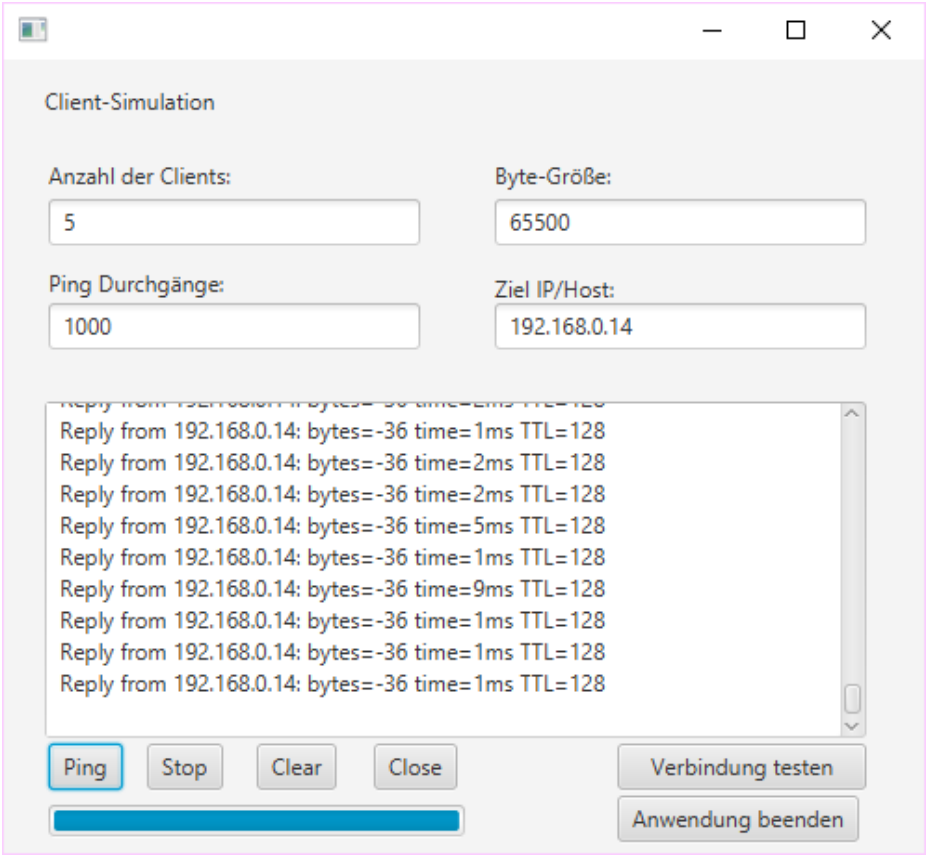


Fig. 46: Ping of Death Simulation

Zur Erreichbarkeitsüberprüfung muss während des Ping of Deaths eine Verbindung zur Datenbank hergestellt werden. War die Überprüfung erfolgreich, besteht keine Gefahr vom ausgehenden Ping of Death. Bei einer nicht erfolgreichen Verbindung würde es bedeuten, dass der Datenbanklistener außer Kraft gesetzt ist und keine Anmeldungen zugelassen werden. In vorliegendem Fall ist, wie in Figur 47 dargestellt, die Anmeldung positiv verlaufen. Es steht außer Zweifel, dass durch die hohe Ping-Anzahl Ressourcen verschwendet werden und die Performance beeinträchtigt ist. Deshalb wurde zur Verbindungsüberpüfung und Auswertung der Sicherheitsziele eine externe Anwendung entwickelt.

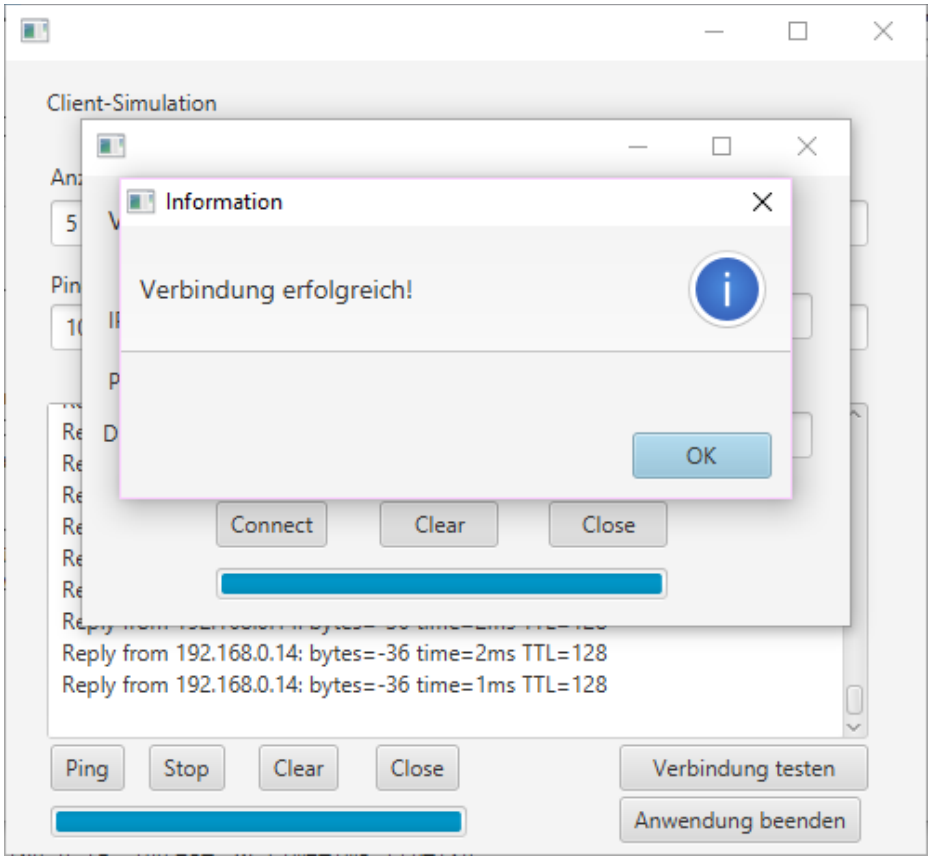


Fig. 47: Erfolgreiche Verbindung während dem Ping of Death

In diesem Testdurchgang konnte der Ping of Death keinen Schaden anrichten, weshalb alle Sicherheitsziele mit gut bewertet werden.

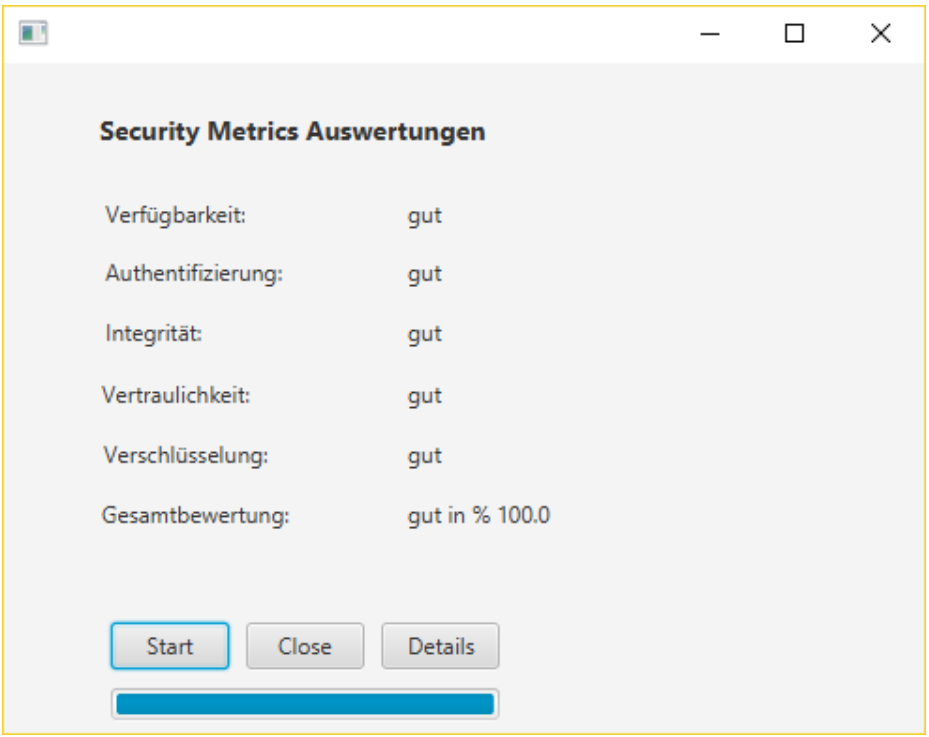


Fig. 48: Auswertung und Bewertung des Ping of Deaths

Wie in den Simulationen davor, können alle Detailinformationen der Bedrohung eingesehen werden. Dabei ist ersichtlich, dass der Benutzer sich erfolgreich anmelden konnte.

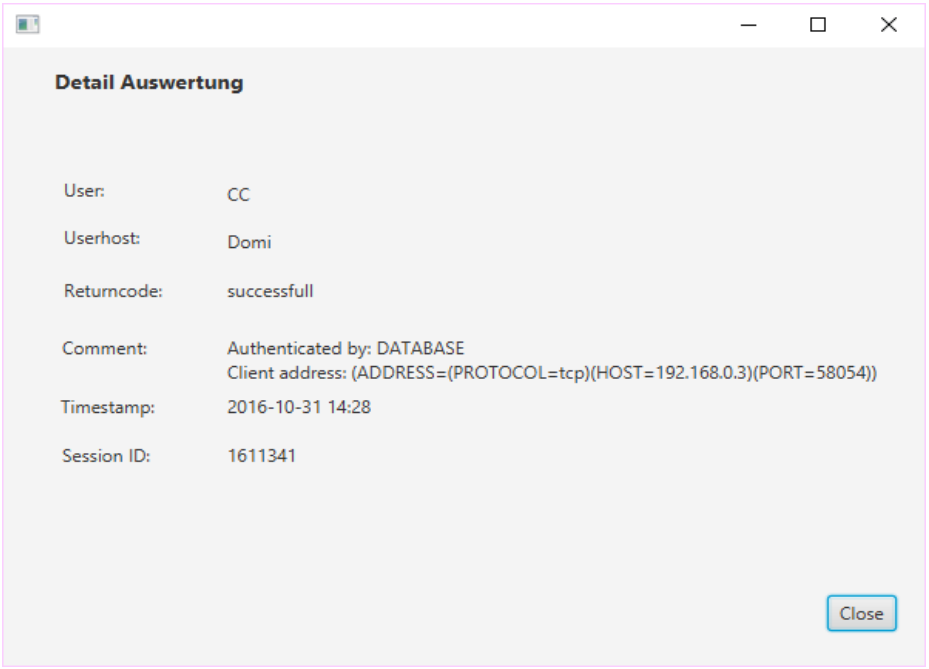


Fig. 49: Detailinformationen des Ping of Deaths

9.1.4 SYN-Flooding Simulation

Für diese Bedrohungen werden zwei Testdurchgänge veranschaulicht. Im ersten Fall, wie in Figur 50 abgebildet, ist die Eingabe der Ziel IP-Adresse und des Ports erforderlich. Danach startet die Anwendung eine große Anzahl an Threads, mit der Intention Datenbankverbindungen zu öffnen. Das Ziel dieser Simulation ist, den Datenbanklistener abstürzen zu lassen und keine Verbindungen mehr zu ermöglichen. Durch die Nutzung einer hohen Anzahl an Threads werden die Ressourcen und Performance beeinträchtigt. Deshalb wurde eine externe Anwendung zur Verbindungsüberpüfung und Auswertung der Sicherheitsziele entwickelt.

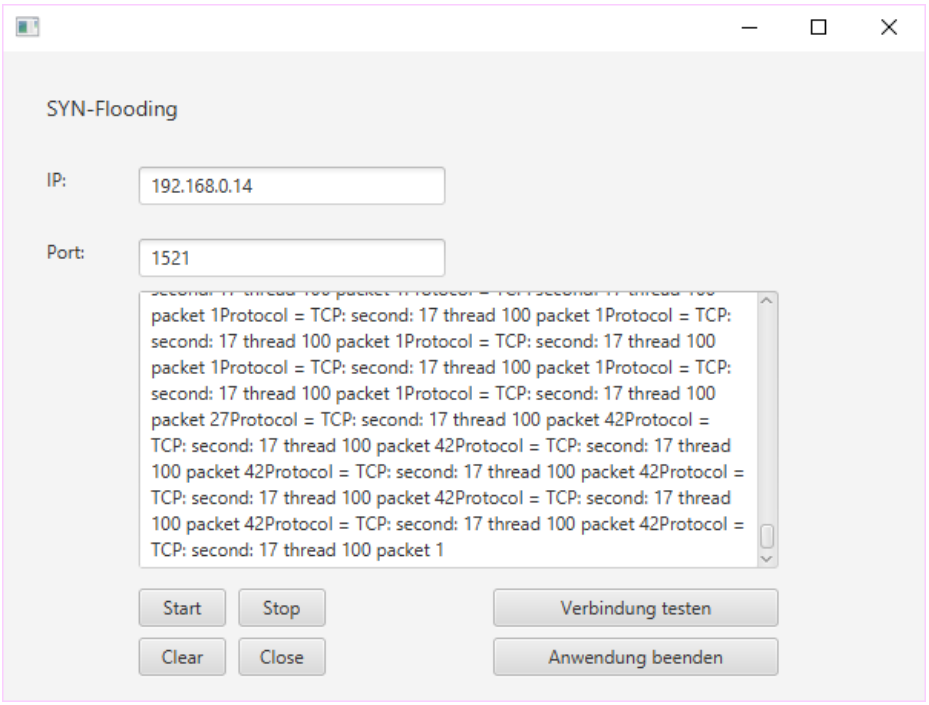


Fig. 50: SYN-Flooding Simulation

Grundsätzlich wurden die Funktionalitäten der externen Anwendung aus der Primären ausgekapselt, wodurch die Workflowschritte identisch sind. Der daraus resultierende Vorteil ist eine bessere Performance und eine effektive Ressourcenaufteilung.

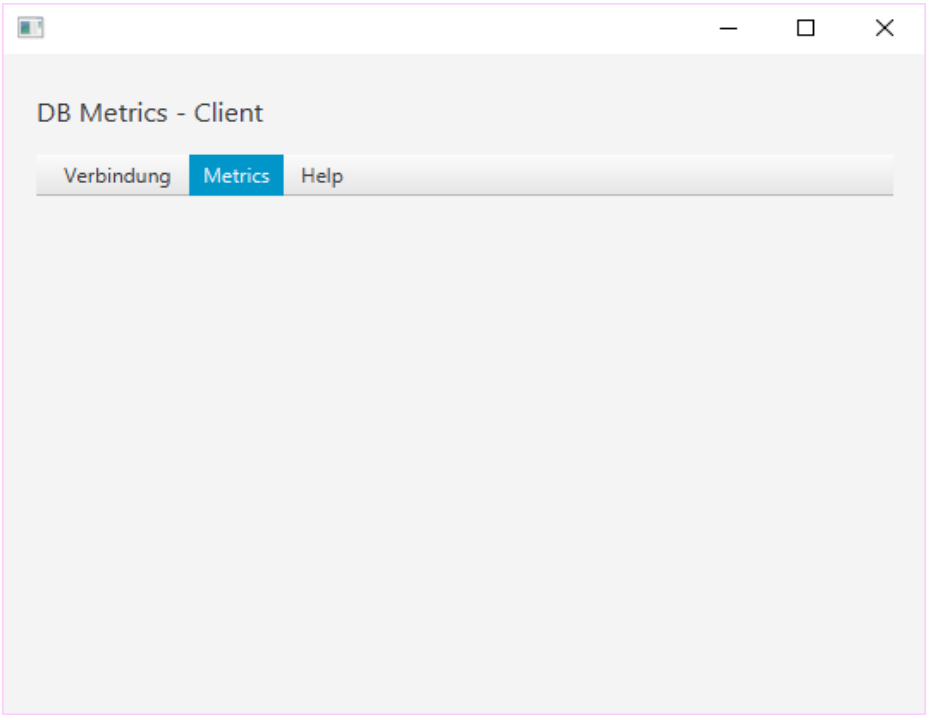


Fig. 51: Externe Anwendung zur Verbindungsüberprüfung während der SYN-Flooding Simulation

Im weiteren Szenario wird auf einem Rechner innerhalb des Netzwerkes eine Verbindungsüberprüfung durchgeführt, während das SYN-Flooding auf einem anderen Rechner ausgeführt wird. Dazu ist, wie in den Simulationen zuvor, die Eingabe der Anmeldeattribute notwendig. Wie in Figur 52 und 53 abgebildet, konnte eine erfolgreiche Verbindung durchgeführt werden. Somit hatte die Bedrohung keine Auswirkung auf die Anmeldung.

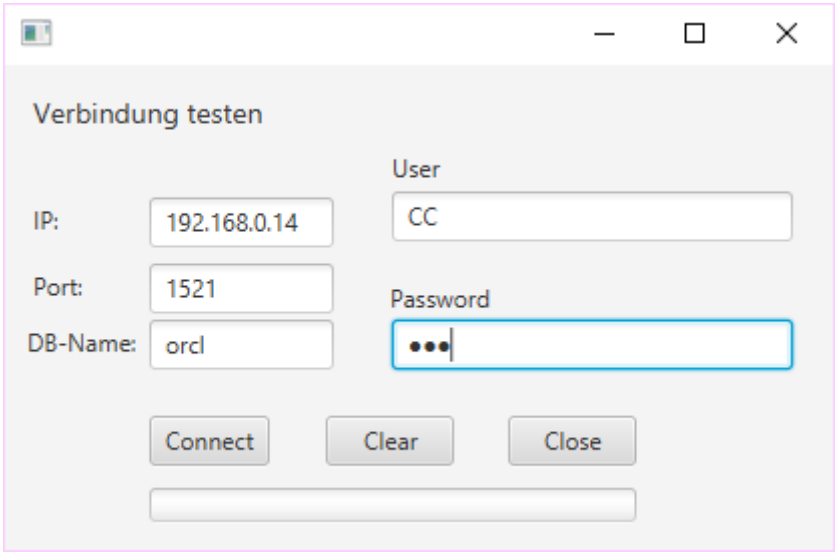


Fig. 52: Eingabe der notwendigen Informationen zur Verbindungsüberprüfung während der SYN-Flooding Simulation

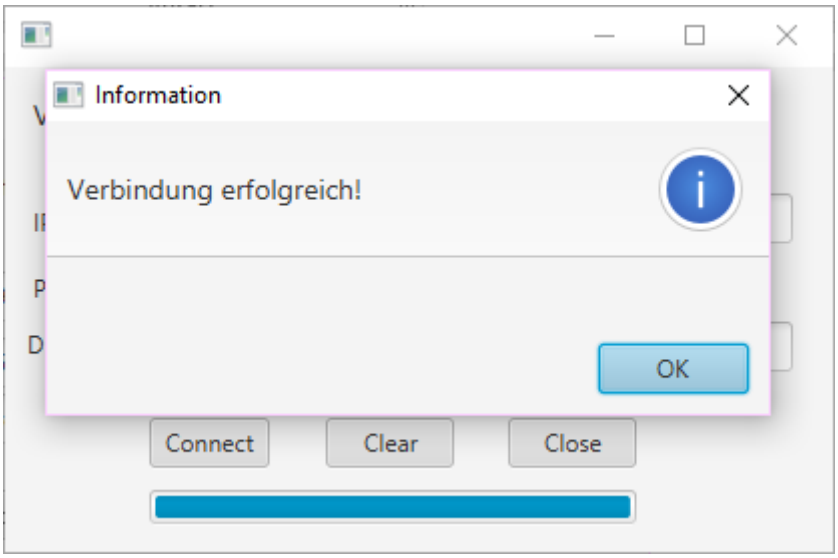


Fig. 53: Erfolgreiche Verbindungsüberprüfung während der SYN-Flooding Simulation

In diesem Testdurchgang konnte die SYN-Flooding Bedrohung keine Gefahr darstellen, weshalb alle Sicherheitsziele mit gut bewertet werden.

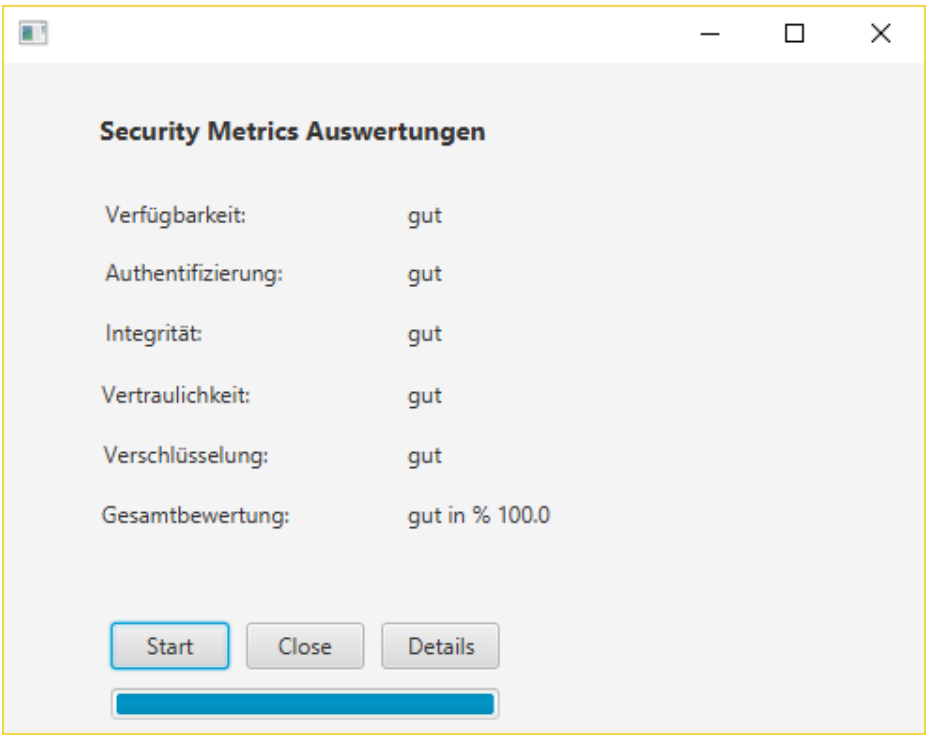


Fig. 54: Auswertung und Bewertung der SYN-Flooding Bedrohung

Letztlich können alle Detailinformationen aufgerufen werden, bei denen eine erfolgreiche Anmeldung verzeichnet ist.

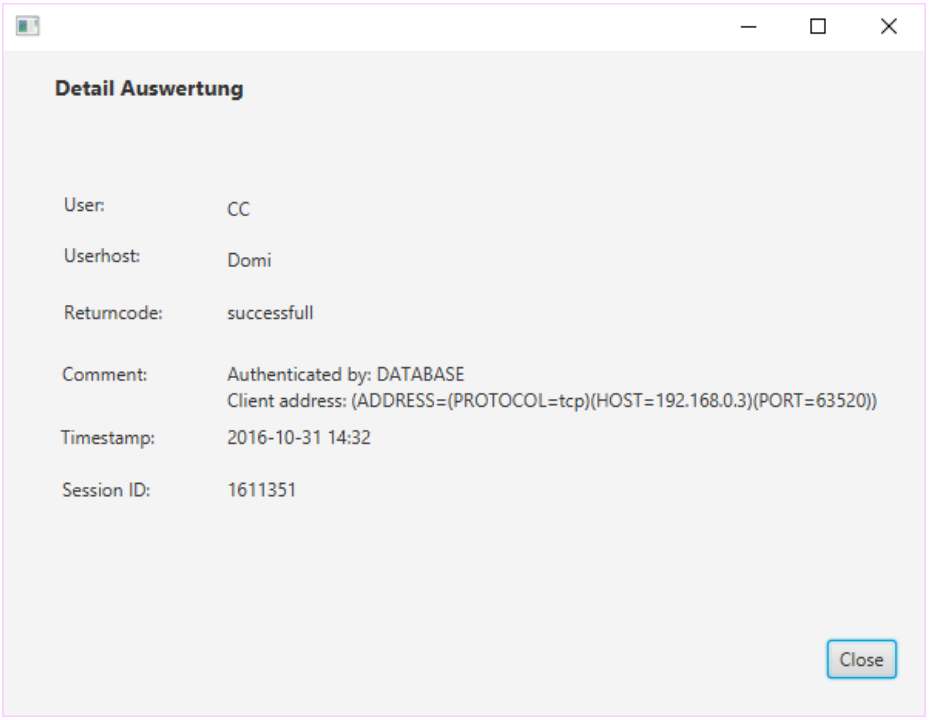


Fig. 55: Detail Auswertung der SYN-Flooding Bedrohung

Im zweiten Testdurchgang wird die Funktionalität nicht über die Anwendung aufgerufen, sondern über die Konsolenanwendung von NetBeans. Wie im Fall davor, muss die Ziel IP-Adresse und der Port angegeben werden. Daraufhin startet die Funktion wie in Figur 56 abgebildet. Währenddessen wird auf einem anderen Rechner im lokalen Netzwerk die externe Anwendung zur Verbindungsüberprüfung gestartet. Hier wird identisch wie in Figur 52 eine Anmeldung durchgeführt.

```
Protocol = TCP: second: 9 thread 100 packet 4
Protocol = TCP: second: 9 thread 100 packet 2
Protocol = TCP: second: 9 thread 100 packet 2
Protocol = TCP: second: 9 thread 100 packet 8
Protocol = TCP: second: 9 thread 100 packet 8
Protocol = TCP: second: 9 thread 100 packet 8
Protocol = TCP: second: 9 thread 100 packet 8
Protocol = TCP: second: 9 thread 100 packet 8
Protocol = TCP: second: 9 thread 100 packet 8
Protocol = TCP: second: 9 thread 100 packet 8
Protocol = TCP: second: 9 thread 100 packet 2
Protocol = TCP: second: 9 thread 100 packet 9
Protocol = TCP: second: 9 thread 100 packet 8
Protocol = TCP: second: 9 thread 100 packet 8
Protocol = TCP: second: 9 thread 100 packet 7
Protocol = TCP: second: 9 thread 100 packet 2
Protocol = TCP: second: 9 thread 100 packet 8
```

Fig. 56: SYN-Flooding Simulation aus der Konsolenanwendung von NetBeans

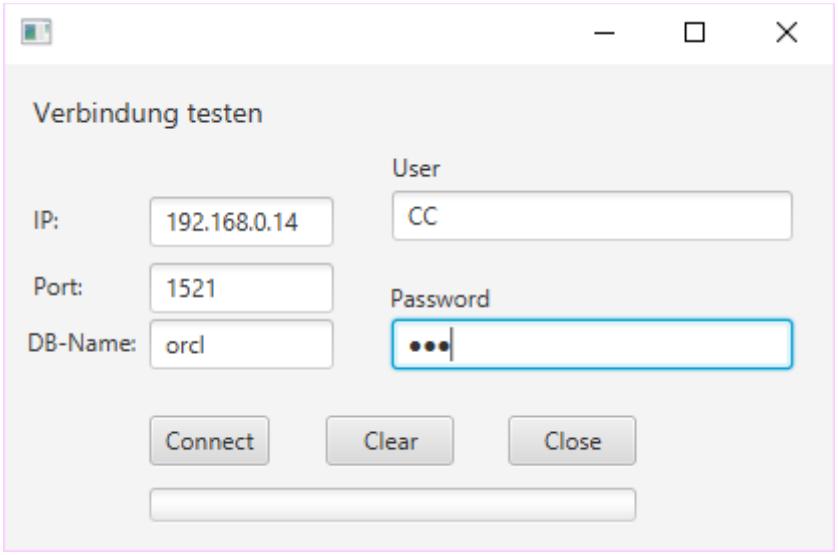


Fig. 57: Eingabe der notwendigen Informationen zur Verbindungsüberprüfung während des SYN-Floodings

In dieser Simulation gelingt es der SYN-Flooding Bedrohung den Datenbanklistener außer Kraft zu setzen und, wie in Figur 58 ersichtlich, keine Verbindungen mehr zuzulassen.

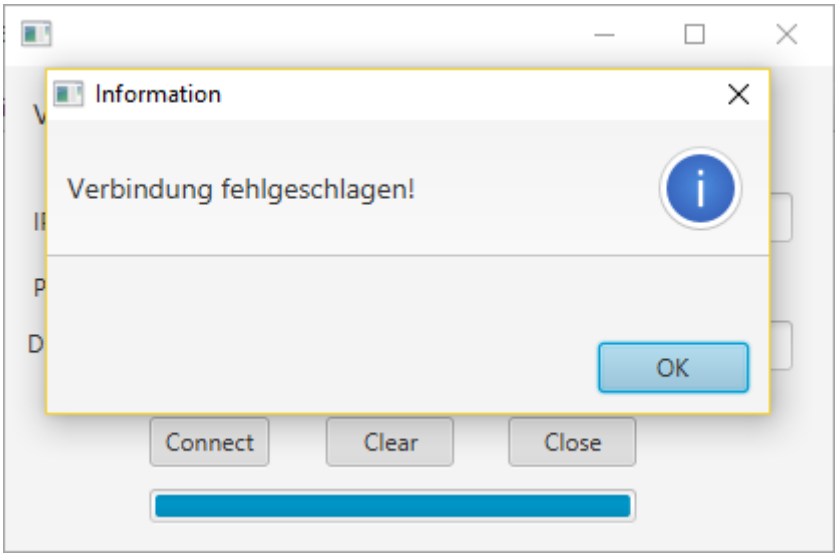


Fig. 58: Erfolglose Verbindungsüberprüfung während des SYN-Floodings

Die Problematik liegt hier nicht nur im Verlust des Datenbanklisteners, sondern auch im Aufzeichnungsstopp der Audit Trails. Somit kann nicht nachvollzogen werden, welcher Benutzer zu welchem Zeitpunkt eine Verbindung startet. Die Unterschiede der zwei Testdurchgänge sind enorm, was sich dadurch erklären lässt, dass die Ausführung der SYN-Flooding Simulation über die Konsolenanwendung mehr Ressourcen zur Verfügung hat und deshalb die Resultate erzielen konnte.

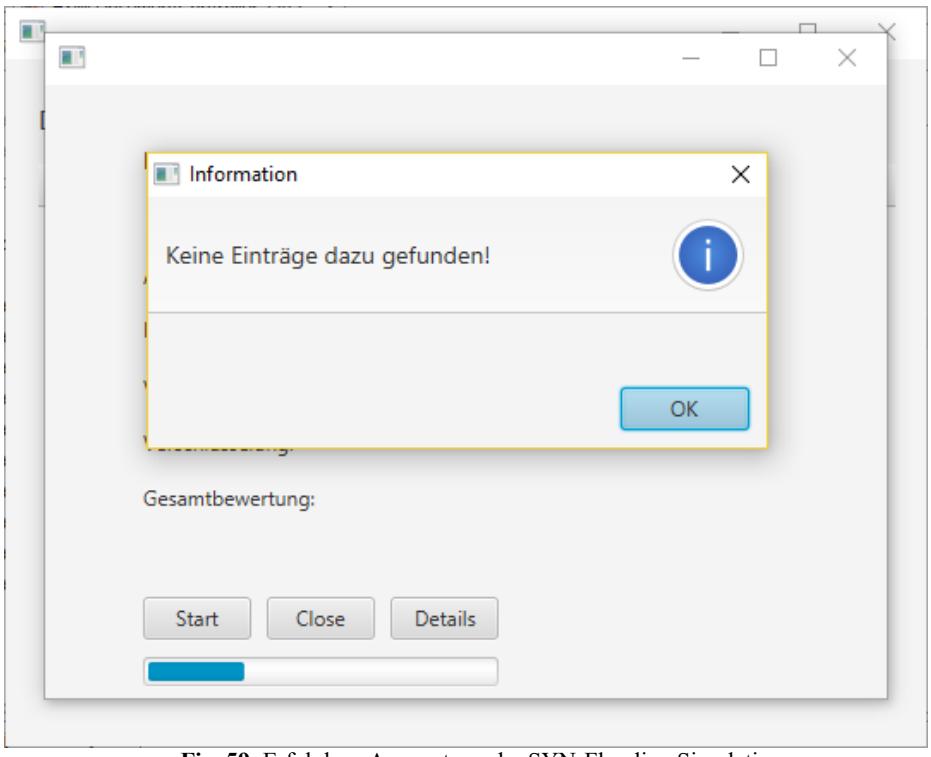


Fig. 59: Erfolglose Auswertung der SYN-Flooding Simulation

9.2 Vergleiche und Bewertungen der Ergebnisse

Die Brute-Force Attacke lieferte, wie auch anhand des vorher illustrierten Beispiels, positiv simulierte Resultate. Obwohl am Anfang Schwierigkeiten in der Umsetzung der Bedrohungen vorhanden waren, konnten beliebig lange Passwörter entschlüsselt werden. Dennoch ist zu beachten, dass durch eine größere Passwortlänge eine größere Zeichenkombination überprüft werden muss. Die Entschlüsselung eines drei Zeichen langen Passwortes dauerte beispielsweise nur einige Sekunden, während bei vier oder fünf Zeichen langen Passwörtern schon einige Minuten benötigt wurden. Sobald das verschlüsselte Passwort bekannt ist, kann die entwickelte Funktionalität dieses entschlüsseln. Im Gegensatz dazu waren die Ergebnisse zu den SQL-Injection Simulationen erfolglos. Es wurden unterschiedliche Tautologie-basierte Angriffe durchgeführt, wie die folgenden Zeichenkombinationen ‘ OR 1=1‘ oder ‘“ OR “=‘, mit dem Ziel das Passwort zu manipulieren und einen Zugriff in die Datenbank zu erhalten. Allerdings konnten zu dieser Bedrohung keine positiv simulierten Resultate erzielt werden, da keine Datenbankzugriffe ermöglicht wurden. In einem weiteren Szenario konnte anhand des bekannten Passwortes eine erfolgreiche Anmeldung durchgeführt werden. Daraufhin wurden Piggy-Backed Queries ausgeführt mit dem Vorhaben, Datensätze zu löschen oder zu manipulieren. Wie im ersten beschriebenen SQL-Injection Szenario, waren die Simulationen hier ebenfalls erfolglos. Letztlich konnte aufgezeigt werden, dass die simulierten SQL-Injections keine Gefahr ausüben konnten.

Die Zielsetzung der Ping of Death Bedrohung ist, eine Menge Pakete an den Datenbanklistener zu schicken und ihn dadurch außer Kraft zu setzen. Nach mehreren Testversuchen, konnten alle Benutzer erfolgreich angemeldet werden. Anschließend wurde eine externe ressourcensparende Anwendung zur Verbindungsüberprüfung entwickelt. Dennoch konnten in weiteren Testdurchläufen, Benutzer auf die Datenbank zugreifen. In den SYN-Flooding Testdurchgängen konnten einige Unterschiede festgestellt werden. Zunächst wurde die Simulation aus der entwickelten Anwendung gestartet, mit dem Ziel den Datenbanklistener abstürzen zu lassen und keine Verbindungen mehr zu ermöglichen. Dazu wurden während der Bedrohungsausführung mehrere Anmeldeversuche durchgeführt. Nachdem alle Verbindungen erfolgreich waren, wurden die Testdurchläufe in der NetBeans Konsole verlagert. Nach dieser Umstellung wurden keine weiteren Anmeldungen zugelassen, wodurch ebenfalls keine Aufzeichnungen erstellt werden konnten. Es wird angenommen, dass durch die wechselnde Testumgebung mehr Ressourcen zur Verfügung standen, die den Datenbanklistener schließlich abstürzen ließen.

Letztlich waren zwei der vier entwickelten Bedrohung erfolgreich. Zweifellos konnten diese Resultate nur durch dauerhafte Testdurchläufe und Codeverbesserungen erzielt werden. Die größten Herausforderungen entstanden in der Brute-Force und SYN-Flooding Entwicklung, weil eine Ressourcenknappheit herrschte und dementsprechend mehrere Testphasen durchlaufen werden mussten. Gleichermaßen bemerkenswert waren die unterschiedlich erzielten Ergebnisse mit denen die Sicherheitsziele besser eingestuft werden können.

10 Erwartete Ergebnisse vs. Case Study

Oracle ist einer der größten Datenbanksystemhersteller und bietet dementsprechend gute Funktionalitäten. Es ist ausgestattet mit Überwachungs- und Datenanalysefunktionen, die dem Benutzer eine Übersicht über das System ermöglichen. Dennoch sind die Datenbanken gegen Bedrohungen wie Denial-of-Services, Trojaner oder Malwares nicht gewappnet [20]. Zu Beginn der Arbeit wurde anhand Literaturrecherchen versucht, die zu erwartenden Ergebnisse aufzustellen. Dabei lag der Fokus auf Brute-Force, SQL-Injection, Ping of Death und SYN-Flooding Attacken. Die Brute-Force Attacke ist eine zweifellos bekannte Bedrohung, bei der zunächst von keiner erfolgreichen Umsetzung ausgegangen wurde, weil der Zusammenhang mit den in Oracle gespeicherten Passwörtern fehlte. Nach einigen Recherchen wurde festgestellt, dass Passwörter in Oracle mittels SHA-1 Algorithmus automatisch verschlüsselt werden [59]. Zur zusätzlichen Sicherheitserhöhung, wird eine zufällig gewählte Zeichenfolge, auch Salt genannt, anhand eines gegebenen Klartextes an den Hashwert angehängt [59]. Der Nachteil des Salts ist die dauerhafte Belegung der letzten 20 Zeilen des Hashwertes, wodurch alle möglichen Klartextkombinationen getestet und mit dem Salt verschlüsselt werden. Darauf basierend kann der neue verschlüsselte Wert mit dem verschlüsselten Passwort verglichen werden. Dieser Vorgang kann beliebig wiederholt werden, bis das ursprünglich verschlüsselte Passwort identifiziert wurde. Dementsprechend konnte man mit Hilfe der Brute-Force Attacken erfolgreiche Angriffe verbuchen und dadurch positive Resultate erzielen, was anfangs noch als sehr unwahrscheinlich eingestuft wurde.

Nach [22], [23] und [41] wurden zu Beginn der Arbeit SQL-Injection als größte Gefahr für Datenbanken angesehen. Nach einigen Testdurchläufen konnten aber keine erfolgreichen Angriffe erzielt werden. Es bestand weder die Möglichkeit sich mit Hilfe des Tautologie-basierten Angriffs als Benutzer anzumelden, noch unterschiedliche Tabellen manipulieren oder löschen zu können. Weiters funktionierten Piggy-Backed Queries, wie beispielsweise‘ OR 1=1‘, nur über die Kommandozeile der Datenbank. Zwar konnte durch das entwickelte Tool mittels SQL-Injection kein Schaden angerichtet werden, jedoch ist es möglich direkt über die Kommandozeile Manipulationen vorzunehmen. Letztlich konnte, diese Bedrohung mit Hilfe des Tools keine positiven Ergebnisse erzielen, weil alle Angriffe verhindert wurden.

Bei Ping of Deaths war zunächst unklar, ob ein Schaden in den weiterentwickelnden Datenbanksystemen entstehen könnte. Nach einigen Simulationen über die Kommandozeile eines Rechners konnte keine Beeinträchtigung wahrgenommen werden. Durch die Zuhilfenahme des entwickelten Tools konnte das Ziel, den Datenbanklistener zu Fall zu bringen, allerdings erreicht werden. Dementsprechend können Ping of Deaths für die heutigen Datenbanksysteme als ungefährlich eingestuft werden.

Die wohl wirksamste Attacke war das SYN-Flooding. Mit Hilfe dieser Attacke war es möglich, den Datenbanklistener außer Kraft zu setzen und somit keine Anmeldungen mehr zu ermöglichen. Weiters wird davon ausgegangen, dass die Simulation innerhalb

des Tools eine Menge Ressourcen verbraucht und deshalb das SYN-Flooding zunächst nicht in der Lage war den Datenbanklistener zu beschädigen. Nachdem nur der Source Code außerhalb des Tools ausgeführt wurde, bestand für den Benutzer keine Anmeldemöglichkeit mehr. Außerdem konnten die Audit Trails dadurch die Anmeldungen nicht mehr aufzeichnen, was den Schaden zusätzlich erhöht.

Letztlich haben die Brute-Force und SYN-Flooding Attacken am besten abgeschnitten. Wie bereits erwähnt, war am Anfang der Arbeit das Ziel, mit SYN-Floodings die Datenbank zu beeinträchtigen, welches schließlich auch erreicht wurde. Den Brute-Force Attacken wurde kein besonders hohes Gefährdungspotential zugeschrieben, was jedoch durch die Simulation als falsch erwies. Es konnten hiermit Passwörter, entgegen der anfänglichen Annahme, entschlüsselt werden. Die erzielten Resultate der Ping of Death und SQL-Injection Simulationen waren allerdings mangelhaft, obwohl davon ausgegangen wurde, das zumindest SQL-Injections eine große Gefahr darstellen können. Dennoch konnten Schwachstellen mit zwei von vier Bedrohungen in der Datenbank identifiziert werden.

11 Conclusio

Datenbanken sind vielen Risiken ausgesetzt. Angreifer sind kontinuierlich auf der Suche nach Schwachstellen, um vorhandene Daten zu stehlen oder zu manipulieren. Das Ziel dieser Arbeit war es, die verbreitetsten Datenbankbedrohungen auf einer Oracle Datenbank zu simulieren und den hierbei entstandenen Schaden anhand der Security Metrics zu bewerten. Dabei war die Aufstellung der Sicherheitsziele von großer Bedeutung. Aufgrund der Werke [3][5][46] lag der Fokus auf den folgenden zu schützenden Zielen: Vertraulichkeit, Integrität, Verfügbarkeit, Authentifizierung und Verschlüsselung. Die Security Metrics konnten anhand der von Wang und Wulf beschriebenen Dekompositionsmethode erstellt werden, in dem die Sicherheitsziele als High-Level Ebene definiert und daraufhin ein Fokus auf die zu erstellenden Low-Level Ebenen gelegt wurde [46]. Dadurch entstanden die zu bewertenden Faktoren der Security Metrics. Außerdem war es notwendig Bedrohungen auszuforschen, die eine potenzielle Gefahr auf die Sicherheitsziele ausüben. Folglich wurde ein Framework entwickelt, in dem unterschiedliche Angriffe ausgeführt wurden und daraufhin der Schaden anhand der Security Metrics bewertet werden konnte. SQL-Injection ist beispielsweise eine Bedrohung, mit dem Ziel die SQL-Syntax auszunutzen und bestehende Schwachstellen zu identifizieren, um einen Datenbankzugriff zu ermöglichen [2]. Weiterhin sind Denial of Services (DoS) eine große Gefahr, die eine Datenbank oder ein Netzwerk unzugänglich machen können [2]. Ping of Deaths und SYN-Flooding Attacken haben dasselbe Ziel, in dem sie die Datenbank mit Anfragen überfluten und den Datenbanklistener zum Absturz führen wollen. Eine große Sicherheitslücke entsteht durch ein schwaches oder kaum vorhandenes Audit Trail. Das Audit Trail ist in dieser Arbeit ein wichtiger Bestandteil für die Überprüfung und Aufnahme von sensiblen Daten oder ungewöhnlichen Datenbanktransaktionen [2]. Weiterhin sind Security Metrics ein bedeutendes Fundament zur Entscheidungsfindung aller betroffenen Sicherheitsaspekte wie zum Beispiel das Design der Sicherheitsarchitektur oder der Abwicklung von Sicherheitsoperationen [3][5]. Wichtige Kriterien zur Erstellung von Security Metrics sind zum Beispiel die Korrektheit und Effektivität eines Systems sowie die Definition von Sicherheitszielen oder die Nutzung von qualitativen oder quantitativen Faktoreigenschaften [5]. Die Problematik liegt in der Unterscheidung zwischen brauchbaren und unbrauchbaren Security Metrics [49]. Wichtig ist, dass die erfassten Security Metrics mit den Sicherheitszielen harmonieren.

Die Literatur zu diesem Thema ist eine wichtige Stütze zur praktischen Umsetzung. Es steht außer Zweifel, dass unterschiedliche Verfahrensmöglichkeiten, wie die Dekomposition Methode oder das GQM Verfahren, möglich sind. Dennoch sind theoretische Faktoren wie die Auswahl der Sicherheitsziele, Identifizierung der Schwachstellen und Bedrohungen für die Praxis von Bedeutung. Die Bewertungsmethoden anhand einer Skala oder die Auswahl von qualitativen und quantitativen Attributen sind ebenfalls wichtige Hilfestellungen zur Erstellung von Security Metrics. Letztlich ist die Literatur für die Praxis unumgänglich.

Tabellenverzeichnis

Tabelle 1: Instanzen der Relation Mitarbeiter 13
Tabelle 2: Instanzen der Tabelle Mitarbeiter 17
Tabelle 3: Instanzen der Tabelle Projekt 17
Tabelle 4: Definitionen der Security Metrics [7][54][55][56] 56
Tabelle 5: Gap Analyse 60

Abbildungsverzeichnis

Fig. 1: Hierarchisches Datenbankmodell [25] 18
Fig. 2: Beispielhafte Netzwerkdatenbank [25] 19
Fig. 3: Abstraktionsebenen einer DBMS [17] 24
Fig. 4: Dekomposition Methode anhand eines Haussystems [46] 39
Fig. 5: Dekomposition Methode zur Authentifizierung 40
Fig. 6: Dekomposition Methode anhand der Verschlüsselung 41
Fig. 7: Dekomposition Methode anhand der Integrität 42
Fig. 8: Dekomposition Methode anhand der Vertraulichkeit 43
Fig. 9: Dekomposition Modell anhand der Verfügbarkeit 44
Fig. 10: Business-level Security Metrics [7] 48
Fig. 11: Security Metrics für das Information Security Management [7] 49
Fig. 12: Security, Dependability und Trust Metrics für Produkte, Systeme und Services [7] 50
Fig. 13: MVC Pattern basierend auf [61] 61
Fig. 14: Übersichtsdiagramm der Klassen und deren Beziehungen 62
Fig. 15: Klassendiagramm des erstellten Frameworks 67
Fig. 16: Metamodell des erstellten Frameworks 68
Fig. 17: JavaFX Scene Builder 69
Fig. 18: Code Auszug aus NetBeans zur Verbindung mit der virtuellen Maschine ... 70
Fig. 19: Code Auszug der Klasse PWEncoder aus NetBeans 71
Fig. 20: Code Auszug zur SQL-Injection Simulation aus NetBeans 72
Fig. 21: Code Auszug der Ping of Death Simulation aus NetBeans 73
Fig. 22: Code Auszug der SYN-Flooding Simulation aus NetBeans 74
Fig. 23: Code Auszug des Zeitvergleiches aus NetBeans 75
Fig. 24: Code Auszug zur detaillierten Auswertung der Security Metrics aus Net Beans 76
Fig. 25: Startoberfläche der entwickelten Anwendung 77
Fig. 26: Auswahl der möglichen Bedrohungen 77
Fig. 27: Verbindungsüberprüfung anhand eines Pings 78
Fig. 28: Brute-Force Simulation 79
Fig. 29: Anmeldeversuch nach der Brute-Force Simulation 79
Fig. 30: Positive Anmeldebestätigung nach der Brute-Force Simulation 80
Fig. 31: Anwendungsoberfläche der Security Metrics 81
Fig. 32: Anmeldung für den Zugriff auf die virtuelle Maschine durch die Eingabe der notwendigen Attribute 82
Fig. 33: Auswertung und Bewertung der Sicherheitsziele nach der Brute-Force Simulation 83
Fig. 34: Detailansicht der ausgewerteten Brute-Force Simulation 84
Fig. 35: Anwendungsoberfläche zur SQL-Injection Simulation 85
Fig. 36: Positives SQL-Injection Anmeldeverfahren 86
Fig. 37: Erfolgreiches SQL-Statement 87
Fig. 38: Erfolgloses Piggy-Backed Query 87
Fig. 39: Schließen der Verbindung nach der SQL-Injection Simulation 88
Fig. 40: Auswertung und Bewertung der SQL-Injection Simulation 89
Fig. 41: Detailansicht der Informationen zur SQL-Injection Simulation 90

Fig. 42: Tautologie-basiertes SQL-Injection 91
Fig. 43: Erfolglose SQL-Injection Simulation..... 91
Fig. 44: Auswertung und Bewertung der SQL-Injection..... 92
Fig. 45: Detailansicht der SQL-Injection Auswertung 93
Fig. 46: Ping of Death Simulation 94
Fig. 47: Erfolgreiche Verbindung während dem Ping of Death 95
Fig. 48: Auswertung und Bewertung des Ping of Deaths..... 96
Fig. 49: Detailinformationen des Ping of Deaths 97
Fig. 50: SYN-Flooding Simulation 98
Fig. 51: Externe Anwendung zur Verbindungsüberprüfung während der SYN-Flooding Simulation..... 99
Fig. 52: Eingabe der notwendigen Informationen zur Verbindungsüberprüfung während der SYN-Flooding Simulation 100
Fig. 53: Erfolgreiche Verbindungsüberprüfung während der SYN-Flooding Simulation 100
Fig. 54: Auswertung und Bewertung der SYN-Flooding Bedrohung 101
Fig. 55: Detail Auswertung der SYN-Flooding Bedrohung 102
Fig. 56: SYN-Flooding Simulation aus der Konsolenanwendung von NetBeans 103
Fig. 57: Eingabe der notwendigen Informationen zur Verbindungsüberprüfung während des SYN-Floodings..... 103
Fig. 58: Erfolglose Verbindungsüberprüfung während des SYN-Floodings..... 104
Fig. 59: Erfolglose Auswertung der SYN-Flooding Simulation 105

References

1. Ramakrishnan, R., Gehrke, J. (2003): Database Management Systems, Third Edition, Wisconsin, USA 2003
2. Shulman, A. (2006): Top Ten Database Security Threads, Imperva, Inc., Israel, Ramat-Gan 2006
3. Savola, R. M. (2009): A Security Metrics Taxonomization Model for Software-Intensive Systems [1], Journal of Information Processing Systems, South Korea, Seoul 2009
4. Trimintzios, P. (2011): Measurement Frameworks and Metrics for Resilient Networks and Services: Technical Report, European Network and Information Security Agency (ENISA), Greece 2011
5. Jansen, W. (2009): Directions in Security Metrics Research, National Institute of Standards and Technology, NISTIR 7564, USA 2009
6. Peureux, F., Großmann, J., Legeard, B., Schneider, M., Seehusen, F. (2013): Baseline for Compositional Risk-Based Security Testing, Community Research and Development Information Service, Belgium, Brüssel 2013
7. Savola, R. M. (2009): A Novel Security Metrics Taxonomy for R&D Organisations, VTT Technical Research Centre of Finland, Oulu 2009
8. Swanson, M., Bartol, N., Sabato, J., Hash, J., Graffo, L. (2005): Security metrics guide for information technology systems. NIST Special Publication 800-55, National Institute of Standards and Technology, USA 2005
9. Vaughn, R. B. Jr., Henning, R., Siraj, A. (2002): Information assurance measures and metrics – state of practice and proposed taxonomy, In: Proceeding of the 36th Annual Hawaii International Conference on System Sciences (HICSS'03). IEEE 2002
10. Vladoiu, L. A. (2012): Metrics Model for Assessing the Security of ORACLE Databases, Journal of Applied Business Information Systems, Romania, Timisoara 2012
11. Swanson, M. (2001): Security Self-Assessment Guide for Information Technology Systems, NIST Special Publication 800-26, National Institute of Standards and Technology, USA 2001
12. Ross, R., Johnson, A., Katzke, S., Toth, P., Rogers, G. (2006): Guide for Assesing the Security Controls in Federal Information Systems. NIST Publication 800-53A, USA 2006
13. FIPS Publication 199 (2004): Standards for Security Categorization of Federal Information and Information Systems, Federal Information Processing Standards Publication 2004
14. Lennon, E. (2003): IT Security Metrics. ITL Bulletin, National Institute of Standards and Technology, USA 2003
15. ISO/IEC 21827 (2008): Information Technology – Systems Security Engineering – Capability Maturitiy Model (SSE-CMM). International Orgainzation of Standardization 2008
16. ISO/IEC 15408-1 (2009): Information technology – Security techniques – Evaluation criteria for IT security – Part 1: Introduction and general model, International Organization of Standardization 2009
17. Watt, A. (2013): Database Design, Canada 2013
18. Pöschek, A. (2000): Objektorientierte Datenbanken, Austria, Vienna 2000
19. Karagiannis, D., Kühn, H. (2002): Metamodelling Platforms, Springer-Verlag, Austria, Vienna 2002
20. Information Systems Audit and Control Association (ISACA) (2015): Database Security – A threat from within, 2015

21. Kolodgy, C., Burke, B., Pintal, G. (2008): Oracle Database Security: Preventing Enterprise Data Leaks at the Source, International Data Cooperation (IDC), USA 2008
22. W3Schools (2016): SQL Injection, URL: http://www.w3schools.com/sql/sql_injection.asp, Stand: Dezember 2016
23. Open Web Application Security Project (OWASP) (2016): Testing for SQL Injection (OTG-INPVAL-005), URL: https://www.owasp.org/index.php/Testing_for_SQL_Injection_(OTG-INPVAL-005), Stand: April 2016
24. Wikibooks (2016): Ausführliche SELECT-Struktur, URL: https://de.wikibooks.org/wiki/Einf%C3%BChrung_in_SQL:_Ausf%C3%BChrliche_SELECT-Struktur, Stand: Januar 2016
25. Compendio Bildungsmedien (2004): Modul 141: Datenbanksysteme in Betrieb nehmen, URL: http://slideplayer.org/slide/2900209/, Stand: November 2015
26. Bundesamt für Sicherheit in der Informationstechnik (2016): IT-Grundschutz-Kataloge, 15. Ergänzungslieferung, Deutschland, Bonn 2016
27. Natan, R. (2010): Acht Schritte zur ganzheitlichen Datenbanksicherheit, IBM 2010
28. Ferraiolo, D., Kuhn, R. (1992): Role-Based Access Controls, 15th National Computer Security Conference, USA, Baltimore 1992
29. National Institute of Standards and Technology (NIST) (2016): Role Based Access Control (RBAC) and Role Based Security, URL: http://csrc.nist.gov/groups/SNS/rbac/, Stand: April 2016
30. Hadi, M. (2011): Access Control List (ACL), Electronic Engineering Polytechnic of Surabaya, Indonesia, Surabaya 2011
31. The Internet Engineering Task Force (IETF) (2000): Remote Authentication Dial in User Service (RADIUS), RFC2865, URL: https://tools.ietf.org/html/rfc2865, Stand: Juni 2000
32. CISCO (2008): TACACS+ and RADIUS Comparison, URL: http://www.cisco.com/c/en/us/support/docs/security-vpn/remote-authentication-dial-user-service-radius/13838-10.html, Stand: Januar 2008
33. The Internet Engineering Task Force (IETF) (1993): An Access Control Protocol, Sometimes called TACACS, RFC1492, URL: https://tools.ietf.org/html/rfc1492, Stand: Juli 1993
34. FIPS Publication 46-3 (1999): Data Encryption Standard (DES), Federal Information Processing Standards Publication 1999
35. FIPS Publication 197 (2001): Advanced Encryption Standard (AES), Federal Information Processing Standards Publication 2001
36. Bogdanov, A., Khovratovich, D., Rechberger, C.: Biclique Cryptanalysis of the Full AES, USA, Microsoft Research Redmond
37. Crypttol, URL: https://www.cryptool.org/images/ct1/presentations/RSA/RSA-Flash-de/player.html
38. Becker, J. (2008): RSA-Verschlüsselung, Germany, Gießen 2008
39. Codd, E.F. (1990): The Relational Model for Database Management - Version 2, Addison-Wesley Publishing Company, Inc., USA 1990
40. T-Online (2010): Datensicherung bei Datenbanken, URL: http://www.t-online.de/computer/id_41922194/datensicherung-bei-datenbanken-hot-backup-und-cold-backup.html, Stand: Juni 2010
41. Halfond, W., Viegas, J., Orso, A. (2006): A Classification of SQL Injection Attacks and Countermeasures, Georgia Institute of Technology, USA, Georgia 2006
42. Mittner, P., Schmid, I., Studer, B. (2000): DDOS Attacken Tutorial, University of Applied Science, Switzerland 2000
43. Erickson, J. (2008): Hacking: The Art of Exploitation, 2nd Edition, USA, San Francisco 2008

44. Dean, M. (2015): All About Oracle Auditin – Updated for 12C, Database Specialists, Inc., USA, San Francisco 2015
45. Savola, R.: Identification of Basic Measurable Security Components in Software Intensive Systems, VIT Technical Research Centre of Finland, Oulu
46. Wang, C., Wulf, W. (1990): Towards a Framework for Security Measurement, University of Virginia, USA 1990
47. Bundeskanzleramt Rechtsinformationssystem (2016): Bundesrecht konsolidiert: Gesamte Rechtsvorschrift für Datenschutzgesetz 2000, Fassung vom 19.01.2017, URL: https://www.ris.bka.gv.at/GeltendeFassung.wxe?Abfrage=bundesnormen&Gesetzesnummer=10001597, Stand: Januar 2016
48. Tashi, I., Ghernaouti-Helie, S. (2007): Security metrics to improve information security management, HEC Business School – University of Lausanne, Switzerland 2007
49. Vijayan, J. (2016). The Four Big Problems with Security Metrics, URL: http://www.darkreading.com/risk/the-four-big-problems-with-security-metrics/d/d-id/1323849, Stand: November 2016
50. Ponemon Institute (2014): Security Metrics to Manage Change: Which Matter, Which Can Be Measured?, Sponsored by Firemon, USA, Michigan 2014
51. Savola, R., Abie, H. (2009): Development of Measurable Security for a Distributed Messaging System, International Journal on Advances in Security, 2009
52. Habenicht, S. (2008): Das GQM-Paradigma, Department für Informatik, Universität Oldenburg, Germany, Oldenburg 2008
53. Basili, V.R., Caldiera, G., Rombach H.D. (1994): Goal Question Metric Paradigm, Encyclopedia of Software Engineering – 2 Volume Set, John Wiley & Sons, Inc., USA 1994
54. Yasasin, E., Schryen, G. (2015): Requirements for IT Security Metrics – An Argumentation Theory Based Approach, Twenty-Third European Conference on Information Systems (ECIS), Germany, Münster 2015
55. Kaur, M., Jones, A. (2008): Security Metrics – A Critical Analysis of Current Methods, Proceeding of the 9th Australian Information Warfare and Security Conference. pp. 41-47
56. Preschern, C., Kajtazovic, N., Höller, A., Kreiner, C. (2014): Quantitative Security Estimation Based on Safety Architecture Design Patterns, Lecture Notes on Software Engineering 2(4), pp. 307-313
57. Heitman, M. (2014): Updated Metrics and values for the CVSS v3.0, URL: https://www.first.org/_assets/downloads/cvss/cvss-v30-preview2-metricvectorstring-december-2014.pdf, Stand: Dezember 2014
58. MindTools (2017): Gap Analysis – Identifying What Needs to be Done in a Project, URL: https://www.mindtools.com/pages/article/gap-analysis.htm, Stand: Jänner 2017
59. Experts Exchange (2009): How Oracle Stores Passwords, URL: https://www.experts-exchange.com/articles/855/How-Oracle-Stores-Passwords.html, Stand: Dezember 2013
60. Oracle (2016): JavaFX Scene Builder, URL: http://www.oracle.com/technetwork/java/javase/downloads/sb2download-2177776.html, Stand: Dezember 2016
61. Zdun, U. (2013): Software Architecture: Patterns for Structuring the Architecture, Faculty of Computer Science, University of Vienna, Austria, Vienna 2013
62. The H Security (2012): Brute-force attack on Oracle passwords feasible, URL: http://www.h-online.com/security/news/item/Brute-force-attack-on-Oracle-passwords-feasible-1714357.html, Stand: September 2012

63. Aboba, B., Calhoun, P., Glass, S., Hiller, T., McCann, P., Shiino, H., Walsh, P. Zorn, G., Dommety, G., Perkins, C., Patil, B., Mitton, D., Manning, S., Beadles, M., Chen, X., Sivalingham, S., Hameed, A., Munson, M., Jacobs, S., Lim, B., Hirschmann, B., Hsu, R., Koo, H., Lipford, M., Campbell, E., Xu, Y., Baba, S., Jaques, E. (2000): Criteria for Evaluating AAA Protocols for Network Access, RFC2989, URL: https://www.ietf.org/rfc/rfc2989.txt, Stand: November 2000
64. Arora, M. (2012): How secure is AES against brute force attacks?, EE-Times, URL: http://www.eetimes.com/document.asp?doc_id=1279619, Stand: Juli 2012
65. Fischer, L. (2013): Einführung in die IT-Sicherheit, Wintersemester 2013, Universität Siegen URL: https://www.wiwi.uni-siegen.de/itsec/lehre/ws-1314/ws13-itsec/lecture_slides.pdf, Stand: 2013
66. Hevner, A. R., March, S.T., Park, J., Ram, S. (2004): Design Science in Information System Research, MIS Quarterly Vol. 28 No. 1, pp. 75-105, USA 2004