Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Sascha Reuter

Aufwertung von Schwachstellenscanreports durch priorisierte Handlungsempfehlungen auf Basis der β-TTC Metrik

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Sascha Reuter

Aufwertung von Schwachstellenscanreports durch priorisierte Handlungsempfehlungen auf Basis der β-TTC Metrik

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

im Studiengang Bachelor of Science Wirtschaftsinformatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Klaus-Peter Kossakowski
Zweitgutachter: Prof. Dr. Michael Köhler-Bußmeier

Eingereicht am: 18.12.2018

Sascha Reuter

Thema der Arbeit
Aufwertung von Schwachstellenscanreports durch priorisierte Handlungsempfehlungen auf Basis der β-TTC Metrik

Stichworte
IT-Sicherheit, IT-Security-Metriken, Time-to-Compromise, Schwachstellen, CVE, CVSS, Openvas

Kurzzusammenfassung
In der heutigen Zeit ist es für Unternehmen wichtiger als jemals zuvor IT-Systeme vor Angriffen zu schützen. Um dies gewährleisten zu können, müssen die Angriffsflächen in den Systemen bekannt sein. Zur Identifikation von Angriffsflächen stehen Schwachstellenscanner bereit, welche auf Grundlage von Prüfmustern mögliche Schwachstellen erkennen. Ziel der Arbeit ist es zu untersuchen, ob mit der erweiterten Cyber Security Metrik β-Time-To-Compromise (β-TTC) die Handlungsempfehlungen für Sicherheitsverantwortliche als Ergebnis von Schwachstellenscans genauer und dem Risiko angemessener priorisiert werden können.

Sascha Reuter

Title of the paper
Enhancement of vulnerability scan reports by means of prioritized recommended actions based on β-TTC metrics

Keywords
IT Security, IT-Security Metrics, Time-to-Compromise, Vulnerabilities, CVE, CVSS, OpenVAS

Abstract
Nowadays it is more important than ever for companies to protect IT systems from attacks. In order to ensure this, the attack surfaces in the systems must be known. To be able to identify the attack surfaces, vulnerability scanners are available, which detect possible vulnerabilities on the basis of pattern. The goal of this thesis is to investigate whether the extended cybersecurity metric β-Time-To-Compromise can be used to prioritize the recommended actions for security managers more precisely and more appropriately as a result of vulnerability scans.

Inhaltsverzeichnis

Abbildungsverzeichnis

Listings

1 Einleitung

1.1 Motivation

In der heutigen Zeit ist es für Unternehmen wichtiger als jemals zuvor IT-Systeme vor Angriffen zu schützen. Um dies gewährleisten zu können, müssen die Angriffsflächen in den Systemen bekannt sein. Zur Identifikation von Angriffsflächen stehen Schwachstellenscanner bereit, welche auf Grundlage von Prüfmustern mögliche Schwachstellen erkennen. Diese Schwachstellenscanner zeigen zwar auf, welches System anfällig ist und durch das Common Vulnerability Scoring System (CVSS) steht eine Bewertung zu jeder Schwachstelle bereit. Daraus kann aber nicht geschlossen werden, welches System dem Risiko ausgesetzt ist, als erstes kompromittiert zu werden.
Diese Bachelorarbeit soll bei der Entscheidungsfindung helfen, welches System als erstes zur Schließung der Schwachstellen betrachtet werden soll. Ein wichtiger Faktor, der in eine solche Entscheidung mit einfließen sollte, ist die Time-to-Compromise (McQueen u. a., 2006). Die Time-to-Compromise ist eine Metrik die angibt, wie lange es dauert, bis ein System kompromittiert wird.

1.2 Ziel der Arbeit

Ziel der Arbeit ist es zu untersuchen, ob mit der erweiterten Cyber Security Metrik β-Time-To-Compromise (β-TTC) die Handlungsempfehlungen für Sicherheitsvorantwortliche als Ergebnis von Schwachstellenscans genauer und dem Risiko angemessener priorisiert werden können.
Im Rahmen dieser Untersuchung wird zunächst ein Prototyp eines Systems entwickelt, der auf dem Output eines Open-Source-Schwachstellenscanner aufsetzt. Damit nicht jedes Mal ein aktueller Schwachstellenscan erfolgen bzw. veränderte Einstellungen mit den gleichen Ausgangsdaten untersucht werden, wird hierfür eine geeignete Schnittstelle gewählt.
Während bei heutigen Schwachstellenscannern üblicherweise anhand des CVSS-Scores der gefundenen Schwachstellen eine Priorisierung erfolgt, sollen für die identifizierten Schwachstellen zusätzlich die β-TTC-Werte berechnet werden. Des Weiteren sollen sinnvolle Konfigurationsmöglichkeiten geschaffen werden, die ein Administrator oder Sicherheitsverantwortlicher

festlegen kann, um seine Bewertungen abzubilden. Hierbei wird es auch notwendig sein, Werte pro System unterschiedlich festlegen zu können, da die einzelnen Systeme eine unterschiedliche Wichtigkeit für ein Unternehmen haben.
Basierend auf den β-TTC-Werten kann dann eine Anzeige der notwendigen Maßnahmen in einer veränderten Prioritätenrangfolge mit der Rangfolge anhand der CVSS-Scores verglichen werden.
Der Prototyp des Systems soll ausschließlich priorisierte Handlungsempfehlungen bestimmen und speichern bzw. anzeigen können. Die für die Speicherung und Anzeige geschaffenen Schnittstellen sollen weiteren Anwendungen erlauben, direkt mit den Ergebnissen arbeiten zu können, z. B. zum Zwecke der Visualisierung oder für weitergehende Auswertungen im Kontext eines Netzwerks.

1.3 Zielgruppe der Arbeit

Diese Arbeit richtet sich an alle Personen, die im Bereich IT-Sicherheit tätig sind und sich mit Schwachstellenmanagement beschäftigen. Hierzu zählen unter anderem Sicherheitsverantwortliche, die sich im Rahmen ihrer Tätigkeit um die Sicherheit der IT-Infrastruktur kümmern. Diese Arbeit soll der Zielgruppe eine Kennzahl aufzeigen, die sie beim Schwachstellenmanagement unterstützen kann.

1.4 Struktur der Arbeit

Die Einleitung führt in das Thema der Bachelorarbeit ein. Hier werden Hauptziel, Teilziele sowie die Zielgruppe der Arbeit dargelegt.
Im folgenden Kapitel werden grundlegende Begriffe wie Schwachstelle, CVE, CVSS und Metrik beschrieben, verschiedene Schwachstellenscanner vorgestellt und ihre Einsatzzwecke erläutert.
Im dritten Kapitel wird eine Anforderungsanalyse durchgeführt, wobei unter anderem die gegebenen Komponenten, die DFN-CERT Advisory-Datenbank und der OpenVAS-Report, vorgestellt werden.
Im vierten Kapitel geht es um das Software Design, also darum wie mit Hilfe der Schwachstellenscanreports und der β-TTC Metrik eine Anwendung konzipiert wird. Dabei wird das Design eines möglichen Produktivsystems und des Prototypen erläutert.
Im fünften Kapitel werden die Implementation des Prototypen vorgestellt und die Ergebnisse präsentiert.
Den Abschluss der Arbeit bildet Kapitel sechs mit Fazit und Ausblick.

2 Grundlagen

In diesem Kapitel wird zunächst eine Einführung in die Terminologie des Schwachstellenmanagements gegeben. Danach wird das Risikomanagement erläutert, welches dem Schwachstellenmanagement überordnet ist. Außerdem werden die zugehörigen IT-Sicherheitsmetriken vorgestellt, darunter die Time-To-Compromise (TTC) und die β-Time-To-Compromise (β-TTC), als eine Erweiterung dieser.

2.1 Schwachstellenmanagement

> „Schwachstellenmanagement bezeichnet die zyklische Ausübung der Identifikation, Klassifizierung, Beseitigung und Abschwächung von Schwachstellen“ (Foreman, 2009).

Das Schwachstellenmanagement umfasst mehrere Schritte und ist mehr als nur das Scannen nach Schwachstellen. Abbildung 2.1 zeigt eine mögliche Vorgehensweise. Die einzelnen Schritte in diesem vereinfachten Modell können dabei von mehreren Lösungen begleitet werden. Ein vollständiges Schwachstellenmanagement benötigt demnach verschiedene Komponenten, um verlässlich eine Aussage darüber treffen zu können, wo mögliche Angriffsflächen bestehen. Verschiedene Unternehmen bieten dafür Softwarelösungen an. Der erste Schritt, das „Erfassen“, kann dabei klassisch von einem Schwachstellenscanner durchgeführt werden. Es gibt aber auch Lösungen, in denen die Konfiguration der IT-Landschaft abgebildet werden kann, damit Schwachstellen auch ohne Scan erkannt werden. Dies bietet sich für Systeme an, die nicht durch die zusätzliche Last eines Scans ausgebremst werden sollen. Schwachstellen werden meist auf Grundlage der CVSS-Metrik bewertet und liefern dem Schwachstellenmanagement einen wichtigen Parameter für den nächsten Schritt, die „Priorisierung“. Des Weiteren bieten kommerzielle Lösungen eine Einordnung von Standorten und Regionen oder der Wichtigkeit der Systeme an, wodurch eine genauere Priorisierung durchgeführt werden kann (Steinke, 2015). Weiterhin müssen die gewonnen und verarbeiteten Informationen geprüft, ein Bericht erstellt, die einzelnen Schwachstellen behoben und die durchgeführte Arbeit überprüft werden. Danach wiederholt sich der Zyklus.

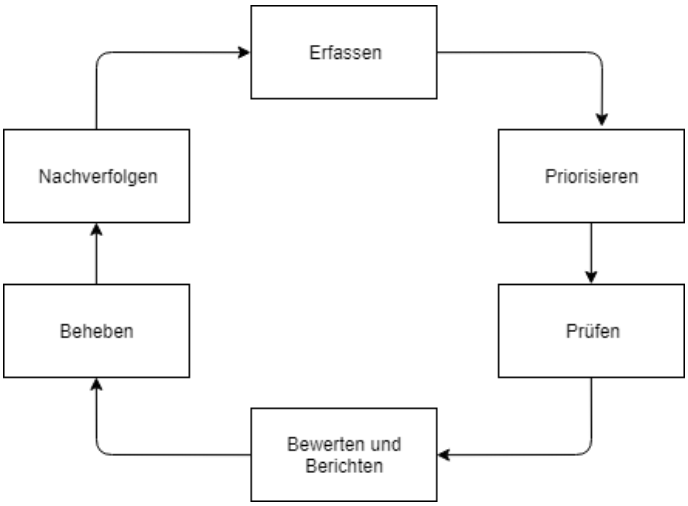

Abbildung 2.1: Schwachstellenmanagement: Zyklus
Quelle: Modifiziert übernommen aus (Casper und Strobel, 2018)

2.1.1 Risikomanagement

Stoneburner u. a. (2002) beschreiben den Begriff Risiko im Zusammenhang mit Informationssystemen wie folgt:

> „*Risk* is a function of the *likelihood* of a given threat-source's exercising a particular potential *vulnerability*, and the resulting *impact* of that *adverse event* on the organization" (Stoneburner u. a., 2002).

Ein solches „unerwünschtes Ereignis" ist im Rahmen dieser Arbeit die gezielte Ausnutzung einer Schwachstelle eines Computersystems. Das Risiko, welches also berücksichtigt werden muss, ist die Kompromittierung eines Computersystems (Mantel, 2018). Stoneburner u. a. (2002) gibt dafür einen Prozess für das Risikomanagement in neun Schritten an.

1. *System Characterization* ist die Beschreibung des Computersystems. Dazu gehören alle Informationen, welche das System betreffen. Dies sind z. B. Hardwarekomponenten, Software, Systemschnittstellen, Personen die das System benutzen, usw.

2. *Threat Identification* bestimmt die möglichen Gefahrenquellen. Es soll eine Bedrohungsaussage erstellt werden, welche potentielle Gefahrenquellen auflistet.

3. *Vulnerability Identification* ist die Bestimmung von Schwachstellen in einem System. Es soll eine Liste von Schwachstellen erstellt werden, welche von den möglichen Gefahrenquellen ausgenutzt werden kann.

4. *Control Analysis* ist die Analyse von aktuellen und geplanten Maßnahmen, um die Wahrscheinlichkeit des Eintretens einer Bedrohung und damit die möglichen Auswirkungen zu minimieren.

5. *Likelihood Determination* ist die Ermittlung der Wahrscheinlichkeit, dass eine mögliche Schwachstelle in einer Bedrohungsumgebung ausgenutzt werden kann.

6. *Impact Analysis* ist die Analyse, um den möglichen Schaden zu bestimmen, der durch den Verlust der drei Kriterien Integrität, Verfügbarkeit und Vertraulichkeit entstehen kann.

7. *Risk Determination* ist die Ermittlung des Risikos für eine Systemkomponente. Diese kann als die Kombination von Wahrscheinlichkeits- und Auswirkungsanalyse angesehen werden. Über eine Risiko-Level-Matrix kann das Risiko beispielsweise bestimmt werden, was zu drei möglichen Ergebnissen führt: hoch, mittel und niedrig.

8. *Control Recommendations* ist die Bestimmung von Empfehlungen, um die Risiken abzuschwächen oder zu beseitigen.

9. *Results Documentation* ist die Dokumentation der Ergebnisse nach dem Abschluss der Risikobewertung in einem offiziellen Bericht.

2.1.2 Schwachstelle

Eine Schwachstelle im Kontext der Informationssicherheit ist ein unbeabsichtigter Fehler oder eine Schwäche im Programmcode oder in einem System, die es einem Angreifer ermöglicht die Integrität, Verfügbarkeit oder Vertraulichkeit des Systems zu stören (Wang und Yang, 2017). Damit eine Schwachstelle ausgenutzt werden kann, müssen aber verschiedene Bedingungen zutreffen. Als erstes muss eine Schwachstelle in einem System bestehen. Zweitens muss ein Angreifer Zugriff auf die Schwachstelle haben und drittens muss ein Angreifer die Fähigkeit besitzen diese auszunutzen. Eine Schwachstelle ist deshalb immer eine Sicherheitslücke, die behoben werden muss.

2.1.3 Common Vulnerabilities and Exposures (CVE)

> „CVE, [...] is a method used to assign identiers to publicly known vulnerabilities found in IT products and to provide information (e.g., aected products) about the vulnerabilities. Across organizations, anti-virus vendors, and security experts, CVE

> has become the de facto standard to share information on known vulnerabilities and exposures“ (Sanguino und Uetz, 2017).

Eine CVE hilft also eine Schwachstelle eindeutig zu identifizieren. Dadurch kann eindeutig über eine Schwachstelle diskutiert werden. Das CVE-Verzeichnis wurde dabei von MITRE ins Leben gerufen und wird vom US-CERT des U.S. Department of Homeland Security gefördert. Eine CVE-ID ist nach folgender Form aufgebaut:

```
1 CVE - YYYY - NNNN[N]*
```

Listing 2.1: Aufbau einer CVE-ID

Die CVE-IDs werden von aktuell 92 sogenannten CVE Numbering Authorities (CNAs) vergeben (MITRE1, 2018). Dazu zählen Computer Emergency Response Teams (CERTs), eine Reihe von Softwareherstellern und MITRE selbst.

2.1.4 National Vulnerability Database (NVD)

Die National Vulnerability Database (NVD) ist eine Schwachstellen-Datenbank, die auf dem CVE-Verzeichnis aufbaut und vollständig mit ihm synchronisiert ist. Sie wurde vom National Institute of Standards and Technology (NIST) eingeführt. Diese Datenbank ist öffentlich verfügbar und jeder Eintrag (Schwachstelle) hat einen eindeutigen Bezeichner. Zu jeder Schwachstelle können zusätzliche Informationen, wie eine CPE (Common Platform Enumeration), ein eindeutiger Bezeichner für ein Produkt, ein CVSS-Scoring, welches im nächsten Abschnitt erläutert wird und eine Referenz, in der über das Vorhandensein dieser Schwachstelle berichtet wird, bekannt sein (MITRE2, 2018).

2.1.5 Common Platform Enumeration (CPE)

Um Schwachstellen eindeutig Anwendungen, Hardware und Betriebssystemen zuordnen zu können, gibt es die Common Platform Enumeration (CPE). Über CPE-Bezeichner kann angegeben werden, welches Produkt in welcher Ausführung von einer Schwachstelle betroffen ist (Sanguino und Uetz, 2017). Eine CPE ist nach dem Schema von Uniform Resource Identifiers (URI) aufgebaut und enthält folgende Informationen, wie sie von Buttner und Ziring (2009) angegeben wird:

1. **Part** gibt an, um welche Art es sich handelt. Hier gibt es drei verschiedene Arten: Hardwarekomponente, Betriebssystem, Anwendung.

2. **Vendor** identifiziert den Hersteller oder Anbieter des Systems, z. B. Microsoft
3. **Products** ist der allgemeine Name des Produkts, z. B. Windows XP
4. **Version** gibt die Version des Produkts an, z. B. Version 62.04
5. **Update** wird für Update- oder Service Pack-Informationen verwendet, z. B. Service Pack 3 (SP3)
6. **Edition** wird verwendet, um die Edition des Produkts zu bestimmen, z. B. 64-Bit

```
1 cpe:/ {part} : {vendor} : {product} : {version} : {update} :
    ↪ {edition}
```

Listing 2.2: Aufbau einer CPE

Das CPE-Verzeichnis kann über die National Vulnerability Database (NVD) online abgerufen werden und ist wie das CVE-Verzeichnis öffentlich zugänglich.

2.2 Schwachstellenscanner

Ein Schwachstellenscanner ist ein Programm, das es ermöglicht, nach Schwachstellen in einem System oder Netzwerk zu suchen (Wang und Yang, 2017). Sicherheitsbeauftragte oder Administratoren können mithilfe eines Schwachstellenscanners Sicherheitslücken in Systemen aufdecken und diese danach mit entsprechenden Patches beheben. Dabei erstellen Schwachstellenscanner in der Regel detaillierte Berichte mit dem Schweregrad jeder erkannten Schwachstelle.
In einigen Scannern werden dazu die eindeutigen Bezeichner einer Schwachstelle, die CVE-IDs mit angegeben. Es gibt verschiedene Schwachstellenscanner, wobei einige frei verfügbar und andere kostenpflichtig sind. Da in dieser Arbeit mit kostenfreier Software gearbeitet werden soll, ist der wohl bekannteste Schwachstellen Nessus (Nessus, 2018) nicht geeignet. OpenVAS (OpenVAS, 2018) ist ein Fork von Nessus und wird seit der Abspaltung 2005 als kostenfreie Software weiterentwickelt. Den meisten als Portscanner bekannt, kann aber auch Nmap (Lyon, 2009) zum Auffinden von Schwachstellen genutzt werden. Metaspolit (Maynor, 2011) ist unter anderem ein Framework zum Ausführen von Exploits, welches auch als Schwachstellenscanner genutzt werden kann. Dafür werden innerhalb von Metasploit Tools wie Nmap oder OpenVAS benutzt.

2.3 IT-Security-Metriken

Hier werden die IT-Security-Metriken, die in dieser Arbeit relevant sind, vorgestellt. Dies sind das Common Vulnerability Scoring System (CVSS), die Time-To-Compromise (TTC) und die β-Time-To-Compromise (β-TTC).

2.3.1 Common Vulnerability Scoring System (CVSS)

Das Common Vulnerability Scoring System (CVSS) ist eine Metrik, welche die mögliche Gefahr einer Schwachstelle beschreibt. Da die bereitgestellte Datenbank (Kapitel 3.1) CVSS in Version 2.0 verwendet, wird diese hier beschrieben. CVSS besteht aus drei Gruppen von Metriken, die bei der Beurteilung mit einbezogen werden: **Base, Temporal und Environmental.** Um eine Bewertung zu erzeugen muss dabei die Base Gruppe angegeben werden, die anderen beiden Gruppen sind optional. Jede einzelne Gruppe erzeugt einen numerischen Wert im Bereich 0.0 bis 10.0 und daraus einen Vektor, der die komprimierte Darstellung dieser Metrik widerspiegelt. Die **Base Score Metrics** Gruppe repräsentiert dabei die wesentliche Qualität der Schwachstelle. Folgende Metriken sind in dieser Gruppe vorhanden (Mell u. a., 2006):

1. **Access Vector (AV)**: Gibt die räumliche Entfernung eines Angreifers an. Hierbei gibt es drei Möglichkeiten: es handelt sich um einen lokalen Angreifer, einen Angreifer im benachbarten Netzwerk oder einen entfernten Angreifer.

2. **Access Complexity (AC)**: Gibt die Komplexität des Angriffs an. Mögliche Werte sind niedrig, mittel und hoch.

3. **Authentication (Au)**: Wie und in welchem Maße muss ein Angreifer sich authentifizieren, um die Schwachstelle auszunutzen? Mögliche Werte sind keine, einfache oder mehrfache Authentifizierung.

4. **Confidentiality Impact (C)**: Gibt an, wie die Vertraulichkeit betroffen ist. Die Vertraulichkeit kann komplett, teilweise oder nicht betroffen sein.

5. **Integrity Impact (I)**: Gibt an, wie die Integrität betroffen ist. Die Integrität kann komplett, teilweise oder nicht betroffen sein.

6. **Availability Impact (A)**: Gibt an, ob ein Angreifer die Verfügbarkeit eines Systems oder einzelner Komponenten stören kann. Die Verfügbarkeit kann komplett, teilweise oder nicht betroffen sein.

Dabei geben die ersten drei Metriken an, wie ein Produkt angegriffen werden kann und ob zusätzliche Bedingungen oder Benutzerrechte erforderlich sind. Die letzten drei Metriken geben den Einfluss an, den ein Angreifer auf ein System nehmen kann. Die Base Gruppe gibt einen Anhaltspunkt, durch den abgeschätzt werden kann, wie groß die Bedrohung für ein System ist.
Der generische Vektor, der sich aus der Bewertung der **Base Score Metrics** Gruppe ergibt, sieht folgendermaßen aus:

```
1 (AV:[L,A,N]/AC:[L,M,H]/Au:[N,S,M]/C:[N,P,C]/I:[N,P,C]/A:[N,P,C])
```

Listing 2.3: Generischer CVSS Vektor mit der Gruppe Base

Die erste optionale Metrik-Gruppe ist die Gruppe **Temporal Score Metrics**. Sie umfasst drei Metriken, die angeben, ob ein Exploit oder eine Möglichkeit zur Behebung der Schwachstelle verfügbar und ob die Informationen über die Schwachstelle vertrauenswürdig sind.

1. **Exploitability (E)**: Bei der Ausnutzbarkeit gibt es fünf verschiedene Möglichkeiten, diese anzugeben. Diese Möglichkeiten sind Not Defined (ND), Unproven (U), Functional (F), High (H) und Proof-of-Concept (POC).

2. **Remediation Level (RL)**: Bei den Möglichkeiten eine Schwachstelle zu beheben stehen folgende Möglichkeiten zur Auswahl: Official Fix (OF), Temporary Fix (TF), Workaround (W), Unavailable (U) und Not Defined (ND).

3. **Report Confidence (RC)**: Diese Metrik misst den Grad des Vertrauens in die Existenz der Schwachstelle. Hier stehen folgende zur Auswahl: Unconfirmed (UC), Uncorroborated (UR), Confirmed (C) und Not Defined (ND).

Diese Gruppe kann sich, wie der Name schon sagt, über die Zeit verändern. Bei der Verfügbarkeit eines offiziellen Patches verringert sich der Score signifikant, wobei die Verfügbarkeit eines Exploits diesen erhöht.
Der generische Vektor der die Gruppen **Base Score Metrics** und **Temporal Score Metrics** enthält, hat folgendes Format.

```
1 (AV:[L,A,N]/AC:[L,M,H]/Au:[N,S,M]/C:[N,P,C]/I:[N,P,C]/A:[N,P,C]/
    ↪ E:[U,POC,F,H,ND]/RL:[OF,TF,W,U,ND]/RC:[UC,UR,C,ND])
```

Listing 2.4: Generischer CVSS Vektor mit den Gruppen Base und Temporal

Die letzte Gruppe **Environmental Score Metrics** bezieht sich auf die spezielle IT-Umgebung, in der eine Schwachstelle vorhanden ist.

1. **Collateral Damage Potential (CDP)**: Gibt den wirtschaftlichen Verlust von Produktivität oder Umsatz an.
2. **Target Distribution (TD)**: Über diese Metrik kann der Prozentsatz der Systeme angegeben werden, die von der Schwachstelle betroffen sein könnten.
3. **Security Requirements (CR, IR, AR)**: Bei dieser Metrik kann die Bedeutung der verschiedenen Einflussarten Vertraulichkeit, Integrität und Verfügbarkeit auf ein System bewertet werden.

Der generische Vektor mit allen Gruppen hat folgendes Format:

```
1 (AV:[L,A,N]/AC:[L,M,H]/Au:[N,S,M]/C:[N,P,C]/I:[N,P,C]/A:[N,P,C]/
    ↪ E:[U,POC,F,H,ND]/RL:[OF,TF,W,U,ND]/RC:[UC,UR,C,ND]/
    ↪ CDP:[N,L,LM,MH,H,ND]/TD:[N,L,M,H,ND]/
    ↪ CR:[L,M,H,ND]/IR:[L,M,H,ND]/AR:[L,M,H,ND])
```

Listing 2.5: Generischer CVSS Vektor mit den Gruppen Base, Temporal und Environmental

CVSS ermöglicht es allen Interessengruppen, in einer einheitlichen Sprache über die Bewertung von IT-Schwachstellen zu sprechen und über diese zu diskutieren.

2.3.2 Time-To-Compromise (TTC)

Time-To-Compromise (TTC) von McQueen u. a. (2006) ist eine IT-Sicherheitsmetrik, welche den Sicherheitsstatus einer Systemkomponente bewertet. Sie ist dabei definiert als die Zeit, die ein Angreifer benötigt, um ein gewisses Maß an Privilegien auf einer Systemkomponente zu erlangen (McQueen u. a., 2006). Dafür wurde ein Prozess mit folgenden drei Unterprozessen beschrieben.

- **Prozess 1**: Mindestens eine Schwachstelle ist auf einer Systemkomponente bekannt, für die ein Exploit verfügbar ist, mit der ein Angreifer gewisse Berechtigungen erreichen kann.
- **Prozess 2**: Mindestens eine Schwachstelle ist auf einer Systemkomponente bekannt, für die kein Exploit verfügbar ist.

- **Prozess 3**: Keine Schwachstelle ist auf einer Systemkomponente bekannt. Steht für die Identifizierung neuer Schwachstellen und Exploits, auch bekannt als Zero-Day-Exploit.

Time-To-Compromise (TTC) gibt die erwartete Zeit an, die benötigt wird, um ein System mit v Schwachstellen zu kompromittieren, unter Berücksichtigung des Parameters k für die Gesamtzahl der bekannten Schwachstellen und des Parameters s für die Fertigkeit eines Angreifers.
Die nachfolgende Definition ist eine Zusammenfassung von McQueen u. a. (2006) erstellt von Zieger u. a. (2018).

Definition 1 (Originale Time-To-Compromise (TTC) (siehe McQueen u. a. (2006))) *Let $\mathcal{S} = \{novice, beginner, intermediate, expert\}$ be a set of discrete skill levels. Let the number of vulnerabilities of the component be $v \in \mathbb{N}$, and let $s \in \mathcal{S}$ be the skill level of the adversary, and $k \in \mathbb{N}$ the total number of vulnerabilities, then the* time-to-compromise

$$TTC : \mathbb{N} \times \mathcal{S} \times \mathbb{N} \to \mathbb{R}$$

is defined as

$$TTC(v, s, k) = t_1 \cdot P_1 + t_2 \cdot (1 - P_1) \cdot (1 - u) + t_3 \cdot u \cdot (1 - P_1)$$

with

- $t_1 = c_1$ *as the average time it takes to tune an available exploit,*
- $P_1 = 1 - e^{-v \cdot \frac{m(s)}{k}}$ *as the probability to have an exploit at hand,*
- $t_2 = c_2 \cdot E(v, s)$ *as the time to write an exploit times the estimated tries it takes,*
- $t_3 = (\frac{1}{f(s)} - 0.5) \cdot c_3 + c_2$ *as the time to find/wait for a new usable vulnerability and creation of a working exploit, and*
- $u = (1 - f(s))^v$ *as the probability that process 2 is unsuccessful.*

The estimated tries in process 2 as a function of vulnerabilities and skill are defined as follows:

$$E(v, s) = f(s) \cdot \left(1 + \sum_{t=2}^{v - v \cdot f(s) + 1} \left[t \cdot \prod_{i=2}^{t} \left(\frac{v \cdot (1 - f(s)) - i + 2}{v - i + 1} \right) \right] \right)$$

with f and m as function of skill:

- $m : \mathcal{S} \to \mathbb{N}$ *giving the number of readily available exploits for the given skill level,*
- $f : \mathcal{S} \to [0, 1]$ *providing the fraction of vulnerabilities usable at the given skill level,*

with the following constants:

- $c_1 = 1\text{d}$ *as the time to tune and use a readily available exploit*
- $c_2 = 5.8\text{d}$ *as the time to develop a new exploit*
- $c_3 = 32.42\text{d}$ *as the avg. time for a new vulnerability to occur*

2.3.3 β-Time-To-Compromise (β-TTC)

Zieger u. a. (2018) haben einige Probleme bezüglich der Time-To-Compromise (TTC) entdeckt und Verbesserungen an dieser vorgenommen. Zum einen wurden die vier möglichen Fertigkeitsstufen $\{\text{novice}, \text{beginner}, \text{intermediate}, \text{expert}\}$ durch ein Intervall $[0, 1]$ ersetzt, um auszudrücken, dass es mehr Abstufungen der Fertigkeiten gibt. Zum anderen wurde eine mathematische Anomalie behoben, die durch eine steigende Anzahl von Schwachstellen einen höheren TTC-Wert liefert, also ein System als sicherer bewertet. Dies wurde durch eine Anpassung an der Funktion E (siehe Definition 1), zur Schätzung von Versuchen, durchgeführt. Dies hat die Funktion insgesamt monoton fallend werden lassen.

$$\xi(a, v) = \frac{a}{v} \cdot \left(1 + \sum_{t=2}^{\lfloor v \cdot (1 - \frac{a}{v}) \rfloor + 1} \left[t \cdot \prod_{i=2}^{t} \left(\frac{v \cdot (1 - \frac{a}{v}) - i + 2}{v - i + 1} \right) \right] \right)$$

$$E(s, v) = \xi(\lfloor f(s) \cdot v \rfloor, v) \cdot (\lceil f(s) \cdot v \rceil - f(s) \cdot v) + \xi(\lceil f(s) \cdot v \rceil, v) \cdot (1 - \lceil f(s) \cdot v \rceil + f(s) \cdot v)$$

β-Time-To-Compromise (β-TTC) von Zieger u. a. (2018) ist, genau wie die TTC von McQueen u. a. (2006), eine IT-Sicherheitsmetrik, welche den Sicherheitsstatus einer Systemkomponente bewertet. Die β-TTC benötigt im Gegensatz zur TTC nur zwei Argumente. Dies ist einmal k, die Gesamtzahl der Schwachstellen und v, die Anzahl der bekannten Schwachstellen einer Systemkomponente. Der Parameter s (Fähigkeit) wird nicht als Argument übergeben. Wie oben erwähnt, wird stattdessen ein Intervall $[0, 1]$ verwendet. Dies wird von Zieger u. a. (2018) über eine β-Verteilung mit den Werten $\alpha = 1, 5$ und $\beta = 2$ als Fähigkeitswert umgesetzt. Dadurch wird erreicht, dass Angreifer mit verschiedenen Fähigkeitsstufen mit einer gewissen Wahrscheinlichkeit angreifen.

Definition 2 (β-Time-To-Compromise (siehe Zieger u. a. (2018)))

$$\beta TTC(v,k) = \int_0^1 TTC(v,s,k) \cdot \mathrm{Beta}_{1.5,2.0}(s)ds$$

β-Time-To-Compromise (TTC) gibt die erwartete Zeit an, die benötigt wird, um ein System mit v Schwachstellen zu kompromittieren, unter Berücksichtigung des Parameters k für die Gesamtzahl der bekannten Schwachstellen. Die Variable s für die Fähigkeit muss nicht als Parameter angegeben werden.

2.3.4 Typisierte Time-To-Compromise

Zieger u. a. (2018) haben darüber hinaus weitere Time-To-Compromise eingeführt. Diese werden von den Parametern Vertraulichkeit (TTCc), Verfügbarkeit (TTCa) und Integrität (TTCi) beeinflusst. Da diese Parameter den Einfluss auf eine Systemkomponente bestimmen, wäre es zu sehr vereinfacht, diese gemeinsam in einer einfachen Zahl zusammenzufassen. Deshalb wurden diese Time-To-Compromise eingeführt, um zwischen verschiedenen Arten von Auswirkungen zu unterscheiden. Eine weitere Time-To-Compromise, die Zieger u. a. (2018) einführt, bezieht sich darauf, wie eine Schwachstelle ausgenutzten werden kann (TTCe). Dabei wird die Komplexität des Angriffs, das Authentifizierungslevel und das mögliche Vorhandensein eines Exploits berücksichtigt. Bei diesen Konzepten müssen jeweils Teilmengen von Schwachstellen berücksichtigt werden, die für eine Gefährdung durch den jeweiligen Typ verantwortlich sind. Dazu kann CVSS verwendet werden, da diese Metrik die Parameter Vertraulichkeit, Verfügbarkeit und Integrität mitbringen, um eine Unterscheidung zu treffen.

3 Anforderungsanalyse

In Kapitel 2.1 wurden das Schwachstellenmanagement eingeführt und wichtige IT-Security Metriken vorgestellt. Im folgendem Kapitel werden die gegebenen Komponenten und bestehende Software analysiert. Dazu gehören eine Advisory-Datenbank mit Schwachstelleninformationen und der Schwachstellenscanner OpenVAS. Des Weiteren werden die einzelnen Schritte des Risikomanagementprozesses erläutert, die durch den zu entwickelnden Prototyp unterstützt werden sollen. Weiterhin werden die Kriterien für eine priorisierte Rangfolge vorgestellt.

3.1 Advisory-Datenbank

Eine Komponente, die für den Prototyp benötigt wird, ist eine Advisory-Datenbank, welche die DFN-CERT Services GmbH für diese Arbeit zur Verfügung stellt. Diese Datenbank enthält Informationen über die Schwachstellen einschließlich des CVSS-Scores, Informationen zu CPEs und die Advisories selbst. Die verwendete Datenbank ist PostgreSQL in der Version 9.3. In Abbildung 3.1 wird ein Überblick über die in dieser Arbeit verwendeten Tabellen gegeben. Dies ist zum einen die Tabelle `authoring_vulnerability`, welche die Schwachstellen enthält. Diese wird für die Zuordnung zu den gefunden Schwachstellen aus dem Schwachstellenreport benutzt. Des Weiteren enthält diese Tabelle den CVSS-Vektor sowie Texte zu Beschreibung und Auswirkung der Schwachstelle. Hierbei ist zu erkennen, dass der CVSS-Vektor die Base Score Metrics und die Temporal Score Metrics enthält. Die Tabellen `authoring_vulnerabilitycpereleation` und `cpe_cpe` werden zusätzlich benötigt, um die verwundbaren Produkte zu identifizieren.

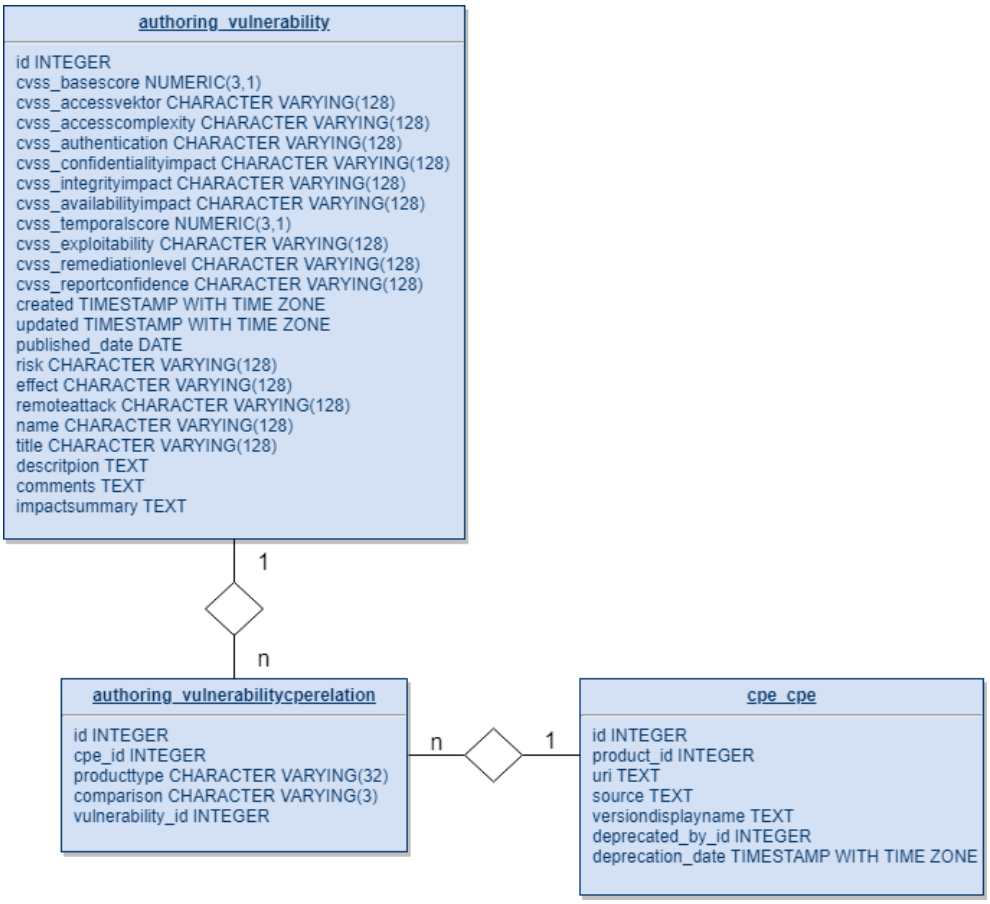

Abbildung 3.1: Datenmodell: Teil des Entity-Relationship-Modell der Advisory-Datenbank

3.2 Schwachstellenscanner OpenVAS

Der Schwachstellenscanner, der in dieser Arbeit verwendet wird, ist OpenVAS. OpenVAS ist ein Framework, welches aus mehreren Tools besteht und verschiedene Einstellungsmöglichkeiten für Scans bietet. Durch die Verknüpfung der gefundenen Schwachstellen mit aktuellen CVEs ist dieser Schwachstellenscanner genau das richtige Werkzeug für diese Arbeit, da mit den CVEs weitergearbeitet werden soll.

3.2.1 Analyse OpenVAS Report

OpenVAS wurde verwendet, um in einem Netzwerk mit verschiedenen Systemen einen Schwachstellenscan durchzuführen. Dabei wurde ein Report im XML-Format generiert, welcher

in der Arbeit als Eingabe verwendet wird, um für die Systeme eine Rangfolge zu bestimmen. Dabei fällt auf, dass OpenVAS selbst keine Rangfolge vorschlägt, welche Systeme bevorzugt behandelt werden sollen. Die gefunden Schwachstellen in allen Systemen werden lediglich nach CVSS-Score sortiert aufgelistet. Das heißt, im XML-Format findet keine Priorisierung der einzelnen Systeme, sondern nur eine Auflistung mit den jeweiligen Schwachstellen statt.
Zu Anschauungszwecken wurde der Report zusätzlich im PDF-Format exportiert (Abbildung 3.2). Hierbei ist aufgefallen, dass der PDF-Konverter die Schwachstellen für die jeweiligen Systeme zusammenfasst. Dies macht OpenVAS anhand der Portnummern und der Kriterien *high*, *medium* und *low*, die den Schwachstellen zugeordnet sind. Das heißt, ein Port kann auch mehrfach auftauchen. Bei der Sortierung im PDF-Format kann eine Priorisierung gesehen werden, auch wenn OpenVAS dies in dem Report nicht dokumentiert hat.

Abbildung 3.2: OpenVAS PDF Report: Inhaltsverzeichnis

Jede Schwachstelle enthält eine Reihe von Informationen wie zum Beispiel einen CVSS-Score, welches Protokoll betroffen ist, eine Erläuterung, warum die Systemkomponente betroffen ist, ein Ergebnis der Schwachstellenerkennung, einen Lösungsvorschlag, die Auswirkungen,

Referenzen usw. Es sind aber nicht immer für eine Schwachstelle alle Informationen verfügbar. Die Abbildung 3.3 zeigt die Beschreibung einer Schwachstelle im PDF-Format.

Medium (CVSS: 4.3)
NVT: Apache Web Server ETag Header Information Disclosure Weakness

Summary
A weakness has been discovered in Apache web servers that are configured to use the FileETag directive.

Vulnerability Detection Result
Information that was gathered:
Inode: 198461
Size: 54

Impact
Exploitation of this issue may provide an attacker with information that may be used to launch further attacks against a target network.

Solution
OpenBSD has released a patch that addresses this issue. Inode numbers returned from the server are now encoded using a private hash to avoid the release of sensitive information.
Novell has released TID10090670 to advise users to apply the available workaround of disabling the directive in the configuration file for Apache releases on NetWare. Please see the attached Technical Information Document for further details.

Vulnerability Detection Method
Due to the way in which Apache generates ETag response headers, it may be possible for an attacker to obtain sensitive information regarding server files. Specifically, ETag header fields returned to a client contain the file's inode number.
Details:Apache Web Server ETag Header Information Disclosure Weakness
OID:1.3.6.1.4.1.25623.1.0.103122
Version used: $Revision: 6700 $

References
CVE: CVE-2003-1418
BID:6939
Other:
URL:https://www.securityfocus.com/bid/6939
URL:http://httpd.apache.org/docs/mod/core.html#fileetag
URL:http://www.openbsd.org/errata32.html
URL:http://support.novell.com/docs/Tids/Solutions/10090670.html

Abbildung 3.3: OpenVAS PDF Report: Informationen zu einer Schwachstelle

Der Aufbau des Reports im XML-Format ist in Abbildung 3.4 stark vereinfacht dargestellt. Der erste Tag '<report id=...>' steht dabei für den kompletten Report. Der zweite Tag spezifiziert einen Unterreport. In dem ausgewählten OpenVAS-Report tritt dieser nur einmalig auf. Falls

ein Schwachstellenscan mit mehreren verschiedenen Scans, also mit unterschiedlichen Konfigurationen, in einem Task durchgeführt wird, sind diese unter mehreren '<report id=...>' Tags zu finden. Der Tag '<results ...>' umfasst alle Schwachstellen, die identifiziert worden sind. Jede gefundene Schwachstelle wird unter einem eigenen Tag '<result id=...>' gelistet. Damit hier eine Zuordnung zu dem jeweiligen Host stattfinden kann, gibt es in jedem '<result id=...>' einen Tag '<host>', der eine IP-Adresse enthält. Ein weiterer Tag, der hohe Relevanz für diese Arbeit hat, ist '<nvt>'. Hier sind Informationen wie Typ, Name, CVSS-Score, CVE-IDs, externe Referenzen, CVSS Base Vektor, welcher in Kapitel 2.3.1 erläutert wurde und eine Kurzbeschreibung der Schwachstelle enthalten. Also die Informationen, die auch schon im PDF-Report identifiziert worden sind.

```
<report id=...>
    <report id=...>
        <results ...>
            <result id=..>
                <host>192.168.0.101 ... </host>
                <nvt ...>
                    <type>nvt</type>
                    <name>SSL/TLS: Report Vulnerable Cipher Suites for HTTPS</name>
                    <cvss_base>5.0</cvss_base>
                    <cve>CVE-2016-2183, CVE-2016-6329</cve>
                    <xref>URL:https://mozilla.github.io/server-side-tls/ssl-config-generator/, ...</xref>
                    <tags>cvss_base_vector=AV:N/AC:L/Au:N/C:P/I:N/A:N|summary=This routi... </tags>
                </nvt>
            </result>
            <result id=..>
            ...
            </result>
            ....
        </results>
        <host>
            <ip>192.168.0.101</ip>
            ....
        </host>
        <host>
            <ip>192.168.0.103</ip>
            ....
        </host>
        <host>
            <ip>192.168.0.102</ip>
            ....
        </host>
    </report>
</report>
```

Abbildung 3.4: OpenVAS vereinfachter XML Report: XML Struktur

Wie schon erwähnt, sind nicht immer alle Informationen für jede Schwachstelle vorhanden. Da die gefunden Schwachstellen aber fast alle CVE-IDs enthalten, können zumindest in diesem Fall die benötigten Informationen aus der Advisory-Datenbank abgefragt werden. Die gefundenen Schwachstellen ohne CVE-ID werden nicht weiter betrachtet, da ohne eindeutigen Bezeichner keine weiteren Informationen gewonnen werden können. Die Schwachstellen, die nicht in der Advisory-Datenbank enthalten sind, werden trotzdem aufgenommen. Diesen fehlen dann zwar detaillierte Informationen, sie können aber bei der Priorisierung berücksichtigt werden.

3.3 Risikomanagementprozess

In Kapitel 2.1.1 wurden die Schritte eines Risikomanagementprozesses aufgezeigt. Im folgenden wird auf die Schritte eingegangen, welche der zu entwickelnde Prototyp unterstützen soll.

3.3.1 System Characterization

Der erste Schritt im Risikomanagementprozess ist die Beschreibung des Computersystems. Diese wird hier nicht explizit durchgeführt, aber durch den Schwachstellenscan kann die vorhandene Software und auch das Betriebssystem erkannt werden. Diese Informationen sind aber nicht in jedem Fall vollständig, da die Erkennung des Betriebssystems nicht immer funktioniert und auch nur Software identifiziert werden kann, die eine Schwachstelle enthält.

3.3.2 Threat Identification

Die Bedrohung, die im Kontext der Arbeit behandelt werden muss, ist die Ausnutzung einer Schwachstelle.

3.3.3 Vulnerability Identification

Die Schwachstellenerkennung wird von einem Schwachstellenscanner durchgeführt. Dieser untersucht auf Grundlage von Prüfmustern die Systeme in einem Netzwerk und liefert Informationen zu den gefundenen Schwachstellen. In diesen Informationen ist, wie in Kapitel 3.2.1 erläutert, der eindeutige Bezeichner, die CVE-ID, vorhanden. Für die weitere Verarbeitung werden die CVEs mit der in Kapitel 3.1 vorgestellten Advisory-Datenbank abgeglichen und die benötigten Informationen aus dieser exportiert.

3.3.4 Control Analysis

Die Kontrollanalyse von aktuellen und geplanten Maßnahmen ist nicht Bestandteil dieser Arbeit.

3.3.5 Likelihood Determination

Zur Ermittlung der Wahrscheinlichkeit, dass die Bedrohung tatsächlich eintritt, schlägt Stoneburner u. a. (2002) vor, die Bedrohung in die drei Kategorien hoch, mittel und niedrig einzuteilen. Durch die Berechnung der β-Time-To-Compromise wird die Wahrscheinlichkeit bestimmt, dass eine Bedrohung tatsächlich eintritt, also ein System ausgebeutet wird. Es wird darüber hinaus keine Einteilung in Kategorien vorgenommen.

3.3.6 Impact Analysis

Bei der Auswirkungsanalyse müssen die Kriterien Integrität, Verfügbarkeit und Vertraulichkeit berücksichtigt werden. Diese drei Kriterien bringt jede Schwachstelle über den CVSS-Score mit. Die typisierten Time-To-Compromise könnte dazu verwendet werden, die Berechnung auf Grundlage der Kriterien durchzuführen. Diese TTCs sollen in späteren Implementierungen berücksichtigt werden. In dieser Arbeit wird der Parameter „Wichtigkeit“ stattdessen verwendet. Dieser soll den Stellenwert jedes Systems im Unternehmen angeben.

3.3.7 Risk Determination

Um das Risiko zu ermitteln, muss die Wahrscheinlichkeits- und die Auswirkungsanalyse kombiniert werden. Bei der Wahrscheinlichkeitsanalyse wurde die β-TTC als Parameter für die Wahrscheinlichkeit festgelegt, bei der Auswirkungsanalyse die Wichtigkeit eines Systems.

3.3.8 Control Determination

Die Kontrollempfehlungen sollen den Verantwortlichen eine Hilfestellung geben, wie mit den gefunden Schwachstellen umzugehen ist. Die notwendigen Informationen sollen aus der gegebenen Advisory-Datenbank gewonnen werden. Dies kann zum Beispiel das Update auf eine höhere Version, das Deaktivieren einer Funktion oder der Wechsel auf ein anderes Softwareprodukt sein.

3.3.9 Results Documentation

Die Dokumentation der Risikobewertung soll in einem Webfrontend visualisiert werden.

3.4 Kriterien für eine priorisierte Rangfolge

Da diese Arbeit eine priorisierte Rangfolge zu Behebung der Schwachstellen liefern soll, müssen entsprechende Kriterien herausgearbeitet werden. Diese Kriterien gehen aus den ermittelten Informationen hervor. Zunächst werden Möglichkeiten aufgeführt, bevor am Ende die Auswahl erfolgt.

3.4.1 β-TTC

Ein Kriterium, das dabei hilft eine Rangfolge festzulegen, soll der ermittelte Wert aus der β-TTC Metrik sein. Dabei wird diese für jedes betroffene System ermittelt und alle Werte werden

miteinander verglichen. Das System, welches den niedrigsten Wert vorweist, soll dabei eine höhere Priorität bekommen, weil die Metrik einen Wert für die Dauer bis zu einer möglichen Kompromittierung angibt.

3.4.2 Standort der Systeme

Der Standort der betroffenen Systeme beeinflusst diese Rangfolge sehr stark. Hierbei stellt sich die Frage, ob sich ein System im Backend befindet, nur intern für einen gewissen Personenkreis verfügbar ist oder aus dem Internet erreichbar. Ein Sicherheitsbeauftragter oder Administrator kann anhand dieses Kriteriums abwägen, welches System bevorzugt behandelt werden sollte.

3.4.3 Wichtigkeit des Systems

Dass verschiedene Systeme in Unternehmen von unterschiedlicher Wichtigkeit sind, soll ebenfalls betrachtet werden. Dabei soll für jedes System ein Wert *hoch*, *mittel* oder *niedrig* einstellbar sein.

3.4.4 CVSS

Der CVSS-Score bietet mit seinem numerischen Werten von 0 bis 10 einen Parameter, durch den bestimmt werden kann, wie groß die Gefahr einer Schwachstelle ist. Bei der Betrachtung eines Systems ist die Schwachstelle mit dem höchsten Score, im Sinne eines “Worst Case“-Ansatzes, ausschlaggebend.

3.4.5 Anzahl der Schwachstellen

Die Anzahl der Schwachstellen, die ein System hat, muss auch berücksichtigt werden. Je mehr Schwachstellen in einem System existieren, um so größer ist die Wahrscheinlichkeit, dass eine dieser Schwachstellen ausgenutzt wird.

3.4.6 Fazit zu den Kriterien

Der Standort der Systeme ist ein wichtiger Faktor. Es ist aber schwierig mit einem Schwachstellenscanner durch ein komplexes Netz zu scannen, da hier verschiedene Firewalls Ports filtern können und so ein unvollständiges Bild entsteht. Deshalb wird dieser Parameter nicht berücksichtigt.
Um dem Benutzer aber ähnliche Einstellungsmöglichkeit bieten zu können, wird als ein Kriterium der Parameter Wichtigkeit festgelegt. Der Benutzer muss diese allerdings manuell

zuweisen.
Ein weiteres Kriterium soll der CVSS-Score sein. Wie oben erwähnt, soll ein System auf Grundlage des höchsten CVSS-Scores aller Schwachstellen bewertet werden können.
Die Anzahl der Schwachstellen, die ein System aufweist, wird ebenfalls als Kriterium betrachtet. Dabei bekommt das System mit einer größeren Anzahl von Schwachstellen eine höhere Priorisierung.
Das vierte und letzte Kriterium, welches in dieser Arbeit berücksichtigt wird, ist der β-TTC Wert.

4 Software Design

In diesem Kapitel werden die Softwarearchitektur der Anwendung und die einzelnen Komponenten und deren Aufgaben vorgestellt. Dabei wird zuerst die Architektur eines möglichen Produktivsystems und danach die des in dieser Arbeit zu entwickelnden Prototypen erläutert. Der Prototyp wird weniger Komponenten umfassen als das spätere Produktivsystem.

4.1 Architektur

In Abbildung 4.1 wird ein Überblick über die Systemarchitektur eines möglichen Produktivsystems mit seinen verschiedenen Komponenten gegeben.

Die Komponente *ScaniX* soll dabei verschiedene Möglichkeiten bieten, Informationen über Systeme mit deren Schwachstellen bereitzustellen. Die Daten von einem Schwachstellenscan soll die Subkomponente *VulniX* aufbereiten und bereitstellen. Die Subkomponente *eTTCs* wurde von Mantel (2018) im Rahmen seiner Bachelorarbeit entwickelt und liefert die Informationen über Schwachstellen auf Grundlage eines Asset-Managements. Um das System zu erweitern, können hier weitere Komponenten zur Datenbeschaffung implementiert werden. Dies drückt der Platzhalter *other* aus. Darüber hinaus sollen alle Komponenten aus *ScaniX* Zugriff auf die Advisory-Datenbank haben, um weitere Informationen abzufragen.
Die Komponente *DBiX* liefert die Schnittstelle, um die Daten aus der Advisory-Datenbank abzufragen. Die Daten sollen dabei über eine RESTful-API abrufbar sein. Damit wird eine einheitliche Schnittstelle bereitgestellt, wodurch bei der Entwicklung von Komponenten in *ScaniX* keine Datenbankabfragen formuliert werden müssen.
Zur Berechnung der β-TTC Werte soll die Komponente *MetriX* dienen. Diese soll durch weitere Metriken erweiterbar sein und ebenfalls eine RESTful-API bereitstellen. Um dem Benutzer die Möglichkeit für Einstellungen zu geben und um die Ergebnisse visualisieren zu können, sollen die Komponenten *DashiX*, *CliX*, *CroniX* und *ReportiX* dienen. *DiriX* soll als zentrale Einheit die Daten der *ScaniX* Komponenten über eine RESTful-API abfragen und diese für *DashiX*, *CliX*, *CroniX* und *ReportiX* ebenfalls über eine solche bereitstellen.

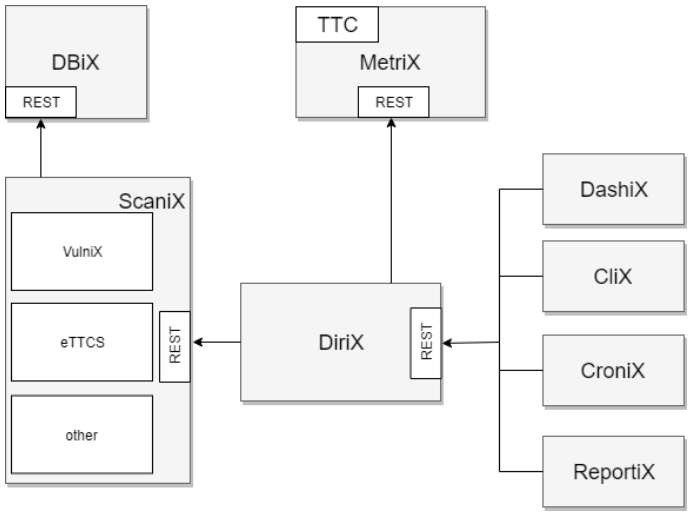

Abbildung 4.1: Systemarchitektur: Überblick ohne Schwachstellenscanner

4.2 Vorstellung der Komponenten

Die oben erläuterte Architektur zeigt ein Gesamtbild eines möglichen Produktivsystems. Da in dieser Arbeit nur ein Teil als Prototyp entwickelt werden soll, werden in diesem Abschnitt die umzusetzenden Komponenten vorgestellt.

4.2.1 DBiX

Die Advisory-Datenbank wird in diesem Prototyp in einer PostgreSQL-Datenbank gehalten. Es wird keine Komponente *DBiX* entwickelt, die die Daten über eine RESTful-API bereitstellt.

4.2.2 VulniX

Aus der Komponente *ScaniX* wird in diesem Prototyp die Subkomponente *Vulnix* umgesetzt. *Vulnix* soll dabei nicht automatisch Schwachstellenscans durchführen, sondern über eine RESTful-API einen Schwachstellenscanreport von OpenVAS im XML-Format erhalten. Das heißt, ein Schwachstellenscan muss manuell durchgeführt und ein Report generiert werden. Aus diesem Report sollen pro System die Schwachstellen in Form der CVE-IDs extrahiert und mit diesen die benötigten Informationen aus der Advisory-Datenbank abgefragt werden. Es werden keine weiteren Daten, außer der IP-Adresse und den CVE-IDs, aus dem Schwachstellenscanreport benutzt, um die Daten einheitlich zu halten. Die Antwort auf die Anfrage über die RESTful-API wird im JSON-Format gegeben.

4.2.3 DiriX/MetriX/DashiX

Des Weiteren werden in diesem Prototyp die Komponenten *DiriX, MetriX und DashiX* zusammen in einer Komponente entwickelt. Diese Komponente wird *PresentX* genannt. Die anderen Komponenten werden nicht entwickelt, da sie zum Erreichen des Ziels dieser Arbeit nicht benötigt werden. Auf einer bereitgestellten Weboberfläche soll dem Benutzer die Möglichkeit gegeben werden, einen Schwachstellenscanreport hochzuladen. Dieser wird über eine RESTful-API an die Komponente *VulniX* weitergeleitet. Die Antwort im JSON-Format soll in einer SQLite-Datenbank zwischengespeichert werden. Danach sollen dem Benutzer auf einer weiteren Weboberfläche Konfigurationsmöglichkeiten in Form verschiedener Kriterien zur Verfügung stehen, wie diese in Kapitel 3.4 erläutert worden sind. Hier soll der Benutzer die Möglichkeit haben, den einzelnen Systemen eine *Wichtigkeit* zuzuordnen. Darüber hinaus kann der Benutzer die Kriterien *Wichtigkeit*, *CVSS*, *Anzahl der Schwachstellen* und β*-TTC* für seine persönliche priorisierte Rangfolge auswählen. Nach Auswahl der Kriterien und der Bestätigung soll dem Benutzer eine ausführliche Liste aller Systeme und Schwachstellen in der gewählten Rangfolge präsentiert werden. Weiterhin soll dem Benutzer zur Veranschaulichung ein Dashboard bereitstehen, auf denen die einzelnen Kriterien pro System visualisiert sind.

4.3 Architektur des Prototypen

Mit den in 4.2 erläuterten Änderungen ergibt sich damit die folgende Systemarchitektur für den Prototyp.

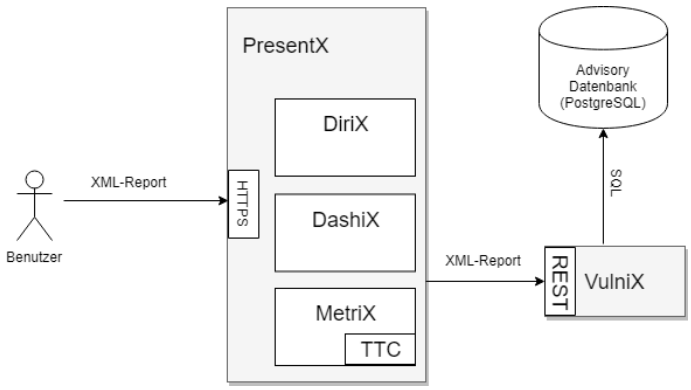

Abbildung 4.2: Architektur des Prototypen

5 Implementation

In Kapitel 4 wurde die Systemarchitektur des Prototypen vorgestellt. In diesem Kapitel wird nun auf die Implementierung der Komponenten eingegangen und es werden die verwendeten Werkzeuge vorgestellt.

5.1 Entwicklungsumgebung/Frameworks/Tools

5.1.1 PyCharm/Python

Zur Entwicklung dieses Prototypen wurde die Entwicklungsumgebung PyCharm eingesetzt. Diese wird von JetBrains entwickelt und basiert auf der Entwicklungsumgebung IntelliJ IDEA, welche für Java Projekte konzipiert ist. PyCharm wurde für die Programmiersprache Python optimiert (PyCharm, 2018). „Python ist eine universelle, höhere Programmiersprache, die üblicherweise interpretiert wird.“ (Steyer, 2018) Des Weiteren steht PyCharm für die Betriebssysteme Windows, macOS und Linux bereit. Dieser Prototyp wurde auf einem Ubuntu 16.04 entwickelt.

5.1.2 Django

Zur Entwicklung wurde das Python Web-Framework Django verwendet. Dieses hat einen großen Funktionsumfang und ermöglicht die einfache Erstellung von Web-Anwendungen. Django unterstützt dabei das Model-View-Controller (MVC)-Entwurfsmuster (Gamma u. a., 2011). In Django wurde MVC als Model-Template-View (MTV) (Abbildung 5.1) umgesetzt. Dabei ist das Model die Abstraktionsschicht für den Datenzugriff. Die dahinter liegende Datenquelle kann dabei variieren. Das Template ist für die Anzeige der Informationen zuständig und stellt die Schnittstelle zwischen Benutzer und Django-Anwendung dar. Die View ist die Logikkomponente und verarbeitet die Anwendungsdaten und ist für die Rückgabe der Antwort zuständig (George, 2017).

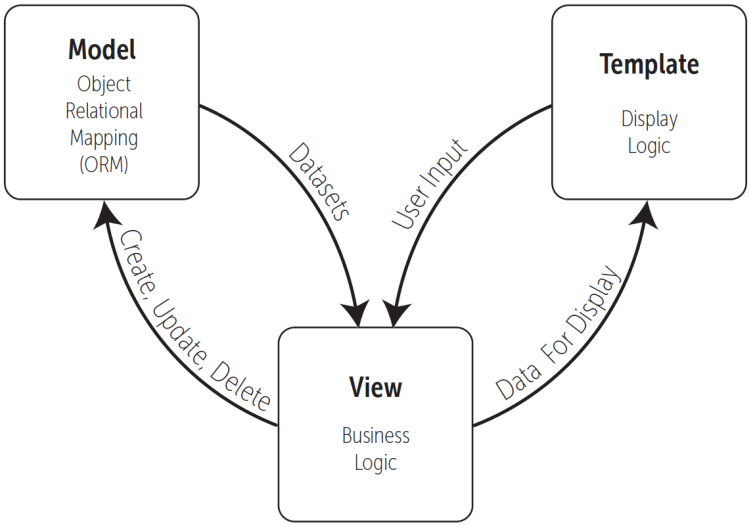

Abbildung 5.1: Model-Template-View (MTV)-Entwurfsmuster
Quelle: (George, 2017)

Darüber hinaus hat Django einen integrierten Entwicklungsserver, welcher die Möglichkeit bietet, die Applikation direkt zu verwenden, was sich bei der Entwicklung von Prototypen als besonders vorteilhaft darstellt.

5.1.3 SQLite

Zum Zwischenspeichern der Ergebnisse in *PresentX*, welche die Komponente *VulniX* als JSON-Datei liefert, wurde SQLite verwendet. SQLite ist ein eingebettetes Datenbanksystem, das heißt, die Datenbank ist direkt in der Anwendung enthalten (Owens, 2006). Sie besteht auch nur aus einer einzigen Datei, die alle Informationen wie Tabellen, Indizes, Trigger usw. enthält. Django erlaubt mit dem Eintrag *os.path.join(BASE_DIR, 'db.sqlite3')*, dass die SQLite Datei direkt im Projektverzeichnis gespeichert wird.

5.1.4 Object-relational mapping

Für die Interaktion mit der Advisory-Datenbank aus der Komponente *VulniX* wird ein Datenbankadapter benötigt. Da *VulniX* nur Daten aus einer Datenbank abfragt und diese im JSON-Format zurück gibt, wird ein einfach zu benutzender ORM (Object Relational Mapper) benötigt. Mit *dataset* steht so ein leichtgewichtiger ORM für Python bereit, mit dem die Daten abgefragt werden können (Dataset, 2018). Listing 5.1 zeigt ein Beispiel für die Abfrage des Werts für ein ID-Feld.

```
import dataset

# Verbindung zur Datenbank aufbauen
db = dataset.connect('postgresql://user:pw@localhost:5432/adv_db')

# Tabelle auswaehlen
table = db[\'authoring_vulnerability\']

# finde Eintrag
item = table.find_one(name='CVE-2018-1234')

# Eintrag durchgehen und den Wert zu passendem Schluessel ausgeben
for k, v in item.items():
    if(k == 'id')
        print(v)
```

Listing 5.1: Programmcode: Beispiel für die Benutzung von dataset

5.1.5 RESTful-API

Die Kommunikation zwischen den Komponenten soll über eine RESTful-API durchgeführt werden. Dafür wurde das *Django REST Framework* ausgewählt. Es ist ein leistungsfähiges und flexibles Toolkit zum Erstellen von Web-APIs. Es unterstützt die Serialisierung von ORM- und Nicht-ORM-Datenquellen und bringt einen großen Funktionsumfang mit. Gleichzeitig können aber auch einfache funktionsbasierte Ansichten verwendet werden, wenn leistungsfähigere Funktionen nicht benötigt werden (Django REST Framework, 2018). Listing 5.2 zeigt eine klassenbasierte Ansicht, die *"Hello World"* zurückgibt.

```
import rest_framework.views import APIView
import rest_framework.views import Response

class VulniX(APIView)
    def get(self, request):
        return Response("Hello World")
```

Listing 5.2: Programmcode: Beispiel „Hello World“ Django REST Framework

5.2 Umsetzung

5.2.1 DBiX

Wie im Konzept erläutert, wird eine PostgreSQL Datenbank in der Version 9.3 auf dem Entwicklungssystem installiert und eine Instanz für die Advisory-Datenbank angelegt. In diese wird der zur Verfügung stehende Dump der Datenbank eingespielt.

5.2.2 VulniX

Vulnix wurde als Django-Projekt in einer eigenständigen Komponente entwickelt. Sie bietet eine RESTful-API, welche mit dem *Django REST Framework* umgesetzt wurde. Über diese nimmt *VulniX* den Schwachstellenscanreport im XML-Format entgegen. Danach wird der Report, wie in Kapitel 3.2 erläutert, nach den Host-IP-Adressen durchsucht und die Schwachstellen werden für den jeweiligen Host identifiziert. Im nächsten Schritt werden alle CVE-IDs aus dem XML-Report extrahiert. Bei einer fehlerhaften oder manipulierten Datei wird an dieser Stelle abgebrochen. Entspricht die Datei dem richtigen Format, werden die extrahierten Daten in Objekten gehalten und nicht in einer Datenbank zwischengespeichert. Nachdem die XML-Datei durchsucht und alle Host-IP-Adressen und die dazugehörigen Schwachstellen extrahiert worden sind, werden für jede einzelne Schwachstelle mit Hilfe von *dataset* die benötigten Informationen aus der Advisory-Datenbank abgefragt. An dieser Stelle hätte auch der von Django bereitgestellte ORM verwendet werden können, da aber zu diesem Zeitpunkt noch nicht angedacht war, *VulniX* als Django-Anwendung zu entwickeln, wurde dies mit *dataset* umgesetzt. Die benötigten Informationen, die aus der Advisory-Datenbank abgefragt werden, sind der CVSS-Score, der Titel, die Beschreibung und die zugeordneten CPEs. Im letzten Schritt werden die Objekte mittels *json.dumps* in das JSON-Format konvertiert und als Antwort zurückgegeben. Listing 5.3 zeigt die Rückgabe einer Liste mit Host-Objekten.

```
1 return Response(json.dumps({"Hosts":hosts}, indent=1)
```

Listing 5.3: Programmcode: Beispiel Response RESTful-API mit *json.dumps*

5.2.3 PresentX

DiriX, MetriX und DashiX werden, wie im Konzept vorgestellt, zusammen in einem Django-Projekt umgesetzt. Dabei wird das Model-Template-View-Entwurfsmuster verwendet. Der Einstiegspunkt der Anwendung ist ein Dialog, bei dem der Benutzer aufgefordert wird, einen

Schwachstellenscanreport im XML-Format an die Anwendung zu übergeben. Wenn der Benutzer einen Report ausgewählt hat, kann er mit einem „Upload“-Button die Datei an das System übergeben. Eine Prüfung untersucht dabei, ob es sich um eine XML-Datei handelt. Falls dies nicht der Fall ist, wird dies dem Benutzer in Form einer Warnmeldung mitgeteilt und er hat erneut die Möglichkeit eine Datei auszuwählen. Fällt diese Prüfung positiv aus, wird der Report über die RESTful-API an *VulniX* weitergeleitet und die Antwort im JSON-Format entgegengenommen.

Abbildung 5.2: Benutzeroberfläche: Dateieingabe Dialog

Der Inhalt der JSON-Antwort wird dann in der SQLite-Datenbank zwischengespeichert. Dafür wurden *Models* (Listing 5.4) angelegt, die die Datenstruktur widerspiegeln und einen einfachen Zugriff auf die Daten ermöglichen. Zusätzlich wird pro System die β-TTC-Metrik berechnet und auch diese persistiert.

```
1  class Hosts(models.Model):
2      name = models.CharField(max_length=500)
3      count = models.IntegerField()
4      ttc = models.IntegerField()
5      prio = models.IntegerField()
6      cvss = models.FloatField()
7
8  class CVEs(models.Model):
9      host = models.ForeignKey(Hosts, on_delete=models.CASCADE)
10     name = models.CharField(max_length=500)
11     description = models.CharField(max_length=5000)
12     title = models.CharField(max_length=500)
13     cpes = models.TextField()
14
15 class CVSS(models.Model):
```

```
16      cve = models.OneToOneField(CVEs, on_delete=models.CASCADE,
            ↪ related_name='cvss')
17      basescore = models.FloatField()
18      temporalscore = models.FloatField()
19      cvssVektor = models.CharField(max_length=500)
```

Listing 5.4: Programmcode: PresentX Models

Im nächsten Schritt werden dem Benutzer die gefundenen Systeme präsentiert, die in der *SQLite-Datenbank* gespeichert worden sind. Um dies anzuzeigen, wird ein *Template* benutzt, welches mit den Daten, die die *View* liefert, gefüllt wird. Der Benutzer kann hier jedem System, wie in Kapitel 4.2.3 erklärt, eine Wichtigkeit aus den Kategorien *hoch*, *mittel* und *niedrig* zuweisen. Zusätzlich kann der Benutzer seine persönliche Prioritätenrangfolge für die Kriterien Wichtigkeit, β-TTC, CVSS und Anzahl Schwachstellen festlegen. Danach wird die Eingabe bestätigt.

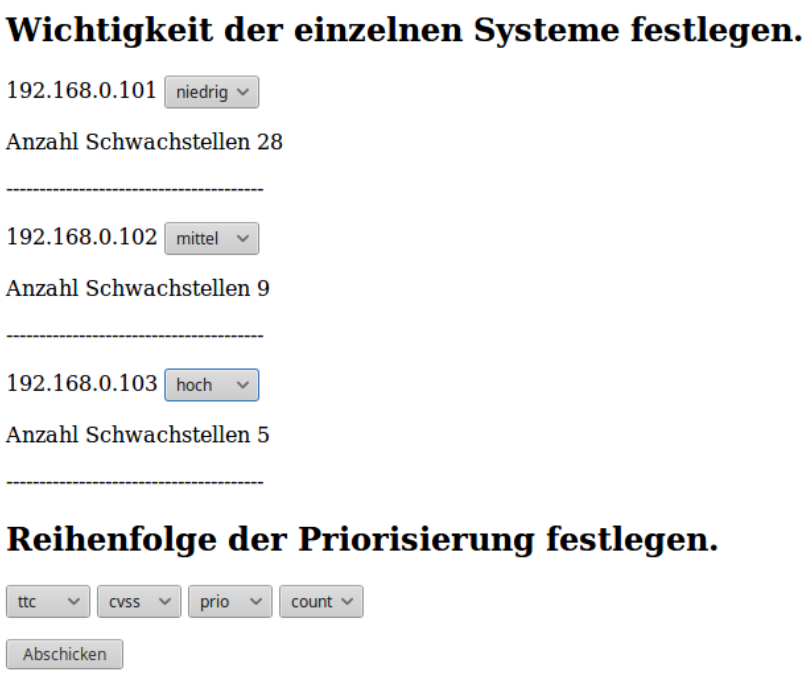

Abbildung 5.3: Benutzeroberfläche: Wichtigkeit setzen, Rangfolge festlegen

Die Ergebnisse werden dem Benutzer in einer textuellen Ansicht präsentiert, wobei die Systeme nach der Rangfolge seiner eingestellten Kriterien sortiert sind. Die Systeme sind dabei von

1 bis n durchnummeriert. Darüber hinaus werden die Kriterien und Informationen zu jeder Schwachstelle angezeigt.

Abbildung 5.4: Benutzeroberfläche: textuelle Auswertung

Zur besseren Visualisierung steht zusätzlich ein Dashboard bereit, über das die Kriterien einzeln angezeigt werden. Dadurch bekommt der Benutzer eine bessere Sicht auf die unterschiedlichen Parameter. Der Parameter „Wichtigkeit“ ist dabei in der Beschriftung hinter der jeweiligen IP-Adresse aufgeführt. Die Grafiken in diesem Dashboard zeigen die β-TTC-Metrik, den höchsten CVSS-Score für die jeweiligen Systeme sowie alle Systeme mit ihren Schwachstellen und den zugehörigen CVSS-Scores. Dabei sind diese jeweils nach der Kritikalität sortiert. Die Gestaltung ist dabei nach den Ampelfarben vorgenommen worden, da diese jedem bekannt sind. Rot steht dabei für einen kritischen, gelb für einen geringfügig kritischen und grün für einen nicht besonders kritischen Wert. Die Intervalle wurden dabei in drei Bereiche aufgeteilt und die Farbgestaltung darauf angewendet. Dafür wurden immer der höchste und der niedrigste Wert

zur Berechnung herangezogen. Diese Art der Visualisierung bietet auf den ersten Blick eine intuitive Grundlage, kann aber noch optimiert werden.

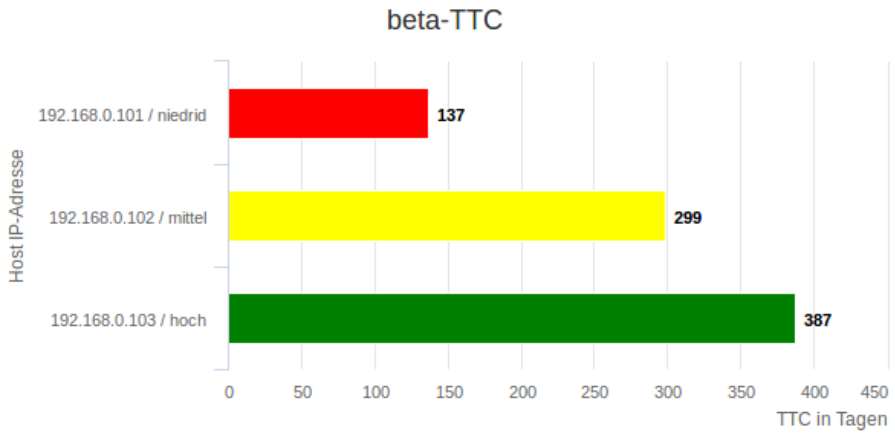

Abbildung 5.5: Benutzeroberfläche: Chart Auswertung β-TTC

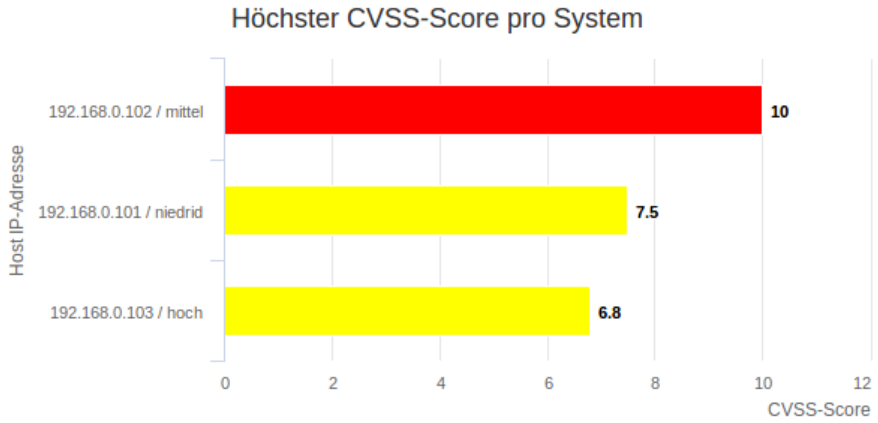

Abbildung 5.6: Benutzeroberfläche: Chart Auswertung CVSS

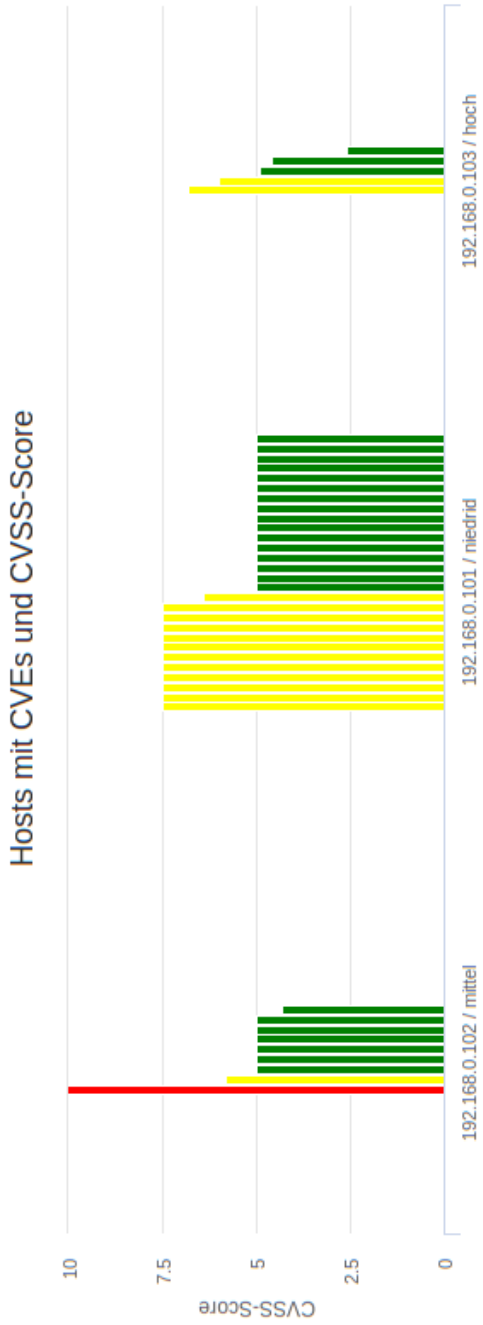

Abbildung 5.7: Benutzeroberfläche: Chart Auswertung Hosts mit CVEs (Balken) und CVSS (Höhe der Balken)

6 Fazit und Ausblick

Das Ziel dieser Arbeit war es, einen Schwachstellenscanreport durch die Einführung der β-Time-To-Compromise Metrik genauer und dem Risiko angemessener zu priorisieren. Dazu wurde als erstes der Report des Schwachstellenscanners OpenVAS analysiert. Des Weiteren wurde ein Risikomanagementprozess begleitet und Kriterien für eine priorisierte Rangfolge wurden herausgearbeitet. Mit Hilfe eines Prototypen wurde unter Berücksichtigung der Kriterien eine priorisierte Rangfolge bestimmt und der Unterschied zwischen CVSS und β-TCC in einem Dashboard dargestellt.

6.1 Fazit

Bei der Analyse des PDF-Report des Schwachstellenscanners OpenVAS wurde festgestellt, dass eine Rangfolge nur auf Grundlage des CVSS-Score erfolgt. Idealerweise sollten hier aber mehrere Möglichkeiten für die Nutzung bereitstehen.
Diese Arbeit hat gezeigt, dass eine Rangfolge immer von den individuellen Kriterien abhängt, da die Festlegung auf nur ein Kriterium die Sichtweise extrem einschränken kann. Die β-Time-To-Compromise benutzt als Eingangsparameter nur die Anzahl der Schwachstellen und nicht den Schweregrad bzw. Einfluss, die diese auf das System haben. Deshalb muss diese Metrik kritisch betrachtet werden. Ein System mit vielen Schwachstellen besitzt einen niedrigen TTC-Wert (Tage, bis das System kompromittiert wird). Erfüllen diese Schwachstellen aber nicht alle als relevant bewerteten Eigenschaften, ist die Schwachstellenzahl eventuell auch nicht so kritisch für ein System. Daher kann diese Metrik nur im Zusammenspiel mit dem CVSS-Score eine plausible Einschätzung darüber abgeben, wie gefährdet ein System tatsächlich ist.
Wie in Kapitel 2.3.1 erläutert, besitzt CVSS Parameter für die räumliche Entfernung (AV) und das benötigte Authentifizierungslevel (AU) eines Angreifers. Über diese Parameter können für die verschiedenen Systeme die Schwachstellen herausgefiltert werden, welche als relevant angesehen werden. Das heißt, es wird nur eine Teilmenge der Schwachstellen eines Systems betrachtet und auf dieser Grundlage der β-TTC neu berechnet. Dadurch wird ein Wert generiert, der sich genau auf die Schwachstellen dieses Systems bezieht, welche die Angreifbarkeit gemäß

AV und AU beschreiben. Da dies für alle Systeme individuell durchführbar ist, kann die β-TTC so systemübergreifend verglichen und eine exaktere Rangfolge geliefert werden. Dieses Beispiel zeigt, dass die β-TTC Metrik von der Berücksichtigung der CVSS-Einzelwerte profitiert und dadurch eine genauere Priorisierung erreicht wird.
Das Dashboard bietet dem Benutzer dabei eine Übersicht über diese zwei sehr unterschiedlichen Metriken. Die Ansicht der einzelnen Systeme mit ihren jeweiligen Schwachstellen schafft darüber hinaus einen Eindruck über die Gesamtlage. Auf dieser Grundlage kann ein Benutzer seine eingestellte Rangfolge überdenken. Des Weiteren sind die in Form der Kriterien geschaffenen Konfigurationsmöglichkeiten einfach zu verstehen, leicht zu bedienen und bieten dem Benutzer die Flexibilität, diese anzupassen, um die Rangfolge unter verschiedenen Gesichtspunkten zu vergleichen.

6.2 Ausblick

Um das System produktiv einsetzen zu können, sollten noch folgende Erweiterungen in Betracht gezogen werden:

1. *Asset-Management*
   Zur Erweiterung des Systems könnte ein umfassendes Asset-Management mit zusätzlicher zyklischer Kontrolle durch einen Schwachstellenscan eingeführt werden. Dadurch können weitere Schwachstellen entdeckt werden, die nicht durch die Prüfmuster des Scans erkannt worden sind. Zusätzlich können so Schwachstellen in einem System bestimmt werden, die eventuell durch eine Firewall unerkannt bleiben. Ein Problem, das hierbei auftritt, ist, dass einige Hersteller von Betriebssystemen Backport-Sicherheitsupdates für betroffene Software einspielen und sich so ein Abgleich über versionsbehaftete CPEs schwierig gestalten lässt. Backport-Sicherheitsupdates sind Updates auf eine stabile, aber nicht mehr aktuelle Version eines Paketes oder einer Software. Dabei werden die Sicherheitsupdates in diese ältere Version eingespielt, ohne eine Update auf eine höhere Version durchzuführen. Es müssten Konfigurationsmöglichkeiten geschaffen werden, um hier nicht bei jedem Scan alte und schon behobene Schwachstellen in die Anwendung importiert zu bekommen.

2. *Bestimmung des Betriebssystems*
   Zur Ergänzung von Schwachstellen, die durch den Schwachstellenscanner aufgrund der Sichtbarkeit von Schwachstellen nicht gefunden wurden, könnte ein Benutzer sein Betriebssystem aus einer Liste auswählen. Solche Schwachstellen könnten zum Beispiel

nur lokal ausnutzbare sein. Der Benutzer müsste dann noch ein Datum des letzten Sicherheitsupdates angegeben, um nur die Schwachstellen zu erhalten, die noch nicht in seinem System behoben worden sind. Dies ist nötig, da der Kernel von vielen Herstellern auf eigene Patchlevel geupdatet wird und dies nicht über CPEs unterscheidbar ist.

3. *Typisierte TTC*
   In Kapitel 2.3.4 wurden typisierte TTCs vorgestellt. Diese Metriken sollten weiter untersucht werden, da hierbei die Schwachstellen nach ihren Auswirkungen eingeteilt und darüber bewertet werden. Sie könnten die Rangfolge weiter konkretisieren, wenn für die verschiedenen Schutzziele individuelle Prioritäten angegeben werden.

4. *CVSS-Score-Grenzen festlegen*
   Schwachstellen, die unter einem gewissen CVSS-Score liegen, können aus der Wertung genommen werden. Damit würde sich der β-TTC-Wert erhöhen und es ergäbe sich eventuell eine genauere Einschätzung, da die Schwachstellen mit einem geringen CVSS-Score meist keine kritischen Auswirkungen haben.

5. *Advisories als Patches aus der ADV-Datenbank bereitstellen*
   Für Administratoren, die Schwachstellen in einem System beheben wollen, ist es von Vorteil, wenn sie Patches nicht manuell suchen müssten. Deshalb wäre eine Auflistung der verfügbaren Advisories als Patches sinnvoll. Dabei sollte gezeigt werden, welche Schwachstellen die verschiedenen Advisories beheben. Durch die Anwendung des entsprechenden Advisories würden die Schwachstellen als „gepachted“ aus der Wertung genommen werden und der Administrator müsste nur noch den Anweisungen aus dem Advisory folgen, um die Schwachstellen zu beheben.

Literaturverzeichnis

[Buttner und Ziring 2009] Buttner, Andrew ; Ziring, Neal: *Common Platform Enumeration CPE Specification*. http://cpe.mitre.org/files/cpe-specification_2.2.pdf. 2009

[Casper und Strobel 2018] Casper, Mirko ; Strobel, Stefan: Schwachstellen finden und beheben. In: *iX* (2018), November, S. 44 – 52

[Dataset 2018] Dataset: *Dokumentation dataset*. 2018. – URL https://dataset.readthedocs.io/en/latest/api.html

[Django REST Framework 2018] Django REST Framework: *Dokumentation Django REST framework*. 2018. – URL https://www.django-rest-framework.org/

[Foreman 2009] Foreman, Park: *Vulnerability Management*. CRC Press, New York, 1. Auflage, 2009

[Gamma u. a. 2011] Gamma, Erich ; Johnson, Ralph ; Helm, Richard ; Vlissides, John: *Entwurfsmuster: Elemente wiederverwendbarer objektorientierter Software*. Pearson Deutschland GmbH, 2011. – ISBN 978-3-8273-3043-7

[George 2017] George, Nigel: *Build Your First Website with Python and Django*. GNW Independent Publishing 1. Auflage, 2017. – ISBN 978-0-9946168-4-5

[Lyon 2009] Lyon, Gordon F.: *Nmap Network Scanning: The Official Nmap Project Guide to Network Discovery and Security Scanning*. USA : Insecure, 2009. – ISBN 0979958717, 9780979958717

[Mantel 2018] Mantel, Alex: *Risk Management System Prototype based on an Extended Time-To-Compromise (TTC) Metric*, Hochschule für Angewandte Wissenschaften Hamburg, Bachelorarbeit, 2018

[Maynor 2011] Maynor, David: *Metasploit toolkit for penetration testing, exploit development, and vulnerability research*. Elsevier, 2011

[McQueen u. a. 2006] McQueen, Miles A. ; Boyer, Wayne F. ; Flynn, Mark A. ; Beitel, George A.: Time-to-Compromise Model for Cyber Risk Reduction Estimation. In: Gollmann, Dieter H. (Hrsg.) ; Massacci, Fabio H. (Hrsg.) ; Yautsiukhin, Artsiom H. (Hrsg.): *Quality of Protection: Security Measurements and Metrics*. Boston, MA : Springer US, 2006, S. 49 – 64. – URL http://dx.doi.org/10.1007/978-0-387-36584-8_5. – ISBN 978-0-387-36584-8

[Mell u. a. 2006] Mell, Peter ; Scarfone, Karen ; Romanosky, Sasha: CVSS: A complete Guide to the Common Vulnerability Scoring System Version 2.0. URL https://www.first.org/cvss/cvss-v2-guide.pdf, 2006

[MITRE1 2018] MITRE1: *CVE Numbering Authorities*. https://cve.mitre.org/cve/cna.html. 2018. – [Online; abgerufen 08.12.2018]

[MITRE2 2018] MITRE2: *Frequently Asked Questions*. https://cve.mitre.org/about/faqs.html. 2018. – [Online; abgerufen 29.09.2018]

[Nessus 2018] Nessus: *Dokumentation Nessus*. 2018. – URL https://docs.tenable.com/nessus/Content/GettingStarted.htm

[OpenVAS 2018] OpenVAS: *Dokumentation OpenVAS*. 2018. – URL http://www.openvas.org/software.html

[Owens 2006] Owens, Michael: *The Definitive Guide to SQLite*. Berkeley, CA : Apress, 2006. – URL https://doi.org/10.1007/978-1-4302-0172-4_1. – ISBN 978-1-4302-0172-4

[PyCharm 2018] PyCharm: *Dokumentation PyCharm*. 2018. – URL https://www.jetbrains.com/pycharm/documentation/

[Sanguino und Uetz 2017] Sanguino, Luis Alberto B. ; Uetz, Rafael: Software Vulnerability Analysis Using CPE and CVE. In: *CoRR* abs/1705.05347 (2017). – URL http://arxiv.org/abs/1705.05347

[Steinke 2015] Steinke, Michael: *Schwachstellenmanagement in Hochschulnetzen am Beispiel des Münchner Wissenschaftsnetzes*, Ludwig-Maximilians-Universität München, Masterarbeit, 2015

[Steyer 2018] Steyer, Ralph: *Programmierung in Python Ein kompakter Einstieg für die Praxis*. Springer Vieweg, Wiesbaden, 2018. – URL http://dx.doi.org/10.1007/978-3-658-20705-2. – ISBN 978-3-658-20705-2

[Stoneburner u. a. 2002] Stoneburner, Gary ; Goguen, Alice Y. ; Feringa, Alexis: SP 800-30. Risk Management Guide for Information Technology Systems. Gaithersburg, MD, United States : National Institute of Standards & Technology, 2002. – Forschungsbericht

[Wang und Yang 2017] Wang, Y. ; Yang, J.: Ethical Hacking and Network Defense: Choose Your Best Network Vulnerability Scanning Tool. In: *2017 31st International Conference on Advanced Information Networking and Applications Workshops (WAINA)*, March 2017, S. 110–113

[Zieger u. a. 2018] Zieger, Andrej ; Klaus-Peter, Kossakowski ; Felix, Freiling: The β-Time-to-Compromise Metric for Practical Cyber Security Risk Estimation. In: *11th Internaional Conference On It Security Incident Management And It Forensics*. IEEE, 2018. – ISBN 978-1-5386-6632-6

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 18.12.2018 Sascha Reuter