HOCHSCHULE
MITTWEIDA
University of Applied Sciences

Bachelorarbeit

Entwicklung einer Control-Policy für eine KI-basierte Datenflusssteuerung

Henning Ullrich

2022

Fakultät Angewandte Computer- und Biowissenschaften

Bachelorarbeit

Entwicklung einer Control-Policy für eine KI-basierte Datenflusssteuerung

Autor:
Henning Ullrich

Studiengang:
Angewandte Informatik - IT-Sicherheit

Seminargruppe:
IF19wi2-B

Erstprüfer:
Prof. Dr. Ing. Thomas Beierlein

Zweitprüfer:
Dr. Ing. Jörg Kebbedies

Einreichung:
Mittweida, 22. August 2022

Faculty Applied Computer Sciences and Biosciences

Bachelor Thesis

Development of a control-policy for a AI-based data-flow control

author:
Henning Ullrich

course of studies:
Applied Computer Science - IT Security

seminar group:
IF19wi2-B

first examiner:
Prof. Dr. Ing. Thomas Beierlein

second examiner:
Dr. Ing. Jörg Kebbedies

submission:
Mittweida, August 22th, 2022

Abstract

Die Arbeit beschäftigt sich mit der Entwicklung eines Policy-Konzepts zur Steuerung von Netzwerkverkehr an den Außengrenzen eines digitalen Raums. Um Ressourcen innerhalb des Raums zu schützen, definiert die Autorität des Raums mit Hilfe einer semantischen KI Kommunikationsziele. Diese müssen mit Hilfe des zu entwickelnden Policy-Konzepts den Grenzpunkten des Raumes mitgeteilt und dort umgesetzt werden. Die Bachelorarbeit beschäftigt sich mit der Entwicklung des Policy-Konzepts und einer prototypischen Implementierung eines solchen Grenzpunktes.

Danksagung

Ich bedanke mich bei meinen Betreuern Herrn Dr. Ing. Kebbedies und Herrn Prof. Dr. Ing. Beierlein für die Unterstützung während der Bachelorarbeit. Besonderer Dank gilt Herrn Dr. Ing. Kebbedies für die inhaltlichen Reviews der Bachelorarbeit während des Schreibprozesses. Weiterer Dank gilt meiner Familie für die Korrektur der Grammatik und Rechtschreibung. Des weiteren bedanke ich mich bei dem Ersteller der LaTeX-Vorlage für deren Aktualisierung und Bereitstellung.

Bibliografische Angaben:

Ullrich, Henning

Entwicklung einer Control-Policy für eine KI-basierte Datenflusssteuerung

Development of a control-policy for a AI-based data-flow control

50 Seiten, Hochschule Mittweida - University of Applied Sciences, Fakultät Angewandte Computer- und Biowissenschaften, Bachelorarbeit, August 2022

Inhaltsverzeichnis

Abbildungsverzeichnis iii

1 Einleitung 1

1.1 Raumkonzept . . . 1
1.2 Motivation . . . 2
1.3 Zielstellung . . . 2

2 Problembeschreibung 3

3 Anforderungen 4

3.1 Allgemeine Anforderungen an die Netzwerkpolicy . . . 4
3.2 Anforderungen an das Policy-Management . . . 4
3.3 Use-Cases . . . 5
3.3.1 Use-Case Diagramm . . . 5
3.3.2 Use-Case Beschreibungen . . . 6

4 Konzept 10

4.1 Ergebnisse des Praktikums . . . 10
4.2 Akteure des Raumkonzepts . . . 11
4.3 Aufbau und Inhalt der Policy . . . 13
4.3.1 Format des Regelsatzes . . . 15
4.3.2 JSON-Schema . . . 17
4.3.3 Erweiterungen des Regelsatzes (Ausblick) . . . 17
4.4 Aufbau des Policy-Enforcers . . . 17
4.4.1 Interaktionen des Policy-Enforcers im Raum . . . 19
4.4.2 Innensicht des Policy-Enforcers . . . 21
4.4.3 Nutzung mehrerer aktiver Policies (Ausblick) . . . 25
4.5 Policy-Management . . . 25
(Iteration I) einfaches Überschreiben . . . 26
(Iteration II) activeRulesetQueue . . . 27
(Iteration III) priority 0 Variable . . . 28
4.5.1 Erweiterung des Policy-Managements (Ausblick) . . . 29

5 Prototypische Implementierung **30**

5.1 Finaler Implementationsstand 30
5.2 Nachweis der Wirksamkeit 31
5.3 Ergebnisse 35

6 Fazit **36**

6.1 Zusammenfassung 36
6.2 Ausblick für weitere Implementierungsschritte 37

Literaturverzeichnis **I**

A JSON-Schema **A1**

Abbildungsverzeichnis

3.1 Use-Case Diagramm 5

4.1 relevante Komponenten des Raumkonzepts 11
4.2 Policy Datenstruktur 13
4.3 Interaktionen des Policy-Enforcers 18
4.4 Sequenzdiagramm Interaktion im Raum 19
4.5 Sequenzdiagramm Policy umsetzen 21
4.6 Komponentendiagramm 23
4.7 Iteration I 26
4.8 Iteration II 27
4.9 Iteration III 28

5.1 Screenshot-Policy-Enforcer 32
5.2 Screenshot-Rules-File 33
5.3 Screenshot Suricata Log 34
5.4 Screenshot Suricata-Rule-Matches 34

1. Einleitung

Die Bachelorarbeit beschäftigt sich damit, ein System zu entwickeln das Verhalten einer Netzwerk-Datenflusssteuerung über Policies zu steuern.
Dafür muss eine solche Policy definiert und der Inhalt sowie das Verhalten der Datenflusssteuerung beschrieben werden. Des Weiteren soll die beschriebene Datenflusssteuerung prototypisch implementiert und die Wirksamkeit der umgesetzten Policy nachgewiesen werden. Die Inhalte der Policy erstellt dabei eine semantischen KI.
Das System aus Policy und Datenflusssteuerung ist dabei nicht alleinstehend, sondern ein Teil eines Raumkonzeptes, welches parallel weiterentwickelt wurde.

1.1. Raumkonzept

Grundlage für die Betrachtung einer Policy bildet ein digitales Raumkonzept. Für diesen digitalen Raum soll an definierten Grenzpunkten sämtlicher Netzwerkverkehr überwacht und kontrolliert werden. Dieser Raum wird als digitaler Souveränitäts-Bereich (Digital Area of Sovereignty, DAoS) bezeichnet. Eine digitale Autorität hat die Hoheit über den ihr zugewiesenen Raum und weiß über alle Netzwerkteilnehmer, Server und andere Ressourcen und Strukturen innerhalb des Raumes Bescheid. Da die DAoS georeferenziert ist, kann eindeutig bestimmt werden, wer für den Raum verantwortlich ist.
Die Grenzpunkte der DAoS bilden sogenannte Digital Area Points (DAP). Diese Komponenten kennen ihren eigenen geografischen Standpunkt und machen die DAoS lokalisierbar. Jedem DAP ist mindestens ein DAoS-Controller zugeordnet. Diese Controller sind in der Lage, den durch sie fließenden Netzwerkverkehr zu steuern. Durch das Steuern dieser DAoS-Controller kontrolliert die digitale Autorität den Netzwerkverkehr der DAoS. Die Technologie könnte somit in Rechenzentren, Firmen-Netzen oder kritischen Infrastrukturen zum Einsatz kommen. Eine staatliche Kontrolle mit Hilfe der Technologie sollte kritisch betrachtet werden. [1]
Die Komponenten des Raumkonzepts werden unter 4.2 genauer erklärt.

1.2. Motivation

Die Sicherheitsstrategie des Raumkonzepts lässt sich nur sicher umsetzen, wenn die DAoS-Controller den Netzwerkverkehr in und aus der DAoS kontrollieren. Die digitale Autorität hat die Kontrolle über die DAoS und steht damit in der Verantwortung, den Netzwerkverkehr an den DAoS-Controllern zu steuern. Um diese Kontrolle des Netzwerkverkehrs ausüben zu können, formuliert die Autorität Regeln. Diese Regeln sind von den DAoS-Controllern umzusetzen. Um nun die DAoS-Controller, und damit die Außengrenzen des Raums kontrollieren zu können, müssen diese Regeln in Form von Policies an diese DAoS-Controller übertragen und umgesetzt werden.

1.3. Zielstellung

Ziel dieser Arbeit ist es, eine Policy zu entwickeln, welche den Netzwerkverkehr aus und in einen digitalen Raum steuert. Die Policy soll dabei einen Regelsatz enthalten, anhand dessen der Netzwerkverkehr wirksam gesteuert werden kann. Als Nachweis der Wirksamkeit des Konzepts soll eine Beispiel-Policy prototypisch umgesetzt werden.

Die digitale Autorität erstellt für jeden DAoS-Controller einen eigenen Regelsatz. Dieser Regelsatz enthält alle Regeln, die an dem entsprechenden DAoS-Controller umgesetzt werden sollen. Die Übertragung und Umsetzung des Regelsatzes sollen in dieser Arbeit anhand einer Policy konzeptioniert und implementiert werden.
Um den DAoS-Controllern ihre umzusetzenden Regeln mitzuteilen, wird der Regelsatz eines DAoS-Controllers in einer Policy beschrieben.

2. Problembeschreibung

Ein Raum lässt sich aus Sicherheitsgründen grundsätzlich nicht betreiben, ohne den Netzwerkverkehr an seinen Außengrenzen zu regeln und gleichzeitig zu überwachen. Ohne eine Überwachung des Netzwerkverkehrs an den Außengrenzen könnte Jeder ohne Einschränkungen versuchen, auf Ressourcen im Raum zuzugreifen, diese zu verändern oder den Raum zu passieren. Im abgesteckten Raum könnte der Netzwerkverkehr nicht souverän festgelegt werden und er hätte keinen Effekt.
Aus Sicherheitsgründen ist auch eine Regelung des Netzwerkverkehrs innerhalb des Raums notwendig. Ohne diese innere Steuerung und Überwachung könnten Teilnehmer, die nur innerhalb des Raumes operieren, ohne Einschränkungen die Ressourcen des Raums nutzen.
Um diese Überwachung des Netzwerkverkehrs zu steuern benötigt man Regeln, die die gewünschten Kommunikationsziele auf Netzwerkebene beschreiben und durchsetzen. Dabei müssen die Regeln viele komplexe Aspekte des Netzwerkverkehrs berücksichtigen. Um sicher zu stellen, dass auch zukünftige Überwachungsziele dargestellt werden können, müssen die Regeln flexibel an diese neuen Ziele angepasst werden können.
Da sich Regeln im Rahmen der Raumverwaltung ändern, wird ein zugeschnittenes Policy-Konzept benötigt, um sie an neue Anforderungen wirksam anzupassen.

3. Anforderungen

Die hier vorgestellten Anforderungen leiten sich aus Sicherheitszielen der Raumverwaltung ab. Sowohl an den Außengrenzen als auch im Inneren des Raums müssen sich Kommunikationsbeziehungen zwischen den IT-Ressourcen ausrichten, steuern und gleichzeitig überwachen und nachweisen lassen.

3.1. Allgemeine Anforderungen an die Netzwerkpolicy

Aus der Problemstellung ergeben sich folgende Aufgaben:

1. Eine Policy und ihre enthaltenen Regeln für die Ausrichtung und Steuerung des Netzwehrverkehrs müssen spezifiziert werden.
2. Für den Austausch von Policies zwischen Autorität (Policy-Management) und den DAoS-Controllern (Policy-Umsetzung) muss ein Protokoll entwickelt werden.
3. Der Kommunikationsendpunkt eines DAoS-Controllers muss in der Lage sein, eine spezifizierte Policy zu interpretieren und technisch wirksam anzuwenden.
4. Die wirksame Datenflusssteuerung des DAoS-Controllers muss prototypisch implementiert und demonstriert werden.
5. Die Aktivierung und das Inkrafttreten einer Policy muss von dem DAoS-Controller an die digitale Autorität attestiert werden.

3.2. Anforderungen an das Policy-Management

Für die Übertragung einer Policy sind folgende Akteure erforderlich:

- Der **Policy-Enforcer** eines DAoS-Controllers empfängt die neue Policy und setzt die Regeln der Policy mit Hilfe einer Datenflusssteuerung um.
- Das **Policy-Management** ist dafür verantwortlich, die entsprechenden Policies an die DAoS-Controller des Raums zu verteilen.

Für das Policy-Management wurden folgende Rahmenbedingungen gestellt:

- Um sicher zu stellen, dass es keine Konflikte oder Überschneidungen zwischen den Regeln gibt, kann immer nur eine Policy auf einmal aktiv sein. Diese Policy enthält alle umzusetzenden Regeln.
- Wenn es Änderungen an der Policy gibt oder einzelne Regeln aktiviert oder deaktiviert werden sollen, wird eine neue Policy erstellt. Diese enthält den gesamten aktualisierten Regelsatz.
- Bei der ersten Inbetriebnahme des Policy-Enforcers wird eine Default-Policy verwendet, die zunächst alle Verbindungen zulässt. Um Verbindungen zu blockieren oder eine Benachrichtigung auszulösen, muss das Policy-Management eine neue Policy erstellen und an alle betreffenden Policy-Enforcer senden.

3.3. Use-Cases

Die Anforderungen an das Policy-System werden mit den folgenden Use-Cases weiter konkretisiert.

3.3.1. Use-Case Diagramm

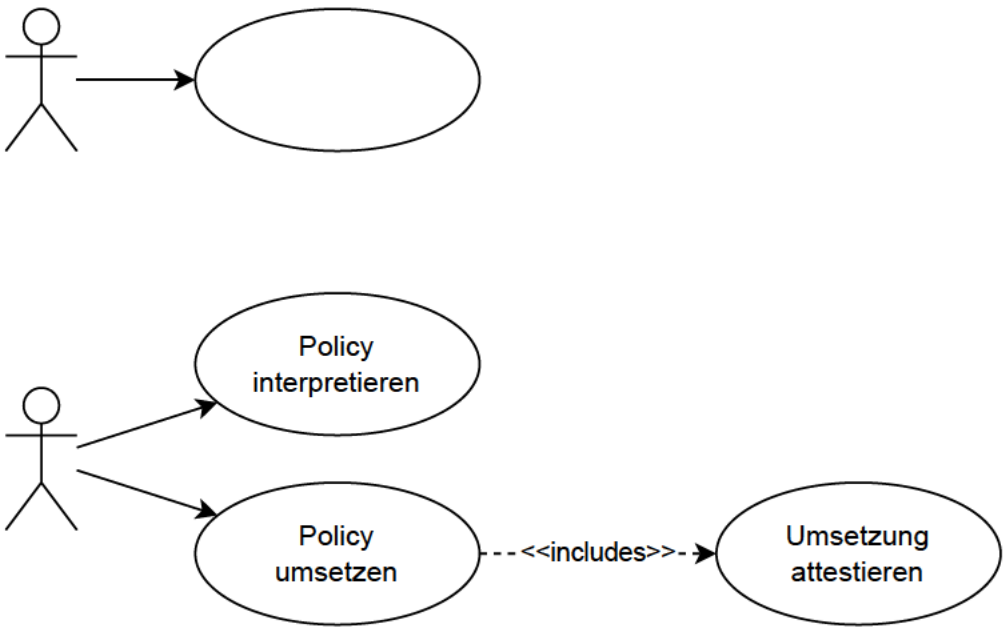

Abbildung 3.1.: Use-Case Diagramm

3.3.2. Use-Case Beschreibungen

Tabelle 3.1.: Use-Case Policy definieren

Titel	**Policy definieren**
Ziel	Eine neue Policy wird definiert und allen Policy-Enforcern bereitgestellt.
Vorbedingung	Das Policy-Management hat den Regelsatz neu definiert. Eine Verbindung zu allen Control-Points über ein *Management-Netz* ist möglich.
Akteure	Policy-Management
Auslöser	Das Policy-Management muss neue Regeln definieren.
Beschreibung (Standartablauf)	1. Der neue Regelsatz wird in eine Datenstruktur eintragen. Die ausgefüllte Datenstruktur beschreibt eine neue Policy. 2. Die Policy wird allen Policy-Enforcern bereitgestellt.
Nachbedingung Erfolg Fehlschlag	 Allen Policy-Enforcern steht eine neue Policy zu Verfügung und eine Attestierung der Bereitstellung der Policy liegt vor. Nicht allen Policy-Enforcern liegt die Policy korrekt vor.
Erweiterungen	-
Alternativen	-

Tabelle 3.2.: Use-Case Policy interpretieren

Titel	**Policy interpretieren**
Ziel	Eine neue Policy soll interpretiert und attestiert werden.
Vorbedingung	Dem Policy-Enforcer liegt eine neue Policy vor.
Akteure	Policy-Enforcer
Auslöser	Der Use-Case wird ausgelöst, wenn dem Policy-Enforcer eine neue Policy bereitgestellt wird.
Beschreibung (Standartablauf)	1. Eine Policy wird vom Policy-Management bereitgestellt. 2. Die Policy wird interpretiert und deren Inhalt geprüft. 3. Die Umsetzung der Policy zum darin spezifizierten Zeitpunkt wird initiiert. 4. Ein Attest über die korrekte Interpretation der Policy wird dem Policy-Manager bereit gestellt.
Nachbedingung Erfolg Fehlschlag	 Die Umsetzung der Policy wurde korrekt interpretiert und dem Policy-Manager liegt ein Attest darüber vor. Die Policy wurde nicht korrekt interpretiert oder es wurde kein Attest ausgestellt.
Erweiterungen	-
Alternativen	-

Tabelle 3.3.: Use-Case Policy umsetzen

Titel	**Policy umsetzen**
Ziel	Der Netzwerkverkehr wird nach den Regeln der neuen Policy kontrolliert und attestiert.
Vorbedingung	Die Umsetzung einer Policy wurde durch den Use-Case *Policy interpretieren* (Tab. 3.2) initiiert und der Zeitpunkt der Umsetzung ist erreicht.
Akteure	Policy-Enforcer
Auslöser	Die Umsetzung einer Policy wurde initiiert und der Zeitpunkt der Umsetzung ist erreicht.
Beschreibung (Standardablauf)	1. Die Regeln der umzusetzenden Policy werden extrahiert. 2. Die extrahierten Regeln werden in ein Format übersetzt, welches von der Datenflusssteuerung genutzt werden kann. (abhängig von der verwendeten Anwendung) 3. Die übersetzten Regeln werden an die Datenflusssteuerung übergeben und von dieser umgesetzt. Dabei wird der Regelsatz der alten Policy durch den der umzusetzenden Policy ersetzt. Die Datenflusssteuerung ermöglicht dabei einen unterbrechungsfreien Wechsel zwischen den Regelsätzen. 4. Die Datenflusssteuerung attestiert den Empfang von Paketen, die den Regeln entsprechen an die Wissens-Datenbank der digitalen Autorität.
Nachbedingung Erfolg Fehlschlag	 Die Datenflusssteuerung steuert den Netzwerkverkehr nach den Regeln der neuen Policy. Bei einem Paket, welches den Regeln der Policy entspricht, wird der Empfang attestiert. Die Regeln der Policy werden nicht korrekt umgesetzt, oder es wird noch der Regelsatz der vorherigen Policy verwendet.
Erweiterungen	-
Alternativen	-

Tabelle 3.4.: Use-Case Umsetzung attestieren

Titel	**Umsetzung attestieren**
Ziel	Die korrekte Umsetzung des Regelsatzes der Policy wird sichergestellt und dem Policy-Management attestiert.
Vorbedingung	Der Policy-Enforcer hat eine Verbindung zu der Wissens-Datenbank der digitalen Autorität. Die Datenflusssteuerung kontrolliert den Netzwerkverkehr nach den Regeln der aktuellen Policy.
Akteure	Policy-Enforcer
Auslöser	Die Datenflusssteuerung empfängt ein Paket, auf welches mindestens eine Regel der Policy zutrifft.
Beschreibung (Standartablauf)	1. Die Ausgabe der Datenflusssteuerung wird ausgelesen 2. Aus der Ausgabe werden alle Relevanten Informationen extrahiert. 3. Aus den extrahierten Informationen wird ein neuer Eintrag erstellt. 4. Der Eintrag wird in der Wissens-Datenbank gesichert.
Nachbedingung Erfolg Fehlschlag	 Bei Eintreffen eines Paketes, welches einer Regel der Policy entspricht, wird ein Eintrag in der externen Datenbank erstellt. Es werden keine oder falsche Einträge in der Wissens-Datenbank erstellt.
Erweiterungen	-
Alternativen	-

4. Konzept

4.1. Ergebnisse des Praktikums

In einem vorangegangenen Praktikum wurde bereits ein Test-Netzwerk aufgebaut und erste Voruntersuchungen zur Kontrolle und Steuerung des Netzwerkverkehrs zwischen den Inside- und Outside-Netzen durchgeführt.
Das aufgebaute Test-Netzwerk (TestLab) besteht aus einem Management-Netz, sowie einem Inside- und Outside-Netz. Die Netze werden an einem Switch durch untagged VLANs mit eigenen IP-Adressbereichen getrennt.
In den Inside- und Outside-Netzen befinden sich jeweils ein Netzwerkteilnehmer, um die Kommunikation zwischen den beiden Netzen zu testen.
Verbunden werden die drei Netze durch eine DPU (4.1), die als DAoS-Controller arbeiten soll.

> Smart-NIC / DPU:
> Eine Smart-NIC (Smart Network-Interface-Card) bezeichnet eine Netzwerkkarte, die zusätzlich zur normalen Netzwerkanbindung einen dedizierten Prozessor zur Verfügung stellt. Dieser Prozessor kann dann dafür verwendet werden, am Netzwerkverkehr eine Vorverarbeitung durchzuführen, ohne dabei die CPU des Host-Systems zu belasten.
> Die (im DAoS-Controller) verbaute Bluefield-2-Karte wird als DPU (Data Processing Unit) bezeichnet. Diese Bezeichnung beschreibt eine Form von Smart-NIC, welche ein komplettes System mit CPU, RAM, Speicher und Betriebssystem auf der Karte verbaut hat. So kann im Gegensatz zu Smart-NICs mit einfacheren Prozessoren eine höhere Flexibilität in der Anwendung erreicht werden. [2]
>
> Die (genutzte) Bluefield-2-DPU besitzt spezielle Hardware, um Datenströme nach angegebenen Regular-Expressions sehr schnell zu untersuchen. Die dafür nötigen Treiber und bereitgestellten Interfaces werden als RegEx-Engine bezeichnet. [3]

Über einen Port im Management-Netz sollen die Policies und zugehörige Benachrichtigungen empfangen werden. Über Ethernet-Ports in den Inside- und Outside-Netzen funktioniert die DPU als Gateway in das jeweils andere Netz. So kann auf der DPU sämtlicher Netzwerkverkehr zwischen diesen beiden Netzen kontrolliert werden.

Es wurde entschieden ein Intrusion-Prevention-System (IPS) zum Steuern des Netzwerkverkehrs zu nutzen. Wie von der Policy gefordert, analysiert und steuert ein solches System den Netzwerkverkehr mit Hilfe von selbst definierbaren Regeln. Für die im Praktikum verwendete DPU wurde vom Hersteller eine Beispielanwendung eines IPS-Systems zur Verfügung gestellt. Diese Beispielanwendung nutzte die RegEx-Engine der DPU um den Netzwerkverkehr zu kontrollieren und gegebenenfalls zu unterbrechen. Im Praktikum wurden die Möglichkeiten dieser Beispielanwendung untersucht. Wie sich herausstellte, unterstützte die Beispielanwendung einige wichtige Funktionen, wie das Aktualisieren der genutzten Regeln während der Laufzeit, noch nicht. Die Implementation der fehlenden Funktionen kam aufgrund der Komplexität der DPU nicht in Frage.
In der Bachelorarbeit wurde deshalb entschieden, ein normales CPU-basiertes IPS-System zu verwenden. Da die Anwendung der DPU bereits das Suricata-Format für ihre Regeln nutzte, wurde Suricata als IPS-System auf der DPU installiert.

4.2. Akteure des Raumkonzepts

Das beschriebene Raumkonzept umfasst folgende Komponenten:

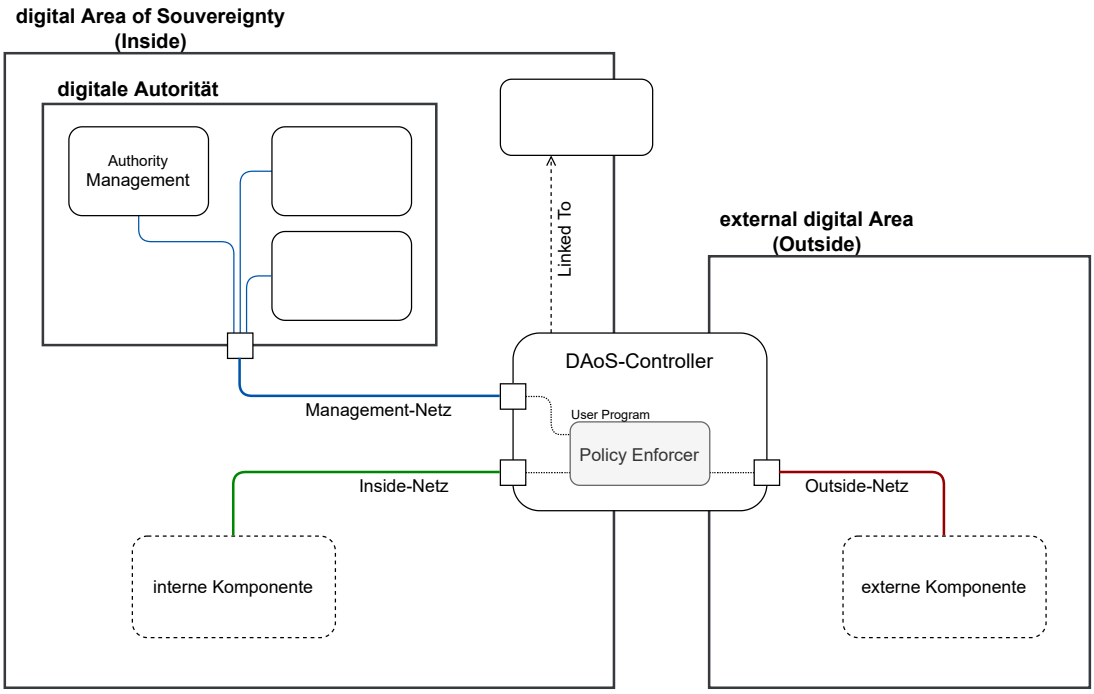

Abbildung 4.1.: relevante Komponenten des Raumkonzepts

semantische KI: Die digitale Autorität des Raums wird durch eine semantische KI abgebildet. Die semantische KI hat Zugriff auf eine Wissenssammlung mit allen Strukturen, Teilnehmern und Komponenten der DAoS. Aus diesem Wissen kann sie Regelsätze definieren, nach denen die DAoS-Controller den Netzwerkverkehr der DAoS steuern. Eine Besonderheit der semantischen KI ist es, dass es sich nicht um ein neuronales Netz handelt sondern die KI versucht, die reale Welt durch Objekte und Beziehungen abzubilden. Strukturen und Komponenten des Raums müssen dabei vorher durch solche Objekte und Beziehungen beschrieben werden. Die Aufgabe der KI ist es, die beschriebenen Strukturen in den realen Daten zu erkennen und aus den gewonnenen Erkenntnissen Regelsätze zu erstellen. In dem folgenden Dokument bezieht sich der Begriff KI auf die beschriebene semantische KI.

Wissens-Datenbank: Die Wissensdatenbank ist ein Datenbanksystem, welches alle gesammelten Daten der KI enthält. Interessant für die Bachelorarbeit sind hier die Attest-Datenbanken *Update-DB* und *Match-DB*. Darin sind sämtliche von den DAoS-Controllern gesammelten Daten enthalten. Die Daten umfassen Atteste über die (De-)Aktivierung von Policies sowie das Auslösen einzelner Regeln.

Policy-Service: Um die Regelsätze an die DAoS-Controller zu übertragen, wurde der Policy-Service definiert. Damit die DAoS-Controller nicht direkt auf die KI zugreifen, wird diese Aktion durch den Policy-Service abstrahiert. Für den Rückgabewert des Services wurde ein JSON-Schema definiert. (siehe 4.3.2)

Authority-Management: Das Authority-Management dient der Verwaltung der DAoS. Es ist für die Entwicklung der Struktur und die Einhaltung eines definierten Verhaltens in einem digitalen Raum zuständig. Um diese Zuständigkeiten wahrzunehmen, legt das Authority-Management Policies fest, die über DAoS-Controller umgesetzt werden. Unter die Zuständigkeit des Authority-Managements fällt auch die Benachrichtigung der DAoS-Controller, sobald eine neue Policy vorliegt. Das Authority-Management kann über eine grafische Oberfläche bedient werden.

Digital Area Point: Digital Area Points (DAP) dienen dazu, die DAoS in der realen Welt abzubilden. Jeder dieser Punkte hat einen Geo-Locator, welcher den geografischen Standpunkt des DAPs kennt. Durch die Anwendung von mindestens drei solcher DAPs kann so eine Fläche und damit eine DAoS aufgespannt werden. Jedem DAP ist mindestens ein DAoS-Controller zugeordnet.

DAoS-Controller: Wie oben beschrieben, kontrolliert ein Digital Area Point den Netzwerkverkehr des Raums und hilft dabei, den Raum geografisch lokalisierbar zu machen. Bei DAoS-Controllern wird zwischen internen und externen Controllern unterschieden.

Ein interner DAoS-Controller steuert den Netzwerkverkehr zwischen Teilnehmern des Raums. (in Abb. 4.1 nicht abgebildet)
Ein externer DAoS-Controller kontrolliert und steuert den Netzwerkverkehr zwischen dem Raum (Inside) und der Außenwelt (Outside). Die Kontrolle an den Außengrenzen stellt sicher, dass nur autorisierte externe Verbindungen auf definierte Ressourcen des digitalen Raumes zugreifen können.
Diese Arbeit befasst sich mit Policies, die in einem externen DAoS-Controller zur Anwendung kommen.

Policy-Enforcer: Der Policy-Enforcer kommuniziert mit der digitalen Autorität und steuert den Empfang und die Aktivierung von Policies für die Netzwerkkommunikation des DAoS-Controllers. Er empfängt neue Policies und extrahiert die darin enthaltenen Regeln. Diese Regeln übergibt er anschließend der Datenflusssteuerung des DAoS-Controllers. Eine prototypische Implementation des Policy Enforcers weist in der Arbeit dabei die Umsetzung des Policy-Konzeptes nach.

4.3. Aufbau und Inhalt der Policy

Eine Policy beschreibt eine Richtlinie, nach welcher ein Teilsystem verändert werden soll.
Diese Richtlinie könnte beispielsweise das Ändern der Regeln einer Datenflusssteuerung oder einen Prozess zum Starten eines Services bewirken.
Um das Policy-Konzept auch in anderen Teilsystemen einsetzen zu können, muss zunächst eine allgemeine Form einer Policy definiert werden. Diese sollte wesentliche Struktureigenschaften enthalten, so dass auch eine flexible Steuerung anderer Teilsysteme möglich ist.
Die Bachelorarbeit spezialisiert sich auf die Anwendung des Policy-Konzepts an den Außengrenzen der DAoS, den externen Netzwerkpolicies.
Die hierfür entwickelte Policy-Struktur umfasst folgende Datenfelder:

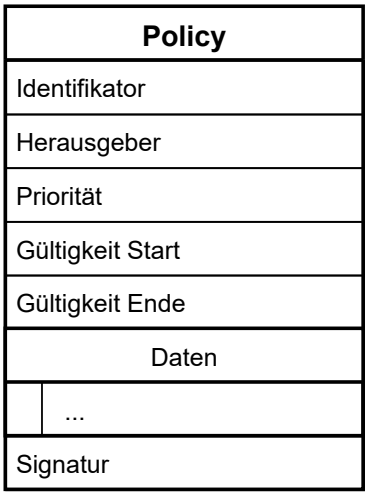

Abbildung 4.2.: Aufbau der Policy-Datenstruktur

Identifikator: Um Policies voneinander unterscheiden zu können, ist ein Identifikator nötig. Dieser wird im Feld *Identifikator* angegeben und ist vom Datentyp String (Zeichenkette). Der Identifikator kann neben der einfachen Unterscheidung der Policies auch der Angabe des Subjekts und der Versionierung dienen. Dafür müssen die entsprechenden Informationen darin codiert werden.
Ein solcher Identifikator könnte folgendermaßen Aussehen:
`ExtNetwork_22082022_042`
(externe Netzwerkpolicy für das Datum 22.08.2022 Nummer 42)

Herausgeber: Um den Herausgeber der Policy nachverfolgen zu können, wird dieser durch eine ID im Feld *Herausgeber* angegeben. Der Policy-Enforcer benötigt die ID des Herausgebers nur für die Verifikation der Signatur der Policy. Der Datentyp ist hier Zeichenkette (String).

Priorität: Da Policies auch in Reaktion auf ein Ereignis ausgestellt werden können und somit aktuell gültige Policies ersetzen können sollen, benötigt jede Policy eine Priorität. Das Feld *Priorität* gibt diese als Integer an. Als mögliche Werte für die Priorität wurde Folgendes definiert:

0 URGENT:
Die Policy wird sofort umgesetzt und ersetzt die aktuell gültige. Sie tritt außer Kraft sobald der Endzeitpunkt erreicht ist. Der Startzeitpunkt ist in diesem Fall unwichtig, da die Policy ohne Verzögerung umgesetzt wird.

1 NORMAL:
Diese Policy tritt in Kraft sobald der angegebene Startzeitpunkt erreicht ist, und endet wenn der Endzeitpunkt erreicht ist.

2 FALLBACK:
Diese Policy tritt in Kraft wenn keine Policy der Priorität NORMAL oder URGENT aktiv ist. Der Start- und End-Zeitpunkt ist hier unbedeutend, da die Aktivität der Policy durch das In- und Außerkrafttreten von anderen Policies bestimmt wird.

Gültigkeit Start, Gültigkeit Ende: Da eine Policy nur eine bestimmte Zeit lang aktiv ist, muss die Zeitspanne der Aktivität angegeben werden. Dafür wird ein Start- und ein End-Zeitpunkt benötigt. Diese Zeitpunkte werden in den Feldern *Gültigkeit Start* und *Gültigkeit Ende* angegeben.
Der Datentyp dieser Felder wurde auf Datetime festgelegt.

Daten: Die spezifischen Regeln einer Policy werden im Feld *Daten* angegeben. Das Feld *Daten* muss jedoch als Struktur mehrere Regeln aufnehmen können. Als Datentyp wurde daher ein String (Zeichenkette) genutzt. Durch codieren der Regeln in ein JSON-Format kann so ein kompletter Regelsatz in dem Feld *Daten* angegeben werden.
Je nach Einsatzgebiet der Policy kann das Datenfeld so auch andere Strukturen und Regeln enthalten. Dafür muss lediglich ein neues JSON-Format definiert und die neuen Daten dort eingetragen werden. Um dem Empfänger der Policy klar zu machen welches JSON-Format in dem Datenfeld genutzt wurde, muss die Art der Policy in dem Identifikator der Policy codiert werden.
Im Fall der Netzwerkpolicies enthält das Datenfeld eine Sammlung an Regeln (Regelsatz), nach welchen der Netzwerkverkehr kontrolliert werden soll.

Signatur: Eine Policy wird von ihrem Herausgeber im Feld *Signatur* signiert. Diese Signatur weist nach, dass die Policy tatsächlich von dem Herausgeber stammt und der Inhalt der Policy nicht nachträglich verändert wurde. Die Signatur ist vom Datentyp String (Zeichenkette).

4.3.1. Format des Regelsatzes

Um die Kommunikationsziele auf Netzwerkebene zu beschreiben, wird ein allgemeines Regel-Format benötigt. Damit das Format unabhängig von der genutzten Datenflusssteuerung wird, muss auch hier ein eigenes Format definiert werden.
Das entwickelte Format orientiert ich an den Filter-Regeln der Programme *tcpdump*[4] (*Berkerley Packet Filter*[5]), *Suricata*[6], sowie an den von Kubernetes genutzten Netzwerkpolicies [7].

Jede Regel umfasst die Felder:

action
protocol
source IP
source Port
destination IP
destination Port
direction
optional data

Die Bedeutung und mögliche Werte sind in der Tabelle 4.1 aufgeführt.

Tabelle 4.1.: Bedeutung und Inhalt der Datenfelder einer Regel

Datenfeld	**Inhalt und mögliche Werte**
action	Dieses Feld beschreibt die Aktion die ausgeführt werden soll, wenn ein Paket die beschriebenen Eigenschaften aufweist. mögliche Werte sind: **alert** - Das IPS-System gibt eine Nachricht, wenn ein Paket die beschriebenen Eigenschaften vorweist. **pass** - Das IPS-System lässt das Paket ohne weitere Aktionen passieren. **drop** - Das IPS-System leitet die beschriebenen Pakete nicht weiter. Zusätzlich wird eine Nachricht ausgegeben.
protocol	Dieses Feld gibt an, welches Protokoll die beschriebenen Pakete nutzen. Da zum Umsetzen der Regeln Suricata verwendet wird, können nur die davon unterstützten Protokolle genutzt werden. einige mögliche Werte sind: **ip**, **icmp**, **tcp** oder **http** Eine vollständige Übersicht ist hier zu finden: [8]
source IP	Dieses Feld beschreibt die IP-Adresse der Quelle des zu filternden Pakets. Es kann eine einzelne IP-Adresse oder ein Adressbereich angegeben werden. Es gilt die Suricata-Syntax [9]. Der Wert **any** deaktiviert das Filtern nach dieser Eigenschaft.
source Port	Dieser Wert gibt den Port der Quelle eines zu filternden Pakets an. Es wird sowohl ein einzelner Port als auch eine eine Sammlung von Ports akzeptiert. Es gilt die Suricata-Syntax [9]. Der Wert **any** deaktiviert das Filtern nach dieser Eigenschaft.
destination IP	Dieses Feld beschreibt die IP-Adresse des Ziels des zu filternden Pakets. Es kann eine einzelne IP-Adresse oder ein Adressbereich angegeben werden. Es gilt die Suricata-Syntax [9]. Der Wert **any** deaktiviert das Filtern nach dieser Eigenschaft.
destination Port	Dieser Wert gibt den Port des Ziels eines zu filternden Pakets an. Es wird sowohl ein einzelner Port als auch eine eine Sammlung von Ports akzeptiert. Es gilt die Suricata-Syntax [9]. Der Wert **any** deaktiviert das Filtern nach dieser Eigenschaft.
direction	Gibt an, ob nur Pakete von Quelle zu Ziel kontrolliert werden sollen, oder ob auch Pakete in der umgekehrten Richtung kontrolliert werden sollen. mögliche Werte sind: **unidirectional/uni** und **bidirectional/bi**
optional data	Dieses Feld kann für Erweiterungen genutzt werden.

4.3.2. JSON-Schema

Um die beschriebene Policy- und Regelstruktur von dem Policy-Service der KI bereit zu stellen, wurde ein JSON-Schema entwickelt. Darin sind die Datenfelder und deren Datentypen definiert. Das Schema wird nicht nur zum Übertragen der neuen Polices verwendet, sondern dient auch als eine erste Verifikation der empfangenen JSON-Struktur. (mehr dazu unter 4.4.2) Für die Übertragung mehrerer Policies, beinhaltet das Schema eine Liste aus Policy-Strukturen.
Wie in Abschnitt 3.2 beschrieben, wird im Rahmen der Bachelorarbeit angenommen, dass zu jedem Zeitpunkt nur eine der übertragenen Policies aktiv ist. Durch das Übertragen von mehreren Policies kann so eine Abfolge an Policies festgelegt werden, ohne jede einzeln übertragen zu müssen.
Das vollständige Schema ist im Anhang unter A.1 aufgeführt.

4.3.3. Erweiterungen des Regelsatzes (Ausblick)

Umleitungsregeln: Eine mögliche Erweiterung ist die Implementation einer Aktion, um Pakete umzuleiten. Das Ziel, an welches die Pakete umgeleitet werden sollen, könnte in dem Feld *optional data* hinterlegt werden. Da Suricata eine solche Aktion nicht unterstützt, muss für die Umsetzung dieser Aktion eine eigene Komponente entwickelt werden.

Einsatz von Variablen: Um die Möglichkeiten der Suricata-Regeln voll auszunutzen, ist der Einsatz von Variablen bei der Angabe von IP-Adressen vorgesehen. Die Felder *source IP* und *destination IP* sollen anstelle von IP-Adressen eine Variable in der Form `$variable` beinhalten können. Die Angabe von vielen IP-Adressen oder (Teil-)Netzen gestaltet sich so einfacher und die Regeln sind besser lesbar. Der Inhalt der Variablen kann in dem Feld *optional Data* definiert werden. Eine Regel mit solchen Variablen könnte so aussehen:

```
drop ip $INSIDE_HOSTS any -> any any (msg:"Rule A"; sid:1000;)
```

während in dem Feld *optional Data* der Wert der Variable definiert wird:

```
$INSIDE_HOSTS: [ 192.168.2.7, 192.168.2.42, 192.168.2.99 ]
```

4.4. Aufbau des Policy-Enforcers

Der Policy-Enforcer setzt eine Policy um.
Er regelt den Empfang der Policies und setzt die darin angegebenen Regeln mit Hilfe eines IPS-Systems um. Nach dem Empfang und der anschließenden Signatur-Kontrolle wird die Policy zwischengespeichert. Je nach Priorität wird die Policy sofort, oder erst zum angegebenen Startzeitpunkt umgesetzt.

Zu Beginn des Projekts war geplant ein IPS-System zu verwenden, welches die Hardwarebeschleunigung der Bluefield-2 DPU nutzt. Dazu wurde eine vorhandene Beispielanwendung angepasst und genutzt. [10]
Die beschriebene Beispielanwendung nutzt Regeln im Format der IPS-Software Suricata. Ein Test der Beispielanwendung ergab, dass es unter Verwendung der Hardwarebeschleunigung nur möglich war, TCP und UDP Verbindungen zu kontrollieren. [10] [11] Da die Policy nicht nur TCP/UDP Verbindungen beschreibt, wird im Rahmen der Bachelorarbeit eine klassische Suricata-Installation als IPS-System verwendet. Sobald die Beispielanwendung der DPU den vollen Umfang des Suricata-Regelsatzes unterstützt, kann das IPS-System relativ einfach getauscht werden. Neben dem vollen Umfang der Suricata-Regeln, bietet eine klassische Suricata-Installation eine Funktion, die das aktuell genutzte Regelwerk im laufenden Betrieb tauscht. [12]

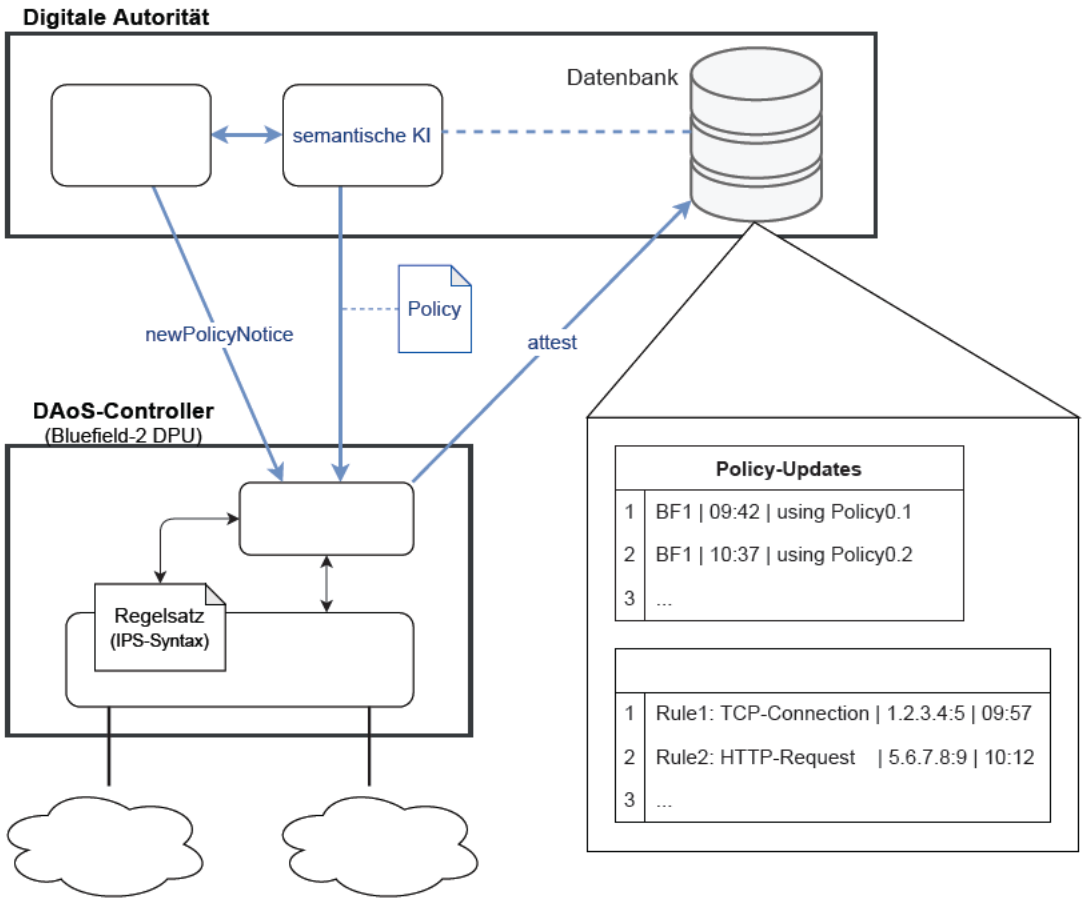

Abbildung 4.3.: Interaktionen des Policy-Enforcers

In Abbildung 4.3 wird die Kommunikation des Policy-Enforcers mit der digitalen Autorität, sowie dem IPS-System veranschaulicht. Der Ablauf der Kommunikation wird in den folgenden Kapiteln erläutert.

4.4.1. Interaktionen des Policy-Enforcers im Raum

Das Sequenzdiagramm in Abbildung 4.4 zeigt, wie eine Policy an den Policy-Enforcer übertragen und attestiert wird. Dabei wird deutlich, wann der Policy-Enforcer mit welchem Akteur im Raum kommuniziert. Die dargestellte Kommunikation findet ausschließlich im Management-Netz statt.

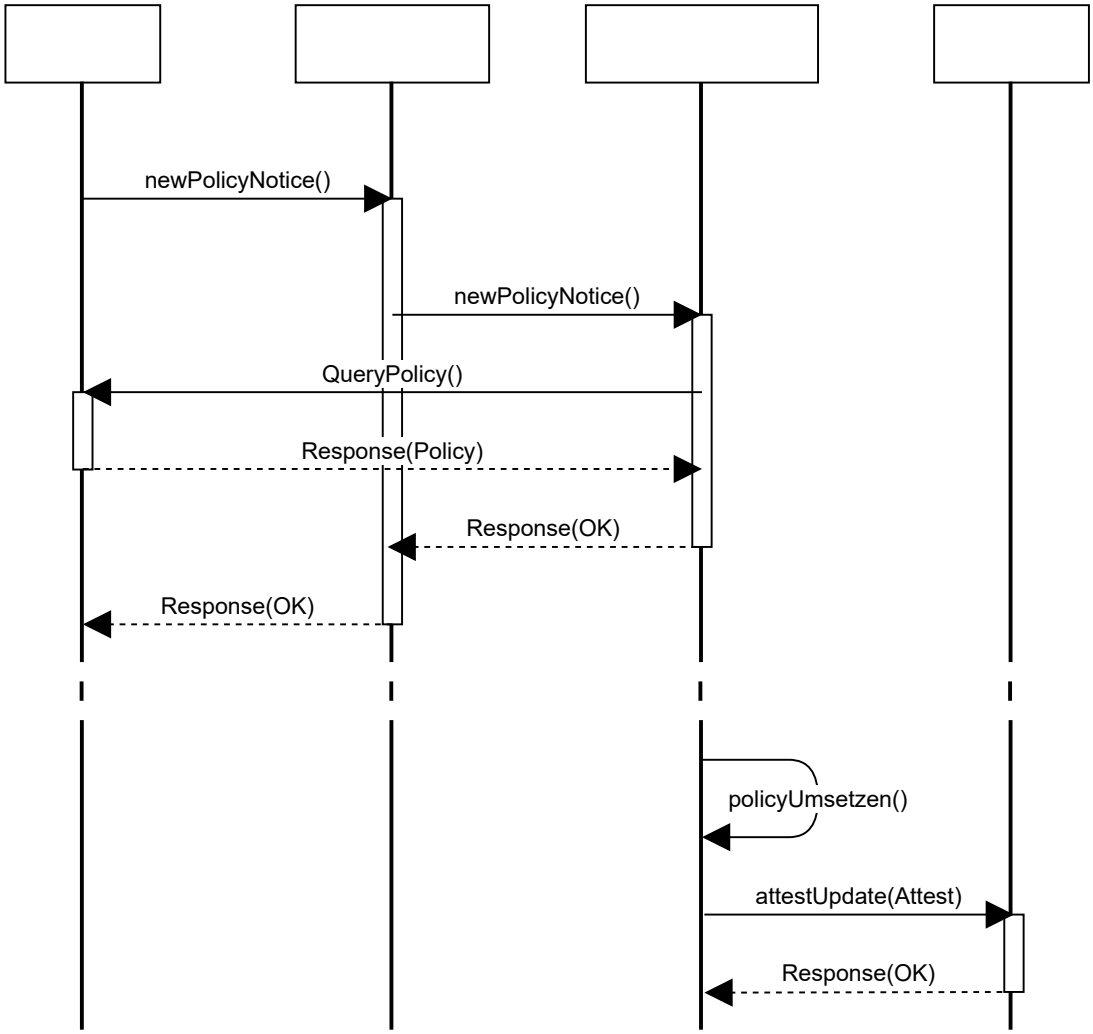

Abbildung 4.4.: Sequenzdiagramm Interaktion im Raum

Nachdem eine neue Policy von der KI erstellt wurde, muss sie an alle Enforcement-Points des Raums verteilt werden. Die Benachrichtigung der Enforcement-Points wird durch das Authority-Management vorgenommen. Diese Benachrichtigungen sind in Abbildung 4.4 als *newPolicyNotice()* dargestellt.

Nachdem der Enforcement-Point die Nachricht über eine neue Policy erhalten hat, muss er diese von der KI anfordern. Dafür stellt die KI den Policy-Service bereit. Dieser trägt alle von der KI generierten Daten in ein JSON-Schema ein. (siehe 4.3.2)

Mit *QueryPolicy()* wird die neue Policy angefragt. Die Anfrage an den Policy-Service beinhaltet einen Nachweis über die Identität des Anfragenden. So wird sicher gestellt, dass jedem Enforcement-Point die korrekte Policy übertragen wird. Außerdem wird so die Identität der Enforcement-Points kontrolliert. Nicht jeder Teilnehmer des Management-Netzes kann eine Policy empfangen.
Die Antwort der KI *Response(Policy)* enthält alle neuen Policies für den anfragenden Policy-Enforcer.

Nachdem der Policy-Enforcement-Point die neue Policy erfolgreich empfangen hat, sendet er eine Antwort an das Authority-Management und bestätigt somit den Empfang der Policy. In der Abbildung 4.4 ist diese Empfangsbestätigung als *Response(OK)* dargestellt.

Die neue Policy ist dem Enforcement-Point nun bekannt. Da sie meist nicht sofort in Kraft tritt, wartet der Enforcement-Point zunächst bis das Startdatum der Policy erreicht ist. Erst dann setzt er die Policy um. Im Sequenzdiagramm wird diese Aktion mit *policyUmsetzen()* abgebildet.

Nach dem erfolgreichen Umsetzen der Policy wird ein Attest über die Umsetzung der Policy an das Datenbanksystem der KI übertragen. Die Attestierung wird in Abbildung 4.4 durch *attestUpdate()* und *Response()* dargestellt.

4.4.2. Innensicht des Policy-Enforcers

Da eine interpretierte Policy in der Regel nicht sofort umgesetzt werden soll, muss der Enforcer bis zum Start der Policy-Gültigkeit warten. Der Empfang der Policy erfordert jedoch eine Bestätigung. Das Programm darf dabei nicht blockieren oder in einer Warteschleife die Rechenzeit des Prozessors unnötig belasten.
Um diese Anforderungen umzusetzen, wurde der Policy-Enforcer in die Komponenten *Policy Receiver*, *Timer* und *IPS Controller* unterteilt.

Die Abbildung 4.5 zeigt die Interaktion dieser Komponenten miteinander.

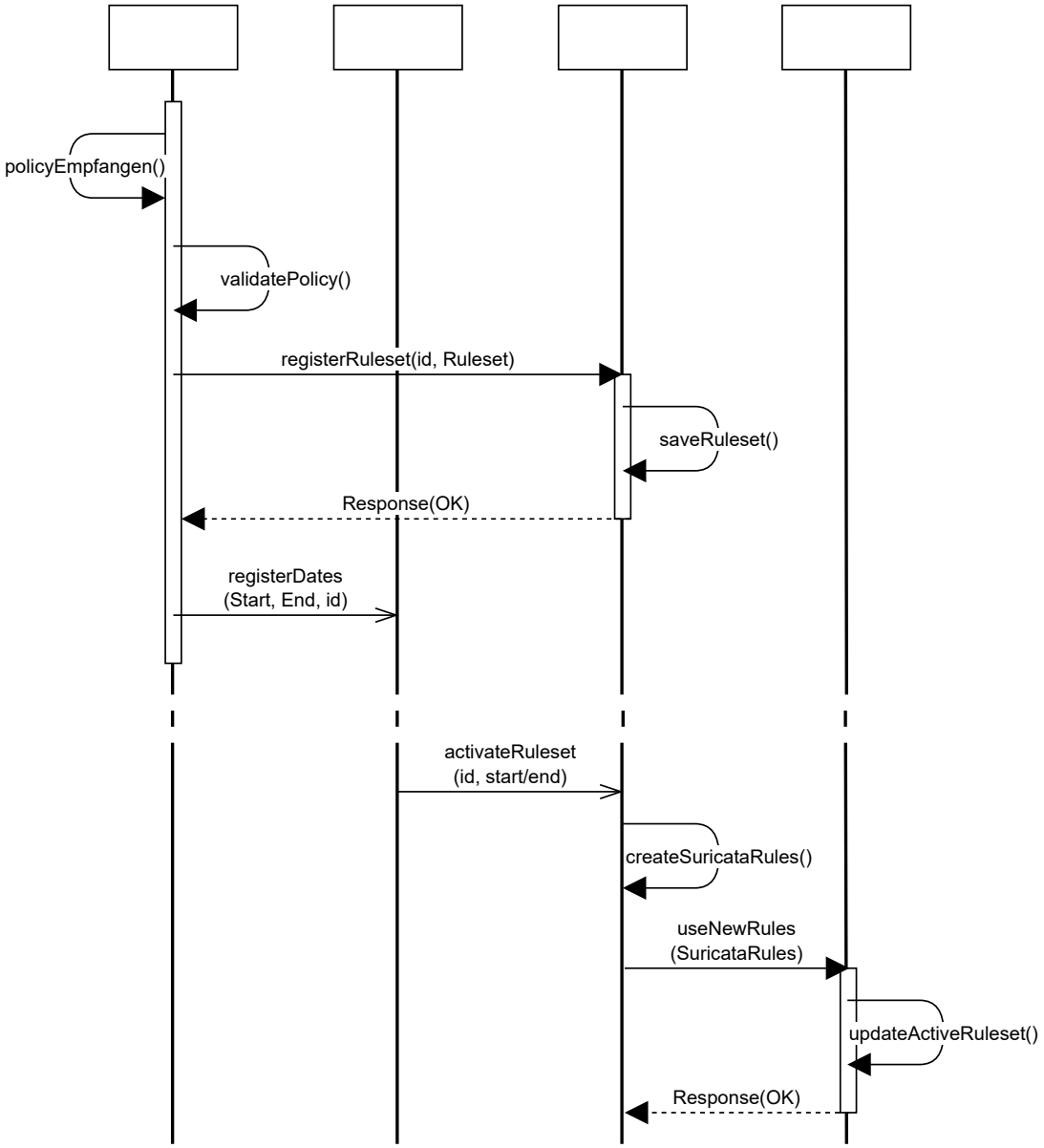

Abbildung 4.5.: Sequenzdiagramm Policy umsetzen

Der Empfang und die Interpretation der Policy wird von der Komponente *Policy-Receiver* übernommen.
Bei Empfang einer Policy, prüft der Policy-Receiver zunächst, ob die Policy valide ist. Wenn die Policy valide ist, übergibt er den Regelsatz und die ID der Policy an den IPS Controller, der diese Daten zunächst zwischenspeichert. Im Anschluss registriert der Policy Receiver den Start- und End-Zeitpunkt der Policy indem er diese an den Timer übergibt.

Der Timer wartet nun, bis ein registrierter Zeitpunkt erreicht ist. Anstatt in einer Schleife auf den Zeitpunkt zu warten, kann der Timer interne Funktionen des Betriebssystems nutzen, wodurch die Prozessorlast reduziert wird.

Sobald der Start- oder Endzeitpunkt der Policy erreicht ist, sendet der *Timer* ein Signal mit Policy-ID und gewünschtem Zustand (aktiv oder inaktiv) an den *IPS Controller*. Dieser setzt die Regeln der Policy daraufhin unter Beachtung der Priorität um. Dazu übersetzt er die Regeln der Policy in ein für das IPS lesbares Format. Diese übersetzten Regeln übergibt er dann an das IPS-System, und informiert dieses, dass es ab sofort die neuen Regeln umsetzen soll.

Das Komponentendiagramm Abbildung 4.6 zeigt den Aufbau des Policy-Enfrocers und dessen Subkomponenten.

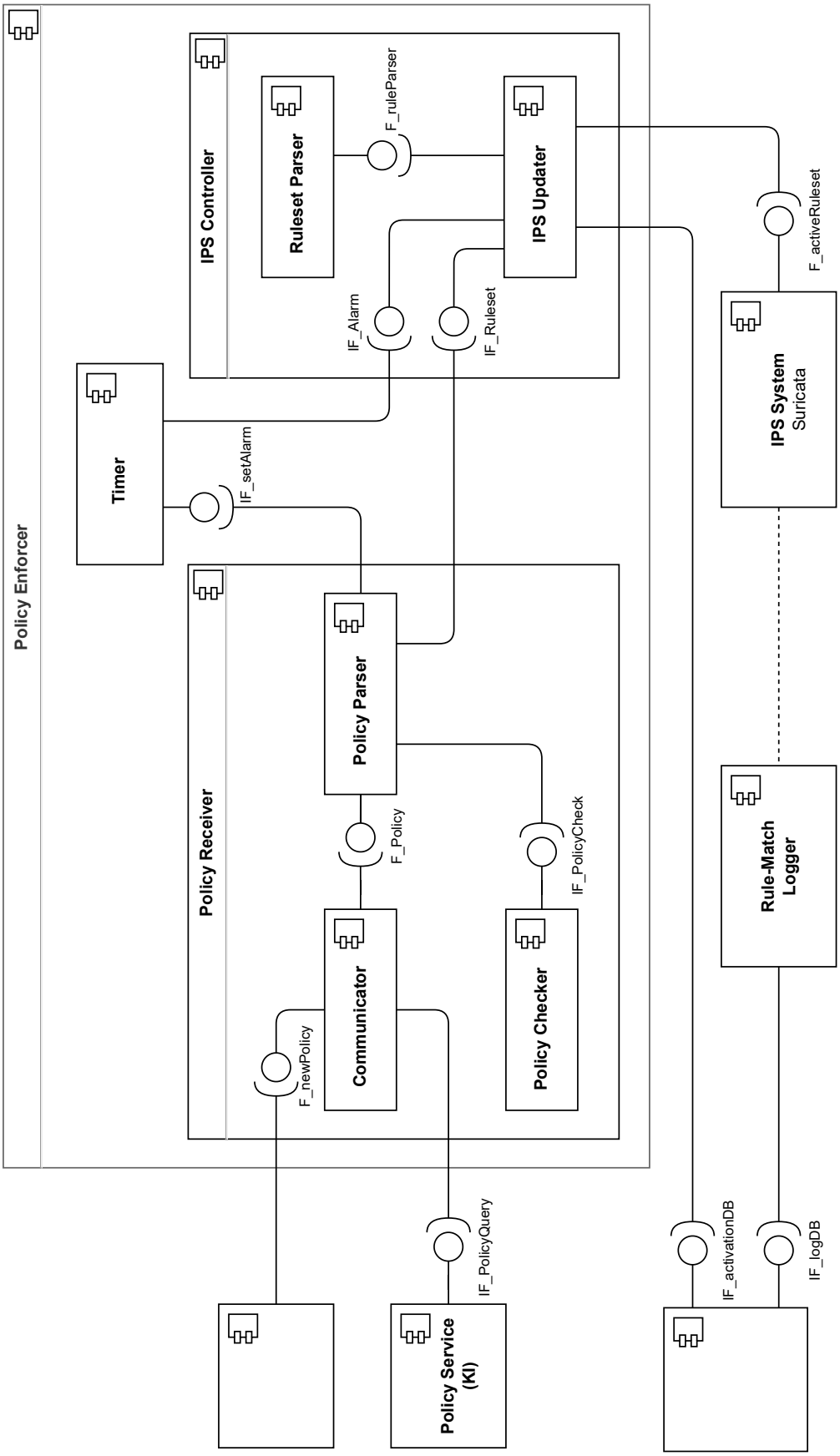

Abbildung 4.6.: Komponentendiagramm des Policy Enforcement Points

Policy-Receiver: Wie in Abbildung 4.6 dargestellt, besteht der Policy-Receiver aus den Subkomponenten *Communicator*, *Policy Parser* und *Policy Checker*.
Die Subkomponente *Communicator* übernimmt den Empfang der Policy und die Übergabe der empfangenen Policy an den *Policy Parser*. Über das Interface *IF_newPolicy* empfängt der Communicator eine Benachrichtigung des Authority-Managements, dass eine neue Policy vorliegt. (siehe Abb. 4.5) Um die neue Policy zu empfangen, nutzt der Communicator das Interface *IF_PolicyQuery* des Policy-Services. Dieser liefert die neue Policy in Form einer JSON-Struktur. (siehe 4.3.2)
Sobald eine vollständige JSON-Struktur vorliegt, wertet der Communicator diese mit Hilfe des *Policy Checkers* und des *Policy Parsers* aus.
Der Policy Checker prüft die empfangenen Daten zunächst gegen das in 4.3.2 definierte Schema. So kann festgestellt werden, ob das empfangene JSON die korrekten Datenfelder und Datentypen enthält.
Im Anschluss wertet der *Policy Parser* das empfangene JSON-Objekt aus und trägt die darin enthaltenen Daten in ein Policy-Objekt (Class Policy) ein.
Um auch den Inhalt der Policy zu prüfen, stellt der Policy Checker eine weitere Funktion *CheckPolicy()* bereit. Die Funktion prüft die Signatur der Policy, die angegebenen Start- und End-Zeitpunkte, sowie den Regelsatz der Policy auf ihren Inhalt.
Nachdem die Daten der Policy geprüft wurden, wird die Policy umgesetzt. Dazu wird zunächst der Regelsatz der Policy an den IPS Controller übergeben. Um den Regelsatz später einer Policy zuordnen zu können, wird auch die ID der Policy übergeben. Anschließend werden mit Hilfe des *Timers* der Start- und End-Zeitpunkt der Policy registriert. Auch hier wird die ID der Policy an den Timer übergeben, damit festgestellt werden kann, zu welcher Policy der Alarm gehört.
Der obere Teil des Sequenzdiagramms Abb. 4.5 stellt den beschriebenen Ablauf des Policy-Receivers dar.

IPS Controller: Die Komponente *IPS Controller* steuert das IPS-System (Suricata), und aktualisiert die umgesetzten Regeln. Die Komponente verhält sich dabei passiv und arbeitet nur, wenn ihr Daten übergeben oder Signale gesendet werden. Sobald das Programm gestartet oder eine neue Policy von der Komponente *Policy-Receiver* empfangen wird, sendet diese einen Regelsatz an das Interface *IF_Ruleset* der Komponente *IPS Updater*. Der empfangene Regelsatz wird dort zunächst zwischengespeichert.
Wenn der Startzeitpunkt der Policy erreicht ist und der Regelsatz damit aktiv werden soll, sendet der Timer ein Signal an das Interface *IF_Alarm*. Daraufhin wird der IPS-Updater aktiv und lässt den zwischengespeicherten Regelsatz von der Subkomponente *Ruleset Parser* in ein Format parsen, welches von dem IPS-System gelesen werden kann. (in diesem Fall Suricata)
Nachdem die Regeln nun für das IPS-System lesbar sind, nutzt der IPS Updater eine Update-Funktion des IPS-Systems um die neuen Regeln in Betrieb zu nehmen. Diese Funktion ist in Abb. 4.6 als *IF_activeRuleset* dargestellt.

Nachdem der neue Regelsatz durch das IPS-System in Betrieb genommen wurde, sendet der IPS-Updater ein Attest über die Inbetriebnahme des Regelsatzes an eine Datenbank der KI. Hier wird das Interface *IF_activation* DB genutzt.
Der untere Teil des Sequenzdiagramms Abb. 4.5 zeichnet den beschriebenen Ablauf noch einmal auf.

4.4.3. Nutzung mehrerer aktiver Policies (Ausblick)

Das System wurde so konzipiert, dass es in Zukunft mit relativ wenig Implementationsaufwand möglich ist, mehrere Policies parallel umzusetzen.
Da die Komponente *Policy-Receiver* unabhängig von der Komponente *IPS Controller* arbeitet, ist es dem Policy-Enforcer möglich, mehrere Policies zu empfangen. Der IPS Controller kann nach dem Empfang auch mehrere Regelsätze zwischenspeichern.
Der IPS Controller verfügt nun über eine Sammlung an Regelsätzen. Sobald einer der Regelsätze aktiv oder inaktiv geschaltet werden soll, erhält er eine Nachricht vom Timer mit der ID und dem Status des betreffenden Regelsatzes. Durch das Führen einer internen Liste kann der IPS Updater nun alle aktiven Regelsätze zu einem großen Regelsatz zusammenführen. Dabei sollte darauf geachtet werden, dass keine Überschneidungen oder Konflikte zwischen den Regeln entstehen.
Der zusammengesetzte Regelsatz kann nun genau wie im bestehenden System geparst und an das IPS-System übergeben werden.

4.5. Policy-Management

In der Bachelorarbeit wird angenommen, dass zu jedem Zeitpunkt nur eine Policy aktiv ist. Daher muss definiert werden was passiert, wenn Überschneidungen in den Aktivitätszeiträumen von mehreren Policies entstehen. Zudem muss definiert werden, welchen Einfluss die Priorität einer Policy auf die eigene Aktivität und die anderer hat.
Um die erstellte Lösung besser zu erklären, wird zunächst von einem einfachen Überschreiben der aktuell aktiven suricata-Regeln ausgegangen. In den folgenden Iterationen (I-III) werden die Mängel der Vorgehensweisen analysiert und die implementierte Lösung gezeigt.

(Iteration I) einfaches Überschreiben

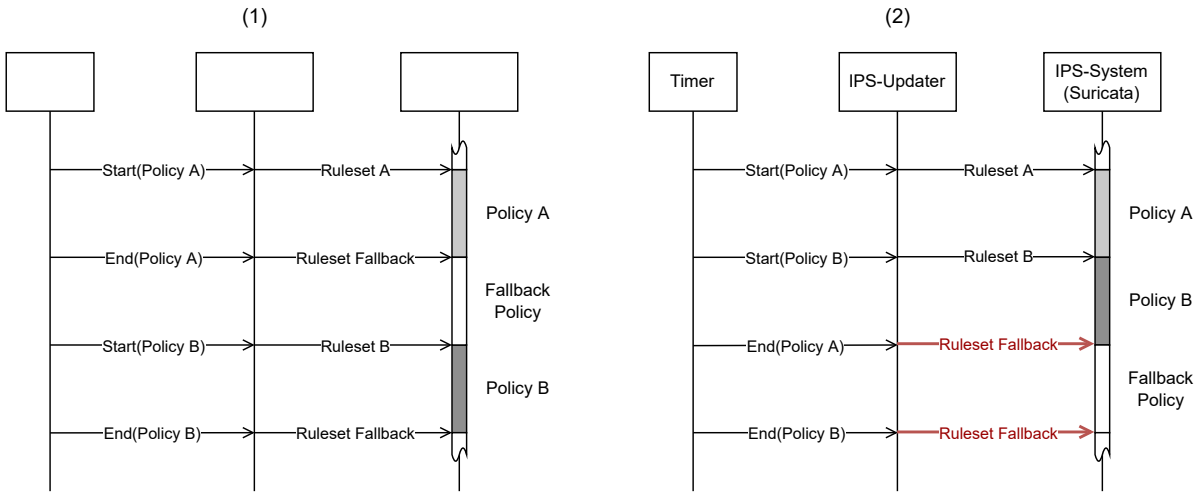

Abbildung 4.7.: Policy-Management Iteration I

Im einfachsten Fall überschreibt jede (De-)Aktivierung einer Policy die aktuell gültigen Suricata-Regeln.
Wenn nun eine Policy A aktiviert wird, werden die aktuellen Suricata-Regeln durch die der Policy A ersetzt. Die Deaktivierung der Policy A überschreibt nun die Suricata-Regeln mit den Regeln der Fallback Policy. Damit ist Policy A inaktiv und die Fallback Policy aktiv. Bei einem Wechsel von Policy A zu Policy B tritt kurzzeitig die Fallback-Policy in Kraft. (Abb. 4.7 (1))

Wenn bei diesem Vorgehen Policy B aktiviert wird während Policy A noch aktiv ist, funktioniert das System nicht mehr korrekt. Bei der Aktivierung von Policy B würde diese korrekt in Kraft treten und die aktuellen Suricata-Regeln (Policy A) durch die der Policy B ersetzen. Wenn nun Policy A deaktiviert werden soll, geht der Enforcer davon aus, dass die aktuellen Suricata-Regeln zu Policy A gehören. So ersetzt er die aktuellen Regeln mit denen der Fallback-Policy. Sobald Policy A deaktiviert wurde ist die Fallback Policy anstelle der Policy B aktiv. (Abb. 4.7 (2))
Dieses Problem wurde durch eine Warteschlange in Iteration II (4.5) behoben.

(Iteration II) activeRulesetQueue

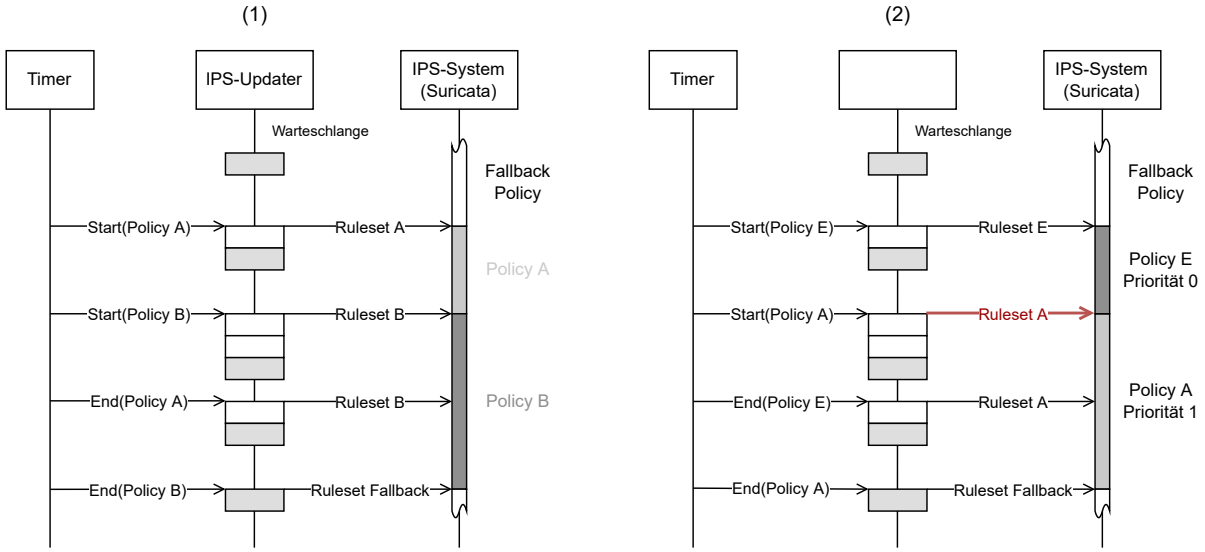

Abbildung 4.8.: Policy-Management Iteration II

Durch Implementation einer Warteschlange (activeRulesetQueue) können Überlappungen zwischen Policies korrekt behandelt werden.
Bei Initialisierung des Enforcers enthält die Warteschlange zunächst nur die Fallback Policy, deren Regeln im Anschluss umgesetzt werden. Wenn nun eine Policy A aktiv werden soll, wird sie der Warteschlange hinzugefügt. Der IPS-Updater liest nun den letzten Eintrag der Warteschlange aus und setzt die Regeln der entsprechenden Policy um. Beim Start einer zweiten Policy B wird auch diese der Warteschlange hinzugefügt und im Anschluss umgesetzt. Wenn nun Policy A beendet wird, wird einfach der Eintrag aus der Warteschlange entfernt. Da Policy B immer noch das letzte Element ist, wird weiterhin Policy B umgesetzt. Ein solcher Ablauf ist in Abb. 4.8 (1) dargestellt.

Bisher wurde davon ausgegangen, dass jede Policy die Priorität 1 (NORMAL) besitzt. Eine Emergency-Policy der Priorität 0 (Policy E) kann in diesem System von einer Policy der Priorität 1 (Policy A) überschrieben werden. Dieses Verhalten ist in der Abbildung 4.8 (2) dargestellt. Um Policies der Priorität 0 (URGENT) korrekt zu behaldeln, wurde das System um eine Prioritätsvariable erweitert (Iteration III (4.5)).

(Iteration III) priority 0 Variable

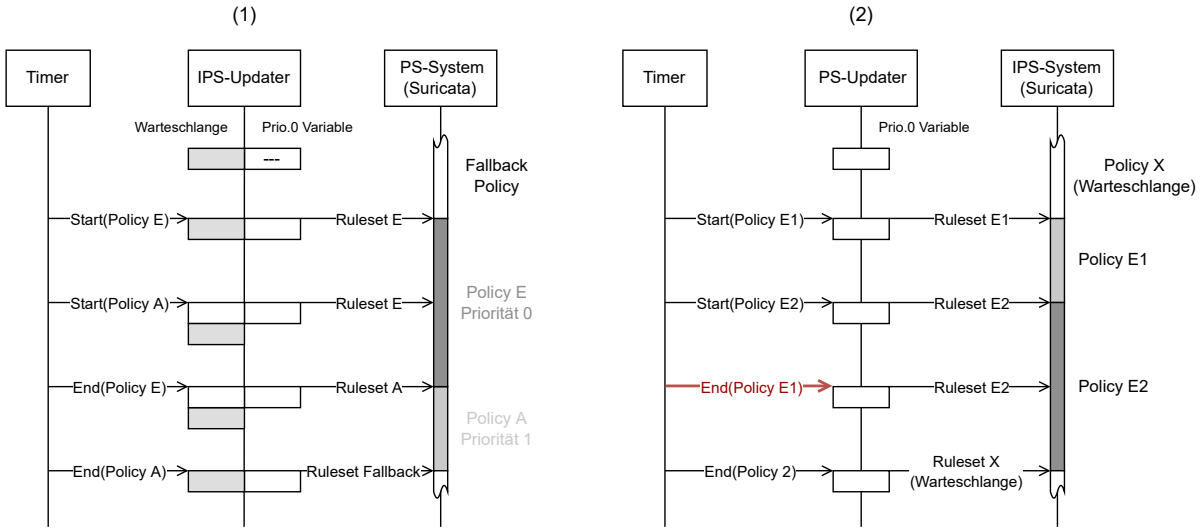

Abbildung 4.9.: Policy-Management Iteration III

Durch Einführen einer Prioritätsvariable werden Policies der Priorität 0 (URGENT) anderen Polices vorgezogen.
Wenn eine Policy der Priorität 0 (URGENT) umgesetzt werden soll, wird sie in diese Variable eingetragen. Immer wenn der IPS-Updater neue Regeln an das IPS-System sendet, prüft er ob die Prioritätsvariable eine Policy enthält. Wenn eine Policy in der Variable eingetragen ist, werden die Regeln dieser Policy verwendet. (Abb. 4.9 (1))
So ist es möglich, eine Policy A der Priorität 1 (NORMAL) zu aktivieren, ohne dass dabei eine aktuell aktive Policy E der Priorität 0 (URGENT) unterbrochen wird. Die Regeln der Policy A werden erst umgesetzt, wenn Policy E deaktiviert wird. (vorausgesetzt Policy A wurde noch nicht deaktiviert) Dieses Verhalten ist in Abbildung 4.9 (1) dargestellt.

Auch hier wurde davon ausgegangen, dass nur eine Policy der Priorität 0 (URGENT) auf einmal aktiv ist.
Genau wie in Iteration I (4.5) kann es mit einer zweiten Policy der Priorität 0 (URGENT) dazu kommen, dass keine der beiden Policies mehr aktiv ist. Anders als in Iteration I (4.5) ist hier bekannt welche Policy gerade aktiv ist. Bei einer Deaktivierung einer Policy der Priorität 0 (URGENT) kann so einfach geprüft werden, ob die zu deaktivierende Policy der Policy in der Prioritätsvariable entspricht. Wenn eine andere Policy eingetragen ist, wird die Variable nicht geändert. So bleibt die neue Policy der Priorität 0 (URGENT) aktiv bis explizit diese deaktiviert wird. Ein solches Szenario ist in Abbildung 4.9 (2) dargestellt.

4.5.1. Erweiterung des Policy-Managements (Ausblick)

Damit der Policy-Enforcer mehrere Policies auf einmal umsetzen kann, muss der IPS-Updater eine Möglichkeit bieten, die Regelsätze aller aktiven Policies zu einem Gesamtregelsatz zusammenzuführen.
Alle aktuell aktiven Policies könnten dabei in einer Liste *currentlyActive* gesammelt und zu dem Gesamtregelsatz zusammengeführt werden.
Auch die Prioritätsvariable müsste durch eine Liste ersetzt werden, welche alle aktiven Policies der Priorität 0 (URGENT) enthält.

5. Prototypische Implementierung

Neben der Entwicklung des Konzepts sollte im Rahmen der Bachelorarbeit ein Prototyp des Policy-Enforcers entwickelt werden. Die Ergebnisse der Implementation werden hier vorgestellt und deren Wirksamkeit nachgewiesen.

5.1. Finaler Implementationsstand

Policy-Receiver: Da die Kommunikationspartner (Authority-Management, Policy-Service) des Policy-Receivers noch nicht implementiert waren, konnte die Interaktion mit diesen Systemen nur in Form eines Mock-Ups umgesetzt werden. Mit Hilfe einer *sleep()* Funktion und der Übergabe von statischen Antworten wurde die Interaktion simuliert.
Die Interaktion mit dem Policy-Service wurde zusätzlich durch ein JSON-Schema zur Definition der zu übertragenden Datenstruktur beschrieben. Das Schema ist im Anhang unter A.1 aufgeführt.
Auch der Policy Checker konnte nur teilweise implementiert werden. Der Check der empfangenen JSON-Struktur gegen das JSON-Schema konnte mit Hilfe der Python-Bibliothek *jsonschema* [13] implementiert werden.

IPS Controller: Der IPS-Controller wurde wie in 4.4.2 beschrieben implementiert.
Bei der Implementierung und dem anschließenden Testen der Komponente fiel auf, dass ein wichtiger Punkt, das Policy-Management, noch nicht definiert war. Unter 4.5 ist das Vorgehen und die erarbeitete Lösung erläutert.
Der Rule Parser unterstützt die in 4.1 beschriebenen Werte. Das Suricata-System kann nur die Aktionen *alert* und *pass* umsetzen, da es im Moment nur im IDS-Modus betrieben werden kann. Auch das Feld *opt* wird bisher nicht ausgewertet.

Timer: Die Komponente Timer konnte funktional umgesetzt werden.
Das Ziel der Komponente war es, betriebssysteminterne Funktionen für das Auslösen eines Alarms zu verwenden um die Ressourcen der DPU möglichst wenig zu belasten. Das konnte in der Zeit der Bachelorarbeit leider nicht auf diese Weise erreicht werden. Stattdessen erstellt der Timer für jeden Alarm einen eigenen Timer-Thread [14] [15] um den Alarm nach einer berechneten Verzögerung auszulösen. Da diese Verzögerung in Sekunden angegeben werden muss, können Policies nur im Sekunden-Takt geändert werden. (Eine Verzögerung von 0 Sekunden wird allerdings unterstützt.)

Attestierung: Die Attestierung konnte ähnlich wie der Policy-Receiver nur als Mock-Up implementiert werden. Hier waren die Datenbank-Systeme, in die attestiert werden sollte, noch nicht vorhanden. Die Funktionen zum Attestieren sind implementiert, auch wenn sie bisher nur eine Konsolen-Ausgabe umfassen.

5.2. Nachweis der Wirksamkeit

In folgendem Testaufbau wurde der Policy-Enforcer gestartet und es wurden zwei Policies nacheinander aktiviert und deaktiviert. Nach dem Start des Policy-Enforcers wurden durch eine Test-Funktion die beiden Policies generiert und an den Policy-Enforcer übergeben.

Die erste Policy `TestPolicy_1` definierte folgende Regeln:

- Erstelle eine Benachrichtigung bei allen ICMP-Paketen
- Erstelle eine Benachrichtigung bei allen Paketen von und zu 192.168.2.99

Sie wurde 10 Sekunden nach dem Empfang der Policies aktiviert und 10 Sekunden später deaktiviert.

Die zweite Policy definierte nur eine Regel:

- Erstelle eine Benachrichtigung bei jedem Paket (allen IP Paketen)

Sie wurde 30 Sekunden nach dem Empfang der Policies aktiviert und 10 Sekunden später deaktiviert.
Mit dem Befehl `telnet` wurden TCP Verbindungen zwischen einem Host im Outside-Netz (192.168.3.42) und einem im Inside-Netz (192.168.2.99) simuliert.

Die folgenden Punkte beschreiben die erwarteten Ausgaben des Prototypen und zeigen mittels Screenshots, ob das System korrekt arbeitet und die Regeln der Policies wirksam umgesetzt wurden.

Konsolen-Ausgaben des Policy-Enforcers

In den Konsolen-Ausgaben des Policy-Enforcers lässt sich gut beobachten, wie sich die einzelnen Komponenten verhalten. Durch einen Zeitstempel und eine Angabe der auslösenden Komponente kann gut nachvollzogen werden wie die der Policy-Enforcer arbeitet.

Wichtige Ausgaben dieses Screenshots sind:

- `[Communicator] querying new Policies from AI` – Hier werden die neuen Policies von der Test-Funktion empfangen.
- `[IPSUpdater] <Policy X> to Dict` – Diese Ausgabe zeigt an, dass Policy X im IPS-Updater zwischengespeichert wurde.
- `[Timer] registered Alarms for <Policy X>` – Diese Ausgabe zeigt an, dass die Start- und End-Zeitpunkte der Policy X registriert wurden.
- `[IPSUpdater] <Policy X> added to Queue` – Diese Ausgabe zeigt, dass Policy X gerade aktiviert wurde.

```
ubuntu@localhost:~/policy_enforcer$ python3 ./Policy_Enforcer.py
10:04:01 [Enforcer]       setting up OvS Bridges
10:04:01 [suricata]       starting ...
10:04:01 [suricata]       subprocess PID: 12754
10:04:01 [AttestHelper]   attested "fallback" to activation-DB
10:04:01 [Communicator]   start
10:04:01 [Communicator]   waiting for new-policy-notice...
10:04:04 [Communicator]   querying new policies from AI ...
10:04:05 [IPSUpdater]     Policy_A added to Dict
10:04:05 [Timer]          registered Alarms for Policy_A:
10:04:05 [Timer]             |-> start: 20.08.2022 10:04:15
10:04:05 [Timer]             '-> end:   20.08.2022 10:04:25
10:04:05 [IPSUpdater]     Policy_B added to Dict
10:04:05 [Timer]          registered Alarms for Policy_B:
10:04:05 [Timer]             |-> start: 20.08.2022 10:04:35
10:04:05 [Timer]             '-> end:   20.08.2022 10:04:45
10:04:15 [IPSUpdater]     Policy_A added to Queue
10:04:15 [AttestHelper]   attested "Policy_A" to activation-DB
10:04:25 [IPSUpdater]     Policy_A removed from Queue
10:04:25 [IPSUpdater]     Policy_A removed from Dict
10:04:25 [AttestHelper]   attested "fallback" to activation-DB
10:04:35 [IPSUpdater]     Policy_B added to Queue
10:04:35 [AttestHelper]   attested "Policy_B" to activation-DB
10:04:45 [IPSUpdater]     Policy_B removed from Queue
10:04:45 [IPSUpdater]     Policy_B removed from Dict
10:04:45 [AttestHelper]   attested "fallback" to activation-DB
^C
10:04:56 [Enforcer]       Recieved SIGINT, stopping Components...
10:04:56 [Communicator]   stop
10:04:56 [suricata]       interrupting (SIGINT) ...
10:04:56 [suricata]       waiting for process to exit ...
10:04:57 [suricata]       exited with Code: 0
10:04:57 [Timer]          canceled all Alarms
ubuntu@localhost:~/policy_enforcer$
```

Abbildung 5.1.: Konsolen-Ausgaben des Policy-Enforcer Prototypen

Der Screenshot zeigt, dass der Policy-Enforcer die Policies korrekt interpretiert und zu den korrekten Zeitpunkten deren Umsetzung initiiert. Die tatsächliche Wirksamkeit der Regeln ist hiermit jedoch noch nicht nachgewiesen.

suricata.rules

Der Inhalt dieser Datei während des Versuchs wird in Abbildung 5.2 gezeigt.
Die `suricata.rules` Datei gibt dem IPS-System vor, welche Regeln es verwenden soll. Die Regeln müssen in dieser Datei in der Suricata-Syntax angegeben werden. Durch regelmäßiges Ausgeben der `suricata.rules` Datei kann beobachtet werden, ob die Regeln der Policies korrekt übersetzt werden.

```
ubuntu@localhost:~/policy_enforcer/suricata$ cat suricata.rules
alert IP any any <> any any (msg:"DAP_fallback_rule_0"; sid: 1000;)
ubuntu@localhost:~/policy_enforcer/suricata$
ubuntu@localhost:~/policy_enforcer/suricata$
ubuntu@localhost:~/policy_enforcer/suricata$ cat suricata.rules
alert ICMP any any -> any any (msg:"DAP_Policy_A_rule_0"; sid: 1000;)
alert IP 192.168.2.99 any <> any any (msg:"DAP_Policy_A_rule_1"; sid: 1001;)
ubuntu@localhost:~/policy_enforcer/suricata$
ubuntu@localhost:~/policy_enforcer/suricata$
ubuntu@localhost:~/policy_enforcer/suricata$ cat suricata.rules
alert IP any any <> any any (msg:"DAP_fallback_rule_0"; sid: 1000;)
ubuntu@localhost:~/policy_enforcer/suricata$
ubuntu@localhost:~/policy_enforcer/suricata$
ubuntu@localhost:~/policy_enforcer/suricata$ cat suricata.rules
alert IP any any <> any any (msg:"DAP_Policy_B_rule_0"; sid: 1000;)
ubuntu@localhost:~/policy_enforcer/suricata$
ubuntu@localhost:~/policy_enforcer/suricata$
ubuntu@localhost:~/policy_enforcer/suricata$ cat suricata.rules
alert IP any any <> any any (msg:"DAP_fallback_rule_0"; sid: 1000;)
ubuntu@localhost:~/policy_enforcer/suricata$
```

Abbildung 5.2.: Inhalt der suricata.rules Datei

Dieser Screenshot zeigt, dass die Regeln korrekt in die Suricata-Systax übersetzt werden. Um die in `suricata.rules` beschriebenen Regeln umzusetzen, muss das IPS-System erst die Regeln neu laden.

suricata.log

Die Konsolen-Ausgaben des Suricata-Prozesses werden in eine Datei geschrieben. Neben dem Starten und Beenden des Prozesses kann beobachtet werden, dass der Suricata-Prozess die Regeln der `suricata.rules` Datei nach jedem Policy-Update neu lädt.

```
ubuntu@localhost:~/policy_enforcer/suricata$ cat suricata.log
20/8/2022 -- 10:04:01 - <Notice> - This is Suricata version 6.0.5 RELEASE running in SYSTEM mode
20/8/2022 -- 10:04:01 - <Notice> - all 8 packet processing threads, 2 management threads initialized,
 engine started.
20/8/2022 -- 10:04:15 - <Notice> - rule reload starting
20/8/2022 -- 10:04:15 - <Notice> - rule reload complete
20/8/2022 -- 10:04:25 - <Notice> - rule reload starting
20/8/2022 -- 10:04:25 - <Notice> - rule reload complete
20/8/2022 -- 10:04:35 - <Notice> - rule reload starting
20/8/2022 -- 10:04:35 - <Notice> - rule reload complete
20/8/2022 -- 10:04:45 - <Notice> - rule reload starting
20/8/2022 -- 10:04:45 - <Notice> - rule reload complete
20/8/2022 -- 10:04:56 - <Notice> - Signal Received.  Stopping engine.
20/8/2022 -- 10:04:57 - <Notice> - Stats for 'p0':  pkts: 44, drop: 0 (0.00%), invalid chksum: 0
ubuntu@localhost:~/policy_enforcer/suricata$
```

Abbildung 5.3.: Ausgaben der Suricata-Anwendung

fast.log

Die Regeln der Policies besagen, dass für bestimmte Pakete des Netzwerkverkehrs eine Benachrichtigung erstellt werden soll. Da parallel zur Ausführung des Policy-Enforcers Pakete durch den DAoS-Controller geleitet wurden, sollten bei Erkennung der Pakte entsprechende Benachrichtigungen ausgegeben werden. Die Benachrichtigungen werden von Suricata in die Datei `fast.log` geschrieben. Der Inhalt der `fast.log` Datei nach Beenden des Policy-Enforcers wird Abbildung 5.4 gezeigt.

```
ubuntu@localhost:~/policy_enforcer/suricata$ cat fast.log
08/20/2022-10:04:08.077204  [**] [1:1000:0] DAP_fallback_rule_0 [**] [Classification: (null)] [Priority: 3]
 {TCP} 192.168.2.99:59360 -> 192.168.3.42:443
08/20/2022-10:04:08.142121  [**] [1:1000:0] DAP_fallback_rule_0 [**] [Classification: (null)] [Priority: 3]
 {TCP} 192.168.3.42:443 -> 192.168.2.99:59360
08/20/2022-10:04:20.177696  [**] [1:1001:0] DAP_Policy_A_rule_1 [**] [Classification: (null)] [Priority: 3]
 {TCP} 192.168.2.99:59362 -> 192.168.3.42:443
08/20/2022-10:04:20.228890  [**] [1:1001:0] DAP_Policy_A_rule_1 [**] [Classification: (null)] [Priority: 3]
 {TCP} 192.168.3.42:443 -> 192.168.2.99:59362
08/20/2022-10:04:29.748484  [**] [1:1000:0] DAP_fallback_rule_0 [**] [Classification: (null)] [Priority: 3]
 {TCP} 192.168.3.42:443 -> 192.168.2.99:59364
08/20/2022-10:04:32.182087  [**] [1:1000:0] DAP_fallback_rule_0 [**] [Classification: (null)] [Priority: 3]
 {UDP} 0.0.0.0:68 -> 255.255.255.255:67
08/20/2022-10:04:40.663446  [**] [1:1000:0] DAP_Policy_B_rule_0 [**] [Classification: (null)] [Priority: 3]
 {TCP} 192.168.2.99:59366 -> 192.168.3.42:443
08/20/2022-10:04:40.722087  [**] [1:1000:0] DAP_Policy_B_rule_0 [**] [Classification: (null)] [Priority: 3]
 {TCP} 192.168.3.42:443 -> 192.168.2.99:59366
08/20/2022-10:04:50.399475  [**] [1:1000:0] DAP_fallback_rule_0 [**] [Classification: (null)] [Priority: 3]
 {TCP} 192.168.3.42:443 -> 192.168.2.99:59370
ubuntu@localhost:~/policy_enforcer/suricata$
```

Abbildung 5.4.: Logeinträge der fast.log Datei

Der Screenshot zeigt für jede Policy zwei Einträge. Die Einträge stehen jeweils für die Verbindungs-Anfrage und die Antwort des Ziels. Da auf dem Ziel kein HTTPS Service (Port 443) läuft, wird die Verbindung verweigert und es werden keine weiteren Pakete übertragen.

5.3. Ergebnisse

Die Policy-Struktur wurde spezifiziert. Das entwickelte Konzept lässt durch ein Data-Feld genügend Spielraum, um es auch für andere Teilsysteme des digitalen Raumes anzuwenden.

Mit dem Regelsatz wurde ein Format geschaffen, um Netzwerkregeln darzustellen. Das Konzept orientierte sich dabei an der Suricata-Syntax. Die Regeln enthalten ein optionales Feld, welches zur Zeit nicht von dem Parser ausgewertet wird. Dieses Feld kann dazu verwendet werden, um das Regelformat wie in Abschnitt 4.3.3 beschrieben zu erweitern.

Da sich das Raumkonzept noch in Entwicklung befindet, war zum Zeitpunkt der Bachelorarbeit der Policy-Service noch nicht vorhanden. So konnte die Kommunikation des Prototypen noch nicht implementiert werden. Für den Austausch von Policies wurde bereits ein JSON-Schema entwickelt, auf dessen Grundlage Policies formuliert und übergeben werden konnten. Im Prototypen wird das erwartete Verhalten des Policy-Services vorerst simuliert.

Die Use-Cases *Policy definieren* und *Umsetzung attestieren* konnten aufgrund fehlender Kommunikations-Endpunkte nicht umgesetzt werden. Für die Kommunikation wurden Mock-Up-Funktionen implementiert und die Attestierung mittels Konsolenausgaben simuliert.

Die Entwicklung eines funktionierenden Prototypen, und damit der Use-Case *Policy umsetzen*, gelang mit einigen Einschränkungen. Die Suricata-Installation gelang nur im IDS-Modus, wodurch der Netzwerkverkehr durch die DPU nur betrachtet und nicht verändert werden konnte. Verbindungen zwischen beiden Netzen ließen sich noch nicht blockieren.

Die korrekte Umsetzung von Policies unter dem beschriebenen Policy-Management konnte implementiert werden. Auch das Aktualisieren des Suricata-Systems während der Laufzeit funktioniert ohne Probleme. Die gesammelten Log-Daten werden lokal in einer Datei *fast.log* (siehe Abb. 5.4) gesichert.

Die Wirksamkeit des Policy-Enforcers konnte mittels Test-Funktionen in Abschnitt 5.2 nachgewiesen werden.

6. Fazit

6.1. Zusammenfassung

Das Ziel der Arbeit war es, ein Policy-Konzept für die Steuerung des Netzwerkverkehrs zu entwickeln und dessen Wirksamkeit anhand einer Beispiel-Policy nachzuweisen.
Die Arbeit wurde in folgende Aufgaben unterteilt:

- Entwicklung einer Policy- und Regel-Struktur
- Entwicklung eines Protokolls für den Austausch von Policies zwischen Autorität und DAoS-Controller
- prototypische Implementierung des Kommunikationsendpunkts des DAoS-Controllers
- Umsetzung der Policy von einer Datenflusssteuerung
- Attestierung von Ereignissen an die Wissensdatenbank.

Die Aufgaben und das erwartete Verhalten des zu implementierenden Programms wurden zunächst mittels Use-Cases definiert. So wurde klar, welche technischen Akteure welche Aufgaben zu übernehmen hatten.
Anschließend wurde die Policy entwickelt. Hier wurde zunächst eine allgemeine Policy-Struktur entwickelt. Um den Netzwerkverkehr an der Außengrenze einer DAoS zu steuern wurde eine Regelsatz-Struktur entwickelt. Durch Zusammenführen der allgemeinen Policy-Struktur und der Regelsatz-Struktur entstand eine Netzwerk-Policy zur Regelung einer Datenflusssteuerung.
Nach der Konzeptionsphase wurde ein Prototyp des Policy-Enforcers mit zugehöriger Datenflusssteuerung implementiert. Während der Implementierung wurde deutlich, dass in der Konzeptphase noch nicht ausreichend das Problem sich überschneidender Policies betrachtet wurde. Das Konzept wurde daher um den Punkt *Policy-Management* erweitert.
Nach erfolgreicher Implementation eines Prototypen wurde dessen Wirksamkeit getestet und so nachgewiesen.

Das Ziel der Bachelorarbeit konnte so größtenteils erreicht werden. Ein geeignetes Policy-Konzept mit zugehörigem Regelsatz wurden in den Abschnitten 4.3 und 4.3.1 spezifiziert.
Das Umsetzen einer Policy an einem Grenzpunkt des Raums wurde in Kapitel 4.4 beschrieben und in Kapitel 5 implementiert.
Anhand einer Beispiel-Policy wurde in Kapitel 5.2 die Wirksamkeit der Policy-Umsetzung erbracht. Die Übertragung von Policies zwischen Digitaler Autorität und Grenzpunkt konnte jedoch nur spezifiziert und nicht implementiert werden.

6.2. Ausblick für weitere Implementierungsschritte

Mit der Weiterentwicklung des Raumkonzepts muss auch der Policy-Enforcer fertiggestellt und erweitert werden.

Der Typ der Policy wird im aktuellen Konzept in der ID der Policy codiert. Besser wäre es, ein Feld *Type* für den Typ der Policy vorzusehen. So könnte die ID frei gewählt werden und der Typ der Policy könnte einfacher festgestellt werden.

Durch Hinzufügen eines Feld *Name* zu jeder Regel können Paket-Matches besser identifiziert werden. In der aktuellen Implementation wird der Name der Regel aus der ID der Policy und dem Index der Regel im Regelsatz gebildet. (siehe Abb. 5.2) Diese Bezeichnug ist nicht gut lesbar und setzt voraus, dass die Reihenfolge der Regeln innerhalb des Regelsatzes bekannt ist. Das beschriebene Feld *Name* würde das beheben.

Für eine vollständige Implementierung der Attestierung fehlen noch Schnittstellen zur Wissens-Datenbank der Digitalen Autorität. Im aktuellen Prototypen wurden Funktionen bereitgestellt, die später das Senden der Atteste beinhalten sollen.

Der DAoS-Controller benötigt zukünftig noch eine Funktion zur Umleitung des Netzwerkverkehrs, so dass bestimmte Anfragen auf bestimmte Netzwerkziele ausgerichtet werden. Der Regelsatz lässt sich um eine *redirect* Funktion erweitern. Die Umsetzung dieser Aktion ist nicht mehr mit einem gängigen IPS-System möglich. Eine passende Technologie, um auch diese Aktion zu ermöglichen, müsste daher noch gefunden werden.

Über die Arbeit hinweg wurde eine mögliche Erweiterung für das Ausführen mehrerer Policies erläutert. Der Policy-Enforcer würde dabei aus den Regeln aller aktiven Policies einen Gesamtregelsatz erstellen. Dieser kann wie bisher an das IPS-System übergeben und umgesetzt werden. Eine Policy beschreibt so nicht alle umzusetzenden Regeln des Raums, sondern konzentriert sich nur auf einen bestimmten Aspekt des Netzwerkverkehrs. Die Umsetzung dieser Erweiterung wurde in den Textpassagen 4.3.3, 4.4.3 und 4.5.1 erläutert.

In der Bachelorarbeit wurden bisher nur die externen DAoS-Controller des Raums betrachtet. DAoS-Controller sind auch für die Steuerung des internen Netzwerkverkehrs zuständig. Um einen solchen internen DAoS-Controller zu entwickeln muss zunächst untersucht werden, welche Anforderungen es an die Überwachung des internen Netzwerkverkehrs gibt und inwiefern sie sich von denen der externen DAoS-Controller unterscheiden.

Literaturverzeichnis

[1] J. Kebbedies and M. Mundt, "Konzeptansatz: Digitaler raum - grundlage strukturierter digitaler souveränität." Sept. 2021.

[2] H. Ullrich, "Praktikumsbericht." Zitat - Seite 4, May 2022.

[3] H. Ullrich, "Praktikumsbericht." Zitat - Seite 15, May 2022.

[4] "tcpdump ■ Wiki ■ ubuntuusers.de." https://www.tcpdump.org/manpages/pcap-filter.7.html, June 2022. [Online; accessed 28. Jun. 2022].

[5] "IBM Docs." https://www.ibm.com/docs/en/qsip/7.4?topic=queries-berkeley-packet-filters, Feb. 2022. [Online; accessed 29. Jun. 2022].

[6] "6.1. Rules Format — Suricata 6.0.1 documentation." https://suricata.readthedocs.io/en/suricata-6.0.1/rules/intro.html, Jan. 2021. [Online; accessed 28. Jun. 2022].

[7] "Network Policies." https://kubernetes.io/docs/concepts/services-networking/network-policies, June 2022. [Online; accessed 28. Jun. 2022].

[8] "6.1. Rules Format — Suricata 6.0.1 documentation." https://suricata.readthedocs.io/en/suricata-6.0.1/rules/intro.html#protocol, Jan. 2021. [Online; accessed 1. Jul. 2022].

[9] "6.1. Rules Format — Suricata 6.0.1 documentation." https://suricata.readthedocs.io/en/suricata-6.0.1/rules/intro.html#source-and-destination, Jan. 2021. [Online; accessed 1. Jul. 2022].

[10] H. Ullrich, "Praktikumsbericht." May 2022.

[11] "NVIDIA DOCA DPI Compiler." https://docs.nvidia.com/doca/sdk/dpi-compiler/index.html#description, May 2022. [Online; accessed 28. Jun. 2022].

[12] "7.3. Rule Reloads — Suricata 6.0.1 documentation." https://suricata.readthedocs.io/en/suricata-6.0.1/rule-management/rule-reload.html, Jan. 2021. [Online; accessed 28. Jun. 2022].

[13] "jsonschema 4.9.0 documentation." https://python-jsonschema.readthedocs.io/en/latest/#jsonschema, July 2022. [Online; accessed 1. Aug. 2022].

[14] "threading — Thread-based parallelism — Python 3.10.5 documentation." https://docs.python.org/3/library/threading.html#timer-objects, Aug. 2022. [Online; accessed 2. Aug. 2022].

[15] "How to Perform Threading Timer in Python." https://www.section.io/engineering-education/how-to-perform-threading-timer-in-python, Aug. 2022. [Online; accessed 2. Aug. 2022].

Anhang

Anhang A.

JSON-Schema

Listing A.1: JSON-Schema zum Übertragen der Policies

```
{
    "$schema": "https://json-schema.org/draft/2019-09/
       schema",
    "description": "DAP Policy JSON Schema - describes all
       Policies to be enforced",
    "type": "array",
    "items": { "$ref": "#/$defs/Policy" },
    "$defs": {
       "Policy": {
          "type": "object",
          "properties": {
             "ID": {
                "type": "string" },
             "Publisher": {
                "type": "string" },
             "Priority": {
                "type": "number",
                "enum": [ 0, 1, 2 ] },
             "Validity-Start": {
                "type": "string",
                "format": "date-time" },
             "Validity-End": {
                "type": "string",
                "format": "date-time" },
             "Rules": {
                "type": "array",
                "items": { "$ref": "#/$defs/Rule"}
             },
             "Signatur": { "type": "string" }
          },
```

```
29          "required": [ "ID", "Publisher", "Priority", "
               Validity-Start", "Validity-End", "Rules", "
               Signatur" ],
30          "additionalProperties": false
31       },
32       "Rule": {
33       "type": "object",
34          "items": false,
35          "properties": {
36             "Action": {
37                "type": "string" },
38             "Protocol": {
39                "type": "string" },
40             "SourceIP": {
41                "type": "string",
42                "format": "regex" },
43             "SourcePort": {
44                "type": "string",
45                "format": "regex" },
46             "DestinationIP": {
47                "type": "string" },
48             "DestinationPort": {
49                "type": "string" },
50             "Direction": {
51                "type": "string",
52                "enum": [ "uni", "bi" ] },
53             "Option": {
54                "type": "string" }
55          },
56          "required": [ "Action", "Protocol", "SourceIP", "
               SourcePort", "DestinationIP", "DestinationPort
               ", "Direction"
57          ],
58          "additionalProperties": false
59       }
60    }
61 }
```

Selbstständigkeitserklärung

Hiermit erkläre ich, daß ich die vorliegende Arbeit selbstständig angefertigt, nicht anderweitig zu Prüfungszwecken vorgelegt und keine anderen als die angegebenen Hilfsmittel verwendet habe. Sämtliche wissentlich verwendete Textausschnitte, Zitate oder Inhalte anderer Verfasser wurden ausdrücklich als solche gekennzeichnet.

Mittweida, den 22. August 2022

Henning Ullrich