HAW HAMBURG

MASTERTHESIS
Arne Thiele

Planung und Umsetzung einer Alarmkorrelation für ein Security Information and Event Management System

FAKULTÄT TECHNIK UND INFORMATIK
Department Informations- und Elektrotechnik

Faculty of Computer Science and Engineering
Department of Information and Electrical Engineering

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN HAMBURG
Hamburg University of Applied Sciences

Arne Thiele

Planung und Umsetzung einer Alarmkorrelation für ein Security Information and Event Management System

Masterarbeit eingereicht im Rahmen der Masterprüfung
im Studiengang *Master of Science Automatisierung*
am Department Informations- und Elektrotechnik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Klaus Peter Kossakowski
Zweitgutachter: Prof. Dr. Olaf Zukunft

Eingereicht am: 18. August 2022

Arne Thiele

Thema der Arbeit

Planung und Umsetzung einer Alarmkorrelation für ein Security Information and Event Management System

Stichworte

SIEM, SOC, Alarmkorrelation, IT-Security

Kurzzusammenfassung

Diese Arbeit leitet eine skalierbare Architektur einer Alarmkorrelation zum Einsatz in einem SIEM-System her und implementiert diese prototypisch. Das entwickelte System beinhaltet Möglichkeiten zur flexiblen Konfiguration, um für verschiedene Einsatzgebiete nutzbar zu sein.

Arne Thiele

Title of Thesis

Planning and implementation of a security information and event management system considering alert-correlation aspects

Keywords

SIEM, SOC, Alertcorrelation, IT-Security

Abstract

This thesis introduces a scalable architecture of an alert correlation. The resulting prototype can be configured and thus be used flexibly, depending on the requirements of the operating environment.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungen

A/s	Alarme pro Sekunde
APT	Advanced Persistent Threat
BSI	Bundesamt für Sicherheit in der Informationstechnik
CSIRT	Computer Security Incident Response Team
DDoS	Distributed Denial of Service
FP	False Positive
HAW	Hochschule für Angewandte Wissenschaften
HIDS	Host-based Intrusion Detection System
IDS	Intrusion Detection System
IPS	Intrusion Prevention System
ML	Machine Learning
NIDS	Network-based Intrusion Detection System
SIEM	Security Information and Event Management
SOC	Security Operations Center

Listings

1 Einleitung

Durch die steigende Vernetzung von Informationssystemen und die voranschreitende Digitalisierung wird der Bedarf geeigneter Sicherheitsmechanismen im digitalen Raum größer. Die Anzahl der durchgeführten Cyber-Angriffe ist in den letzten Jahren gestiegen. Dabei wurde festgestellt dass immer häufiger zielgerichtetere Angriffe durchgeführt werden, welche durch den steigenden Grad der Automatisierung auf diesem Gebiet unterstützt werden.[1][2][3]
Auf der anderen Seite werden auch die Verteidigungs- und Detektionsmechanismen innerhalb von Netzwerken kontinuierlich weiterentwickelt. Dies führt zu einer starken Steigerung der Anzahl von erzeugten Alarmen, welche von Security Information and Event Management (SIEM)-Systemen gebündelt verarbeitet, korreliert und aggregiert werden. Das Ziel ist hierbei die Reduzierung von Alarmmengen, sowie die Informationsgewinnung über stattfindende oder bereits stattgefundene Angriffe.[4][5]
In dieser Arbeit wird die Architektur einer Alarmkorrelation für die Verwendung in einem SIEM konzipiert und prototypisch umgesetzt. Dabei steht die Flexibilität des Systems und die Skalierbarkeit der Architektur im Vordergrund.

1.1 Problemstellung

Durch die Automatisierung von Cyber-Angriffen und eine wachsende Verbreitung von Intrusion Detection System (IDS)-Infrastrukturen, steigt die Anzahl von ausgelösten Alarmen von Sicherheitsmechanismen. Diese werden häufig in ein SIEM-System eingespielt, welches wiederum von einem Security Operations Center (SOC) genutzt und überwacht wird. Die so verarbeiteten Alarme werden von menschlichen Analysten untersucht. Potentiell muss jeder eingehende Alarm betrachtet werden. Dies führt zu einer hohen Auslastung der Analysten.[1][5]
Teilweise werden Alarme bereits von der im SIEM-System enthaltenen Alarmkorrelation zusammengefasst und priorisiert. Allerdings ist die Genauigkeit dieses Mechanismus nicht

immer ausreichend. Zusätzlich liefert die IDS-Infrastruktur auch Fehlalarme, welche wiederum den Aufwand für die Analysten steigern. Häufig liefern verschiedene IDS-Systeme auch unterschiedlich strukturierte Daten, sodass diese normalisiert werden müssen, damit der zugrundeliegende Korrelationsmechanismus greifen kann.[5]
Die hohe Datenlast des SIEM-Systems, sowie der Anspruch eine angemessene Reaktionszeit auf tatsächliche Angriffe einzuhalten, macht die Konzeption eines SIEM-Systems zu einer Herausforderung in Bezug auf Performanz und Skalierbarkeit des Systems.

1.2 Zielsetzung

Das Ziel dieser Arbeit ist die Planung und die prototypische Umsetzung einer Korrelationskette für den Einsatz in einem SIEM-System. Die zugrundeliegende Alarmkorrelation soll eine möglichst starke Zusammenfassung von eingehenden Alarmen realisieren, um Analysten zu entlasten.
Das umgesetzte System soll um weitere IDS-Alarmformate erweiterbar sein. Außerdem soll die Konfigurierbarkeit und Parametrisierung des Systems gewährleistet werden, um individuelle Anpassungen zu ermöglichen.
Da das SIEM-System potentiell stark ausgelastet wird, soll bei der Konzeption die Skalierbarkeit des Systems im Vordergrund stehen. Für eine flexible Einsetzbarkeit soll das System Aspekte aus drei verschiedenen Anwendungsbereichen berücksichtigen. Dazu soll ein Frühwarnsystem entworfen werden, welches bei breitgefächerten Angriffen wie Distributed Denial of Service (DDoS)-Attacken greift und möglichst früh im Korrelationsprozess eine Alarmierung vornimmt. Des Weiteren soll eine grafische Aufbereitung der Ergebnisse unterstützt werden. Zusätzlich soll ermöglicht werden, das System für forensische Untersuchungen einzusetzen, um bereits vergangene Angriffe zu analysieren und Informationen über den Hergang zu gewinnen.

1.3 Abgrenzung

Die in dieser Arbeit beschriebene Alarmkorrelation orientiert sich an der von Valeur et al. [4] geschilderten Korrelationskette und setzt auf vorangegangenen Ausarbeitungen auf. In den Projektarbeiten wurden bereits vier Basis-Services umgesetzt. Daran anschließend wurde die Möglichkeit untersucht Testdaten in das System einzuspielen. In dieser Arbeit werden die bereits untersuchten Bestandteile der Alarmkorrelation erweitert und weitere

Services hinzugefügt.[6][7][8][9][10] Dabei werden keine Vergleiche von Algorithmen innerhalb der einzelnen Services vorgenommen und es wird auf den Einsatz von Machine Learning (ML) verzichtet. Ebenfalls wird davon abgesehen Mechanismen zu implementieren, welche die False Positive (FP)-Rate verringern, da dies als eigener Bereich der Alarmkorrelation betrachtet werden kann und über den Umfang dieser Arbeit hinausgehen würde.

1.4 Struktur der Arbeit

Diese Arbeit ist im Weiteren wie folgt strukturiert. In Kapitel 2 werden verwandte Arbeiten betrachtet und grundlegende Aspekte zur Alarmkorrelation erläutert. Darauf aufbauend werden in Kapitel 3 die Ziele der Verarbeitungskette erläutert und daraus Anforderungen an den Prototypen hergeleitet. Kapitel 4 beinhaltet die Architektur des Prototypen und erläutert die Funktionen der entwickelten Services. In Kapitel 5 werden Messungen anhand des entwickelten Prototypen dargestellt und durchgeführt. Daran anschließend werden die Ergebnisse der Messungen in Kapitel 6 ausgewertet und interpretiert. Abschließend werden in Kapitel 7 die Ergebnisse dieser Arbeit zusammengefasst, ein Fazit gezogen und ein Ausblick auf zukünftige Arbeiten gegeben.

2 Verwandte Arbeiten

In diesem Kapitel werden verwandte Arbeiten und grundlegende Konzepte betrachtet. Dabei steht die Verwendung eines SIEM-Systems innerhalb eines SOC im Vordergrund. Anschließend werden verschiedene Korrelationstechniken erläutert.

2.1 SOC

Ein SOC setzt sich aus Sicherheitsanalysten zusammen, deren Aufgabe es ist, Sicherheitsvorfälle (engl. *Security Incidents*) innerhalb der überwachten Infrastruktur zu entdecken, zu analysieren, auf diese zu reagieren, Berichte zu verfassen und zukünftig weiteren Vorfällen vorzubeugen. Dabei überwacht ein SOC die für die IT-Security relevanten Ereignisse einer Organisation, einer Firma oder eines Netzwerks. Die Sicherheitsanalysten überprüfen eingehende Alarme/Events und treffen eine Entscheidung über den Umgang mit diesem spezifischen Alarm. Dazu ist es notwendig, möglichst viele Informationen detailliert und übersichtlich darzustellen, damit manuelle Aufwände zur Bestimmung der Art der Bedrohung und dem weiteren Umgang mit dieser reduziert werden können. Diese Herausforderung kann großteils von geeignetem Tooling wie bspw. SIEM Systemen übernommen werden (siehe Abschnitt 2.1.4), allerdings müssen die Analysten des SOC dennoch filtern, welche Alarme Fehlalarme und welche Alarme tatsächliche Bedrohungen sind. [11][12][13]
Um diese Aufgabe besser zu unterstützen werden teilweise ML Algorithmen eingesetzt, um Handlungsempfehlungen für den Umgang mit bestimmten Alarmen auszusprechen. Laut Alahmadi et al. [11] misstrauen allerdings viele Analysten diesem Ansatz. Der Artikel endet mit einem Aufruf die FPs zu reduzieren und Alarme besser verständlich zu gestalten, damit ein Vertrauensverhältnis zwischen Analysten und Tooling entstehen kann.

2.1.1 Aufbau

Der Aufbau eines SOC basiert auf der Organisation und Anordnung der Analysten um den zugrunde liegenden Korrelationsprozess, welcher bspw. von einem SIEM-System unterstützt werden kann. Zimmermann [12] benennt fünf Kategorien in die man SOCs aufteilen kann.

1. **Security Team**: Es handelt sich um ein kleines Netzwerk mit bis zu 1.000 Nutzern bzw. IP-Adressen. Es existieren keine ausgereiften Prozesse zur Reaktion auf Sicherheitsvorfälle. Reaktive Maßnahmen werden ad hoc von einem Sicherheitsteam eingeleitet. Die Ergebnisse dieses Aufbaus variieren stark, da sie abhängig von der Expertise des agierenden Teams sind.

2. **Internes verteiltes SOC**: Es wurde bereits ein dediziertes SOC-Team aufgestellt, allerdings besteht dieses aus Personen, welche zusätzlich noch andere Aufgaben in der Organisation übernehmen. Zwar werden reaktive und proaktive Maßnahmen eingeleitet und koordiniert, allerdings werden die damit verbundenen Aufgaben von extern hinzugezogenen Analysten übernommen. Viele kleine bis mittelgroße Netzwerke mit mit 500 bis 5.000 Nutzern bzw. IP-Adressen.

3. **Internes zentralisiertes SOC**: Ein dediziertes Team wird aus erfahrenen Analysten und IT-Security Experten gebildet, welche ein Netzwerk mit 5.000 bis 100.000 Nutzern bzw. IP-Adressen betreuen. Dabei sind grundlegende Prozesse zur täglichen Verteidigung des Netzwerks definiert und die notwendigen Befugnisse, um sicherheitsrelevante Maßnahmen einzuleiten, sind erteilt und legitimiert. SOCs dieser Kategorie haben in der Regel ihr eigenes Budget und sind losgelöst von den Aufgabenbereichen innerhalb des Netzwerks. Das so gebildete Team unterliegt der Verantwortung eines SOC-Managers, welcher die Ziele des Teams gegenüber der netzwerknutzenden Institution, Organisation oder Firma in regelmäßigen Abständen überprüft.

4. **Kombination aus internem zentralisierten und internem dezentralisierten SOC**: Zusätzlich zu einem internen zentralisiertem SOC werden weitere Kapazitäten, wie beschrieben in Punkt 2, hinzugezogen. Dabei wird die Koordination von dem zentralisierten SOC übernommen, was dazu beiträgt eine zentralisierte Wissensbasis beizubehalten. Allerdings wird es gerade bei einer verteilt genutzten Infrastruktur notwendig, weitere Personen einzubeziehen, welche nicht dediziert dem SOC-Team zugeordnet sind. Diese Maßnahme ermöglicht die Wartung und

Überwachung von Enklaven-Systemen und sorgt für eine Härtung des Gesamtnetzwerks durch Verteilung des Wissens an bspw. Software-Entwickler. SOCs dieser Kategorie betreuen in der Regel Netzwerke mit 25.000 bis 500.000 Nutzern bzw. IP-Adressen.

5. **Koordinierendes SOC**: Ein übergeordnetes SOC übernimmt die Koordination von Verteidigungsmaßnahmen von mehreren untergeordneten SOCs. Dabei liegen die Befugnisse und Legitimationen in der Umsetzung der Maßnahmen weiterhin bei den untergeordneten SOCs. Das koordinierende SOC übernimmt Schulungen, bietet Consulting Services an die Netzwerk nutzende Institution an oder koordiniert den Austausch von Cyber Threat Intelligence und Best Practices zwischen den untergeordneten SOCs. Zusätzlich können bei Bedarf forensische Untersuchungen angeboten werden. Die Größenordnung des betrachteten Netzwerks liegt bei über 500.000 Nutzern bzw. IP-Adressen.

Die wesentlichen Aufgaben eines SOC lassen sich wie folgt zusammenfassen: [5][12][14]

1. Sammeln und Archivieren von sicherheitsrelevanten Events und Daten;
2. Aufbereitung und Überwachung von Lagebildinformationen;
3. Herausfiltern von FPs und Eskalation von Sicherheitsvorfällen;
4. Reaktion und Eindämmung von Angriffen;
5. Nachhaltige Verbesserung der Sicherheitsmaßnahmen und Härtung der Infrastruktur;
6. Beratende Tätigkeiten und Schulungen für Organisationspersonal;
7. Zusammenarbeit mit anderen zugehörigen Sicherheitsteams. Darunter fallen bspw. Computer Security Incident Response Teams (CSIRTs) oder auch Cyber Threat Intelligence Teams.

Wie diese Aufgaben übernommen werden können, wird in Abschnitt 2.1.2 dargestellt.

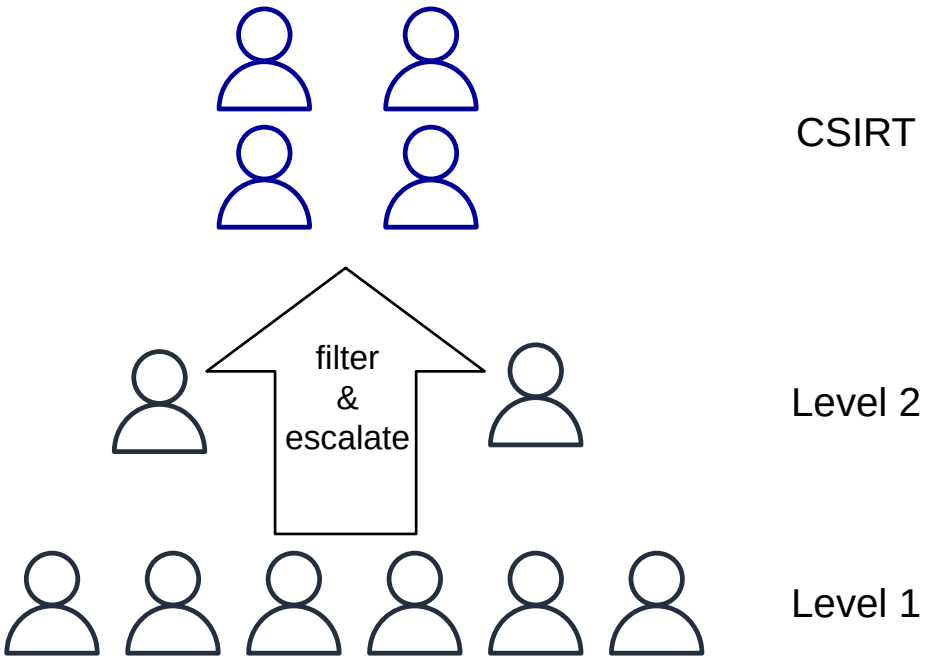

Abbildung 2.1: Hierarchische Anordnung von Analysten in einem SOC [8]

2.1.2 Arbeitsweise eines SOCs anhand eines Beispiels

In Abschnitt 2.1.1 wurden verschiedene Kategorien vorgestellt, wie SOCs aufgebaut sein können. Im Folgenden wird ein konkretes Beispiel für ein *internes zentralisiertes SOC* zur Verdeutlichung der Arbeitsweise erläutert. So kann eine hierarchische Anordnung von Sicherheitsanalysten unterschiedlicher Erfahrungsstufen aufgebaut werden. Diese Hierarchie kann dann in einem CSIRT enden, welches aus sehr erfahrenen Analysten besteht und im Falle eines erfolgreichen Angriffs (Security Incident) eingeschaltet wird, um die Eindämmung, Schadensanalyse und digitale Forensik zu übernehmen. Dies ist insbesondere dann notwendig, wenn das SOC eine primär koordinierende Rolle einnimmt und somit ein Team benötigt wird, welches die Umsetzung der Sicherheitsmaßnahmen übernimmt.[5][12]
Abbildung 2.1 zeigt eine beispielhafte Hierarchie von Sicherheitsanalysten in einem SOC. Dabei ist die hierarchische Anordnung auf zwei Stufen beschränkt. Analysten der ersten Stufe betrachten Alarme und filtern auf FPs. Wenn ein Alarm von Analysten als FP klassifiziert wird, schließt dieser den Alarm. Falls die Analysten mehr Informationen benötigen, oder feststellen können, dass es sich um einen tatsächlichen Sicherheitsvorfall

handelt, wird der Alarm an Analysten der zweiten Stufe weitergeleitet (eskaliert).
Analysten der zweiten Stufe sind erfahrenere Analysten, welche die Alarme ihrerseits überprüfen. So werden auch in diesem Verarbeitungsschritt Fehlalarme aussortiert und Alarme die auf einen Sicherheitsvorfall hindeuten eskaliert.
An dieser Stelle könnte die Hierarchie um weitere Stufen erweitert werden, oder ein zugehöriges CSIRT eingeschaltet werden, um auf potentielle Sicherheitsvorfälle zu reagieren. Dabei ist zu beachten, dass der Aufbau eines SOC sehr individuell ist und dabei unterschiedliche Ansätze verfolgt werden können. Elementar ist allerdings die Unterstützung menschlicher Analysten und die Automatisierung von Verarbeitungsschritten. So liegt häufig ein Prozess zugrunde, welcher eingehende Daten miteinander korreliert und durch strukturiertes Zusammenführen dafür sorgt, dass Analysten weniger Alarme betrachten müssen, bzw. zielgerichteter arbeiten können, da die Alarme bereits zusammengefasst wurden. Die technische Umsetzung eines SOC sollte die Anbindung von Log-Daten und Sicherheitsevents, wie bspw. IDS-Alerts, übernehmen.[5]
Mit Hilfe der so gesammelten Daten können Visualisierungen erzeugt werden, welche Lagebildinformationen veranschaulichen und die Überwachung dieser vereinfacht. Des Weiteren kann eine Alarmierungsstrategie verfolgt werden, welche bei besonders relevanten Daten (wie bspw. dem Verdacht auf eine DDoS-Attacke) frühzeitig warnen. Je nach Anwendungsfall entstehen durch die Anforderungen an das SOC auch Anforderungen an die zugrundeliegende Alarmkorrelation. Eine genauere Betrachtung dieser Unterteilung wird in Abschnitt 2.3 vorgenommen. Trotz der technischen Unterstützung müssen Sicherheitsanalysten eine Vielzahl von Alarmen bearbeiten. Diese können Fehlalarme sein, oder tatsächlich relevante Informationen beinhalten. Um diese Fehlalarme herauszufiltern und aus dem weiteren Verarbeitungsprozess auszuschließen, kann die am Anfang dieses Abschnitts geschilderte Hierarchie von Analysten eingesetzt werden. Teilweise werden zusätzlich noch andere Mechanismen herangezogen, um bereits Vorschläge für Analysten zu tätigen, oder automatisiert zu filtern. [15][16]
Auf den zugrundeliegenden Korrelationsprozess wird in Abschnitt 2.2 eingegangen. Technische Umsetzungsmöglichkeiten dessen werden in Abschnitt 2.1.4 erläutert.

2.1.3 Alert Fatigue

Durch die Vielzahl der eingehenden Alarme in einem SOC sind die beschäftigten Sicherheitsanalysten stark ausgelastet. Laut [5] zielen die meisten SOCs auf eine Durchsatzrate von 1.000 bis 3.000 Alarmen pro Tag pro Stufe der Analystenhierarchie ab. Dabei

verarbeitet die Infrastruktur des genannten Beispiel-SOC täglich ca. 3 Milliarden Sicherheitsevents, während das gesamte Netz zwischen 100 Milliarden und 1 Billionen Events pro Tag produziert. Das dauerhafte Untersuchen von potentiell relevanten Alarmen mit einer hohen Fehlalarmrate führt zu dem Effekt des *Alert Fatigue*. Dieser besagt, dass Sicherheitsanalysten die eingehenden Alarme weniger aufmerksam betrachten, da die Annahme verstärkt wird dass es sich um einen Fehlalarm handelt. Zusätzlich nimmt die Reaktionszeit zu. Als Resultat werden wichtige Alarme potentiell falsch eingestuft und die Reaktionszeit auf einen tatsächlichen Sicherheitsvorfall steigt. Dies senkt die Qualität des angebotenen Services und damit auch die Qualität der Netzwerksicherheit des betreuten Netzwerks. [17]
Um diesem Problem vorzubeugen gibt es verschiedene ML basierte Ansätze, um die Priorisierung von Alarmen zu automatisieren. Dabei soll die Arbeitsauslastung der Sicherheitsanalysten reduziert werden, indem FPs und Alarme niedriger Priorität herausgefiltert werden. Allerdings wird in diesem Zusammenhang die zugrundeliegende Problematik von „explainable AI“ deutlich. Wenn trainierte ML Models Rückschlüsse ziehen, welche für Analysten und Stakeholder nicht mehr nachvollziehbar sind, ist es schwer dem System zu vertrauen [11][18][19][20]. In [15] wird ein Ansatz vorgestellt bei welchem die Rückschlüsse eines Models verdeutlicht und kritisch hinterfragt werden sollen.
Allerdings kann eine Abschwächung des Alert Fatigue Effekts auch mit der Reduzierung von Fehlalarmen und einer hochqualitativen Alarmkorrelation erreicht werden.

2.1.4 Tooling: SIEM

SIEM-Systeme spielen eine zentrale Rolle im Aufbau eines SOCs, da sie dazu beitragen die Sicherheitsanalysten in ihrer Arbeit zu unterstützen und zu entlasten. SIEM-Systeme haben das Ziel die Informationen aus den zugrundeliegenden Datenquellen (siehe Abschnitt 2.1.4) zusammenzuführen und somit den lokalen Fokus der jeweiligen Datenquellen aufzuheben. Somit können gesamtheitliche Informationen über die Vorkommnisse von sicherheitsrelevanten Events in der betrachteten Infrastruktur gewonnen werden. In Abschnitt 2.2 wird genauer erläutert welche Techniken der Alarmkorrelation verwendet werden können, um eine kompakte Repräsentation von IDS-Alarmen zu erzielen. Diese Repräsentation unterstützt die Sicherheitsanalysten, da diese weniger manuellen Aufwand betreiben müssen um tatsächliche Sicherheitsvorfälle von FPs zu unterscheiden und diese ggf. zu eskalieren.[5][4]
Abbildung 2.2 zeigt die Bildung und Darstellung von Meta-Alerts im Kontext eines SOC.

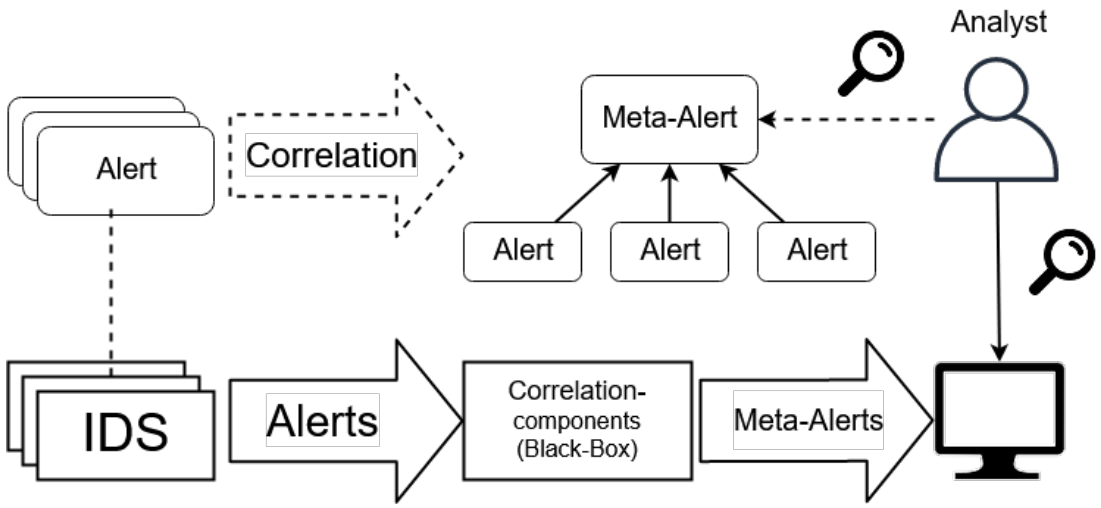

Abbildung 2.2: Bildung von Meta-Alerts im Zuge der Korrelation innerhalb eines SIEM-Systems unter Berücksichtigung der Arbeitsweise von SOC-Analysten

Dabei werden in diesem Beispiel IDS-Alerts mit Hilfe eines Korrelationsprozesses in eine Baumstruktur gebracht, welche auf Meta-Alert Ebene wichtige Informationen bündelt und gleichzeitig auf die Basis-Alerts verweist. Somit können Analysten schneller erkennen um welche Art von Alarmen es sich handelt. Wenn noch weiterführende Korrelationsschritte durchgeführt wurden, können ggf. noch weitere Informationen, wie bspw. ein vermutetes Angriffsmuster auf Meta-Alert Ebene hinterlegt sein.[4]

Aufbau

SIEM-Systeme bündeln angeschlossene Datenquellen, normalisieren eingehende Events und aggregieren diese. Ziel ist es, den lokalen Fokus der angebundenen Datenquellen aufzuheben und ein präziseres Bild von den sicherheitsrelevanten Vorgängen innerhalb des überwachten Netzwerks zu erlangen. Abbildung 2.3 zeigt den grundsätzlichen Aufbau eines SIEM-Systems. Dabei wird die Heterogenität der angebunden Datenquellen verdeutlicht, welche neben IDS auch Intrusion Prevention Systems (IPS), wie bspw. Firewalls, beinhaltet oder die System-Logs eines Application Servers. Die von diesen Datenquellen erzeugten Events werden von dem SIEM-System entgegengenommen und weiterverarbeitet. Die Verarbeitung endet dann in einem User Interface welches die relevanten, aus den Daten entstandenen Alarme darstellt.[5]
Die Architektur eines solchen Systems kann zentralisiert oder ein verteiltes System sein.

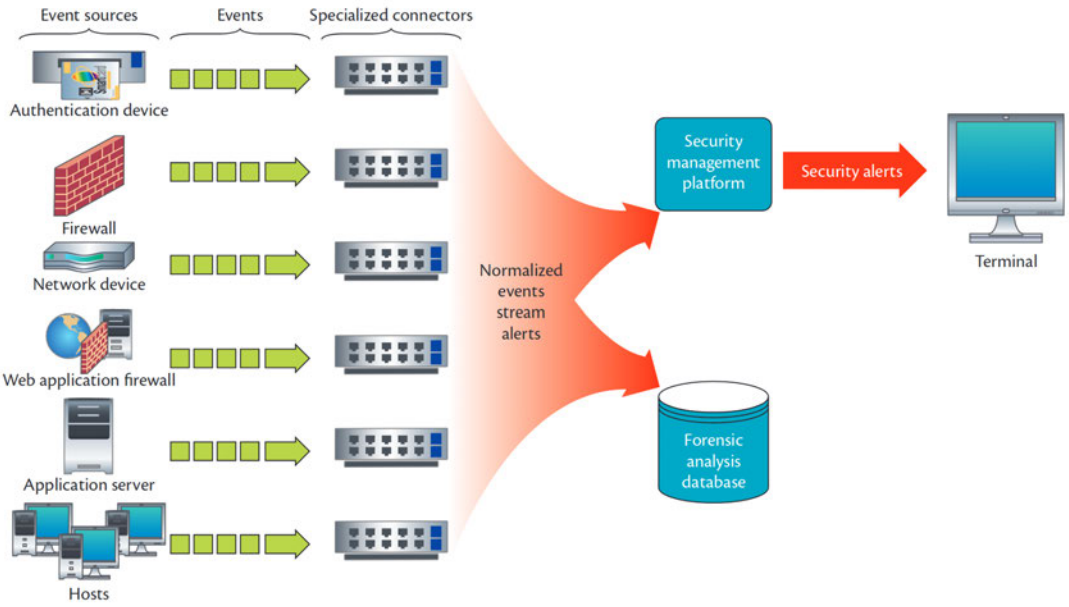

Abbildung 2.3: Allgemeiner Aufbau eines SIEM-Systems [5]

Diese beiden Ansätze beinhalten folgende Vor- und Nachteile:
Ein **zentraler Ansatz** beinhaltet keinen Kommunikationsaufwand über die Systemgrenze hinweg, stellt zentralisiert Daten zur Verfügung und kann ein einem Punkt gewartet werden. Die Nachteile eines solchen Ansatzes sind allerdings die fehlende Skalierbarkeit des Systems und der resultierende „Single point of Failure“, welcher für Angreifer ein lukratives Angriffsziel darstellt, da die gesamte IDS-Infrastruktur in diesem System zusammengeführt wird.
Ein **verteilter Ansatz** adressiert die Nachteile des zentralen Ansatzes und bietet Vorteile in Skalierbarkeit und Ausfallsicherheit. Allerdings beinhalten verteilte Systeme gesteigerte Komplexität in der Kommunikation der Subsysteme sowie in der Wartung.

Datenquellen

Die Datenquellen eines SIEM-Systems können aus verschiedenen Kategorien stammen. Dabei liefern IDSs bereits vorstrukturierte Daten in Form von IDS-Alerts. Diese beinhalten auf Basis des zugrundeliegenden Systems zusammengefasste Informationen zu sicherheitsrelevanten Events, die entweder auf Netzwerkebene analysiert wurden (Network-based IDS), oder direkt auf überwachten Systemen festgestellt wurden (Host-basted IDS).

Im Gegensatz zu dieser Art von Datenquellen eines SIEM-Systems existieren noch weitere Datenquellen, welche Rohdaten wie bspw. System-Logs liefern. Der Umgang mit der Diversität der Datenquellen und die Normalisierung der so eingehenden Events ist eine der Hauptaufgaben eines SIEM-Systems. Die in der Abbildung 2.3 dargestellte Anbindung von Datenquellen beinhaltet die Besonderheit, dass die Events der jeweiligen Datenquellen bereits von spezieller Connector Hardware normalisiert werden. Dies ist allerdings eine andere Form der Normalisierung, da dies beinhaltet dass bspw. Log-Dateien in ein Format gebracht werden, welche im Anschluss weiterverarbeitet werden können. Nichtsdestotrotz sind die so entstehenden Formate nicht unbedingt einheitlich. Dadurch muss das verarbeitende SIEM-System als ersten Schritt der Verarbeitung eine Normalisierung der angenommenen Formate vornehmen (siehe auch Abschnitt 2.2). Dies ist notwendig um in den Folgeschritten der Verarbeitung eine möglichst präzise Korrelation zwischen den verschiedenen Daten durchführen zu können. Dieser Zusammenhang zwischen den Verarbeitungsschritten wird auch in Abbildung 2.4 verdeutlicht. Dort bildet die Normalisierung den Einstiegspunkt. [4][5]

2.2 Alarmkorrelation

Dieser Prozess ist das Kernstück dieser Arbeit und basiert maßgeblich auf der Korrelationskette nach Valeur et. al. [4] und auf den Untersuchungen des BSI [21]. Das bei der Umsetzung gewählte Vorgehen wird in Kapitel 4 erläutert. Die Korrelation von Alarmen oder Events ist eine der Hauptaufgaben eines SIEM-Systems. Dabei werden Datenpunkte anhand bestimmter Kriterien in Relation gesetzt und ggf. zusammengefasst. Valeur et. al. beschreiben einen ausgereiften Korrelationsprozess anhand mehrerer Verarbeitungsschritte. Die Verarbeitungsschritte sind in Abbildung 2.4 dargestellt. Das Resultat sind *Meta-Alerts*, welche aus den Informationen der zusammengeführten *Basis-Alarme* bestehen. Somit spielen bestimmte Kernkriterien der verarbeiteten Events wie bspw. die Ausgangs- und Ziel-IP-Adresse eine besondere Rolle, welche auf höchster Ebene des Meta-Alerts zusammengeführt werden. Allerdings gibt es neben dem Vorgehen von Valeur et. al. noch weitere Korrelationstechniken, auch wenn diese eher Teilgebiete der Korrelation betrachten. Diese werden in Abschnitt 2.2.1 kurz erläutert.
Die Korrelationskette von Valeur et. al. besteht aus zehn Arbeitsschritten, welche im Folgenden kurz zusammengefasst werden:

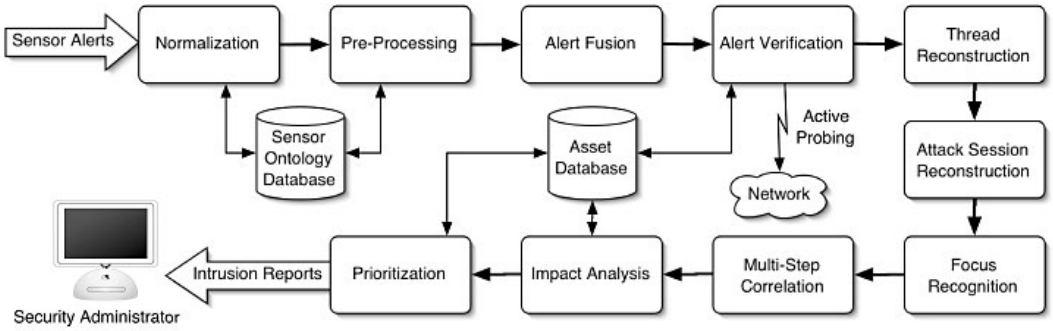

Abbildung 2.4: Korrelationsschritte nach Valeur et al. [4]

1. **Normalization**: Der erste Schritt besteht, wie bereits in Abschnitt 2.1.4 bereits erläutert, darin die eingehenden Events in ein einheitliches Format zu bringen. Somit wird eine Entkopplung der nachfolgenden Verarbeitungsschritte von den jeweiligen sensorspezifischen Formaten gewährleistet. Die in der Abbildung dargestellte Sensor Ontology Database wird verwendet um die Werte des eingehenden Formats auf das intern verwendete Format abzubilden.

2. **Pre-Processing**: In diesem Korrelationsschritt werden normalisierte Alarme mit Informationen angereichert. Dies kann beispielsweise das Abbilden von Verbindungsinformationen auf ein intern verwendetes Flow-Typen-System sein, welches später dazu verwendet werden kann, Alarme zu gruppieren.

3. **Alert Fusion**: Im Zusammenhang mit redundant ausgelegter IDS-Infrastruktur kann es dazu kommen, dass von verschiedenen Sensoren der gleiche Alarm ausgelöst wird. Der Schritt der Alert Fusion führt eine Deduplizierung von Alarmen durch.

4. **Alert Verification**: Um frühzeitig überprüfen zu können, ob es sich um einen FP handelt, wird in diesem Korrelationsschritt versucht zu ermitteln ob der im Alarm enthaltene Angriff erfolgreich war. Dabei wird auf eine Asset Database zugegriffen, welche Informationen über die im Netzwerk enthaltene Infrastruktur enthält. So kann zum Beispiel überprüft werden, ob der im Alarm enthaltene Port in der Beschreibung des angegriffenen Systems auftaucht. Zusätzlich kann aktiv gescannt werden, ob der hinter dem Port ansprechbare Service noch erreichbar ist.

5. **Thread Reconstruction**: In diesem Korrelationsschritt werden Alarme zusammengefasst, die vom gleichen Angreifer ausgehen und das gleiche Ziel adressieren.

Dazu können die IP-Adressen und die verwendeten Ports der involvierten Systeme verwendet werden.

6. **Attack Session Reconstruction**: An dieser Stelle des Korrelationsprozesses sollen Network-based Intrusion Detection System (NIDS)-Alarme mit Host-based Intrusion Detection System (HIDS)-Alarmen in Verbindung gebracht werden, um die Ausbreitung des Angriffs zu erfassen. Allerdings kann dieses Ziel nicht immer erreicht werden, da der Zusammenhang zwischen Netzwerkaktivitäten und Host-Aktivitäten nicht präzise hergestellt werden kann. Als Hilfsmittel für diese Aufgabe kann das zeitliche Auftreten der Ereignisse genutzt werden, auch wenn dieses Vorgehen keine lang anhaltende oder langsame Angriffe erkennen kann.

7. **Focus Recognition**: Das Ziel dieses Verarbeitungsschrittes ist die Aggregation von Alarmen die entweder vom gleichen System ausgehen, oder Alarmen die das gleiche System als Ziel haben. So können breitgefächerte Angriffe wie DDoS Attacken oder groß angelegte Scans des Netzwerks zu einem einzigen Meta-Alarm zusammengeführt werden.

8. **Multi-Step Correlation**: In diesem Schritt sollen Angriffsmuster erkannt werden und so ein abstrakterer Kontext zwischen den Meta-Alarmen hergestellt werden. Dazu muss allerdings ein Mechanismus existieren, anhand dessen Angriffsszenarien beschrieben werden können, um Alarme die zu einem Szenario passen zusammenzufassen.

9. **Impact Analysis**: Um die Auswirkungen eines Angriffs zu erfassen, werden in diesem Schritt Meta-Alarme mit den betroffenen Systemen abgeglichen, um somit einen Zusammenhang zu auftretenden Seiteneffekten wie bspw. Störungen eines Services herzustellen. Dazu müssen in der Asset Database Informationen über die auf den Systemen betriebenen Services, sowie über Abhängigkeiten dieser enthalten sein.

10. **Prioritization**: In diesem letzten Verarbeitungsschritt werden die resultierenden Meta-Alarme anhand der betroffenen Systeme und Services priorisiert. Diese Informationen müssen ebenfalls in der Asset-Datenbank gespeichert sein.

2.2.1 Korrelationstechniken

Die Korrelation von Alarmen kann anhand verschiedener Techniken vorgenommen werden. Im Folgenden werden einige Kategorien von Strategien vorgestellt und verschiedene Ausprägungen dieser Kategorien betrachtet und erläutert.[22]

Similarity based

Korrelationstechniken dieser Kategorie gehen davon aus, dass gleichartige Alarme auch die gleiche Ursache (engl. *Root Cause*) haben. Deshalb werden Alarme die bestimmte Kriterien erfüllen gruppiert und zusammengefasst. Anhand welcher Kriterien dies geschieht ist abhängig von der gewählten Strategie. Im Folgenden werden zwei Ausprägungen erläutert:
Attribute based: Bestimmte Attribute der Alarme werden betrachtet, wie bspw. die Ausgangs- und Ziel-IP-Adresse, der Ausgangs- und Ziel-Port, der betriebene Service, etc.. Dabei werden Metriken herangezogen um den mathematischen Abstand zu bestimmen. Anhand der festgelegten Schwellenwerte kann so bestimmt und konfiguriert werden, wie ähnlich die Attribute sein müssen, damit Alarme korreliert werden. Diese Technik kann eingesetzt werden um verschiedene Ziele zu erreichen. Dabei kann bspw. die Fusion (Deduplizierung) von Alarmen aus unterschiedlichen Quellen angestrebt werden [23], oder eine DDoS-Erkennung implementiert werden. [24]
Temporal based: Es wird davon ausgegangen, dass relevante Alarme kurz nach der eigentlich schädlichen Handlung auftreten, oder in einem zeitlichen Zusammenhang zueinander stehen. Dabei können bspw. Zeitfenster festgelegt werden, in denen Alarme betrachtet werden. Somit wird die Menge der zu betrachtenden Alarme eingeschränkt. Dieses Vorgehen kann bspw. für die Fusion von Alarmen genutzt werden, die von der gleichen (vermeintlich schädlichen) Aktion im Netzwerk ausgelöst wurden (siehe auch Abschnitt 4.3.6). [25][26][27]

Sequential based

In dieser Kategorie von Korrelationstechniken wird die Reihenfolge der eingegangenen Alarme betrachtet. Dabei werden anhand von Prädikaten Vorbedingungen (engl. *prerequisites*) und die daraus folgenden Auswirkungen (engl. *consequences*) eines Angriffs definiert. So können direkt Rückschlüsse über potentielle Angriffsstrategien gezogen werden.

Dieses Vorgehen kann in weitere Subkategorien unterteilt werden, welche im Weiteren erläutert werden.

Prerequisites and Consequences: Bei diesem Ansatz wird davon ausgegangen, dass alle eingehenden Alarme in ihrer Repräsentation für schädliches Verhalten, mit Vor- und Nachbedingungen erfasst werden können. Dabei wird angenommen, dass ältere Alarme Vorbereitungen für jüngere Alarme waren. Somit sind ältere Alarme oftmals auch die Vorbedingung für erfolgreiche jüngere Alarme. Dabei werden alle Alarme miteinander korreliert, auf die eine so zustande kommende Vorgänger-/Nachfolgerbeziehung zutrifft. Dabei wird auf eine definierte Wissensdatenbank zugegriffen, welche die bekannten Vor- und Nachbedingungen beinhaltet.[28]

Graphs: Bei dieser Technik wird das das überwachte Netzwerk als *Dependency Graph* modelliert. Dabei bilden die Systeme des Netzwerks die Knoten des Graphen. Die gerichteten Kanten des Graphs werden durch „ist-abhängig-von“ Relationen gebildet. Diese können bspw. zwischen einem Webservice und dem Router über welchen die Kommunikation stattfindet bestehen. Die eingehenden Events werden dann auf die zugehörigen Knoten des Graphen abgebildet. Anschließend wird eine Breitensuche durchgeführt um möglichst viele markierte Knoten mit der gleichen „ist-abhängig-von“ Relation zu einem gemeinsamen Zielknoten zu finden. Alle Events für die diese Eigenschaft zutrifft, werden korreliert. [29]

Codebook: Um ein *Codebook* zu generieren wird Expertenwissen über das überwachte Netzwerk verwendet um eine Kodierung von feststellbaren Symptomen und den zugrundeliegenden Problemen vorzunehmen. Dabei bilden die Symptome die Spalten und die Probleme bilden die Zeilen der so entstehenden Matrix. Anschließend wird die Matrix an jeder Stelle mit einer 1 oder einer 0 gefüllt, je nachdem ob das gegebene Symptom festgestellt wurde oder nicht. Alarme können mit Symptomen in Verbindung gebracht werden und so als Vektor codiert werden. Alle Alarme in einem bestimmten Mathematischen Abstand zu einem Problem werden korreliert.[30]

Hidden Markov Models: Hidden Markov Models eignen sich zur Erkennung von mehrstufigen Angriffen. Dabei kann anhand der Statustransitionen abgebildet werden, dass bestimmte Angriffe auf vorher erfolgten Angriffsvorbereitungen aufbauen. Diese zeitlich von einander losgelösten Angriffsphasen werden bspw. nicht von den im vorherigen Abschnitt *Similarity based* erläuterten Korrelationstechniken erfasst, da die einzelnen Angriffsphasen zeitlich weit auseinander liegen können. [31]

Bayesian Networks: Diese Korrelationstechnik nutzt Bayesian Networks, um die wahrscheinlichste Erklärung für eine Sequenz von Alarmen zu finden. Ein Bayesian Network kann aus einem Abhängigkeitsgraphen abgeleitet werden, welcher eine formale Reprä-

sentation der Abhängigkeiten von Services und betrachteten Rechnern darstellt (siehe Kategorie „Graph“). Dieser Ansatz ist robuster gegenüber nicht detektierten Alarmen, da das Ergebnis eine Wahrscheinlichkeit für einen detektierten Angriff ist, welche im Fall eines ausbleibenden Alarms lediglich geringer ausfällt.[32]

Neural Networks: Neuronale Netze (engl. *Neural Networks*) können ebenfalls für die Alarmkorrelation eingesetzt werden. Dabei können unterschiedliche Ansätze gewählt werden. Neben der eigentlichen Alarmkorrelation kann auch das Herausstellen von Angriffsstrategien erzielt werden.[33]

Allerdings ist die Genauigkeit der Alarmkorrelation stark abhängig von der Qualität der Trainingsdaten. Sofern diese sichergestellt ist, bringen Neuronale Netze Vorteile in den Bereichen Geschwindigkeit der Korrelation, Flexibilität in der Erkennung oder Unabhängigkeit von menschlichen Experten. Auf der anderen Seite kann das Training und die Optimierung eines Neuronalen Netzes viel Zeit beanspruchen und basiert häufig auf einem „Versuch und Irrtum“-Ansatz.[34]

Kontextfreie Grammatik: Al-Mamory und Zhang beschreiben in ihrem Ansatz [35] wie eine kontextfreie Grammatik (engl. *Attribute Context-Free Grammar*) in Kombination mit dem Left-to-right (LR) Parser Algorithmus verwendet werden kann, um Alarme von mehrstufigen Angriffen zu korrelieren. Dabei wird Wissen über die Vor- und Nachbedingungen der Angriffsschritte und weiteres Wissen über Angriffsszenarien in die Grammatik einbezogen um eine Verbesserung der Korrelationsergebnisse zu erreichen. Die Erkannten Angriffsszenarien werden als Korrelationsgraphen dargestellt, in welchen die Kanten die zeitliche Abfolge darstellen und die Vertices die erkannte Handlung.

Case based

Korrelationstechniken dieser Kategorie arbeiten anhand einer Wissensbasis, welche konkrete Angriffsszenarien und deren Entgegenwirken beschreibt. Dabei wird diese Wissensbasis immer dann erweitert, wenn ein Angriff erfolgreich verhindert werden konnte bzw. aufgeklärt werden konnte. Dabei werden Lösungsansätze persistiert und bei weiteren ähnlichen Angriffen als Wissensgrundlage herangezogen um Aktionen zur Bekämpfung des Angriffs zu empfehlen. Ein solcher Ansatz kann gerade dann Aufschluss über mögliche Angriffsszenarien und Auswirkungen, sowie mögliche Reaktionen geben, wenn die Wissensbasis ausreichend und übersichtlich gefüllt ist. Allerdings kann das Erreichen dieses Ziels eine herausfordernde Aufgabe sein, da die Aufbereitung der Szenarien sehr aufwändig sein kann. Ein weiterer Nachteil der Ansätze dieser Kategorie ist, dass sie neuartige

Angriffe nicht erkennen können, da diese nicht Teil der Wissensbasis sind.[34]

Expert based: Experten basierte Ansätze (engl. *expert based*) bauen ihre Wissensbasis anhand menschlicher Experten auf. Dabei wird das Wissen entweder anhand von Expertenregeln (engl. *expert rules*) oder vordefinierten Szenarien (engl. *pre-defined scenarios*) definiert. Diese beiden Ansätze werden in den nächsten Abschnitten genauer erläutert. Beide Ansätze haben allerdings gemeinsam, dass der Definitionsaufwand der Wissensbasis hoch ist und gleichzeitig unflexibel gegenüber Änderungen. Dies führt zu höherem Wartungsaufwand, falls Änderungen an der betrachteten Infrastruktur vorgenommen werden. [34]

Expert rules: In diesen Fall wird die Wissensbasis mittels von Experten formulierten Regeln beschrieben. Hierbei handelt es sich um eine „WENN → DANN“ (engl. *„IF → THEN“*) Formulierung, sodass auf der linken Seite die zu erfüllenden Bestandteile der Regel stehen. Auf der rechten Seite stehen dann die Aktionen, welche im Falle des Auslösens einer solchen Regel ausgeführt werden sollen. [36] Dieser Ansatz ist weit verbreitet, da die Formulierung solcher Regeln unkompliziert ist.

Pre-defined scenarios: Ansätze dieser Kategorie ähneln dem Vorgehen der Expertenregeln stark. Allerdings werden in diesem Anwendungsfall eigene Beschreibungssprachen entwickelt um die Szenarien der Wissensbasis zu definieren. Dies führt dazu dass die beschriebenen Szenarien nicht generisch sind und je nach Beschreibungssprache anders definiert werden. Der Vorteil der definierten Szenarien ist, dass im Falle einer Korrelation bereits eine starke semantische Prägung stattgefunden hat, sodass das Angriffsmuster leicht erkennbar und verständlich ist.[34]

Inferred knowledge: Um den manuellen Aufwand der oben genannten Experten basierten Vorgehensweisen zu reduzieren, wurden Korrelationstechniken entwickelt, welche die zugrundeliegende Wissensbasis automatisiert erstellen. Dabei werden ML und Data Mining Algorithmen verwendet, um aus bestehenden Datensätzen Regeln oder Szenarien zu extrahieren. Ein Nachteil bei diesem Vorgehen ist, dass nur Regeln oder Szenarien extrahiert werden können, welche auch tatsächlich in dem Trainingsdatensatz enthalten waren. Dadurch wird die resultierende Korrelation unflexibel gegenüber neuartigen Angriffen (wie auch der zugrundeliegende Ansatz der Expertenregeln). Zusätzlich kann ein optimiertes Training viel Zeit und Rechenleistung in Anspruch nehmen.[34][37]

2.2.2 Multi-step correlation

Wie bereits eingangs ind diesem Abschnitt erwähnt, ist die Aufgabe dieses Verarbeitungsschritts die Erkennung von Angriffsszenarien. Diese Aufgabe unterscheidet sich von den anderen Aufgaben der Korrelationskette nach Valeur et. al. darin, dass Expertenwissen benötigt wird, um mögliche Szenarien zu definieren. Teilweise werden auch auf das betrachtete Netzwerk zugeschnittene Trainingsdaten benötigt, um präzise Ergebnisse liefern zu können. Für die Implementierung einer solchen Korrelation können die Korrelationstechniken aus den Kategorien *sequential based* (siehe Abschnitt 2.2.1) und *case based* (siehe 2.2.1) verwendet werden. Die dort geschilderten Techniken sind unabhängig von der chronologischen Reihenfolge der eingehenden Events und können so auch Advanced Persistent Threats (APTs) erkennen, sofern die Aktivitäten von der IDS-Infrastruktur erkannt werden. APTs sind Angriffe, welche nach erfolgreichem Eindringen über einen langen Zeitraum unauffällig bleiben, um nicht von IDS-Systemen bemerkt zu werden. Selbst wenn nach einiger Zeit Auffälligkeiten detektiert werden, fällt es häufig schwer den Zusammenhang zum initialen Angriff herzustellen. [3][38][39][40][2][41]
Multi-step correlation hat einen hohen Wert für die Verarbeitungskette, da resultierende Alarme direkt einem Angriffsszenario zugeordnet werden können. Dadurch können bei frühzeitiger Erkennung Gegenmaßnahmen eingeleitet werden, falls der Angriff noch aktiv ist und durch das Szenario potentielle Ziele bekannt sind. Zusätzlich kann durch die Erkennung eines bestimmten Angriffsmusters auch die Eindämmung von erfolgreichen Angriffen schneller erfolgen.

2.2.3 Ontologie basierte Korrelation

Korrelationssysteme dieser Kategorie greifen auf Ontologien zurück, welche Wissen über Angriffsmuster, Klassen von Angriffen, Kontexte von detektierten Alarmen und Abhängigkeiten von Angriffsschritten beschreiben. Dabei wird auf Beschreibungssprachen zurückgegriffen, welche teilweise auf Prädikatenlogik beruhen. Dies ermöglicht logische Zusammenhänge durch Abfragen an die Ontologie basierend auf den verarbeiteten Events. Durch die flexible Erweiterbarkeit von Informationsquellen in Ontologie basierten Ansätzen und den Versuch intuitive Beschreibungen von Expertenwissen zu ermöglichen, wird die Korrelations- und Aggregationsqualität solcher Systeme gesteigert. [42][43]

2.3 Abgrenzung der Einsatzgebiete

Das in einem SOC verwendete SIEM-System kann je nach Anforderung unterschiedliche Einsatzgebiete haben. Die so abzugrenzenden Themenschwerpunkte haben unterschiedliche Anforderungen in Bezug auf die Verarbeitungszeit und das Reporting. Die im Folgenden beschriebenen Einsatzgebiete sind Alarmierung, Monitoring und Forensik.

2.3.1 Alarmierung

Wenn ein System hauptsächlich zur Alarmierung eingesetzt werden soll, wird zugunsten der Verarbeitungszeit die Genauigkeit der Korrelation vernachlässigt. Dabei geht es darum, möglichst frühzeitig Indikatoren für dedizierte Angriffe zu erkennen und zu melden. Dies kann neben dem Fokus eines Systems auch als Anforderung an ein System der anderen beiden Einsatzgebiete verstanden werden. [21][5]

2.3.2 Monitoring

Bei diesem Einsatzgebiet geht es darum, Lagebildinformationen des überwachten Netzwerks zu erheben und geeignet darzustellen. Da verschiedene SIEM-Systeme je nach Implementierung unterschiedliche Daten generieren und der Visualisierung zur Verfügung stellen, müssen für das jeweilige SIEM geeignete Visualisierungsmöglichkeiten verwendet oder implementiert werden.[44][45]
Die Qualitätsanforderung an die durchgeführte Korrelation steigt im Vergleich zum Einsatzgebiet der Alarmierung bzw. Frühwarnung. In diesem Fall wird allerdings mehr Zeit für eine präzisere Korrelation benötigt. Dadurch sinkt die Laufzeitanforderung an ein solches System.

2.3.3 Forensik

SIEM-Systeme haben durch die Qualität der zugrundeliegenden Alarmkorrelation die Möglichkeit, auch post-hoc forensische Untersuchungen zu unterstützen [5]. Dabei sinkt die Laufzeitanforderung an das System, da bereits festgestellt wurde, dass erfolgreiche Angriffe auf das überwachte Netzwerk durchgeführt wurden. So steigt allerdings der Anspruch an die Korrelationsqualität, damit detaillierte Informationen über die vorangegangenen Angriffe erhalten werden können.

3 Ziele und Anforderungen an die Alarmkorrelation

In diesem Kapitel werden die Ziele und die resultierenden Anforderungen an die Alarmkorrelation hergeleitet und erläutert.

3.1 Ziele

Die in Kapitel 2 herausgearbeiteten Grundlagen und aktuellen Entwicklungen auf dem Gebiet der Alarmkorrelation liefern grundlegende Ziele, die im Folgenden genauer betrachtet werden.

3.1.1 Variabilität im Einsatzgebiet

Je nach Einsatzgebiet werden verschiedene Anforderungen an die Alarmkorrelation gestellt (siehe Abschnitt 2.3). Tabelle 3.1 zeigt eine Einordnung der Anforderung von drei Einsatzgebieten. Wenn das System zur frühzeitigen Alarmierung eingesetzt werden soll, bestehen bspw. keine hohen Anforderungen an die Vollständigkeit der Korrelation oder eine geringe FP-Rate. Diese Eigenschaften werden zugunsten einer geringen Laufzeit (vergangene Zeit von Eingang des Alarms im SIEM-System bis zur Alarmierung eines Analysten) in der Priorität herabgestuft.
Wenn das System dazu eingesetzt werden soll, Lagebildinformationen zu sammeln, befindet es sich in allen in der Tabelle genannten Eigenschaften im mittleren Bereich, da keine der Eigenschaften einer anderen gegenüber bevorzugt wird. Allerdings folgt daraus, dass keine der Eigenschaften mit erhöhter Priorität behandelt wird.
Das dritte Einsatzgebiet stellt gegenteilige Anforderungen an das System, als das Einsatzgebiet der frühzeitigen Alarmierung. Bei der Forensik ist die maximal tolerierte Laufzeit zweitrangig, während die Vollständigkeit der Korrelation die primäre Anforderung an

Eigenschaft/Einsatzgebiet	Alarmierung	Monitoring	Forensik
Vollständigkeit der Korrelation	-	o	+
Minimale Laufzeit	+	o	-
Geringe False-Positive Rate	o	o	+
Aussagekraft der Ergebnisse	o	o	+

Tabelle 3.1: Anforderungen an Eigenschaften der Alarmkorrelation von drei Einsatzgebieten.
Legende: +/o/- bedeutet starke/mittlere/geringe Anforderung

das System ist. Zusätzlich ist eine geringe FP-Rate von Interesse, da die Ergebnisse des Systems sehr genau betrachtet werden. Das aus dieser Darstellung resultierende Ziel ist die Flexibilität im Einsatzgebiet. Das System soll konfigurierbar sein und somit für mindestens die genannten drei Anwendungsgebiete einsetzbar sein.

3.1.2 Reduzierung der Anzahl von Alarmen

In Sektion 2.1 werden grundlegende Eigenschaften von SOCs genannt. Dabei wird u.a. auf die hierarchische Struktur von Analysten und deren Arbeitsweise eingegangen. Die damit einhergehenden Herausforderungen, wie der Umgang mit einer Vielzahl von (Fehl-) Alarmen, führen zum nächsten Ziel des in dieser Arbeit betrachteten Systems. Die hohe Auslastung von Analysten führt zu hohen Reaktionszeiten auf einen tatsächlichen Sicherheitsvorfall. Gelingt es die Gesamtanzahl der Alarme zu reduzieren, ohne dass dabei wichtige Informationen verloren gehen, würden die Analysten in ihrer Arbeit deutlich unterstützt werden, da Informationen in einer höheren Dichte vorliegen würden. Damit würde die manuelle Aggregation von Informationen reduziert werden oder ganz entfallen, welche in Abhängigkeit der verarbeiteten Alarme im System sehr aufwändig sein kann.

3.1.3 Reaktionszeit vermindern

Durch die Aggregation von Alarmen soll für die Nutzer des Systems eine Möglichkeit geschaffen werden, kompakter und zielgerichteter Alarme zu analysieren. Somit würde die Zeit reduziert werden, welche benötigt wird, um auf einen tatsächlichen Sicherheitsvorfall zu reagieren. Dieser Arbeitsschritt kann durch den Einsatz eines geeigneten Graphical

User Interfaces (GUI) unterstützt werden. Die Entwicklung eines solchen GUIs ist allerdings kein Bestandteil dieser Arbeit, da eine solche Untersuchung über den Fokus und den Rahmen dieser Arbeit hinausgehen würde. Wenngleich eine Möglichkeit geschaffen werden soll, die verarbeiteten Alarme zu sichten, zu sortieren und graphisch darzustellen. Zusätzlich kann eine Alarmierung in gewissen Szenarien dazu beitragen, dass die Reaktionszeit auf einen Angriff reduziert wird. Daher soll analysiert werden, an welchen Stellen im System eine Alarmierung für bestimmte Angriffsszenarien sinnvoll eingesetzt werden kann.

3.1.4 Erweiterbarkeit

Da sich die zugrundeliegende IDS-Infrastruktur stetig verändern kann, sollte das System auf weitere IDS-Alarmformate anpassbar sein. Dies bietet die Flexibilität, viele verschiedene Sensoren an das System anzuschließen und damit für die Korrelation relevante Daten an das System zu liefern. Auch die Erweiterung der Korrelationskette um weitere Verarbeitungsschritte sollte möglich sein, falls in Zukunft weitere Anforderungen an das System formuliert werden.

3.2 Anforderungen

Aus den in Abschnitt 3.1 geschilderten Zielen, lassen sich technische Anforderungen an das System ableiten. Diese werden im Folgenden erläutert.

3.2.1 Aggregation

Das beschriebene System soll IDS-Alarme verarbeiten und zusammenfassen (siehe Abschnitt 2.2). Ziel ist es dabei, eine möglichst starke Reduzierung der Anzahl der eingespielten Alarme zu erreichen, ohne dabei relevante Informationen zu verlieren. Dies ermöglicht, dass in einem auf das System aufsetzenden Prozess, Alarme konkreten Angriffsszenarien zugeordnet werden können (siehe 2.2.2).
Zusätzlich zahlt eine solche Aggregation auf die Reaktionszeit des Teams ein, welches das beschriebene System verwendet, da die Übersichtlichkeit der eingehenden Alarme gesteigert wird und der manuelle Aufwand des Suchens zusammengehöriger Alarme reduziert wird.

3.2.2 Laufzeit, Messbarkeit und Darstellung

Die Laufzeitanforderung an das System ergibt sich aus dem gewählten Einsatzgebiet (siehe Abschnitt 3.1.1). Bei der Entwicklung des Systems soll ermöglicht werden, die maximal tolerierte Laufzeit zu erhöhen (also die Anforderung an die Verarbeitungsgeschwindigkeit zu senken), um eine potentiell stärkere Reduzierung der Alarmmengen zu erzielen.
Zur Informationsgewinnung aus dem implementierten Proof of Concept soll eine Möglichkeit gefunden werden, die Laufzeit des Systems verständlich darzustellen. Um die Laufzeit zu messen, kann die Zeit ab der Erzeugung des ersten Alarms bis zur Erzeugung des resultierenden Meta-Alarms gemessen werden. Zusätzlich soll die Verarbeitungszeit auf Ebene der einzelnen Services anhand der Ein- und Austrittszeiten eines Alarms ermittelt werden können.

3.2.3 Flexibilität und Erweiterbarkeit

Der Korrelationsprozess soll um zusätzliche Arbeitsschritte erweiterbar sein. Dazu muss eine Architektur gewählt werden, welche das flexible Einbinden weiterer Services in die Verarbeitungskette ermöglicht.
Zusätzlich soll ermöglicht werden, weitere IDS-Formate zu verarbeiten. Vorerst kann davon ausgegangen werden dass ausschließlich NIDSs eingebunden werden sollen, da die Erweiterung um HIDSs als Quellen zu einer komplexeren Verarbeitung innerhalb der jeweiligen Services führen würde. Die Untersuchung von Netzwerkpaketen, bzw. NIDS-Alarmen wird unter dem Gesichtspunkt der Netzwerkanalyse als elementarem Bestandteil der IT-Sicherheit als ausreichend angenommen.

3.2.4 Skalierbarkeit

Damit das zu implementierende System eine hohe Anzahl von Alarmen verarbeiten kann, muss neben der vertikalen Skalierbarkeit auch die Möglichkeit berücksichtigt werden, anhand von Cloud-Technologien horizontal zu skalieren. Die horizontale Skalierbarkeit wird im Weiteren konzeptuell betrachtet und abschließend anhand einer der vorgenommenen Messreihen untersucht. Das Ziel der Arbeit ist die Implementierung lauffähiger Software, welche mit geringem Aufwand in einer Cloud-Umgebung in Betrieb genommen werden kann.

4 Architektur

Dieses Kapitel beschreibt die wesentlichen Architekturentscheidungen der Umsetzung des Systems. Dabei setzt die Konzeption auf bisherigen Arbeiten auf und orientiert sich an der Verarbeitungskette nach Valeur et al. [4] und den Ergebnissen des Bundesamt für Sicherheit in der Informationstechnik (BSI) [21]. Das Kapitel ist wie folgt gegliedert: Abschnitt 4.1 geht auf die infrastrukturellen Entscheidungen ein und beschreibt gewählte Technologien und die zugrundeliegende Hardware. Abschnitt 4.2 vergleicht verschiedene Varianten der Korrelationskette. Abschnitt 4.3 geht anschließend auf die Umsetzung der einzelnen Services ein. In Abschnitt 4.4 werden verwendete Datenstrukturen beschrieben und Übergabeformate erläutert. In Abschnitt 4.5 wird auf die Konzeption eines Frühwarnungsmechanismus eingegangen. Abschließend werden in Abschnitt 4.6 vorgenommene Optimierungen und die iterative Weiterentwicklung des Systems dargestellt.

4.1 Infrastruktur

Im Zuge der Umsetzung wurde eine Möglichkeit gesucht, ein technisch sowie fachlich erweiterbares System zu implementieren. Dazu wurde teilweise auf bereits existierenden Technologien und Open-Source Software aufgebaut. Die so getroffenen Architekturentscheidungen werden im Folgenden erläutert.

4.1.1 Hardware

Für die Entwicklung des Systems und die Durchführung der in Kapitel 5 geschilderten Messungen wurden drei verschiedene Rechner verwendet. Zum Betrieb der Umgebungsservices wurde eine von der Hochschule für Angewandte Wissenschaften (HAW) zur Verfügung gestellte virtuelle Maschine verwendet. Diese wurde mit zehn Kernen eines Intel(R) Xeon(R) E5-2650 v3 @ 2,30GHz Prozessors und 64GB RAM betrieben. Der

entwickelte Prototyp des Systems wurde auf einem weiteren Rechner mit 16 Kernen eines AMD Ryzen 7 5800u @ 1,9-4,4 GHz Prozessors und 32GB RAM ausgeführt. Um die Erzeugung von Testdaten von dem eigentlichen Prototypen zu entkoppeln, wurde für die Ausführung des Simulationsservice ein dritter Rechner verwendet. Dieser verfügte über 8 physische und 16 logische Kerne eines AMD Ryzen 7 2700X @ 3,7 GHz Prozessors und 32 GB RAM.

4.1.2 Docker Container und Docker-Compose

Um die Services auch in einer Cloud-Umgebung betreiben zu können und somit weitere Flexibilität des Systems zu gewährleisten, wurde von Beginn an mit Containern gearbeitet. In diesem Fall wurde die Docker-Engine verwendet. Durch die Verwendung von Containern kann eine Abstraktion vom verwendeten Betriebssystem geschaffen werden, sodass der entwickelte Proof of Concept reproduzierbar auf verschiedenen Host-Betriebssystemen nachgestellt werden kann, sofern Docker installiert ist. Unter Verwendung von docker-compose können zur Entwicklungszeit Docker-Container orchestriert und verwaltet werden. Im Anhang A befinden sich die verwendeten *docker-compose.yml* Dateien. Dabei wird zwischen der Konfigurationsdatei der entwickelten Services und der Konfigurationsdatei der Umgebungs-Services unterschieden. Mit Hilfe dieser Dateien können die einzelnen Services anhand von Umgebungsvariablen konfiguriert werden und verschiedene Architekturen miteinander verglichen werden. Dies kann bspw. durch die Veränderung der Reihenfolge der implementierten Services innerhalb der Korrelationskette realisiert werden. Allerdings ist zu beachten, dass die Verwendung der Konfigurationsdateien zur Inbetriebnahme auf einem einzelnen Hostsystem geeignet ist, nicht jedoch für eine Produktivumgebung. In diesem Fall sollten die definierten Container für die maximale horizontale sowie vertikale Skalierbarkeit als eigene Services in einer Cloud-Umgebung in Betrieb genommen werden.[46][47]

4.1.3 Verwendete Software

Für die Umsetzung des Systems wurde bestimmte Open Source Software verwendet. Die verwendete Software wird im Weiteren in ihrer Funktion für das implementierte System beschrieben.

Elastic-Stack

Der Elastic-Stack besteht aus vier verschiedenen Open Source Produkten zur Annahme, Normalisierung, Speicherung, zum Durchsuchen und Visualisieren von Datenreihen. Darunter fällt die Elasticsearch Engine, welche das Herzstück des Elastic-Stacks bildet und für das Speichern und Durchsuchen der Daten zuständig ist. Bezogen auf das zu implementierende System dient Elasticsearch als Speicher von Zeitreihen von korrelierten Alarmen.[48][49][50]
Die zweite verwendete Software-Komponente aus dem Elastic Stack ist Kibana. Kibana dient als Oberfläche um Messungen in Diagrammen darzustellen und um die vom System hinterlegten Daten zu durchsuchen. Dies zahlt auf die Anforderungen der Messbarkeit und Darstellung aus Abschnitt 3.2.2 ein.[51]
Elasticsearch und Kibana werden jeweils als eigene Docker-Container (siehe 4.1.2) betrieben. Kibana kann über einen ssh-Tunnel erreicht werden und über einen Internetbrowser bedient werden.

RabbitMQ

RabbitMQ ist ein Message-Queue-System mit integriertem Message-Broker [52][53] und dient dazu, die Kommunikation der einzelnen Services von einander zu entkoppeln. Dabei legen die jeweiligen Services Nachrichten in einer konfigurierbaren Output-Queue ab und lesen aus einer konfigurierbaren Input-Queue ein. Im Prototypen wurde darauf verzichtet, RabbitMQ als Cluster zu betreiben. Dies ist allerdings im Kontext einer größer angelegten Alert-Korrelation durchaus sinnvoll, um die zu verarbeitende Last auf verschiedene Instanzen zu verteilen. Es wurde für den Proof of Concept als ausreichend angenommen lediglich von einander unabhängige Queues auf einer einzigen Instanz des Message-Queue-Systems anzulegen.

4.1.4 Suricata

Suricata ist ein Open Source System, welches Funktionalitäten von IDSs, IPSs und *Network Security Monitoring*-Systemen kombiniert.[54][55] In dieser Arbeit wird Suricata als Beispiel-NIDS verwendet, dessen Output von dem entwickelten System verarbeitet

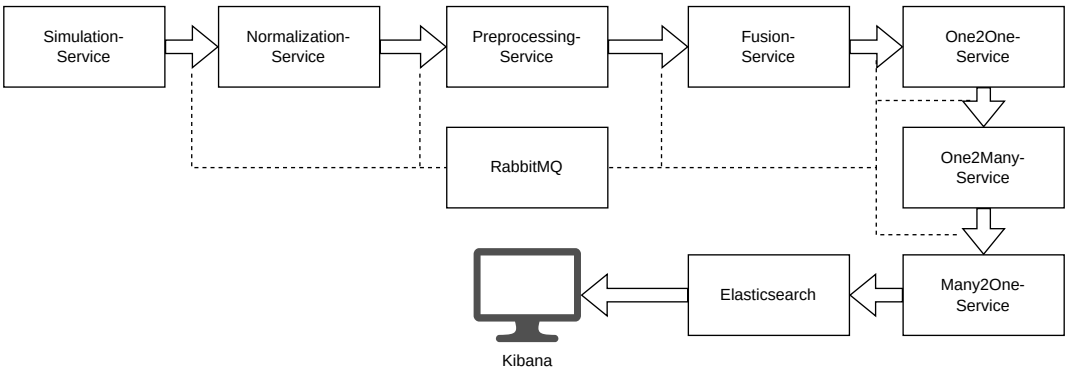

Abbildung 4.1: Abgeleitete Services aus der Verarbeitungskette nach Valeur et al.

werden kann. Dabei wird auf das *EVE-JSON-Format* zurückgegriffen, welches eine konfigurierbare Output-Schittstelle von Suricata bildet.[56] Die verwendeten Felder und damit die Konfiguration des EVE-Formats können dem Abschnitt 4.4.1 entnommen werden.

4.2 Anordnung der Services

Die in Abschnitt 2.2 vorgestellte Korrelationskette kann in einzelne Services unterteilt werden. Da die Reduzierung der FP-Rate kein Ziel dieser Arbeit ist, entfällt der Schritt der Alert Verificaton. Des Weiteren werden keine Services für die Schritte der *Impact Analysis* und der *Prioritization* implementiert, da diese vorraussetzen, dass die Systeme des überwachten Netzwerks bekannt sind. Von der Implementierung eines Services für den Korrelationsschritt der *Multi-Step Correlation* wird ebenfalls abgesehen, da dies ein eigenes Thema in der Alarmkorrelation darstellt, welches den Rahmen dieser Arbeit überschreiten würde. Da für den Prototypen angenommen wurde, das ausschließlich NIDS-Alarme ins System eingespielt werden, entfällt ebenfalls die Implementation einer *Attack Session Reconstruction*. Die resultierenden Services können Abbildung 4.1 entnommen werden.
Die Services auf der rechten Seite der Abbildung übernehmen die Aufgaben *Thread Reconstruction* (One2One-Service) und *Focus Recognition* (One2Many-Service und Many2One-Service) [21]. Die restlichen Services wurden analog zu den übernommenen Korrelationsschritten benannt. Die Reihenfolge der Services der *Focus Recognition* kann Auswirkungen auf das Ergebnis der Aggregation haben. Im Folgenden werden acht verschiedene Varianten ab dem One2One-Service dargestellt. Die vorangegangenen Services werden

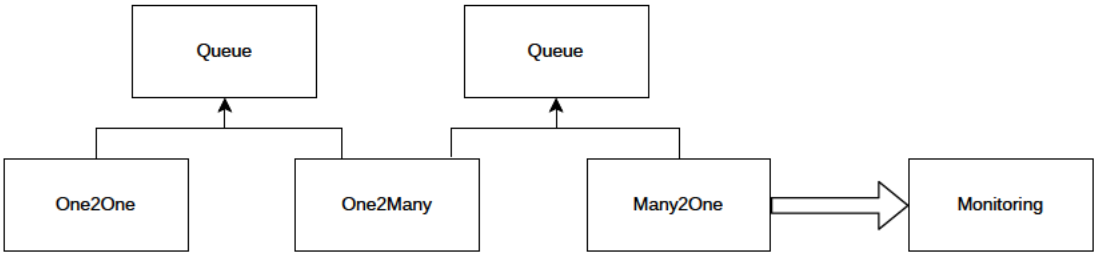

Abbildung 4.2: **Variante A**: Naiver Ansatz ohne variable Verarbeitungsmöglichkeit mit dem One2Many-Service vor dem Many2One-Service

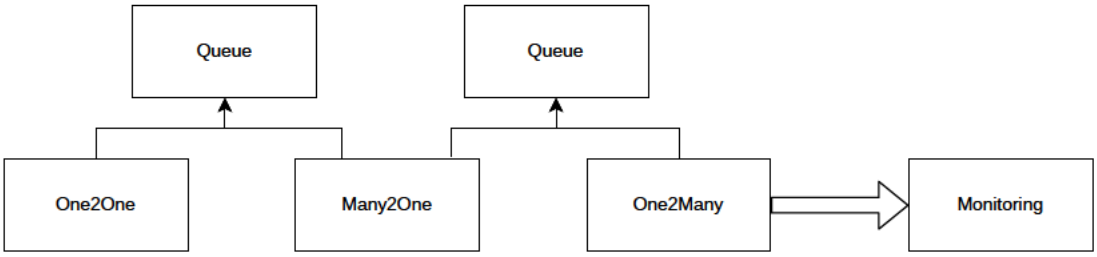

Abbildung 4.3: **Variante B**: Naiver Ansatz ohne variable Verarbeitungsmöglichkeit mit dem Many2One-Service vor dem One2Many-Service

von dieser Reihenfolge nicht betrachtet und tauchen deshalb nicht in den Darstellungen auf.

4.2.1 Varianten A und B: Sequentielle Verarbeitung

Die ersten beiden Varianten beschreiben einen sequentiellen Ablauf der Services in zwei unterschiedlichen Reihenfolgen. In der Variante A, dargestellt in Abbildung 4.2, werden die One2OneAlerts and den One2Many-Service weitergeleitet und im Anschluss werden die One2ManyAlerts vom Many2One-Service verarbeitet. In der Variante B, dargestellt in Abbildung 4.3, ist es genau andersherum. Dabei werden die One2OneAlerts an den Many2One-Service weitergeleitet und im Anschluss werden die resultierenden Many2OneAlerts vom One2Many-Service verarbeitet, bevor die Ergebnisse an die Darstellungs-Services übergeben werden. Die beiden geschilderten Varianten sind leicht umsetzbar, da lediglich die Reihenfolge der Services beachtet werden muss. Allerdings bieten die Varianten keinerlei Flexibilität in der Verarbeitung.

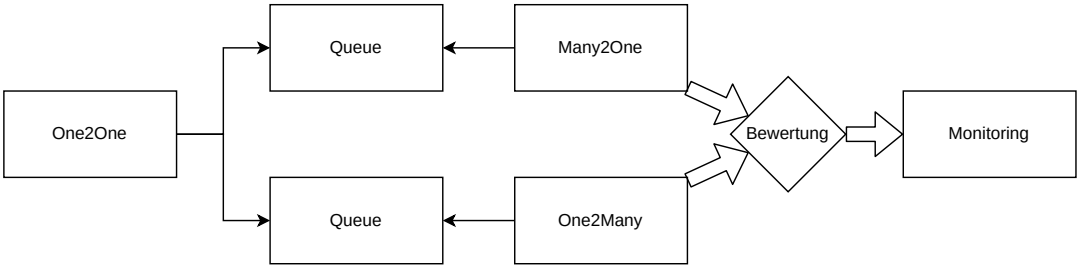

Abbildung 4.4: **Variante C**: Parallele Verarbeitung der One2One-Alerts mit nachgelagerter Bewertungskomponente

4.2.2 Variante C: Parallele Verarbeitung

Die in Abbildung 4.4 dargestellte, dritte Variante löst das Flexibilitätsproblem der Varianten A und B indem die One2OneAlerts dupliziert werden und die dahinter positionierten Services parallel arbeiten. In diesem Fall muss allerdings im Nachhinein festgestellt werden, welches der Ergebnisse akzeptiert werden soll, um eine Duplizierung der Alarme im Endresultat zu vermeiden. Dies kann durch einen nachgelagerten Bewertungsservice realisiert werden, welcher die beiden eingehenden Alarmströme miteinander vergleicht und im Anschluss zusammenführt. Dabei muss sichergestellt werden, dass die an den One2Many- und Many2One-Service gelieferten One2OneAlerts eine ID („correlationID“) zugewiesen bekommen, damit der Bewertungs-Service erkennen kann welche Alarme miteinander verglichen werden müssen.
Die in diesem Abschnitt geschilderte Variante C weist Verbesserungen in der Verarbeitungsflexibilität auf, allerdings muss ein zusätzlicher Service entwickelt werden, da die Monitoring-Services nicht ohne größeren Aufwand erweitert werden können. Zusätzlich kann die Bewertung nicht von den Services übernommen werden, da diese keine Informationen über die Verarbeitung des jeweils anderen Services haben. Für den so entstehenden Bewertungs-Service muss dann ein Regelsatz oder eine Heuristik erzeugt werden, damit dieser fundierte Entscheidungen treffen kann. Dabei müssen verschiedene Angriffsszenarien gegeneinander abgewogen werden. Zusätzlich können in dieser Variante nicht beide Services (One2Many und Many2One) durchlaufen werden, sodass ein Schritt der Korrelationskette ausbleibt.

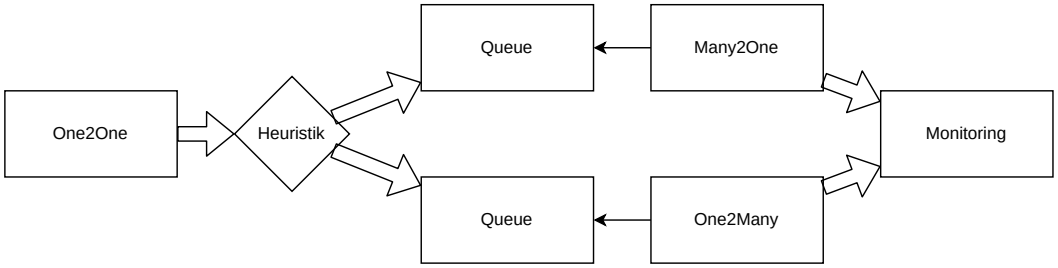

Abbildung 4.5: **Variante D**: Parallele Verarbeitung von One2One-Alerts durch Verteilung der Alarme auf den jeweils wahrscheinlichen Nachfolge-Service anhand einer Heuristik

4.2.3 Variante D: Heuristische Auswahl des Nachfolgeservices

Die Variante D bildet das Gegenstück zur beschriebenen Variante C und ist in Abbildung 4.5 dargestellt. Diese Variante entscheidet anhand einer vorgelagerten Heuristik zwischen dem One2One und den darauf folgenden Services an welchen dieser Services die One2OneAlerts weitergeleitet werden. Der Vorteil gegenüber der Variante C ist, dass keine One2OneAlerts dupliziert werden und demnach auch nicht im Nachhinein herausgefiltert werden müssen. Der Nachteil dieser Variante ist, dass die Entscheidung über den gewählten Verarbeitungszweig vor der eigentlichen Weiterverarbeitung getroffen werden muss. So kann bspw. nicht der Grad der Korrelation als Entscheidungskriterium herangezogen werden. Des Weiteren wird auch in dieser Verarbeitungsvariante nur einer der beiden Services durchlaufen. Dies verhindert einen potentiell höheren Grad der Korrelation. Allerdings können die Klassifizierungen durch den One2One-Service genügen um eine hinreichend fundierte Entscheidung zu treffen. Zusätzlich kann die Heuristik in den One2One-Service integriert werden, sodass kein weiterer Service entwickelt werden muss, sondern lediglich ein bereits bestehender Service erweitert.

4.2.4 Variante E: Heuristische Auswahl des Verarbeitungsabzweiges

Die Variante E beschreibt eine Erweiterung der Variante D um einen zusätzlichen Durchlauf. Dadurch soll die Möglichkeit geschaffen werden, einen höheren Korrelationsgrad und damit eine Stärkere Reduzierung der Gesamtalarme zu erzielen. Abbildung 4.6 zeigt die kompakte Darstellung der Variante E, wohingegen eine sequentielle Darstellung der

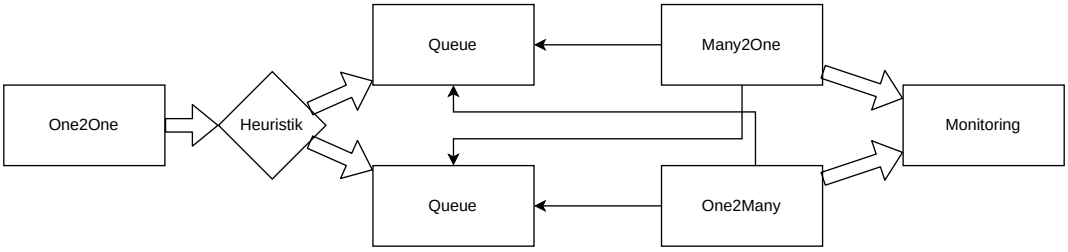

Abbildung 4.6: **Variante E**: Erweiterung der Variante D um einen weiteren Durchlauf im anderen Verarbeitungsabzweig, nachdem der von der Heuristik bestimmte Zweig abgearbeitet wurde

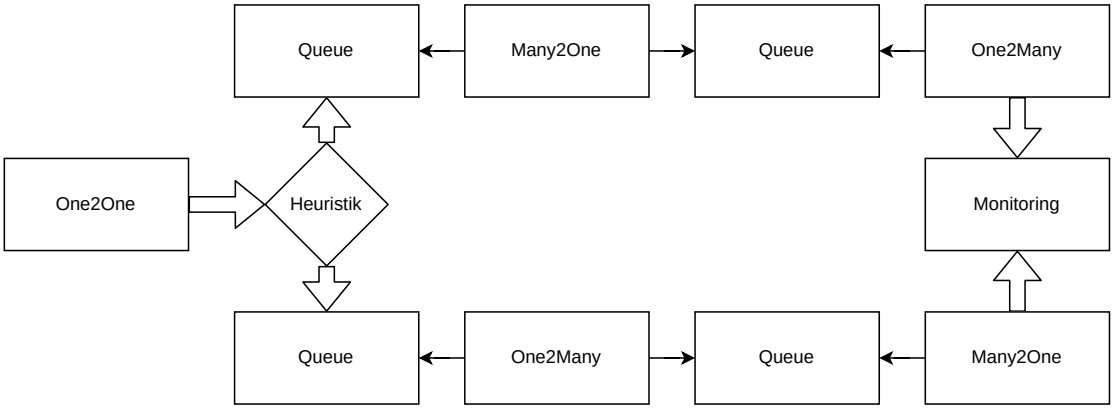

Abbildung 4.7: Sequentielle Darstellung der Variante E zur Verdeutlichung des Ablaufs.

Variante in Abbildung 4.7 enthalten ist. Die sequentielle Variante kann mit dem geschilderten Tooling und Docker-compose leicht umgesetzt werden. Dabei würden dann der One2Many- und der Many2One-Service in zwei verschiedenen Konfigurationen in die Verarbeitungskette integriert werden. Zusätzlich bleibt die Entwicklung eines Heuristik-Services analog zu Variante D aus.

4.2.5 Variante F: Parallele Verarbeitung von One2One-Alerts mit Auswertungskomponente in Monitoring-Services

Variante F löst die angesprochenen Herausforderung der Deduplizierung, dem Korrelationsgrad und der Abwägung des Verarbeitungszweiges auf eine andere Art und Weise als die bisherigen Varianten. Abbildung 4.8 zeigt den Ablauf der Variante F. Dabei werden

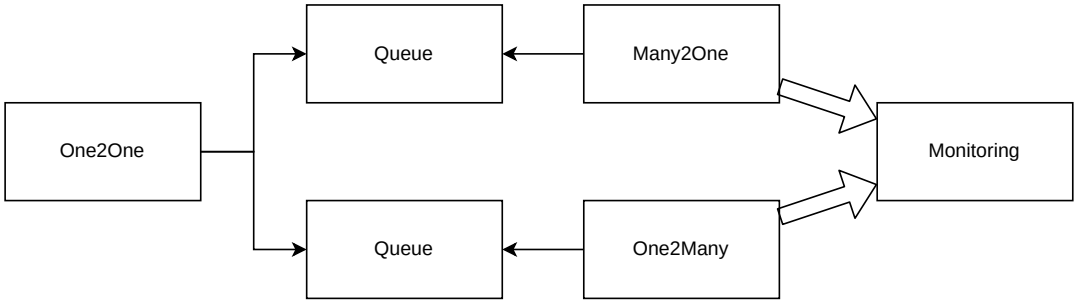

Abbildung 4.8: **Variante F**: Parallele Verarbeitung der One2One-Alerts mit Auslagerung der Priorisierung und Abwägung der Resultate an die Monitoring Services

One2OneAlerts wie in den Varianten C und D parallel verarbeitet. Allerdings werden die beschriebenen Herausforderungen an die Monitoring-Services ausgelagert. Diese Auslagerung würde eine Integration von Graph-Datenbank oder ähnlichen Services beinhalten, welche im Nachhinein Verbindungen zwischen den resultierenden Alarmen herstellen können. Dabei würde die Deduplizierung stattfinden können, indem jeder Basisalarm und jeder darauf aufbauende Alarm zu einer Baumstruktur zusammengefasst wird, in welcher das doppelte Vorkommen von Knoten anhand einer Kante dargestellt wird. So würden auch gleichzeitig die Resultate des One2Many- und Many2One-Services zu einem Ergebnis zusammengefasst werden. Dies würde gleichzeitig die Herausforderungen der Abwägung der Ergebnisse, die Deduplizierung und den Korrelationsgrad betreffen. Zusätzlich würde dieser Ansatz weitere analytische Möglichkeiten unter Verwendung von graphentheoretischen Algorithmen bieten.

4.2.6 Abwägung der Varianten

Die Varianten A und B sind am einfachsten in der Umsetzung. Allerdings sind die Varianten auf einen festen Ablauf und die damit verbundene Reihenfolge der Services beschränkt. Somit kann nicht flexibel auf sich verändernde Anforderungen an das System reagiert werden.
Die im Anschluss geschilderten Varianten C und D lösen die Problematik der Flexibilität indem entweder vor oder nach der Verarbeitung entschieden wird, welcher Service die wahrscheinlich akkurateren Ergebnisse liefert. Allerdings fehlt hier ein weiterer Verarbeitungsschritt, welcher die Möglichkeit bieten würde noch weitere Zusammenfassungen

durch den jeweils anderen Service vorzunehmen.
Aus dieser Erkenntnis wurde die Variante E entwickelt, welche die bereits angesprochenen Problemstellungen löst, indem eine Heuristik über die Weiterleitung an einen Verarbeitungszweig entscheidet, in welchem beide Services enthalten sind. Der Vorteil des Einsatzes einer vorgelagerten Heuristik ist, dass kein zusätzlicher Service entwickelt werden muss, da die Heuristik in den One2One-Service integriert werden kann. Zusätzlich muss durch die Verwendung einer Heuristik keine nachgelagerte Deduplizierung stattfinden, da keine Alarme dupliziert werden sondern distinkt an den jeweiligen Verarbeitungsabzweig weitergeleitet werden.
Auch wenn Variante F in der Theorie vielversprechend ist, würde die Umsetzung zusätzliche Aufwände in der Implementierung der Monitoring-Services mit sich bringen, weshalb von dieser Variante abgesehen wurde. Stattdessen ist die Variante E am besten für die Umsetzung im Rahmen dieser Arbeit geeignet.

4.3 Services

Die in Abschnitt 4.2 hergeleiteten Services lassen sich anhand ihrer Arbeitsweise in drei Kategorien einteilen. Der Simulation-Service ist der einzige Service, welcher ausschließlich Alarme produziert und keine Alarme konsumiert. Zugleich ist dieser Service auch der einzige, der anhand einer Schnittstelle von außerhalb des Systems bedient werden kann. Die zweite Kategorie bilden, welche jeden eingehenden Alarm einzeln bearbeiten, ohne Informationen aus anderen Alarmen zu beziehen. Daher werden die Services dieser Kategorie im Folgenden *One-by-One Services* genannt. Dieses Verhalten ist in Abbildung 4.9 abstrahiert dargestellt. Dabei übernimmt der *Application Core* die zentrale Aufgabe des jeweiligen Services (z. B. die Normalisierung).
In die dritte Kategorie fallen alle Services, die aufgrund ihrer Aufgabe eine Menge von Alarmen betrachten. Dabei wird regelmäßig auf einen Puffer zugegriffen und die Alarme werden anhand bestimmter Kriterien weiterverarbeitet. Dieses Verhalten wird in Abbildung 4.10 dargestellt. Auch in dieser Abbildung handelt es sich bei dem *Application Core* um die Komponente, welche die Hauptaufgabe des jeweiligen Services übernimmt. Aufgrund der beschriebenen Eigenschaften wird die dritte Kategorie im Folgenden *Buffer-based Services* genannt. Im Folgenden werden die Funktionalitäten der implementierten Services dargestellt. Die Implementationssprache des Proof of Concepts ist Java, da mit dieser Sprache bereits ausreichend Erfahrung gesammelt wurde.

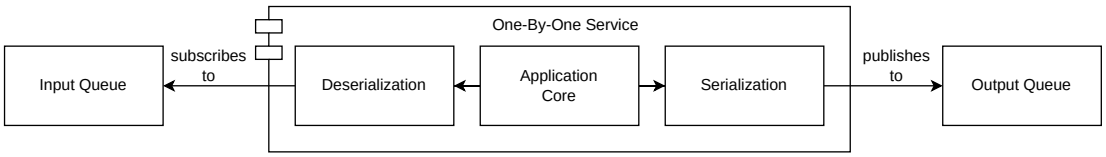

Abbildung 4.9: Abstrakte Darstellung der *One-by-One Services*. Der Übersichtlichkeit halber, wurden die Message-Queue Adapter Klassen in der Darstellung ausgelassen.

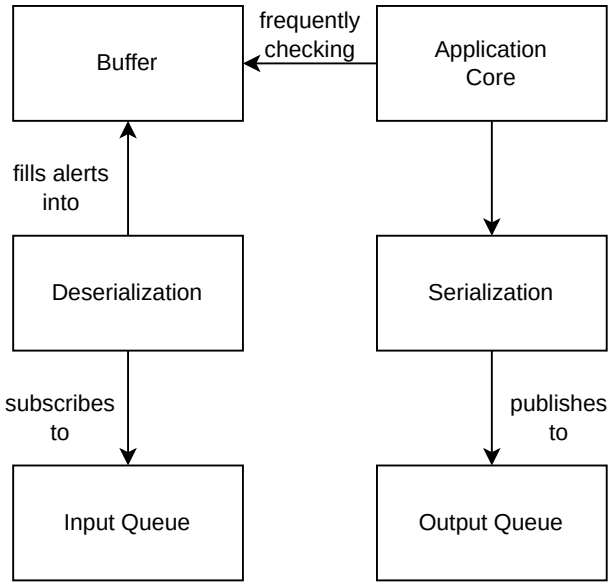

Abbildung 4.10: Abstrakte Darstellung der Services unter Verwendung eines Puffers

4.3.1 Abstrakter Algorithmus der Buffer-based Services

Vorbereitend auf die in Abschnitt 4.3.2 beschriebene Architekturvariante, wird das abstrakte Vorgehen der Buffer-based Services im Vorfeld genauer betrachtet. Der von den Buffer-based Services verwendete Algorithmus, arbeitet anhand einer Mischung aus den in Abschnitt 2.2.1 erläuterten Vorgehen aus den Bereichen *Attribute based* und *Temporal based*. Abbildung 4.10 zeigt den abstrakten Aufbau eines solchen Services. Die Application-Core-Komponente prüft regelmäßig die im Puffer enthaltenen Elemente. Dazu wird eine Kopie des Puffers erzeugt. Anschließend wird für jedes Element im Puffer ein Schlüssel generiert, welcher die Gleichheit der Elemente bestimmen soll. Dies kann anhand der für den Service relevanten Attribute wie z. B. *srcIp*, *dstIp*, *srcPort*, *dstPort* reali-

```
1 for list of map:
2     if list.size >= MAX_ALERTS_TO_WAIT_FOR:
3         doCoreTask(list)
4     else:
5         youngestElement = searchYoungestElement(list)
6         if youngestElement.timestamp.isBefore(timestamp.now().minus(
              TIME_TO_REMAIN_IN_BUFFER)):
7             doCoreTask(list)
```

Listing 4.1: Pseudocode des abstrakten Algorithmus der Buffer-based Services

siert werden. Der so erzeugte Schlüssel ist die Verkettung der relevanten Attribute zu einer Zeichenkette. Diese Zeichenkette kann nun zum Aufbau einer $Map : String \rightarrow [Alarm]$ genutzt werden, sodass jeder Alarm des Puffers in eine Liste mit Alarmen des gleichen Schlüssels geschrieben wird. Im Anschluss wird nun über die Map iteriert und jede enthaltene Liste betrachtet. Die im Folgenden beschriebenen Schritte können anhand des in Listing 4.1 beschriebenen Pseudocodes nachvollzogen werden.
Die Buffer-based Services verfügen über zwei Konstanten (verwendet in Zeile 2 und 6) $MAX\_ALERTS\_TO\_WAIT\_FOR$ und $TIME\_TO\_REMAIN\_IN\_BUFFER$. Erstere legt fest auf wie viele Alarme mit gleichem Schlüssel maximal gewartet werden soll. Dies soll die Ausnutzbarkeit des Systems durch einen Angreifer einschränken, welcher die Konfiguration der einzelnen Services kennt. So wird verhindert, dass der Service stark verlangsamt wird, wenn das System entweder mit vielen gleichartigen Alarmen oder mit gleichartigen Alarmen mit maximaler Verzögerung gefüllt wird. Diese maximale Verzögerung wird durch die zweite Konstante definiert. Sie gibt an, wie lange auf das Eintreffen eines weiteren Alarms der gleichen Liste gewartet werden soll, bevor der Korrelationsprozess des jeweiligen Services greift und die Alarme der Liste korreliert werden. Zeile 5 und 6 des Listings beinhalten die Überprüfung, wie lange der zuletzt eingetroffene Alarm bereits im Puffer liegt. Wird die durch die zweite Konstante definierte Zeit überschritten, wird der Korrelationsschritt des Services auf die Liste angewendet. Der so entstehende Meta-Alarm wird in die Output-Queue des Services und damit gleichzeitig in die Input-Queue des nachfolgenden Services geschrieben. Anschließend werden die in der Liste enthaltenen Elemente aus dem Puffer des Services gelöscht. Wenn alle Listen der Map bearbeitet wurden, ist der Zyklus des Algorithmus beendet und alle korrelierten, bzw. weitergeleiteten Alarme wurden aus dem Puffer entfernt. Im nächsten Zyklus startet der Algorithmus wieder am Anfang und erstellt eine neue Map.

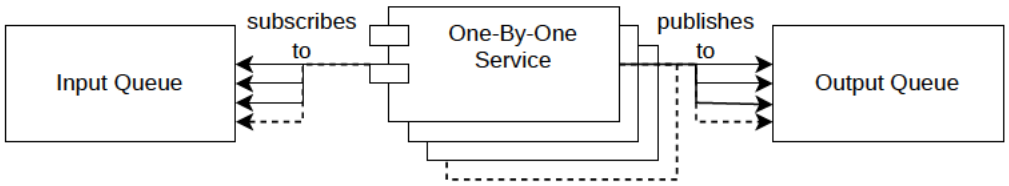

Abbildung 4.11: Horizontale Skalierung der „One-by-One Services“

4.3.2 Skalierbarkeit der Architektur

Die in Abschnitt 3.2.4 gestellten Anforderungen an die Architektur werden anhand der folgenden Maßnahmen realisiert:
Die Services arbeiten jeweils anhand einer dazwischengeschalteten Message-Queue, um den permanenten Nachrichtenfluss zu gewährleisten. Im realisierten Prototypen des Systems sind die im Architekturdiagramm eingezeichneten Message-Queues lediglich Kanäle einer einzelnen RabbitMQ Instanz. Dies kann dazu führen, dass die Instanz der RabbitMQ überlastet wird und eine weitere Skalierbarkeit verhindert. Um dem vorzubeugen, kann entweder auf eine verteilte Lösung von RabbitMQ gesetzt werden oder jeweils eine eigene Instanz der Message-Queue mittels Docker-Container erzeugt werden. Da der Prototyp jedoch als Verbund auf einer virtuellen Maschine ausgeführt wird, wurde von dieser Lösung vorerst abgesehen.
Die vertikale Skalierbarkeit der Services ist durch die Verwendung von Containern gewährleistet, da diese leicht in Cloud-Systemen, wie bspw. dem „Elastic Container Service“ von *Amazon Web Services*, betrieben werden können [57]. Der Normalisierungs- und Preprocessing-Service könnten mit weiteren Instanzen des jeweiligen Services parallel arbeiten, da sie auf keine übergreifenden Zusammenhänge der Alarme angewiesen sind. Diese horizontale Skalierung der *One-by-One Services* ist in Abbildung 4.11 dargestellt. Die *Buffer-based Services* (ab Fusion-Service), müssen auf andere Weise skaliert werden, damit die Qualität der Ergebnisse nicht vermindert wird.
Ein naiver Ansatz wäre die Replikation des Gesamtsystems mit einer Zusammenführung der IDS-spezifischen Datenströme. Diese Orchestrierung ist in Abbildung 4.12 dargestellt. Zu beachten ist der zwischengeschaltete Fusion-Service, welcher duplizierte Alarme herausfiltern soll, da diese im Falle der Trennung nach IDS-Typ nicht übergreifend gewährleistet ist. Somit hat man jedoch wieder einen Flaschenhals in der Verarbeitung, sofern die Reduzierung der Gesamtanzahl von Alarmen durch die jeweiligen Replicas nicht ausreicht.

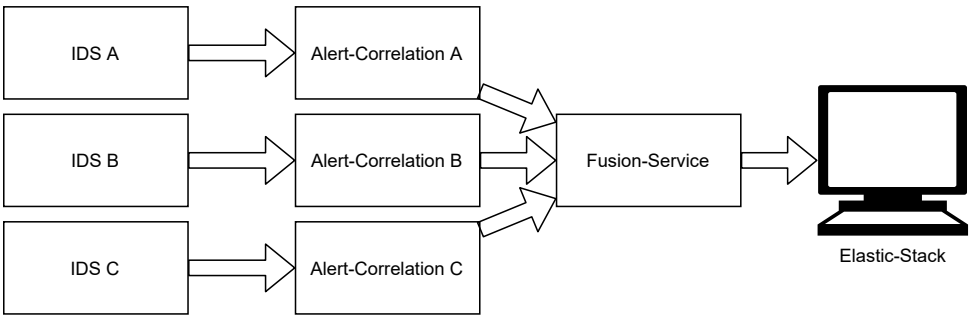

Abbildung 4.12: Replikation des Gesamtsystems mit Zusammenführung der IDS spezifischen Datenströme

Dieser Ansatz wäre auch auf globaler Ebene denkbar, betrachtet man bspw. eine Firma mit verschiedenen Standorten, deren Alarme der IDS-Infrastruktur verarbeitet werden sollen. Allerdings ist in diesem Fall zu beachten, dass die Verarbeitung mit einem lokalen Fokus stattfindet. Dieses Verhalten widerspricht dem Kerngedanken eines SIEM-Systems, da die lokalen Betrachtungen der einzelnen Sensoren zu einem Gesamtbild zusammengeführt werden sollen. Dies ist nur dann möglich, wenn die Alarmkorrelation Zugriff auf alle Alarme aus unterschiedlichen Quellen hat. Wenn die Alarmströme separat betrachtet werden, ist dies nicht gegeben. Im Falle einer Alarmkorrelation über mehrere Standorte hinweg sind die Alarmströme divers, sofern der jeweilige Standort über verschiedene IDS-Sensoren verfügt. Allerdings müsste in diesem Fall ein Weg gefunden werden, die lokal korrelierten Alarme weiter zu verarbeiten, um Informationen über die globalen Geschehnisse zu erlangen.
Um eine horizontale Skalierbarkeit der Buffer-based Services zu gewährleisten wurde eine Architektur ausgearbeitet, welche anhand eines *Coordination-Services* die Verteilung auf verschiedene Services gewährleistet. Diese Architekturvariante wird in Abbildung 4.13 dargestellt. Der abgebildete Coordination-Service ist ebenfalls ein *One-by-One Service*, weshalb dieser ebenfalls horizontal skalierbar ist. Jeder Coordination-Service bildet den Schlüssel für einen Alert auf die gleiche Art wie der nachfolgende Service. Anschließend werden verarbeitete Alarne mittels einer Hash-Funktion über den Schlüssel auf n Queues aufgeteilt. Dabei entspricht n der Anzahl der benötigten Services zum Bearbeiten der aufkommenden Last. Durch die Verwendung einer Hash-Funktion in Kombination mit den erzeugten Schlüsseln wird sichergestellt, dass nach den Kriterien des Buffer-based Services gleichartige Alarme in die gleiche Queue geschrieben werden. Durch dieses Ver-

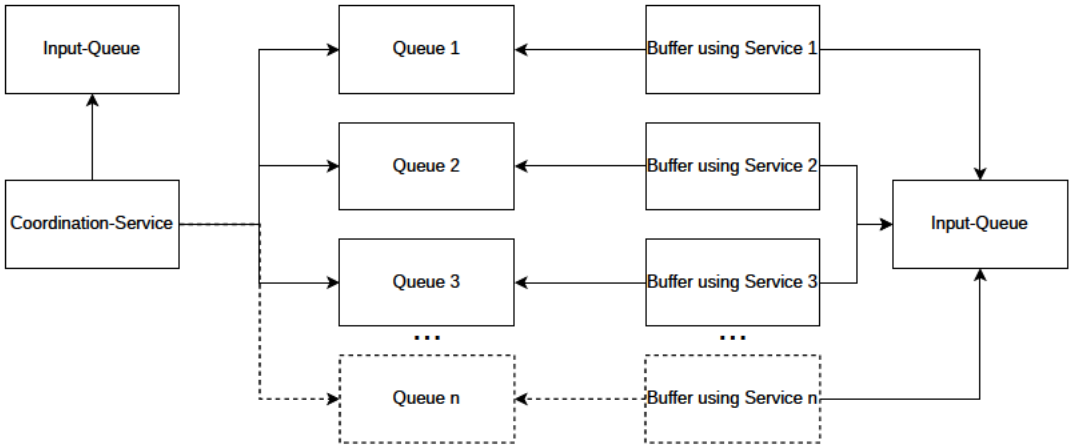

Abbildung 4.13: Horizontale Skalierung der Buffer-based Services

halten wird ermöglicht, dass die Buffer-based Services nach Anpassung der Architektur des Prototypen nicht zusätzlich modifiziert werden müssen.

4.3.3 Simulation

Aufgabe des Simulation-Services ist es, Alarme für ausgewählte Testszenarien zu erzeugen und diese in die Input-Queue des Normalisierungs-Services (siehe 4.3.4) zu schreiben. Dabei wird eine Schnittstelle angeboten, die in Listing 40 angegeben wird. Mittels der Schnittstelle können einzelne Alarme unterschiedlicher Typisierung (Zeilen 2-8), mehrere Alarme gleicher Typisierung (Zeilen 11-19), oder dedizierte Testszenarien (Zeilen 25-35) erzeugt werden. Die Testszenarien beinhalten u. a. Stresstests, welche innerhalb eines anzugebenden Zeitintervalls Testdaten für eine Alarmrate (gemessen in Alarme pro Sekunde (A/s)) erzeugen und versenden (Zeilen 34 und 35). In Zeile 28 wird eine Ressource beschrieben, die als manueller Health-Check verwendet werden kann, um sicherzustellen dass das System antwortet.

In Kapitel 3 wird die Anforderung an das System beschrieben, auch im Kontext einer forensischen Untersuchung eingesetzt werden zu können. Um dies zu realisieren muss lediglich der Simulationsservice dahingehend angepasst werden, dass Datenquellen wie bspw. eine Alarmdatenbank eingebunden wird und die darin enthaltenen Daten in chronologischer Reihenfolge simuliert werden. Für diesen Anwendungsfall existieren bereits Bausteine aus vorangegangenen Projektarbeiten. Unter anderem wurde dort ein Mechanismus zur Simulation von Testdaten erarbeitet, welcher die Abstände zwischen den

```
1  # Einzelne Alerts um die Funktionalitaet des Preprocessing-Services
       zu untersuchen
2  /simulations/preprocessing/connection
3  /simulations/preprocessing/exploit
4  /simulations/preprocessing/malwaredl
5  /simulations/preprocessing/password-guessing
6  /simulations/preprocessing/pingscan
7  /simulations/preprocessing/portscan
8  /simulations/preprocessing/probe
9
10 # Multiple Alerts gleicher Klasse, um das Verhalten des One2One-
       Services zu untersuchen
11 /simulations/one2one/vertical-port-scan
12 /simulations/one2one/DOS
13 /simulations/one2one/multiple-portscans
14 /simulations/one2one/multiple-pingscans
15 /simulations/one2one/multiple-probes
16 /simulations/one2one/multiple-password-guessings
17 /simulations/one2one/multiple-connections
18 /simulations/one2one/multiple-exploits
19 /simulations/one2one/multiple-malware-downloads
20
21 # Testszenarien und einem Noise-Endpunkt fuer nicht aggregierbare
       Alarme
22 /simulations/noise
23
24 # Stresstests
25 /simulations/stress/one2one/unique
26 /simulations/stress/one2one/aggregatable
27 /simulations/stress/one2many/aggregatable
28 /simulations/stress/many2one/aggregatable
29 /simulations/stress/fusion/same-alert
30 /simulations/stress/fusion/max-agg-with-delay
31 /simulations/stress/fusion/max-agg-fixed-rate
32
33 # Stresstests mit stabiler Alarmrate
34 /simulations/rates/fixed/unique
35 /simulations/rates/fixed/max-aggregation
36
37 # Urspruenglich verwendete Schnittstellen (deprecated)
38 /test
39 /simulations
```

Listing 4.2: Die durch den Simulation-Service angebotene Schnittstelle

```
1  public enum FlowType {
2      PORTSCAN,
3      PINGSCAN,
4      PROBE,
5      PASSWORD_GUESSING,
6      EXPLOIT,
7      MALWARE_DOWNLOAD,
8      CONNECTION;
9  ...
10 }
```

Listing 4.3: Durch den Preprocessing-Service vorgenommene Klassifikation des Netzwerkverkehrs

Zeitstempeln der Testdaten untersucht und auf zukünftige Zeitstempel übersetzt. Dadurch kann der Service die Daten ins System einspielen, so wie sie ursprünglich zeitlich aufgetreten sind.[9][10]

4.3.4 Normalisierung

Der Normalisierungs-Service ist dafür zuständig, die in das System gelangenden Alarme in ein einheitliches Format zu bringen (siehe auch 4.4). Dabei greift der Service auf eine Input-Queue zu und arbeitet die dort abgelegten Alarme einzeln ab. Anschließend werden die so verarbeiteten Alarme an die Input-Queue des nächsten Systems weitergeleitet (siehe 4.3.5). In dem implementierten Prototypen wird davon ausgegangen, dass die ins System eingespielten Alarme von dem IDS-System *Suricata* erzeugt werden. Darauf aufsetzend wurde ein Meta-Format entwickelt, welches von den weiteren Services adaptiert wird. Anhand dieses Meta-Formats kann von dem Format der verwendeten Datenquellen (wie bspw. Suricata) abstrahiert werden. Die genaue Ausprägung des Meta-Formats wird in Abschnitt 4.4 erläutert.

4.3.5 Preprocessing

Der Preprocessing-Service reichert die normalisierten Alarme um weitere Attribute an. Beispielsweise wird in diesem Schritt bestimmt um welche Art von Paketen es sich innerhalb der Alarme handelt und im Anschluss eine grobe Klassifikation des Netzwerkverkehrs vorgenommen. In Listing 4.3 wird das Java Enum `FlowType` dargestellt, welches die Klassifikation des Netzwerkverkehrs abbildet. Die Regeln zu diesem Prozess sind an

die Arbeit des BSI [21] angelehnt und können wie folgt beschrieben werden:

- **Portscan:** Wenn maximal fünf Pakete bei der Verbindung ausgetauscht wurden und es sich um die Netzwerkprotokolle TCP oder UDP handelt, wird von einem Portscan ausgegangen.
- **Pingscan:** Wenn maximal fünf Pakete bei der Verbindung ausgetauscht wurden und es sich um das Netzwerkprotokoll ICMP handelt, wird von einem Pingscan ausgegangen.
- **Probe:** Bei einer Anzahl von sechs bis zwölf Paketen einer Verbindung, wird der Verkehr als Probe klassifiziert.
- **Password Guessing:** Wenn der Zielport der Verbindung einer der bekannten Ports für Password-Guessing-Angriffe ist und das Netzwerkprotokoll TCP ist, wird angenommen, dass es sich um einen Password-Guessing-Angriff handelt. Die Ports der Protokolle FTP (21), SSH (22), Telnet (23), POP3 (110), IMAP (143), IMAP verschlüsselt (993) und POP3 verschlüsselt (995) werden in diesem Schritt berücksichtigt.
- **Exploit:** Wenn der vom Initiator der Verbindung versendete Verkehr und der vom Empfänger zurückgesendete Verkehr mehr als 100 Bytes aufweisen, wird davon ausgegangen, dass es sich bei der Verbindung um einen Exploit handelt.
- **Malware Download:** Wenn der Zielport größer als 1024 ist und mehr als 10.000 Bytes in mehr als 100 Paketen ausgetauscht wurden, wird angenommen, dass es sich um den Download von Malware handelt.
- **Connection:** Wenn keine der oben genannten Regeln greift, wird die Verbindung nicht weiter klassifiziert und als *Connection* markiert.

Das resultierende Format des Preprocessing-Services wird in Abschnitt 4.4.4 genauer erläutert.

4.3.6 Fusion

Der Fusion-Service ist ein Buffer-based Service und dient direkt dem in Abschnitt 3 genannten Ziel der Reduzierung der Gesamtanzahl der Alarme. In diesem Service werden

```
1  public enum One2OneClassification {
2      MULTIPLE_PORTSCANS,
3      MULTIPLE_PINGSCANS,
4      MULTIPLE_EXPLOITS,
5      MULTIPLE_PROBES,
6      MULTIPLE_MALWARE_DOWNLOADS,
7      MULTIPLE_PASSWORD_GUESSINGS,
8      MULTIPLE_CONNECTIONS,
9      VERTICAL_PORTSCAN,
10     DOS,
11     ...
12 }
```

Listing 4.4: Durch den One2One-Service vorgenommene Klassifikation der eingehenden Alarme

duplizierte Alarme zu einem gemeinsamen Meta-Alarm zusammengeführt. Für die Deduplizierung werden dabei die folgenden Kriterien betrachtet: Zeitstempel, srcPort, srcIp, dstPort, dstIp. Somit gelten Alarme als *fusionierbar*, wenn ihre Attribute srcPort, srcIp, dstPort, dstIp und der auf 100 Millisekunden gerundete Zeitstempel wertgleich sind.

4.3.7 One2One

Der One2One-Service ist ein Buffer-based Service und implementiert den Korrelationsschritt der *Thread Reconstruction* (siehe 2.2). In diesem Fall werden alle Alarme des Puffers zusammengefasst, welche wertgleich in Hinblick auf die Attribute srcIp und dstIp sind. Die Port-Attribute werden in diesem Prozess nicht berücksichtigt, da bspw. bei einem Port-Scan mehrere Ports in den Alarmen vertreten sind, diese aber dennoch zusammengefasst werden sollen.
Anhand des FlowType-Attributs werden dann weitere Klassifikationen der zusammenfassbaren Alarme vorgenommen. Diese sind in Listing 4.4 dargestellt. Die Klassifikation wird vorgenommen, sofern eine Aggregation stattgefunden hat. Dabei werden die in Abschnitt 4.3.5 beschriebenen Basisklassifikation in die jeweils zutreffende Klassifikation übersetzt. Besonderheiten weisen dabei die Klassifikation `DOS` und `VERTICAL_PORTSCAN` auf. Als Einschätzung eines *Denial of Service* (DoS) Angriffs wurde dabei das Vorkommen eines Ports, innerhalb der aggregierten Alarme, von über fünf mal angenommen. Dies deutet darauf hin, dass ein angreifendes System mehrfach versucht auf den selben Port zuzugreifen. Als *Vertical Portscan* wird das Auftreten von mindestens 10 verschiedenen Ports, innerhalb von als Portscan eingestuften Alarmen, klassifiziert. Die in Abschnitt

4.2 hergeleitete Variante sieht vor, dass anhand einer Heuristik entschieden wird, welcher der beiden nachfolgenden Services (One2Many und Many2One) als erstes durchlaufen wird. Die dazu verwendete Heuristik kann in den One2One-Service eingebaut werden, da dieser durch die vorgenommenen Klassifikationen bereits eine Zuordnung vornehmen kann.
Dazu werden lediglich die Klassifikationen `DOS`, `CONNECTION`, `MALWARE_DOWNLOAD`, `PASSWORD_GUESSING` und `MULTIPLE_PASSWORD_GUESSINGS` an den Verarbeitungsabzweig mit dem vorangestellten Many2One-Service weitergeleitet. Bei diesen Klassifikationen handelt es sich um potentiell verteilt stattfindende Angriffe, weshalb sie zu erst vom Many2One-Service verarbeitet werden sollten, um eine höhere Aggregation zu erzielen.

4.3.8 One2Many

Der One2Many-Service fasst Alarme anhand des *srcIp*-Attributs zusammen. Dabei wird das *dstIps*-Attribut mit der Vereinigung der *dstIp*-Attribute der zusammengefassten Alarme gefüllt. So können Teilangriffe abgebildet werden, welche von einem System ausgehen und viele Systeme adressieren. Dies kann bspw. eine Angriffsvorbereitung wie ein Pingscan sein, der auf das lokale Netzwerk ausgeführt wird, um zu erkennen welche IP-Adressen im Netzwerk vergeben sind. Daran anknüpfend werden dann ggf. weitere Vorbereitungen getroffen, oder gezielt einzelne Systeme attackiert.
Da der One2Many-Service in der Verarbeitungskette auch hinter dem Many2One-Service stehen kann, werden in diesem Schritt alle Alarme mit mehreren eingetragenen *srcIps* ignoriert.

4.3.9 Many2One

Der Many2One-Services korreliert und aggregiert Alarme in gegenläufigen Szenarien zum One2Many-Service. Dabei werden Alarme anhand des *dstIps*-Attributs zusammengefasst. Allerdings können in diesem Verarbeitungsschritt mehrere IP-Adressen im dstIps-Attribut hinterlegt sein, da der Many2One-Service in der Verarbeitungskette des implementierten Prototypens hinter dem One2Many-Service stehen kann (siehe Abbildung 4.7). Damit der Many2One-Service korrekt arbeiten kann, müssen dementsprechend Alarme mit mehr als einer eingetragenen IP-Adressen im dstIps-Attribut herausgefiltert werden.

4.4 Datenstrukturen

Die in Abschnitt 4.3 beschriebenen Services arbeiten jeweils mit Input- und Output-Formaten. Diese werden im Folgenden genauer beschrieben. Zusätzlich wird auf das Metaformat eingegangen, welches über die verschiedenen Services hinweg angereichert wird.

4.4.1 Suricata Message

Der Simulation-Service besitzt als einziger Service kein Input-Format, da dieser Service ausschließlich als Produzent von Alarmen agiert. Diese werden an die Input-Queue des Normalisierungs-Services übergeben. In dieser Arbeit wurde angenommen, dass vorerst nur Nachrichten im Suricata-Output-Format verarbeitet werden. Dabei handelt es sich um das *EVE-JSON-Format* [56]. Die so erzeugten Testdaten werden durch die Klasse `SuricataMessage` abgebildet. Das Besondere an diesem Format ist, dass das Basisformat von Suricata-Messages durch Subformate erweiterbar ist. Diese müssen allerdings nicht zwangsläufig in allen verarbeiteten Nachrichten present sein. In dieser Arbeit wurde davon ausgegangen, dass das Subformat *flow* immer vorhanden ist, da dort essentiell wichtige Informationen zu Netzwerkverbindungen und Datenflüssen enthalten sind. Das `SuricataFlowSubFormat` ist die Repräsentation eines bidirektionalen Netzwerk-Flows welche unter anderem Informationen darüber enthält wie viele Pakete vom Client an den Server übertragen wurden oder andersherum. Hierbei sind die im Outputformat verwendeten Terminologien „Server“ und „Client“ so zu verstehen, dass der Client der Initiator der Verbindung ist und der Server dessen Gegenspieler. Das resultierende Format kann Listing 4.5 entnommen werden.

4.4.2 Metaformat Message

Die abstrakte Klasse `MetaFormatMessage` bündelt Informationen die unabhängig vom eigentlichen Verarbeitungsprozess des jeweiligen Services sind. Unter anderem Meta-Informationen darüber wann der Alarm vom Service generiert wurde, also zu welchem Zeitpunkt die Verarbeitung durch den Service abgeschlossen wurde. An dieser Stelle werden auch weitere Felder implementiert die auf die Messbarkeit des Systems einzahlen. Bspw. ein Zeitstempel welcher beschreibt, wann der Alarm bei dem Service eingegangen

```
1  public class SuricataMessage extends TestDataEntity {
2  public Instant timestamp;
3  public String eventType;
4  public String srcIp;
5  public Integer srcPort;
6  public String dstIp;
7  public Integer dstPort;
8  public String proto;
9
10 public SuricataAlertSubFormat alert;
11
12 public SuricataTLSSubFormat tls;
13
14 public SuricataHTTPSubFormat http;
15
16 public SuricataDNSSubFormat dns;
17
18 // bi-direktional
19 public SuricataFlowSubFormat flow;
20 ...
21 }
```

Listing 4.5: Gekürzte Darstellung der SuricataMessage Klasse

ist, um die Verarbeitungszeit innerhalb des jeweiligen Services zu betrachten. Jede Klasse, die einen Input oder Output Alarm eines Services darstellt erbt von dieser abstrakten Klasse. Die Werte der Attribute werden im nächsten Verarbeitungsschritt übernommen und um weitere Attribute ergänzt. In dem Schritt, in welchem dann ein neuer Output-Alarm erzeugt und versendet wird, wird die vergangene Zeit seit dem Eingangszeitsempel ermittelt und als weiteres Meta-Format Attribut hinzugefügt. Dieser Wert kann zum Ermitteln von Verarbeitungsengpässen verwendet werden.

4.4.3 Normalized Alert

Bei diesem Format handelt es sich um das Output-Format des Normalisierungs-Services. Dieser legt die Grundlage für die Weiterverarbeitung innerhalb der anderen Services, da der Service die akzeptierten Sensorformate bündelt und eingehende Alarme vereinheitlicht. Im implementierten Prototypen wurde darauf verzichtet eine Abstraktion von dem einzig unterstützen Format vorzunehmen. Daher wird in dem Normalisierungs-Service das in Abschnitt 4.4.1 erläuterte Suricata-Message-Format in Kombination mit der Klasse `MetaFormatMessage` als Output-Format genutzt. Wie in Abschnitt 4.4.2 erläutert,

```
1  public class FusedAlert extends MetaFormatMessage {
2      public Instant timestamp;
3      public String srcIp;
4      public Integer srcPort;
5      public String dstIp;
6      public Integer dstPort;
7
8      List<SuricataMessage> fusedSuricataMessages;
9      ...
10 }
```

Listing 4.6: Gekürzte Darstellung der FusedAlert Klasse

werden in diesem Service zusätzliche Attribute eingeführt. Der Zeitstemppel *normalizedAt* beschreibt den Zeitpunkt der Erzeugung der `MetaFormatMessage` und das Attribut *timeInNormalization* beschreibt die während der Verarbeitung innerhalb des Services vergangene Zeit in Nanosekunden.

4.4.4 Preprocessed Alert

Der Preprocessing-Service nutzt die MetaFormatMessage-Abstraktion um den Netzwerkverkehr zu klassifizieren und an den Fusion-Service zu übergeben. Dazu wird das Feld *flowType* zur abstrakten Klasse `MetaFormatMessage` hinzugefügt und gefüllt. Die Werte des zugehörigen Enums können dem Abschnitt 4.3.5 und dem Listing 4.3 entnommen werden. Zusätzlich werden analog zu zum Normalized-Alert-Format die Felder *preprocessedAt* und *timeInPreprocessing* angelegt und befüllt.

4.4.5 Fused Alert

Das Fused-Alert-Format vereint Alarme des Preprocessed-Alert-Formats. Dabei werden, wie in Abschnitt 4.3.6 erläutert, Alarme, die auf die gleiche, potentiell schadhafte Aktion zurückzuführen sind, zu einem Meta-Alarm zusammengefasst. Das resultierende Format besteht somit aus den in der `MetaFormatMessage`-Klasse definierten Feldern, sowie einer Liste aller zusammengefassten Alarme. Auch in diesem Fall werden ein Zeitstempelattribut und ein Attribut zur Beschreibung der Verweilzeit innerhalb des Services hinzugefügt. Letzteres wird in Sekunden gemessen. Die resultierende `FusedAlert`-Klasse kann Listing 4.6 entnommen werden.

```
1  public class One2OneAlert extends MetaFormatMessage {
2      public Instant timestamp;
3      public String srcIp;
4      public Set<Integer> srcPorts;
5      public String dstIp;
6      public Set<Integer> dstPorts;
7
8      List<MetaFormatMessage> aggregatedAlerts;
9
10     One2OneClassification classification;
11
12     ...
13 }
```

Listing 4.7: Gekürzte Darstellung der One2OneAlert Klasse

4.4.6 One2One Alert

Das One2One-Alert-Format ähnelt dem Fused-Alert-Format (siehe 4.4.5). Der Unterschied ist, dass die Port-Attribute (srcPorts und dstPorts) Listen sind. Dies ist damit zu begründen, dass im Falle einer One2One-Aggregation viele Aktionen von einer IP-Adresse mit einer Ziel-IP-Adresse ausgeführt werden. Im Fall eines verticalen Port-Scans kann dies bedeuten dass die IP-Adresse des Angreifers viele verschiedene Zielports (dstPorts) über variierende Ausgangsports anspricht. Das resultierende Formalt kann Listing 4.7 entnommen werden. Neben der Änderung, dass nun Listen von Ports in den Portfeldern hinterlegt werden können, erbt auch dieses Format von der abstrakten Klasse MetaFormatMessage und erweitert diese um ein Zeitstempel-Attribut zur Beschreibung des Zeitpunkts der Verarbeitung und ein Attribut zur Beschreibung der Verarbeitungsdauer in Sekunden.

4.4.7 One2Many Alert

Das One2Many-Alert-Format unterscheidet sich von dem One2One-Alert-Format darin, dass in der *aggregatedAlerts*-Liste entweder One2OneAlerts oder Many2OneAlerts enthalten sein können (siehe 4.7). Um diese beiden Konfigurationen abzudecken, sind in diesem Format zusätzlich zu den Port-Attributen auch die IP-Attribute vom Datentyp Set. Dies verhindert, wie auch im Beispiel der One2One-Alerts bezogen auf die Port-Attribute, das doppelte Abspeichern von Werten. Eine gekürzte Version der One2ManyAlert-Klasse ist in Listing 4.8 dargestellt.

```
1  public class One2ManyAlert extends MetaFormatMessage {
2      public Instant timestamp;
3      public Set<String> srcIps;
4      public Set<Integer> srcPorts;
5      public Set<String> dstIps;
6      public Set<Integer> dstPorts;
7
8      List<MetaFormatMessage> aggregatedAlerts;
9      ...
10 }
```

Listing 4.8: Gekürzte Darstellung der One2ManyAlert Klasse

4.4.8 Many2One Alert

Das Many2One-Alert-Format ist das Gegenstück zum One2Many-Alert-Format. Der Unterschied liegt lediglich in der Erzeugung dieser Alarme. Dort wird auf das jeweilige Input-Alarm-Format reagiert. Da Listing 4.8 bereits die Attribute und Datentypen abbildet, wurde auf ein weiteres gekürztes Listing verzichtet.

4.5 Konzeption eines Frühwarnungsmechanismus

Da die Verarbeitungszeit je nach Konfiguration der einzelnen Services einige Minuten dauern kann, sollte ein Mechanismus existieren um bei bestimmten Events eine Frühwarnung abzugeben, welcher die Korrelationskette zwar nicht übergeht, aber dennoch eine Benachrichtigung vor vollständiger Verarbeitung versendet. Die Implementation eines solchen Mechanismus würde die Zeit bis zur Alarmierung stark reduzieren und könnte in Abhängigkeit zu den jeweiligen Services umgesetzt werden. Im Folgenden wird ein Beispielanwendungsfall erläutert und anschließend auf eine mögliche Architektur eines Alarmierungsmechanismus eingegangen. Abbildung 4.14 zeigt eine beispielhafte Implementation eines Alarmierungsmechanismus, welcher sich in die bisherige Architektur einfügt. Dabei würde jeder qualifizierte Buffer-based Service eine Überprüfung implementieren, welche bei Erfolg ein Zwischenergebnis in eine Alarmierungs-Queue schreibt. Da die Alarme der bisherigen Architektur nicht anhand von IDs gespeichert werden, müssten alle relevanten Informationen in dieser Benachrichtigung enthalten sein. Darunter fällt das Fünftupel aus *timestamp*, *srcIp*, *dstIp*, *srcPorts* und *dstPorts* welches bereits in

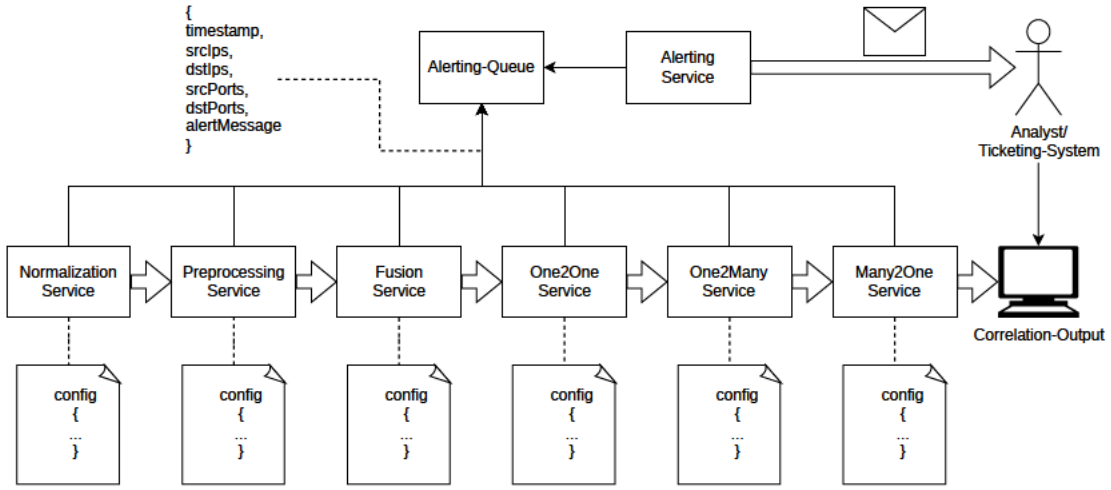

Abbildung 4.14: Entwurf eines beispielhaften Alarmierungsmechanismus. Die Messagequeues zwischen den Korrelationsservices wurden der Übersichtlichkeit halber nicht eingezeichnet.

der MetaFormatMessage-Klasse enthalten ist. Zusätzlich sollte der Grund für die Alarmierung angegeben werden. Dies könnte bspw. die Überschreitung eines Schwellenwertes sein, welcher besagt dass in einem konfigurierten Zeitfenster mehrfach das Maximum von zu aggregierbaren Alarmen erreicht wurde. Die Konfigurationsdateien könnten bspw. im yaml oder JSON Format repräsentiert werden und den Namen der Alarmierungsregel, die Nachricht, den Schwellenwert oder andere Kriterien enthalten.
In einer weiteren Ausbaustufe sollte pro Service eine Konfigurationsdatei existieren, welche gewünschte Schwellenwerte und die damit einhergehende Benachrichtigung anpassbar macht, ohne dass das System neu kompiliert werden muss. Im besten Fall sollte das System sogar dynamisch mit Änderungen umgehen können, auch wenn dies ggf. einen größeren Aufwand bedeuten kann, da dann nicht mehr nur zur Startzeit des Systems einmalig die Konfiguration ausgelesen werden müsste, sondern regelmäßig wenn Änderungen vorgenommen werden.
Beim Betrieben eines solchen Alarmierungsmechanismus, bzw. eines solchen Frühwarnsystems sollte beachtet werden, keine doppelte Alarmierung zu erzeugen. Dazu ist es notwendig eine Beziehung zwischen den Meta-Alarmen, welche die Korrelationskette vollständig durchlaufen haben und den Frühwarnungen herzustellen. Letztere müssen demnach in einem persistenten Speicher hinterlegt werden, welcher bei Bedarf durchsucht werden kann. Eine Möglichkeit wäre dann die Speicherung des Namens des alarmierenden Services, sowie des erzeugten Schlüssels in dem Frühwarnungsformat. Des Weiteren

müsste das Meta-Format dahingehend angepasst werden, dass jeder durchlaufene Service den von ihm erzeugten Schlüssel darin abspeichert. Anschließend müsste dann vor dem Weiterleiten an zuständige Analysten überprüft werden, ob es eine Frühwarnung mit einem der gespeicherten Schlüssel gab. Wenn dies der Fall ist, müsste ein Verweis auf diese Frühwarnung erstellt werden und eine weitere Alarmierung ausbleiben. Der Verweis könnte anhand des Schlüssels als Identifier geschehen, da dieser als fachliche ID ausreicht. Dies setzt allerdings voraus, dass der Zeitstempel der Generierung des Alarms in allen Services Bestandteil der Schlüsselerzeugung ist. Andernfalls wäre es notwendig bei der Suche nach dem Schlüssel im Frühwarnungsspeicher einen Zeitraum festlegen zu können in welchem die Alarmierung stattgefunden haben muss. Dieses Zeitfenster sollte der maximalen Verarbeitungszeit des Systems entsprechen, damit der Zeitraum möglichst wenig Spielraum lässt, sodass Alarmierungen nicht ungewollt ausbleiben. Um diesen Effekt weiterhin abzumindern wäre es eine Art Benachrichtigung mit niedrigerer Priorität vorstellbar, welche über neu eingetroffene Informationen zu einer Frühwarnung informiert. Diese Benachrichtigung ist auch unabhängig von der Vermeidung von ausbleibenden Alarmierungen sinnvoll.

4.6 Iterative Optimierungen des Prototypen

Nachdem eine erste lauffähige Version des Systems existierte wurden einige Evaluationen durchgeführt, die Optimierungen an verschiedenen Stellen nach sich zogen. Dabei handelt es sich neben Optimierungen der Performanz auch um Anpassungen der Architektur bzw. Klassenstrukturum eine bessere Lesbarkeit, Erweiterbarkeit und Testbarkeit des Sourcecodes zu gewährleisten.

4.6.1 Entfernen von nicht benötigten Abhängigkeiten

Im ersten Prototypen des Systems wurde auf der Struktur des Simulationsservices aufgesetzt, welcher unter Verwendung des SpringBoot Frameworks viele externe Abhängigkeiten hat. Diese machen das Entwickeln eines Microservices mit Datenbanken und REST-Schnittstellen komfortabel, da ein großer Teil des Einrichtungsaufwands vom Framework übernommen wird. Allerdings ist bei genauerer Betrachtung klar geworden, dass ein Großteil der implementierten Services keine REST-Schnittstelle anbieten sollten und

auch keine Datenbank benötigen. Somit konnten die externen Abhängigkeiten deutlich reduziert werden. Dies spielt insbesondere für die Größe des kompilierten Java-Programms eine Rolle, sowie für die Startgeschwindigkeit des Gesamtsystems. So konnte die Startzeit, bis das System gänzlich hochgefahren wurde, von ca. 30 Sekunden auf ca. 5 Sekunden reduziert werden.

4.6.2 Implementierung von Multithreading in One-by-One Services

Nach anfänglichen Lasttests ist aufgefallen, dass sich Alarme in der Eingangs-Queue stauen. Dies war duch eine zu langsame Abarbeitung durch den Normalisierungsservice zu erklären. Dieser wurde in diesem Zuge dahingehend restrukturiert, dass im Einstiegspunkt des Programms zwei Threads gestartet werden. Beide Threads verwenden die selbe Input- und Output-Queue. Die lokale Skalierung der Services ist wie in Sektion 4.3.2 beschrieben möglich, da es sich bei den beiden betrachteten Services um One-by-One Services handelt, die keinen Kontext über eine Menge von Alarmen bilden, sondern ausschließlich einzelne Alarme betrachten.
Nach Anpassung des Normalisierungsservices stauten sich bei weiteren Messungen die Alarme in der Output-Queue des Services. Da die Output-Queue die Input-Queue des Preprocessing-Services ist ließ dies auf eine zu langsame Verarbeitung durch diesen schließen. So wurde dann ebenfalls der Preprocessing-Service auf zwei Threads umgestellt.

4.6.3 Erweiterung der Buffer-based Services um Consumer-Threads

Nach der in Abschnitt 4.6.2 genannten Anpassungen stauten sich in weiteren Lasttests Alarme in dem Fusionsservice. Dies ist damit zu erklären, dass durch das Umstellen auf Multithreading innerhalb der zuarbeitenden Services eine deutlich höhere Input-Rate für den Fusionsservice entstanden ist. Um das Zwischenspeichern in der Messagequeue zu reduzieren und unter der Prämisse dass der Fusionsservice eine höhere Last verarbeiten kann wurde ein Konzept zum schnelleren Abrufen aus der Messagequeue entwickelt. Dazu wurde der verwendete Puffer in eine eigene Klasse ausgelagert. Diese `Synchronized-Buffer`-Klasse funktioniert als Wrapper-Klasse einer Liste, um synchronisierten Zugriff verschiedener Threads zu gewährleisten. Anschließend wurden alle Operationen die das Einlesen der Nachrichten aus der Input-Queue betrafen in eine eigene Klasse `MQConsumer` ausgelagert. Diese Änderung ermöglichte das Einführen eines eigenen Threads,

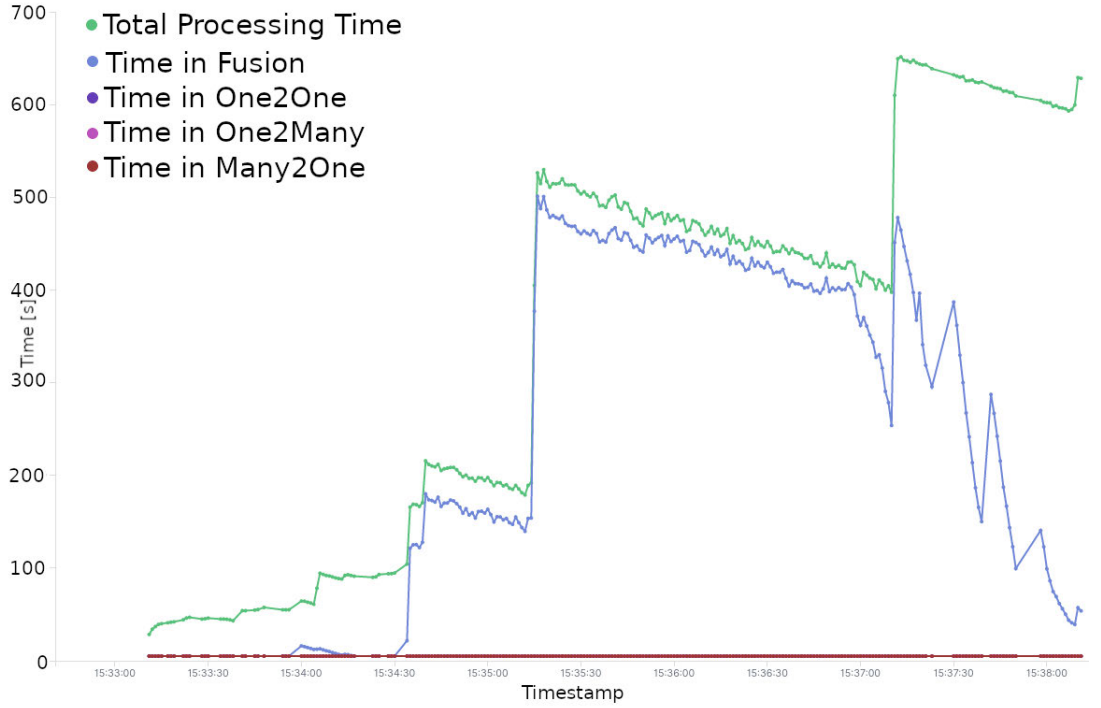

Abbildung 4.15: Abbildung der Messung eines Lasttest des Gesamtsystems, welcher einen Flaschenhals in der Verarbeitung innerhalb des Fusion-Services offenbart. Es wurden in einem Zeitintervall von 5 min 223876 einzgartige Alarme ins System eingespielt.

dessen einzige Aufgabe das Auslesen von Alarmen aus der Messagequeue und das Schreiben in den `SynchronizedBuffer` ist. Somit konnte der Schritt des Einlesens aus der Messagequeue parallelisiert werden und bei Bedarf um weitere Threads erweitert werden.

4.6.4 Entkopplung des Versendens von Alarmen

Nach der Einführung von zwei *MQConsumer*-Threads stauten sich keine Nachrichten mehr in der Messagequeue. Allerdings führte dies dazu, dass der `FusionHandler` und damit der eigentliche Algorithmus größere Alarmmengen verarbeiten musste, da diese nun nicht mehr durch die RabbitMQ gepuffert wurden, sondern im internen Puffer abgespeichert wurden. Abbildung 4.15 zeigt die Verarbeitungszeiten der Buffer-based Services sowie die Gesamtverarbeitungszeit pro Alarm. Hierbei lässt sich eine „Treppenfunktion“

bei der Verarbeitungszeit des Fusion-Services erkennen. Dies lässt darauf schließen, dass die Verarbeitung der empfangenen Alarme einen geringeren Durchsatz aufweist als die Rate der eingehenden Alarme. Deshalb wurde der verwendete Algorithmus, sowie der Programmablauf genauer betrachtet. Dabei fiel auf, dass am Ende des Durchlaufs jeder Verarbeitungsschleife Nachrichten an die Messagequeue gesendet wurden. Da dies allerdings eine Netzwerkoperation ist, verlangsamte dies den Ablauf des Programms deutlich. Um diesem Umstand zu begegnen wurde ein weiterer Puffer der Klasse *SynchronizedBuffer* eingeführt. Dieser sollte dazu beitragen den Versendungsschritt an die Messagequeue zu entkoppeln und zu parallelisieren. Zusätzlich wurde jeglicher Code, der mit dem Versenden von Alarmen an die Message Queue zusammenhängt, in eine Neue Klasse namens *MQProducer* verschoben. Diese Klasse implemiert das Runnable-Interface und kann somit in einem Thread ausgeführt werden. Durch diese Änderung wurde der Arbeitsschritt des Versendens und damit auch eine weitere Netzwerkoperation aus dem eigentlichen Algorithmus entfernt. Abbildung 4.16 zeigt die Laufzeiten der Puffer-Services nach der Umsetzung der *MQProducer*-Klasse. Auch in diesem Fall können analog zum in Abschnitt 4.6.3 erläuterten Vorgehen bei Bedarf weitere MQProducer-Threads gestartet werden, sofern die Leistungsfähigkeit des eigentlichen Algorithmus ausreichend ist um den Output-Puffer hinreichend zu füllen.
Abbildung 4.17 zeigt die in diesem und in Abschnitt 4.6.3 beschriebenen Änderungen anhand eines abstrakten, Buffer-based Services. Dabei beinhaltet die `MQAdapter`-Klasse zwei Verbindungen zur RabbitMQ. Eine Verbindung für die MQConsumer-Threads und eine Verbindung für die MQProducer-Threads. Diese Separation verhindert, dass sich die unterschiedlichen Thread-Arten blockieren, falls eine Operation zu einer zeitlichen Verzögerung führen sollte. Falls noch weitere Instanzen des MQAdapters benötigt werden sollten, kann dies einfach per Dependency-Injection umgesetzt werden. Dazu müssen nur weitere Instanzen des MQAdapters erzeugt werden und anschließend an die gewünschten Threads überreicht werden, sodass bspw. Paare aus Consumer- und Producer-Threads entstehen. So würden sich die unterschiedlichen Threads der gleichen Thread-Art keine Verbindung mehr teilen, da genau ein Consumer-Thread die bestehende *consumerConnection* verwendet und genau ein Producer-Thread die *producerConnection* verwendet. Somit würde eine noch stärkere Unabhängigkeit der Threads realisiert werden.

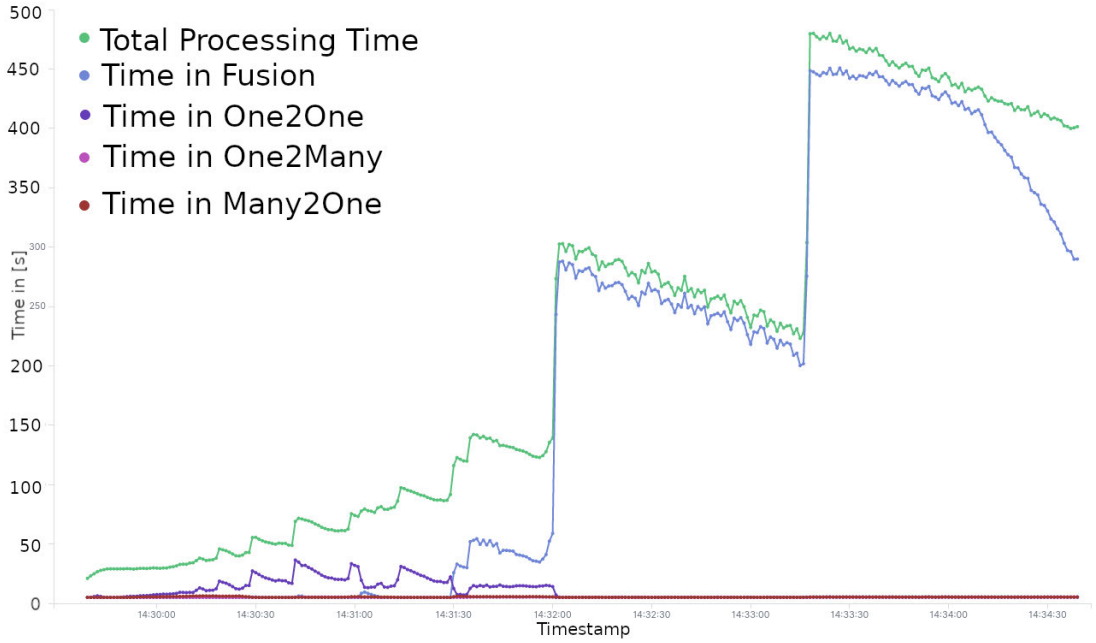

Abbildung 4.16: Abbildung der Messung nach Einführung von MQConsumer und MQProducer Threads.

4.6.5 Proof of Concept: Preprocessing Hash-Funktion

Die in Abschnitt 4.3.2 erläuterte Erweiterung der Architektur um einen Koordinator-Service vor den jeweiligen Buffer-based Services lässt sich in einem Proof of Concept auch bei dem Betrieb der Services auf einer einzelnen Maschine umsetzen. Dabei wurde innerhalb des Fusion-Services ein weiterer FusionHandler-Thread gestartet. Dieser hatte einen eigenen Input-Buffer, sodass beide Threads unabhängig voneinander Nachrichten annehmen und verarbeiten können. Die MQConsumer-Klasse hat als Parameter eine Liste von Inputbuffern erhalten, welche mit einem Index gekennzeichnet wurden. Des Weiteren wurde ein zusätzliches Feld zur abstrakten Klasse `MetaformatMessage` hinzugefügt, welches den Index des Input-Puffers enthält. Dieses Feld musste durch den Preprocessing-Service befüllt werden. Um dies zu realisieren wurde die `generateKey()`-Methode aus dem Fusionsservice kopiert. Anschließend wurden die resultierenden Werte mittels einer Hash-Funktion auf die Anzahl von Input-Puffern abgebildet. Dieser Proof of Concept ermöglicht es bei einem lokalen Deployment ebenfalls die Anzahl von Verarbeitungs-Threads losgelöst von MQConsumer- oder MQProducer-Threads zu erhöhen und somit einen erhöhten Durchsatz zu ermöglichen.

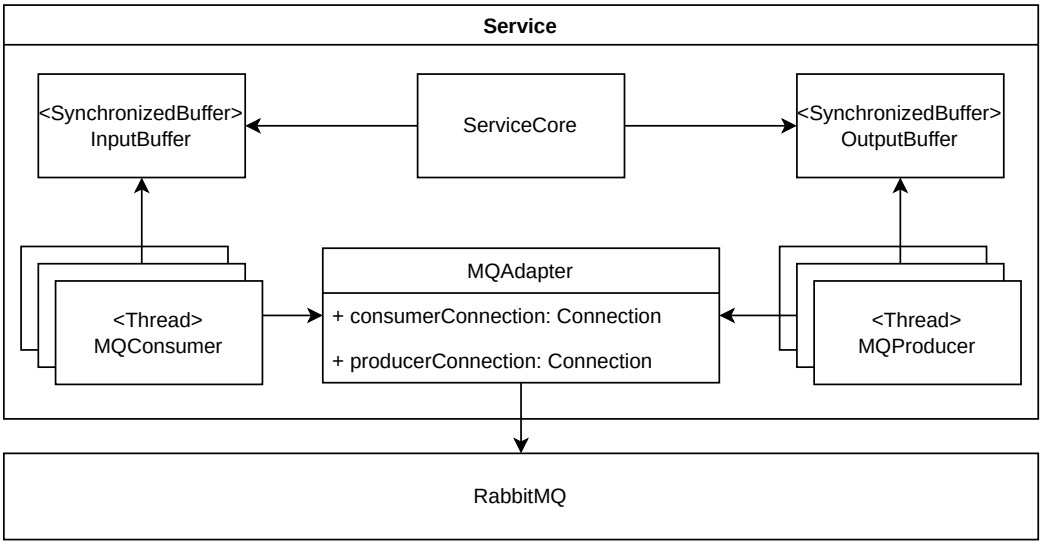

Abbildung 4.17: Abstrakte Darstellung des Zusammenwirkens der Service-Komponenten nach Einführung zweier synchronisierter Puffer und dem Auslagern des Empfangs und Versandes von Alarmen in eigene Threads. Der Übersichtlichkeit halber wurde die RabbitMQ als eigene Komponente ohne Unterscheidung der darin definierten Queues dargestellt.

Abbildung 4.18 zeigt die Verarbeitungszeiten der Puffer-basierten Services nach Einführung der Hash-Funktion. Dabei ist in dem betrachteten Zeitinterval eine starke Senkung der Verarbeitungszeit in dem Fusion-Service zu beobachten. So lag der höchste Wert der Verarbeitungszeit in Abbildung 4.16 bei etwa 450 Sekunden, in Abbildung 4.18 hingegen lediglich bei 250 Sekunden. Ungeklärt bleibt bei dieser Darstellung allerdings, weshalb die Gesamtlaufzeit so weit von der Laufzeit des Fusionsservices ist, während keine weiteren Serviceaktivitäten in der Abbildung auftauchen. Dies ist nur durch das Auftreten einer Verzögerung innerhalb der RabbitMQ oder in einem der nicht in der abgebildeten Services zu erklären.
Die Hash-Funktion wurde im weiteren Verlauf der Entwicklung des Systems wieder entfernt, da diese Aufgabe später von einem Koordinator-Service übernommen werden sollte (siehe Abschnitt 4.6.11).

4.6.6 Verwendung der Java Streaming API

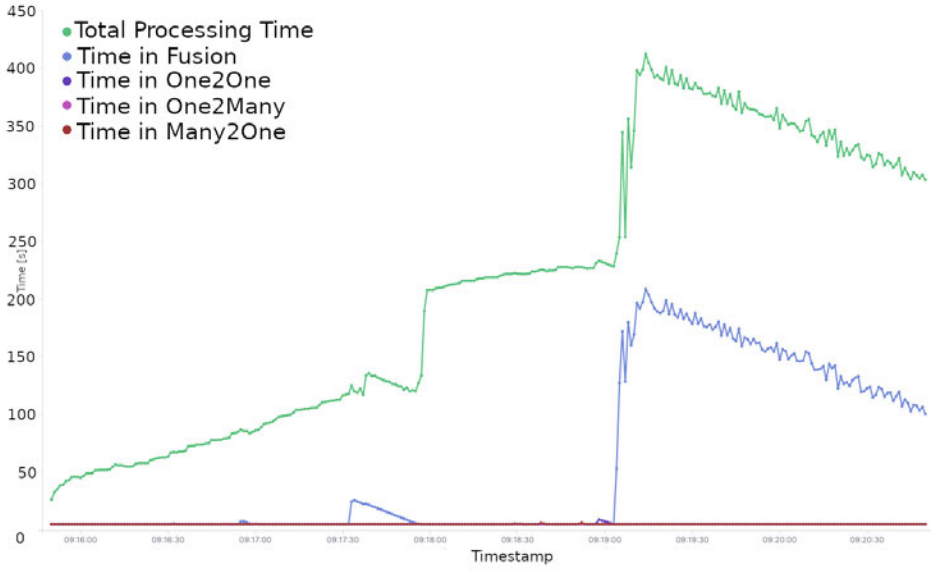

Abbildung 4.18: Ergebnisse eines Lasttests nach Einführung der prototypischen Hash-Funktion. In einem Zeitfenster von 5 min wurden 239467 einzigartige Alarme in das System eingespielt.

```
1  private HashMap<String, List<MetaFormatMessage>> generateCandidateMap(List<
       MetaFormatMessage> bufferedMessages) {
2          var candidateMap = new HashMap<String, List<MetaFormatMessage>>();
3          for (var bufferedMessage : bufferedMessages) {
4              var key = createKey(bufferedMessage);
5              if (!candidateMap.containsKey(key)) {
6                  candidateMap.put(key, new ArrayList<>());
7              }
8              var foundList = candidateMap.get(key);
9              foundList.add(bufferedMessage);
10             candidateMap.put(key, foundList);
11         }
12         return candidateMap;
13 }
```

Listing 4.9: Erzeugung der CandidateMap vor der Verwendung der Java Streaming API

```
1  private Map<String, List<MetaFormatMessage>> generateCandidateMap(
       List<MetaFormatMessage> bufferedMessages) {
2      return bufferedMessages.parallelStream().collect(Collectors.
           groupingBy(this::createKey));
3  }
```

Listing 4.10: Erzeugung der CandidateMap unter Verwendung der Java Streaming API

Die Verwendung der Java Streaming API konnte bei allen Buffer-based Services zur Steigerung der Übersichtlichkeit eingesetzt werden. Zusätzlich bietet die Streaming API eine Möglichkeit an parallelisiert über Listen zu iterieren, welches eine weitere Optimierung der Laufzeit nach sich ziehen kann, sofern genug Ressourcen zur Verfügung stehen um weitere Threads zu verwenden. Listing 4.9 zeigt die Methode „generateCandidateMap“ohne die Verwendung der Streaming-API. Diese Darstellung macht die Komplexitätsanalyse einfacher, da die einzelnen Schritte der Generierung der CandidateMap explizit durch den Sourcecode beschrieben werden. Allerdings muss sich der Leser des Codes Schritt für Schritt ein Gesamtbild schaffen. Wohingegen die in Listing 4.10 beschriebene Variante unter Verwendung der Streaming-API die Anzahl der Lines of Code verringert und gleichzeitig die Lesbarkeit steigert. Die resultierende Zeile zwei bildet die gleiche Funktionalität wie die Zeilen zwei bis zwölf aus Listing 4.9 ab.

4.6.7 Verschiebung der Queue-Deklaration und Entfernen von debug-level Log-Nachrichten

Im ersten Schritt der Umsetzung wurde statt einem initialen Erzeugen einer Message-Queue bei jeder Veröffentlichung einer Nachricht die Methode „queueDeclare()“ im RabbitMQ-Adapter aufgerufen. Da diese Funktion idempotent arbeitet, wurde zwar keine Queue überschrieben, dennoch beanspruchte dieser Aufruf Zeit, da im Hintergrund Kommunikation auf Netzwerkebene stattfinden musste. Die Verschiebung dieses Aufrufs sorgte für eine enorme Durchsatzsteigerung etwa in dem Simulationsservice, welcher nun in der Lage ist Alarmraten von über 1000 Alarmen die Sekunde in das System einzuspielen. In einem Zeitintervall von 5 Minuten konnten nun über 600.000 Alarme ins System eingespielt werden, statt wie vorher nur um die 230.000 Alarme.
Daraus resultierend entstand ein starker Stau in den letzten beiden Queues. Die Messung beinhaltete 387280 Nachrichten, welche in 10 min versendet wurden. Dabei waren zum Zeitpunkt des größten Staus etwa 59000 Nachrichten in der letzten Queue. Um diesen Stau zu verhindern wurden weitere Consumer-Threads zu den letzten beiden Services hinzugefügt. Der erwartete Effekt der Reduzierung des Staus trat ein, allerdings befinden sich so nun deutlich mehr Alarme in den Input-Puffern der letzten beiden Services, wodurch die Auslastung bzw. Verarbeitungszeit pro Verarbeitungszyklus steigt. Durch die starke Steigerung der Output-Rate wrude eine Connection-Timeout erkennbar, welcher nocht nicht korrekt behandelt wurde.

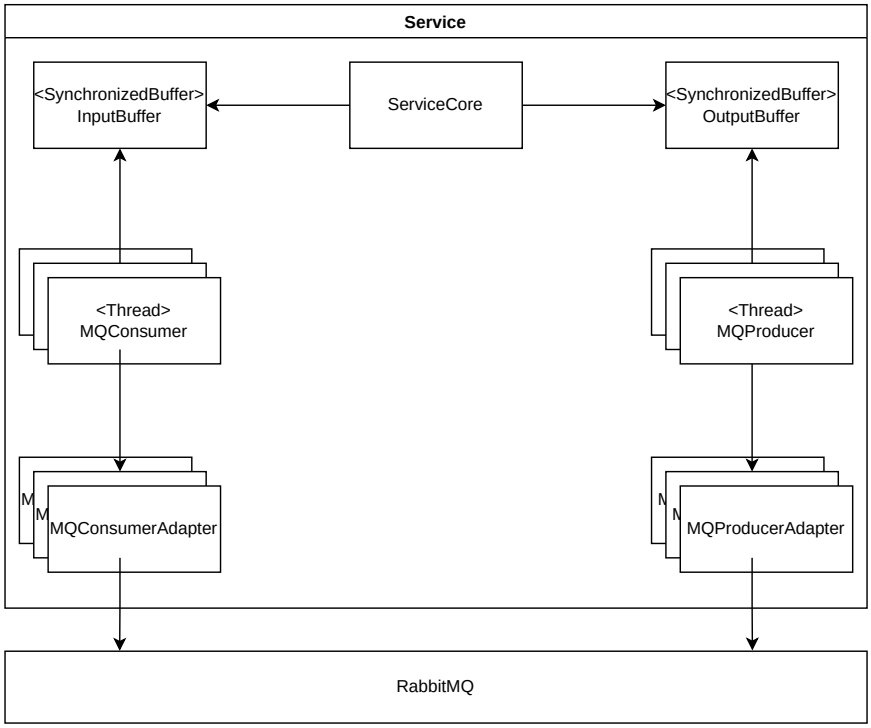

Abbildung 4.19: Abstrakte Darstellung des Zusammenwirkens der Service-Komponenten nach Aufteilung der MQAdapter-Klasse in eine dedizierte Klasse für den Empfang sowie eine dedizierte Klasse für den Versandt von Alarmen an die RabbitMQ. Der Übersichtlichkeit halber wurde die RabbitMQ als eigene Komponente ohne Unterscheidung der darin definierten Queues dargestellt.

4.6.8 Anpassung des Simulationsservices zur besseren Vergleichbarkeit von Lasttests

Die API und damit verbundene Funktionsweise des Simulationsservices wurde dahingehend angepasst, dass nun statt einer Simulationslänge in Sekunden und einer Anzahl von maximal vorzubereitenden Testalarmen eine Simulationslänge in Sekunden und eine Alarmrate in A/s angegeben werden kann. Dies führt zu einer besseren Vergleichbarkeit von Simulationsläufen und trägt zu einer besseren Veranschaulichung der Testläufe bei.

4.6.9 Aufteilung des MQAdapters in MQConsumer- und MQPublisher-Adapter

Um eine explizitere Trennung von Consumer- und Producer-Verbindungen zur RabbitMQ zu erreichen wurde die MQAdapter-Klasse in zwei dedizierte Klassen aufgeteilt. So wird eine Trennung der Zuständigkeiten erreicht, da die MQAdapter-Klasse bisher sowohl für den Empfang, als auch für den Versandt von Nachrichten an die RabbitMQ verantwortlich war. Eine Unterteilung in je eine dedizierte Klasse für den Empfang und den Versandt von Nachrichten macht ebenfalls auf Ebene der Datei- bzw. Klassenstruktur deutlich wie der Nachrichtenfluss stattfindet.
Abbildung 4.19 zeigt das Zusammenwirken der so entstehenden Klassen. Die in Abschnitt 4.6.4 angesprochene Paarbildung zwischen MQConsumer- und MQProducer-Threads ist nun nicht länger notwendig, da jeder Thread eine jeweilige Adapterinstanz zur Verfügung gestellt bekommt. Somit sind die RabbitMQ-Verbindungen nach Art der Verbindung getrennt und können unabhängig von einer Paarbildung erzeugt werden. So können bspw. mehr Producer-Threads und damit einhergehend mehr MQProducer-Adapter erzeugt werden, ohne dass gleichzeitig auch die Anzahl der Consumer-Verbindungen erhöht wird.
Des Weiteren ist dieser Ansatz offen gegenüber der Wiederverwendung von RabbitMQ-Verbindungen. Damit können Verbindungen und damit Ressourcen gespart werden, sofern dies notwendig werden sollte. Um dies zu erreichen können die erzeugten MQConsumerAdapter bzw. MQProducerAdapter zwischen verschiedenen Threads geteilt werden. Diese Änderung führt demnach zu einer besseren Kontrolle über die erzeugten RabbitMQ-Verbindungen, sowie zu einer besseren Verständlichkeit der Nachrichtenflüsse, auch unter Berücksichtigung der Klassenhirarchie und deren Darstellung in der Dateistruktur innerhalb des Projekts.

4.6.10 Seperates Deployment des Elastic-Stacks und der RabbitMQ

Um die Leistungsfähigkeit des Systems bei steigender Anzahl von Threads zu untersuchen, wurden der Elastic-Stack und die RabbitMQ auf einem seperaten Rechner im selben Netzwerk in Betrieb genommen. Durch diesen Versuch sollte in Erfahrung gebracht werden inwiefern mehr Ressourcen (CPU-Zeit und RAM) zu einer höheren Performanz des Systems führen würden. Da sich die Leistungsfähigkeit allerdings nicht veränderte, kann

angenommen werden, dass bereits in den vorherigen Versuchen ausreichend Ressourcen zur Verfügung standen um das System zu betreiben.

4.6.11 Entwicklung eines abstrakten Koordinator-Services

Wie in Abschnitt 4.3.2 beschrieben, ist die Einführung eines Koordinator-Services wichtig um die Skalierbarkeit der Buffer-based Services zu gewährleisten. Die Einführung eines solchen Services kann als Weiterentwicklung des in Abschnitt 4.6.5 erläuterten Ansatzes betrachtet werden. In diesem Fall wird allerdings die Hash-Funktion und die damit zusammenhängende Verteilung gleichartiger Alarme als eigener One-by-one Service vor dem jeweils nachgelagerten Service der Verarbeitungskette eingesetzt. Dieses Verhalten macht diesen Service ebenfalls skalierbar, wie auch in Abbildung 4.11 und 4.13 bereits dargestellt.
In Abschnitt 4.2 wird thematisiert, dass die Anordnung der Services unterschiedliche Ergebnisse in der Aggregation herbeiführen kann. In diesem Proof-Of-Concept wurde der Koordinator-Service als vorgelagerter Service des zweiten One2Many-Services implementiert. Der Koordinations-Service teilt die Last der in Elasticsearch endenden Systeme auf wie angenommen.

4.6.12 Verschiebung der Environment Services auf externe Hardware

Um eine realistischere Umgebung zu schaffen, in welcher die verwendeten Systeme stärker von einander getrennt sind, wurden der Elastic-Stack und die RabbitMQ Instanz auf eine zur Verfügung gestellte Virtuelle Maschine verschoben. Dadurch wurde die Kommunikationszeit der Services deutlich erhöht. Bei genauerer Untersuchung konnte festgestellt werden, dass gerade das Publizieren von Nachrichten bei steigenden Alarmzahlen viel Zeit in Anspruch nimmt. Dies ist damit zu erklären, dass RabbitMQ keine BULK-API anbietet. Dies bedeutet, dass für jeden versendeten Alarm ein HTTP-Request gestellt werden muss, anstatt mehrere Alarme zu bündeln und gleichzeitig zu versenden.
Um dies zu umgehen, müsste eine weitere Softwarekomponente geschrieben werden, welche eine Schnittstelle zum Bündeln von Nachrichten anbietet. Diese Komponente sollte auf der gleichen Virtuellen Maschine, oder dem gleichen Rechner wie die RabbitMQ Instanz betrieben werden, um die Zustellung der Nachrichten von dort aus zu beschleunigen.

5 Messungen

Um verschiedene Aspekte des implementierten Prototypen zu untersuchen, wurden Messreihen in drei unterschiedlichen Kategorien durchgeführt.

1. **Lasttest des Gesamtsystems**: Um den Umgang der gesamten Verarbeitungskette mit hoher Last zu untersuchen, wurden Messungen mit verschiedenen Alarmraten durchgeführt. Dabei wurden insbesondere die Extremfälle der minimalen und maximalen Aggregation betrachtet. Des weiteren kann anhand dieser Messungen bestimmt werden, welche Alarmrate für die anderen Messreihen verwendet werden kann, ohne dabei das System zu überlasten.

2. **Stresstests auf Service Ebene**: Die Leistungsfähigkeit der einzelnen Buffer-based Services wurde exemplarisch mit drei verschiedenen Extremfällen untersucht. Darüber hinaus wurde betrachtet inwiefern der Einsatz des Koordinationsservices zu einer Verminderung der Last auf einem der Puffer-basierten Services führt.

3. **Vergleich dreier Konfigurationen**: Um das Verhalten dreier möglicher Konfigurationen zu bestimmen, wurden die Extremfälle der minimalen und maximalen Aggregation bei gleichbleibender Alarmrate untersucht.

In den Messreihen der genannten Testszenarien wurden die folgenden drei Werte genauer betrachtet:

- **Verarbeitungszeit**: Die im System verbrachte Zeit aller Alarme eines Simulationslaufes. Gemessen wird dabei ab dem *timestamp* Attribut des ersten versendeten Alarms, bis zum Zeitstempel des Versendens and die Monitoring-Services des letzten Alarms der Simulation. Gemessen wird dieser Wert in Sekunden.

- **Alarmreduzierung**: Das Verhältnis zwischen Input und Output Alarmen. Dabei wird die Anzahl der Output Alarme durch die Anzahl der Input Alarme geteilt.

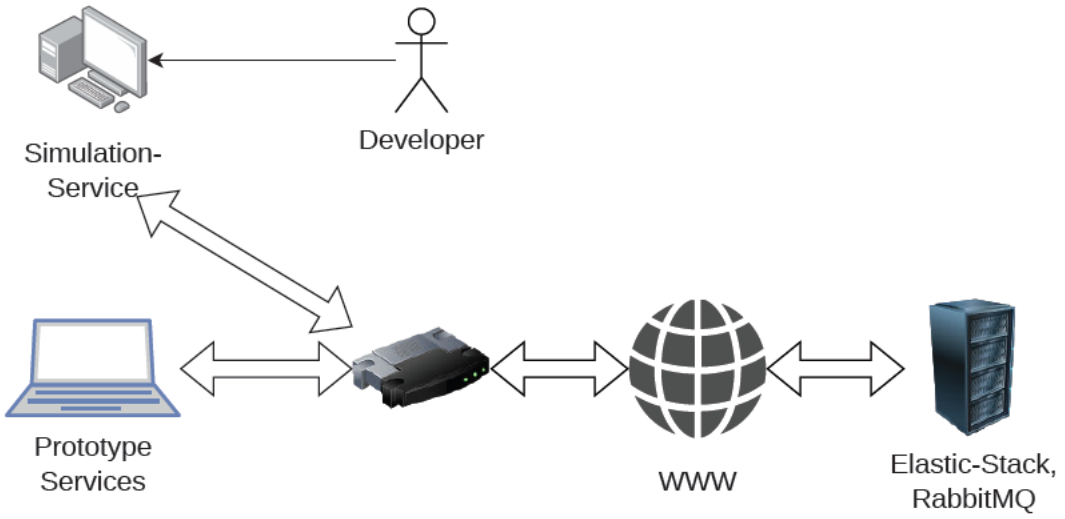

Abbildung 5.1: Verteilung der Services auf die genannte Hardware. Die Kommunikation der Services findet über einen ssh-Tunnel statt.

- **Durchsatz**: Die Verarbeitungsgeschwindigkeit des Systems gemessen in Alarmen pro Sekunde. Dabei wird die Anzahl der Input Alarme durch die Verarbeitungszeit geteilt.

Für die Messungen wurde die in Abschnitt 4.1.1 genannte Hardware verwendet. Um zu verhindern, dass die Erzeugung von Testdaten Ressourcen der Korrelationskette bindet, wurde der Simulationsservice auf einem zusätzlichen Rechner betrieben. Daraus ergibt sich die in Abbildung 5.1 dargestellte Verteilung der Services auf die verwendete Hardware.

Eine Laufzeitabschätzung der Verarbeitungskette kann mit den folgenden beiden Formeln beschrieben werden. Dabei beschreibt die Formel 5.1a die Verarbeitungszeit der Korrelationskette ohne integrierten Koordinationsservice und die Formel 5.1b die Verarbeitungszeit der Korrelationskette mit integriertem Koordinationsservice.

$$\text{Laufzeit [s]} \approx t_{Sim.} + t_{Norm.} + t_{Pre.} + t_{Fusion} + t_{O2O} + t_{O2M} + t_{M2O} \tag{5.1a}$$

$$\text{Laufzeit [s]} \approx t_{Sim.} + t_{Norm.} + t_{Pre.} + t_{Coord.} + t_{Fusion} + t_{O2O} + t_{O2M} + t_{M2O} \tag{5.1b}$$

Da in den getätigten Messungen immer auf Systemebene definiert wurde, welche Pufferzeit pro Puffer basiertem Service verwendet wird, kann die Formel vereinfacht werden. Zusätzlich wurde bei früheren Messungen festgestellt, dass die One-by-one Services eine

Alarmrate [Alarme/s]	Simulationszeit [s]
250	60
250	300
250	600
500	60
500	300
500	600
1000	60
1000	300
1000	600

Tabelle 5.1: Konfigurationen der verschiedenen Testreihen der Lasttests

Laufzeit von unter einer Sekunde aufweisen und können deshalb in der Formel vernachlässigt werden. Grundsätzlich können daher für die Abschätzung der Verarbeitungszeit des Systems die folgenden beiden Formeln herangezogen werden:

$$\text{Laufzeit [s]} \approx t_{Sim.} + \text{Services}_{Puffer} \cdot \text{Verarbeitungszeit}_{Service} \tag{5.2a}$$

$$\text{Laufzeit [s]} \approx t_{Sim.} + \text{Services}_{Puffer} \cdot t_{Puffer} \tag{5.2b}$$

Dabei bildet die Formel 5.2a die Best Case Betrachtung ab und Formel 5.2b die Worst Case Betrachtung. Dies trifft allerdings nur dann zu, wenn die Verarbeitungszeit der Buffer-based Services geringer als die konfigurierte Pufferzeit ist. Die erhobenen Messwerte der im Folgenden geschilderten Szenarien werden in Kapitel 6 ausgewertet und interpretiert.

5.1 Lasttests des Gesamtsystems

Um Referenzwerte für zukünftige Arbeiten und geeignete Alarmraten für die anderen Messreihen zu ermitteln, wurde untersucht wie viele Alarme maximal bei einem Deployment der Services auf einer einzelnen Maschine und einem separaten Deployment der Umgebungs-Services auf dem zur Verfügung gestellten Server verarbeitet werden können. Dazu wurde versucht sich einem Maximum schrittweise zu nähern. Da der Simulations-Service kein Bestandteil der eigentlichen Korrelationskette ist, wurde der Service auf einem weiteren Rechner in Betrieb genommen. Um zwei Extremfälle der Korrelation abzudecken wurden getrennte Untersuchungen für ausschließlich aggregierbare und einzigartige Alarme durchgeführt. Dadurch ergeben sich zwei relevante Parameter für die

Sim. Zeit [s]	Laufzeit [s]	Input Alerts	Output Alerts	Reduzierungsfaktor	Durchsatz [A/s]
60	99.46	15000	81.33	184.85	150.80
300	338	75000	357.66	214.05	221.89
600	641.66	150000	706.33	212.85	233.77

Tabelle 5.2: Messwerte der Lasttests mit einer Alarmrate von 250 Alarmen pro Sekunde und aggregierbaren Alarmen

Sim. Zeit [s]	Laufzeit [s]	Input Alerts	Output Alerts	Reduzierungsfaktor	Durchsatz [A/s]
60	94.89	30000	185.66	161.89	316.16
300	339	150000	646.33	239.67	442.48
600	638.5	300000	964.66	324.36	469.85

Tabelle 5.3: Messwerte der Lasttests mit einer Alarmrate von 500 Alarmen pro Sekunde und aggregierbaren Alarmen

Simulationen. Die Rate der eingehenden Alarme und die Simulationszeit. Anhand dieser beiden Parameter können verschiedene Testreihen durchgeführt werden um die Leistungsfähigkeit des Systems zu messen. Diese sind in Tabelle 5.1 dargestellt.
Setzt man die jeweiligen Werte in die Formel zur Berechnung des Best Case der Verarbeitungszeit ein (Formel 5.2a), so erhält man folgende Berechnungen:

$$68s \approx 60s + 4 \cdot 2s \tag{5.3a}$$

$$308s \approx 300s + 4 \cdot 2s \tag{5.3b}$$

$$608s \approx 600s + 4 \cdot 2s \tag{5.3c}$$

Für die Worst Case Betrachtung (Formel 5.2b) ergeben sich folgende Werte:

$$100s \approx 60s + 4 \cdot 10s \tag{5.4a}$$

$$340s \approx 300s + 4 \cdot 10s \tag{5.4b}$$

$$640s \approx 600s + 4 \cdot 10s \tag{5.4c}$$

Im Folgenden werden die beiden Extremfälle genauer erläutert.

5.1.1 Aggregierbare Alarme

Um eine maximale Aggregation zu simulieren, muss jeder Service in der Korrelationskette in der Lage sein Alarme zu aggregieren. Daraus ergibt sich ein etwas komplexerer Testaufbau als der im Abschnitt 5.2 beschrieben. Die Komplexität soll durch Abbildung 5.2 verdeutlicht werden. Die in der Abbildung verwendeten Buchstaben a bis k stehen

Sim. Zeit [s]	Laufzeit [s]	Input Alerts	Output Alerts	Reduzierungsfaktor	Durchsatz [A/s]
60	111.97	60000	103.67	585.53	535.87
300	413.57	300000	449	669.65	725.40
600	795.43	600000	805.67	746.04	754.31

Tabelle 5.4: Messwerte der Lasttests mit einer Alarmrate von 1000 Alarmen pro Sekunde und aggregierbaren Alarmen

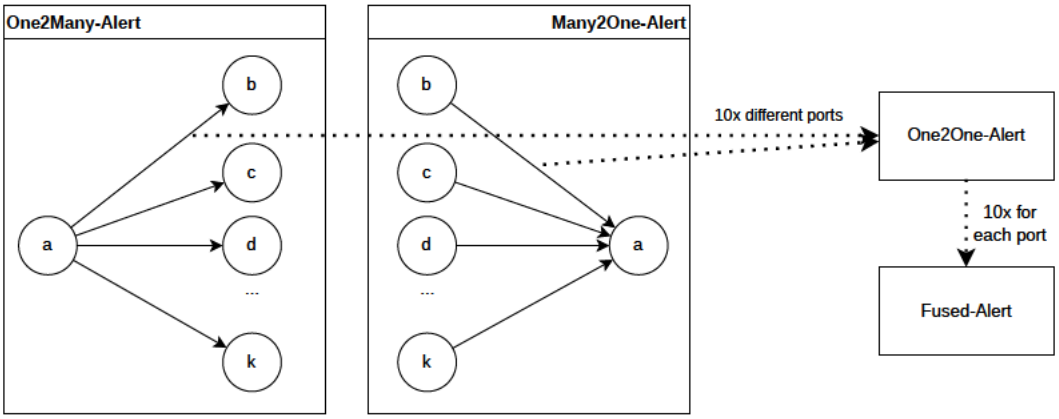

Abbildung 5.2: Zusammensetzung der Alarme zur maximalen Aggregation durch das Gesamtsystem

für IP Adressen, deren Verbindungsrichtung durch die Pfeile dargestellt wird. Insgesamt werden Sammlungen von 2.000 Alarmen ins System eingespielt, welche am Ende der Verarbeitungskette zu zwei Meta-Alarmen zusammengefügt werden. Abbildung 5.2 zeigt den Aufbau der Alarme, welcher sich ergibt indem man die Verarbeitungskette rückwärts durchläuft. Somit besteht ein One2Many- oder Many2One-Alert aus je zehn One2One-Alerts. Jeder One2One-Alert bündelt dabei zehn Verbindungen mit verschiedenen Ports. Davon wiederum wird jede Verbindung verzehnfacht ins System eingespielt, damit der Fusion-Service diese wiederum zusammenfasst. Daraus resultiert eine Alarmreduzierung um den Faktor zehn pro Verarbeitungsschritt. Zusätzlich wird die in Abschnitt 5.3 geschilderte Monitoring-Konfiguration aus Tabelle 5.15 verwendet, da diese den Mittelwert der Konfigurationen bildet.
Für dieses Szenario kann von einer Tendenz in Richtung der Best Case Betrachtung ausgegangen werden, sofern die Services nicht überlastet werden und das jeweilige Maximum der zu aggregierenden Alarme erreicht wird (Formeln 5.3a bis 5.3c). In diesem Fall wäre dann die Verarbeitungszeit der Buffer-based Services geringer als die definierte Pufferzeit. Die Messergebnisse sind in den Tabellen 5.2, 5.3 und 5.4 dargestellt.

Sim. Zeit [s]	Laufzeit [s]	Input Alerts	Output Alerts	Reduzierungsfaktor	Durchsatz [A/s]
60	105.83	15000	15000	1	141.73
300	344.87	75000	75000	1	217.48
600	646	150000	150000	1	232.20

Tabelle 5.5: Messwerte der Lasttests mit einer Alarmrate von 250 Alarmen pro Sekunde und einzigartigen Alarmen

Sim. Zeit [s]	Laufzeit [s]	Input Alerts	Output Alerts	Reduzierungsfaktor	Durchsatz [A/s]
60	107.9	30000	30000	1	278.04
300	533.23	150000	150000	1	281.30
600	1154.33	300000	300000	1	259.89

Tabelle 5.6: Messwerte der Lasttests mit einer Alarmrate von 500 Alarmen pro Sekunde und einzigartigen Alarmen

5.1.2 Einzigartige Alarme

Die in diesem Abschnitt ausgeführten Lasttests sollen die Leistungsfähigkeit des Systems in Bezug auf die Verarbeitung von nicht aggregierbaren Alarmen messen. Dabei wird in jedem beteiligten Pufferservice die maximale Pufferzeit ausgereizt, wodurch die Puffer des Systems stark gefüllt werden und damit der Verwaltungsaufwand innerhalb der Services steigt. In diesem Fall kann von einer Tendenz in Richtung der Worst Case Berechnung ausgegangen werden (Formeln 5.4a bis 5.4c). Die Messergebnisse werden in den Tabellen 5.5, 5.6 und 5.7 dargestellt.

5.2 Stresstests auf Service Ebene

Für jeden Buffer-based Service existiert ein Szenario in dem minimal oder maximal aggregiert werden kann. Zusätzlich zu diesen beiden Extremfällen existiert ein Szenario, welches die Robustheit gegenüber Verzögerungen überprüfen kann. Anhand dieser Szenarien können die oben genannten Werte (Verarbeitungszeit, Alarmreduzierung und Durchsatz) von verschiedenen Konfigurationen untersucht werden. Da die Buffer-based Services sehr ähnlich arbeiten, wurden drei Messungen mit je drei Konfigurationen exemplarisch am Fusion-Service durchgeführt. Um die geschilderten Szenarien auf Service Ebene zu untersuchen, wurden die Messreihen auf den gewählten Service zugeschnitten. Dies bedeutet im Fall der maximalen Aggregation, dass ausschließlich der Fusion Service im Laufe der Messung aggregieren soll. Daraus folgt, dass um die Verarbeitungszeit des Gesamtsystems zu reduzieren, alle weiteren Services für maximalen Durchsatz konfiguriert werden

Sim. Zeit [s]	Laufzeit [s]	Input Alerts	Output Alerts	Reduzierungsfaktor	Durchsatz [A/s]
60	186.63	60000	60000	1	321.49
300	1054.27	300000	298605	1.01	284.56
600	3226	600000	589392.67	1.02	185.99

Tabelle 5.7: Messwerte der Lasttests mit einer Alarmrate von 1000 Alarmen pro Sekunde und einzigartigen Alarmen

Pufferzeit [s]	Laufzeit [s]	Anzahl Output Alarme	Reduzierungsfaktor	Durchsatz [Alarme/s]
5	64.46	3000	10	465.36
30	63.18	3000	10	467.44
120	64.91	3000	10	462.18
Erwartung:	66s	3000	10	454.55

Tabelle 5.8: Gemessene Werte des Fusion-Services bei maximaler Aggregationsmöglichkeit und unterschiedlichen Pufferzeiten

können. So wurde die Konfiguration aller anderen Services auf eine Pufferzeit von einer Sekunde gesetzt. Zusätzlich wurde die Anzahl der maximal zu aggregierenden Alarme auf eins gesetzt. Es wurden für eine Konfiguration des Systems mit einem vorgeschalteten Coordination-Service, sowie ohne Coordination-Service jeweils drei Messreihen durchgeführt. Diese stellen die drei Extremfälle der Verarbeitung innerhalb des Services dar. Des weiteren wurden für jede Messreihe drei Messläufe durchgeführt, welche anschließend gemittelt wurden, um Ausreißer in den Messungen abzufangen.
Zur Abschätzung der Verarbeitungszeit der Korrelationskette wird eine abgewandelte Version der Formel 5.2a verwendet, welche die anderen Buffer-based Services nicht berücksichtigt, da deren Verhalten wie das eines One-by-one Services konfiguriert sind. Wenn man die Verarbeitungszeit der One-by-one Services allerdings mit abschätzen möchte, kann ein Wert von einer halben Sekunde angenommen werden. Somit ergeben sich folgende Formeln zur Best Case (Formel 5.5a) und Worst Case Abschätzung (Formel 5.5b):

$$\text{Laufzeit [s]} \approx t_{Sim} + t_{Fusion} + \text{Services}_{\textit{One-by-one}} \cdot 0.5s \tag{5.5a}$$

$$\text{Laufzeit [s]} \approx t_{Sim} + t_{Pufferzeit} + \text{Services}_{\textit{One-by-one}} \cdot 0.5s \tag{5.5b}$$

Maximale Aggregation

Die erste Messreihe behandelt den Fall, dass ausschließlich fusionierbare Alarme in das System eingespielt werden. Dabei werden Alarmpakete in der Größe der maximalen Anzahl von Alarmen pro Aggregationsschritt vorbereitet. Nur die Alarme innerhalb eines

Pufferzeit [s]	Laufzeit [s]	Anzahl Output Alarme	Reduzierungsfaktor	Durchsatz [Alarme/s]
5	65.8	3000	10	455.93
30	66.43	3000	10	451.58
120	67.22	3000	10	446.30
Erwartung:	67s	3000	10	447.76

Tabelle 5.9: Gemessene Werte des Fusion-Services bei maximaler Aggregationsmöglichkeit und unterschiedlichen Pufferzeiten mit integriertem Koordinationsservice

solchen Pakets sind miteinander fusionierbar. Dies soll verhindern, dass die Reduzierung der Alarme fälschlicherweise stärker ausfällt. Im Zuge dieser Messreihe werden drei verschiedene Konfigurationen gemessen. Dabei wird die Pufferzeit des Fusion-Services im ersten Durchlauf auf fünf Sekunden, im zweiten Durchgang auf 30 Sekunden und im letzten Durchgang auf 60 Sekunden gesetzt. Pro Konfiguration werden 60 Sekunden lang 500 fusionierbare Alarme pro Sekunde von dem Simulationsservice versendet. Demnach werden 30.000 Alarme in das System eingespielt. Die Anzahl der maximal zu fusionierenden Alarme wurde im Fusion-Service auf 10 gesetzt, in allen anderen Services auf 1. Dies führt in diesem Fall zu einer maximalen Reduzierung der Gesamtalarme um den Faktor 10, da nur der Fusion-Service Aggregationen durchführt. Sofern die Leistung des Systems in der gegebenen Konfiguration ausreicht, wird demnach für die Anzahl der Output Alarme ein Wert von 3000 erwartet.
Die Verarbeitungszeit des Fusion-Services ist schwer abzuschätzen, da in dieser Messreihe der Wert der maximal zu aggregierenden Alarme entscheidend ist. Bei Erreichen der konfigurierten Anzahl von Alarmen wird direkt aggregiert, ohne auf das Eintreffen weiterer Alarme zu warten. Da in diesem Fall ausschließlich fusionierbare Alarme in das System eingespielt werden, wird dieser Wert vermutlich immer erreicht. Daher kann die Verarbeitungszeit anhand der Formel 5.5a berechnet werden, welche die Berechnung der Verarbeitungszeit im Best Case beschreibt. Wird von dem Positiv-Fall ausgegangen, so kann die Verarbeitungszeit des Fusion-Services auf eine Sekunde geschätzt werden. Im Fall der Verarbeitungskette mit integriertem Koordinationsservice muss dessen Verarbeitungszeit ebenfalls betrachtet werden. Da es sich hierbei um einen One-by-one Service wie den Normalisierungsservice oder den Proprocessing-Service handelt, kann auch hier eine Verarbeitungszeit von einer halben Sekunde angenommen werden.
Zusammengefasst ergeben sich nun für die geschilderten Messkonfigurationen ohne Berücksichtigung von Netzwerkinterferenzen oder ggf. auftretende Verarbeitungsstaus diese Werte:

Pufferzeit [s]	Laufzeit [s]	Anzahl I/O Alarme	Durchsatz [Alarme/s]
5	67.5	30000	444.44
30	92.5	30000	324.32
60	122.5	30000	244.9

Tabelle 5.10: Erwartete Werte des Fusion-Service zentrierten Aufbaus bei minimaler Aggregationsmöglichkeit

Pufferzeit [s]	Laufzeit [s]	Anzahl I/O Alarme	Durchsatz [Alarme/s]
5	68	30000	441.18
30	93	30000	322.58
60	123	30000	243.90

Tabelle 5.11: Erwartete Werte des Fusion-Service zentrierten Aufbaus bei minimaler Aggregationsmöglichkeit unter Verwendung eines Koordinationsservices

$$63.5s \approx 60s + 1s + 5 \cdot 0.5s \tag{5.6a}$$

$$64s \approx 60s + 1s + 6 \cdot 0.5s \tag{5.6b}$$

Dabei beinhaltet die Formel 5.6a die geschätzte Verarbeitungszeit ohne Koordinationsservice und 5.6b die geschätzte Verarbeitungszeit mit integriertem Koordinationsservice. Tabelle 5.8 zeigt die Ergebnisse der drei beschriebenen Testreihen inklusive der erwarteten Werte in der letzten Zeile. Tabelle 5.9 stellt die Ergebnisse der gleichen Messungen unter Verwendung eines Koordinationsservices dar.

Minimale Aggregation

Diese Testreihe beschreibt den gegensätzlichen Extremfall zum vorangegangenen Abschnitt. In diesem Fall werden ausschließlich nicht miteinander aggregierbare Alarme in einem Zeitraum von 60 Sekunden bei einer Rate von 500 Alarmen pro Sekunde in das System eingespielt. Dies ergibt eine Gesamtanzahl von 30000 zu verarbeitenden Alarmen. Anders als im vorherigen Abschnitt hat der Wert der maximal zu aggregierenden Alarme in dieser Messreihe keine große Bedeutung, da ihr Wert nie erreicht wird. Da dies der Fall ist, wird die konfigurierte Pufferzeit das Maximum der Verarbeitungszeit des Fusion-Services bestimmen.
Die im vorangegangenen Abschnitt getroffenen Annahmen über die Verarbeitungszeiten

Pufferzeit [s]	Laufzeit [s]	Anzahl I/O Alarme	Durchsatz [Alarme/s]
5	80.12	30000	374.45
30	105.47	30000	284.45
60	178.73	30000	167.85

Tabelle 5.12: Gemessene Werte des Fusion-Service zentrierten Aufbaus bei minimaler Aggregationsmöglichkeit

Pufferzeit [s]	Laufzeit [s]	Anzahl I/O Alarme	Durchsatz [Alarme/s]
5	101.24	30000	296.32
30	118.47	30000	253.24
60	141.57	30000	211.91

Tabelle 5.13: Gemessene Werte des Fusion-Service zentrierten Aufbaus bei minimaler Aggregationsmöglichkeit unter Verwendung eines Koordinationsservices

der einzelnen Services trifft auch in diesem Fall zu, abgesehen von der Verarbeitungszeit des Fusion-Services. Dieser Wert kann auf den Wert der Pufferzeit approximiert werden. Setzt man nun die Werte in die Formel zur Berechnung der Verarbeitungszeit im Worst Case ein (Formel 5.5b), ergeben sich die folgenden Berechnungen:

$$\text{Laufzeit [s]} \approx t_{Sim} + t_{Pufferzeit} + \text{Services}_{\text{One-by-one}} \cdot 0.5s \tag{5.7a}$$

$$\text{Laufzeit [s]} \approx 60s + t_{Pufferzeit} + 5 \cdot 0.5s \tag{5.7b}$$

$$\text{Laufzeit [s]} \approx 60s + t_{Pufferzeit} + 6 \cdot 0.5s \tag{5.7c}$$

Die Formel 5.7a entspricht der Formel 5.5b und wurde nur zur Nachvollziehbarkeit erneut aufgeführt. Die Formeln 5.7b und 5.7c stellen der Konfiguration ohne und mit Koordinationsservice dar.
Setzt man die Werte der Pufferzeit in Formel 5.7b ein, so ergeben sich die approximierten Laufzeitwerte 67.5s, 92.5s und 122.5s für die Konfiguration ohne Koordinationsservice. Setzt man die Werte der Pufferzeit in Formel 5.7c ein, so ergeben sich die approximierten Laufzeitwerte 68s, 93s und 123s für die Konfiguration mit Koordinationsservice. Eine Übersicht der erwarteten Werte ist in den Tabellen 5.10 und 5.11 abgebildet. Die gemessenen Werte sind in den Tabellen 5.12 und 5.13 dargestellt.

Verzögerung [s]	Geschätzte Laufzeit [s]	Laufzeit [s]	Output Alarme
3	147	148.83	5
4	196	197.37	5
5	255	252.73	41

Tabelle 5.14: Gemessene Werte des Fusion-Services bei mit Verzögerung versendeten, fusionierbaren Alarmen. Die konfigurierte Pufferzeit beträgt fünf Sekunden, die Anzahl der Input Alarme beträgt 50, darin enthalten sind fünf Pakete mit je zehn fusionierbaren Alarmen.

Maximale Verzögerung der Alarme

Die Messung des Verhaltens des Systems bei maximaler Verzögerung der einzelnen Alarme ist nur in Kombination mit aggregierbaren Alarmen sinnvoll, da andernfalls ausschließlich die maximale Laufzeit gemessen würde, ohne Erkenntnisse über die Stabilität der Korrelation zu gewinnen. Deshalb werden in dieser Messreihe fusionierbare Alarme mit einer Verzögerung in das System eingespielt. Dabei handelt es sich wie in Abschnitt 5.2 um Pakete in der Größe der maximal zu fusionierenden Alarme. In diesem Fall wurde der Fusion-Service auf zehn Alarme konfiguriert. Mit diesem Aufbau soll untersucht werden, ob das definierte Limit der maximal zu aggregierenden Alarme erreicht wird und wie sich die Verzögerung der Alarme auf die Gesamtlaufzeit sowie die Alarmreduzierung des Systems auswirkt. Der gewählte Verzögerungswert wird schrittweise an die Pufferzeit angeglichen. Somit werden pro Konfigurationen der Pufferzeit drei Messungen durchgeführt und anschließend gemittelt. Im ersten Durchlauf wird eine Verzögerung von $t_{Puffer} - 2s$ gewählt, im zweiten Durchlauf ein Wert von $t_{Puffer} - 1s$ und abschließend werden Verzögerung und Pufferzeit gleichgesetzt.
Durch die Verzögerung im Versenden der Alarme ergibt sich eine deutlich geringere Rate von Alarmen pro Sekunde, welche ins System eingespielt werden. Da das Ergebnis der Reduzierung hängt stark vom Zeitpunkt des Eintreffens der Alarme in dem Service ab. Die Anzahl der Input Alarme kann anhand der Simulationszeit und der Pufferzeit des jeweiligen Messlaufs berechnet werden. Da die Last auf dem System sehr gering ist und alle weiteren Services so konfiguriert wurden, dass sie wie ein One-by-one Service agieren, ist deren Verarbeitungszeit in der Berechnung der Laufzeit zu vernachlässigen. Die Laufzeit ist mit der folgenden Formel abschätzbar:

$$\text{Laufzeit [s]} \approx \text{Input Alarme} \cdot t_{Puffer} - t_{Puffer} \tag{5.8a}$$

$$\text{Laufzeit [s]} \approx \text{Input Alarme} \cdot t_{Puffer} + t_{Puffer} \tag{5.8b}$$

Konfiguration	Pufferzeit [s]	Max. Alerts to wait for	Alarmrate [Alarme/s]
Alarmierung	5	5	1000
Monitoring	10	10	1000
Forensik	60	10	1000

Tabelle 5.15: Mögliche Konfigurationen des Systems für die drei genannten Anwendungsfälle

Formel 5.8a zeigt die Berechnung für den Fall dass mit Eintreffen des zehnten Alarms eines Pakets keine Verzögerung mehr stattfindet und direkt fusioniert wird. Formel 5.8b zeigt den entgegengesetzten Fall, dass die einzelnen Alarme eines Pakets erst nach Ablauf der Pufferzeit beim Fusion-Service eintreffen und somit keine Fusion stattfindet. Die Messungen wurden exemplarisch anhand einer Pufferzeit von fünf Sekunden durchgeführt. Für die geschilderten Messreihen ergeben sich somit folgende Laufzeitabschätzungen:

$$147s \approx 50 \cdot 3s - 3s \tag{5.9a}$$

$$196s \approx 50 \cdot 4s - 4s \tag{5.9b}$$

$$255s \approx 50 \cdot 5s + 5s \tag{5.9c}$$

Dabei bilden die Formeln 5.9a und 5.9b die Fälle ab, dass Alarme mit einer Verzögerung von drei bzw. vier Sekunden versendet werden, diese somit noch innerhalb der Pufferzeit liegen und fusioniert werden. Formel 5.9c zeigt den Fall dass die Verzögerung auf den gleichen Wert wie die Pufferzeit gesetzt wurde und somit keine Fusion stattfindet. Die Ergebnisse der Messungen sind in Tabelle 5.14 dargestellt.

5.3 Vergleich dreier möglicher Konfigurationen

Wie in Sektion 2.3 bereits erläutert, kann das System für drei verschiedene Einsatzgebiete konfiguriert werden. Im Folgenden werden die zwei Extremfälle der maximalen und minimalen Aggregation für die drei Konfigurationen miteinander verglichen. Die Messungen werden dahingehend nur beispielhaft durchgeführt, dass nicht pro Puffer-basiertem Service in den Konfigurationen differenziert wird, sondern lediglich ein Wert für alle Puffer-basierten Services pro Konfiguration gewählt wird. Die Konfigurationen für die jeweiligen Einsatzgebiete kann der Tabelle 5.15 entnommen werden.

Konfiguration	Sim. Zeit [s]	Laufzeit [s]	Input Alarme	Output Alarme	Reduzierungsfaktor
Alarmierung	60	84.96	30000	760.67	39.47
Monitoring	60	98.72	30000	152.67	212.58
Forensik	60	257.13	30000	147.67	218.06

Tabelle 5.16: Messergebnisse der jeweiligen Konfigurationen bei ausschließlich aggregierbaren Alarmen

5.3.1 Aggregierbare Alarme

Die Erzeugung der Testdaten dieser Messreihen orientiert sich an der in Sektion 5.1.1 beschriebenen Vorgehensweise. Die Laufzeit der in Tabelle 5.15 dargestellten Konfigurationen kann mit folgender Formel abgeschätzt werden, dabei ist Formel 5.10a die Best Case Betrachtung und die Formel 5.10b die Worst Case Betrachtung:

$$\text{Laufzeit [s]} \approx t_{Sim.} + Services_{Puffer} \cdot 2s \tag{5.10a}$$

$$\text{Laufzeit [s]} \approx t_{Sim.} + Services_{Puffer} \cdot t_{Puffer} \tag{5.10b}$$

In Formel 5.10a wird die Verarbeitungszeit der Services auf zwei Sekunden geschätzt. In der gesamten Korrelationskette werden vier Puffer basierte Services durchlaufen. Formel 5.11a beinhaltet die konfigurierten Werte eingesetzt in Formel 5.10a, die Formeln 5.11b bis 5.11d die Werte der jeweiligen Konfigurationen eingesetzten in die Formel 5.10b.

$$68s \approx 60s + 4 \cdot 2s \tag{5.11a}$$

$$80s \approx 60s + 4 \cdot 5s \tag{5.11b}$$

$$100s \approx 60s + 4 \cdot 10s \tag{5.11c}$$

$$300s \approx 60s + 4 \cdot 60s \tag{5.11d}$$

Des weiteren wird eine Reduzierung der Alarme um den Faktor 1000 angenommen, sofern die maximal zu aggregierenden Alarme immer erreicht werden und kein Service überlastet ist. Die gemessenen Werte können Tabelle 5.16 entnommen werden.

5.3.2 Einzigartige Alarme

Wenn das System ausschließlich mit nicht aggregierbaren Alarmen gefüllt wird, so wird von jedem Service die maximale Laufzeit erreicht. Diese entspricht der maximalen Pufferzeit addiert zur Verarbeitungszeit innerhalb des Services. Zusätzlich wird die Verarbei-

Konfiguration	Laufzeit [s]	I/O Alarme	Reduzierungsfaktor
Alarmierung	93.16	30000	1
Monitoring	111.37	30000	1
Forensik	257.73	30000	1

Tabelle 5.17: Messergebnisse der jeweiligen Konfigurationen bei einzigartigen Alarmen bei einer Alarmrate von 500 Alarmen pro Sekunde und einer Simulationszeit von 60 Sekunden

tungszeit innerhalb des Services in Abhängigkeit zum Füllstand des Puffers gesteigert, da dieser über alle gepufferten Alarme iterieren muss. Daher kann die bereits in Abschnitt 5.3.1 eingeführte Formel zur Worst Case Abschätzung genutzt werden: Dabei beschreibt die Formel 5.11b die eingesetzten Werte der Alarmierungskonfiguration, 5.11c die eingesetzten Werte der Monitoring-Konfiguration und 5.11d die eingesetzten Werte der Forensik-Konfiguration. Die Ergebnisse dieser Testreihe können der Tabelle 5.17 entnommen werden.

6 Auswertung

In diesem Kapitel werden die Ergebnisse der Messungen aus Kapitel 5 ausgewertet und mit den in Kapitel 3 definierten Zielen und Anforderungen abgeglichen. Dabei werden erst die einzelnen Messreihen betrachtet. Anschließend werden die Messergebnisse und die in Kapitel 4 geschilderten architekturellen Eigenschaften verwendet um Aussagen über das Gesamtsystem zu treffen.

6.1 Lasttests

Im Folgenden werden die Ergebnisse der durchgeführten Lasttests ausgewertet. Dazu werden die Messwerte mit den erwarteten Werten abgeglichen und Rückschlüsse über die maximal zu verarbeitende Alarmrate, die Korrelationsqualität und die Laufzeit getroffen.

6.1.1 Aggregierbare Alarme

Die Tabellen 5.2 bis 5.4 zeigen die Ergebnisse der drei Testreihen mit den Alarmraten 250, 500 und 1000 A/s. Die erwarteten Laufzeiten können den Formeln 5.3a bis 5.3c entnommen werden. Des weiteren ist die Erwartung an das Verhalten des Systems, dass die Laufzeit im weiteren Verlauf der Lasttests mit aggregierbaren Alarmen konstant bleibt, sofern das System nicht überlastet wird.
Die Ergebnisse der Testreihe mit einer Alarmrate von 250 A/s zeigen eine deutliche Abweichung gegenüber der erwarteten Laufzeiten. Auffällig dabei ist, dass die Abweichung bei allen gemessenen Simulationszeiten auftritt und zwischen 39,46s und 41,66s liegt. Zusätzlich wurde laut dem in Abschnitt 5.1.1 geschilderten Testaufbau eine Alarmreduzierung um den Faktor 1000 erwartet. Dieser wurde mit einem minimalen Wert von 184,85 für eine Simulationszeit von 60s und einem maximalen Wert von 214,05 eine Simulationszeit von 300s deutlich verfehlt. Der gemessene Durchsatz beträgt minimal 150,80

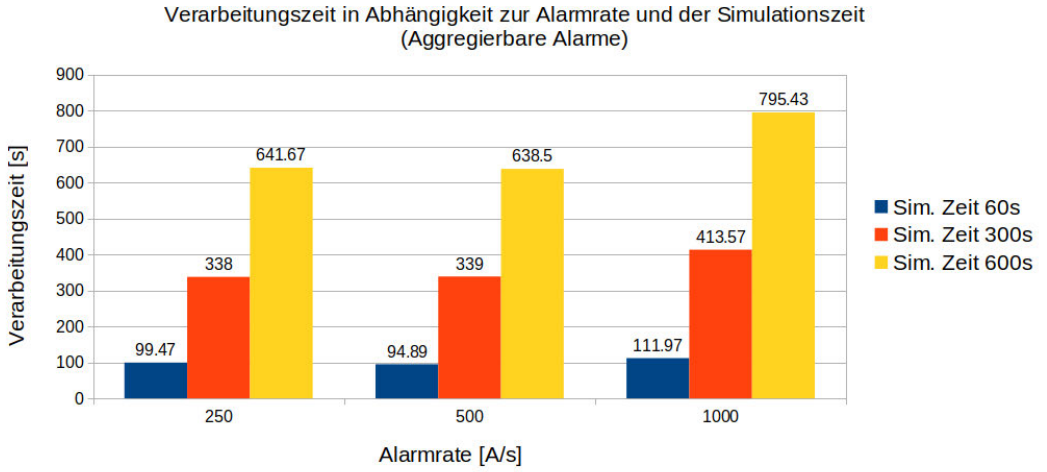

Abbildung 6.1: Gemessene Verarbeitungszeit der Lasttests mit aggregierbaren Alarmen

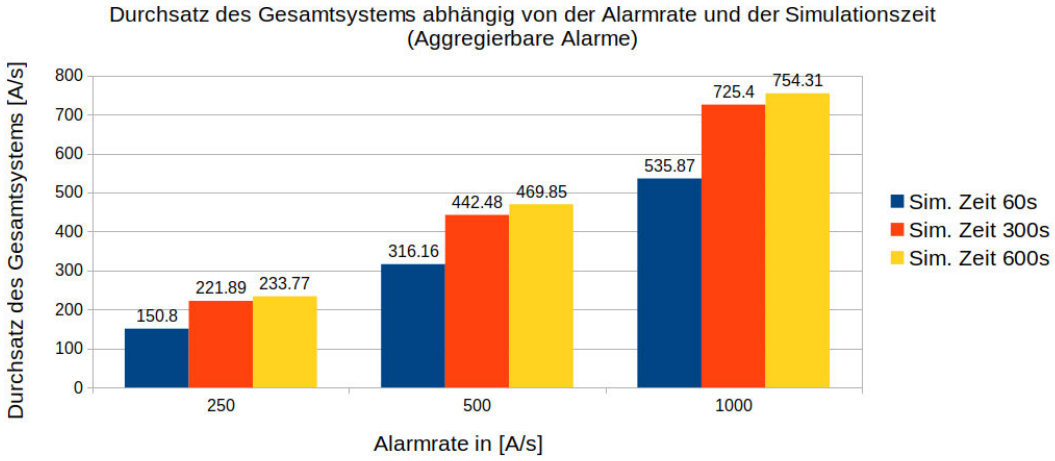

Abbildung 6.2: Gemessener Durchsatz der Lasttests mit aggregierbaren Alarmen

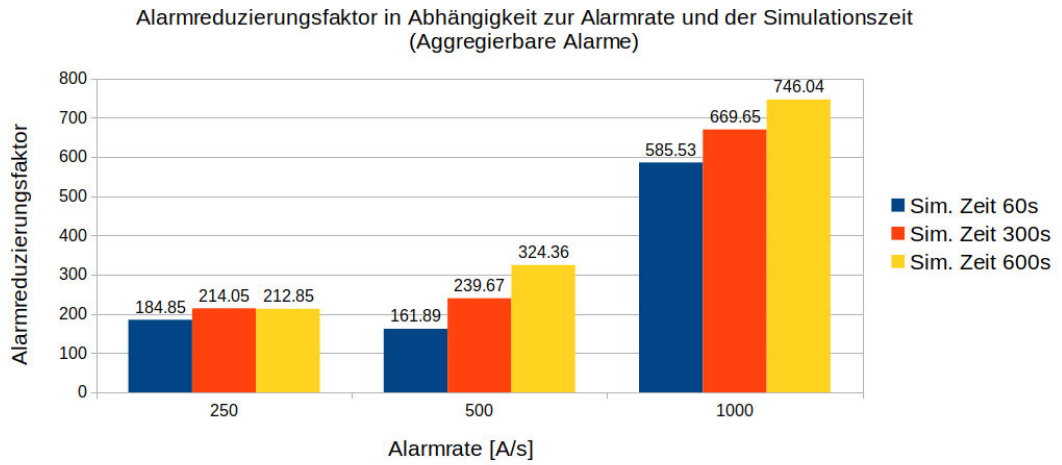

Abbildung 6.3: Gemessene Alarmreduzierungsfaktor der Lasttests mit aggregierbaren Alarmen

A/s für eine Simulationszeit von 60s. Damit ist der Durchsatz 99,2 A/s entfernt von der eingehenden Alarmrate von 250 A/s. Die anderen beiden Messungen liegen mit 221,89 und 233,88 A/s nur knapp unter der eingehenden Alarmrate von 250 A/s.
Die Messwerte der Testreihe mit einer Alarmrate von 500 A/s liefern ein ähnliches Bild. Dabei ist auffällig, dass die Laufzeiten im Vergleich zu den gemessenen Werten der vorangegangen Messreihe mit einer Abweichung von maximal 4,57s in etwa konstant geblieben sind. Dies entspricht dem erwarteten Verhalten, wenngleich die gemessenen Laufzeiten deutlich höher sind als erwartet. Der Reduzierungsfaktor ist für eine Simulationszeit von 60s um einen Wert von 22,96 gesunken. Allerdings ist der Reduzierungsfaktor in den Messungen mit einer Simulationszeit von 300s um einen Wert von 25,62 und für die Simulationszeit von 600s sogar um einen Wert von 111,51 gestiegen. Dennoch ist der damit erreichte Wert von 324,36 deutlich geringer als der mögliche Wert von 1000.
Der Durchsatz der Messreihe ist mit 316,16 A/s bei einer Simulationszeit von 60s maximal 183,84 A/s von der eingespielten Alarmrate entfernt. Auffällig ist hierbei, dass die Werte der anderen beiden Simulationszeiten bei 442,48 und 469,85 A/s liegen. Der minimale Abstand zwischen der eingehenden Alarmrate von 500 A/s und dem maximalen Durchsatz von 469,85 A/s beträgt somit 30,15 A/s. Damit hat sich der Abstand um einen Wert von 13,92 A/s im Vergleich zu den vorherigen Messreihen vergrößert und somit leicht verschlechtert.
Die Ergebnisse der Messreihe mit einer Alarmrate von 1000 A/s zeigen signifikante Unter-

schiede zu den Ergebnissen der bereits betrachteten Messungen. Die Laufzeit, der Reduzierungsfaktor und der Durchsatz sind jeweils deutlich gestiegen. Die Verarbeitungszeit für eine Simulationslänge von 60s liegt in diesem Fall bei 111,97s. Die Verarbeitungszeiten der Messreihen mit einer Simulationszeit von 300s bzw. 600s liegen bei 413,57s bzw. bei 795,43s. Damit betragen die Abweichungen von der erwarteten Verarbeitungszeit 43,51s, 105,57s und 187,43s.
Während die ermittelten Verarbeitungszeiten der Messreihen mit den Alarmraten von 250 und 500 A/s noch ungefähr im gleichen Rahmen lagen, kann im Fall der Messreihe mit einer Alarmrate von 1000 A/s eine klare Steigerung der Verarbeitungszeit in allen Simulationslängen erkannt werden. Dies spricht für eine Überlastung des Systems, wodurch die unlimitierten Puffer der Services überfüllt werden und somit nicht mehr rechtzeitig abgearbeitet werden können. Dieses Verhalten kann Abbildung 6.1 entnommen werden.
Überraschenderweise hat sich jedoch der Durchsatz der Messreihen durchweg gesteigert. Abbildung 6.2 stellt die gemessenen Durchsatz Werte gegenüber. Dabei lässt sich ein klarer Trend erkennen, dass die eine höhere Alarmrate zu einem höheren Durchsatz führt. Dies ist dadurch erklärbar, dass eine höhere Dichte von aggregierbaren Alarmen dazu führt, dass das Maximum von zu aggregierenden Alarmen eher erreicht wird. Dadurch wird nicht länger auf das Eintreffen weiterer Alarme gewartet, sondern direkt aggregiert und das Ergebnis weitergeleitet. Dieses Verhalten reduziert die Zeit der einzelnen Alarme in den Puffern und erhöht somit den Durchsatz. Dieser Umstand wird auch in Abbildung 6.3 verdeutlicht, da dort die Reduzierungsfaktoren dargestellt werden. Dennoch bestehen auch im Fall des Durchsatzes ein Abstand zur eingehenden Alarmrate von 1000 A/s je nach Simulationslänge Werte von 464,13 A/s, 274,6 A/s und 245,69 A/s. Damit ist der Abstand zur eingehenden Alarmrate im Vergleich zur vorangegangenen Messung deutlich gestiegen.
Wie bereits angedeutet sind die Reduzierungsfaktoren für diese Messreihe stark gestiegen. In Abbildung 6.3 ist deutlich erkennbar, dass diese Messreihe die besten Ergebnisse in Bezug auf die Alarmreduzierung liefert.

6.1.2 Zusammenfassung und Interpretation der Ergebnisse

Die gemessenen Werte sollten Aufschluss über das System im Best Case Szenario geben. Die erwarteten Werte konnten in keiner der Kategorien erreicht werden. Dennoch konnten Rückschlüsse über das Verhalten des implementierten Prototypen getroffen werden. Der Aggregationsmechanismus scheint bessere Ergebnisse bei höheren Alarmraten

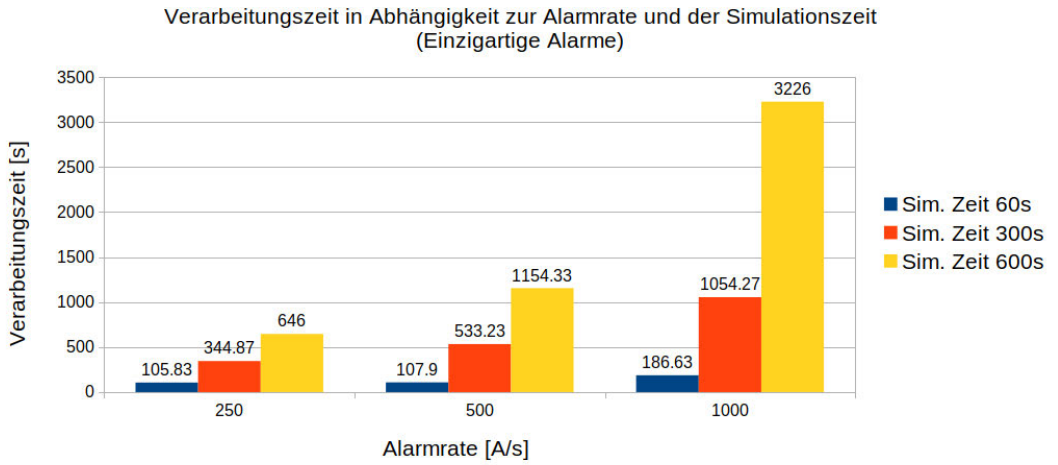

Abbildung 6.4: Gemessene Verarbeitungszeit der Lasttests mit einzigartigen Alarmen

zu erzielen. Dies spricht dafür, dass das Zusammenspiel der Services in Bezug auf ihre konfigurierte Pufferzeit noch nicht optimal ist. Dies hat dazu geführt dass bei den betrachteten Messreihen nicht immer der Wert der maximal zu aggregierenden Alarme erreicht wurde, wodurch im nächsten Schritt der Verarbeitung wiederum mindestens ein Alarm fehlte, um weiterhin aggregieren zu können. Neben dem Effekt der niedrigen Reduzierung führte dies zu maximalen Pufferzeiten bestimmter Alarme, welche wiederum zu einem geringeren Durchsatz geführt haben. Die Steigerung des Durchsatzes und dem Alarmreduzierungsfaktor bei höheren eingehenden Alarmraten bestätigt diese These ebenfalls, da höhere Alarmraten zu einer höheren Dichte von zu aggregierenden Alarmen führt. Dies relativiert die geschilderte Problematik im Zusammenspiels der Services teilweise, wenngleich die Alarmrate von 1000 A/s gemessen an der Verarbeitungszeit zu einer Überlast des Systems geführt hat. Die durchschnittlich besten Werte wurden bei der Alarmrate von 500 A/s und einer Simulationszeit von 300-600s erzielt. Die Verarbeitungszeit weicht nicht stark von der erwarteten Verarbeitungszeit ab und der Durchsatz liegt nur knapp unter der eingehenden Alarmrate. Allerdings bleibt die Alarmreduzierung noch optimierbar.
Weiterhin schien das System durch eine Alarmrate von 500 A/s noch nicht überlastet zu sein, bei einer eingehenden Alarmrate von 1000 A/s hingegen schon.

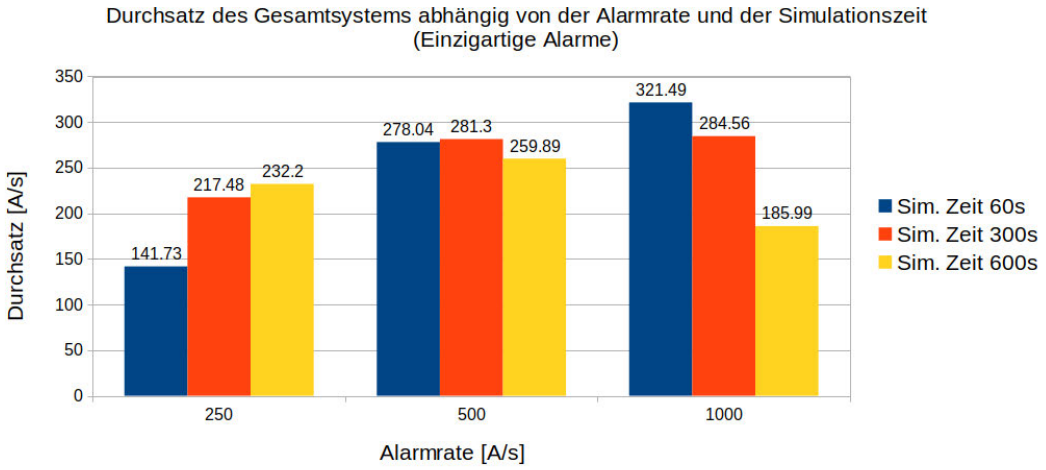

Abbildung 6.5: Gemessener Durchsatz der Lasttests mit einzigartigen Alarmen

6.1.3 Einzigartige Alarme

Die Ergebnisse dieser Messreihen sind in den Tabellen 5.5 bis 5.7 pro Alarmrate dargestellt und werden in den Abbildungen 6.4 und 6.5 gegenübergestellt. Dabei wird die Alarmreduzierung nicht visualisiert, da diese nicht stattfinden kann. Die erwarteten Werte können den Formeln 5.4a bis 5.4c entnommen werden.
In Bezug auf die Verarbeitungszeit ist die Erwartung, dass diese über die verschiedenen eingehenden Alarmraten hinweg konstant bleibt, sofern das System nicht überlastet ist. Allerdings ist in Abbildung 6.4 eine deutliche Steigerung in den Verarbeitungszeiten Alarmraten übergreifend zu erkennen. Bei einer Alarmrate von 250 A/s liegen die Werte nur maximal sechs Sekunden über den erwarteten Werten. Wohingegen bei einer Alarmrate von 1000 A/s im maximal Fall eine Abweichung von 2586s zu erkennen ist. Dies entspricht ca. 43 Minuten. Der Schritt zwischen den beiden niedrigeren Alarmraten ist weniger drastisch. Im Fall einer Simulationszeit von 60s ist ein Anstieg von ca. zwei Sekunden zu beobachten. Wohingegen der Anstieg bei einer Simulationslänge von 300s bereits 188,36s beträgt. Abschließend beträgt der Anstieg in der Verarbeitungszeit bei einer Simulationslänge von 600s bereits 508.33s.
Bei Betrachtung der gemessenen Durchsätze ist zu erkennen, dass im Fall von einer Alarmrate von 250 A/s und einer Simulationslänge von 600s der beste Wert bezüglich des Abstands zur eingehenden Alarmrate erzielt wurde, während der höchste absolute Wert bei einer Simulationslänge von 60s und einer Alarmrate von 1000 A/s erzielt wurde.

Des weiteren ist auffällig, dass die Durchsatz Werte im Fall von einer Alarmrate von 250 A/s und einer Simulationslänge von 60s, sowie bei einer Alarmrate von 1000 A/s und einer Simulationslänge von 600s besonders niedrig ausfielen.
Bei einer eingehenden Alarmrate von 500 A/s befinden sich die gemessenen Werte in einem ähnlichen Rahmen zwischen 259,89 und 281,3 A/s. Wohingegen sich die Messwerte bei einer Alarmrate von 250 A/s genau gegenteilig zu den Messwerten einer Alarmrate von 1000 A/s verhalten. Dabei steigen bei der Messreihe mit der Alarmrate von 250 A/s mit Erhöhung der Simulationszeit leicht an, während die Werte bei einer Alarmrate von 1000 A/s mit Erhöhung der Simulationszeit stark abfallen. Bei der Durchführung letzterer Messreihe ist aufgefallen dass es bei vier von neun Messungen zu einem Fehler bei der Verbindung mit der Messagequeue kam. Aus diesem Grund sind Alarme verloren gegangen, welche in den Angehängten Messungen dementsprechend nicht auftauchen und dazu führen, dass die Reduktion anders als erwartet leicht über dem Wert eins liegt.

6.1.4 Zusammenfassung und Interpretation der Ergebnisse

Aus den geschilderten Ergebnissen lässt sich ableiten, dass das System nur im Fall der Messungen mit einer Alarmrate von 250 A/s nicht überlastet wird. Dabei ist die Überlast im Fall der Alarmrate von 1000 A/s deutlich höher als im Fall der Alarmrate von 500 A/s. Diese Überlast hat in den Messungen der Alarmrate von 1000 A/s dazu geführt, dass ein Verbindungsfehler aufgetreten ist. Dabei ist nicht vollends klar wie dieser Fehler zustande kommt. Allerdings ist dabei aufgefallen dass die Fehlerbehandlung in einem solchen Fall nicht ausreichend implementiert wurde, da dieser Fehler zum Verlust von Nachrichten führt. Da der aufgedeckte Fehler allerdings eine art Timeout-Error ist, erklärt das Auftreten dieses Fehlers die hohen Werte in der Verarbeitungszeit, sowie die niedrigen Werte des Durchsatzes bei den Simulationslänge von 300s und 600s. Bei einer Simulationslänge von 60s ist dieser Fehler nicht aufgetreten. Dies erklärt die relativ niedrige Steigerung der Verarbeitungszeit im Vergleich zur Messung mit einer Alarmrate von 500 A/s und den vergleichsweise hohen Durchsatz.

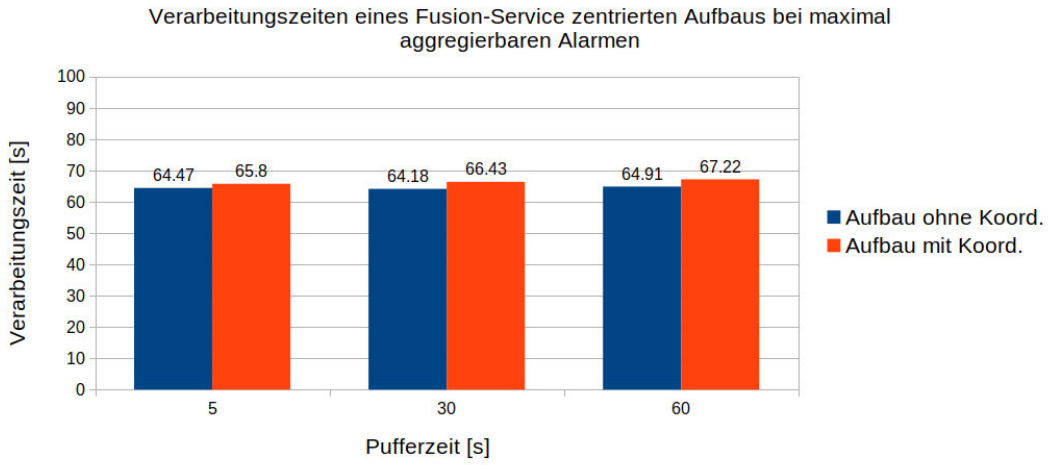

Abbildung 6.6: Gemessene Verarbeitungszeit des Fusion-Service zentrierten Aufbaus bei maximal aggregierbaren Alarmen

6.2 Stresstests auf Service Ebene

Im Folgenden werden die Ergebnisse der durchgeführten Stresstests des Fusion-Services ausgewertet. Dabei werden Rückschlüsse über das Zusammenspiel der Alarme gezogen und Abweichungen von den erwarteten Ergebnissen festgestellt und interpretiert.

6.2.1 Maximale Aggregation

Die Ergebnisse dieser Messungen sind in den Tabellen 5.8 und 5.9 dargestellt. Zusätzlich stellen die beiden Abbildungen 6.6 und 6.7 die Verarbeitungszeiten und den Durchsatz ohne und mit Verwendung eines Koordinationsservices gegenüber. Es wird angenommen dass die Verarbeitungszeit und der Durchsatz im Verlauf dieser Messungen konstant bleibt, da die Pufferzeit nie vollständig ausgereizt wird, sofern das Maximum an zu aggregierenden Alarmen immer erreicht wird. Zusätzlich wurde auf die Darstellung des Alarmreduzierungsfaktors verzichtet, da dieser in allen Messungen den Wert zehn hatte. Die Alarmrate für diese Messung beträgt 500 A/s
Abbildung 6.6 kann man eine leicht erhöhte Verarbeitungszeit im Fall des Aufbaus mit Koordinationsservice entnehmen. Dennoch liegen die Werte nur knapp oberhalb der erwarteten Verarbeitungszeit von 63,5s ohne Verwendung eines Koordinationsservices und 64s mit Verwendung eines Koordinationsservices (siehe Formeln 5.6a und 5.6b). Da sich

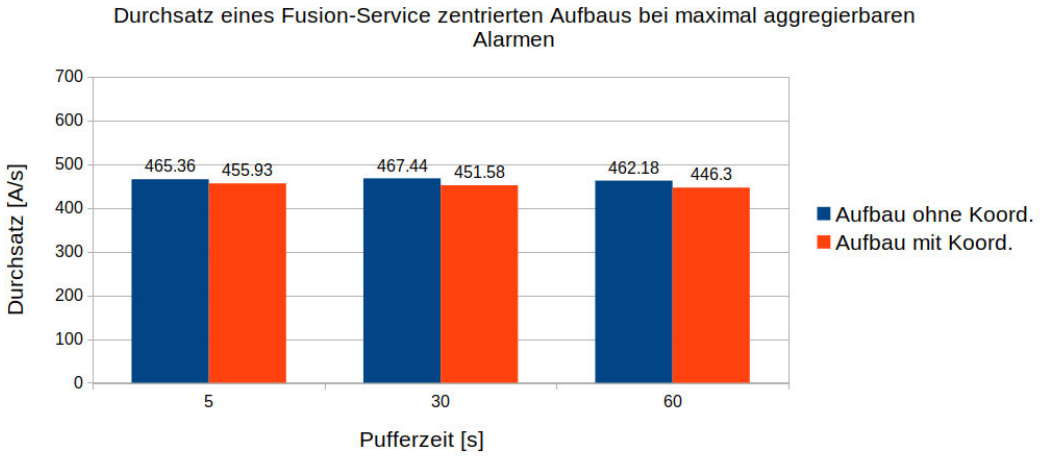

Abbildung 6.7: Gemessener Durchsatz des Fusion-Service zentrierten Aufbaus bei maximal aggregierbaren Alarmen

der Durchsatz des Systems anhand der Verarbeitungszeit berechnen lässt spiegeln die Werte aus Abbildung 6.7 das gleiche Verhalten wieder. Dabei liegt der maximale Durchsatz bei einem Wert von 467,44 A/s und damit nur 32,56 A/s entfernt von der eingehenden Alarmrate.

6.2.2 Zusammenfassung und Interpretation der Ergebnisse

Die geschilderten Ergebnisse bestätigen alle getroffenen Annahmen. Die Verarbeitungszeit ist unabhängig von der Pufferzeit approximiert konstant geblieben. Gleiches gilt für den Durchsatz. Es sind leichte Einbuße gegenüber der Verarbeitungszeit und dem Durchsatz erkennbar, wenn ein Koordinationsservice verwendet wird. Diese sind durch die Verarbeitungszeit des Koordinationsservices und zwei weitere Netzwerkoperationen in der Verarbeitungskette zu erklären, da ein Weiterer Service aus einer Messagequeue lesen und im nächsten Schritt schreiben muss.

6.2.3 Einzigartige Alarme

Die gemessenen Werte dieser Messreihen können den Tabellen 5.12 und 5.13 entnommen werden. Die erwarteten Werte sind in den Tabellen 5.10 und 5.11 dargestellt. Auffällig ist, dass alle erhobenen Werte oberhalb der erwarteten Werte liegen. Für eine Pufferzeit

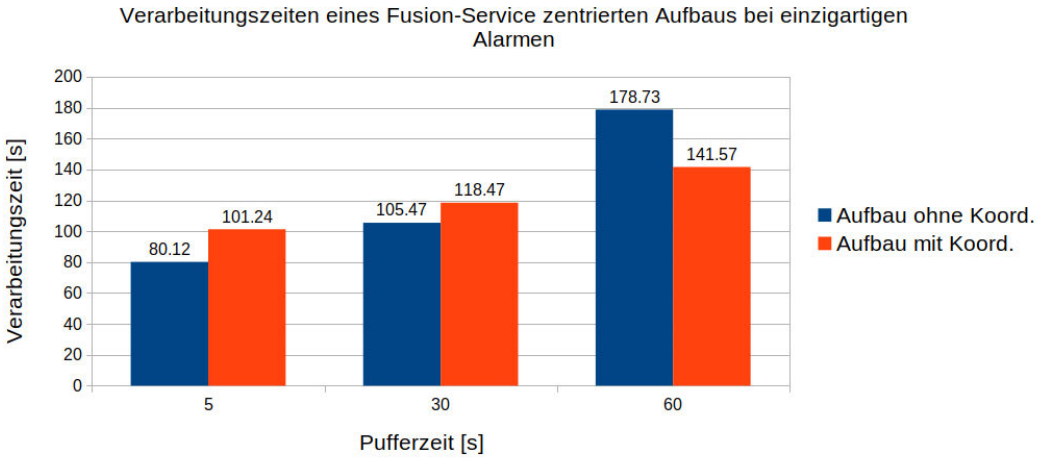

Abbildung 6.8: Gemessene Verarbeitungszeit des Fusion-Service zentrierten Aufbaus bei einzigartigen Alarmen

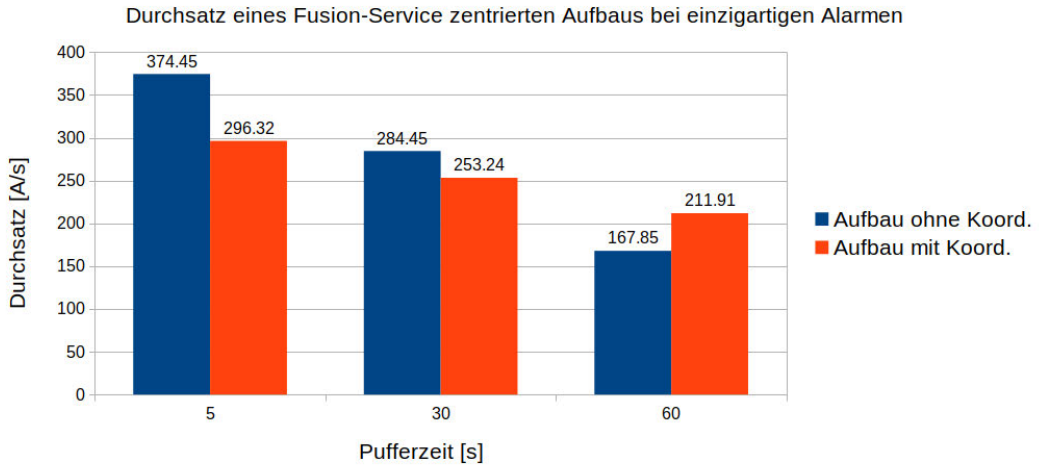

Abbildung 6.9: Gemessener Durchsatz des Fusion-Service zentrierten Aufbaus bei einzigartigen Alarmen

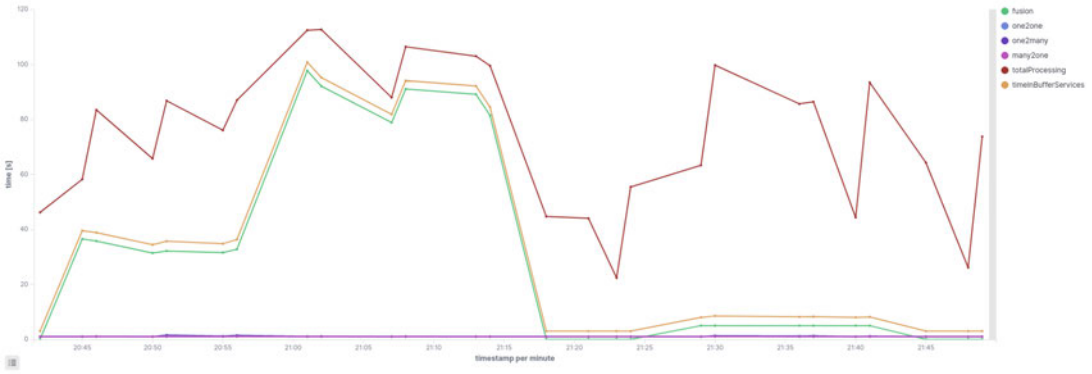

Abbildung 6.10: Zwei nacheinander durchgeführte Messreihen zur Verdeutlichung des Unterschieds zwischen der Verwendung eines Koordinationsservices und keiner Verwendung dessen

wurden Verarbeitungsdauern von 67,5s und 68s angenommen. Die gemessenen Werte liegen allerdings bei 80,12s und 101,24s. Für eine Pufferzeit von 30s wurden die Werte 92,5s und 93s erwartet, die gemessenen Werte lagen bei 105,47s und 118,47s. Besonders auffällig sind allerdings die Werte der Messungen mit einer Pufferzeit von 60s. Dabei liegt der Wert der Messung ohne Koordinationsservice bei 178,73s und damit 37,16s über dem Wert der Messung mit Koordinationsservice, deren Wert bei 141,57s liegt. Dieses Verhalten ist allerdings durch einen Ausreißer in den Messungen erklärbar, da eine Messung eine Verarbeitungszeit von 280.5s aufweist, während die anderen beiden Messungen mit der gleichen Konfiguration Werte im Bereich von 126s bis 129s aufweisen. In diesem Fall liegen die gemessenen Werte allerdings nur knapp über dem erwarteten Wert von 122,5s. Wohingegen die Verarbeitungszeit des Aufbaus mit Koordinationsservice mit einem Wert von 141,51 deutlich über dem erwarteten Wert von 123s liegt.
Da sich der Durchsatz aus den ermittelten Verarbeitungszeiten berechnet, wird nicht genauer auf die Werte eingegangen. Die in Abbildung 6.9 dargestellten Werte weisen die gleichen Besonderheiten auf, wie die der gemessenen Verarbeitungszeiten.

6.2.4 Zusammenfassung und Interpretation der Ergebnisse

Nach der Durchführung der Messungen ist aufgefallen, dass der Fusion-Service unter Verwendung eines Koordinationsservices nicht den Flaschenhals der Verarbeitung bildet, sondern der Service welcher die Alarme an den Elastic-Stack versendet. Anhand der

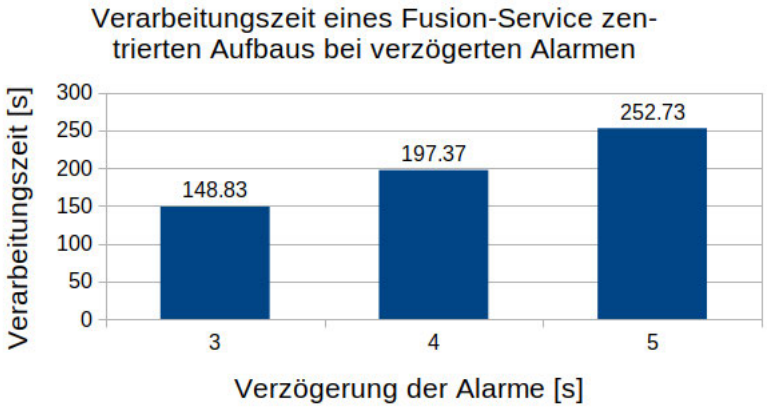

Abbildung 6.11: Verarbeitungszeit eines Fusion-Service zentrierten Aufbaus mit verzögerten Alarmen

Abbildung 6.10 kann deutlich erkannt werden, dass die Verarbeitungszeit des Fusion-Services zur Hälfte der abgebildeten Zeit deutlich sinkt. Dies ist damit zu begründen, dass die Messung mit integriertem Koordinationsservice in etwa bei der Hälfte beginnt. Dabei zeigt die hellgrüne Linie die Verarbeitungszeit des Fusion-Services.
Der Umstand, dass die Verarbeitungszeit höher war als erwartet und somit auch der Durchsatz geringer war als erwartet, kann mit einer Überlast des letzten Services begründet werden, welcher die Kommunikation mit dem Elastic-Stack übernimmt.

6.2.5 Maximale Verzögerung

Die Ergebnisse dieser Messreihe können der Tabelle 5.14 entnommen werden. Die Erwartung an die Ergebnisse ist, dass sobald die Pufferzeit der Verzögerung entspricht keine Fusion mehr vorgenommen werden kann, oder der Mechanismus instabil wird.
Abbildung 6.11 zeigt das Laufzeitverhalten der verschiedenen Verzögerungswerte. Die erwarteten Werte können den Formeln 5.9a bis 5.9c entnommen werden und betragen eine Verarbeitungszeit von 147s, 196s und 255s für die jeweiligen Verzögerunsstufen. Die in der Abbildung dargestellten Werte sind für die ersten beiden Verzögerungswerte leicht oberhalb der erwarteten Werte, während die Verarbeitungszeit bei einer Verzögerung von 5s knapp unterhalb des erwarteten Werts liegt.
Abbildung 6.12 zeigt die Reduzierungsfaktoren der jeweiligen Messläufe. Dabei ist in allen Fällen das erwartete Ergebnis eingetreten. In den ersten beiden Fällen liegt der

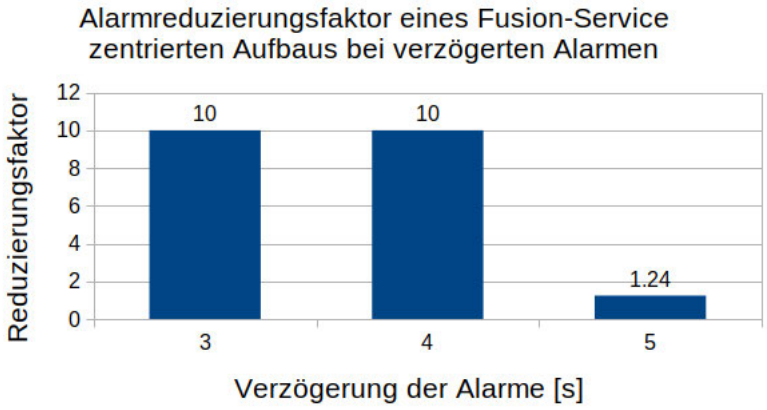

Abbildung 6.12: Reduzierungsfaktor eines Fusion-Service zentrierten Aufbaus mit verzögerten Alarmen

Reduzierungsfaktor bei dem maximal möglichen Wert von zehn, im letzten Fall sinkt der Faktor auf einen Wert von 1,24.

6.2.6 Zusammenfassung und Interpretation der Ergebnisse

Die Werte der Messungen entsprechen in allen Fällen dem erwarteten Verhalten. Allerdings ist aufgefallen, dass die gemessenen Werte der Messreihe mit einer Verzögerung von fünf Sekunden circa drei Sekunden unterhalb des erwarteten Ergebnisses liegen. Dies is im Vergleich zu einer Pufferzeit von fünf Sekunden bemerkenswert. Dieses Verhalten ließe sich nur dadurch erklären, dass die letzten verarbeiteten Alarme das Maximum von zu fusionierenden Alarmen erreichen und somit die Pufferzeit nicht vollends ausgereizt wird. Dies würde allerdings bedeuten dass bei 50 eingehenden Alarmen höchstens 40 ausgehende Alarme resultieren dürften. Dies ist lediglich in einer der drei Messungen der Fall. In den anderen Fällen resultieren 41 und 47 ausgehende Alarme. Auch eine Überschätzung der Verarbeitungszeit der umliegenden Services kann diesen Verhalten nicht erklären, da diese in der Berechnung nicht auftauchen und somit auf einen Wert von null Sekunden approximiert wurden. Eine Unterschätzung würde allerdings in den Messungen mit drei und vier Sekunden Verzögerung die geringe Abweichung zu den erwarteten Werten erklären.
Das Sinken des Reduzierungsfaktors in der Messreihe mit einer Verzögerung von fünf

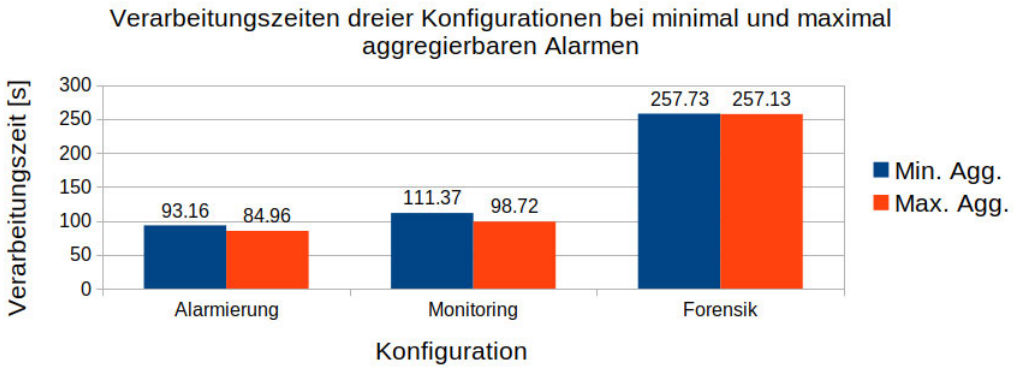

Abbildung 6.13: Verarbeitungszeit dreier verschiedener Systemkonfigurationen bei minimal und maximal aggregierbarer Alarme.

Sekunden ist dadurch erklärbar, dass die eingehenden Alarme nicht mehr vollständig von der Verarbeitungskette fusioniert wurden.

6.3 Vergleich dreier Konfigurationen

Die in diesem Abschnitt betrachteten Ergebnisse stellen drei verschiedene Konfigurationsmöglichkeiten für verschiedene Anwendungsfälle gegenüber und vergleichen diese anhand der Verarbeitungszeit, der Alarmreduzierung und des Durchsatzes. Die Konfigurierten Werte können in Tabelle 5.15 eingesehen werden.

6.3.1 Betrachtung der Ergebnisse

Die Ergebnisse der Messungen können den Tabellen 5.16 und 5.17 entnommen werden. Die Formel 5.11a beschreibt die erwartete Laufzeit der Konfigurationen für den Fall der maximalen Aggregation, welche für alle drei Konfigurationen als konstant angenommen wird. Die erwarteten Werte für die minimale Aggregation können den Formeln 5.11b bis 5.11d entnommen werden und betragen 80s, 100s und 300s.
Abbildung 6.13 stellt die Laufzeiten der drei Konfigurationen im Fall der maximalen und minimalen Aggregation gegenüber. Abbildung 6.14 zeigt den Reduzierungsfaktor im Fall der maximalen Aggregation. Das erwartete Verhalten der drei Konfigurationen ist, dass

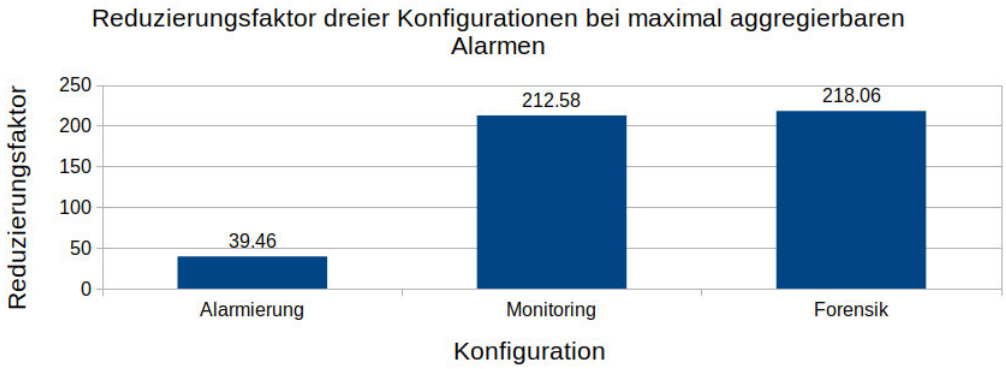

Abbildung 6.14: Reduzierungsfaktor dreier verschiedener Systemkonfigurationen bei maximal aggregierbarer Alarme.

die Alarmierungskonfiguration geringere Laufzeiten als die anderen beiden Konfigurationen aufweist, dabei allerdings ein geringerer Reduzierungsfaktor resultiert. Im Fall der Forensik-Konfiguration wird erwartet, dass der Reduzierungsfaktor maximal ist, die Laufzeit im schlechtesten Fall allerdings auch die höchste ist. Die Monitoring-Konfiguration bildet in etwa den Mittelwert zwischen den anderen beiden Konfigurationen, ist aber anhand der aktuellen Konfiguration näher an den Werten der Alarmierung (siehe Tabelle 5.15).
Die dargestellten Messergebnisse bestätigen die geschilderten Erwartungen dahingehend, dass die Alarmierung sowohl die geringste Laufzeit im Fall der minimalen Aggregation als auch den geringsten Wert in der Alarmreduzierung aufweist. Auffällig ist allerdings, dass keine der drei Konfigurationen im Fall der maximalen Aggregation Werte in nahe der erwarteten Verarbeitungszeit von 68s liefert. Die Werte liegen eher im Bereich der erwarteten Werte für die minimale Aggregation. Diese wiederum liegen im Fall der Alarmierung- und Monitoring-Konfiguration leicht über den erwarteten Werten.
Besonders auffällig ist allerdings, dass die Forensik-Konfiguration für die maximale und minimale Aggregation die gleichen Laufzeitwerte liefert und dabei keine stärkere Reduzierung aufweist als die Monitoring-Konfiguration. Zusätzlich liegt der Wert bei der minimalen Aggregation deutlich unter dem erwarteten Wert von 300s. In Bezug auf die Alarmreduzierung liefert die Alarmierungskonfiguration die geringsten Werte, welche allerdings deutlich unter dem erwarteten Wert liegen. Der maximal mögliche Wert wäre in diesem Fall ein Reduzierungsfaktor von 500 gewesen, während die Monitoring- und Forensik-Konfiguration jeweils einen maximalen Wert von 1000 hätten liefern können.

6.3.2 Zusammenfassung und Interpretation der Ergebnisse

Die drei geschilderten Konfigurationen haben in allen Fällen schwächere Werte geliefert als erwartet. Dies ist teilweise dadurch zu erklären, dass selbst bei einer Simulationszeit von 60s eine leichte Überlast des Systems durch eine Alarmrate von 500 A/s vorliegt. Um diese These zu überprüfen müsste die gleiche Messung erneut durchgeführt werden mit einer Alarmrate von 250 A/s. Die gemessenen Verarbeitungszeiten der Forensik-Konfiguration kann zumindest im Fall der maximalen Aggregation durch einen Ausreißer in einer der Messungen erklärt werden, welcher eine Verarbeitungszeit von 300s aufwies. Die anderen beiden Messungen lieferten die Werte 231,1s und 246s. Damit würde die Verarbeitungszeit im Fall der Forensik-Konfiguration ein ähnliches Verhältnis zwischen den Verarbeitungszeiten bei minimaler und maximaler Aggregation wie die Verarbeitungszeiten der anderen beiden Konfigurationen abbilden. Fraglich bleibt allerdings, weshalb die Konfigurationen einen so geringen Reduzierungsfaktor aufweisen. Da dieser im direkten Zusammenhang zur Laufzeit im Fall der maximalen Aggregation steht, sollte dieser Umstand zukünftig noch einmal genauer untersucht werden.

6.4 Schlussfolgerungen der Messungen

Die geschilderten Messungen liefern wertvolle Einblicke in das Verhalten des Gesamtsystems. Somit konnte bestimmt werden, dass der für die Messungen verwendete Prototyp von einer Alarmrate von 500 A/s leicht und von einer Alarmrate von 1000 A/s stark überlastet wird. Zusätzlich konnte ein Bottleneck in der Verarbeitung aufgedeckt werden, welcher auf der Kommunikation mit dem Elastic-Stack beruht. Das Konzept des Koordinationsservices konnte erfolgreich verifiziert werden und das Verhalten des Fusion-Services bei minimaler und maximaler Aggregationsmöglichkeit, sowie bei verzögerten Alarmen konnte genauer betrachtet werden.
Auffällig war allerdings, dass die Messreihen mit einem erwarteten maximalen Reduzierungsfaktor von 1000 kein einziges mal an diesen Wert herankamen. Dies lässt zwei mögliche Schlussfolgerungen zu. Einerseits kann das System an sich noch Fehler in der Verarbeitung aufweisen. Auf der anderen Seite könnte auch ein Fehler im Simulations-Service enthalten sein, welcher dazu führt dass die als maximal aggregierbaren Alarme nicht korrekt aggregiert werden können.
Nichtsdestotrotz konnte anhand der gewählten Messreihen verifiziert werden, dass verschiedene Konfigurationen des Systems möglich sind. Durch die gewählten Technologien

wäre für weitere Messungen ein Testaufbau mit einem kompletten Deployment interessant, um festzustellen ob eine Hardware-Limitierung vorlag und somit bessere Werte in den Messreihen erreicht werden können.

6.5 Abgleich mit den Anforderungen

Im Folgenden werden die in den vorangegangenen Kapiteln geschilderten Ergebnisse mit den in Kapitel 3 definierten Zielen und Anforderungen abgeglichen.

6.5.1 Aggregation

Durch die vorgenommenen Messungen anhand des implementierten Prototypen konnte nachgewiesen werden, dass die betrachtete Korrelationskette Alarmreduzierung ermöglicht. Allerdings ist diese in einigen Fällen schwächer ausgefallen als erwartet. Dennoch kann das System je nach Anwendungsfall konfiguriert werden, sodass potentiell höhere Werte möglich wären. Bei einer Weiterentwicklung des Systems sollte das Verhalten der Alarmreduzierung genauer betrachtet werden, um sicherzustellen dass das in den Messungen erkannte Verhalten korrekt ist. Ebenfalls könnte in diesem Fall ein Cloud-Deployment Auskunft darüber geben, ob Hardware basierte Verzögerungen dazu geführt haben, dass in einigen Fällen keine Aggregation durchgeführt wurde.

6.5.2 Laufzeit, Messbarkeit und Darstellung

In Bezug auf die Laufzeit konnte gut dargestellt werden, welche Fälle zu einer maximalen bzw. minimalen Laufzeit führen würden. Zusätzlich versieht das System alle verarbeiteten Alarme mit Zeitstempeln, sodass mit Hilfe von Kibana Dashboards erzeugt werden können, welche die Anforderungen an die Messbarkeit und die Darstellung durch das System abdecken. Neben dem Verhalten des Systems bei einer Überlast konnte auch gezeigt werden, dass das System in verschiedenen Konfigurationen verschiedene Laufzeiten aufweist und damit der Konfigurationsanforderung genügt.

6.5.3 Flexibilität und Erweiterbarkeit

Die implementierten Services haben Abhängigkeiten in den unterstützten Formaten, da die nachfolgenden Services immer das Format des vorangegangen Services verarbeiten können müssen. Zusätzlich beinhaltet dieses Format dann teilweise das Format des wiederum vorangegangenen Services, wodurch der letzte Service alle vorangegangenen Formate unterstützen muss. Um den Verwaltungsaufwand dieser Formate zu minimieren könnten alle Formate in eine Bibliothek auslagert werden, welche in jedem Service importiert wird. Dies würde ermöglichen im Nachhinein an einer Stelle neue Formate einzuführen oder bestehende Formate zu verändern und anschließend lediglich in jedem Service die Versionsnummer der Bibliothek zu erhöhen. Dies würde zusätzlich die Menge von geteiltem Sourcecode reduzieren.
Grundsätzlich ist die Erweiterung der Korrelationskette durch die Nutzung von Messagequeues gewährleistet. Dazu müssen nur Input- und Output-Queue des jeweiligen Services konfiguriert werden und sichergestellt werden, dass der nachfolgende Service das Output-Format unterstützt. Durch die Orchestrierung der Services bspw. durch Docker-Compose können flexibel weitere Services wie zum Beispiel Koordinationsservices zwischen die anderen Services geschaltet werden und somit die Verarbeitungskette erweitern.

6.5.4 Skalierbarkeit

Die in Abschnitt 4.3.2 beschriebene Architektur des Systems beinhaltet die Möglichkeit der horizontalen Skalierung der Services. Dabei wurde ebenfalls erläutert wie die Buffer-based Services mit Hilfe eines Koordinationsservices skaliert werden können. Dieser ist ebenfalls horizontal skalierbar, da dieser Service ausschließlich einzelne Alarme betrachtet und koordiniert (one-by-one).
Zusätzlich können die Buffer-based Services mit Hilfe von Multithreading vertikal skaliert werden. Dies ist im Fall des Proof of Concepts allerdings nur bedingt sinnvoll, da die Services auf einem einzigen Host betrieben wurden und somit eine starke Abhängigkeit zur Leistungsfähigkeit der zugrundeliegenden Hardware besteht.
Die Ergebnisse der vorangegangenen Messungen aus Abschnitt 6.2 zeigen dabei deutlich, dass eine Lastreduzierung stattfindet, wenn ein Koordinationsservice eingesetzt wird. Dies könnte für die gesamte Architektur umgesetzt werden, sofern ausreichend Hardware-Ressourcen zur Verfügung stehen, bspw. durch ein Deployment in einer Cloud-Umgebung.

7 Abschluss

In diesem Kapitel werden die Ergebnisse dieser Arbeit zusammengefasst und ein Fazit gezogen. Abschließend werden Themengebiete für zukünftige Arbeiten aufgeführt.

7.1 Zusammenfassung

In dieser Arbeit wurden Anforderungen an eine skalierbare Alarmkorrelation hergeleitet und prototypisch implementiert. Dabei wurden verschiedene Architekturen verglichen und gegeneinander abgewogen. Zusätzlich wurde das Konzept eines direkt in der Verarbeitungskette integrierten Frühwarnsystems entworfen und diskutiert. Ebenfalls wurden Monitoring Services eingebunden, welche die Visualisierung der verarbeiteten Daten übernehmen. Es wurden Möglichkeiten beschrieben, das bestehende System mit leichten Anpassungen ebenfalls als Analysewerkzeug für bereits vergangene Angriffe nutzbar zu machen.
Zur flexiblen Einsetzbarkeit des entwickelten Prototypen der Korrelationskette wurden Konfigurationsmöglichkeiten dargestellt. Außerdem wurde das Konzept eines Frühwarnsystems innerhalb der Korrelationskette hergeleitet und beschrieben, wie der Simulationservice dazu verwendet werden kann, vergangene Angriffsszenarien zu simulieren.
Neben der Entwicklung einer skalierbaren, erweiterbaren und flexibel einsetzbaren Architektur, wurde die Implementierung der Services beschrieben. Anschließend wurden die Ergebnisse eines iterativen Entwicklungsprozesses dargestellt und vorgenommene Optimierungen beschrieben.
Im Verlauf der Arbeit wurden Methoden herausgestellt, mithilfe derer das Verhalten und die Performanz des Systems gemessen werden konnten. Die hergeleiteten Messstrategien wurden anschließend durchgeführt, ausgewertet und mit den Anforderungen an das System abgeglichen.

7.2 Fazit

Zusammenfassend konnten die in Kapitel 3 beschriebenen Ziele erreicht und die Anforderungen umgesetzt werden. Die Ergebnisse der in Kapitel 5 durchgeführten Messungen lieferten teilweise unerwartete Ergebnisse, da die Verarbeitungszeit länger war als angenommen und der Alarmreduzierungsfaktor geringer ausgefallen ist als in der Abschätzung. Dennoch konnte nachvollzogen werden, dass die zentralen Ziele der Alarmkorrelation durch den entwickelten Prototypen abgedeckt werden:
Es werden die genannten Aspekte dreier verschiedener Einsatzgebiete umgesetzt. Dabei ist das System in der Verarbeitungszeit konfigurierbar und stellt Ergebnisse anhand des Elastic-Stacks dar. Des Weiteren wurde das Konzept eines Frühwarnsystems hergeleitet, welches durch kleinere Anpassungen der Services implementiert werden kann. Zusätzlich wurde erläutert, wie das bestehende System als forensisches Analysewerkzeug verwendet werden kann.
Es konnte anhand der vorgenommenen Messungen nachgewiesen werden, dass die Einführung eines Koordinationsservices zu einer Entlastung des nachgeschalteten Services führt und somit den Engpass in der Skalierbarkeit behebt. Die Verwendung von Docker Containern ermöglicht das Deployment des Systems in einer Cloud-Umgebung. Dies ermöglicht neben der horizontalen Skalierung auch die vertikale Skalierung der zugrundeliegenden Hardware.
Durch die Verwendung eines Meta-Formats ist die Integration weiterer Sensoren sichergestellt und durch die Verwendung einer Message Queue können weitere Services zur Verarbeitungskette hinzugefügt werden.

7.3 Ausblick

Der in dieser Arbeit beschriebene Ansatz der Alarmkorrelation und der so entstandene Prototyp können als Basis für zukünftige Arbeiten verwendet werden. Besonders interessant wäre die Durchführung der in Kapitel 5 beschriebenen *Messungen in einer Cloud-Umgebung*. Dabei könnte eine *vollständige Skalierung* des Systems getestet werden, indem Koordinationsservices vor jeden anderen Service geschaltet werden. Anschließend daran könnten dann *automatische Skalierungsmaßnahmen* eingeführt werden, welche im Fall von Lastspitzen weitere Services starten und für den jeweiligen Koordinationsservice zur Verfügung stellen.
Eine Erweiterung der Korrelationskette um einen *Multi-step-Correlation Service* wäre

denkbar, um eine Zuordnung zwischen den verarbeiteten Alarmen und Angriffsszenarien auf höherer Ebene vorzunehmen. Diese wäre gerade unter Betrachtung von *Realdaten* interessant, welche von dem Simulationsservice durch kleinere Anpassungen aus einer Datei oder einer Datenbank heraus in das System eingespielt werden können. Die Zusätzliche Erweiterung um einen *Alert Verification Service* könnte bei einem Einsatz in einer Produktivumgebung dazu beitragen die Anzahl der Fehlalarme zu reduzieren.
Auch unabhängig von der Erweiterung der Korrelationskette, könnten weitere *Optimierungen auf Service-Ebene* vorgenommen werden. Bspw. könnte die Implementationssprache eines Services geändert werden, um die Auswirkungen auf die Performanz zu untersuchen. Eine weitere Möglichkeit wäre die Implementierung und der nachfolgende *Vergleich von Algorithmen* innerhalb der Korrelationskette. So könnte pro Service entschieden werden, welcher Algorithmus für die jeweilige Aufgabe verwendet wird. Des Weiteren könnten auch *Optimierungen der Heuristik des One2One Services* vorgenommen werden.
Um potentiell eine Leistungssteigerung der Verarbeitungskette zu erreichen, könnte ein *RabbitMQ-Wrapper* implementiert werden, welcher eine *Bulk-API* zur Verfügung stellt, um den Kommunikationsaufwand der Services stark zu vermindern. Ebenfalls könnte eine *feste Puffergröße pro Service* eingeführt werden. Diese Änderung würde das Risiko minimieren, dass überlastete Services aufgrund fehlenden Arbeitsspeichers abstürzen. Zusätzlich könnte somit eine Laufzeitabschätzung des Systems erreicht werden, da es dann einen maximalen Wert für die im Puffer enthaltenen Alarme gäbe.
Die in dieser Arbeit beschriebene Architektur bietet interessante Möglichkeiten für weitere Forschungen und kann als Grundlage für zukünftige Arbeiten verwendet werden. Der entwickelte Prototyp kann teilweise als Rahmen für weitere Messungen und Vergleiche auf dem Gebiet der Alarmkorrelation herangezogen werden.

Literaturverzeichnis

[1] Bundesamt für Sicherheit in der Informationstechnik (BSI), "Die Lage der IT-Sicherheit in Deutschland 2021," Tech. Rep., Aug. 2022. [Online]. Available: https://www.bsi.bund.de/DE/Service-Navi/Publikationen/Lagebericht/lagebericht_node.html

[2] I. Ghafir, M. Hammoudeh, V. Prenosil, L. Han, R. Hegarty, K. Rabie, and F. J. Aparicio-Navarro, "Detection of advanced persistent threat using machine-learning correlation analysis," *Future Generation Computer Systems*, vol. 89, pp. 349–359, Dec. 2018. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0167739X18307532

[3] E. Cole, "The Changing Threat," in *Advanced Persistent Threat*. Elsevier, 2013, pp. 3–26. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/B9781597499491000012

[4] F. Valeur, G. Vigna, C. Kruegel, and R. Kemmerer, "Comprehensive approach to intrusion detection alert correlation," *IEEE Transactions on Dependable and Secure Computing*, vol. 1, no. 3, pp. 146–169, Jul. 2004. [Online]. Available: http://ieeexplore.ieee.org/document/1366134/

[5] S. Bhatt, P. K. Manadhata, and L. Zomlot, "The Operational Role of Security Information and Event Management Systems," *IEEE Security Privacy*, vol. 12, no. 5, pp. 35–41, Sep. 2014.

[6] A. Thiele, "Konfiguration und Evaluation eines Open Source Security Information and Event Management Systems," Bachelor's Thesis, Hamburg University of Applied Science, 2018.

[7] ——, "Security information and event management systems," Grundseminar, Hamburg University of Applied Science, 2019. [Online]. Available: https://users.informatik.haw-hamburg.de/~ubicomp/projekte/master2018-gsem/Thiele_A/bericht.pdf

[8] ——, "Automatisierte Erzeugung von Korrelationsregeln-Unterstützung der Arbeit von Sicherheitsanalysten in einem SOC," Hauptseminar, Hamburg University of Applied Science, 2019. [Online]. Available: https://users.informatik.haw-hamburg.de/~ubicomp/projekte/master2019-hsem/Thiele/bericht.pdf

[9] ——, "Aufbau eines IDS-Systems auf Basis des Valeur Korrelationskonzepts," Hochschule für angewandte Wissenschaften Hamburg, Grundprojekt, Jan. 2021.

[10] ——, "One-to-One - Aggregation als Teil einer verbesserten SIEM-Funktionalität," Hochschule für angewandte Wissenschaften Hamburg, Hauptprojekt, May 2021.

[11] Bushra A. Alahmadi, Louise Axon, and Ivan Martinovic, "99% False Positives: A Qualitative Study of SOC Analysts Perspectives on Security Alarms," Aug. 2022. [Online]. Available: https://www.usenix.org/conference/usenixsecurity22/presentation/alahmadi

[12] Carson Zimmermann, *Ten Strategies of a World-Class Cybersecurity Operations Center*. [Online]. Available: https://www.mitre.org/sites/default/files/publications/pr-13-1028-mitre-10-strategies-cyber-ops-center.pdf

[13] P. R. M. Jagadeesan, "A framework design to improve and evaluate the performance of security operation center (SOC)," Ph.D. dissertation, Dublin, National College of Ireland, Jan. 2020. [Online]. Available: http://norma.ncirl.ie/4179/

[14] D. Kelley and R. Moritz, "Best Practices for Building a Security Operations Center," *Information Systems Security*, vol. 14, no. 6, pp. 27–32, Jan. 2006. [Online]. Available: http://www.tandfonline.com/doi/abs/10.1201/1086.1065898X/45782.14.6.20060101/91856.6

[15] a. sopan, M. Berninger, M. Mulakalur, and R. Katakam, "Machine Learning for Security Operation Center," Open Science Framework, Preprint, Nov. 2018. [Online]. Available: https://osf.io/agsvz

[16] H. Chindove and D. Brown, "Adaptive Machine Learning Based Network Intrusion Detection," in *Proceedings of the International Conference on Artificial Intelligence and Its Applications*. Virtual Event Mauritius: ACM, Dec. 2021, pp. 1–6. [Online]. Available: https://dl.acm.org/doi/10.1145/3487923.3487938

[17] T. Ban, N. Samuel, T. Takahashi, and D. Inoue, "Combat Security Alert Fatigue with AI-Assisted Techniques," in *Cyber Security Experimentation and Test*

Workshop. Virtual CA USA: ACM, Aug. 2021, pp. 9–16. [Online]. Available: https://dl.acm.org/doi/10.1145/3474718.3474723

[18] N. Gupta, I. Traore, and P. M. F. de Quinan, "Automated Event Prioritization for Security Operation Center using Deep Learning," in *2019 IEEE International Conference on Big Data (Big Data)*. Los Angeles, CA, USA: IEEE, Dec. 2019, pp. 5864–5872. [Online]. Available: https://ieeexplore.ieee.org/document/9006073/

[19] K. Gade, S. C. Geyik, K. Kenthapadi, V. Mithal, and A. Taly, "Explainable AI in Industry," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. Anchorage AK USA: ACM, Jul. 2019, pp. 3203–3204. [Online]. Available: https://dl.acm.org/doi/10.1145/3292500.3332281

[20] F. Xu, H. Uszkoreit, Y. Du, W. Fan, D. Zhao, and J. Zhu, "Explainable AI: A Brief Survey on History, Research Areas, Approaches and Challenges," in *Natural Language Processing and Chinese Computing*, ser. Lecture Notes in Computer Science, J. Tang, M.-Y. Kan, D. Zhao, S. Li, and H. Zan, Eds. Cham: Springer International Publishing, 2019, pp. 563–574.

[21] Bundesamt für Sicherheit in der Informationstechnik (BSI), "Frühwarnung vor IT-Angriffen zum Schutz von Informationsinfrastrukturen;Teilprojekt: Weiterentwicklung, Erprobung und Erforschung neuer Frühwarnmethoden - Zeitnahe Information und Alarmierung über neue Sicherheitslagen im deutschen Internet Integration verbesserter Aggregationsverfahren," 2006.

[22] S. Salah, G. Maciá-Fernández, and J. E. Díaz-Verdejo, "A model-based survey of alert correlation techniques," *Computer Networks*, vol. 57, no. 5, pp. 1289–1317, Apr. 2013. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S1389128612004124

[23] A. Valdes and K. Skinner, "Probabilistic Alert Correlation," in *Recent Advances in Intrusion Detection*, G. Goos, J. Hartmanis, J. van Leeuwen, W. Lee, L. Mé, and A. Wespi, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, vol. 2212, pp. 54–68. [Online]. Available: http://link.springer.com/10.1007/3-540-45474-8_4

[24] K. Lee, J. Kim, K. H. Kwon, Y. Han, and S. Kim, "DDoS attack detection method using cluster analysis," *Expert Systems with Applications*, vol. 34, no. 3, pp. 1659–1665, 2008. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0957417407000395

[25] A. S. Sethi, Y. Raynaud, and F. Faure-Vincent, Eds., *Integrated Network Management IV*. Boston, MA: Springer US, 1995. [Online]. Available: http://link.springer.com/10.1007/978-0-387-34890-2

[26] S. H. Ahmadinejad and S. Jalili, "Alert Correlation Using Correlation Probability Estimation and Time Windows," in *2009 International Conference on Computer Technology and Development*, vol. 2, Nov. 2009, pp. 170–175.

[27] J. Ma, Z.-t. Li, and W.-m. Li, "Real-Time Alert Stream Clustering and Correlation for Discovering Attack Strategies," in *2008 Fifth International Conference on Fuzzy Systems and Knowledge Discovery*, vol. 4, Oct. 2008, pp. 379–384.

[28] Z. Lin, S. Li, and Y. Ma, "Real-Time Intrusion Alert Correlation System Based on Prerequisites and Consequence," in *2010 6th International Conference on Wireless Communications Networking and Mobile Computing (WiCOM)*, Sep. 2010, pp. 1–5.

[29] B. Gruschke, "Integrated event management: Event correlation using dependency graphs." 1998.

[30] Q. Zheng and Y. Qian, "An Event Correlation Approach Based on the Combination of IHU and Codebook," in *Computational Intelligence and Security*, D. Hutchison, T. Kanade, J. Kittler, J. M. Kleinberg, F. Mattern, J. C. Mitchell, M. Naor, O. Nierstrasz, C. Pandu Rangan, B. Steffen, M. Sudan, D. Terzopoulos, D. Tygar, M. Y. Vardi, G. Weikum, Y. Hao, J. Liu, Y.-P. Wang, Y.-m. Cheung, H. Yin, L. Jiao, J. Ma, and Y.-C. Jiao, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, vol. 3802, pp. 757–763. [Online]. Available: http://link.springer.com/10.1007/11596981_111

[31] D. Ourston, S. Matzner, W. Stump, and B. Hopkins, "Applications of hidden Markov models to detecting multi-stage network attacks," in *36th Annual Hawaii International Conference on System Sciences, 2003. Proceedings of The*. Big Island, HI, USA: IEEE, 2003, p. 10 pp. [Online]. Available: http://ieeexplore.ieee.org/document/1174909/

[32] M. Steinder and A. Sethi, "Probabilistic fault localization in communication systems using belief networks," *IEEE/ACM Transactions on Networking*, vol. 12, no. 5, pp. 809–822, Oct. 2004.

[33] B. Zhu and A. A. Ghorbani, "Alert Correlation for Extracting Attack Strategies," p. 15, 2006.

[34] S. Salah, G. Maciá-Fernández, and J. E. Díaz-Verdejo, "A model-based survey of alert correlation techniques," *Computer Networks*, vol. 57, no. 5, pp. 1289–1317, Apr. 2013. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S1389128612004124

[35] S. O. Al-Mamory and H. Zhang, "IDS alerts correlation using grammar-based approach," *Journal in Computer Virology*, vol. 5, no. 4, pp. 271–282, Nov. 2009. [Online]. Available: http://link.springer.com/10.1007/s11416-008-0103-3

[36] R. Cronk, P. Callahan, and L. Bernstein, "Rule-based expert systems for network management and operations: An introduction," *IEEE Network*, vol. 2, no. 5, pp. 7–21, Sep. 1988.

[37] R. Smith, N. Japkowicz, M. Dondo, and P. Mason, "Using Unsupervised Learning for Network Alert Correlation," in *Advances in Artificial Intelligence*, ser. Lecture Notes in Computer Science, S. Bergler, Ed. Berlin, Heidelberg: Springer, 2008, pp. 308–319.

[38] E. Knapp, "Introduction to Industrial Network Security," in *Industrial Network Security*. Elsevier, 2011, pp. 31–54. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/B9781597496452000033

[39] J. Andress and S. Winterfeld, "Computer Network Exploitation," in *Cyber Warfare*. Elsevier, 2014, pp. 169–179. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/B978012416672100009X

[40] S. Haas and M. Fischer, "On the alert correlation process for the detection of multi-step attacks and a graph-based realization," *ACM SIGAPP Applied Computing Review*, vol. 19, no. 1, pp. 5–19, Apr. 2019. [Online]. Available: https://dl.acm.org/doi/10.1145/3325061.3325062

[41] F. C. Ortmann, "Temporal and Spatial Alert Correlation for the Detection of Advanced Persistent Threats," Master's thesis, University of Hamburg, Hamburg, Jul. 2019.

[42] W. Li and S. Tian, "An ontology-based intrusion alerts correlation system," *Expert Systems with Applications*, vol. 37, no. 10, pp. 7138–7146, Oct. 2010. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S095741741000271X

[43] A. Sadighian, J. M. Fernandez, A. Lemay, and S. T. Zargar, "ONTIDS: A Highly Flexible Context-Aware and Ontology-Based Alert Correlation Framework," in

Foundations and Practice of Security, ser. Lecture Notes in Computer Science, J. L. Danger, M. Debbabi, J.-Y. Marion, J. Garcia-Alfaro, and N. Zincir Heywood, Eds. Cham: Springer International Publishing, 2014, pp. 161–177.

[44] Robert Johns, "Visualisierung von Lagebildinformationen innerhalb eines Security Operation Centers Ausarbeitung zum Masterseminar GSM, WS 14/15," Hochschule für angewandte Wissenschaften Hamburg, Hamburg, Tech. Rep., 2014.

[45] I. Kotenko and E. Novikova, "Visualization of Security Metrics for Cyber Situation Awareness," in *2014 Ninth International Conference on Availability, Reliability and Security*, Sep. 2014, pp. 506–513.

[46] D. Inc., "Docker Homepage," May 2022. [Online]. Available: https://www.docker.com/

[47] ——, "Overview of Docker Compose," May 2022. [Online]. Available: https://docs.docker.com/compose/

[48] E. B.V., "Elastic Stack: Elasticsearch, Kibana, Beats & Logstash," Jul. 2022. [Online]. Available: https://www.elastic.co/elastic-stack

[49] ——, "Elasticsearch: The Official Distributed Search & Analytics Engine," May 2022. [Online]. Available: https://www.elastic.co/elasticsearch

[50] ——, "Why free and open? | Elastic." [Online]. Available: https://www.elastic.co/about/free-and-open

[51] ——, "Kibana: Explore, Visualize, Discover Data," Jul. 2022. [Online]. Available: https://www.elastic.co/kibana

[52] VMware, "Messaging that just works — RabbitMQ," May 2022. [Online]. Available: https://www.rabbitmq.com/

[53] ——, "Documentation: Table of Contents — RabbitMQ," May 2022. [Online]. Available: https://www.rabbitmq.com/documentation.html

[54] O. I. S. F. (OISF), "Suricata Homepage," May 2022. [Online]. Available: https://suricata.io/

[55] ——, "Suricata User Guide — Suricata 6.0.5 documentation," May 2022. [Online]. Available: https://suricata.readthedocs.io/en/suricata-6.0.5/

[56] ——, "Suricata Eve JSON Format — Suricata 6.0.5 documentation." [Online]. Available: https://suricata.readthedocs.io/en/suricata-6.0.5/output/eve/eve-json-format.html

[57] "Fully Managed Container Solution – Amazon Elastic Container Service (Amazon ECS) - Amazon Web Services," Jul. 2022. [Online]. Available: https://aws.amazon.com/ecs/

A Konfigurationsdateien der Services

```
version: '3'
services:
  #  simulation-component:
  #    image: "simulation-component:latest"
  #    ports:
  #      - "8080:8080"
  #    networks: [ "es-kibana" ]
  #    extra_hosts:
  #      - "host.docker.internal:host-gateway"
  #    environment:
  #      - MQ_HOST=${MQ_HOST}
  #      - OUTPUT_QUEUE_NAME=sample_alerts

  normalization-service:
    image: "normalization-service:latest"
    network_mode: "host"
    environment:
      - MQ_HOST=${MQ_HOST}
      - INPUT_QUEUE_NAME=sample_alerts
      - OUTPUT_QUEUE_NAME=normalized_alerts
      - THREAD_COUNT=2

  preprocessing-service:
    image: "preprocessing-component:latest"
    network_mode: "host"
    environment:
      - MQ_HOST=${MQ_HOST}
      - INPUT_QUEUE_NAME=normalized_alerts
      - OUTPUT_QUEUE_NAME=preprocessed_alerts
      - THREAD_COUNT=2

  coordination-service-for-fusion:
```

```
33      image: "coordination-service:latest"
34      network_mode: "host"
35      environment:
36        - MQ_HOST=${MQ_HOST}
37        - COORDINATING_FOR_SERVICE=fusion
38        - INPUT_QUEUE_NAME=preprocessed_alerts
39        - OUTPUT_QUEUE_NAMES=coord_fused1,coord_fused2
40        - THREAD_COUNT=2
41
42    fusion-service:
43      image: "fusion-component:latest"
44      network_mode: "host"
45      environment:
46        - MQ_HOST=${MQ_HOST}
47        - INPUT_QUEUE_NAME=preprocessed_alerts
48        - OUTPUT_QUEUE_NAME=fused_alerts
49        - TIME_TO_WAIT_FOR_ALERTS=10
50        - MAX_ALERTS_TO_WAIT_FOR=10
51        - CONSUMER_THREAD_COUNT=2
52
53    one2one-service:
54      image: "one2one-component:latest"
55      network_mode: "host"
56      environment:
57        - MQ_HOST=${MQ_HOST}
58        - INPUT_QUEUE_NAME=fused_alerts
59        - OUTPUT_QUEUE_NAME_ONE2MANY=aggregated_for_one2many_first
60        - OUTPUT_QUEUE_NAME_MANY2ONE=aggregated_for_many2one_first
61        - TIME_TO_WAIT_FOR_ALERTS_SECONDS=10
62        - MAX_ALERTS_TO_WAIT_FOR=10
63        - CONSUMER_THREAD_COUNT=2
64
65    one2many-first:
66      image: "one2many:latest"
67      network_mode: "host"
68      environment:
69        - MQ_HOST=${MQ_HOST}
70        - ES_HOST=${ES_HOST}
71        - IS_FIRST=true
72        - INPUT_QUEUE_NAME=aggregated_for_one2many_first
```

```
73        - OUTPUT_QUEUE_NAME=one2many_alerts
74        - TIME_TO_WAIT_FOR_ALERTS_SECONDS=10
75        - MAX_ALERTS_TO_WAIT_FOR=10
76        - CONSUMER_THREAD_COUNT=2
77
78    many2one-second:
79      image: "many2one:latest"
80      network_mode: "host"
81      environment:
82        - MQ_HOST=${MQ_HOST}
83        - ES_HOST=${ES_HOST}
84        - IS_FIRST=false
85        - INPUT_QUEUE_NAME=one2many_alerts
86        - OUTPUT_QUEUE_NAME=ready_for_monitoring
87        - TIME_TO_WAIT_FOR_ALERTS_SECONDS=10
88        - MAX_ALERTS_TO_WAIT_FOR=10
89        - CONSUMER_THREAD_COUNT=2
90
91    many2one-first:
92      image: "many2one:latest"
93      network_mode: "host"
94      environment:
95        - MQ_HOST=${MQ_HOST}
96        - ES_HOST=${ES_HOST}
97        - IS_FIRST=true
98        - INPUT_QUEUE_NAME=aggregated_for_many2one_first
99        - OUTPUT_QUEUE_NAME=many2one_alerts
100       - TIME_TO_WAIT_FOR_ALERTS_SECONDS=10
101       - MAX_ALERTS_TO_WAIT_FOR=10
102       - CONSUMER_THREAD_COUNT=2
103
104   coordination-service-for-one2many-second:
105     image: "coordination-service:latest"
106     network_mode: "host"
107     environment:
108       - MQ_HOST=${MQ_HOST}
109       - COORDINATING_FOR_SERVICE=one2many_second
110       - INPUT_QUEUE_NAME=many2one_alerts
111       - OUTPUT_QUEUE_NAMES=coordinated1,coordinated2
112       - THREAD_COUNT=2
```

```
113
114   one2many-second:
115     image: "one2many:latest"
116     network_mode: "host"
117     environment:
118       - MQ_HOST=${MQ_HOST}
119       - ES_HOST=${ES_HOST}
120       - IS_FIRST=false
121       - INPUT_QUEUE_NAME=coordinated1
122       - OUTPUT_QUEUE_NAME=ready_for_monitoring
123       - TIME_TO_WAIT_FOR_ALERTS_SECONDS=10
124       - MAX_ALERTS_TO_WAIT_FOR=10
125       - CONSUMER_THREAD_COUNT=2
126
127   one2many-second2:
128     image: "one2many:latest"
129     network_mode: "host"
130     environment:
131       - MQ_HOST=${MQ_HOST}
132       - ES_HOST=${ES_HOST}
133       - IS_FIRST=false
134       - INPUT_QUEUE_NAME=coordinated2
135       - OUTPUT_QUEUE_NAME=ready_for_monitoring
136       - TIME_TO_WAIT_FOR_ALERTS_SECONDS=10
137       - MAX_ALERTS_TO_WAIT_FOR=10
138       - CONSUMER_THREAD_COUNT=2
```

Listing A.1: Beispielhafte Konfiguration der implementierten Services anhand von `docker-compose` und auskommentiertem Simulation-Service

```
1   version: '3'
2 services:
3   elasticsearch:
4     image: elasticsearch:7.6.2
5     ports:
6       - "9200:9200"
7       - "9300:9300"
8     networks: [ "es-kibana" ]
9     environment:
10      - discovery.type=single-node
11  kibana:
```

```
12      image: kibana:7.6.2
13      ports:
14        - "5601:5601"
15      networks: [ "es-kibana" ]
16    rabbitmq:
17      image: rabbitmq:3.9.15-management-alpine
18      ports:
19        - "5672:5672"
20        - "15672:15672"
21      networks: [ "es-kibana" ]
22
23  networks:
24    es-kibana:
```

Listing A.2: Konfigurationsdatei des Elastic-Stacks und RabbitMQ

Erklärung zur selbstständigen Bearbeitung

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

Ort Datum Unterschrift im Original