Fachhochschule Hannover - Fachbereich Informatik
Studiengang Angewandte Informatik

Bachelorarbeit

Intrusion-Detection mit der Hilfe von CEP-basierten Systemen

von

Marius Rohde 1091575

31. August 2011

Erstprüfer

Prof. Dr. Ralf Bruns
Fachhochschule Hannover
Ricklinger Stadtweg 120
30459 Hannover
E-Mail: ralf.bruns@fh-hannover.de

Zweitprüfer

Johannes Westhuis M. Sc.
Fachhochschule Hannover
Ricklinger Stadtweg 120
30459 Hannover
E-Mail: johannes.westhuis@fh-hannover.de

Autor

Marius Rohde
Charlottenstr. 85
30449 Hannover
E-Mail: marius.rohde@stud.fh-hannover.de

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die eingereichte Bachelorarbeit selbständig und ohne fremde Hilfe verfasst, andere als die von mir angegebenen Quellen und Hilfsmittel nicht benutzt und die den benutzten Werken wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Hannover, den 31. August 2011

_____
Marius Rohde

Inhaltsverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

CEP	-	Complex Event Processing
CIDF	-	Common Intrusion Detection Framework
CPU	-	Central Processing Unit
EPA	-	Event Processing Agent
EPN	-	Event Processing Network
GUI	-	Graphical User Interface
HIDS	-	Host-based Intrusion Detection System
IDS	-	Intrusion Detection System
IP	-	Internet Protocol
IPS	-	Intrusion Prevention System
JVM	-	Java Virtual Machine
NIDS	-	Network-based Intrusion Detection System
SMTP	-	Simple Mail Transfer Protocol
TCP	-	Transmission Control Protocol

1 Einleitung

Netzwerke von Unternehmen und Regierungen unterliegen täglich den Angriffen von Hackern. Der Schaden, der durch Hacker-Angriffe angerichtet wird, lässt sich teilweise nur schwer und erst lange Zeit nachdem der Angriff erfolgte feststellen. Nach einer Studie des US-Marktforschungsunternehmens Ponemon Institute beläuft sich der Schaden durch Onlinekriminalität pro befragtem Unternehmen im Jahr 2010 durchschnittlich auf 5,9 Millionen US-Dollar [Pin01]. Zusätzlich zu den direkten Kosten kommen unschätzbare Werte, die durch den Verlust des Rufes entstehen, wenn z.B. die Entwendung von Kundendaten bekannt wird.

Schon vor langer Zeit haben Administratoren und Sicherheitsexperten damit angefangen, kleine und große Programme zu entwickeln, um Hacker-Aktivitäten und andere unerwünschte Vorgänge in Netzwerken zu erkennen. Diese Programme zur Netzwerküberwachung entwickelten sich bis heute zu den Intrusion-Detection-Systemen (IDS). Diese Systeme nutzen Regeln, statistische Methoden und neuronale Netze, um Angriffe zu erkennen. Das ständig wachsende Datenaufkommen, die komplexer werdenden Netzstrukturen und die Vielzahl verschiedenster neuer Angriffsszenarien stellen an IDS hohe Anforderungen. Besonders die Agilität und Flexibilität der Anpassung an die neuen Anforderungen entscheidet darüber, wie sicher Angriffe erkannt werden. Die Hersteller von IDS suchen ständig neue Wege, um die Fehlerrate der Systeme zu minimieren.

Eine andere Technologie, die sich in den letzten Jahren entwickelt hat, ist Complex-Event-Processing (CEP). Auch diese Technik benutzt Regeln, um das Geschäftswissen der Anwendung zu speichern. Im Mittelpunkt von Complex-Event-Processing stehen die Ereignisse, die zum Beispiel gefiltert zu neuen komplexeren Ereignissen zusammengefügt oder mit Informationen angereichert werden kön-

nen. Die Regeln legen diese Vorgänge fest. Ereignisse werden in einem kontinuierlichen Strom betrachtet. Sie besitzen fachliche, zeitliche und örtliche Beziehungen zueinander. Besonders die zeitnahe Verarbeitung sowie die flexible Anpassung und Kontrollierbarkeit der Geschäftsprozesse wird von CEP-Systemen unterstützt.

Da die Anforderungen an CEP-Systeme denen an IDS gleichen, ist es eine interessante Frage, wie gut sich CEP mit IDS verbinden lässt und welche Vor- oder Nachteile der neue Ansatz ergibt. Um dies zu testen, wurde eine Intrusion-Detection-Software geschrieben, die Netzwerkangriffe mit der Hilfe von CEP-Techniken erkennt. Anhand dieser Software wird analysiert, wie sich der Aufbau eines IDS durch die Komponenten eines CEP-Systems integrieren lässt. Außerdem wird mittels eines Anwendungsfalls für Intrusion-Detection die Eignung des Regelumfangs einer ausgewählten CEP-Software überprüft.

Das Kapitel 2 führt die beiden Technologien Complex-Event-Processing und Intrusion-Detection ein. Der CEP-Abschnitt erklärt die Sichtweise und Besonderheiten von CEP. Außerdem erläutert er den allgemeinen Aufbau sowie gängige Begriffe eines CEP-Systems. Im IDS-Abschnitt wird beschrieben, wofür IDS nötig sind. Dieser Abschnitt stellt ebenfalls den üblichen Aufbau und die Begriffe eines IDS vor. Zusätzlich beschreibt er Analyseverfahren, die in IDS Anwendung finden. Ein weiterer Punkt in Kapitel 2 ist die Betrachtung anderer Arbeiten im Kontext von IDS, die mithilfe von CEP umgesetzt wurden.

Das Kapitel 3 stellt zunächst die Anwendungsfälle vor, die für die Testimplementierung ausgewählt wurden. Anschließend wird anhand der in Kapitel 2 vorgestellten Aufbauten von IDS- und CEP-Systemen ein Entwurf für das Testsystem beschrieben. Zudem gibt der fachliche Entwurf Aufschluss darüber, wie die Anwen-

dungsfälle umgesetzt werden. Der Implementierungsteil zeigt die konkrete Umsetzung des Entwurfs. Letztlich beschreibt der Testabschnitt die Überprüfung, ob die Software die Anwendungsfälle richtig umsetzt.

Kapitel 4 und 5 beinhalten eine Bewertung der Software, sowie ein Fazit über die Anwendbarkeit von CEP in IDS. Dabei bezieht sich Kapitel 4 auf die Bewertung der Software selber und den Vergleich zu anderen IDS. Kapitel 5 hingegen gibt neben der Bewertung der grundsätzlichen Eignung von CEP für IDS einen Ausblick für weitere Ansätze.

2 Grundlagen

Dieses Kapitel gibt eine kurze Einführung in die Grundlagen von Complex-Event-Processing und Intrusion-Detection-Systemen. Es werden die Begriffe und Konzepte eingeführt, die in den weiteren Kapiteln Verwendung finden. Außerdem wird vorgestellt wie CEP-Ansätze in IDS bereits eingesetzt werden.

2.1 Einführung CEP

Der Begriff Complex-Event-Processing steht für eine Technologie zur Verarbeitung von komplexen Ereignissen. Dabei steht die kontinuierliche und zeitnahe Verarbeitung von großen Datenmengen im Vordergrund. Der Begriff wurde von David Luckham in seinem Buch [Luc02] eingeführt.

Ereignisse sind in diesem Kontext alle Aktivitäten, Vorgänge oder Entscheidungen, die ein Softwaresystem verarbeiten soll. Ereignisse können z.B. durch einen Aktientransfer in einem Bankunternehmen, durch den Netzwerkverkehr im Unternehmensnetzwerk oder durch eine Lichtschranke an einem Förderband in der Fabrikhalle ausgelöst werden. Ereignisse spielen zur Steuerung von Geschäftsprozessen von Unternehmen eine wichtige Rolle. Meist haben Ereignisse Beziehungen zueinander. Die in Beziehung stehenden Ereignisse bilden dabei ein Ereignismuster oder englisch event-pattern [Luc01]. Das CEP-System stellt Operationen bereit, um erkannte Ereignismuster in komplexere Ereignisse zu abstrahieren. Durch die Abstraktion kann einem Ereignis eine direkte fachliche Bedeutung zugeordnet werden, die dann als Entscheidungsgrundlage für weiterführende Vorgänge herangezogen wird [BD01].

Was ist das Besondere an CEP ?

Mit CEP als Softwaretechnologie werden Ereignisse als zentrale Informationseinheit in der Softwarearchitektur gesehen. Dadurch können Geschäftsprozesse realitätsnah nachgebildet werden und unterstützen somit die Verständlichkeit des Softwaresystems für den Menschen. Die Anforderungen an die Flexibilität und Agilität von Unternehmen wächst von Jahr zu Jahr, was dazu führt, dass sich auch die Unternehmensprozesse den neuen Anforderungen anpassen müssen. Konventionelle Softwaresysteme sind eher starr im Gegensatz zu CEP orientierten Softwaresystemen. Die Flexibilität und Agilität von CEP-Systemen gründet in der Entkopplung der Geschäftsprozesslogik vom restlichen System. Traditionelle Softwaresysteme besitzen die Geschäftslogik fest verdrahtet im Code, wo hingegen CEP-Systeme das Wissen über die Prozesse in den Regeln vorhalten. Die Anpassungen an neue Anforderungen kann in den meisten Fällen ohne teure Neuentwicklungen oder Änderungen der Software erfolgen. Die Regeln können jederzeit dynamisch verändert und neu hinzugefügt werden. Neben der Flexibilität und Agilität bestehen weitere Anforderungen an Softwaresysteme, wie die Echtzeitfähigkeit oder Massendatenverarbeitung. Auch diese beiden Punkte decken CEP-Systeme gut ab. Eine ausführliche Betrachtung dieser und weiterer Punkte kann in [BD01] nachgeschlagen werden.

Was sind die Hauptunterschiede zu einer klassischen Rule-Engine (Business-Rule-Engine)?

CEP-Engines besitzen immer zustandsorientierte Sitzungen (stateful-session), die auf eine lange Laufzeit und den kontinuierlichen Eingang von Ereignissen ausgelegt sind. Klassische Rule-Engines können durchaus auch im zustandslosen Modus betrieben werden. Dies ist möglich, wenn einzelne Fakten gegen eine Regelbasis geprüft werden sollen. Der weitaus wichtigere Punkt ist die Verarbeitung von Ereignissen im zeitlichen Kontext. Eine Rule-Engine kennt

keine Zeit. Dadurch können keine temporalen Bedingungen zwischen Ereignissen mit Rule-Engine eigenen Funktionen erkannt werden. Auch der Ansatz, neue Ereignisse aus mehreren Einzelereignissen zu generieren und dadurch ein höheres Abstraktionsniveau zu erreichen, ist neu. Weitere Unterschiede werden in [Etz01] und in [Vin01] diskutiert.

2.1.1 Begriffserläuterungen und Aufbau von CEP-Systemen

Im Zusammenhang mit CEP werden in dieser Arbeit weitere Begriffe genannt, die der folgende Abschnitt beschreibt. Zudem verdeutlicht die Abbildung 1 den schematischen Gesamtaufbau eines CEP-Systems und bringt einige Begriffe in Verbindung zueinander. Anhand dieser Abbildung können die Beziehungen zwischen den grundsätzlichen Komponenten eines CEP-Systems nachvollzogen werden.

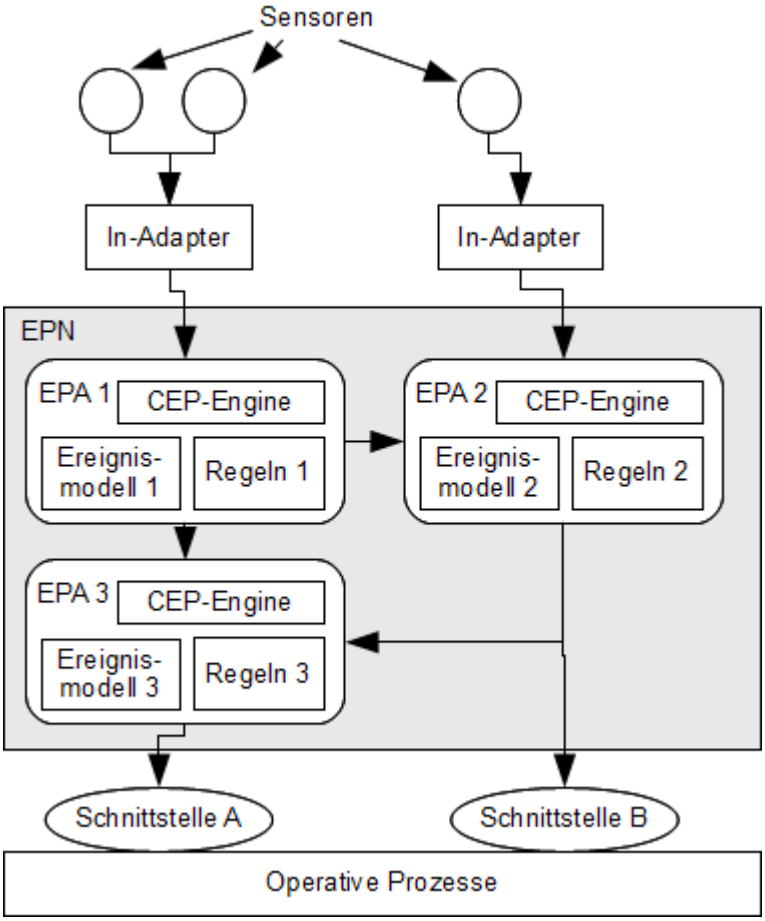

Abbildung 1: Schema CEP-System

Eine **CEP-Engine** ist ein Teil der konkreten Implementierung des CEP-Konzepts und übernimmt die Ereignisverarbeitung. Zusätzlich zur Ereignisverarbeitung gibt es die **Ereignisquellen** und die **Ereignissenken** [Luc02]. Eine Ereignisquelle ist eine beliebige Entität, die Ereignisse sendet. Dazu gehören **Sensoren,** die einfache Ereignisse senden, sowie die Verarbeitungskomponenten selbst. Die Verarbeitungskomponenten generieren in der Regel komplexere Ereignisse und fügen diese in neue Ereignisströme ein. Ereignissenken empfangen Ereignisse. Diese können zum Beispiel das CEP-System selbst, andere Anwendungen oder grafische Benutzeroberflächen sein.

Die **In-Adapter** übernehmen die Aufgabe, Ereignisse bei den Sensoren oder anderen Anwendungen abzuholen. Da die Daten nicht in einem für das CEP-System lesbarem Datenformat vorliegen, transformiert der In-Adapter die Daten. Anschließend sendet er die Ereignisse an die CEP-Komponente [BD01].

Die CEP-Engine bildet zusammen mit dem **Ereignismodell** und den **Ereignisregeln** einen **Event-Processing-Agent (EPA),** auch **CEP-Komponente** genannt. Bevor Ereignisse erkannt werden können, müssen sie genau spezifiziert werden. Die Spezifikation besteht aus Pflichtangaben und fachlichen Zusatzattributen. Zu den Pflichtangaben gehören eine Ereignis-ID, der Ereignistyp, ein Zeitstempel und die Quelle. Die fachlichen Attribute beinhalten die Nutzdaten [BD01]. Mit dem Ereignismodell werden zusätzlich die Beziehungen und Vererbungen zwischen den spezifizierten Ereignistypen deutlich. Die Ereignisregeln definieren, welche Ereignisse des Modells von der CEP-Engine verarbeitet werden. Zusätzlich bestimmen sie die Art und Weise der Ereignisverarbeitung. Die Regeln spiegeln somit die fachliche Logik der Anwendung wieder.

Die CEP-Engine hat die Aufgabe, den kontinuierlich einfließenden Ereignisstrom auf die in den Regeln spezifizierten Ereignismuster zu überprüfen. Tritt ein Muster auf, so kümmert sie sich um die Ausführung der Aktion, die für dieses Muster angegeben ist. Auch die Ausführungsreihenfolge der Regeln wird von der CEP-Engine verwaltet. Ereignisse werden meistens in einem **Zeitfenster** betrachtet, wodurch die CEP-Engine gezwungen ist, Ereignisse zwischenzuspeichern. Die intelligente Verwaltung der Speichernutzung ist deshalb eine weitere wichtige Aufgabe.

Ein **Event-Processing-Network** (EPN) ist ein Zusammenschluss von mehreren EPAs [Luc02]. Ein EPN kann mehreren Zwecken dienen. Eine mögliche Anwendung ist die Lastverteilung. Die Verteilung von EPAs auf verschiedene Rechner behebt Performanzprobleme schnell und einfach. In den meisten Fällen dient ein

EPN zur logischen Aufteilung der Regeln. Der Grundsatz dieser Überlegung ist die einfachere Verständlichkeit mehrerer kleiner hintereinandergeschalteter Regelsätze, im Gegensatz zu einem Großen.

CEP-Schnittstellen sind Ausgangspunkte für Ereignisse aus dem Ereignisstrom. Die wichtigen Informationen, die aus den Ereignisströmen gewonnen werden konnten, sind somit über diese Schnittstellen an andere Komponenten und Softwaresysteme übermittelbar. Dabei ist zu beachten, welche Formate die Empfänger annehmen. Zudem muss die CEP-Schnittstelle wissen, welche Ereignisse auf welche Methode abgebildet werden [BD01].

2.1.2 CEP-Designmuster

Die nachfolgend vorgestellten Designmuster beschreiben allgemeingültige Konzepte, die in CEP basierten Systemen zum Einsatz kommen. Die Auswahl begrenzt sich auf die in dieser Arbeit verwendeten. Diese und weitere Designmuster werden auch in [BD01] vorgestellt.

Das **Filtern von Ereignissen** beschreibt den Vorgang, bei dem nur für die nachfolgenden Verarbeitungsschritte notwendigen Ereignisse aus dem Ereignisstrom extrahiert werden. Die Filterung kann dabei nach beliebigen Kriterien erfolgen, vorausgesetzt die CEP-Engine unterstützt die Definition der für den Anwendungsfall nötigen Ereignismuster. Dieser Schritt reduziert die Ereignismenge für die nachfolgenden EPAs.

Mit **Content-Based-Routing** ist die Aufteilung eines Ereignisstroms durch den Typ oder die Attributwerte der Ereignisse gemeint. Die resultierenden Teilströme besitzen dadurch Ereignisse mit einem für den EPA zugeschnittenen Informationsgehalt. Ein EPA braucht dadurch keine irrelevanten Regeln für andere Ereignistypen zu führen. Content-Based-Routing fördert damit die Kohäsion und Modularität der EPAs in einem EPN.

Reduktionsfenster verringern die zu betrachtende Ereignismenge in einem Ereignisstrom. Sie werden in zwei Kategorien unterteilt. Zeitliche Reduktionsfenster werden durch einen Zeitraum definiert. Längen- Reduktionsfenster geben die Anzahl zu betrachtender Ereignisse an.

Ein **Granularitäts-Shift** impliziert einen Aggregationsschritt, in dem die Daten von Einzelereignissen zu einem neuen komplexen Ereignis zusammengefasst werden. Die Einzelereignisse unterliegen dabei typischerweise einem fachlichen Kontext, der über einen Zeitraum oder eine bestimmte Menge von Ereignissen besteht (vgl. Reduktionsfenster). Im allgemeinen werden einfache Aggregatfunktionen von den Herstellern der CEP-Engines mitgeliefert. Zu den üblichen Funktionen zählen die Summierung, das Minimum, das Maximum und die Zählung von Attributwerten bzw. Ereignistypen. Die neu erzeugten komplexen Ereignisse speichern die aggregierten Daten in ihren Attributen.

Der **Semantik-Shift** fasst, auf der Grundlage definierter Ereignismuster, verschiedenste Ereignisse zu einem neuen komplexen Ereignis zusammen. Mit den neuen Ereignissen kann somit die Information über den Fund eines Musters codiert werden. Da viele Einzelereignisse erst zusammen einen Sinn ergeben, hat die Information über den Fund eines Musters eine große Bedeutung. Durch die erzeugten komplexen Ereignisse können die Informationen gespeichert und anschließend weiterverarbeitet werden.

2.2 Einführung IDS

Angesichts der Risiken, die durch die Vernetzung von Rechnersystemen entstehen, müssen sich alle Teilnehmer des Internets und anderer Netzwerke Gedanken über ihre Sicherheitsmaßnahmen machen. Besonders die drei Punkte Prävention, Detektion und Reaktion sollten dabei berücksichtigt werden [Spe01]. Präventive Maßnahmen sind zum Beispiel der Einsatz von Firewalls, ständige Updates der

Software oder auch die Definition und Einhaltung von Sicherheitsrichtlinien. Die Installation eines IDS ist zwar auch eine präventive Maßnahme, die Systeme übernehmen aber, wie der Name schon sagt, die Aufgabe der Detektion. Reaktionen können unter anderem der Stopp des Einbruchs, die Dokumentation des Zustands oder die Alarmierung anderer Systeme und Personen sein.

Intrusion-Detection-Systeme erkennen Einbrüche und andere ungewöhnliche Ereignisse (folgend als Intrusion bezeichnet) in Computernetzwerken und -systemen. Ralf Spenneberg unterteilt Intrusion-Detection-Systeme in seinem Buch [Spe01] in zwei Arten: netzwerkbasierte IDS (NIDS) und rechnerbasierte IDS (HIDS).

Netzwerkbasierte IDS sind Systeme, die den Netzwerkverkehr analysieren und bei unerlaubten oder ungewöhnlichen Paketen eine Einbruchsmeldung versenden.

Rechnerbasierte IDS überwachen die Rechner in einem Netzwerk. Das Intrusion-Detection-System versucht anhand von Integritätsprüfungen der Dateien, der Analyse von Protokollen und der Überwachung des Betriebssystems gefährliche Aktivitäten zu erkennen und zu melden.

Vorhandene Sicherheitsstrukturen aus Firewalls und Virenscannern können durch den Einsatz von IDS sinnvoll ergänzt werden. Systeme werden evtl. trotz Antivirenprogramm infiziert. Dies kann leicht passieren, wenn das Antivirenprogramm die neuesten Signaturen für den Virus nicht kennt. IDS melden in diesem Fall die Änderungen an den Dateien, die durch den Virus angegriffen werden. Firewalls sind evtl. falsch konfiguriert und lassen unerlaubte Dinge zu. Durch IDS können falsche Konfigurationen erkannt und anschließend behoben werden. Viele Angriffe kommen auch aus dem eigenen Unternehmensnetzwerk. Firewalls sind in diesen Fällen meistens machtlos, da sie nur den Verkehr überwachen, der von innen nach außen oder von außen nach innen gesendet wird. IDS decken diese Lücke ab.

Aber auch IDS sind nicht unfehlbar und erzeugen deshalb regelmäßig falsche Alarmmeldungen. Die Alarmmeldungen werden in zwei Kategorien eingeteilt. Es gibt falsch-positiv-Meldungen und falsch-negativ-Meldungen.

Die **falsch-positiv-Meldungen** bezeichnen eine Alarmierung für eine Intrusion, die in Wirklichkeit keine Intrusion ist. Zu viele dieser falschen Alarmmeldungen können den Benutzer stören, so dass er Alarmmeldungen in Zukunft ignoriert. Die Funktion eines IDS wäre somit außer Kraft gesetzt.

Die **falsch-negativ-Meldungen** bezeichnen eine Intrusion, die nicht gemeldet wird. Diese Fehler stellen ein großes Risiko dar. Besonders wenn sich Benutzer auf die IDS verlassen, kann es dazu kommen, dass die Intrusion erst spät oder gar nicht erkannt wird.

Es gibt verschiedene Lösungsansätze, mit denen eine Intrusion erkannt werden kann. Zum einen gibt es die knowledge-based Intrusion-Detection und zum anderen die behavior-based Intrusion-Detection. Beide Lösungsansätze ergänzen sich gegenseitig.

Bei der **knowledge-based Intrusion-Detection** bzw. **Misuse-Detection** wird eine Intrusion anhand von Mustern, die in dem IDS hinterlegt sind, erkannt. Unerlaubte Ereignisse, die nicht in dem IDS hinterlegt sind, erkennt das System als normale Ereignisse. Daraus ergibt sich, dass die Systeme nur dann gut arbeiten, wenn die Daten über die verschiedenen Angriffe vollständig sind. Ein ständiger Aktualisierungsprozess der Wissensbasis ist somit nötig. Ein großer Vorteil dieses Verfahrens ist die geringe Rate der falsch-positiv-Alarme und die genaue Bestimmung der Angriffe [Deb01].

Behavior-based Intrusion-Detection bzw. **Anomaly-Detection** ist ein Ansatz, bei dem das Normalverhalten von Systemen und Benutzern über eine gewisse Zeitspanne erlernt wird. Erkennt das IDS ein Verhalten, das nicht antrainiert wurde, meldet es einen Alarm. Die behavior-based IDS besitzen eine hohe Rate von

falschen Alarmmeldungen. Neben nicht erlerntem Verhalten kann auch unerlaubtes Verhalten, welches während der Lernphase antrainiert wurde, das System ineffektiv werden lassen. Der Vorteil bei dieser Strategie ist die Erkennung auch neuer unbekannter Angriffe [Deb02].

Neben den IDS gibt es die **Intrusion-Prevention-Systeme (IPS)**. Sie sollen Angriffe erkennen und wirksam verhindern. Mit den Mitteln, die IDS bieten, können Intrusions zwar erkannt, aber nicht verhindert werden. Um diese Angriffe auch verhindern zu können, müssen die Systeme aktiv werden. Dazu werden im Falle von rechnerbasierten IPS zum Beispiel die Rechte des Benutzers verändert, sobald ein ungewöhnliches Ereignis auftritt. Netzwerkbasierte IPS müssen hingegen die Weiterleitung von gefährlichen Paketen sofort stoppen. Der Vorteil gegenüber einem IDS liegt natürlich in der direkten Abwehr von Angriffen. Somit entfällt die teure Arbeit, den Rechner wieder in Stand zu setzen, wenn der Angriff nicht abgewehrt wird. Ein Nachteil von IPS sind die Reaktionen auf falsch-positiv-Alarmmeldungen. Wenn diese auftreten, werden erwünschte Vorgänge unterbrochen. Sollten sich Falschmeldungen bei unternehmenskritischen Prozessen ereignen, können dadurch unnötige Kosten entstehen. Aus diesem Grund wird typischerweise neben einem IPS, das nur gezielte Angriffe verhindert, auch ein IDS eingesetzt, das alle restlichen Angriffe erkennen soll [Spe01].

2.2.1 Aufbau eines IDS

Der gewöhnliche Aufbau eines IDS wird in den Common-Intrusion-Detection-Frameworks (CIDF) [STS01] beschrieben.

Ein IDS besteht darin aus vier Teilen:

- Event generators ("E-boxes")
- Event analysers ("A-boxes")
- Event databases ("D-boxes")
- Response units ("R-boxes")

In der Abbildung 2 wird das Zusammenspiel der verschiedenen Komponenten verdeutlicht.

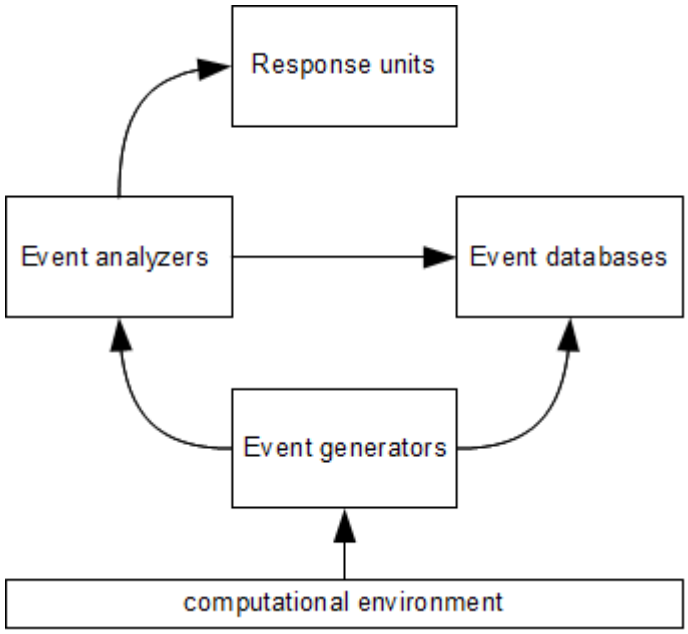

Abbildung 2: IDS-Aufbau des CIDF

Die E-boxes sammeln kontinuierlich Informationen bzw. Ereignisse von Betriebssystemen, Log-Dateien, Netzwerken und anderen Informationsquellen. Die gewonnenen Informationen müssen anschließend durch die E-boxes in ein Datenfor-

mat konvertiert werden, welches das IDS verarbeiten kann. Zur Protokollierung werden die Ereignisse an die D-boxes gesendet. Die D-Boxes verwalten die Speicherung der Ereignisse.

Die A-boxes analysieren anhand verschiedenster Technologien die Ereignisse auf Angriffsmuster und Anomalien. Werden solche erkannt, senden die A-boxes Alarmereignisse an die R-boxes. Zusätzlich werden die Analyseergebnisse durch die D-boxes gespeichert.

Die R-boxes haben im Fall von IDS die Aufgabe, die Benutzer des Systems zu informieren. Im Fall eines IPS leiten die R-boxes die Maßnahmen gegen den Angriff ein.

2.2.2 Technologien zur Paketanalyse in NIDS

Zur Analyse der Netzwerkpakete werden, wie schon erwähnt, verschiedenste Technologien eingesetzt. Ralf Spenneberg stellt in [Spe01] die Wichtigsten vor. Eine Auswahl der NIDS-Technologien sind:

- Signatur-Erkennung
- Zustandsorientierte Signatur-Erkennung
- Statistische Anomalie-Analysen
- Heuristische Analysen

HIDS-Technologien werden hier nicht weiter betrachtet.

Die **Signatur-Erkennung** erkennt Netzwerkangriffe, indem sie Pakete aus dem Netzwerkverkehr mit zuvor gespeicherten Signaturen oder Fingerabdrücken überprüft. Das IDS besitzt hierzu eine Datenbank, um sämtliche Angriffsszenarien zu speichern (vgl. knowledge-based Intrusion-Detection). Erkennt das IDS eine Paketsignatur, meldet es einen Alarm.

Die **Zustandsorientierte Signatur-Erkennung** erweitert die einfache Signatur-Erkennung um Zustände für aufgebaute Verbindungen. Das System überwacht nur Pakete, die in einer aufgebauten Verbindung übertragen werden. Alle anderen Pakete werden vom NIDS verworfen. Die Verbindungen müssen dazu in eine Zustandstabelle eingetragen werden. Dieser Vorgang hat den Vorteil, dass IDS nicht mehr so anfällig gegen Überlastungsangriffe sind, da zunächst eine reguläre Verbindung mit dem Zielrechner aufgebaut werden muss, bevor weitere Pakete versendet werden können. Ohne diesen Mechanismus kann der Angreifer Unmengen von Paketen ohne Bedeutung generieren, die das IDS überfordern würden.

Mit der **statistischen Anomalie-Analyse** werden Statistiken über die transferierten Pakete erstellt. Sollten in den Statistiken ungewöhnliche Werte auftreten, wird das IDS eine Alarmmeldung versenden. Das mögliche Auffinden noch unbekannter Angriffe ist bei dieser Variante der größte Vorteil (vgl. behavior-based Intrusion-Detection).

Heuristische Analysen stellen eine Verbesserung der Signaturerkennung dar. Eins-zu-Eins Vergleiche von Paketen mit Signaturen können sehr langsam sein. Durch heuristische Ansätze kann die Verarbeitungsgeschwindigkeit der Analyse von Paketen beschleunigt werden, ohne die Genauigkeit und Trefferhäufigkeit zu sehr zu beeinträchtigen.

2.3 Bisheriger Einsatz von CEP in IDS

Anhand der Definition von Complex-Event-Processing lässt sich erahnen, dass CEP als gute Grundlage für ein Intrusion-Detection-System dienen könnte. Bei der Recherche von IDS ist jedoch aufgefallen, dass Hersteller den Begriff CEP im Zusammenhang mit ihrer Detektionskomponente nicht verwenden. Zum derzeitigen Stand sind dem Autor keine kommerziellen Produkte bekannt, die eine extra für den CEP-Bereich entwickelte Detektionskomponente (CEP-Engine) einsetzen.

Die Recherche ergab dennoch, dass bestehende IDS besonders im Bereich der knowledge-based Intrusion-Detection Expertensysteme einsetzen (vgl. [Bas01]). Klassische Rule Engines sind eine Form zur Realisierung solcher Expertensysteme. Auch wenn viele IDS mittlerweile zeitliche Abfolgen der Pakete erkennen und sogar Paketinformationen aggregieren können, sind diese augenscheinlich nicht aus dem CEP-Ansatz entwickelt worden.

David Luckham konstruiert bereits in seinem Buch [Luc02] ein kleines Szenario, wie CEP innerhalb eines IDS eingesetzt werden kann. Er beschreibt anhand einer Fallstudie über EPNs für Netzwerkbeobachtungen den rudimentären Aufbau und ein paar typische Regeln des CEP-Systems. Für den Test wurde eine experimentelle CEP-Software erstellt, die ein Universitätsnetzwerk überwacht. Die CEP-Ereignisse beinhalten dabei die typischen Informationen eines Netzwerkpaketes. Eine der vorgestellten Regeln beschreibt ein Ereignismuster, wie ein Portscan erkannt werden kann. Das System überwacht, zusätzlich zur Mustererkennung, die Anzahl des Netzwerkverkehrs für verschiedene Protokolle. Die schematische Struktur sieht ein identisches EPN für jedes Netzwerksegment vor. Anschließend werden die Ergebnisse aus den einzelnen EPNs über weitere EPAs zusammengeführt, um eine Gesamtsicht zu erhalten. Wie genau die CEP-Regeln für die Intrusion-Detection-Software aussehen, wird jedoch (bis auf die beiden gezeigten Regeln) nicht vorgestellt. Eine bessere Übersicht über den Aufbau einer Intrusion-Detection-Software von der Stanford Universität ist in [Per01] einsehbar. Viele andere Publikationen wie zum Beispiel [Eck01] und [Owe01] gehen wie Luckhams auf die grundsätzliche Verwendungsmöglichkeit in IDS ein, beschreiben aber keine genaueren Szenarien, wie man IDS mit CEP umsetzt.

Ein IDS, welches mit einer CEP-Engine entwickelt wurde, ist das Forschungsprojekt „Inter-Domain Stealthy Port Scan Detection through Complex Event Processing“ [ALB01]. In diesem Projekt wird vorgestellt, wie CEP-Techniken dazu führen können, die Fehlerraten von IDS zu senken. Die Tests werden mittels des

Transmission Control Protocol (TCP)-SYN-Portscan (siehe Abschnitt 3.1.2) durchgeführt. Der Netzwerkverkehr liegt als Mitschnitt von echten Netzwerken vor. Die Portscans wurden in den Mitschnitt eingefügt. Die Algorithmen, die zur Portscan-Erkennung Anwendung finden, werden ebenfalls vorgestellt. Neben der Analyse der Paketsequenzen kommen statistische Berechnungen darin vor. Ein weiteres Merkmal, das Beachtung findet, sind horizontale und vertikale Angriffsszenarien. Bei horizontalen Portscans werden viele Rechner auf einem Port und bei vertikalen wird ein Rechner auf vielen Ports gescannt. Die Architektur der Software sieht eine zentrale CEP-Engine vor, die durch In-Adapter (genannt Gateway) mit Ereignissen versorgt werden. Jedes Netzwerksegment besitzt einen eigenen In-Adapter. Die gesammelten Ereignisse werden in der CEP-Engine analysiert und bei bestimmten Mustern in neue Ereignisströme eingeteilt. Diese Ströme zeigen aufgebaute und abgebrochene Verbindungen an. Wird ein definierter Grenzwert für abgebrochene Verbindungen überschritten, generiert das System einen Alarm.

Ein weiteres Forschungsprojekt, welches im Zusammenhang mit Intrusion-Detection und CEP entwickelt wurde, ist auf der Internetseite [TG01] beschrieben. Bei diesem Projekt wird anhand eines IDS-Szenarios die Verbesserung von CEP-Regeln durch die Verwendung des „Prediction-Correction Paradigm“ erläutert. Durch diesen Algorithmus können sich Regeln selbst anpassen, um bessere Ergebnisse zu erzielen. Einige weitere Arbeiten, wie beispielsweise [Far01], benutzen ebenfalls Intrusion-Detection als Testgrundlage für die Verbesserung von Detektionsalgorithmen. Der CEP-Systemaufbau, in Bezug auf IDS, findet aber selten größere Beachtung.

3 Entwicklung des CEP gestützten IDS

In diesem Kapitel wird der Entwurf und die Realisierung der Software zur Verifizierung der Einsatzmöglichkeit von CEP in IDS vorgestellt. Es werden die ausgewählten Anwendungsfälle beschrieben, mit denen die Implementierung getestet wurde. Die Entwicklung der Software erfolgte in zwei Iterationen. In der ersten Phase wurde die technische Infrastruktur realisiert. Die zweite Phase beinhaltet die Entwicklung der Anwendungsfälle. Eine detaillierte Beschreibung der Anwendungsfälle wird in Kapitel 3.1 gegeben. Der Entwurf der Architektur und des Ereignismodells wird in Kapitel 3.2 näher erläutert. Die Implementierung der Software sowie der Regeln für das CEP-System wird in dem Kapitel 3.3 betrachtet.

3.1 Ausgewählter Anwendungsbereich

Um die Eignung eines CEP-Systems für ein IDS testen zu können, musste zunächst ein geeigneter fachlicher Anwendungsfall bestimmt werden. Hierzu wurden Angriffsszenarien auf Computersysteme und Netzwerke betrachtet, die IDS aufspüren und melden sollen. Bei der Recherche dieser Angriffe wurde deutlich, dass eine Vielzahl dieser Angriffe erst durch einen Portscan des anzugreifenden Computersystems bzw. des anzugreifenden Netzwerks möglich werden. Ein Portscan ermittelt die offenen Ports, also die Dienste, die ein Rechner zur Verfügung stellt. Dienste, die Sicherheitslücken beinhalten, bieten Angreifern daher den Eingangspunkt, um die Systeme zu manipulieren. Aus der Sicht des IDS besteht aus diesem Grund eine hohe Relevanz, einen Portscan zu erkennen.

Die Ausschöpfung der Möglichkeiten, die ein CEP-System bietet, ist ein weiterer wichtiger Punkt bei der Suche eines geeigneten Anwendungsfalls. Portscans decken auch diesen Punkt gut ab. Dies zeigen Vorüberlegungen wie zum Beispiel, dass der Datenverkehr, der bei einem Portscan erzeugt wird, aus dem gesamten Datenstrom mit Hilfe von Filterregeln extrahiert werden muss. Außerdem können Paketsequenzen zu komplexeren Ereignistypen zusammengefasst werden. Paket-Statistiken können weitere wertvolle Informationen zur Detektion von Portscans liefern. Hierzu finden Aggregatfunktionen über Zeitfenster Anwendung. Ereignisse können auch mit zusätzlichen Informationen angereichert werden, um zum Beispiel vorherige Angriffsversuche mit in die Entscheidung der Reaktion auf diesen Portscan einfließen zu lassen.

Es gibt verschiedene Möglichkeiten, einen Portscan durchzuführen. Für diese Arbeit wurden ein paar der gängigsten Portscans ausgewählt. Zu diesen Portscans zählen der „TCP-Connect-Scan“, der „TCP-SYN-Scan“ und der „TCP-FIN-Scan“. Diese Portscans werden in den folgenden Kapiteln näher beschrieben. Die Informationen zu den Portscans wurden von der Internetseite [Lyo01] für das kostenlose Portscanner-Werkzeug „nmap“ entnommen. Zusätzlich zu den vorgestellten Portscans bietet die Seite auch eine gute Erklärung zu weiteren Portscan-Verfahren.

3.1.1 Der TCP-Connect-Scan

Der TCP-Connect-Scan ist eine Variante, bei der ein TCP-Verbindungsaufbau durchgeführt wird. Sofern ein geöffneter Port zur Verfügung steht, wird zusätzlich ein Verbindungsabbau durchgeführt. Der Verbindungsaufbau erfolgt mittels des TCP-Drei-Wege-Handschlags, der in der Abbildung 3 gezeigt ist. Zunächst schickt der Angreifer ein TCP-Paket mit der Internet Protocol (IP)-Adresse des Opfers und einer beliebigen Portnummer an den Zielrechner. Dieses Paket hat ein gesetztes SYN-Flag, welches anzeigt, dass eine Verbindung initiiert werden soll.

Auf dieses Paket antwortet der Zielrechner entweder mit einem Paket mit gesetztem RST-Flag oder SYN/ACK-Flag. Das RST-Flag gibt an, dass die Verbindung abgebrochen wird und hat in dieser Phase die Bedeutung, dass der Port nicht erreichbar/offen ist. Wenn ein Paket mit SYN/ACK-Flag zurückkommt, ist dies die Bestätigung dafür, dass der angegriffene Rechner an diesem Port Verbindungen entgegen nimmt. Wenn der Port offen ist, schickt der Angreifer ein weiteres Paket mit gesetztem ACK-Flag an das Opfer, um die Verbindung vollständig zu etablieren. Das ACK-Flag wird verwendet, um den Empfang von TCP-Paketen/Segmenten zu bestätigen.

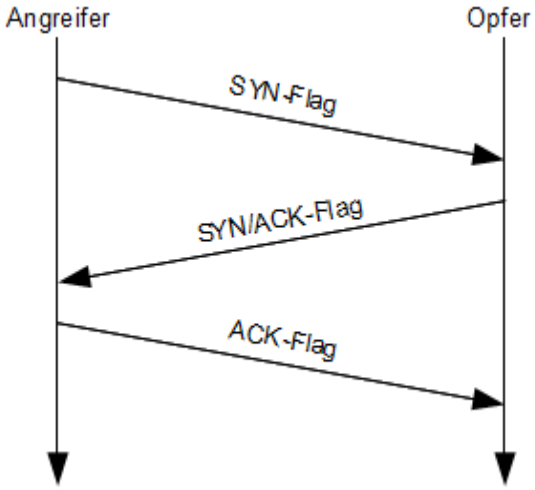

Abbildung 3: TCP-Connect-Scan Verbindungsaufbau

Nachdem der Verbindungsaufbau erfolgt ist, wird im direkten Anschluss die Verbindung wieder abgebaut. Dies geschieht wie in Abbildung 4 mit vier TCP-Paketen. Das erste Paket leitet den Verbindungsabbau aus Sicht des Angreifers mit einem gesetzten FIN-Flag ein. Mit dem dritten Paket wird die Verbindung aus Sicht des angegriffenen Rechners abgebaut. Die Pakete zwei und vier, mit gesetztem ACK-Flag, sind die Bestätigungen auf den Empfang des jeweiligen FIN-Pakets.

Dieser Vorgang entspricht dem normalen Ende einer TCP-Verbindung. Weiterführende Informationen zum TCP-Protokoll können im [RFC793] nachgeschlagen werden.

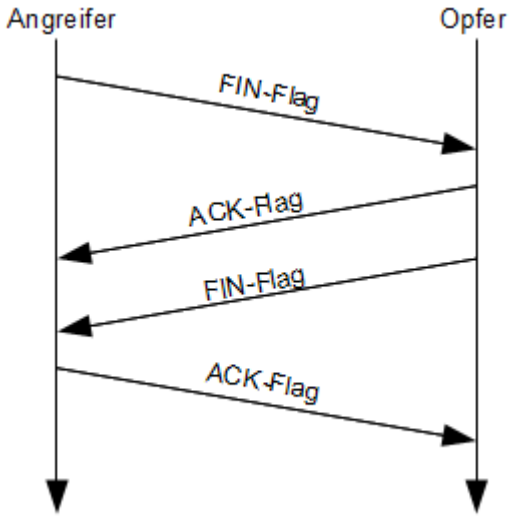

Abbildung 4: TCP-Connect-Scan Verbindungsabbau

Da diese Scan-Methode einen kompletten Verbindungsaufbau realisiert, können solche Scans, anders wie beim TCP-SYN-Scan, auf dem angegriffenen Rechner erkannt werden. Dies ist möglich, da die Verbindungen in den meisten Fällen von den Diensten in eine Log-Datei geschrieben werden. Ein anderer Weg, diesen Scan-Typ zu erfassen, ist die Analyse des Netzwerkverkehrs. Das hat den Vorteil, dass eine zentrale Komponente vor Portscans warnen kann. Ein Punkt, den diese Scan-Methode mit sich bringt, ist eine deutlich längere Laufzeit. Diese entsteht in erster Linie dadurch, dass die Verbindung wieder RFC-konform abgebaut wird. Der TCP-Connect-Scan kann von jedem System aus ausgeführt werden, da er die betriebssystemeigenen Methoden aufruft, um Verbindungen mit anderen Rechnern aufzubauen.

3.1.2 Der TCP-SYN-Scan

Beim TCP-SYN-Scan wird genau wie beim TCP-Connect-Scan versucht eine Verbindung zum Zielrechner aufzubauen. Dadurch unterscheiden sich die ersten beiden Pakete, die zwischen den beteiligten Rechnern versendet werden, nicht von den Paketen des TCP-Connect-Scan. Bei dieser Methode wird ebenfalls vom zweiten Paket angezeigt, ob der Port zur Verfügung steht oder geschlossen ist. Sollte der Port geöffnet sein sieht, der Drei-Wege-Handschlag die Bestätigung des Verbindungsaufbaus mit einem ACK-Paket vor. Anstatt dieses ACK-Paketes wird jedoch ein RST-Paket an den Zielrechner versandt, welches anzeigt, dass diese noch nicht vollständig initiierte Verbindung wieder abgebrochen wird.

Der TCP-SYN-Scan ist durch die geringe Anzahl versendeter Pakete sehr schnell. Da keine vollständige Verbindung zustande kommt, können die Dienste selbständig keine Protokolle erstellen. Aus diesem Grund versuchen IDS über die Analyse des Netzwerkverkehrs solche Portscan-Angriffe zu erkennen. Dieser Portscan kann nicht wie der TCP-Connect-Scan durch einen Systemaufruf erzeugt werden und muss daher durch eine eigens dafür entwickelte Software erfolgen.

3.1.3 Der TCP-FIN-Scan

Der TCP-FIN-Scan baut nicht wie die anderen beiden vorgestellten Scan-Methoden eine Verbindung auf, sondern sendet Pakete mit bestimmten Flags an den Zielrechner. Die Antworten auf diese Pakete geben Aufschluss über geöffnete und geschlossene Ports. Bei diesem Portscan-Verfahren wird auf die korrekte Umsetzung des [RFC793] gesetzt. Dieser RFC besagt, dass auf Pakete, die ein FIN-Flag beinhalten und nicht einer existierenden Verbindung zugehörig sind, der Zielrechner entweder mit einem RST-Paket antwortet oder gar keine Antwort versendet. Wird ein RST-Paket zurückgesendet, weiß der Angreifer, dass dieser Port geschlossen ist. Bekommt der Angreifer keine Antwort, ist der Port mit hoher Wahrscheinlichkeit geöffnet.

Andere Scan-Arten wie Null-Scan oder Xmas-Scan nutzen die selben Schwachstellen des [RFC793], weshalb nur der FIN-Scan stellvertretend für alle anderen Scan-Verfahren in der Testumgebung implementiert wurde. Dieser Scan ist auch über den Netzwerkverkehr zu analysieren und bietet ähnliche Vor- und Nachteile wie der TCP-SYN-Scan.

3.2 Entwurf des Systems

In diesem Kapitel wird der Entwurf des Systems anhand von Klassen- und Deployment-Diagrammen beschrieben. Des Weiteren wird ein Konzept zur Analyse von Netzwerkströmen angeführt, um Portscan-Verfahren von normalem TCP-Datenverkehr zu unterscheiden.

3.2.1 Architekturelles Konzept

Um in einer Testimplementierung ein IDS nachbilden zu können, mussten zunächst die Komponenten identifiziert werden, die ein IDS benötigt. Da in dieser Beispielimplementierung nur Netzwerkverkehr analysiert wird, beziehen sich die folgenden Komponenten auf netzwerkbasierte IDS. Wie in [Spe01] und [STS01] beschrieben, bestehen IDS in der Regel aus mehreren Komponenten. Ein wichtiger Teil des IDS ist die Aufnahme von Datenpaketen. Hierzu muss eine Komponente die empfangenen Netzwerkpakete analysieren und in die vom System unterstützte Form umwandeln. Diese Aufgabe wird im CIDF (siehe Abschnitt 2.2.1) von den E-boxes übernommen. Wenn ein solches IDS von mehreren Netzwerkpunkten Informationen beziehen möchte, müssen diese Komponenten die gewonnen Daten an die zentrale A-box senden. Diese sucht in den Datenpaketströmen anhand von Regeln nach Anomalien und alarmiert über die R-box gegebenenfalls die Administratoren des Netzwerkes. Die Regeln werden entweder aus Dateien oder Datenbanken geladen. Viele IDS speichern gemeldete Angriffe in den Daten-

banken, um nachfolgende Auswertungen durch die Administratoren zu ermöglichen. Die Funktionen zum Speichern werden durch die vorgestellten D-boxes realisiert.

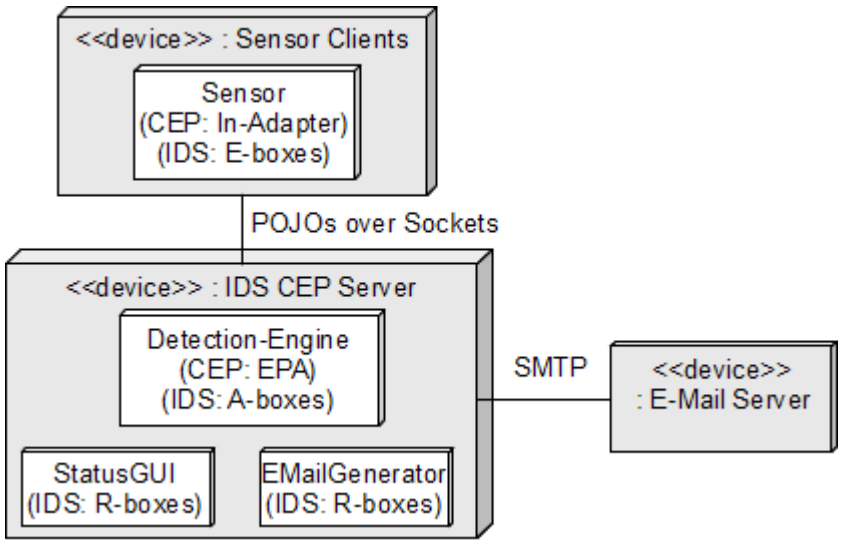

Abbildung 5: Deployment-Diagramm IDS

Der Entwurf für das Testsystem setzt fast alle dieser Komponenten um. Aus den beiden Aufbauten eines IDS und CEP-Systems (vgl. Abschnitt 2.1.1 und 2.2.1) ergibt sich für diesen Entwurf die in der Abbildung 5 enthaltene Struktur. Die Sensoren nehmen den simulierten Netzwerkverkehr auf und generieren daraus einfache Java-Objekte. Aus CEP-Sicht übernehmen die In-Adapter diese Aufgabe. Aus IDS-Sicht stellen die Sensoren die E-boxes dar. Die Plain-Old-Java-Objects (POJO) werden anschließend über eine Socket-Verbindung an die Serversoftware übertragen. Die IDS-CEP-Serversoftware nimmt die gesendeten Ereignisse entgegen. Sie besteht aus den A-boxes und den R-boxes. Die CEP-Engine in der A-box wendet die in Dateien definierten und hinterlegten Regeln auf den Ereignisstrom an. Wenn im Ereignisstrom ein Angriff identifiziert wurde, sendet das System eine E-Mail mit der Alarmmeldung per Simple-Mail-Transfer-Protocol (SMTP) an den

konfigurierten Empfänger. Zudem wird eine Alarmmeldung auf einer grafischen Benutzeroberfläche (GUI) angezeigt. Diese Funktionen stellen CEP-Schnittstellen bereit. Die E-Mail-Generierung und die Alarmierung im GUI stellen somit die R-boxes bzw. operativen Prozesse (siehe Abbildung 1) dar. Eine Datenbank zur Speicherung von Angriffsprotokollen oder Regeln ist in der Testumgebung nicht vorgesehen. Eine eigene D-box entfällt damit.

Den vollständigen Entwurf des Systems stellt das Klassendiagramm zusammen mit dem Ereignisdiagramm dar. In dem Klassendiagramm in Abbildung 6 werden die Bestandteile der Komponenten und deren Beziehungen verdeutlicht. Das Ereignisdiagramm in Abbildung 9 aus Kapitel 3.2.2 gibt einen Überblick über die Beziehungen zwischen den Ereignissen.

Erklärungen zum Klassendiagramm

Sensorkomponente (In-Adapter/E-boxes):

- Der EventBuilder wandelt die erzeugten TCP-IP-Pakete in Ereignisse um.
- Der EventSubmitter baut die Verbindung zur Detection-Engine auf und übermittelt die Events.

Detection-Engine-Komponente (EPA/A-boxes):

- Der ConnectionInitializer nimmt die Verbindungen von den Sensoren an und delegiert diese an den ConnectionHandler.
- Der ConnectionHandler nimmt die gesendeten Ereignisse entgegen und fügt diese über den EventEntrance in die CEP-Engine ein.
- Der EntranceAgent erzeugt und hält die Instanz der CEP-Engine. Außerdem lädt der EntranceAgent die Regeldateien.
- Der PortScanAlarmHandler bietet Methoden, die als Einstiegspunkte in das Programm dienen, um die operativen Prozesse E-Mailerzeugung und GUI-Alarmierung anzustoßen (vgl. CEP-Schnittstellen aus Kapitel 2.1.1). Die Regeln können diese Methoden aufrufen.

E-MailGenerator und Status-GUI (R-boxes)

- Die Klassen dieser beiden Komponenten präsentieren die gewonnenen Informationen über die Portscan-Analyse dem Benutzer entweder per E-Mail oder per GUI.

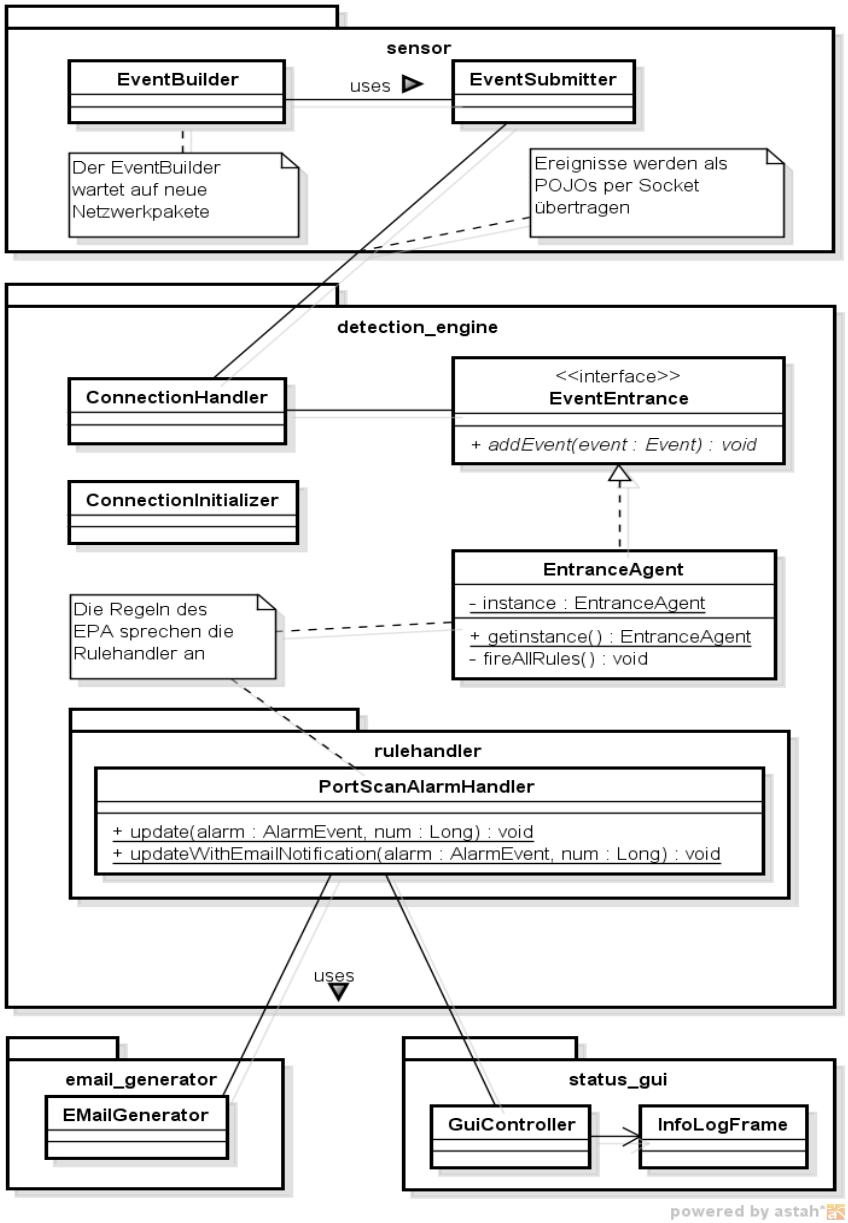

Abbildung 6: Klassendiagramm

3.2.2 Fachliches Konzept

Dieses fachliche Konzept stellt eine Anleitung dar, wie Portscan-Verfahren ermittelt werden können. Aufgrund der verschiedenen Portscan-Ansätze sind auch die Lösungsstrategien für jeden Portscan unterschiedlich. Die folgenden Seiten erläutern die verwendeten Strategien für die unterschiedlichen Scan-Arten.

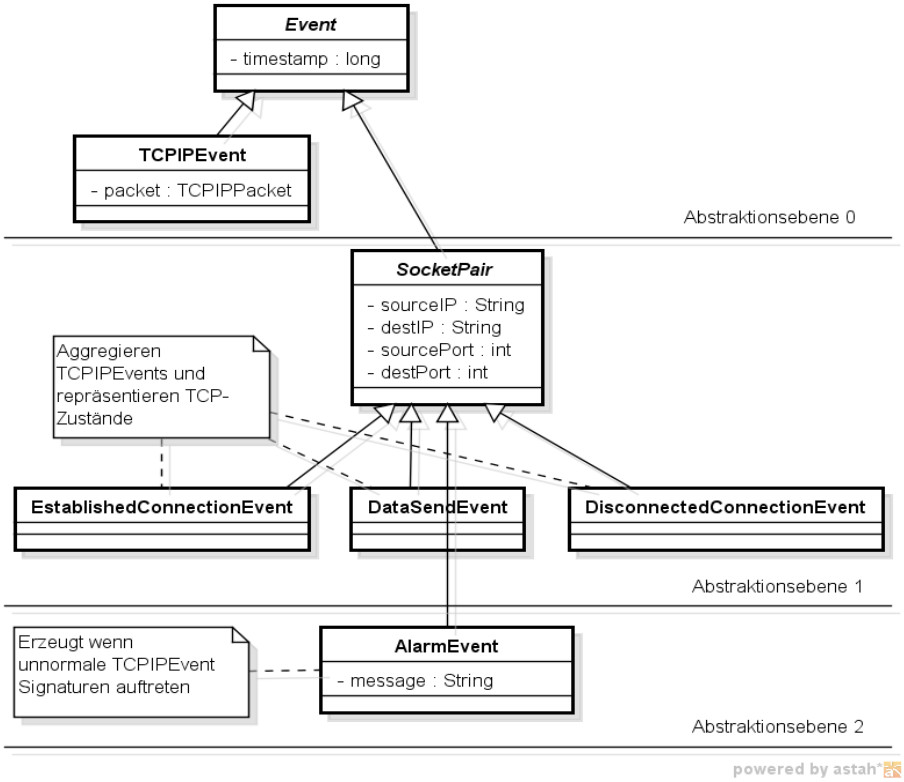

Abbildung 7: Ereignisdiagramm

Das Diagramm in Abbildung 7 zeigt den Aufbau der Ereignisstruktur. Die Abstraktion der Ereignisse wird von oben nach unten höher. Alle Ereignisse sind in dem package „events" hinterlegt und werden von der Klasse Event abgeleitet. Das Attribut timestamp wird benötigt, damit die CEP-Engine die Ereignisse in den zeitlichen Kontext einordnen kann. In dem TCP-IP-Event wird der aufgenomme-

ne Netzwerkverkehr gespeichert. Diese Ereignisse bilden die grundlegende Menge für die Anwendungsfälle. Die Ereignisse „Established Connection“, „Data Send“ und „Disconnected Connection“ bilden die Zustände der TCP-Verbindungen ab. Der nächste Absatz führt die Zustände ein. Diese Zustandsereignisse, sowie das Alarmereignis, erben die Attribute für die Verbindungsinformationen von der Klasse SocketPair. Alarmereignisse besitzen neben den Verbindungsinformationen ein Attribut, um eine individualisierte Alarmmeldung zu speichern.

Die verschiedenen Portscan-Varianten müssen zunächst von den erlaubten Verbindungen unterschieden werden. Um erwünschte Verbindungen zu identifizieren, werden die Verbindungen, wie in Abbildung 8 zu sehen, in drei Zustände eingeteilt.

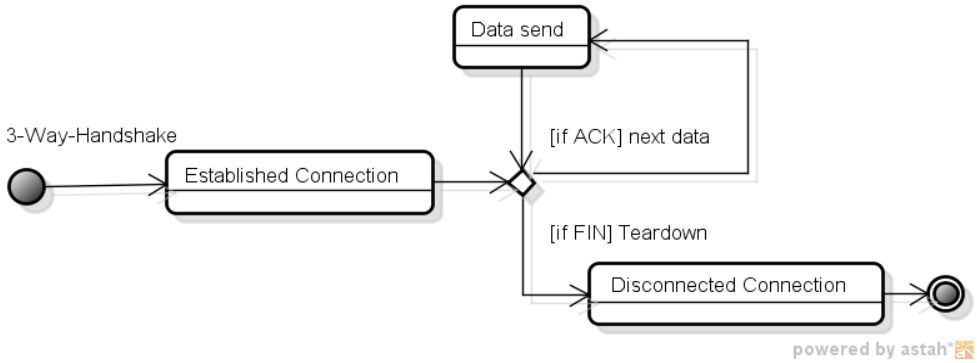

Abbildung 8: Zustandsdiagramm TCP-Verbindungseinteilung

Eine Verbindung, die erfolgreich aufgebaut wurde, geht in den Zustand „Established Connection“ über. Da bei normalen Verbindungen die beteiligten Systeme üblicherweise Daten zwischen einander austauschen, gibt es einen weiteren Zustand „Data Send“. Nachdem alle Daten versendet wurden, werden die Verbindungen ordnungsgemäß wieder abgebaut. Dieser Verbindungsabbau generiert ein weiteres Zustandsereignis „Disconnected Connection“. Die Zustände werden anhand der jeweiligen Signatur von TCP-Verbindungsaufbau, -abbau und Datenaustausch

ausgemacht. Wurde jedes dieser Zustandsereignisse generiert, kann mit sehr hoher Wahrscheinlichkeit von einer erwünschten Verbindung ausgegangen werden. In diesem Entwurf werden Verbindungen, die auf geschlossenen Ports aufgebaut werden sollen, grundsätzlich als gefährlich erachtet. Ausgangspunkt dieser Sichtweise ist die Annahme, dass die Anwender und Dienste in nichtöffentlichen Netzwerken wissen, über welche Verbindungen sie an die gewünschten Dienste gelangen. Die Regeln wurden mit dem Designmuster des Semantik-Shifts umgesetzt. Folgend wird am Beispiel eines TCP-Verbindungsaufbaus eine der drei Regeln im Pseudocode vorgestellt:

```
1 wenn
2       ( TCPIPEvent( ver1 = Socketpaar, Flag == SYN )
3         und
4         TCPIPEvent( ver1 == Socketpaar, Flag == SYN_ACK )
5         und
6         TCPIPEvent( ver1 == Socketpaar, Flag == ACK)
7       ) innerhalb von x Minuten
8 dann
9       con = erzeuge ConnectionEstablishedEvent( ver1 )
10      füge con in EstablischedConnectionEventStream ein
11 ende
```

Der TCP-Connect-Scan unterscheidet sich zu normalen TCP-Verbindungen nur dadurch, dass dieser keine Daten zwischen den beteiligten Systemen hin und her schickt. Wenn eine Verbindung aufgebaut und sofort wieder abgebaut wird, ist es also sehr wahrscheinlich, dass es sich um einen Portscan handelt. Das System kann einen solchen Portscan über den fehlenden Zustand „Data Send“ und die vorhandenen Zustände „Established Connection“ und „Disconnected Connection“ erkennen und ein Alarmereignis erzeugen. Die Regel zum Erkennen von TCP-Connect-Scans sieht folgendermaßen aus:

```
1 wenn
2       DisconnectedConnectionEvent( ver1 = Socketpaar )
3       und
4       kein DataSendEvent( ver1 == Socketpaar )
5       und
6       ConnectionEstablishedEvent( ver1 == Socketpaar )
7 dann
8       alarm = erzeuge AlarmEvent( ver1 )
9       füge alarm in AlarmEventStream ein
10 ende
```

Der TCP-SYN-Scan baut keine vollständige Verbindung auf. Dementsprechend entfällt die Prüfung auf die drei Zustände. Dieser Portscan wird anhand der Signatur der eingehenden Pakete erkannt. Tritt im Ereignisstrom ein Ereignismuster mit den für SYN-Scans üblichen Paketen auf, kann die folgende Regel, die für diesen Scan-Typ verantwortlich ist, ein Alarmevent generieren.

```
1 wenn
2       ( TCPIPEvent( ver1 = Socketpaar, Flag == SYN )
3         und
4         ( TCPIPEvent( ver1 == Socketpaar, Flag == SYN_ACK )
5           und
6           TCPIPEvent( ver1 == Socketpaar, Flag == RST)
7         )
8         oder
9         TCPIPEvent( ver1 == Socketpaar, Flag == RST)
10      ) innerhalb von x Minuten
11 dann
12      alarm = erzeuge AlarmEvent( ver1 )
13      füge alarm in AlarmEventStream ein
14 ende
```

Der TCP-FIN-Scan versucht FIN-Pakete an einen Port zu senden, zu dem zuvor keine Verbindung hergestellt wurde. Sockets an denen ein FIN-Paket ankommt und zu denen kein „Established Connection“-Ereignis existiert, werden als gefährlich eingestuft. Um zwischen Portscan-Angriffen auf geöffnete und geschlossene Ports unterscheiden zu können, werden die folgenden Pakete betrachtet. Wird ein

RST-Paket vom Zielrechner zurückgesendet, ist der Port geschlossen. Wenn kein Paket innerhalb eines gewissen Zeitraums zurückgesendet wird, ist der Port geöffnet. Die Regel für diese Portscan-Art kann so beschrieben werden:

```
1 wenn
2       ( TCPIPEvent( ver1 = Socketpaar, Flag == FIN )
3         und
4         kein  ConnectionEstablishedEvent(ver1 == Socketpaar)
5         und
6           ( TCPIPEvent( ver1 == Socketpaar, Flag == RST )
7           oder
8           kein TCPIPEvent( ver1 == Socketpaar)
9           innerhalb von x Sekunden
10        )
11      )
12 dann
13      alarm = erzeuge AlarmEvent( ver1 )
14      füge alarm in AlarmEventStream ein
15 ende
```

Die von den Portscan-Regeln erzeugten Alarmereignisse werden in einen neuen Ereignisstrom eingefügt. Aggregierungsfunktionen in den nachgelagerten Alarmregeln ermitteln die Anzahl von Alarmereignissen pro Zielrechner. Wenn eine definierte Schwelle überschritten ist, informiert das System die Benutzer. Die Portscan-Alarmregeln führen dabei einen Granularitäts-Shift durch. Die gewonnenen Informationen werden anschließend an die CEP-Schnittstelle gesendet. Die Regel sieht schematisch so aus:

```
1 wenn
2       Anzahl von( AlarmEvent(mit gleicher ZielIP)) > 5
3       innerhalb von x Tagen
4 dann
5       sende letztes AlarmEvent an CEP-Schnitstelle
6       sende Anzahl an CEP-Schnittstelle
7 ende
```

Die Regeln geben zusammen mit dem Ereignisdiagramm eine Struktur für die Ereignisverarbeitung vor. Diese ist in der Abbildung 9 dargestellt.

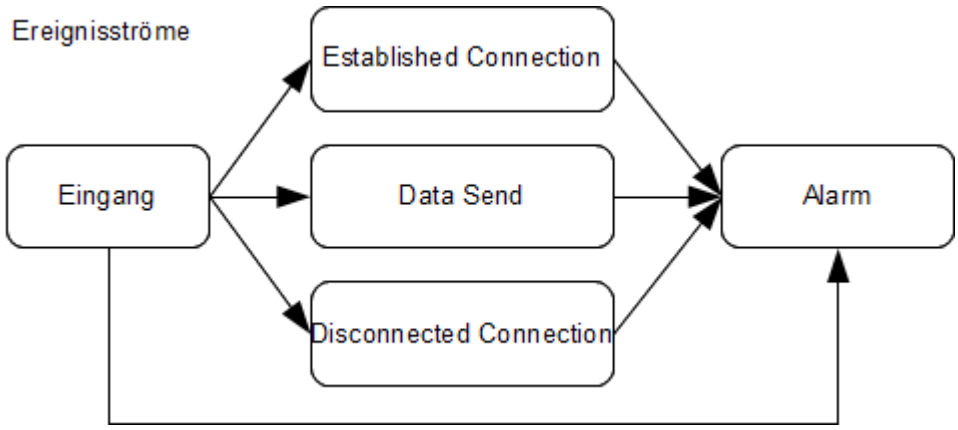

Abbildung 9: Struktur Ereignisströme

3.3 Implementierung

Ziel der Implementierung ist die Umsetzung und anschließende Testmöglichkeit des entwickelten Konzepts. Dabei sollen mögliche Hindernisse bei der Entwicklung von IDS mit der Hilfe von CEP-Systemen erkannt und dokumentiert werden. Dieses Kapitel teilt sich in drei Abschnitte auf. Im ersten Abschnitt wird die verwendete CEP-Engine vorgestellt. Im zweiten Abschnitt werden Implementierungsdetails und Regeln besprochen. Der dritte Abschnitt beschäftigt sich mit den Problemen und Schwierigkeiten, die bei der Entwicklung auftraten.

3.3.1 Drools Fusion

Die zur Verarbeitung der Ereignisse eingesetzte Complex-Event-Processing-Engine ist Drools Fusion. Drools Fusion bietet eine Weiterentwicklung der Open-Source Rule-Engine Drools Expert von der JBoss Community der RedHat incorporated. Unter den beiden Quellen [JBo01] und [JBo02] können die Produkte und dazugehörigen Dokumentationen heruntergeladen werden.

Ein besonderes Merkmal von Drools Fusion ist die Integrationsweise in eine Rule-Engine. Bei der Integration wurde darauf geachtet, dass beide Konzepte, die von klassischen Rule-Engines und CEP-Engines, als gleichwertig angesehen werden. Drools Fusion ist zwar ein eigenständiges Modul, bietet jedoch den vollen Umfang der gesamten Drools-Plattform.

Ein weiteres Merkmal ist die allgemeine Regelsprache die Drools bietet. Die Regeln können mit diesem Regelsprachentyp in dem Stil von:

wenn

Ereignis A auftritt,

dann

führe Aktion B aus;

definiert werden.

Dieser Stil ermöglicht eine klarere und strukturiertere Definition der Regeln, als der von Continuous Query Languages (CQL). Die Event-Processing-Engine Esper setzt beispielsweise eine Ausprägung der CQL ein. Eine genauere Beschreibung der beiden Sprachtypen findet sich in [BD01]. Information zu der Open-Source Event-Processing-Engine Esper sind unter [Esp01] erhältlich.

Drools erweitert den bekannten RETE-Algorithmus um objektorientierte Aspekte. Eine Beschreibung des Algorithmus ist unter [Doo01] erhältlich. Der Algorithmus ist zur effizienten Erkennung von definierten Mustern in Datenmengen entwickelt worden. Die erste Version des Algorithmus wurde 1979 von Dr. Charles Forgy vorgestellt. Ein Auszug des Originaltextes ist unter [For01] einsehbar. Das JBoss Cummunity Team hat mit Drools Fusion zusätzlich zeitliche Operatoren implementiert, um Event Processing besser zu unterstützen.

Die Engine von Drools Expert und Fusion wird vereinfacht in Abbildung 10 dargestellt.

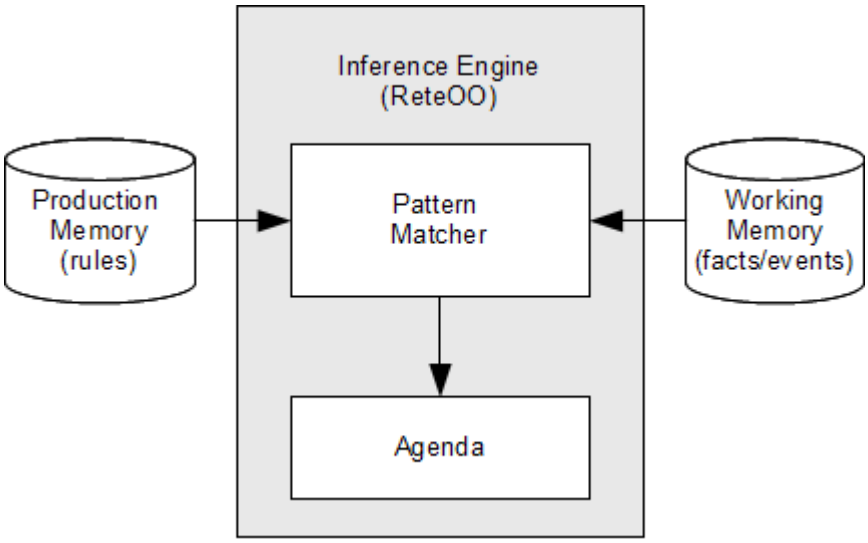

Abbildung 10: Schematischer Aufbau der Drools Rule-Engine basierend auf [JBo05]

Die Regeln werden in dem Production-Memory gespeichert. Die Fakten und Ereignisse sind im Working-Memory gespeichert. Die Inference-Engine vergleicht die aufgetretenen Fakten bzw. Ereignisse mit den in den Regeln definierten Ereignismustern. Wenn mehrere Regeln auf ein Muster hin aktiviert werden, entscheidet die Agenda anhand einer Konfliktlösungsstrategie, welche der Regeln als erstes ausgeführt wird [JBo05]. Aus der Sicht von Drools sind Ereignisse spezielle Typen von Fakten [JBo03].

3.3.2 Implementierungsdetails

Der erste Schritt der Implementierung beinhaltet die Entwicklung der In-Adapter, die in erster Linie die Sensoren darstellen. Der Netzwerkverkehr wird ausschließlich simuliert, wodurch dieser in Java-Objekten vorliegt. Aus diesem Grund entfallen die Syntaxanalysierer (Parser), die den Netzwerkverkehr in Java-Objekte

umwandeln würden. Der Sensor setzt dadurch nur die bereits in Kapitel 3.2.1 beschriebenen zwei Funktionen um. Zum einen horcht er auf neue TCP-IP-Pakete, die in eine Containerklasse eingetragen werden. Dieser Mechanismus funktioniert über das Publisher-/Subscriber-Verfahren. Zum anderen erzeugt er Ereignisse aus den TCP-IP-Paketen und leitet diese über einen Socket an die Serverkomponente weiter.

Warum Sockets zur Datenübertragung?

IDS und besonders IPS müssen möglichst reaktionsschnell arbeiten, damit man auf Angriffe reagieren kann, noch bevor es zu Schäden kommt. Remote Methode Invocation oder Java Messaging Service erzeugen im Vergleich zu Sockets zu viele Zusatzdaten und haben eine viel höhere Laufzeit. Damit der Zeitverlust bei der Datenübertragung so gering wie möglich ist, wurden Sockets verwendet.

Der zweite Schritt stellt die Realisierung der Serverkomponente dar. Die Annahme der Sensorverbindungsanfragen erfolgt serverseitig über das Server-Handler-Muster. Dies ermöglicht den gleichzeitigen Einsatz mehrerer Sensoren an einer Server-Komponente. Die Drools Fusion Event-Processing-Engine wird in der Klasse EntranceAgent instantiiert. Dies geschieht mit den folgenden Anweisungen.

```
1 KnowledgeBaseConfiguration config =
2    KnowledgeBaseFactory.newKnowledgeBaseConfiguration();
3 config.setOption( EventProcessingOption.STREAM );
4 final KnowledgeBuilder kbuilder =
5    KnowledgeBuilderFactory.newKnowledgeBuilder();
6 kbuilder.add( ResourceFactory.newFileResource( file1 ),
7    ResourceType.DRL );
8 final KnowledgeBase kbase =
9    KnowledgeBaseFactory.newKnowledgeBase(config);
10 kbase.addKnowledgePackages( kbuilder.getKnowledgePackages());
11 ksession = kbase.newStatefulKnowledgeSession();
```

In den ersten beiden Zeilen des Quellcodeblocks wird die Konfiguration für die Wissensbasis instantiiert. Der Parameter EventprocessingOption.STREAM gibt an, dass die Rule-Engine im Event-Processing-Modus startet. Ohne diese Angabe startet die CEP-Engine im Cloud-Modus, der für den Betrieb als klassische Rule-Engine vorgegeben ist. Der Befehl in Zeile vier und fünf erzeugt einen KnowledgeBuilder. Dieser wird benötigt, um ein Regelpaket aus einer Ressource zu generieren. In diesem Fall werden die Regeln, wie in den Zeilen sechs und sieben zu sehen, aus einer Datei geladen. Über diesen Weg können beliebig viele Dateien eingebunden werden. Im nächsten Schritt erzeugt die KnowledgeBaseFactory anhand der in der Konfiguration angegebenen Einstellungen eine leere KnowledgeBase. Diese wird mit den Regelpaketen aus den Dateien in der Zeile zehn befüllt. Zum Schluss muss eine StatefulKnowledgeSession erstellt werden, die im weiteren Programmverlauf genutzt wird, um Ereignisse einzufügen.

Der Programmcode zum Einfügen von Ereignissen setzt sich folgendermaßen zusammen.

```
1 public synchronized void addEvent(Event event) {
2       ksession.insert( event );
3       ksession.fireAllRules();
4 }
```

Über das StatefulKnowledgeSession-Objekt ksession können die Ereignisse mit der Methode insert eingefügt werden. Die Methode fireAllRules startet den Prozess, der prüft, ob die Ereignisse auf die in den Regeln definierten Muster passen. Wenn eine Regel ein passendes Muster findet, löst sie ihren Aktionsteil aus. Dieser Vorgang wird als feuern bezeichnet. Die Methode addEvent wird synchronisiert, damit mehrere Sensoren über den selben Zugang Ereignisse in die CEP-Engine einfügen können.

Mehr als der vorgestellte Quellcode ist zum Betrieb der CEP-Engine nicht nötig. Um Drools Fusion zu nutzen, müssen folgende Bibliotheken eingebunden werden:

- antlr-runtime-3.3.jar
- drools-compiler-5.2.0.Final.jar
- drools-core-5.2.0.Final.jar
- drools-decisiontables-5.2.0.Final.jar
- drools-jsr94-5.2.0.Final.jar
- knowledge-api-5.2.0.Final.jar
- mvel2-2.1.0.drools2.jar

Für die Interpretation der Regeln wird ein EPA verwendet. Entry-Points unterteilen den EPA durch verschiedene Ereignisströme. Die Verwendung eines EPN aus logischer Sicht erschien dadurch nicht nötig. Außerdem ist die Verteilung von EPAs auf mehrere Rechner zur Lastverteilung nicht vorgesehen. In Abbildung 11 wird der Aufbau des EPAs gezeigt. Sie erweitert die Struktur aus Abbildung 9 in Kapitel 3.2.2. Die Pfeile stellen den Informationsfluss zwischen den Ereignisströmen da.

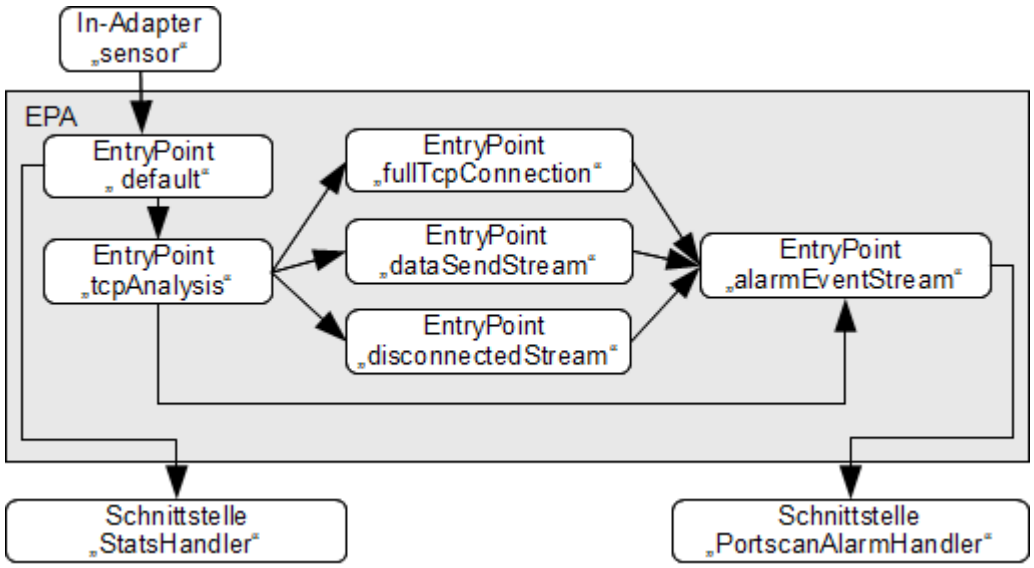

Abbildung 11: EPA-Aufteilung

Die erstellten Regeln wurden ebenfalls aufgeteilt. Daraus ergeben sich für den Anwendungsteil drei Dateien. Die Regeln für die gewünschten TCP-Verbindungen, die Portscan-Analyse und die Alarmmeldungen sind jeweils in eigenen Dateien zusammengefasst. Als Dialekt für die Regelsprache kommt MVEL zum Einsatz. MVEL ist eine Ausdruckssprache für Java basierte Anwendungen. Informationen zu MVEL können unter [Cod01] abgerufen werden.

Die Regelbasis für den Anwendungsfall umfasst elf Regeln. Im folgenden werden zwei ausgewählte Regeln vorgestellt, um den Aufbau und die Funktionsweise zu erläutern. Die restlichen Regeln stehen im Anhang. Damit die CEP-Engine von Drools Fusion die Ereignisse als Ereignisse und nicht als Fakten annimmt, müssen sie in den Regeldateien folgendermaßen gekennzeichnet werden.

```
1 declare AlarmEvent
2       @role( event )
3 end
```

Mit der Annotation @role(event) werden Objekte der Klasse AlarmEvent als Ereignisse definiert. Es können weitere Annotationen, wie zum Beispiel timestamp oder expires, benutzt werden. Timestamp gibt an, welches Attribut der Klasse den Zeitstempel speichert. Expires definiert den Zeitpunkt, wann ein Ereignis von der Event-Processing-Engine gelöscht werden darf.

Die folgende Regel namens „tcp connect on closed port“ veranschaulicht, wie ein Verbindungsversuch auf einen geschlossenen Port erkannt wird.

```
1  rule "tcp connect on closed port"
2    dialect "mvel"
3    when
4      $e1 : TCPIPEvent(packet.tcpHeader.controlFlag ==
5        CONTROL_FLAG.SYN) from entry-point "tcpAnalysis"
6      $e1tcpHeader : TCPHeader() from $e1.packet.tcpHeader
7      $e1ipHeader : IPHeader() from $e1.packet.ipHeader
8
9      $e2 : TCPIPEvent(packet.tcpHeader.controlFlag ==
10       CONTROL_FLAG.RST
11       && $e1ipHeader.sourceIP == packet.ipHeader.destIP
12       && $e1ipHeader.destIP == packet.ipHeader.sourceIP
13       && $e1tcpHeader.sourcePort == packet.tcpHeader.destPort
14       && $e1tcpHeader.destPort == packet.tcpHeader.sourcePort
15       ) from entry-point "tcpAnalysis"
16   then
17     retract($e1);
18     retract($e2);
19     AlarmEvent alarm = new AlarmEvent(
20       System.currentTimeMillis(), $e1ipHeader.sourceIP,
21       $e1ipHeader.destIP, $e1tcpHeader.sourcePort,
22       $e1tcpHeader.destPort,
23       "Portscan (SYN- or connect scan on closed Port)!")
24     drools.entryPoints["alarmEventStream"].insert( alarm );
25 end
```

Damit die Regel feuert, müssen drei Bedingungen erfüllt sein. Als Erstes muss ein Paket über das Netzwerk versendet worden sein, welches das SYN-Flag gesetzt hat. Zum Zweiten muss ein Folgepaket versendet worden sein, dass zu dieser Verbindungsanfrage passt. Drittens muss dieses Paket das RST-Flag gesetzt haben. Beide Ereignisse sind aus dem Entry-Point tcpAnalysis. In den Zeilen sechs und sieben werden Hilfsvariablen für IP- und TCP-Header angelegt, die für die Ver-

bindungsüberprüfung des Folgepakets in den Zeilen elf bis vierzehn benutzt werden. In dem Aktionsteil der Regel, der mit dem Schlüsselwort then beginnt, werden die beiden gefundenen Ereignisse aus dem tcpAnalysis EntryPoint entfernt. Die beiden Ereignisse werden gelöscht, damit sie beim erneuten Aufruf von fireAllRules im dreiminütigen Zeitfenster nicht noch einmal zum Feuern der Regel führen. Anschließend wird ein Alarmereignis instantiiert und in den EntryPoint alarmEventStream eingefügt. Die Alarmevents werden in der nächsten präsentierten Regel über den Zeitraum von zwei Tagen gezählt. Die Regel stößt mit jedem fünftem Alarmereignis die Erzeugung einer E-Mail an. Die Erzeugung der E-Mail ist über eine statische Methode in der Klasse PortscanAlarmHandler realisiert. Statische Methoden können direkt aus der Regel aufgerufen werden.

```
1 rule "get all events from alarmEventStream to send email"
2   dialect "mvel"
3   when
4     $maxTimestamp : Number()
5       from accumulate(
6         $ae1 : AlarmEvent() over window:time( 2d )
7         from entry-point "alarmEventStream" ,
8         max($ae1.timestamp))
9
10    $alarmEvent: AlarmEvent(
11      (Double)timestamp == (Double)$maxTimestamp )
12      from entry-point "alarmEventStream"
13
14    $num : Number(((intValue % 5) == 0), intValue > 0)
15      from accumulate($ae2 : AlarmEvent(
16        $alarmEvent.destIP == destIP)
17        over window:time( 2d )
18        from entry-point "alarmEventStream" ,
19        count($ae2))
20  then
21    PortscanAlarmHandler.updateWithEmailNotification(
22      $alarmEvent, $num);
23 end
```

Mit der Funktion accumulate können Ereignisse gezählt werden. Das Schlüsselwort over window:time legt fest, über welchen Zeitraum gezählt werden soll, in diesem Fall die angesprochenen zwei Tage. In der Variable $maxTimestamp wird

der Zeitstempel des jüngsten Alarmereignisses gespeichert, um dieses in den Zeilen zehn, elf und zwölf zu selektieren. Die Menge von Alarmereignissen an der gleichen Opfer-IP-Adresse wird in $num gespeichert. Die Regel feuert nur, wenn der Wert größer null und durch fünf teilbar ist.

Zusätzliche Regeln können über neue Regeldateien nachgeladen werden. Das GUI bietet dazu eine Funktion. Dieser Mechanismus funktioniert auch, während die Event-Processing-Engine arbeitet.

Die Konfiguration der Sensoren und der Serverkomponente erfolgt über Konfigurationsdateien, die in dem Ordner #Programm-Pfad#/config liegen müssen. Einstellungsmöglichkeiten sind die IP und der Port der IDS-Serverkomponente, E-Mailadressen für Sender und Empfänger der Alarm-E-Mails und die SMTP-Serveradresse, an die die E-Mail gesendet wird.

Die Klasse ServerStarter beinhaltet die Startmethode für die Serverkomponente. Der Sensor startet mit der Klasse SensorStarter. Die Serverkomponente muss vor der Sensorkomponente ausgeführt werden.

3.3.3 Probleme bei der Implementierung

Dieses Kapitel gibt einen Überblick über ein paar Hindernisse, die bei der Entwicklung auftraten. Falls weitere Projekte in diesem Bereich angedacht sind, bieten die folgenden Anhaltspunkte Erklärungen, wie die Hindernisse zu überwinden sind.

Bei der Entwicklung mit Eclipse, ohne das angebotene Drools Plugin, tritt ein Fehler auf, bei dem nach der org.eclipse.jdt.core_version.jar verlangt wird. Diese muss zusätzlich in das Projekt eingebunden werden.

Die Funktion fireUntilHalt, die Drools besonders für den Event Processing Einsatz eingeführt hat und ein reaktives Verhalten unterstützt [JBo04], führte in dieser Implementierung nach einiger Zeit zu Laufzeitfehlern. Die einfachste Möglich-

keit, diesen Fehler zu umgehen, ist fireAllRules nach jedem Einfügen eines Ereignisses aufzurufen. Wenn Regeln auch ohne neue Ereignisse feuern sollen, kann ein Thread geschrieben werden, der fireAllRules in dem nötigen Intervall aufruft. Eine weitere Lösungsmöglichkeit bietet die Timer-Funktion, die Drools Fusion für Regeln anbietet. Eine kontinuierliche Ausführung der Regeln ist zum Beispiel bei einer Statusabfrage, wie viele Ereignisse sich zur Zeit in der Event-Processing-Engine befinden, nötig.

Beim Nachladen von Regeln während der Laufzeit sind Abhängigkeiten zu beachten. Ereignisse müssen, bevor sie in die Event-Processing-Engine eingefügt werden, als Ereignisse deklariert werden. Wird dies nicht getan, sind die eingefügten Ereignisse nur Facts. Facts sind Objekte ohne einen Zeitstempel. Diese können nur von dem klassischen Rule-Engine-Anteil benutzt werden. Sobald eine Regel in die Event-Procesing-Engine eingefügt wird, die einen Ereignistypen deklariert, von dem aber schon ein Exemplar als Fact in den EPA eingefügt ist, kommt es zu Fehlern. Diese Fehler treten jedoch erst auf, sobald eine Regel den zeitlichen Kontext auf diesen Ereignissen, die aus Sicht der CEP-Engine immer noch Facts sind, ausnutzt.

Eine weitere Abhängigkeit besteht zwischen Entry-Points. Die Reihenfolge der Ereignis einfügenden und abnehmenden Regeln in und aus Entry-Points ist vorgegeben. Es dürfen keine Regeln definiert werden, die Ereignisse in einen Entry-Point einfügen, aus dem nicht gelesen wird. Beim Nachladen von Regeln kann diese Bedingung zu Fehlern führen. Wenn die einfügenden und konsumierenden Regeln in zwei verschiedenen Dateien geschrieben wurden, passiert es schnell, dass sie falsch herum geladen werden.

Regeln, bei denen Ereignisse nicht eintreten sollen damit sie feuern, erfordern eine besondere Überlegung. Die Event-Processing-Engine muss auf das Nichteintreffen eines in der Zukunft liegenden Ereignisses warten. Für diese Fälle muss der

zeitliche Operator after verwendet werden. Regeln, die diesen Operator verwenden, feuern erst nach der angegebenen Zeit. Dies ist ein großer Unterschied zu klassischen Rule-Engines. Deshalb wurde eine eigene Sektion in der Dokumentation [JBo03] von Drools Fusion dafür eingerichtet.

3.4 Test des implementierten IDS

Die Implementierung wurde durch die Erzeugung von simuliertem Netzwerkverkehr getestet. Dazu wurde ein Paketgenerator erstellt, der Java-Objekte erzeugt, welche den Netzwerkverkehr imitieren. Der Paketgenerator nimmt dabei die Stellung der initialen Ereignisquelle ein.

Testdaten

Die erzeugten TCP-IP-Pakete beinhalten nur Informationen, die für die Portscan-Analyse notwendig sind. Dazu gehören die IP-Adressen und Portnummern der Verbindung sowie die Flags der einzelnen Pakete. Der Generator produziert sowohl normalen Verbindungsverkehr mit Datenübertragung als auch die verschiedenen Portscan-Arten.

Testaufbau

Die normalen Verbindungen werden alle von der IP-Adresse 192.168.0.100 auf die IP-Adresse 192.168.0.200 aufgebaut. Der Port hat die Nummer 2000. Der Connect-Scan erfolgt von der IP-Adresse 192.168.0.101 auf einen offenen Port des Opferrechners. Die IP-Adresse und der Port sind beim SYN- und Connect-Scan identisch. Per Zufallsprinzip werden an dem Opferrechner beim SYN-Scan offene und geschlossene Ports simuliert. Da sich das Verhalten auf geschlossenen Ports bei Connect- und SYN-Scan nicht unterscheidet, ist dieser Fall stellvertretend für beide Scan-Arten nur einmal implementiert. Der FIN-Portscan wird auf

eine andere IP-Adresse angewendet, um die Generierung der Alarmmeldungen für unterschiedliche Opfer IP-Adressen zu testen. Auch bei dieser Scan-Methode wird per Zufall zwischen offenen und geschlossenen Ports unterschieden.

Die Generator-GUI stellt eine Funktion bereit, mit der auf Knopfdruck, kontinuierlich alle 10ms eine Verbindung aufgebaut, Daten versendet und die Verbindung wieder abgebaut wird. Zusätzlich können die verschiedenen Portscan-Arten jeweils über einzelne Schaltflächen angestoßen werden. So kann anhand der Anzahl der Starts von Portscans kontrolliert werden, ob die Serversoftware auch alle erkennt.

Die Implementierung wurde auf einem Rechner mit Intel Core i7 950 Prozessor @ 3.6GHz getestet. Die Größe des Arbeitsspeichers betrug 6 GByte. Windows 7 x64 stellte die Betriebssystemplattform für den Test dar. Die Java Virtual Machine (JVM) in der Version 1.6.0_22 wurde mit Standardeinstellungen betrieben.

Testdurchführung und Ergebnisse

Bei den Tests wurden fünf Sensoren mit den dazugehörigen Paketgeneratoren verwendet. Diese sieben Generatoren produzierten kontinuierlich insgesamt ungefähr 40000 Ereignisse normalen Netzwerkverkehrs innerhalb von 10 Sekunden. Die IDS-Server-Komponente belegte allein ca. 1,20 GB Arbeitsspeicher. Weitere Tests mit sechs und sieben Sensoren ergaben eine kurzfristige maximale Verarbeitungsmenge von 50000 Ereignissen, die jedoch stark schwankend war und dann bis auf 30000 Ereignisse abfiel. Die Prozessorauslastung betrug während der Tests mit 40000 Ereignissen 25%. Offensichtlich sind für die Schwankungen die verringerten Speicherreserven der JVM verantwortlich. Die vielen Garbage-Collection-Operationen scheinen, nach ersten Profiler-Analysen, Grund für die Schwelle von 40000 Ereignissen in 10 Sekunden zu sein.

Zunächst wurde der Test zum Erkennen von Portscans mit einem Sensor durchgeführt. Dabei erzeugte der Paketgenerator ca. 5500 Ereignisse vom normalen Pa-

ketverkehr innerhalb von 10 Sekunden. Während der Erzeugung wurden zusätzlich zehn Portscan-Angriffe in schneller Abfolge per Betätigung der Portscan-Schaltflächen durchgeführt. Die Anzahl setzt sich aus drei TCP-Connect-Scans, drei TCP-SYN-Scans und vier TCP-FIN-Scans zusammen. Auf der Serverseite wurde das Auftreten der verschiedenen Scan-Typen durch die Kontrolle der Log-Anzeige überprüft. Anschließend wurde der Test mit der maximalen Verarbeitungsmenge wiederholt. Alle Testversuche konnten die Portscan-Angriffe schnell und korrekt erkennen. Auch die E-Mailgenerierung erfolgte reibungslos und korrekt. Da der Opferrechner mit der IP 192.168.0.200 fünf mal gescannt wurde (genau genommen sechs mal), hat das System eine Alarm-E-Mail versendet.

Des Weiteren wurde das Nachladen von Regeldateien erprobt. Für diesen Test wurde ebenfalls ein Paketgenerator benutzt, der kontinuierlich Ereignisse gesendet hat. Nachdem alle Schwierigkeiten (siehe Abschnitt 3.3.3) erkannt waren, konnten Regeldateien während der Laufzeit der CEP-Engine hinzugefügt werden.

4 Bewertung

Die kritische Betrachtung der Software erfolgt in diesem Kapitel. Es ist unterteilt in die Bewertung der Software und den Vergleich zu echten IDS. Die Bewertung der Software bezieht sich auf die Umsetzung der Anwendungsfälle, den Aufbau des Systems sowie die Tests, die durchgeführt wurden. In dem Kapitel „Vergleich mit IDS“ werden allgemeine Punkte betrachtet, die echte IDS anders umsetzen. Dabei dient der Umfang der Testimplementierung als Grundlage dieses Vergleiches.

4.1 Bewertung der Software

Anwendungsfälle

Ein großer Kritikpunkt der Software ist die Herangehensweise an die Anwendungsfälle. Andere Möglichkeiten zur Erkennung von Portscans finden keine ausreichende Betrachtung. Statistische Analysen des Paketaufkommens geben evtl. bessere Schlüsse für die Portscan-Detektion. Eine weitere Überlegung ist, nur Pakete zu betrachten, die in einer initiierten Verbindung versendet werden. Der Rest der Pakete wird dabei als ungewöhnlicher Netzwerkverkehr angesehen. Wie gut dieser Ansatz in einer realen Umgebung funktioniert, müsste geprüft werden.

Die Lesbarkeit der Regeln kann weiter optimiert werden. Andere Formatierungen und das intelligente Zusammenfassen mehrerer Regeln könnte eine noch bessere Übersicht über die Logik geben. In der aktuellen Version sind Regeln sehr einfach gehalten, was dazu führt, dass Redundanzen auftreten.

Aufbau des Systems

Die fachlichen Anforderungen können aufgrund der Vorteile, die sich durch die Verwendung einer CEP-Software ergeben, sehr schnell und einfach umgesetzt werden. Auch die technische Umsetzung des Nachladens von neuen Regeldateien unterstützt die dynamische Anpassung an neue Anwendungsfälle optimal.

Die Funktion zum Nachladen von weiteren Regeldateien bietet eine einfache Möglichkeit, die Wissensbasis des Systems im laufenden Betrieb zu erweitern. Das Entfernen von Regeldateien ist in dieser Version nicht möglich, kann aber ohne großen Aufwand hinzugefügt werden. Um Änderungen der Regeln in bereits geladenen Dateien automatisch in die CEP-Engine zu übernehmen, gibt es den Drools Knowledge-Agent. Dieser kann per Polling-Verfahren die Dateien auf Änderungen überprüfen. Wenn das Änderungsdatum der Datei geändert wurde, lädt der Agent die Regeldatei erneut in die CEP-Engine. Eine automatische Aktualisierung der Regelbasis wäre durch die Implementierung dieses Knowledge-Agents ohne weiteres möglich. Die einfache Funktion, Regeldateien nachzuladen, ist für den Test, neues Regelwissen während des Betriebs einzufügen, völlig ausreichend.

Die erstellte Software enthält eine eigene Softwarekomponente für die Sensoren. Durch die eigenständigen Sensoren ist es möglich, mehrere Netzwerksegmente nach Portscans abzusuchen. In großen Netzwerken können nicht alle Daten über einen Punkt versendet werden, wodurch eine Aufteilung der Netzwerke in Segmente erfolgen muss. Diese Segmente haben jeweils eigene Zugänge zum Internet oder anderen Netzsegmenten. Erst durch die Platzierung der Sensoren in jedem Netzwerksegment ist die Analyse des gesamten Netzwerkes möglich. Durch die zentrale CEP-Komponente, an die die Sensoren ihre Ereignisse schicken, wird ein Gesamtüberblick über die Vorgänge im Netzwerk möglich. Ein weiterer Vorteil ist

die Analyse der Skalierbarkeit der Software. Durch das Hinzufügen weiterer Sensoren kann näherungsweise die maximale Leistungsfähigkeit des Systems einfach und schnell ermittelt werden.

Die Software bietet eine Testplattform, die sehr einfach gehalten ist und dadurch schnelle Änderungen zulässt. Parser, die zur Netzwerkpaketkonvertierung dienen, können ohne weiteres in Sensoren eingebaut werden. Ein Tausch der CEP-Engine wäre ebenfalls schnell erledigt. Nur eine Klasse referenziert die CEP-Engine und nutzt ihre Methoden. Der vorgestellte Quellcode, der zum Betrieb nötig ist, wäre somit sehr schnell getauscht.

Test und Testergebnisse

Ein weiterer Kritikpunkt ist die Überwachung des simulierten Netzwerkverkehrs. Die Simulation deckt natürlich nicht alle Konstellationen von Netzwerkpaketen in einer realen Umgebung ab. Wenn echter Netzwerkverkehr analysiert würde, wären folglich weitere Anpassungen der Regeln nötig. Es kann also auch keine genaue Aussage getroffenen werden, wie hoch die Rate der Falschmeldungen wäre.

Die Speicherauslastung der Software ist sehr hoch. Um diese zu senken, können evtl. durch einfachere Ereignismodelle und Regeln Ressourcen gespart werden. Möglichkeiten, die Performanz zu steigern, sind vielleicht andere JVM- oder CEP-Engine-Einstellungen. Einerseits können die belegbaren Speicherressourcen der JVM angehoben werden, andererseits kann das Threading-Verhalten der CEP-Engine angepasst werden. Diese Änderungen unterliegen jedoch ausgiebigen Tests.

Um das hohe Ereignisaufkommen in einem großen Netzwerk bewältigen zu können, muss die CEP-Engine möglichst performant arbeiten. Wie der Performanzvergleich [GSS01] zeigt, ist die CEP-Engine von Drools Fusion nicht die schnellste. Welche CEP-Engine für einen IDS-Einsatz in einem echten Netzwerk in Frage kommt, müsste in einem weiteren Performanztest evaluiert werden.

4.2 Vergleich mit IDS

In diesem Kapitel wird ein Vergleich der erstellten Software zu echten IDS durchgeführt. Der Vergleich ist so allgemein gehalten wie möglich und hat keinen Anspruch auf Vollständigkeit. Dennoch sollen einige wichtige Punkte aufgezeigt werden, die sich zwischen der erstellten Software bzw. der CEP-Engine und den IDS unterscheiden.

Die Regelsprachen von IDS sind wie beispielsweise bei Snort [Sno01] sehr gut an die Aufgaben eines IDS angepasst. Diese domänenspezifischen Sprachen bieten mit vorgegebenen Schlüsselwörtern eine viel einfachere und damit schnellere Regeldefinition. Unter der domänenspezifischen Sprache leidet jedoch die Flexibilität. Die komplexen Ereignisse, die durch CEP-Regeln erzeugt werden, können von jeglichen anderen Regeln wiederverwendet werden. Deshalb lassen sich neue fachliche Anwendungsfälle auf den Alten aufbauen. Zudem können komplexere Aufgaben, die ein IDS im Allgemeinen nicht vorsieht, wie zum Beispiel die Überprüfung von Kontobewegungen während eines Netzwerkangriffs, mitentwickelt werden.

Ein weiterer Punkt ist die Verwendung einer Datenbank. Im Grundaufbau eines IDS, wie in Kapitel 2.2 beschrieben, wird die Datenbank als Speicherort für die Protokollierung, Signaturen und Regeln benutzt. Eine Datenbank erleichtert die zentrale und strukturierte Speicherung. Um Angriffe besser nachvollziehen zu können, werden Protokolle angefertigt und in den Datenbanken gespeichert. Aus den Protokollen können anschließend wichtige Informationen abgeleitet werden, die bei der Senkung der Fehlerrate des IDS oder zur Behebung von Einbruchsschäden an den Computersystem nützlich sind. Zusätzlich können die vielen Regeln eines echten IDS in einer Datenbank verwaltet werden. Updates für die Signaturerkennung und allgemeine Regeln können durch den Einsatz einer Datenbank vom IDS-Hersteller besonders einfach erstellt und verteilt werden. Für die

Testsoftware wurde keine Datenbank angelegt. Die Anwendungsfälle sind so klein gehalten, dass sie keine Datenbank erfordern. Wenn ein reales Einsatzszenario geplant ist, würde eine Datenbank zur komfortablen Nutzung des IDS beitragen.

Die Sicherheit der IDS-Software spielt eine weitere wichtige Rolle. Das Netzwerk kann nur dann gut geschützt sein, wenn die Sicherheitskomponenten auch gut geschützt sind. Um dies zu erreichen, wird eine IDS-Software unter hohen Sicherheitsanforderungen entwickelt. Die Sicherheitsanforderungen fanden bei der Entwicklung dieser Testsoftware keine Betrachtung.

5 Fazit und Ausblick

Fazit

Nach der Implementierung und den Tests der Software sowie der Regeln lässt sich feststellen, dass Complex-Event-Processing gut für den Einsatz in einem Intrusion-Detection-System geeignet ist. CEP bietet eine sehr flexible Plattform, Lösungen für den Intrusion-Detection-Bereich zu entwickeln. Der Einsatz von Signaturen und das Erkennen von bestimmten Mustern kann sehr einfach implementiert werden. Zusätzlich dazu können Statistiken über das Verhalten generiert werden, die zur Einbruchserkennung beitragen. Somit können sowohl behavior-based als auch knowledge-based IDS mit CEP umgesetzt werden.

Auch der Grundaufbau eines IDS kann mit einem CEP-System ohne Umstände, wie die Testimplementierung zeigt, nachgebaut werden. Sensoren können unkompliziert Ereignisse an die Detektionskomponente senden, die CEP-Engine erkennt ohne Probleme die Portscan-Angriffe und die Reaktion auf Alarmmeldungen kann durch den Aktionsteil der Regeln erfolgen. Sollten die von der CEP-Engine bereitgestellten Funktionen nicht ausreichen, können auch eigene Funktionen geschrieben werden. Diese Funktionen können dann in Regeln verwendet werden, um noch komplexere Aufgaben zu lösen. Auch die Protokollierung von Ereignissen kann durch Regeln erfolgen. Diese führen Transaktionen zur Datenbank durch, indem sie im Aktionsteil Speicherfunktionen über CEP-Schnittstellen aufrufen.

Die verwendete CEP-Engine Drools Fusion kann zudem durch zusätzliche Bibliotheken erweitert werden, um zum Beispiel direkt aus Regeln Datenbankvorgänge durchzuführen. Die einfache Nutzung einer Datenbank zur Protokollierung von

Angriffen und zur Speicherung von Statistiken ist somit auch gewährleistet. Selbst die Visualisierung der Informationen, die das System erstellt, kann durch Zusatzprodukte der verwendeten CEP-Engine erfolgen.

Ausblick

Erste Recherchen ergaben ein viel höheres Paketaufkommen in echten Netzwerken als jenes, das die Testimplementierung verarbeiten konnte. Für die Zukunft wäre deshalb die grundlegende Betrachtung der Performanz von CEP-Systemen für den IDS-Einsatz sehr interessant. Dabei spielt natürlich auch die Optimierung der Regeln, der CEP-Engine und des restlichen Systems eine große Rolle. In diesem Bereich können Strategien und Algorithmen entwickelt werden, um die Performanz zu steigern. Um noch gezieltere Aussagen zur Eignung von CEP in IDS machen zu können, wäre eine Erweiterung der Testsoftware auf andere Anwendungsfälle und echten Netzwerkverkehr wünschenswert. In diesem Umfeld könnten dann auch Host-Intrusion-Detection und Intrusion-Prevention-Ansätze weiter verfolgt werden.

Eine andere interessante Frage ist, in wieweit sich Intrusion-Detection-Systeme mit anderen Unternehmensbereichen koppeln lassen. Da CEP-Systeme grundsätzlich sehr flexibel sind und in vielen anderen Bereichen eingesetzt werden, könnte die Verzahnung des IDS-Bereichs mit anderen Unternehmensbereichen evtl. äußerst elegant über die CEP-Technologie erfolgen. Welche Synergien mit anderen Bereichen existieren und welche Vorteile daraus entstehen, könnte in einer Studie ergründet werden.

Anhang

Der Arbeit ist eine Compact Disc mit dem Quellcode der erstellten Software sowie der benutzten Bibliotheken beigelegt. Zusätzlich befindet sich das Dokument der Bachelorarbeit darauf. Die nachfolgend vorgestellten Regeldateien sind ebenfalls in dem Quellcodeordner enthalten.

Regeldatei „distributor.drl“:

Mit der Regel „distributor for Analyse TCP Stream“ werden die Ereignisse aus dem default Entry-Point in den Entry-Point "tcpAnalysis" eingefügt. Diese Regel führt das Content-Based-Routing durch.

```
1 package drl.distributor;
2 import de.fhhannover.ba.rohde.events.Event
3
4 declare Event
5       @role( event )
6       @timestamp( timestamp )
7 end
8
9 rule "distributor for Analyse TCP Stream"
10     dialect "mvel"
11     when
12             $e1 : Event();
13     then
14        drools.entryPoints["tcpAnalysis"].insert( $e1 );
15 end
```

Regeldatei „normalTcpConnection.drl“:

Die Regeln aus dieser Datei teilen Pakete anhand der vorgegebenen Muster in die Ereignisströme „fullTcpConnection“, „disconnectedStream“ und „dataSend-Stream“ ein.

```
1  package drl.normalTcpConnection;
2  import de.fhhannover.ba.rohde.entities.TCPIPPacket;
3  import de.fhhannover.ba.rohde.entities.IPHeader;
4  import de.fhhannover.ba.rohde.entities.TCPHeader;
5  import
   de.fhhannover.ba.rohde.entities.TCPHeader.CONTROL_FLAG;
6  import de.fhhannover.ba.rohde.events.TCPIPEvent;
7  import
   de.fhhannover.ba.rohde.events.EstablishedConnectionEvent;
8  import de.fhhannover.ba.rohde.events.DataSendEvent;
9  import
10  de.fhhannover.ba.rohde.events.DisconnectedConnectionEvent;
11
12 declare TCPIPEvent
13       @role( event )
14       @timestamp( timestamp )
15       @expires(3m)
16 end
17
18 declare EstablishedConnectionEvent
19       @role( event )
20       @timestamp( timestamp )
21       @expires(30m)
22 end
23
24 declare DataSendEvent
25       @role( event )
26       @timestamp( timestamp )
27       @expires(30m)
28 end
29
30 declare DisconnectedConnectionEvent
31       @role( event )
32       @timestamp( timestamp )
33 end
34
35 rule "full established tcp connection"
36   dialect "mvel"
37   when
38     $e1 : TCPIPEvent(packet.tcpHeader.controlFlag ==
39     CONTROL_FLAG.SYN) from entry-point "tcpAnalysis"
40     $e1tcpHeader : TCPHeader() from $e1.packet.tcpHeader
41     $e1ipHeader : IPHeader() from $e1.packet.ipHeader
42
43     $e2 : TCPIPEvent(packet.tcpHeader.controlFlag ==
44     CONTROL_FLAG.SYN_ACK
45     && $e1ipHeader.sourceIP == packet.ipHeader.destIP
46     && $e1ipHeader.destIP == packet.ipHeader.sourceIP
47     && $e1tcpHeader.sourcePort == packet.tcpHeader.destPort
48     && $e1tcpHeader.destPort == packet.tcpHeader.sourcePort
```

```
49      ) from entry-point "tcpAnalysis"
50      $e2tcpHeader : TCPHeader() from $e2.packet.tcpHeader
51      $e2ipHeader : IPHeader() from $e2.packet.ipHeader
52
53      $e3 : TCPIPEvent(packet.tcpHeader.controlFlag ==
54      CONTROL_FLAG.ACK
55      && $e2ipHeader.sourceIP == packet.ipHeader.destIP
56      && $e2ipHeader.destIP == packet.ipHeader.sourceIP
57      && $e2tcpHeader.sourcePort == packet.tcpHeader.destPort
58      && $e2tcpHeader.destPort == packet.tcpHeader.sourcePort
59      ) from entry-point "tcpAnalysis"
60    then
61      retract($e1);
62      retract($e2);
63      retract($e3);
64      EstablishedConnectionEvent con = new
65      EstablishedConnectionEvent( System.currentTimeMillis(),
66        $e1ipHeader.sourceIP, $e1ipHeader.destIP,
67        $e1tcpHeader.sourcePort, $e1tcpHeader.destPort);
68      drools.entryPoints["fullTcpConnection"].insert( con );
69  end
70  rule "data send over tcp connection"
71    dialect "mvel"
72    when
73      $e1 : TCPIPEvent(packet.tcpHeader.controlFlag ==
74      CONTROL_FLAG.NULL) from entry-point "tcpAnalysis"
75      $e1tcpHeader : TCPHeader() from $e1.packet.tcpHeader
76      $e1ipHeader : IPHeader() from $e1.packet.ipHeader
77
78      $e2 : TCPIPEvent(packet.tcpHeader.controlFlag ==
79      CONTROL_FLAG.ACK
80      && $e1ipHeader.sourceIP == packet.ipHeader.destIP
81      && $e1ipHeader.destIP == packet.ipHeader.sourceIP
82      && $e1tcpHeader.sourcePort == packet.tcpHeader.destPort
83      && $e1tcpHeader.destPort == packet.tcpHeader.sourcePort
84      ) from entry-point "tcpAnalysis"
85
86      not (DataSendEvent( sourceIP == $e1ipHeader.sourceIP
87      && destIP ==  $e1ipHeader.destIP
88      && sourcePort == $e1tcpHeader.sourcePort
89      && destPort == $e1tcpHeader.destPort
90      ) from entry-point "dataSendStream")
91
92      $eConnection : EstablishedConnectionEvent( sourceIP ==
93      $e1ipHeader.sourceIP
94      && destIP ==  $e1ipHeader.destIP
95      && sourcePort == $e1tcpHeader.sourcePort
96      && destPort == $e1tcpHeader.destPort
97      ) from entry-point "fullTcpConnection"
98    then
```

```
 99     retract($e1);
100     retract($e2);
101     DataSendEvent dse = new DataSendEvent(
102        System.currentTimeMillis(), $e1ipHeader.sourceIP,
103        $e1ipHeader.destIP, $e1tcpHeader.sourcePort,
104        $e1tcpHeader.destPort);
105     drools.entryPoints["dataSendStream"].insert( dse );
106 end
107
108 rule "full tcp disconnect"
109   dialect "mvel"
110   when
111     $e1 : TCPIPEvent(packet.tcpHeader.controlFlag ==
112     CONTROL_FLAG.FIN) from entry-point "tcpAnalysis"
113     $e1tcpHeader : TCPHeader() from $e1.packet.tcpHeader
114     $e1ipHeader : IPHeader() from $e1.packet.ipHeader
115
116     $e2 : TCPIPEvent(packet.tcpHeader.controlFlag ==
117     CONTROL_FLAG.ACK
118     && $e1ipHeader.sourceIP == packet.ipHeader.destIP
119     && $e1ipHeader.destIP == packet.ipHeader.sourceIP
120     && $e1tcpHeader.sourcePort == packet.tcpHeader.destPort
121     && $e1tcpHeader.destPort == packet.tcpHeader.sourcePort
122     ) from entry-point "tcpAnalysis"
123     $e2tcpHeader : TCPHeader() from $e2.packet.tcpHeader
124     $e2ipHeader : IPHeader() from $e2.packet.ipHeader
125
126     $e3 : TCPIPEvent(packet.tcpHeader.controlFlag ==
127     CONTROL_FLAG.FIN
128     && $e2ipHeader.sourceIP == packet.ipHeader.sourceIP
129     && $e2ipHeader.destIP == packet.ipHeader.destIP
130     && $e2tcpHeader.sourcePort == packet.tcpHeader.sourcePort
131     && $e2tcpHeader.destPort == packet.tcpHeader.destPort
132     ) from entry-point "tcpAnalysis"
133     $e3tcpHeader : TCPHeader() from $e3.packet.tcpHeader
134     $e3ipHeader : IPHeader() from $e3.packet.ipHeader
135
136     $e4 : TCPIPEvent(packet.tcpHeader.controlFlag ==
137     CONTROL_FLAG.ACK
138     && $e3ipHeader.sourceIP == packet.ipHeader.destIP
139     && $e3ipHeader.destIP == packet.ipHeader.sourceIP
140     && $e3tcpHeader.sourcePort == packet.tcpHeader.destPort
141     && $e3tcpHeader.destPort == packet.tcpHeader.sourcePort
142     ) from entry-point "tcpAnalysis"
143   then
144     retract($e1);
145     retract($e2);
146     retract($e3);
147     retract($e4);
148     DisconnectedConnectionEvent dc = new
```

```
149     DisconnectedConnectionEvent( System.currentTimeMillis(),
150       $e1ipHeader.sourceIP, $e1ipHeader.destIP,
151       $e1tcpHeader.sourcePort, $e1tcpHeader.destPort);
152     drools.entryPoints["disconnectedStream"].insert( dc );
153 end
154
155 rule "full tcp disconnect clean up"
156   dialect "mvel"
157   when
158     $e1 : DisconnectedConnectionEvent() from entry-point
159     "disconnectedStream"
160
161     $e2 : DataSendEvent(sourceIP == $e1.sourceIP
162     && destIP ==  $e1.destIP
163     && sourcePort == $e1.sourcePort
164     && destPort == $e1.destPort
165     ) from entry-point "dataSendStream"
166
167     $eConnection : EstablishedConnectionEvent(
168     sourceIP == $e1.sourceIP
169     && destIP ==  $e1.destIP
170     && sourcePort == $e1.sourcePort
171     && destPort == $e1.destPort
172     ) from entry-point "fullTcpConnection"
173   then
174     retract($e1);
175     retract($e2);
176     retract($eConnection);
177 end
```

Regeldatei „tcpPortscanDetection.drl“:

Die Regeln dieser Datei erkennen die Muster, die bei einem Portscan auftreten und generieren anschließend Alarmereignisse, die in den „alarmEventStream“ eingefügt werden.

```
1 package drl.tcpPortscanDetection;
2 import de.fhhannover.ba.rohde.events.TCPIPEvent;
3 import de.fhhannover.ba.rohde.events.AlarmEvent;
4 import de.fhhannover.ba.rohde.events.DataSendEvent;
5 import
6 de.fhhannover.ba.rohde.events.EstablishedConnectionEvent;
7 import
8 de.fhhannover.ba.rohde.events.DisconnectedConnectionEvent;
9 import de.fhhannover.ba.rohde.entities.TCPIPPacket;
10 import de.fhhannover.ba.rohde.entities.IPHeader;
```

```
11 import de.fhhannover.ba.rohde.entities.TCPHeader;
12 import
13 de.fhhannover.ba.rohde.entities.TCPHeader.CONTROL_FLAG;
14
15 declare AlarmEvent
16       @role( event )
17       @timestamp( timestamp )
18 end
19
20 rule "tcp connect on closed port (TCPConnectScan and
21       TCPSYNScan)"
22   dialect "mvel"
23   when
24     $e1 : TCPIPEvent(packet.tcpHeader.controlFlag ==
25     CONTROL_FLAG.SYN) from entry-point "tcpAnalysis"
26     $e1tcpHeader : TCPHeader() from $e1.packet.tcpHeader
27     $e1ipHeader : IPHeader() from $e1.packet.ipHeader
28
29     $e2 : TCPIPEvent(packet.tcpHeader.controlFlag ==
30     CONTROL_FLAG.RST
31     && $e1ipHeader.sourceIP == packet.ipHeader.destIP
32     && $e1ipHeader.destIP == packet.ipHeader.sourceIP
33     && $e1tcpHeader.sourcePort == packet.tcpHeader.destPort
34     && $e1tcpHeader.destPort == packet.tcpHeader.sourcePort
35     ) from entry-point "tcpAnalysis"
36   then
37     retract($e1);
38     retract($e2);
39     AlarmEvent alarm = new AlarmEvent(
40       System.currentTimeMillis(), $e1ipHeader.sourceIP,
41       $e1ipHeader.destIP, $e1tcpHeader.sourcePort,
42       $e1tcpHeader.destPort, "Potential Portscan (SYN- or
43       connect scan on closed Port)!")
44     drools.entryPoints["alarmEventStream"].insert( alarm );
45 end
46
47 rule "tcp connect on open port with direct reset
48      (TCPSYNScan)"
49   dialect "mvel"
50   when
51     $e1 : TCPIPEvent(packet.tcpHeader.controlFlag ==
52     CONTROL_FLAG.SYN) from entry-point "tcpAnalysis"
53     $e1tcpHeader : TCPHeader() from $e1.packet.tcpHeader
54     $e1ipHeader : IPHeader() from $e1.packet.ipHeader
55
56     $e2 : TCPIPEvent(packet.tcpHeader.controlFlag ==
57     CONTROL_FLAG.SYN_ACK
58     && $e1ipHeader.sourceIP == packet.ipHeader.destIP
59     && $e1ipHeader.destIP == packet.ipHeader.sourceIP
60     && $e1tcpHeader.sourcePort == packet.tcpHeader.destPort
```

```
61      && $e1tcpHeader.destPort == packet.tcpHeader.sourcePort
62      ) from entry-point "tcpAnalysis"
63      $e2tcpHeader : TCPHeader() from $e2.packet.tcpHeader
64      $e2ipHeader : IPHeader() from $e2.packet.ipHeader
65
66      $e3 : TCPIPEvent(packet.tcpHeader.controlFlag ==
67      CONTROL_FLAG.RST
68      && $e2ipHeader.sourceIP == packet.ipHeader.destIP
69      && $e2ipHeader.destIP == packet.ipHeader.sourceIP
70      && $e2tcpHeader.sourcePort == packet.tcpHeader.destPort
71      && $e2tcpHeader.destPort == packet.tcpHeader.sourcePort
72      ) from entry-point "tcpAnalysis"
73    then
74      retract($e1);
75      retract($e2);
76      retract($e3);
77      AlarmEvent alarm = new AlarmEvent(
78        System.currentTimeMillis(), $e1ipHeader.sourceIP,
79        $e1ipHeader.destIP, $e1tcpHeader.sourcePort,
80        $e1tcpHeader.destPort, "Potential Portscan (SYN scan on
81        open Port)!")
82      drools.entryPoints["alarmEventStream"].insert( alarm );
83  end
84
85  rule "tcp connect on closed port with FIN"
86    dialect "mvel"
87    when
88      $e1 : TCPIPEvent(packet.tcpHeader.controlFlag ==
89      CONTROL_FLAG.FIN) from entry-point "tcpAnalysis"
90      $e1tcpHeader : TCPHeader() from $e1.packet.tcpHeader
91      $e1ipHeader : IPHeader() from $e1.packet.ipHeader
92
93      $e2 : TCPIPEvent(packet.tcpHeader.controlFlag ==
94      CONTROL_FLAG.RST
95      && $e1ipHeader.sourceIP == packet.ipHeader.destIP
96      && $e1ipHeader.destIP == packet.ipHeader.sourceIP
97      && $e1tcpHeader.sourcePort == packet.tcpHeader.destPort
98      && $e1tcpHeader.destPort == packet.tcpHeader.sourcePort
99      ) from entry-point "tcpAnalysis"
100   then
101     retract($e1);
102     retract($e2);
103     AlarmEvent alarm = new AlarmEvent(
104       System.currentTimeMillis(), $e1ipHeader.sourceIP,
105       $e1ipHeader.destIP, $e1tcpHeader.sourcePort,
106       $e1tcpHeader.destPort, "Potential Portscan (FIN scan on
107       closed Port)!")
108     drools.entryPoints["alarmEventStream"].insert( alarm );
109 end
110
```

```
111
112 rule "tcp connect on open port with FIN"
113   dialect "mvel"
114   when
115     $e1 : TCPIPEvent(packet.tcpHeader.controlFlag ==
116     CONTROL_FLAG.FIN) from entry-point "tcpAnalysis"
117     $e1tcpHeader : TCPHeader() from $e1.packet.tcpHeader
118     $e1ipHeader : IPHeader() from $e1.packet.ipHeader
119
120     not ( TCPIPEvent(packet.tcpHeader.controlFlag ==
121     CONTROL_FLAG.RST
122     && $e1ipHeader.sourceIP == packet.ipHeader.destIP
123     && $e1ipHeader.destIP == packet.ipHeader.sourceIP
124     && $e1tcpHeader.sourcePort == packet.tcpHeader.destPort
125     && $e1tcpHeader.destPort == packet.tcpHeader.sourcePort
126     && this after[0,1s] $e1
127     ) from entry-point "tcpAnalysis" )
128
129     not (TCPIPEvent(packet.tcpHeader.controlFlag ==
130     CONTROL_FLAG.FIN
131     && $e1ipHeader.sourceIP == packet.ipHeader.destIP
132     && $e1ipHeader.destIP == packet.ipHeader.sourceIP
133     && $e1tcpHeader.sourcePort == packet.tcpHeader.destPort
134     && $e1tcpHeader.destPort == packet.tcpHeader.sourcePort
135     ) from entry-point "tcpAnalysis" )
136
137     not (EstablishedConnectionEvent(
138     $e1ipHeader.sourceIP == sourceIP
139     && $e1ipHeader.destIP == destIP
140     && $e1tcpHeader.sourcePort == sourcePort
141     && $e1tcpHeader.destPort == destPort
142     ) from entry-point "fullTcpConnection" )
143   then
144     retract($e1);
145     AlarmEvent alarm = new AlarmEvent(
146       System.currentTimeMillis(), $e1ipHeader.sourceIP,
147       $e1ipHeader.destIP, $e1tcpHeader.sourcePort,
148       $e1tcpHeader.destPort, "Potential Portscan (FIN scan on
149       open Port)!")
150     drools.entryPoints["alarmEventStream"].insert( alarm );
151 end
152
153 rule "tcp connect on open port (TCP connect Scan)"
154   dialect "mvel"
155   when
156     $e1 : DisconnectedConnectionEvent()
157     from entry-point "disconnectedStream"
158
159     not( DataSendEvent(sourceIP == $e1.sourceIP
160     && destIP ==  $e1.destIP
```

```
161     && sourcePort == $e1.sourcePort
162     && destPort == $e1.destPort
163     ) from entry-point "dataSendStream")
164
165     $eConnection : EstablishedConnectionEvent(
166     sourceIP == $e1.sourceIP
167     && destIP ==  $e1.destIP
168     && sourcePort == $e1.sourcePort
169     && destPort == $e1.destPort
170     ) from entry-point "fullTcpConnection"
171   then
172     retract($e1);
173     retract($eConnection);
174     AlarmEvent alarm = new AlarmEvent(
175       System.currentTimeMillis(), $e1.sourceIP,$e1.destIP,
176       $e1.sourcePort, $e1.destPort,"Potential Portscan (TCP
177       connect scan)!");
178     drools.entryPoints["alarmEventStream"].insert( alarm );
179 end
```

Regeldatei „portscanAlarmRules.drl“:

Die Regeln dieser Datei zeigen Möglichkeiten, wie Informationen aus Ereignissen aggregiert werden können. In diesem Fall werden Alarmereignisse der letzten zwei Tage gezählt. Das letzte aufgetretene Ereignis wird mit der Summe der Alarmereignisse an die CEP Komponente übergeben.

```
1 package drl.portscanAlarmRules;
2 import de.fhhannover.ba.rohde.events.AlarmEvent;
3 import
4 de.fhhannover.ba.rohde.ids_server.rulehandler.PortscanAlarmH
5 andler;
6
7 rule "get all events from alarmEventStream to display"
8   dialect "mvel"
9   when
10    $maxTimestamp : Number()
11    from accumulate( $ae1 : AlarmEvent()
12      over window:time( 2d ) from entry-point
13     "alarmEventStream" ,
14      max($ae1.timestamp))
15
16    $alarmEvent: AlarmEvent(
17      (Double)timestamp == (Double) $maxTimestamp )
18    from entry-point "alarmEventStream"
19
```

```
20     $num : Number(!((intValue % 5) == 0), intValue > 0)
21       from accumulate( $ae2 : AlarmEvent(
22         $alarmEvent.destIP == destIP)
23         over window:time( 2d ) from entry-point
24         "alarmEventStream" , count($ae2))
25   then
26     PortscanAlarmHandler.update($alarmEvent, $num);
27 end
28
29 rule "get all events from alarmEventStream to display and
30      send email"
31   dialect "mvel"
32   when
33     $maxTimestamp : Number()
34     from accumulate($ae1 : AlarmEvent()
35       over window:time( 2d ) from entry-point
36       "alarmEventStream" , max($ae1.timestamp))
37
38     $alarmEvent: AlarmEvent(
39       (Double)timestamp == (Double)$maxTimestamp )
40     from entry-point "alarmEventStream"
41
42     $num : Number(((intValue % 5) == 0), intValue > 0)
43       from accumulate( $ae2 : AlarmEvent(
44         $alarmEvent.destIP == destIP)
45         over window:time( 2d ) from entry-point
46         "alarmEventStream" , count($ae2))
47   then
48     PortscanAlarmHandler.updateWithEmailNotification(
49       $alarmEvent, $num);
50 end
```

Literaturverzeichnis

[ALB01] Aniello L., Lodi G., Baldoni R.: Inter-Domain Stealthy Port Scan Detectionthrough Complex Event Processing. (2011), http://www.dis.uniroma1.it/~midlab/articoli/portscan_detection_esperMidlabTechRep1:11.pdf. Abgerufen 6. August 2011

[Bas01] Bass, T.: Simple Event Processing != Complex Event Processing. (2007), http://www.thecepblog.com/2007/12/16/simple-event-processing-complex-event-processing/. Abgerufen 6. August 2011

[BD01] Bruns, R., Dunkel, J. : Event-Driven Architecture: Softwarearchitektur für ereignisgesteuerte Geschäftsprozesse. Springer, Berlin Heidelberg (2010)

[Cod01] Codehaus: mvel: mvflex expression language: Home. (o.J.), http://mvel.codehaus.org/. Abgerufen 2. August 2011

[Deb01] Debar H.: Intrusion Detection FAQ: What is knowledge-based intrusion detection?. (2010), http://www.sans.org/security-resources/idfaq/knowledge_based.php. Abgerufen 5. August 2011

[Deb02] Debar H.: Intrusion Detection FAQ: What is behavior-based intrusion detection?. (2010), http://www.sans.org/security-resources/idfaq/behavior_based.php. Abgerufen 5. August 2011

[Doo01] Doorenbos, R.: Production Matching for Large Learning Systems. (1995), http://reports-archive.adm.cs.cmu.edu/anon/1995/CMU-CS-95-113.pdf. Abgerufen 1.August 2011

[Eck01] Eckert M.: Complex Event Processing with XChangeEQ: Language Design, Formal Semantics, and Incremental Evaluation for Querying Events. (2008), http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.145.3266&rep=rep1&type=pdf. Abgerufen 15. August 2011

[Esp01] ESPERTECH: Event Stream Intelligence: Esper & Nesper. (2011), http://esper.codehaus.org/. Abgerufen 1. August 2011

[Etz01] Etzion O.: Event Processing Thinking: On BRMS and EP . (2009), http://epthinking.blogspot.com/2009/02/on-brms-and-ep.html. Abgerufen 6. August 2011

[Far01] Farroukh A.: Enhancing Performance of Vulnerability-Based IntrusionDetection Systems. (2010), https://tspace.library.utoronto.ca/bitstream/1807/25574/3/Farroukh_Amer_201011_MASc_thesis.pdf. Abgerufen 15. August 2011

[For01] Forgy C.L.: Rete: A Fast Algorithm for theMany Pattern/Many ObjectPattern Match Problem*. (1982), https://cit-server.cit.tu-berlin.de/~battre/db.rdf.forgy.90.rete.pdf. Abgerufen 5. August 2011

[GSS01] Grohe, S., Schlameuß, C., Sommer, R.: Performancevergleich von CEP-Engines. (2010), http://elib.uni-stuttgart.de/opus/volltexte/2010/5526/pdf/FACH_0115.pdf. Abgerufen 15. August 2011

[JBo01] The JBoss Drools team: Drools Expert. (2011), http://www.jboss.org/drools/drools-expert.html. Abgerufen 1. August 2011

[JBo02] The JBoss Drools team: Drools Fusion. (2011), http://www.jboss.org/drools/drools-fusion.html. Abgerufen 1. August 2011

[JBo03] The JBoss Drools team: Drools Fusion User Guide. (2011), http://docs.jboss.org/drools/release/5.2.0.Final/drools-fusion-docs/html_single/index.html. Abgerufen 3. August 2011

[JBo04] The JBoss Drools team: Drools Introduction and General User Guide. (2011), http://docs.jboss.org/drools/release/5.2.0.Final/droolsjbpm-introduction-docs/html_single/index.html. Abgerufen 5. August 2011

[JBo05] JBoss Cummunity Team: Drools Expert User Guide. (2011), http://docs.jboss.org/drools/release/5.2.0.Final/drools-expert-docs/html_single/index.html. Abgerufen 5. August 2011

[Luc01] Luckham D., Schulte R.: EPTS Event Processing Glossary v1.1 . (2008), http://www.complexevents.com/2008/08/31/event-processing-glossary-version-11/. Abgerufen 3. August 2011

[Luc02] Luckham, D.: The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems. Addison-Wesley, Boston (2002)

[Lyo01] Lyon, G.: Nmap-Referenz-Handbuch (Man Page). (o.J.), http://nmap.org/man/de/man-port-scanning-techniques.html. Abgerufen 26. Juli 2011

[Owe01] Owens T. J.: SURVEY OF EVENT PROCESSING. (2007), http://www.dtic.mil/cgi-bin/GetTRDoc?Location=U2&doc=GetTRDoc.pdf&AD=ADA475386. Abgerufen 15. August 2011

[Per01] Perrochon L.: Using Context-Based Correlation inNetwork Operations and Management. (1999), http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.44.849&rep=rep1&type=pdf. Abgerufen 15. August 2011

[Pin01] Pinnow D.: HP-Studie zu Schäden durch Online-Kriminalität: Anstieg um 56 Prozent im Jahr 2010. (2011), http://www.datensicherheit.de/aktuelles/hp-studie-zu-schaeden-durch-online-kriminalitaet-anstieg-um-56-prozent-im-jahr-2010-15401. Abgerufen 5. August 2011

[RFC793] o.A.: RFC793 TRANSMISSION CONTROL PROTOCOL. (1981), http://tools.ietf.org/html/rfc793. Abgerufen 26. Juli 2011

[Sno01] Snort Team: SNORT Users Manual 2.9.0: The Snort Project. (2011), http://www.snort.org/. Abgerufen 10. August 2011

[Spe01] Spenneberg, R.: Intrusion Dectection und Prevention mit Snort 2 & Co. Einbrüche auf Linux-Servern erkennen und verhindern. Addison-Wesley, München (2005)

[STS01] Staniford-Chen, S., Tung, B., Schnackenberg, D.: The Common Intrusion Detection Framework (CIDF). (1998), http://gost.isi.edu/cidf/drafts/architecture.txt. Abgerufen 6. August 2011

[TG01] Turchin, Y.,Gal, A.: Tuning Complex Event Processing Rules using the PredictionCorrectionParadigm. (2009), http://ie.technion.ac.il/~avigal/trp.pdf. Abgerufen 6. August 2011

[Vin01] Vincent, P.: Differences between a BRE and a rule-driven CEP engine (Part 1). (2007), http://tibcoblogs.com/cep/2007/06/26/differences-between-a-bre-and-a-rule-driven-cep-engine-part-1/. Abgerufen 6. August 2011