X.systems.press

X.systems.press ist eine praxisorientierte Reihe zur Entwicklung und Administration von Betriebssystemen, Netzwerken und Datenbanken.

Michael Meier

Intrusion Detection effektiv!

Modellierung und Analyse von Angriffsmustern

Mit 104 Abbildungen, 16 Tabellen
und CD-ROM

Springer

Dr. Michael Meier
Fachbereich Informatik
Universität Dortmund
44221 Dortmund
michael.meier@udo.edu

Bibliografische Information der Deutschen Nationalbibliothek
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.d-nb.de abrufbar.

ISSN 1611-8618
ISBN-13 978-3-540-48251-2 Springer Berlin Heidelberg New York

Springer ist ein Unternehmen von Springer Science+Business Media
springer.de

Satz: Druckfertige Daten des Autors
Herstellung: LE-TEX, Jelonek, Schmidt & Vöckler GbR, Leipzig
Umschlaggestaltung: KünkelLopka Werbeagentur, Heidelberg
Gedruckt auf säurefreiem Papier 33/3100 YL – 5 4 3 2 1 0

Widmung

Meinen Eltern.

Vorwort

In dem Forschungsgebiet *Intrusion Detection* wird seit Mitte der 80er Jahre gearbeitet. Seit einigen Jahren sind kommerzielle *Intrusion-Detection-Systeme (IDS)* verfügbar, die ergänzend zu präventiven Sicherheitsmechanismen zum Schutz von informationstechnischen Systemen eingesetzt werden können. Die Wirksamkeit zurzeit verfügbarer Systeme bleibt jedoch weit hinter den Erwartungen der Nutzer zurück. Ursache dafür ist eine große Zahl von Fehlalarmen sowie eine schwer zu beziffernde Zahl von unerkannten Sicherheitsverletzungen, durch die der tägliche Einsatz von IDS geprägt ist. Darüber hinaus wird es aufgrund der anfallenden Datenvolumen zunehmend schwieriger in modernen leistungsfähigen Systemen zeitnah Sicherheitsverletzungen zu erkennen.

Dieses Buch betrachtet diese Problemfelder. Dabei werden systematisch fundiert die folgenden Fragestellungen diskutiert und entsprechende Lösungen dargestellt:

- Was sind die relevanten Charakteristika von Sicherheitsverletzungen, die in Angriffsmustern spezifiziert werden müssen, um eine exakte Erkennung zu ermöglichen?
- Wie können komplexe Angriffsmuster geeignet modelliert und beschrieben werden?
- Wie kann die Erkennung der Angriffsmuster effizient(er) realisiert werden?

Dem Leser werden dabei Werkzeuge zur Bewertung existierender und Konstruktion neuer IDS vorgestellt. Schwerpunkte sind dabei die systematische Betrachtung sowie exakte Modellierung und Beschreibung von Angriffsmustern. Darüber hinaus werden existierende Analyseverfahren zur Erkennung von Angriffsmustern beschrieben und ein hinsichtlich Effizienz optimiertes Verfahren vorgestellt.

Dieses Buch ist zum größten Teil während meiner Tätigkeit als wissenschaftlicher Mitarbeiter am Lehrstuhl Rechnernetze und Kommunikationssysteme der Brandenburgischen Technischen Universität Cottbus entstanden. Für die wissenschaftliche Betreuung meiner Arbeiten, die eingeräum-

ten Freiräume sowie die in vielerlei Hinsicht angenehme Arbeitsatmosphäre an seinem Lehrstuhl danke ich Hartmut König. Meine Arbeiten zu diesem Buch profitierten unter anderem von der fruchtbaren Zusammenarbeit mit Ulrich Flegel. Während zahlloser Treffen auf Workshops und Konferenzen sowie Besuchen in Dortmund und Cottbus diskutierten wir die Ausdrucksstärke von Modellierungsansätzen für Angriffssignaturen. Als Ergebnis unserer Zusammenarbeit entstand ein allgemeiner Ansatz zur Modellierung von Angriffssignaturen. Mario Schölzel gilt besonderer Dank für seine Unterstützung bei der formalen Definition von Signaturnetzen. Niels Bischof, Christian Rohr und Sebastian Schmerl danke ich für ihre Arbeiten und ihre konstruktiven Ideen zu den Sprachen SHEDEL und EDL sowie den Werkzeugen SAM und SEG, die auf beiliegender CD-ROM enthalten sind.

Dortmund, im November 2006

Inhaltsverzeichnis

Abkürzungsverzeichnis

AID	Adaptive Intrusion Detection system
ANIDA	Aachen Network Intrusion Detection Architecture
ASAX	Advanced Security audit trail Analyzer on uniX
BSM	Basic Security Module
CC	Coordination Center
CERT	Computer Emergence Response Team
CIDF	Common Intrusion Detection Framework
CLIPS	C Language Integrated Production System
CMDS	Computer Misuse Detection System
CPA	Coloured Petri Net Automaton
DARPA	Defense Advanced Research Projects Agency
DPEM	Distributed Program Execution Monitor
ECA	Event Condition Action
EDL	Event Description Language
EGID	Effektive Gruppen-ID
EMERALD	Event Monitoring Enabling Responses to Anomalous Live Disturbances
ET	Ereignis-Token
EUID	Effektive User-ID
FL	Failed Login
FRT	Fire Ready Tokens
GID	Gruppen-ID
GUI	Graphical User Interface
ID	Identifikator
IDIOT	Intrusion Detection In Our Time
IDMEF	Intrusion Detection Message Exchange Format
IDRS	Intrusion Detection and Response System
IDS	Intrusion-Detection-System
IDWG	Intrusion Detection Working Group
IETF	Internet Engineering Task Force
Inter-EB	Inter-Ereignis-Bedingung
Intra-EB	Intra-Ereignis-Bedingung
IP	Internet Protokoll
IPS	Intrusion-Prevention-System

IT-	Informationstechnisch (im Zusammenhang)
MIDAS	Multics Intrusion Detection and Alerting System
MuSig	Misuse Signature
OID	Objekt-ID
P-BEST	Production-Based Expert System Toolset
PID	Reale Prozess-ID
RGID	Reale Gruppen-ID
RUID	Reale User-ID
RUSSEL	RUle-baSed Sequence Evaluation Language
SAM	Signature Analysis Module
SEG	Snoopy-based EDL GUI
SHEDEL	Simple Hierarchical Event DEscription Language
STAT	State Transition Analysis Technique
StraFER	Straight Forward Event Recognition
SUID	Set-User-ID
TT	Token-Token
UID	User-ID

1 Einleitung

Die moderne Informationsgesellschaft basiert auf komplexen informationstechnischen Infrastrukturen, die einen zunehmenden Grad an Vernetzung aufweisen. Immer mehr private und öffentliche Institutionen verwenden *informationstechnische Systeme (IT-Systeme)*, um ihre Dienste effektiver und effizienter anbieten und abwickeln zu können. Durch die globale Vernetzung wird der Zugang zu Diensten in Zukunft für jedermann, zu jederzeit und von jedem Ort möglich sein. Die Vorteile dieser Entwicklungen sind unbestritten. Sie bringen jedoch auch Nachteile mit sich. Aus der Verlagerung vieler gesellschaftlicher Prozesse auf IT-Systeme resultiert eine direkte Abhängigkeit unserer Gesellschaft von diesen Systemen. Dadurch gewinnt der Schutz von IT-Systemen zunehmend an Bedeutung.

Klassische Sicherheitsmechanismen, z. B. kryptographische Verfahren und Zugangskontrollsysteme, wie beispielsweise Firewalls, sind notwendige Komponenten zum Schutz von IT-Infrastrukturen und heute weit verbreitet. Während bisher vorrangig präventive Maßnahmen und Mechanismen im Vordergrund standen, wird zunehmend deutlich, dass die Sicherheit von IT-Systemen nicht allein durch Prävention erreichbar ist. Vielmehr stellt Prävention einen Grundpfeiler dar, neben dem reaktive Aspekte der Sicherheit von IT-Systemen stehen.

Um in der Lage zu sein, auf Sicherheitsverletzungen reagieren zu können, sind Mechanismen zur Erkennung von Sicherheitsverletzungen erforderlich. Seit Mitte der 80er Jahre hat sich daher das Forschungsgebiet der *Angriffserkennung (Intrusion Detection)* entwickelt. Seit einiger Zeit sind kommerzielle *Intrusion-Detection-Systeme (IDS)* verfügbar, die ergänzend zu präventiven Sicherheitsmechanismen zum Schutz von IT-Systemen eingesetzt werden können.

Die in diesem Gebiet entwickelten Verfahren zur Erkennung von Sicherheitsverletzungen können grob in Anomalieerkennung und Missbrauchserkennung unterteilt werden. Verfahren zur *Anomalieerkennung* basieren auf der expliziten Definition von normalem Verhalten und erkennen Abweichungen von dieser Norm. Problematisch bei diesen Verfahren sind die inhärente Unschärfe der gelieferten Ergebnisse sowie die Frage, ob jede Anomalie eine Sicherheitsverletzung darstellt.

Zur *Missbrauchserkennung* ist das Vorliegen expliziter Definitionen von Angriffsmustern in Form von Signaturen erforderlich. Beobachtete Aktionen werden auf Übereinstimmung mit den Signaturen überprüft. Prinzipiell sind Verfahren zur Missbrauchserkennung bezüglich der Erkennungsgenauigkeit sehr robust und liefern scharfe Ergebnisse, auf deren Grundlage Reaktionen auf Angriffe veranlasst werden können. Die Missbrauchserkennung stellt daher ein unverzichtbares Basisauswertungsverfahren von IDS dar, das um Anomalieerkennung ergänzt werden kann. Naturgemäß sind Verfahren zur Missbrauchserkennung jedoch auf die Erkennung von bekannten, durch Signaturen repräsentierten Angriffen beschränkt.

Obwohl die Notwendigkeit von Systemen zur Missbrauchserkennung unwidersprochen bleibt, ist der Einsatz derzeit verfügbarer Systeme mit einer Reihe von Problemen verbunden. Eines der Hauptprobleme ist die Vielzahl der von den Systemen erzeugten Alarme. Häufig werden tausende von Alarmen pro Tag erzeugt, von denen 99% Fehlalarme sind. Aufgrund dieser Alarmflut ist es schwierig die Alarme zu identifizieren, die tatsächlich Sicherheitsverletzungen anzeigen. Dadurch wird der Nutzen von IDS infrage gestellt.

Unter der Vorraussetzung einer korrekten Spezifikation von Signaturen, können Fehlalarme durch Missbrauchserkennungssysteme ausgeschlossen werden. Missbrauchserkennungssysteme erkennen genau die Muster, die in den Signaturen beschrieben sind. Die Ursache für hohe Fehlalarmraten ist dementsprechend auf der Ebene der Spezifikation der Signaturen zu suchen. Zum einen fehlt eine Systematik zur Betrachtung der relevanten Aspekte von Angriffssignaturen. Mit verschiedenen existierenden Sprachen zur Beschreibung von Signaturen sind unterschiedliche Mengen von Signaturen beschreibbar. Vielfach können Signaturen mit den zur Verfügung stehenden Sprachmitteln nicht exakt spezifiziert werden. Zum anderen führt die Komplexität von Angriffssignaturen zu einem Fehlerpotential bei ihrer Entwicklung. In existierenden Sprachen fehlen geeignete Mittel, um den Signaturentwickler bei der Beherrschung dieser Komplexität zu unterstützen.

Die steigende Leistungsfähigkeit von IT-Systemen führt zu einem weiteren Problem für die Missbrauchserkennung. Der damit einhergehende Anstieg des Datenaufkommens führt existierende Missbrauchserkennungssysteme an die Grenzen ihrer Leistungsfähigkeit. Die aus der zunehmenden Komplexität der IT-Systeme resultierende Steigerung der Anzahl zu analysierender Angriffsmuster verschärft dieses Problem zusätzlich. Aktuelle Systeme zur Missbrauchserkennung setzen verschiedene Standardverfahren zur Analyse ein. Der Entwicklung und Untersuchung optimierter Analyseverfahren zur Missbrauchserkennung wurde bisher kaum Auf-

merksamkeit geschenkt. Dadurch wird eine zeitnahe Erkennung von Angriffen schwieriger. Es ist notwendig, effiziente Analyseverfahren zur Missbrauchserkennung zu entwickeln.

Zur Steigerung der Wirksamkeit von IDS müssen Missbrauchserkennungssysteme effizient korrekte Ergebnisse liefern. Insbesondere ist die Zahl der Fehlalarme zu reduzieren. Dazu ist zunächst eine systematische Betrachtung von Angriffsmustern erforderlich. Des Weiteren werden Werkzeuge zur Modellierung und Beschreibung von Signaturen benötigt, die eine effiziente und fehlerfreie Entwicklung von Signaturen erlauben. Aufgrund der resultierenden kürzeren Entwicklungszeiten für Signaturen wird es möglich, nach dem Auftreten bzw. bekannt werden neuer Sicherheitslücken kurzfristig entsprechende Signaturen zu erstellen. Dadurch können die betroffenen Systeme vor entsprechenden Angriffen geschützt werden, bis die Sicherheitslücken behoben werden. Effiziente Analyseverfahren erlauben es, einen Angriff zeitnah zu erkennen und geeignete Gegenmaßnahmen einzuleiten, beispielsweise indem Aktivitäten des Angreifers blockiert werden.

Im Kapitel 2 geben wir zunächst eine grundlegende Einführung in den Bereich IT-Sicherheit. Eine Einordnung des Gebiets Intrusion Detection wird vorgenommen. Verschiedene Verfahren zur Angriffserkennung werden vorgestellt und diskutiert. Vertiefend werden in Kapitel 3 Verfahren zur Missbrauchserkennung behandelt und aktuelle Herausforderungen an diese Technologie herausgearbeitet. Beispiele für ein konkretes IT-System und verschiedene Attacken, auf die in diesem Buch Bezug genommen wird, werden in Kapitel 4 vorgestellt.

In Kapitel 5 wird ein Modell für die Semantik von Angriffssignaturen entwickelt, das systematisch die semantischen Aspekte von Signaturen betrachtet. Es bildet die Grundlage für Vergleiche der Ausdrucksstärke von Signaturbeschreibungssprachen. Kapitel 6 führt einen Ansatz zur Modellierung von Signaturen ein, bevor in Kapitel 7 zwei Sprachen zur Beschreibung von Signaturen vorgestellt werden.

Im Kapitel 8 untersuchen wir Analyseverfahren zur Missbrauchserkennung. Es werden Optimierungsstrategien zur Steigerung der Analyseeffizienz entwickelt und experimentell evaluiert. Im abschließenden Kapitel 9 werden eine Zusammengefassung und ein Ausblick gegeben.

2 IT-Sicherheit und Intrusion Detection

Eine der wichtigsten Entwicklungstendenzen der letzten Jahre ist der rasche Vormarsch moderner Kommunikations- und Informationstechnologien in vielen gesellschaftlichen Bereichen. Insbesondere mit dem rasanten Wachstum des Internets findet eine zunehmende Verlagerung wirtschaftlicher und privater Werte auf leistungsfähige informationstechnische Systeme statt. Die somit immer stärker werdende Abhängigkeit vieler gesellschaftlicher Prozesse von IT-Systemen sowie deren zunehmende technologische Komplexität fördern jedoch auch ein stetig steigendes Bedrohungspotential, das diese Systeme gefährdet. Gleichzeitig wächst die Attraktivität der Systeme für gezielte Missbräuche. Dadurch gewinnen Aspekte der IT-Sicherheit[1] immer mehr an Bedeutung. Während in der Praxis bisher hauptsächlich präventive IT-Sicherheitsmechanismen eingesetzt wurden, zeigen die Entwicklungen, dass Sicherheitsziele nicht allein durch Prävention erreicht werden können. Vielmehr müssen präventive Verfahren um reaktive Mechanismen ergänzt werden. Voraussetzung jedes reaktiven Verfahrens ist die Erkennung von Sicherheitsvorfällen.

In diesem Kapitel führen wir grundlegende Begriffe der IT-Sicherheit ein und geben einen Überblick über existierende Sicherheitsmechanismen. Die Rolle der im Weiteren betrachteten Erkennungsmechanismen wird diskutiert und ihre Notwendigkeit motiviert.

2.1 IT-Sicherheit

Aufgabe der IT-Sicherheit ist der Schutz von informationstechnischen Werten und Gütern. Um den Begriff der IT-Sicherheit fassen zu können, wird typischerweise betrachtet, wie informationstechnische Güter kom-

[1] Im Englischsprachigen Raum werden zur Unterscheidung von intentionalen und nichtintentionalen Beeinträchtigungen von Systemen überwiegend die Begriffe Security und Safety gebraucht (vgl. [Die04]), die im Deutschen mit demselben Wort, nämlich Sicherheit, übersetzt werden. Wir betrachten Security-Aspekte von Systemen und der Begriff IT-Sicherheit wird ausschließlich in dieser Bedeutung gebraucht.

promittiert werden können bzw. welche Schutzziele verfolgt werden. Im Allgemein werden die folgenden vier Schutzziele unterschieden (vgl. [Wo+00, Eck02]):

- *Vertraulichkeit* - Schutz vor unautorisierter Kenntnisnahme von Informationen.
- *Integrität* - Schutz vor unautorisierter unbemerkter Modifikation von Informationen.
- *Verfügbarkeit* - Schutz vor unautorisierter Vorenthaltung von Informationen oder Ressourcen.
- *Zurechenbarkeit* - Verursacher von Aktionen und Ereignissen sind ermittelbar.

Für spezifische Dienste existieren verschiedene Konkretisierungen dieser Schutzziele. Beispielsweise werden Vertraulichkeitsziele bei Kommunikationsdiensten ausgehend von den zu schützenden Gegenständen (Kommunikationsinhalte vs. Kommunikationsumstände) in Vertraulichkeit und Verdecktheit (Kommunikationsinhalte) sowie Anonymität und Unbeobachtbarkeit (Kommunikationsumstände) unterschieden (vgl. [Wo+00]). Ziel der IT-Sicherheit ist die Erreichung von Schutzzielen trotz der Präsenz intelligenter Angreifer.

Bedrohungen, wie z. B. der Verlust der Vertraulichkeit, beschreiben Situationen oder Ereignisse, die die Sicherheit eines Systems potentiell beeinträchtigen. Unter *Verwundbarkeiten* von IT-Systemen werden Schwächen der Systeme verstanden, die ausgenutzt werden können, um IT-Sicherheitsverletzungen durchzuführen. Bedrohungen sind das Ergebnis der Ausnutzung einer oder mehrerer Verwundbarkeiten [Ba00]. Anleitungen, Prozeduren bzw. Programme zur gezielten Ausnutzung von Verwundbarkeiten werden als *Exploits* bezeichnet.

Die Kontrolle, Steuerung und Autorisierung von Zugriffen auf informationstechnische Ressourcen setzt die Existenz einer *Sicherheitspolitik* voraus, die eine Menge von Regeln enthält, die festlegen was erlaubt und was verboten ist. Unter *(IT-)Sicherheitsverletzungen*, *Angriffen*, *Attacken* bzw. *Einbrüchen (Intrusions)* werden alle Aktionen oder Ereignisse verstanden, die den Regeln der Sicherheitspolitik zuwiderlaufen. Diese vier Begriffe werden in diesem Buch synonym verwendet.

2.2 Sicherheitsmechanismen

Zur Durchsetzung der Schutzziele werden IT-Systeme mit Sicherheits- bzw. Schutzmechanismen versehen. Eine grobe Unterteilung der Sicherheitsmechanismen unterscheidet zwischen

- präventiven und
- reaktiven Verfahren.

Präventive Mechanismen realisieren Maßnahmen, die eine Beeinträchtigung von informationstechnischen Ressourcen verhindern. Verschiedene präventive Verfahren zur Durchsetzung der Schutzziele sind in Tabelle 2-1 dargestellt. Präventive Mechanismen zur Gewährleistung der Vertraulichkeit von Informationen sind z. B. Verschlüsselungsverfahren, Zugriffskontrollverfahren oder Zugangskontrollen wie Firewalls (vgl. [Eck02, Sta03]). Angemerkt sei, dass Verfahren zur Prüfung der Integrität von Informationen, z. B. Digitale Signaturen, entsprechend dem verbreiteten Integritätsbegriff (s. o.) präventive Verfahren sind. Ihr Ziel ist es, zu *verhindern*, dass Informationen *unbemerkt* unautorisiert modifiziert werden.

Tabelle 2-1. Beispiele für präventive Sicherheitsmechanismen

Schutzziele	Präventive Mechanismen
Vertraulichkeit	Verschlüsselungsverfahren, Zugriffskontrollverfahren, Zugangskontrollen (Firewalls)
Integrität	Zugriffs- und Zugangskontrollverfahren, Digitale Signaturen
Verfügbarkeit	Zugriffs- und Zugangskontrollen, Redundante Auslegung von Ressourcen
Zurechenbarkeit	Digitale Signaturen, Protokollierung sicherheitsrelevanter Aktivitäten

Zum Schutz von IT-Systemen wurden bisher hauptsächlich präventive Verfahren verwendet. Der rapide Zuwachs an Sicherheitsvorfällen macht jedoch deutlich, dass durch präventive Sicherheitsmechanismen allein nur ein gewisses Maß an Schutz geboten werden kann. Trotz verstärkter Anwendung dieser Verfahren war in den letzten Jahren ein jährlicher Anstieg der beim US-amerikanischen CERT/CC (Computer Emergence Response Team / Coordination Center) gemeldeten Vorfälle um mehr als 50% zu beobachten (vgl. Abb. 2-1, [CE05])[1]. Präventive Mechanismen können

[1] Das CERT/CC bemerkt zu diesen Statistiken: „Given the widespread use of automated attack tools, attacks against Internet-connected systems have become so commonplace that counts of the number of incidents reported provide little information with regard to assessing the scope and impact of attacks. Therefore,

keinen Schutz vor missbräuchlichen Aktionen von autorisierten Nutzern, so genannten Insidern (vgl. [So99]), bieten oder durch Verwundbarkeiten aufgrund von fehlerhaften Implementierungen oder Konfigurationen in Systemen umgangen werden (vgl. [Bü01]). Daher sind präventive Mechanismen um reaktive Verfahren zu ergänzen.

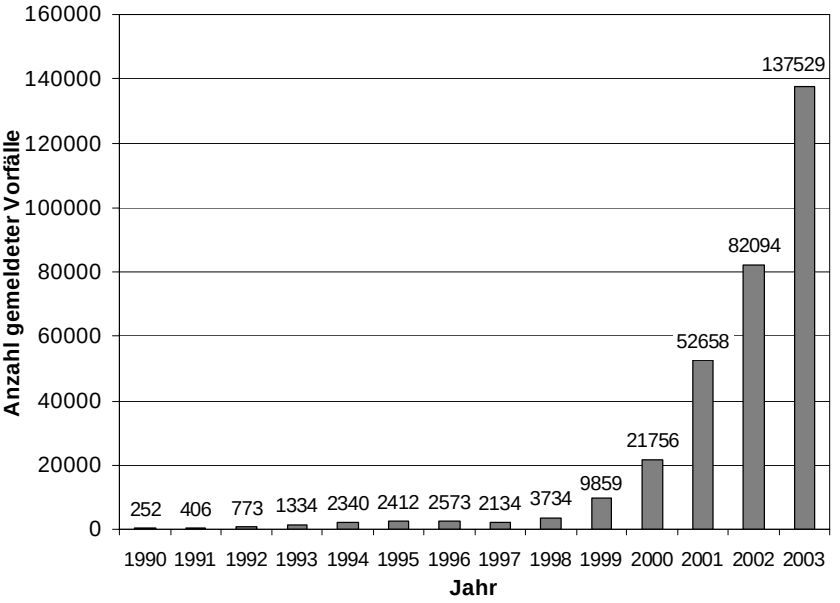


Abb. 2-1. Entwicklung beim CERT/CC gemeldeter Sicherheitsvorfälle

Ziel *reaktiver* Maßnahmen ist die Begrenzung und Beseitigung von verursachten Schäden sowie die Identifikation verantwortlicher Akteure. Sie sind Voraussetzung für eine Bestrafung von Verantwortlichen oder der Geltendmachung von Schadensersatzansprüchen. Unter Umständen werden bei Ihrer Umsetzung Abschreckungseffekte erreicht, die zusätzlich präventiv wirken [So99, Go99]. Voraussetzung für reaktive Maßnahmen ist eine zuverlässige Erkennung von Sicherheitsverletzungen. Zur automatischen Erkennung von Sicherheitsverletzungen werden *Intrusion-Detection-Systeme (IDS)* [Ba00, Mc01] verwendet. Um die Möglichkeiten der Systeme zur automatischen Reaktion auf erkannte Attacken zu unterstrei-

as of 2004, we will no longer publish the number of incidents reported. Instead, we will be working with others in the community to develop and report on more meaningful metrics,...“ [CE05].

chen, wird auch von *Intrusion-Detection-and-Response-Systemen (IDRS)* gesprochen[1].

2.3 Intrusion-Detection-Systeme

Ziel des Einsatzes von Intrusion-Detection-Systemen ist eine möglichst frühzeitige Erkennung von Angriffen, um den Schaden zu minimieren und Angreifer identifizieren zu können. Darüber hinaus erlauben IDS das Sammeln von Informationen über neue Angriffstechniken, die zur Verbesserungen präventiver Maßnahmen genutzt werden können. Dazu analysieren IDS Daten über Abläufe und Zustände von IT-Systemen. Im Folgenden wird der allgemeine Aufbau von IDS beschrieben.

Die *Defense Advanced Research Projects Agency (DARPA)* initiierte ein Projekt, in dem die Kooperation von und Kommunikation zwischen verschiedenen IDS bzw. IDRS ermöglicht werden sollte [Tu99]. Dadurch sollte auch eine Wiederverwendung von IDS-Komponenten erreicht werden. Ergebnis der Standardisierungsbemühungen ist das *Common Intrusion Detection Framework (CIDF)* [Ka+98], das u. a. mögliche Architekturen von IDS beschreibt. Das CIDF sieht vier Arten von IDS-Komponenten vor (vgl. Abb. 2-2):

- *Ereigniskomponenten* stellen Informationen über das zu schützende System bereit. Systemfunktionen zur Protokollierung sicherheitsrelevanter Aktivitäten sind Beispiele für Ereigniskomponenten.
- *Analysekomponenten* realisieren die eigentliche Erkennung von Angriffen und analysieren dazu die von den Ereigniskomponenten protokollierten Informationen.
- *Datenbankkomponenten* speichern weitere zur Analyse erforderliche Informationen sowie Zwischenergebnisse.
- *Reaktionskomponenten* führen auf Veranlassung durch andere Komponenten Gegenmaßnahmen durch.

Abb. 2-2 veranschaulicht exemplarisch eine mögliche Anordnung verschiedener Komponenten sowie den Informationsaustausch zwischen ihnen.

[1] In letzter Zeit sind viele kommerzielle Anbieter dazu übergegangen ihre IDS- bzw. IDRS-Produkte als *Intrusion-Prevention-Systeme (IPS)* zu vermarkten. In den meisten Fällen handelt es sich dabei um *reaktive* IDRS. Vereinzelt werden auch *präventive* Zugangs- bzw. Zugriffskontrollsysteme (z.B. Firewalls) mit diesem Schlagwort beworben. Wir verwenden diesen irreführenden Begriff nicht.

Inspiriert durch die CIDF-Aktivität wurden zwischenzeitlich verschiedene Standardisierungsbemühungen der *Intrusion Detection Working Group (IDWG)* der *Internet Engineering Task Force (IETF)* initiiert [Tu99, Be01], deren Ziel es ist, Formate und Protokolle für den Informationsaustausch zu entwickeln. Nachfolgend werden die Funktionen der verschiedenen CIDF-Komponenten sowie verwendete Verfahren diskutiert.

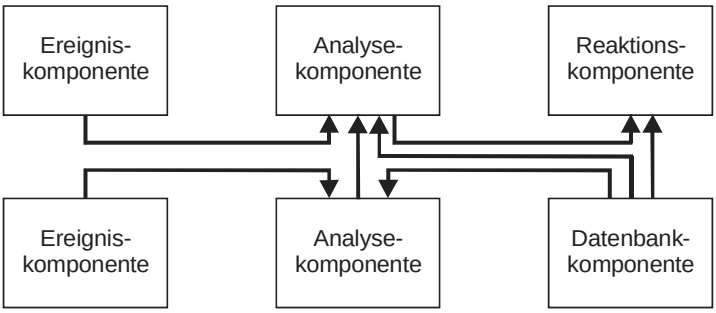


Abb. 2-2. Informationsaustausch verschiedener CIDF-Komponenten

2.3.1 Ereigniskomponenten und Audit

Voraussetzung für eine automatische Erkennung von Sicherheitsverletzungen ist die Aufzeichnung von Informationen über sicherheitsrelevante Abläufe oder Zustände des zu schützenden IT-Systems. Im Zusammenhang mit diesen Verfahren wird der Begriff *Audit* mit verschiedenen Bedeutungen verwendet. Verschiedene Autorengruppen [Pri97, So99, Bi03] fassen unter diesem Begriff Verfahren

- zur Protokollierung,
- zur Analyse oder
- zur Protokollierung und Analyse

zusammen. Wir schließen uns der ersten Gruppe an und verwenden den Begriff Audit in diesem Buch allein für Verfahren zur Protokollierung von sicherheitsrelevanten Abläufen oder Zuständen von IT-Systemen.

Entscheidend für die Möglichkeiten und die Qualität der Erkennung von Sicherheitsverletzungen ist der Informationsgehalt der erhobenen Audit-Daten. Andererseits muss die Menge gesammelter Informationen handhabbar bleiben. Das Problem, genügend aber nicht zuviel Audit-Daten zu sammeln, wird etwas humorvoll beschrieben durch „You either die of thirst, or you are allowed a drink from a fire hose ...“ [Ax98]. Aus diesem

Grund verfügen Audit-Funktionen von IT-Systemen typischerweise über umfangreiche Konfigurationsmöglichkeiten, die es dem Systemadministrator ermöglichen die Protokollierung an die Erfordernisse des Einsatzumgebung des Systems anzupassen. Prinzipiell können zwei Arten von Audit unterschieden werden: zustandsbasiertes Audit und transitions- bzw. aktionsbasiertes Audit (vgl. [Bi03]).

Beim *zustandsbasierten Audit* werden, typischerweise regelmäßig, Informationen über den Zustand bzw. über Teilzustände des IT-Systems, z. B. die Auslastung bestimmter Ressourcen, aufgezeichnet. Durch eine spätere Analyse dieser Informationen werden die Zustände des IT-Systems als sicherheitskonform oder sicherheitsverletzend klassifiziert. Problematisch bei diesem Ansatz ist, dass eine periodische vollständige Aufzeichnung des Zustandes eines IT-Systems zu großen schwer handhabbaren Datenmengen führt. Aus diesem Grund werden in der Praxis nur Informationen über kritische oder signifikante Teilzustände aufgezeichnet. Ein Beispiel für diese Vorgehensweise ist die regelmäßige Protokollierung der Auslastung eines Netzwerkes, eines Prozessors oder eines Pufferspeichers.

Ein *transitions- bzw. aktionsbasierter Audit-Mechanismus* zeichnet Informationen über sicherheitsrelevante Aktivitäten im IT-System auf. Typischerweise umfassen diese Informationen

- wer,
- wann,
- welche Aktion,
- wie (erfolgreich bzw. erfolglos)

ausgeführt hat sowie aktionsspezifische Zusatzinformationen wie z.B. Parameter (vgl. [So99]). Durch Analyse dieser Audit-Daten wird später entschieden, ob durch die protokollierten Aktionen ein sicherheitskritischer Systemzustand erreicht wurde. Beispiele für (hauptsächlich) transitionsbasiertes Audit sind die Audit-Funktionen der Betriebssysteme Solaris [Sun02] und Windows NT / 2000 / XP [Ju+98] aber auch Netzmonitore wie TCPDUMP [Ja+89]. Problematisch am transitionsbasierten Audit ist, dass auf der Grundlage der Daten nicht in jedem Fall zuverlässig entschieden werden kann, ob ein sicherheitskritischer Zustand erreicht wurde. Dies gilt insbesondere dann, wenn sich das IT-System bereits zu Beginn der Protokollierung in einem kritischen Zustand befindet [Bi03]. Aus diesem Grund wird häufig eine Kombination aus zustands- und transitionsbasiertem Audit realisiert.

Die protokollierten Informationen werden typischerweise als *Audit-Records* in einer zeitlich geordneten Sequenz organisiert, die als *Audit-Trail*

bezeichnet wird. Unsere weiteren Betrachtungen gehen von folgenden selten explizit genannten jedoch üblichen Annahmen hinsichtlich der Audit-Funktion des IT-Systems aus:

- Zu Beginn des transitionsbasierten Audits befand sich das zu schützende System in einem zur Sicherheitspolitik konformen Zustand.
- Die Audit-Trail enthält durch zustands- und transitionsbasiertes Audit generierte Audit-Records.
- Die Audit-Records in der Audit-Trail sind zeitlich geordnet.
- Die Integrität der Audit-Daten ist sichergestellt.

2.3.2 Analyse- und Datenbankkomponenten

Analysekomponenten führen die eigentliche Erkennung von Sicherheitsverletzungen durch. Je nach verwendeter Analysetechnik werden dazu zusätzliche Informationen verwendet, die in den Datenbankkomponenten organisiert werden. Mittels entsprechender Verfahren werden die in den Audit-Daten dokumentierten Beobachtungen analysiert, um den Zustand des überwachten Systems als sicherheitskonform oder sicherheitsverletzend zu klassifizieren. Zur Diskussion und Bewertung dieser binären Klassifikationsverfahren werden typischerweise vier Werte herangezogen:

- *Wahr-Positive (True Positives)*: Beobachtungen, die korrekt als positiv (sicherheitsverletzend) klassifiziert wurden.
- *Wahr-Negative (True Negatives)*: Beobachtungen, die korrekt als negativ (sicherheitskonform) klassifiziert wurden.
- *Falsch-Positive (False Positives)*: Beobachtungen, die inkorrekt als positiv (sicherheitsverletzend) klassifiziert wurden.
- *Falsch-Negative (False Negatives)*: Beobachtungen, die inkorrekt als negativ (sicherheitskonform) klassifiziert wurden.

Insbesondere werden die Häufigkeiten bzw. Raten inkorrekter Klassifikationsergebnisse betrachtet. Falsch-Positive von IDS beschreiben Fehlalarme, also sicherheitskonforme Zustände, die als Sicherheitsverletzungen angezeigt wurden. Falsch-Negative stellen Sicherheitsverletzungen dar, die nicht als solche erkannt wurden.

Es existieren zwei allgemeine Analysetechniken zur Einbruchserkennung, die sich sowohl in der Vorgehensweise als auch durch die verwendeten Referenzinformationen unterscheiden:

- Anomalieerkennung und
- Missbrauchserkennung bzw. Signaturanalyse.

Anomalieerkennung

Der *Anomalieerkennung (Anomaly Detection)* liegt die Annahme zugrunde, dass Abweichungen von Normen, so genannte Anomalien, auf Sicherheitsverletzungen hindeuten [Ba00]. Neben der Konformität zur Sicherheitspolitik wird hier eine zweite Klassifikationsebene eingeführt (vgl. Abb. 2-3), die zwischen normalen und anomalen Abläufen und Zuständen unterscheidet. Darüber hinaus wird unterstellt, dass weder normale sicherheitsverletzende noch anomale sicherheitskonforme Aktivitäten existieren. Dementsprechend werden durch Audit-Daten dokumentierte Aktionen und Zustände als normal bzw. anomal klassifiziert und auf sicherheitskonforme respektive sicherheitsverletzende Aktivitäten geschlossen.

Bei der Anomalieerkennung wird angenommen, dass ein normales Verhalten existiert und geeignet mess- oder beschreibbar ist. Diese Analysetechnik verwendet Referenzinformationen über normales Verhalten und vergleicht diese mit den in Audit-Daten dokumentierten Ereignissen. Abweichungen werden als Sicherheitsverletzungen angezeigt. Abb. 2-3 veranschaulicht die Grundidee der Anomalieerkennung anhand der zugrunde liegenden Klassifikationsebenen.

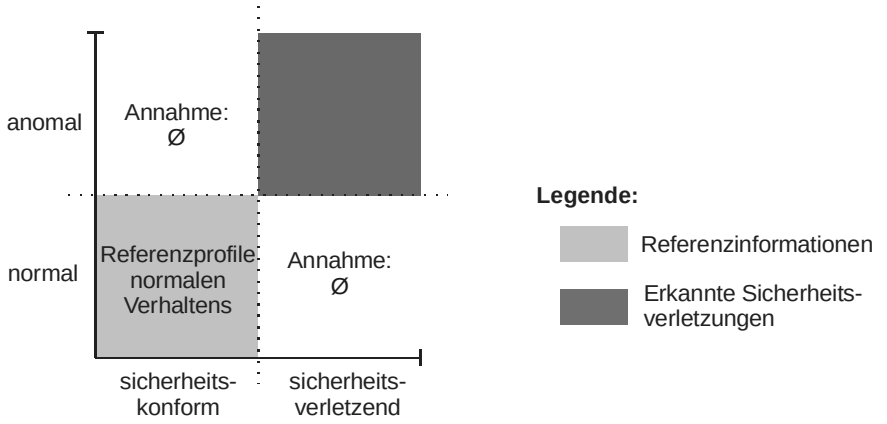


Abb. 2-3. Klassifikationsebenen der lern- bzw. messbasierten Anomalieerkennung (nach [So99])

Zur Erzeugung der erforderlichen Referenzinformationen, die normales Verhalten repräsentieren, werden zwei grundlegende Ansätze unterschieden:

- das Erlernen bzw. Messen normalen Verhaltens und
- die Spezifikation normalen Verhaltens.

Zur Umsetzung des ersten Ansatzes ist eine Lernphase des Systems erforderlich, in der Referenzprofile normalen Verhaltens erhoben werden. Da sich insbesondere nutzerbezogenes Verhalten über die Zeit verändern kann, wiederholen verschiedene IDS, die diesen Ansatz realisieren, regelmäßig oder kontinuierlich diese Phase. Problematisch hierbei ist, dass die Sicherheitskonformität des gemessenen Verhaltens während der Lernphase durch andere Mechanismen sicherzustellen ist, um auszuschließen, dass Sicherheitsverletzungen in die Referenzprofile normalen Verhaltens einfließen. Techniken, die zur Umsetzung dieses Ansatzes zum Einsatz kommen, umfassen statistische Methoden [Ja+91], neuronale Netze [De+92], genetische Algorithmen [Mé96], Support Vector Machines [La+04], Methoden des Data Minings [Le+99] und Analogien zum menschlichen Immunsystem [Fo+96].

Der zweite auch als *spezifikationsbasierte Anomalieerkennung* bezeichnete Ansatz basiert auf der expliziten häufig formalen Spezifikation normalen Verhaltens. Dieser Ansatz wird oft auch ohne Bezug auf die Klassifikationsebene normal-anomal realisiert, indem explizit sicherheitskonforme Aktionen und Zustände spezifiziert werden.[1]

Die spezifikationsbasierte (Anomalie-)Erkennung eignet sich insbesondere zum Schutz von Systemen für die bereits Spezifikationen des möglichen Verhaltens bzw. möglicher Aktionen und Zustände vorliegen, wie dies bei Kommunikationsprotokollen häufig der Fall ist. Beispiele für IDS, die diesen Ansatz realisieren sind *DPEM (Distributed Program Execution Monitor)* [Ko+94, Ko96] und *ANIDA* (*Aachen Network Intrusion Detection Architecture*) [Bü+99, Bü01]. Die verwendeten Techniken erinnern an die Compiler-Technik: die verwendeten Spezifikationen repräsentieren eine Grammatik und ein entsprechender Parser überprüft die Konformität beobachteter Aktionen und Zustände zu den Spezifikationen. Abb. 2-4 veranschaulicht die Grundidee der spezifikationsbasierten (Anomalie-)Erkennung: sämtliche sicherheitskonformen Aktivitäten sind durch die Refe-

[1] Aus diesem Grund betrachten verschiedene Autoren die spezifikationsbasierte Erkennung nicht als Art der Anomalieerkennung sondern als eigenen dritten Analyseansatz [Bi03]. Wieder andere Autoren unterscheiden nur Anomalieerkennung und politikbasierte Erkennung, die ausgehend von der Art der zugrunde liegenden Sicherheitspolitik in Default-Permit- (Missbrauchserkennung) und Default-Deny- (spezifikationsbasierte Erkennung) Verfahren unterteilt werden [Ax98].

renzdaten spezifiziert und jegliche Abweichung wird als Sicherheitsverletzung klassifiziert.

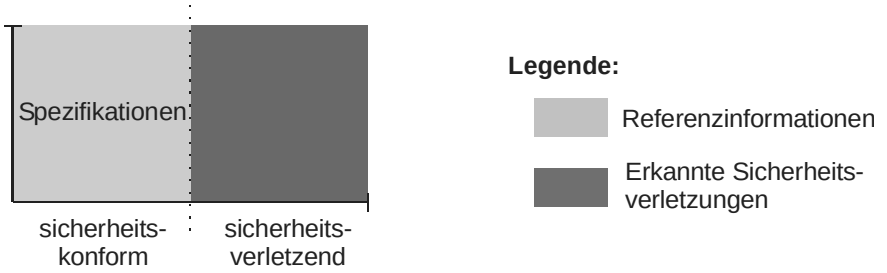


Abb. 2-4. Grundidee der spezifikationsbasierten (Anomalie-)Erkennung

Missbrauchserkennung

Bei der *Missbrauchserkennung (Misuse Detection)*, die auch als *Signaturanalyse (Signature Analysis)* bezeichnet wird, suchen die Analysekomponenten nach konkreten Sicherheitsverletzungen. Dazu verwenden sie definierte Angriffsmuster, die als Signaturen bezeichnet werden. Während der Analyse werden die Audit-Daten auf Übereinstimmung mit den spezifizierten Signaturen untersucht und die Übereinstimmungen als Sicherheitsverletzung angezeigt (vgl. Abb. 2-5). Voraussetzung für die Erkennung eines Angriffs ist das Vorliegen einer entsprechenden Signatur. Dementsprechend können mit diesem Ansatz nur bekannte Angriffsarten erkannt werden.

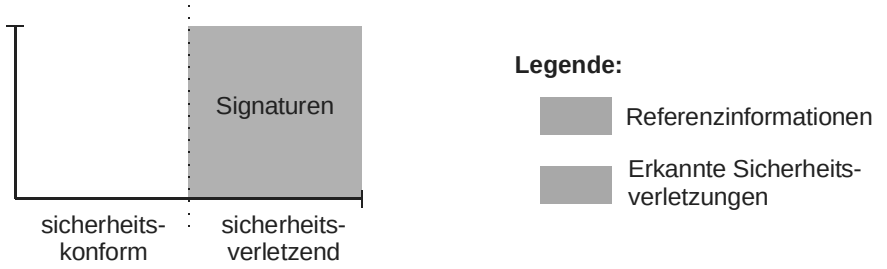


Abb. 2-5. Grundidee der Missbrauchserkennung

Zur Umsetzung der Missbrauchserkennung werden verschiedene Techniken wie z. B. Expertensysteme eingesetzt. *EMERALD (Event Monitoring Enabling Responses to Anomalous Live Disturbances)* [Po+97], *CMDS (Computer Misuse Detection System)* [Pro94], *AID (Adaptive Intrusion Detection system)* [So+96] sind Beispiele für IDS, die diesem Ansatz folgen. Andere IDS, wie z. B. *STAT (State Transitions Analysis Technique)*

[Vi+00] oder *IDIOT (Intrusion Detection In Our Time)* [Ku95], verwenden dedizierte Verfahren zur Überprüfung von Zustandsänderungen. Auf diese Techniken wird im weiteren Verlauf genauer eingegangen.

Vergleich und Bewertung der Analyseansätze

Der Hauptvorteil der lern- bzw. messbasierten Anomalieerkennung besteht darin, keine expliziten Festlegungen hinsichtlich sicherheitskonformer oder sicherheitsverletzender Aktionen und Zustände treffen zu müssen. Im Unterschied zur Missbrauchserkennung ist es mit diesem Ansatz prinzipiell möglich, neue bzw. unbekannte Sicherheitsverletzungen zu erkennen. Diese Vorteile werden jedoch durch eine Reihe von Nachteilen relativiert. Dies ist zum einen die grundsätzliche Schwierigkeit, geeignete verhaltensspezifische Merkmale zu finden und zum anderen das Problem der Veränderbarkeit des (Nutzer-)Verhaltens über längere Zeiträume. Die gesamte Vorgehensweise weist oftmals eine signifikante inhärente Unschärfe auf, was in der praktischen Anwendung häufig zu unakzeptablen Fehlalarmraten führt (vgl. Abb. 2-6). Ein weiteres Problem der Anomalieerkennung besteht in der Notwendigkeit, die Sicherheitskonformität des Systems während der Lernphase durch andere Mechanismen (z. B. unter Verwendung von Missbrauchserkennungssystemen) sicherzustellen. Außerdem basiert der Ansatz (lediglich) auf der Hypothese, dass sich Sicherheitsverletzungen in anomalem Verhalten manifestieren. Aus diesen Gründen besitzen existierende Anomalieerkennungssysteme eine nicht zu vernachlässigende Falsch-Negativ-Rate (vgl. Abb. 2-6). Darüber hinaus zeigen die von Anomalieerkennungssystemen gelieferten Ergebnisse zunächst nur Anomalien im System an, von denen nicht ohne weiteres auf konkrete stattgefundene Sicherheitsverletzungen geschlossen werden kann. Dadurch sind vor der Einleitung von Gegenmaßnahmen weitere Untersuchungen erforderlich. Das Leistungsvermögen existierender Anomalieerkennungssysteme wird durch Abb. 2-6 veranschaulicht. Die Größe der dargestellten Flächen hat dabei keinen quantitativen sondern nur existentiellen Charakter.

Die lern- bzw. messbasierten Anomalieerkennung liefert unscharfe Analyseergebnisse (z. B. Abweichungen eines numerischen Anomalieindikators). Im Unterschied dazu zeigt die spezifikationsbasierte (Anomalie-)Erkennung typischerweise an, gegen welchen Teil der Spezifikationen verstoßen wurde. Allerdings ist es auch hier nicht in jedem Falle möglich, konkret die aufgetretenen sicherheitsverletzenden Aktionen oder Zustände zu benennen. Technisch ist dieses Problem teilweise vergleichbar mit der Schwierigkeit für Compiler, geeignete Fehlermeldungen zu erzeugen. Hauptproblem bei diesem Ansatz ist jedoch die Notwendigkeit einer fehlerfreien vollständigen Spezifikation aller normalen bzw. sicherheitskon-

formen Aktivitäten, da dies für komplexere Systeme nicht oder nur mit sehr großem Aufwand möglich ist. Deshalb treten auch bei spezifikationsbasierten Verfahren sowohl Fehlalarme als auch Falsch-Negative auf (vgl. Abb. 2-7).

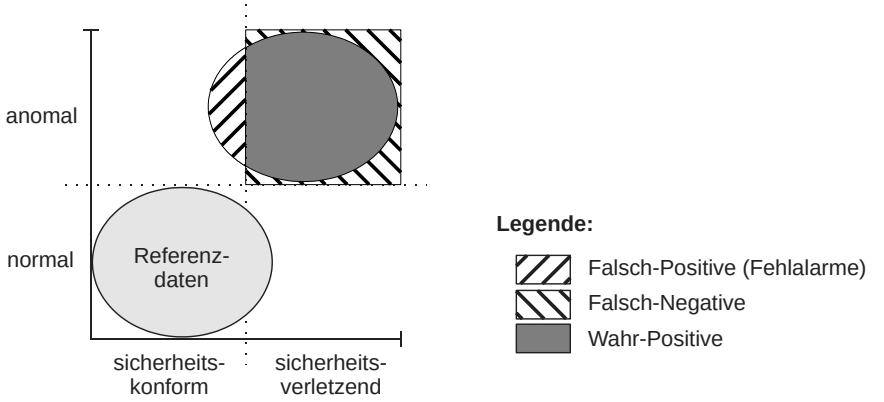


Abb. 2-6. Leistungsvermögen von Anomalieerkennungssystemen

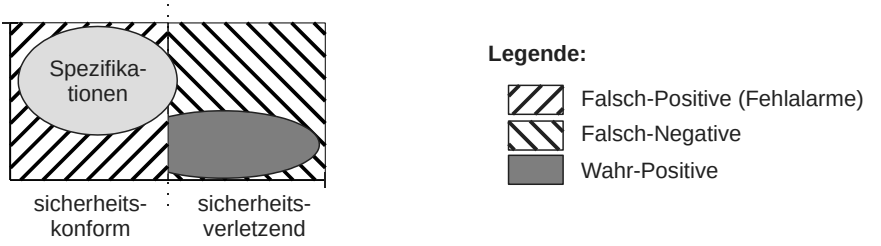


Abb. 2-7. Leistungsvermögen spezifikationsbasierter Anomalieerkennungssysteme

Ein Hauptvorteil der Missbrauchserkennung liegt in den scharfen und nachvollziehbaren Ergebnissen, durch die konkret aufgetretene Sicherheitsverletzungen beschrieben werden können. Das Leistungsvermögen von Missbrauchserkennungssystemen steht und fällt jedoch mit dem Umfang und der Qualität der verwendeten Signaturbeschreibungen. Durch die kontinuierliche Weiterentwicklung von IT-Systemen, z. B. durch so genannte Patches aber auch die regelmäßige Entdeckung neuer Verwundbarkeiten, besteht hier ein permanentes Unvollständigkeitsproblem. Aufgrund der Komplexität der Signaturentwicklung und daraus resultierender Entwicklungszeiten kommen neue Signaturen typischerweise erst einige Zeit nach dem Auftauchen entsprechender Exploits zum Einsatz [Li+02]. Aus diesen Gründen weist in der Praxis auch die Missbrauchserkennung Falsch-Negative auf (vgl. Abb. 2-8).

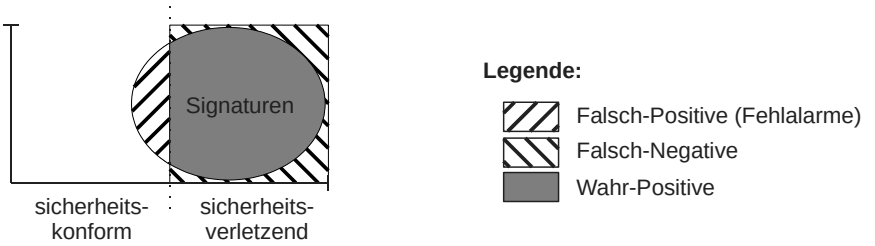


Abb. 2-8. Leistungsvermögen von Missbrauchserkennungssystemen

Da in der Praxis jeder der drei Analyseansätze jeweils nur für das Finden bestimmter Teilmengen von Sicherheitsverletzungen geeignet ist, wird zunehmend eine Kombination der Ansätze angewandt. Wie die Veranschaulichung in Abb. 2-9 suggeriert, kann durch eine Kombination der Ansätze eine Verringerung der Falsch-Negativ-Rate erreicht werden. Hingegen scheint eine Minimierung der Falsch-Positiv-Rate (Fehlalarmrate) durch eine einfache Kombination der Ansätze nicht erreichbar.

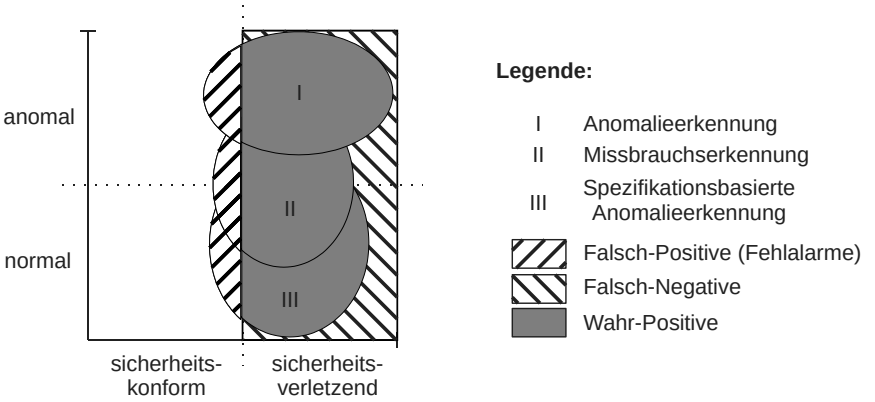


Abb. 2-9. Kombiniertes Leistungsvermögen der Analyseansätze

2.3.3 Reaktionskomponenten

Nachdem Sicherheitsverletzungen durch Analysekomponenten erkannt wurden, werden die Reaktionskomponenten des IDS veranlasst entsprechende Reaktionen durchzuführen. Grundlegend wird zwischen *passiven* und *aktiven* Reaktionen unterschieden. Passive Reaktionen liefern Informationen an den Nutzer des IDS und überlassen diesem die Ergreifung weiterer Maßnahmen. Aktive Reaktionen umfassen das automatische oder halbautomatische Auslösen von Aktionen. Möglich sind hierbei gezielte

Aktionen gegen den Angreifer, z. B. das Blockieren bestimmter Netzwerkdienste, die Benachrichtigung umgebender Systeme und die Sammlung zusätzlicher Informationen. Eine ausführliche Diskussion möglicher Gegenmaßnahmen findet sich in [Ba00].

2.4 Fazit

Mit der wachsenden Abhängigkeit unserer Gesellschaft von der Zuverlässigkeit informationstechnischer Systeme gewinnen Fragen der IT-Sicherheit an Bedeutung. Während bisher vorrangig präventive Sicherheitsmechanismen im Vordergrund standen, zeigt sich zunehmend, dass IT-Sicherheit nicht allein durch Prävention erreicht werden kann. Vielmehr stellt Prävention einen Grundpfeiler dar, neben dem ergänzend die reaktiven Aspekte der IT-Sicherheit stehen. Dem Nachweis von aufgetretenen IT-Sicherheitsverletzungen kommt dabei eine wachsende Bedeutung zu. Auf der Grundlage mittels Audit protokollierter Informationen über sicherheitsrelevante Zustände und Aktionen in IT-Systemen können IDS eingesetzt werden, um automatisch IT-Sicherheitsverletzungen zu erkennen und Reaktionen zu veranlassen. IDS setzen dazu verschiedene Verfahren ein, von denen die mess- bzw. lernbasierte und die spezifikationsbasierte Anomalieerkennung hinsichtlich der Erkennung unbekannter bzw. neuer Attacken eine Reihe von Vorteilen bieten. Diese werden jedoch durch die inhärente Unschärfe der von diesen Verfahren gelieferten Ergebnisse relativiert, so dass diese Auswertungsansätze nur ergänzend von Interesse sind. Entsprechend stellt die Missbrauchserkennung, die auf der Grundlage von definierten Angriffsmustern scharfe Ergebnisse liefert, ein unverzichtbares Basisanalyseverfahren dar.

3 Missbrauchserkennung

Missbrauchserkennungssysteme erkennen Sicherheitsverletzungen auf der Grundlage der bei der Durchführung der Attacke beobachteten und protokollierten Ereignisse. In Abhängigkeit vom Ablauf und damit vom erforderlichen Aufwand zu ihrer Erkennung können Attacken in zwei Klassen unterteilt werden. Sicherheitsverletzungen, die auf der Grundlage eines einzelnen Audit-Records erkannt werden können, werden als *Einzel-Schritt-Attacken* (*Single Step Attacks)* bezeichnet. Signaturen zur Erkennung derartiger Sicherheitsverletzungen beschreiben typischerweise charakteristische Byte-Sequenzen, deren ggf. kombiniertes Auftreten in einem Audit-Record auf eine Sicherheitsverletzung hindeutet. *Mehr-Schritt-Attacken (Multi Step Attacks)* hingegen bezeichnen Sicherheitsverletzungen zu deren Erkennung mehrere Audit-Records in Zusammenhang gebracht und auf charakteristische Merkmale untersucht werden müssen. *Einzel-Schritt-IDS (Single Step IDS)* bezeichnen Systeme, die nur in der Lage sind Einzel-Schritt-Attacken zu erkennen. Die Analyseverfahren dieser Systeme basieren im Wesentlichen auf String-Vergleichen. Die Beschränkungen dieser Systeme überwinden *Mehr-Schritt-IDS (Multi Step IDS)*, die zusätzlich in der Lage sind, Mehr-Schritt-Attacken zu erkennen und dazu wesentlich komplexere und aufwendigere Analyseverfahren verwenden.

Ausgehend von der Art der Audit-Daten, die ein Missbrauchserkennungssystem analysiert, werden *netzbasierte* und *hostbasierte* IDS unterschieden. Netzbasierte IDS analysieren protokollierte Netzwerkpakete, während hostbasierte Systeme Ereignisse auf Betriebssystem- oder Anwendungsebene, z. B. Systemrufe, verarbeiten. Hostbasierte Audit-Daten sind im Hinblick auf eine Einbruchserkennung häufig qualitativ hochwertiger und geben stattgefundene Ereignisse detaillierter wieder. Zur Verwendung dieser Daten ist jedoch die Konfiguration jedes einzelnen Endsystems in der geschützten Umgebung erforderlich, deren Leistungsfähigkeit durch die laufende Protokollierung beeinträchtigt werden kann. Stattdessen können netzbasierte IDS häufig weniger aufwendig an zentraler Stelle im geschützten Netz installiert werden, ohne die Leistungsfähigkeit der Endsysteme zu beeinträchtigen. Sie zeichnen zentral den protokollierten Datenverkehr auf.

Aufgrund vergleichsweise guter Erkennungsgenauigkeit (vgl. Abschn. 2.3.2), einfach zu realisierender Analyseverfahren sowie ihrer einfachen Installation und Konfiguration sind netzwerkbasierte Missbrauchserkennungssysteme zur Erkennung von Einzel-Schritt-Attacken derzeit der populärste und am weitesten verbreitete Ansatz. Ein Beispiel für ein solches System ist das IDS SNORT [Roe99].

Da Einzel-Schritt-IDS auf die Erkennung von Einzel-Schritt-Attacken beschränkt sind, sind mit ihrem Einsatz zwangsläufig Falsch-Negative verbunden. Sie sind nicht in der Lage Sicherheitsverletzungen zu erkennen, die aus mehreren charakteristischen Aktionen bzw. Ereignissen bestehen. Darüber hinaus können auch Attacken, die aus einzelnen Aktionen bestehen, nicht erkannt werden, wenn die charakteristischen Merkmale der Aktionen in mehr als einem Audit-Record dokumentiert werden, die zur Erkennung in Zusammenhang gebracht werden müssen. Einzel-Schritt-IDS verwenden in einem solchen Fall entweder unterspezifizierte Signaturen, die zu Falsch-Positiven (Fehlalarmen) führen, oder verzichten auf eine Erkennung dieser Attacken und nehmen damit Falsch-Negative in Kauf. Aufgrund dieser Situation ist eine intensivere Auseinandersetzung mit Mehr-Schritt-Attacken und -IDS erforderlich.

Wir widmen uns in diesem Buch der Modellierung, Beschreibung und Erkennung von Mehr-Schritt-Attacken. Abgesehen von Beispielen erfolgen die Betrachtungen unabhängig von der Art der Audit-Daten. Im Folgenden werden zunächst anhand eines einfachen operationalen Modells eines Missbrauchserkennungssystems der Ablauf der Erkennung skizziert, relevante Begriffe eingeführt und verwendete Informationsarten diskutiert. Anschließend werden gegenwärtig vorherrschende Probleme beim Einsatz von Missbrauchserkennungssystemen aufgezeigt und Herausforderungen an die Missbrauchserkennung dargestellt. Abschließend werden reale Beispiele für eine Systemumgebung und Attacken vorgestellt, die zur Diskussion im weiteren Verlauf herangezogen werden.

3.1 Systemmodell und Informationsarten

Attacken repräsentieren eine bestimmte Menge an Aktionen bzw. Ereignissen in einem IT-System, die der Sicherheitspolitik zuwiderlaufen (vgl. Abschn. 2.1). Dies kann beispielsweise eine bestimmte Folge von Systemrufen oder Netzwerkpaketen sein. Durch die Audit-Funktion wird die Ausführung sicherheitsrelevanter Aktionen bzw. die Erreichung sicherheitskritischer Zustände in Audit-Records dokumentiert, auf deren Grundlage die spätere Analyse durchgeführt werden kann.

Missbrauchserkennungssysteme versuchen Sequenzen von Audit-Records, die mit bekannten Sicherheitsverletzungen korrespondieren, zu erkennen. Hierbei wird davon ausgegangen, dass sich Sicherheitsverletzungen in den generierten Audit-Daten manifestieren, d. h. beobachtbar sind, und anhand dieser Datensätze erkannt werden können. Aus den vorliegenden Audit-Datensätzen werden Schlussfolgerungen auf die tatsächlich durchgeführten Aktionen gezogen, was die Notwendigkeit adäquater Audit-Funktionen für eine effektive Einbruchserkennung unterstreicht (vgl. Abschn. 2.3.1). Unzureichende Audit-Funktionen in vielen IT-Systemen erschweren häufig die Entwicklung von Angriffssignaturen und führen zu unnötigen Fehlalarmen (vgl. [Pri97, Ka01]).

Die bei der Durchführung von Sicherheitsverletzungen generierten Audit-Datensätze enthalten Spuren der Sicherheitsverletzungen. Die charakteristischen Spuren einer Attacke heißen *Manifestierung*. Als *Signatur* einer Attacke werden die Kriterien bzw. Muster bezeichnet, anhand derer die Manifestierung der Attacke in einem Audit-Datenstrom identifiziert werden kann.

Es ist möglich, dass mehrere Attacken eines Typs gleichzeitig und unabhängig voneinander fortschreiten, beispielsweise indem sie parallel von verschiedenen Angreifern durchgeführt werden. Deshalb ist es erforderlich, von *Instanzen einer Attacke* zu sprechen, deren Manifestierungen sich in der Ausprägung einzelner Merkmale, z. B. dem Benutzernamen, unterscheiden. Eine *Signaturinstanz* beschreibt die Menge der Kriterien, die eindeutig die Manifestierung einer Attackeninstanz im Audit-Datenstrom identifiziert.

Die Analysekomponente eines Missbrauchserkennungssystems durchsucht den eingehenden Audit-Datenstrom nach den in den Signaturen kodierten Mustern. Im Falle einer Übereinstimmung wird dies durch einen Alarm angezeigt. Gegebenenfalls werden Reaktionen, so genannte Responses, eingeleitet, mit denen z. B. Kommunikationsbeziehungen unterbrochen werden. Abb. 3-1 zeigt ein vereinfachtes operationales Modell der Missbrauchserkennung.

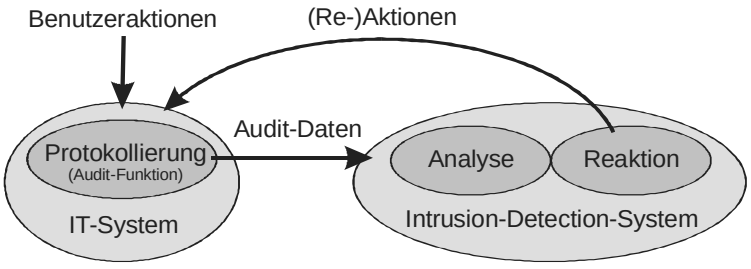


Abb. 3-1. Operationales Modell eines Missbrauchserkennungssystems

Im Zusammenhang mit der Missbrauchserkennung sind eine Reihe von Informationen relevant. Zur Identifikation von Manifestierungen bekannter Attacken in Audit-Daten ist zunächst die Ermittlung entsprechender Signaturen notwendig. Dieser Vorgang erfordert typischerweise eine detaillierte Untersuchung der Einzelaktionen bzw. Ereignisse einer Attacke. Des Weiteren sind entsprechende (Re-)Aktionen und Alarmmeldungen zu beschreiben, die ausgelöst werden sollen, wenn eine Attacke erkannt wurde.

Für die Darstellung dieser Informationen sind verschiedene Sprachen entwickelt worden (s. Abb. 3-2). In [Vig+00] werden sechs verschiedene Klassen so genannter *Attackensprachen (Attack Languages)* unterschieden. *Exploit-Sprachen* (*Exploit Languages)* dienen der Kodierung der Aktionen, die zur Durchführung einer Attacke erforderlich sind. Typischerweise sind dies Programmier- oder Skript-Sprachen wie C oder Perl. *Ereignissprachen (Event Languages)* werden verwendet, um die von einem IDS zu analysierenden Ereignisse, d. h. im Wesentlichen die Formate der Audit-Daten, zu beschreiben. Beispiele sind Datensatzspezifikationen für Solaris- und TCPDUMP-Audit-Daten [Sun02, Ja+89]. Signaturen werden unter Verwendung von *Erkennungssprachen (Detection Languages)* spezifiziert. *Reaktionssprachen (Response Languages)* beschreiben Aktionen, die im Fall einer erkannten Attacke als Gegenmaßnahme zu initiieren sind. Existierende IDS verwenden zu diesem Zweck häufig Bibliotheksfunktionen, die in Programmiersprachen wie C entwickelt wurden. *Reportsprachen (Report Languages)* beschreiben die Formate von Alarmmeldungen, die Informationen über erkannte Angriffe enthalten. Ein Beispiel hierfür ist das *Intrusion Detection Message Exchange Format (IDMEF)* [De+05], welches derzeit das Standardisierungsverfahren der IETF durchläuft. Derartige Alarmmeldungen bilden auch die Grundlage für Analysen auf höherer Ebene. Aus diesem Grund können Reportsprachen als spezielle Ereignissprachen betrachtet werden. Die Analyse von Alarmmeldungen dient beispielsweise der Erkennung von verteilten koordinierten Angriffen. Hierbei werden erkannte Angriffe hinsichtlich bestimmter Zusammenhänge untersucht, die unter Verwendung von *Korrelationssprachen (Correlation Languages)* formuliert werden. Korrelationssprachen stellen somit Erkennungssprachen auf höherer Abstraktionsebene dar.

Einige der genannten Sprachen, z. B. Ereignis- und Reportsprachen, werden hauptsächlich von den Entwicklern von IT-Systemen entworfen und verwendet. Erkennungssprachen adressieren hingegen eine größere Anwendergruppe. IDS werden zwar häufig mit einer Grundmenge von Signaturen ausgeliefert. Aufgrund der Vielfältigkeit heutiger IT-Umgebungen und durch das Auftreten neuer Attacken sind Anpassungen der Signaturen an die konkrete Einsatzumgebung sowie Erweiterungen der Signaturbasis erforderlich. Sicherheitsadministratoren stehen daher häufig vor der

Aufgabe, neue Signaturen zu entwickeln, die den Bedingungen und Sicherheitsrichtlinien der Umgebung entsprechen. Tabelle 3-1 fasst die im Zusammenhang mit der Missbrauchserkennung relevanten Informationen, die verwendeten Sprachen sowie die jeweiligen Anwendergruppen zusammen.

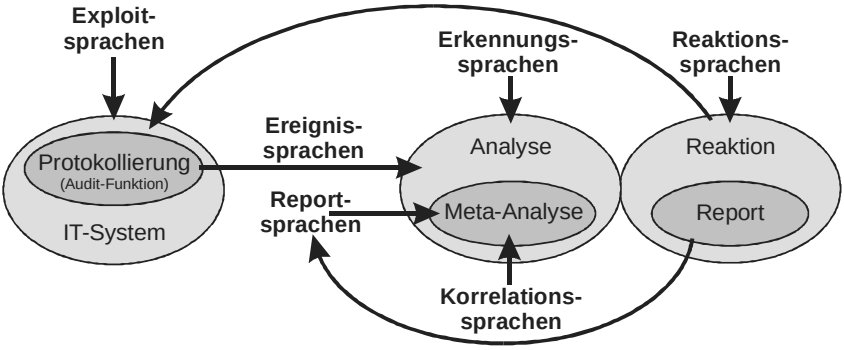


Abb. 3-2. Attackensprachen in der Missbrauchserkennung

Tabelle 3-1. Attackensprachen und deren Anwender

Art der Information	(Re-) Aktionen	Audit-Daten	Analyse-Kriterien
Verwendete Sprachen	Exploit- und Reaktionssprachen	Ereignis- und Reportsprachen	Erkennungs- und Korrelationssprachen
Sprachen-anwender	Angreifer, Systementwickler, IDS-Administratoren	Systementwickler	Systementwickler, IDS-Administratoren

3.2 Aktuelle Herausforderungen

Obwohl an der Entwicklung von Intrusion-Detection-Systemen bereits seit etwa 20 Jahren gearbeitet wird und mittlerweile über 130 freie, kommerzielle und prototypische Systeme bekannt sind [Mei04b], hat die Technologie der Missbrauchserkennung noch nicht die Reife erreicht, um ihre volle Wirksamkeit zu entfalten. Die mangelnde Zuverlässigkeit der Erkennung derzeit verfügbarer Systeme führt oft zu hohen Fehlalarmraten, die häufig den Nutzen der Systeme infrage stellen [Ra02]. Gleichzeitig führt die wachsende Leistungsfähigkeit von IT-Systemen zu höheren Anforderungen hinsichtlich der Analyseeffizienz von Missbrauchserkennungssystemen.

3.2.1 Fehlalarme

Da Missbrauchserkennungssysteme typischerweise deterministische Verfahren zur Suche nach den in den Signaturen kodierten Mustern einsetzen, sind, eine effektive Audit-Funktion vorausgesetzt, Fehlalarme per Definition ausgeschlossen. Das entspricht jedoch nicht den praktischen Erfahrungen [Ra02]. So berichtet Julisch in [Ju00] von Fehlalarmraten von über 99 Prozent und 10.000 Fehlarmen pro Monat beim Einsatz eines kommerziellen Missbrauchserkennungssystems in einem operationalen Netzwerk. Dies entspricht einem Fehlalarm durchschnittlich alle vier Minuten. Andere Evaluierungen von Intrusion-Detection-Systemen bestätigen dieses Bild [Li+00a, Li+00b, De+02].

Als Ursachen für die hohen Fehlalarmraten von Missbrauchserkennungssystemen wurden folgende Faktoren beobachtet (vgl. [Ju03]):

- unterspezifizierte Signaturen,
- mutmaßende (intent-guessing) Signaturen und
- fehlende Abstraktion.

Unterspezifizierte Signaturen nutzen notwendige aber nicht hinreichende Erkennungskriterien einer Attacke. Deshalb werden auch sicherheitskonforme Ereignisse als Attacken angezeigt. Beispielsweise werden statt komplexen regulären Ausdrücken, mit denen eine zuverlässige Erkennung verschiedener Attacken möglich wäre, häufig nur einfache String-Vergleiche in den Signaturen spezifiziert. Gründe für diese Vorgehensweise liegen zum einen in den strengen Anforderungen an die Verarbeitungszeiten von IDS, denen häufig Priorität gegenüber der exakten Erkennung von Attacken gegeben wird [Il93, Pta+98]. Zum anderen werden unscharfe Signaturen als Verallgemeinerungen verwendet, um auch Varianten von Attacken erkennen zu können [De+01]. Außerdem machen informationelle Defizite in den Audit-Daten u. U. eine Unterspezifikation der Signaturen erforderlich [Pri97, Ri99a, Ka01]. Signifikante Informationen, die in den Audit-Daten unvollständig oder unzuverlässig dokumentiert werden, können in den Signaturen nicht verwendet werden. Schließlich ist die Komplexität der Signaturentwicklung nicht zu unterschätzen [Ku95, Mou97, Li+98], durch die die Erstellung fehlerhafter, möglicherweise unterspezifizierter Signaturen gefördert wird.

Mutmaßende Signaturen werden verwendet, um Ereignisse zu erkennen, die möglicherweise Attacken darstellen. Beispielsweise existieren verschiedene Optionen für Netzwerkpakete, bei deren Auftreten pauschal eine Sicherheitsverletzung unterstellt wird, obwohl eine sicherheitskonforme Nutzung dieser Optionen möglich ist.

In existierenden IDS fehlen meistens Möglichkeiten, Zusammenhänge bestimmter Ereignisse in Signaturen auszudrücken. Dieser Mangel an Abstraktion führt häufig zu redundanten Alarmen, die auf eine einzelne Attacke zurückzuführen sind. Beispielsweise wird bei der Verwendung eines Port-Scanners zur systematischen Prüfung der Verfügbarkeit und Verwundbarkeit von Diensten auf einem System typischerweise für jeden getesteten Dienst ein Alarm erzeugt, obwohl alle Dienstzugriffe auf den gleichen Angriff zurückzuführen sind.

Beobachtungen von Axelsson [Ax00] unterstreichen die Problematik schlechter Signaturen, die leicht zu einer überproportionalen Zahl von Fehlalarmen führen können. Da sicherheitskonforme Ereignisse viel häufiger sind als sicherheitsverletzende, führt bereits eine gelegentliche Fehlklassifikation von Ereignissen zu unakzeptablen Fehlalarmraten, wie durch Anwendung des Satzes von Bayes gezeigt werden kann (vgl. [Ax00]).

Obwohl die zu erkennenden Attacken bekannt und in der Regel gut untersucht sind, fehlen geeignete Methoden für die Spezifikation der Erkennungskriterien. Fehlendes Verständnis für die Problematik der Signaturbeschreibung und -modellierung spiegelt sich auch in der Vielzahl unterschiedlichster Sprachen zu diesem Zweck wider (vgl. [Vig+00]), die unterschiedlichsten und zum Teil gegensätzlichen Ansätzen folgen. Es existieren verschiedene Ansichten darüber, was in Attackenmustern zu charakterisieren ist. Systematische Betrachtungen dieser Frage sind nicht bekannt.

Unter Verwendung der existierenden Sprachen werden Signaturen meistens durch ein direktes Herauslesen und Erraten der signifikanten Merkmale aus den Exploits erstellt, wobei typischerweise nur die im Exploit enthalten Aktionen betrachtet werden und nicht auch der zugehörige Einsatzkontext. Deshalb verwundert es kaum, dass validierte Signaturen zu einer Attacke in der Regel erst nach Patches bzw. Updates vorliegen, die die zugrunde liegende Verwundbarkeit beseitigen (vgl. [Li+02]).

3.2.2 Effiziente Erkennung

Ein weitere Herausforderung, der sich die Intrusion-Detection-Technologie gegenüber sieht, ist die wachsende Leistungsfähigkeit sowohl von Netzwerken als auch Endsystemen, mit der Steigerungen des Aufkommens an Protokolldaten einhergehen. Darüber hinaus führt die wachsende Komplexität der IT-Systeme zu neuen Verwundbarkeiten und damit Angriffswegen, so dass die Zahl zu analysierender Signaturen zunimmt. Damit gewinnt die Effizienz von Verfahren zu Missbrauchserkennung an Bedeutung. Bereits heute werden in Lastsituationen Protokolldaten von IDS verworfen [Schae+03] oder die Erkennung von Sicherheitsverletzungen wird

signifikant verzögert, so dass Gegenmaßnahmen nicht mehr oder nur noch eingeschränkt möglich sind.

Um dieser Herausforderung gerecht zu werden, verfolgt man verschiedene Ansätze. Beispielsweise wird die Einbruchserkennung auf der Grundlage einer Analyse kompakterer, weniger detaillierter Netzwerkprotokolldaten (z.B. NETFLOWS [Ci02]) untersucht [Mc04, So+02]. Dabei wird zugunsten einer schnelleren Erkennung auf eine detaillierte Protokollierung und Analyse verzichtet. Daraus resultierende Falsch-Negative (nicht erkannte Attacken) werden in Kauf genommen. Darüber hinaus wurden verschiedene optimierte Analyseverfahren für signaturbasierte, netzwerkbasierte Single-Step-IDS entwickelt. In [Krü+03] wurde ein Verfahren vorgestellt und für SNORT implementiert, welches die Signaturen in einen Entscheidungsbaum überführt, mit dessen Hilfe während der Analyse so wenig wie möglich redundante Vergleiche durchgeführt werden. Optimierte String-Matching-Algorithmen für Single-Step-IDS wurden beispielsweise von Anagnosttakis et al. [An+03] entwickelt und implementiert. Bisher wenig Aufmerksamkeit wurde hingegen der Optimierung der Analyseverfahren von Multi-Step-IDS gewidmet. In diesem Bereich werden hauptsächlich Standardverfahren wie Expertensysteme eingesetzt und die wenigen speziellen Verfahren [Ku95, Vi+00] widmen der Auswertungseffizienz nur wenig Aufmerksamkeit. In [Krü02] wurde zwar Wert auf eine effiziente Analyse gelegt, allerdings auf Kosten der Ausdrucksstärke zur Beschreibung von Attackenmustern. Daraus resultieren zwangsläufig Fehlalarme und nicht erkannte Attacken.

3.2.3 Fazit

Zusammenfassend ist festzuhalten, dass zur Erreichung einer effektiven und effizienten Missbauchserkennung folgende Fragen einer genaueren Untersuchung bedürfen:

- Was sind die relevanten Charakteristika von Attackenmanifestierungen, die in Signaturen spezifiziert werden müssen, um eine exakte Erkennung zu ermöglichen?
- Wie können (komplexe) Signaturen geeignet modelliert und beschrieben bzw. kodiert werden?
- Wie können Signaturen effizient(er) analysiert werden?

Antworten auf diese Fragen werden wir im Weiteren betrachten. Zunächst führen wir jedoch reale Beispiele für Systeme und Attacken ein, die zur Veranschaulichung herangezogen werden.

4 Beispiele

In diesem Buch werden verschiedene reale Beispiele für Sicherheitsverletzungen in IT-Systemen und deren Erkennung diskutiert. Als Beispielumgebung werden hauptsächlich Rechnersysteme mit dem Betriebssystem Solaris betrachtet, da für diese Umgebung die meisten Erfahrungswerte vorliegen. Dieses Kapitel gibt einen Überblick über die Umgebung und stellt Beispielattacken vor.

4.1 Beispielumgebung Solaris

Zuerst werden die Implementierungs- und Konfigurationsschwächen von Solaris erläutert, die die Grundlage für die Beispielattacken bilden. Nach einer kurzen Beschreibung der Audit-Funktion der Beispielumgebung werden vier Beispielattacken vorgestellt und deren Manifestierungen durch Audit-Records diskutiert.

4.1.1 Schwachstellen

Das Betriebssystem Solaris weist verschiedene Schwachstellen auf, die zur Umgehung von Sicherheitsmechanismen verwendet werden können. Zum Verständnis der Beispielattacken und -signaturen werden die zugrunde liegenden Schwachstellen kurz vorgestellt. Eine ausführliche Diskussion findet sich in [Mei+99].

Einmal-Authentifikation durch Passwörter

Wie die meisten verbreiteten Mehrbenutzer-Betriebssysteme realisiert Solaris eine einmalige Authentifizierung eines Benutzers vor Beginn der Systemnutzung. Um Zugang zum System zu erlangen, muss ein Nutzer eine gültige Nutzerkennung vorweisen (Identifikation) und die Kenntnis eines zugehörigen festen Passworts nachweisen (Authentifikation). Nach erfolgter Authentifikation erhält der Nutzer zeitlich unbeschränkten Zugang zum System. Zur Überwindung dieses Sicherheitsmechanismus kann ein An-

greifer beispielsweise durch Erraten oder systematisches Probieren das zur Authentifikation erforderliche Passwort ermitteln.

Umgebungsvariablen PATH

Aus Gründen der Bequemlichkeit ist es beim Start von Programmen nicht notwendig, den vollständigen (absoluten) Pfadnamen der Programmdatei anzugeben. Stattdessen wird die Umgebungsvariable PATH genutzt, in der Verzeichnisse aufgelistet sind, die beim Start eines Programms nach der entsprechenden Programmdatei durchsucht werden. Es kann auch angegeben werden, dass jeweils das aktuelle Verzeichnis durchsucht werden soll. Diese Verzeichnisse werden in der Reihenfolge durchsucht, in der sie in der Variable aufgeführt wurden. Die erste gefundene ausführbare Datei mit dem entsprechenden Programmnamen wird zur Ausführung gebracht.

Dieses weit verbreitete Konzept der Suchpfade bietet verschiedene Möglichkeiten für IT-Sicherheitsverletzungen. Es kann ausgenutzt werden, um Trojanische Pferde im System zu platzieren. Verfügt ein Angreifer über Schreibrechte auf ein Verzeichnis, das am Anfang der PATH-Variable enthalten ist, so kann er hier Programmdateien mit den Namen regulärer Applikationen ablegen. Führt ein Benutzer daraufhin eine dieser Applikationen aus, so wird das Programm des Angreifers ausgeführt, da es als erstes im Suchpfad gefunden wurde. Das Programm des Angreifers wird hierbei mit den Rechten des Benutzers ausgeführt.

Das Konzept der Suchpfade findet nicht nur bei der Suche nach ausführbaren Dateien Anwendung. Häufig benutzen Programme einzelne Funktionen aus dynamischen Bibliotheken (Shared Libraries), die zur Laufzeit geladen werden. Wo die einzelnen Bibliotheken zu finden sind, wird über eine Pfadvariable angegeben. Ist ein Angreifer in der Lage, durch Ausnutzung dieses Konzepts Bibliotheken auszutauschen, so führt dies dazu, dass Anweisungen des Angreifers mit den Rechten eines regulären Benutzers ausgeführt werden.

Super-User root

In UNIX-Systemen wie Solaris existiert zu Administrationszwecken der privilegierte Benutzer root. Da gängige UNIX-Systeme nicht zwischen unterschiedlichen Administrationsrollen trennen, verfügt der Benutzer root über uneingeschränkte Zugriffsrechte. Aus diesem Grund ist die Erlangung der Zugriffsrechte dieses Super-Users ein attraktives Angriffsziel.

SUID-Mechanismen

In UNIX-Betriebssystemen existiert ein *Set-User-ID*-Mechanismus *(SUID)*, bei dessen Verwendung ein Programm nicht mit den Rechten des ausführenden Nutzers sondern mit den Rechten des Eigentümers der Programmdatei ausgeführt wird. Um diesen Mechanismus zu aktivieren, ist ein spezielles Zugriffsrechte-Bit (SUID-Bit) für die Programmdatei zu aktivieren. Ein Beispiel für die Verwendung des SUID-Mechanismus ist das Programm passwd. Das Programm wird verwendet, um das Passwort von Benutzern zu ändern. Es kann von allen Benutzern ausgeführt werden. Passwortdaten werden in der Datei /etc/passwd gespeichert, die nur vom Systemadministrator root modifiziert werden kann. Um Benutzern eine Änderung ihrer Passwörter und somit das Schreiben in die Passwortdatei zu ermöglichen, ist passwd ein SUID-Root-Programm, d.h., es wird immer mit den Rechten des Systemadministrators root ausgeführt.

4.1.2 Audit-Funktion

Das *Basic Security Module (BSM)* ist eine optional einsetzbare Systemerweiterung für das Betriebssystem Solaris. Das Modul enthält zusätzliche Sicherheitsfunktionen für das Betriebssystem. Zu diesen gehört ein Audit-Mechanismus, der die Protokollierung sicherheitsrelevanter Aktivitäten ermöglicht. Für jeden sicherheitsrelevanten Systemruf und verschiedene sicherheitskritische Anwendungsaktionen kann eine Aufzeichnung von Audit-Daten konfiguriert werden. Die Audit-Funktion dokumentiert das Auftreten sicherheitsrelevanter Aktionen durch Generierung eines Audit-Records, in dem die Aktionen, das veranlassende Subjekt sowie die involvierten Systemressourcen beschrieben werden. Eine ausführliche Beschreibung der Audit-Funktion des BSM findet sich in [Ri99a, Sun02].

Zur Verarbeitung und Auswertung der vom BSM erstellten Protokolldaten wurde im Rahmen der Entwicklung des IDS AID eine Sensor-Applikation entwickelt, die die Audit-Daten ausliest und aufbereitet [Ri+99]. Die Aufbereitung der Audit-Daten umfasst dabei u. a. die Erhebung zusätzlicher Informationen. Beispielsweise wird durch den Sensor für jeden Dateizugriff zusätzlich ermittelt, ob es sich bei der Datei um ein Skript handelt, da diese Information zur zuverlässigen Erkennung verschiedener Attacken erforderlich ist. Zur einfacheren Verarbeitung legt der Sensor die aufbereiten Audit-Records in einer festen Record-Struktur ab. Die Elemente dieser Struktur sind in Tabelle 4-1 aufgeführt und kurz erläutert. Sie werden in Abhängigkeit von der zu protokollierenden Aktion mit entsprechenden Werten belegt. Audit-Records in diesem Format bilden die Grundlage für die weiteren Betrachtungen von Attacken- und Signatur-

Beispielen. Eine ausführliche Beschreibung der einzelnen Elemente des AID-Audit-Records erfolgt in [Ri+99].

Tabelle 4-1. Elemente des AID-Audit-Records

Audit-Record-Elemente	Erläuterung
audit_id	Audit-ID des Subjekts
egid	effektive Gruppen-ID des Subjekts
euid	effektive Benutzer-ID des Subjekts
rgid	Reale Gruppen-ID des Subjekts
ruid	Reale Benutzer-ID des Subjekts
tty	verwendetes Terminal
from	von wo aus angemeldet
pid	Prozess-ID
session_id	Sitzungs-ID
time	Datum und Uhrzeit
classes	Klasse des Audit-Ereignisses
event	Ereignis-ID
error	(Fehler-)Rückgabewert
status	finaler Aktionsstatus
rname	Name der involvierten Ressource
rowner	Ressourcen-Eigentümer
rgowner	Eigentümergruppe
rperm	Zugriffsrechte der Ressource
rscript	Leer oder Shell-Name
rinode	Inode-ID der Ressource
rmount	leer oder Name des NFS-Servers
rrname	Name der zweiten Ressource
arg	Aufrufargumente
env	Shell-Umgebungsvariablen
Host	Rechnername

4.2 Beispielattacken

Nachdem in Abschn. 4.1.1 verschiedene Schwachstellen der Beispielumgebung vorgestellt wurden, werden nun konkrete Attacken eingeführt, die diese Schwachstellen ausnutzen. Gleichzeitig werden die Manifestierungen der Attacken diskutiert.

4.2.1 Login-Attacke

Die Login-Attacke zielt auf ein systematisches Ausprobieren von Passwörtern durch wiederholte Authentifikations- bzw. Anmeldeversuche ab. Sie nutzt die in Abschn. 4.1.1 beschriebene Schwachstelle der Einmal-Authentifikation durch Passwörter aus. Die Solaris-Beispielumgebung verfügt über verschiedene Zugangspunkte, an denen eine Authentifikation durchgeführt wird. Neben der Anmeldung an der Systemkonsole können beispielsweise verschiedene Netzdienste oder Programme zur Änderung des Passwortes verwendet werden.

Auf einen systematischen Versuch das Passwort zu ermitteln, wird geschlossen, wenn innerhalb von 30 Sekunden das Passwort eines Nutzers mindestens dreimal falsch eingegeben wurde. Sowohl erfolgreiche als auch fehlgeschlagene Anmeldeversuche werden durch die Audit-Funktion des BSM dokumentiert. Durch den Sensor wird ein Audit-Record generiert, das u. a. die Art der Anmeldung (Konsole, Netzdienst etc.), den Zeitpunkt des Anmeldeversuchs, die Kennung des Nutzers, für den die Anmeldung erfolgt, sowie den Erfolg oder Misserfolg der Aktion dokumentiert. Abb. 4-1 stellt ein AID-Audit-Record für eine fehlerhafte Anmeldung dar. In der Abbildung sind die Record-Elemente hervorgehoben, die zur Erkennung und für ggf. zu erstellende Alarmmeldungen relevant sind. Außerdem ist die ereignisspezifische Interpretation der Elemente angeben. Das dargestellte Audit-Record dokumentiert einen fehlgeschlagenen Anmeldeversuch für den Benutzer mit der ID *1066* auf Rechner *escalus*. Die Anmeldung erfolgte mittels *telnet* von dem Rechner mit der IP-Adresse *141.43.3.165*.

Die Korrelation verschiedener Anmeldeversuche erfolgt über den Namen bzw. die ID des Benutzers. Die Manifestierung einer Login-Attacke besteht aus mindestens drei Audit-Records, die fehlerhafte Anmeldeversuche des gleichen Benutzers in einem Zeitraum von 30 Sekunden beschreiben. Der in Abb. 4-2 dargestellte Automat beschreibt mögliche Aktionsfolgen, die eine erfolgreiche Durchführung der Login-Attacke darstellen. Bei der Erkennung von Login-Attacken ist es zusätzlich erforderlich, auch Aktionen zu verfolgen, die ein Vervollständigen einer Attackeninstanz unmöglich machen. Wird beispielsweise nach der ersten fehlerhaften Anmeldung eine erfolgreiche Anmeldung durchgeführt, die erneut von zwei fehlerhaften Versuchen gefolgt wird, so soll diese Aktionsfolge möglicherweise nicht als Instanz der Login-Attacke betrachtet werden. Dies gilt ebenfalls für fehlerhafte Anmeldungen, die mehr als 30 Sekunden nach dem ersten Anmeldefehler auftreten. Um einen entsprechenden Fehlalarm zu vermeiden, müssen während der Erkennung auch erfolgreiche und verspätete Anmeldeversuche als Abbruchaktionen der Login-Attacke verfolgt

werden. Sie sind ebenfalls in Abb. 4-2 dargestellt. Bei den folgenden Betrachtungen der Aktionen von Beispielattacken verzichten wir auf die Darstellung der Abbruchaktionen, da sie nicht zu den Aktionsfolgen der Attacken gehören.

Audit-Record	Relevante Elemente	Interpretation
SolarisRecord {		
1066, 200, **1066**, 200, **1066**,	Audit-ID effektive Benutzer-ID reale Benutzer-ID	Anmeldeversuch des Benutzers mit der ID 1066
6291458,		
2368406437,	von wo angemeldet	IP-Adresse: 141.43.3.165
18904, 18904,		
1035388720,	Datum und Uhrzeit	23.10.2002 17:58:40 +543 msec
4096,	Ereignisklasse	An-/Abmeldeereignis
6154,	Ereignis-ID	Telnet-Anmeldung
-1,	(Fehler-)Rückgabewert	fehlerhaft
-1, "", -1, -1, -1, "", 0, "",		
"invalid password",	zweite Ressource	hier Fehlerursache
"", "",		
"escalus"	Rechnername	Anmeldung an Rechner escalus
}		

Abb. 4-1. AID-Audit-Record für eine fehlerhafte Anmeldung

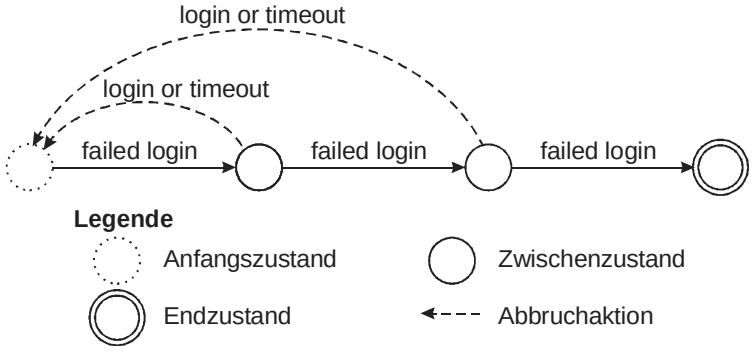


Abb. 4-2. Aktionsfolgen und Abbruchaktionen der Login-Attacke

4.2.2 PATH-Attacke

Eine PATH-Attacke hat zum Ziel, eine vom Angreifer vorgegebene Programmdatei mit fremden Zugriffsrechten und Privilegien auszuführen. Sie nutzt dazu die in Abschn. 4.1.1 beschriebenen Schwachstellen der Umgebungsvariable PATH und den SUID-Mechanismus aus.

Die Durchführung der PATH-Attacke erfordert ein Zusammentreffen von einer kritischen Belegung der PATH-Variable, die hier vom Angreifer erstellt werden kann, und die Existenz eines unvorsichtig (schlecht) programmierten SUID-Skripts. Dieses SUID-Skript ruft eine Anwendung mit einem relativen statt einem absoluten Pfandnamen auf, z. B. ls statt /usr/bin/ls. Durch Manipulation der PATH-Variable veranlasst ein Angreifer, dass bei Ausführung des SUID-Skripts eine von ihm festgelegte Anwendung mit den Privilegien des Skript-Eigentümers ausgeführt wird.

Charakteristisch für das Auftreten dieser Attacke sind folgende Kriterien:

1. Es wird eine Programmdatei ausgeführt.
2. Die Programmdatei befindet sich in einem kritischen Pfad, d.h. einem Verzeichnis, in dem der Angreifer Programme hinterlegen kann.
3. Die Programmdatei wird aus einem SUID-Skript heraus gestartet.
4. Die Ausführung des SUID-Skripts wurde nicht vom Eigentümer des Skripts veranlasst.

Die Ausführung einer Programmdatei aus einem Skript heraus, wird in der Solaris-Umgebung durch folgenden Ablauf realisiert. Das Skript wird gestartet und der ausführende Prozess erzeugt einen Kindprozess. Der Kindprozess führt die Programmdatei aus. Die Kriterien 1 und 2 können direkt auf der Grundlage der bei der Ausführung der Programmdatei durch den Kindprozess erzeugten Audit-Records überprüft werden. Zur Überprüfung der Kriterien 3 und 4 ist es erforderlich, auch die Ausführung des Skripts zu verfolgen. Außerdem muss die Erzeugung des Kindprozesses verfolgt werden, um den Zusammenhang zwischen dem Skript-Prozess und dem Kindprozess, der die Programmdatei ausführt, herstellen zu können.

Die PATH-Attacke manifestiert sich durch drei Audit-Records, die bei der Ausführung des Skripts, der Erzeugung des Kindprozesses und der Ausführung der Programmdatei generiert werden. Jedes dieser Audit-Records enthält die Prozess-ID des ausführenden Prozesses. Die Ausführung des Skripts kann mit der Erzeugung des Kindprozesses über diese Prozess-ID korreliert werden. Das Audit-Record zur Erzeugung des Kindprozesses enthält sowohl die Prozess-ID des Eltern- also auch Kindprozesses. Über die Kindprozess-ID und die Prozess-ID der Ausführung der Programmdatei können diese beiden Aktionen in Beziehung gesetzt werden. Verschie-

dene Instanzen dieser Attacke unterscheiden sich durch unterschiedliche involvierte Prozess-IDs. Der Automat in Abb. 4-3 stellt dem Schema von Abb. 4-2 folgend die relevanten Aktionsfolgen dieser Attacke dar. Die Ziffern in den Zuständen geben zusätzlich an, nach welcher Aktion die Informationen zur Überprüfung der oben genannten Kriterien vorliegen. Beispielsweise liefert das Audit-Record zur Ausführung des Skripts die Informationen zur Prüfung der Kriterien 3 und 4. Die Kriterien 1 und 2 sind erst auf der Grundlage des Audit-Records zur Ausführung der Programmdatei überprüfbar.

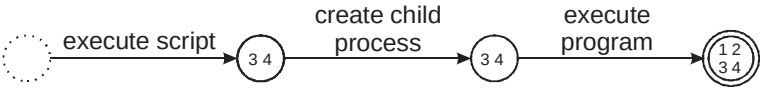


Abb. 4-3. Aktionsfolgen der PATH-Attacke

Wichtig ist hier, dass das Skript mehrere Kindprozesse erzeugen kann, die jeweils verschiedene Programmdateien ausführen. Entsprechend genügt es zur Erkennung der Attacke nicht, nur einen Kindprozess zu verfolgen. Stattdessen müssen alle Kindprozesse betrachtet werden. Mit der Erzeugung eines Kindprozesses wird hier jeweils eine neue potentielle Attackeninstanz erstellt. Das heißt, dass durch die Erzeugung eines Kindprozesses der Vorzustand dieser Aktion nicht verbraucht wird. Hinsichtlich dieser Eigenschaft werden Aktionen in konsumierende und nicht-konsumierende Aktionen unterschieden. Die Konsumeigenschaft von Aktionen wird in Kapitel 5 detailliert betrachtet.

Um Fehlalarme bei einer Erkennung dieser Attacke zu vermeiden, ist es außerdem erforderlich, Aktionen zu berücksichtigen, die begonnene Attacken abbrechen. Dazu gehört hier die vorzeitige Beendigung der involvierten Prozesse. Terminiert beispielsweise ein Kindprozess und damit die entsprechende Attackeninstanz, so darf diese Attackeninstanz durch das IDS nicht weiterverfolgt werden. Andernfalls könnte ein anderer Prozess die Prozess-ID des Kindprozesses zugewiesen bekommen und eine entsprechende Programmdatei ausführen. Dies würde durch das IDS fälschlicherweise als Vervollständigung der Attackeninstanz erkannt werden, obwohl der ausführende Prozess kein Kindprozess eines SUID-Skripts ist. Da Abb. 4-3 nur die zur Durchführung der Attacke erforderlichen Aktionen darstellt, sind derartige Abbruchaktionen nicht aufgeführt.

4.2.3 Link-Attacke

Ziel der Link-Attacke ist es, den Zugang zur Kommandozeile mit Root-Rechten zu erhalten. Die Attacke basiert auf der Ausnutzung des SUID-Mechanismus (vgl. Abschn. 4.1.1) und einer Verwundbarkeit, die in älteren Solaris-Versionen vorzufinden ist und ab Solaris 2.5 beseitigt wurde. In den betroffenen Betriebssystemversionen kann ein Benutzer bzw. Angreifer, der eine Datei ausführt, deren Name mit einem Bindestrich beginnt, z. B. „*-fn*“, eine interaktive Shell mit den Rechten des Administrators (*root*) erlangen. Wir nehmen an, *script1* sei ein Shell-Skript, dessen Eigentümer *root* ist und für das das SUID-Bit gesetzt ist. Zur Durchführung der Attacke erstellt der Nutzer einen Link mit dem Namen *–fn*, der auf *script1* verweist. Mit der Ausführung von *–fn* erlangt der Nutzer eine interaktive Shell, die mit den Rechten des Administrators *root* ausgeführt wird.

Die Attacke kann nicht nur zur Erlangung von Root-Privilegien sondern auch zur Erlangung fremder Privilegien verwendet werden. Vorraussetzung für diese Attacke ist, dass im System SUID-Skripte existieren, die zur Durchführung der Attacke verwendet werden können.

Charakteristisch für diesen Angriff ist das Auftreten des durch die folgenden Kriterien gekennzeichneten Ereignisses:

1. Ein Subjekt (Angreifer) führt ein Objekt aus.
2. Das Objekt ist ein Link, der auf ein SUID-Skript verweist.
3. Der Objektname beginnt mit „–“.
4. Das Subjekt ist nicht Eigentümer des Skripts.
5. Das Subjekt agiert nicht mit seiner ursprünglichen Identität (Auswirkung des SUID-Mechanismus).

Die Ausführung eines Objekts wird von der Audit-Funktion durch ein entsprechendes Audit-Record dokumentiert. Dieses Audit-Record protokolliert außerdem, die aktive Subjektidentität, die ursprüngliche Subjektidentität und den Objektnamen. Anhand dieser Informationen können die Kriterien 1, 3 und 5 überprüft werden. Zur Überprüfung der Kriterien 2 und 4 reichen diese Informationen nicht aus. Die hierfür erforderlichen Informationen, werden durch die Audit-Funktion nur bei Erstellung des Links protokolliert. Aus diesem Grund ist es zur Erkennung dieser Attacke erforderlich, die Erzeugung der Links zu verfolgen und entsprechende Informationen aufzuzeichnen.

Dokumentiert die Audit-Funktion die Erzeugung eines Links, der auf ein SUID-Skript verweist (Kriterium 2), durch ein Subjekt, das nicht Eigentümer des SUID-Skripts (Kriterium 4) ist, dann werden die entspre-

chenden Informationen erfasst. Bei der Ausführung eines Objekts, kann dann anhand des Objektnamens überprüft werden, ob es sich bei dem Objekt um einen Link auf ein SUID-Skript handelt.

Da zur Korrelation der Objektausführung und der Erzeugung des Links der Objektname verwendet wird, ist es erforderlich, möglicherweise auftretende Umbenennungsaktionen des Links ebenfalls zu verfolgen. Weiter erweist es sich als sinnvoll, Kriterium 3 bereits bei der Erzeugung bzw. Umbenennung des Objekts zu überprüfen. Die Ausführung eines Objekts muss dann nur noch näher untersucht werden, wenn durch entsprechende Aktionen ein Zustand erreicht wurde, in dem die Kriterien 2, 3 und 4 erfüllt sind.

Die Link-Attacke manifestiert sich in Audit-Records, die eine entsprechende Erzeugung des Links auf eine SUID-Skript, die Ausführung dieses Links und evtl. aufgetretene Umbenennungen des Links dokumentieren. Verschiedene Instanzen dieser Attacke unterscheiden sich in dem Namen des Links.

Dem Schema von Abb. 4-3 folgend, beschreibt der in Abb. 4-4 dargestellte Automat mögliche Aktionsfolgen, die einer erfolgreichen Durchführung der Attacke entsprechen. Die Ziffern in den Zuständen des Automaten geben dabei an, zu welchen charakteristischen Kriterien des Angriffs im jeweiligen Zustand Informationen vorliegen.

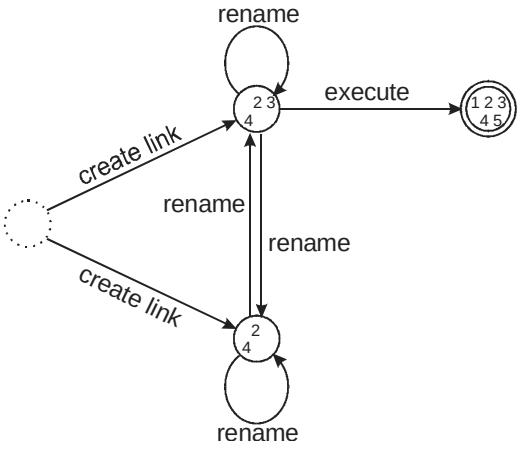


Abb. 4-4. Aktionsfolgen der Link-Attacke

Aktionen, die begonnene Attacken abbrechen, z. B. das Löschen eines Links, sind in dem Automaten nicht dargestellt, da sie nicht zur Attacke gehören. Bei der Erkennung der Attacke müssen sie jedoch berücksichtigt werden, um Fehlalarme zu verhindern.

4.2.4 Nebenläufige Link-Attacke

Die zuvor diskutierte Link-Attacke setzt die Existenz von SUID-Skripten voraus, für die Links erstellt werden. In Abhängigkeit von den Gegebenheiten der konkreten Systemumgebung kann unter Umständen nicht ausgeschlossen werden, dass während des laufenden Betriebs weitere SUID-Skripte entstehen. In diesem Fall werden durch den Automaten in Abb. 4-4 nicht alle möglichen Aktionsfolgen zur Durchführung der Link-Attacke beschrieben. Es sind zusätzliche Szenarien möglich, in denen zunächst ein Link auf ein Shell-Skript angelegt wird und das Skript danach durch eine chmod-Aktion mit dem SUID-Bit versehen wird, jedoch bevor der Link zur Ausführung kommt. Das bedeutet, dass das Erstellen und Umbenennen eines Links zeitlich unabhängig von der Belegung des Skripts mit dem SUID-Bit erfolgen kann. Alle möglichen Reihenfolgen dieser Aktionen können zu einer Durchführung der Attacke genutzt werden.

Die im vorigen Abschnitt beschriebene Link-Attacke, berücksichtigt nur Aktionsfolgen, bei denen zunächst ein SUID-Skript erstellt wird und danach ein Link auf das Skript angelegt wird, d. h., zum Zeitpunkt der Erstellung des Links existiert das SUID-Skript bereits. Alle anderen Reihenfolgen der Aktionen, die ebenfalls eine Attacke darstellen, werden als nebenläufige Link-Attacke bezeichnet. Die zur Erkennung dieser Attacke zu prüfenden Kriterien sind:

1. Ein Subjekt (Angreifer) führt ein Objekt aus.
2. Das Objekt ist ein Link, der auf ein Shell-Skript verweist.
3. Das Shell-Skript ist mit dem SUID-Bit versehen.
4. Der Objektname beginnt mit „–“.
5. Das Subjekt ist nicht Eigentümer des Skripts.
6. Das Subjekt agiert nicht mit seiner ursprünglichen Identität (Auswirkung des SUID-Mechanismus).

Die nebenläufige Link-Attacke manifestiert sich durch Audit-Records, die die Erzeugung von Links auf ein Skript, evtl. aufgetretene Umbenennungen des Links, die Ausführung dieses Links sowie das Setzen des SUID-Bits (per Systemruf chmod) für ein Skript dokumentieren. Verschiedene Instanzen dieser Attacke unterscheiden sich durch den Namen des Links.

Der Automat in Abb. 4-5 beschreibt mögliche Aktionsfolgen zur Durchführung der nebenläufigen Link-Attacke und stellt dar, in welchen Zuständen Informationen zur Prüfung welcher Kriterien vorliegen. Bei der Erkennung von Attacken sind auch hier Aktionen zu berücksichtigen, die ein

Vervollständigen begonnener Attacken unmöglich machen, bspw. das Löschen des Link oder das Entfernen des SUID-Bits von Skripten.

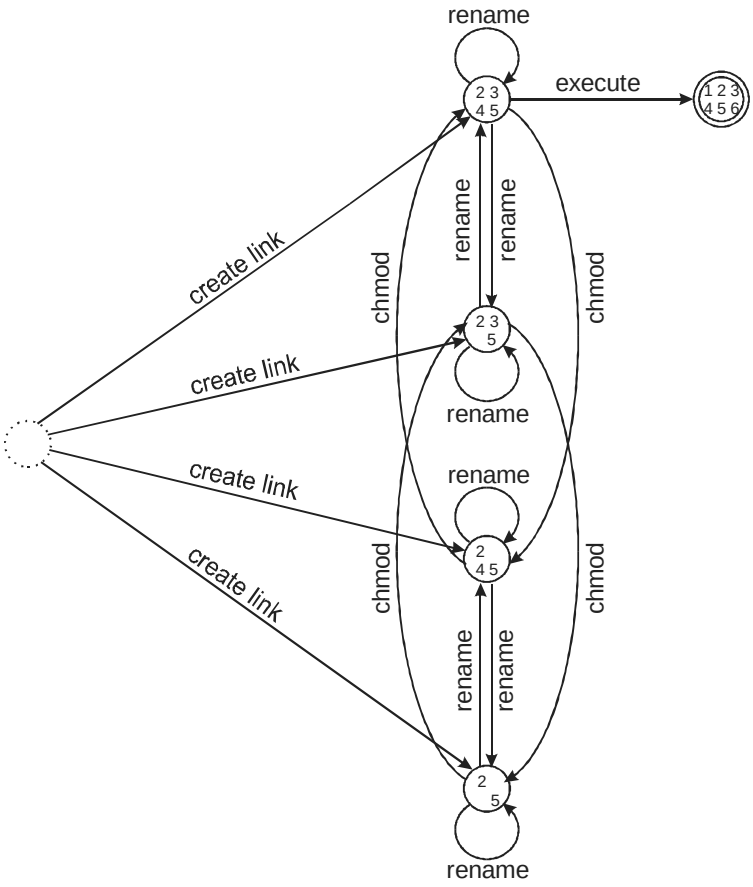


Abb. 4-5. Aktionsfolgen der nebenläufigen Link-Attacke

5 Semantische Aspekte von Angriffssignaturen

Zur Modellierung und Beschreibung von Angriffssignaturen wurden verschiedene Ansätze in der Literatur vorgeschlagen und realisiert. Diese unterscheiden sich hinsichtlich der adressierten Aspekte von Signaturen teilweise deutlich voneinander. Das hat eine sehr unterschiedliche und begrenzte Ausdrucksstärke der jeweiligen Beschreibungsmittel zur Folge, so dass diese Methoden lediglich für eine eingeschränkte Menge von Signaturen geeignet sind. Bevor wir in folgenden Kapiteln die Modellierung und Beschreibung von Signaturen diskutieren, führen wir in diesem Kapitel ein Modell für die Semantik von Signaturen ein. Das Modell stellt systematisch die Aspekte von Signaturen dar, die in einer Beschreibung charakterisiert werden müssen, um eine adäquate Erkennung von Situationen, die Sicherheitsverletzung darstellen, zu ermöglichen. Es definiert Anforderungen an Methoden zur Modellierung und Beschreibung von Signaturen.

Das Kennzeichnen und Identifizieren von relevanten Situationen erfolgt typischerweise durch die Beschreibung eines entsprechenden Ereignisses. Ereignisbeschreibungen enthalten Informationen über die Umstände, die das Ereignis auslösen, und die Auswirkungen auf weitere Ereignisse. Die Charakterisierung von Ereignissen ist in verschiedenen Bereichen der Informationsverarbeitung von Bedeutung. Im Zusammenhang mit dem von uns betrachteten Problemfeld kann insbesondere auf Erkenntnisse aus der Theorie aktiver Datenbanken zurückgegriffen werden.

Aktive Datenbanken erweitern klassische Datenbankmanagementsysteme (DBMS) um aktive Komponenten zur Reaktion auf spezifizierte Ereignisse. In der Theorie aktiver Datenbankmanagementsysteme existieren Modelle zur Charakterisierung von Ereignissen [Zim+99], die auf den Problembereich der Signaturbeschreibung angepasst werden können [Mei04a]. Wir stellen zunächst das Konzept aktiver Datenbanken kurz vor und arbeiten Unterschiede zu Signaturanalysesystemen heraus. Davon ausgehend wird ein Modell für die Semantik von Signaturen entwickelt und die verschiedenen semantischen Aspekte werden anhand von Beispielen diskutiert.

5.1 Aktive Datenbanksysteme

Konventionelle Datenbanksysteme sind passive Systeme. Sie führen Anfragen oder Transaktionen nur auf Veranlassung von außen, z. B. durch den Benutzer oder eine Anwendung, durch. In bestimmten Situationen kann es jedoch wünschenswert sein, auf spezielle Datenbankzustände automatisch reagieren zu können. Zu diesem Zweck wurden aktive Datenbanksysteme entwickelt, die beispielsweise automatisch Integritätsbedingungen durchsetzen, abgeleitete Daten berechnen oder verteilte Berechnungen koordinieren [Pa+99].

5.1.1 Ereignisse in aktiven Datenbanken

Aktive Datenbanken verwenden aktive Regeln, so genannte *Event-Condition-Action-Regeln (ECA-Regeln)* [Pa98], um Situationen zu beschreiben und im Falle ihres Eintretens darauf zu reagieren. Eine ECA-Regel besitzt folgende Form:

```
on event
if condition
do action
```

Die Bedingungen einer ECA-Regel werden nach dem Eintreten des Ereignisses überprüft. Sind sie erfüllt, so wird die Aktion der Regel ausgeführt. Auslösende Ereignisse sind typischerweise Datenmanipulationsoperationen, wie das Einfügen oder Aktualisieren eines Elements. Komplexe Ereignisse, die Kombinationen von Ereignissen darstellen, können unter Verwendung von Operatoren der Ereignisalgebra einer Ereignisbeschreibungssprache spezifiziert werden.

In der Literatur wurden eine Reihe von Ereignissprachen für aktive Datenbanksysteme vorgeschlagen. Beispiele sind HiPAC [Da+96], NAOS [Co+96], SNOOP [Ch+94] und ACOOD [Be91]. Diese unterscheiden sich teilweise stark in der umgesetzten Ereignissemantik und besitzen dadurch unterschiedliche Mächtigkeiten. Zimmer [Zim98] untersuchte existierende Ereignissprachen für aktive Datenbanken und entwickelte ein Meta-Modell für die Semantik von Ereignissen in aktiven Datenbanken, das systematisch die verschiedenen Aspekte von Ereignissen in aktiven Datenbanken darstellt.

5.1.2 Unterschiede zum Signaturkonzept

Das Ereigniskonzept aktiver Datenbanken unterscheidet sich in einigen Aspekten vom Konzept der Signatur, so dass Zimmer's Modell im Bereich der Signaturanalyse nicht ohne weiteres anwendbar ist. Diese Unterschiede werden im Folgenden diskutiert, bevor in den nächsten Abschnitten ein entsprechend angepasstes Modell für die Semantik von Signaturen vorgestellt wird.

Im Unterschied zur ECA-Regeln aktiver Datenbanken besitzen Signaturen keine dreigeteilte Struktur. Stattdessen werden das Ereignis, die Bedingungen und die Aktionen[1] in einer Spezifikation eines komplexen Ereignisses zusammengefasst. Im Kontext der Missbrauchserkennung treten Ereignisse und damit verbundene Auswirkungen auf das Eintreten anderer Ereignisse nur ein, wenn auch die spezifizierten Bedingungen erfüllt sind. Darüber hinaus können mit einer komplexen Signaturereignisbeschreibung assoziierte Aktionen bereits ausgelöst werden, bevor das komplexe Ereignis eingetreten ist. Eine Signatur kann beschreiben, dass Aktionen bereits nach dem Auftreten bestimmter Teile des komplexen Ereignisses ausgelöst werden. Dadurch ist es möglich, auf Sicherheitsverletzungen zu reagieren, die noch nicht vollständig beendet wurden.

Ein weiterer Unterschied besteht in der Handhabung von partiellen Ereignissen. Darunter sind komplexe Ereignisse zu verstehen, die teilweise aber noch nicht vollständig eingetreten sind (*partielle Ereignisinstanzen*). Aktive Datenbanken kennen dieses Konzept nicht, da sie einer anderen Instanzerzeugungsstrategie folgen. Im Kontext aktiver Datenbanken werden Instanzen komplexer Ereignisse durch das Eintreten des letzten Ereignisses, das Teil des komplexen Ereignisses ist, erzeugt (*späte Instanzerzeugung*) und gleichzeitig ausgelöst. Im Gegensatz dazu wird bei der Signaturanalyse mit dem Eintreten des ersten Ereignisses, das Teil eines komplexen Ereignisses ist, eine Instanz des komplexen Ereignisses angelegt (*frühe Instanzerzeugung*). Diese Instanzen sind unvollständig bis alle anderen Ereignisse, die Teil des komplexen Ereignisses sind, aufgetreten sind und werden erst nach ihrer Vervollständigung ausgelöst. Es ist möglich, dass eine solche partielle Instanz niemals vollständig wird und eintritt, sondern stattdessen verworfen wird, da Ereignisse eintreten, die eine Vervollständigung unmöglich machen. Dieses Vorgehen erlaubt es, auf teilweise erkannte Attacken zu reagieren, beispielsweise durch Erzeugen eines „gelben" Alarms.

---

[1] Typischerweise benennen diese Spezifikationen nur die auszulösenden Aktionen, die an anderer Stelle unter Verwendung von Reaktionssprachen (vgl. Abschn. 3.1) spezifiziert wurden.

Aufgrund der beschriebenen und in Tabelle 5-1 zusammengefassten Unterschiede zwischen dem Ereigniskonzept aktiver Datenbanken und dem Signaturkonzept kann Zimmer's Systematik für die Semantik von Ereignissen in aktiven Datenbanken nicht ohne weiteres auf Signaturen übertragen werden. Ein entsprechend angepasstes Modell für die Semantik von Signaturen wird in den folgenden Abschnitten vorgestellt.

Tabelle 5-1. Unterschiede der Ereigniskonzepte

	Aktive Datenbanken ECA-Regeln	**Missbrauchserkennungssysteme Signaturen**
Auswirkungen zwischen Teilereignissen	Auswirkung auch bei nicht erfüllter Teilbedingung möglich	Auswirkung nur bei erfüllter Teilbedingung möglich
	Trennung zwischen Ereignis, Bedingung und Aktion: Ereignisse: (e_1, e_2, e_3) Bedingungen: (c_1 c_2 c_3) Aktionen: (a_1 a_2 a_3)	Integration von Ereignis, Bedingung und Aktionen: Ereignis 1: (e_1, c_1, a_1) Ereignis 2: (e_2, c_2, a_2) Ereignis 3: (e_3, c_3, a_3)
Aktionen	Aktionen erst nach Eintreten des komplexen Ereignisses möglich	Aktionen bereits nach Eintreten von Teilen des komplexen Ereignisses möglich
Erzeugung und Auslösung komplexer Ereingisinstanzen	Erzeugung und Auslösung nach letztem Teilereignis	Erzeugung nach erstem Teilereignis Auslösung nach letztem Teilereignis

5.2 Ereignisse – Begriffseinführung

In diesem Abschnitt werden die relevanten Begriffe und Konzepte zur Beschreibung des Semantikmodells und zur weiteren Diskussion eingeführt. Sie sind weitgehend an die Terminologie aus der Theorie Aktiver Datenbanken [Zim98] angelehnt.

Definition 5-1: (Ereignis, Basisereignis, Ereignistyp und Ereignisinstanz)

Die zentrale verwendete Abstraktion ist das Ereignis. Ein *Ereignis* ist ein Geschehnis, das durch eine Menge von Merkmalen charakterisiert wird. Es existiert eine Menge von *Basisereignissen*, welche die grundlegenden er-

kennbaren Einheiten darstellen. Basisereignisse sind atomar und an einen bestimmten Zeitpunkt gebunden.

Analog zu den Konzepten Datentyp und Instanz in objekt-orientierten Programmiersprachen (siehe z. B. [Brü+04]) wird zwischen Ereignistypen und Ereignisinstanzen unterschieden. Ein *Ereignistyp* beschreibt die Merkmale, die alle Ereignisse dieses Typs besitzen. Eine konkrete Ausprägung eines Ereignistyps wird als *Ereignisinstanz* bezeichnet. Bei der Beschreibung von Attackensignaturen wird demnach ein Ereignistyp spezifiziert, der die Merkmale einer Attackenart charakterisiert. Ein Auftreten einer solchen Attacke resultiert in einer Ereignisinstanz des entsprechenden Ereignistyps.

Im Weiteren wird nur dann von Ereignisinstanzen und Ereignistypen gesprochen, wenn eine Abgrenzung der beiden Konzepte zum Verständnis erforderlich ist. Andernfalls, insbesondere wenn sich die Bedeutung aus dem Kontext ergibt, wird allgemein von Ereignis gesprochen.

Definition 5-2: (Audit-Ereignisse und Zeitereignisse)
Basisereignisse werden in Audit-Ereignisse und Zeitereignisse unterschieden. *Audit-Ereignisse* entsprechen Audit-Records einer Audit-Trail, die Zustände des überwachten Systems oder das Auftreten sicherheitsrelevanter Aktionen dokumentieren. *Zeitereignisse* werden durch einen Zeitgeber generiert. Sie beschreiben bestimmte Zeitpunkte, entweder absolut (um 7.00 Uhr) oder relativ (15 Minuten nach Ereignis *e*).

Definition 5-3: (Komplexes Ereignis, Komponenten-Ereignis, (partielle) komplexe Ereignisinstanz und Eltern-Ereignistyp)
Ein *komplexes Ereignis* besteht aus einer Menge von Ereignissen, die in diesem Zusammenhang als *Komponenten-Ereignisse* bezeichnet werden. Synonym werden auch die Begriffe *Schritt* oder *Teilereignis* verwendet. Die Teilereignisse eines komplexen Ereignisses stehen zueinander zeitlich oder bzgl. ihrer Merkmale in Beziehung. Temporale Relationen werden unter Verwendung von Operatoren einer Ereignisalgebra ausgedrückt.

Eine konkrete Ausprägung eines komplexen Ereignisses wird als *komplexe Ereignisinstanz* bezeichnet. Sie beschreibt die Menge der Komponenten-Ereignisse. Alle nicht-leeren echten Teilmengen dieser Menge, deren Elemente den zeitlichen Beziehungen genügen, werden als *partielle komplexe Ereignisinstanzen* bezeichnet. Sie beschreiben (noch) nicht vollständig aufgetretene komplexe Ereignisinstanzen.

Der Ereignistyp eines komplexen Ereignisses, das aus Teilereignissen besteht, stellte den *Eltern-Ereignistyp* für diese Teilereignisse dar. Ein Ereignis kann Teilereignis mehrerer komplexer Ereignisse verschiedenen Typs sein. Dementsprechend kann ein Ereignis mehrere Eltern-Ereignistypen besitzen.

Definition 5-4: (Kontextbedingungen, Intra-Ereignis-Bedingungen, Inter-Ereignis-Bedingungen)
Zur Charakterisierung komplexer Ereignisse ist typischerweise auch der Kontext relevant in dem die Teilereignisse eintreten. Mit Hilfe von *Kontextbedingungen* können Einschränkungen bzgl. des Kontextes von Ereignissen definiert werden. Kontextbedingungen werden in Intra-Ereignis-Bedingungen und Inter-Ereignis-Bedingungen unterschieden. *Intra-Ereignis-Bedingungen* sind Boolesche Formeln deren Atome Vergleiche zwischen Merkmalen eines Teilereignisses und Konstanten darstellen. Ihre Evaluierung erfordert lediglich die Inspektion der Merkmale des entsprechenden Teilereignisses. *Inter-Ereignis-Bedingungen* sind Boolesche Formeln deren Atome Vergleiche zwischen Merkmalen verschiedener Teilereignisse darstellen.

Definition 5-5: (Initiale, innere, finale und abbrechende Teilereignisse)
Ausgehend von den spezifizierten temporalen Relationen können die Teilereignisse eines komplexen Ereignisses in initiale, innere und finale Schritte unterschieden werden. Der erste Schritt wird als *initiales Teilereignis* bezeichnet. Ein *finales Teilereignis* stellt den letzten Schritt eines komplexen Ereignisses dar. Außerdem existieren *abbrechende Teilereignisse*, die Umstände charakterisieren, die das Eintreten des komplexen Ereignisses unmöglich machen. Beim Auftreten eines abbrechenden Ereignisses wird die bisher unvollständige komplexe Ereignisinstanz verworfen. Alle Schritte, die weder initial, final noch abbrechend sind, werden als *innere Teilereignisse* bezeichnet.

Axiome zum Zeitmodell
Im Hinblick auf die Zeitpunkte zu denen Ereignisse eintreten, gehen wir von der Gültigkeit der folgenden Axiome aus.

1. Entsprechend Definition 5-2 repräsentiert ein Audit-Ereignis ein Audit-Record, das einen beobachteten Systemzustand oder eine sicherheitsrelevante Aktion dokumentiert. Innerhalb des Audit-Records wird auch der Zeitpunkt der Beobachtung bzw. der Ausführung der Aktion festgehalten. Dieser Zeitpunkt bestimmt den Zeitpunkt des Eintretens des korrespondierenden Audit-Ereignisses. Die Eintrittszeitpunkte von Ereignissen reflektieren die Reihenfolge in der Ereignisse aufgetreten sind.
2. Der Eintrittszeitpunkt eines komplexen Ereignisses ist durch den Zeitpunkt des Eintretens seines finalen Teilereignisses bestimmt.
3. Ereignisse können simultan eintreten, d. h., verschiedene Ereignisinstanzen können den gleichen Eintrittszeitpunkt besitzen.

Definition 5-6: ((Totale) Ordnungsrelationen $<_{\text{time}}$ und $<_{\text{trail}}$, Ereignis-Trail, Verschärfung)
Eine Ordnungsrelation $\mathrm{R} \subseteq \mathrm{E} \times \mathrm{E}$ auf einer Ereignismenge E ist *total* wenn

$$\forall \mathrm{a}, \mathrm{b} \in \mathrm{E} \text{ mit } \mathrm{a} \neq \mathrm{b} ((\mathrm{a}, \mathrm{b}) \in \mathrm{R} \vee (\mathrm{b}, \mathrm{a}) \in \mathrm{R}).$$

Die Ordnungsrelation $<_{\text{time}}$ auf der Menge E ist definiert als

$$\{(\mathrm{e}, \mathrm{e}') | \mathrm{e}, \mathrm{e}' \in \mathrm{E} \text{ und der Zeitstempel von e ist kleiner als der von e}'\}$$

.

Für eine Ereignismenge E ist $\mathrm{s} =< \mathrm{e}_1, \mathrm{e}_2, \mathrm{e}_3, ... >$ eine *sequentielle Anordnung*, wenn jedes Ereignis aus E in s vorkommt. Eine sequentielle Anordnung s der Elemente von E *respektiert* die Ordnungsrelation $<_{\text{time}}$ wenn

$$\forall \mathrm{e}, \mathrm{e}' \in \mathrm{s} (\mathrm{e} <_{\text{time}} \mathrm{e}' \Rightarrow \mathrm{e} \text{ steht in s vor e}').$$

Durch eine sequentielle Anordnung s der Elemente einer Menge E ist die totale Ordnung $<_{\mathrm{s}}$ auf E definiert als

$$\{(\mathrm{e}, \mathrm{e}') \mid \mathrm{e}, \mathrm{e}' \in \mathrm{s} \text{ und e steht in s vor e}'\}.$$

Eine *Ereignis-Trail* ist eine $<_{\text{time}}$ respektierende sequentielle Anordnung einer Menge von Ereignissen E. Die durch eine Ereignis-Trail definierte totale Ordnungsrelation wird $<_{\text{trail}}$ bezeichnet. Es gilt: wenn $<_{\text{time}}$ auf E eine totale Ordnungsrelation ist, dann ist $<_{\text{trail}} = <_{\text{time}}$.

Eine Ordnungsrelation r' *verschärft* eine Ordnungsrelation r bzw. ist eine *Verschärfung* der Ordnungsrelation r wenn gilt: $\mathrm{r} \subset \mathrm{r}'$.

Zur Verdeutlichung der beiden Relationen $<_{\text{time}}$ und $<_{\text{trail}}$ sowie der eingeführten Eigenschaften werden diese für die Ereignismengen *E1* und *E2* aus Abb. 5-1 diskutiert. Für die Menge *E1* existiert nur eine $<_{\text{time}}$ respektierende sequentielle Anordnung $\mathrm{s}_1 =< \mathrm{a}_1, \mathrm{c}_2, \mathrm{b}_3, \mathrm{d}_4 >$, da $<_{\text{time}}$ auf *E1* bereits eine totale Ordnung liefert. Dementsprechend sind die Relationen $<_{\text{time}}$ und $<_{\text{trail}}$ auf *E1* gleich. Für *E2* liefert $<_{\text{time}}$ keine totale Ordnung und es existieren mehrere $<_{\text{time}}$ respektierende sequentielle Anordnungen von *E2* z. B. $\mathrm{s}_2 =< \mathrm{a}_1, \mathrm{e}_1, \mathrm{c}_2, \mathrm{f}_2, \mathrm{b}_3, \mathrm{d}_{4,} \mathrm{g}_4 >$ und $\mathrm{s}_3 =< \mathrm{e}_1, \mathrm{a}_1, \mathrm{f}_2, \mathrm{c}_2, \mathrm{b}_3, \mathrm{g}_4, \mathrm{d}_4 >$. Dementsprechend ist $<_{\text{trail}}$ auf der Menge *E2* eine Verschärfung von $<_{\text{time}}$.

Notation
Die Darstellung von Beispielereignissen erfolgt in den weiteren Abschnitten unter Verwendung folgender Notation. Große Buchstaben stellen Ereignistypen und kleine Buchstaben Ereignisinstanzen dar. Bei Ereignisinstanzen ist der Ereignistyp im verwendeten Buchstaben kodiert und ein

numerischer Index wird benutzt, um verschiedene Ereignisinstanzen eines Ereignistyps entsprechend der Reihenfolge ihrer Eintrittszeitpunkte zu benennen. Zum Beispiel enthält die Ereignis-Trail *{a₁ a₂ a₃}* eine Sequenz von drei Ereignisinstanzen vom Ereignistyp *A*. Des Weiteren werden komplexe Ereignisinstanzen und ihre Schrittinstanzen graphisch veranschaulicht. Dazu wird die mit Hilfe von Abb. 5-2 eingeführte Notation verwendet. Die Abbildung zeigt eine Ereignis-Trail *{a₁ a₂ b₁}* sowie die daraus gebildeten (partiellen) Instanzen der komplexen Ereignisse E_1, E_2 und E_3, die wie folgt definiert sind:

- E_1 : Sequenz von drei Ereignissen des Typs *A*.
- E_2 : Sequenz von drei Ereignissen des Typs *A*, während der kein Ereignis vom Typ *B* auftritt.
- E_3 : Sequenz von zwei Ereignissen vom Typ *A*.

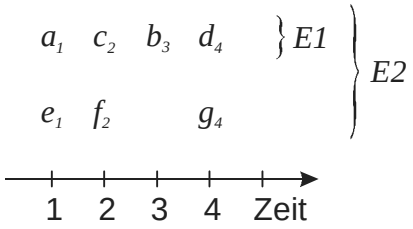


Abb. 5-1. Ordnung und Sequentialisierung

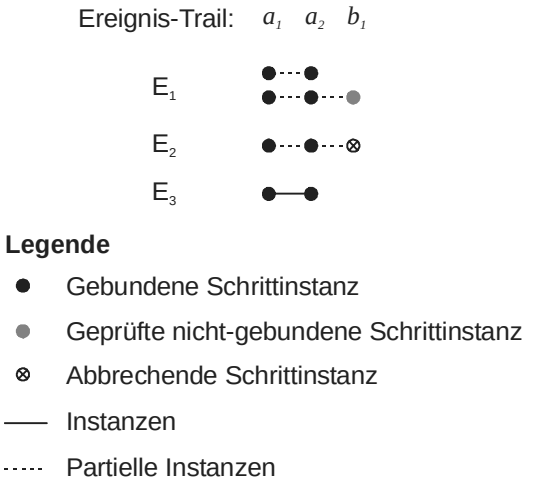


Abb. 5-2. Graphische Notation für Ereignisinstanzen

Instanzen von komplexen Ereignissen werden mit durchgezogenen Linien, partielle Instanzen mit gestrichelten Linien dargestellt. Die Veranschaulichung der an eine (partielle) Instanz gebundenen Schrittinstanzen erfolgt durch schwarze gefüllte Kreise. Dabei werden die Kreise unter dem der Schrittinstanz entsprechenden Ereignis aus der Ereignis-Trail dargestellt. Beispielsweise werden die Ereignisse a_1 und a_2 durch die beiden Instanzen des Ereignistyps E_1 gebunden. In einzelnen Fällen ist es hilfreich, explizit darzustellen, dass ein Ereignis zwar überprüft jedoch nicht an eine Instanz gebunden wurde. Dazu werden graue gefüllte Kreise verwendet, wie für die zweite Instanz des Ereignistyps E_1 demonstriert. Ereignisse aus der Ereignis-Trail, die für eine partielle komplexe Ereignisinstanz einen abbrechenden Schritt darstellen, werden entsprechend durch nicht-gefüllte durchgestrichene Kreise angezeigt. Die Abbildung zeigt dies mit dem Ereignis b_1 für die Instanz des Ereignisses E_3 beispielhaft.

5.3 Dimensionen der Semantik von Signaturen

Ereignisbeschreibungen für Signaturen müssen drei verschiedene Fragen beantworten, um die beschriebene Situation ausreichend zu charakterisieren. Entsprechend kann die Semantik von Signaturen in drei Dimensionen unterteilt werden (vgl. [Zim98, Mei04a]).

Frage 1 (Ereignismuster): Welche Ereignisinstanzen einer Ereignis-Trail verursachen das Eintreten eines komplexen Ereignisses?

Verschiedene Kriterien beeinflussen die Entscheidung, ob eine Ereignis-Trail ein komplexes Ereignis auslöst. Beispielsweise spielen der Typ, die Reihenfolge und die Anzahl der Ereignisinstanzen der Trail eine entscheidende Rolle.

Frage 2 (Schrittinstanzenselektion): Welche Ereignisinstanzen einer Ereignis-Trail, die ein komplexes Ereignis auslöst, werden an die Schritte des komplexen Ereignisses gebunden?

An Schritte gebundene Ereignisinstanzen bilden einerseits die Grundlage für die weitere Korrelation des komplexen Ereignisses und stellen andererseits die Informationsmenge dar, die Grundlage für Reaktionen auf Ereignisse, beispielsweise die Generierung einer Alarmmeldung, ist. Da es mehrere verschiedene passende Ereignisinstanzen in einer Ereignis-Trail geben kann (vgl. Beispiel 5-1), die an Schritte des komplexen Ereignisses gebunden werden können, muss festgelegt werden können, welche Instanzen gebunden werden sollen.

Beispiel 5-1:
Gegeben sei eine Trail $t_1 := \{b_1\ x_1\ a_1\ c_1\ y_1\ b_2\ x_2\ b_3\ y_2\ c_2\}$. Wir nehmen an, dass eine Ereignisbeschreibung E_1 eine Sequenz von einem Ereignis vom Typ *A* gefolgt von einem Ereignis vom Typ *B* gefolgt von einem Ereignis vom Typ *C* spezifiziert. In Abb. 5-3 ist dargestellt, wie das Ereignismuster die Instanzen der Trail auswählt, die ein Auftreten von E_1 auslösen.

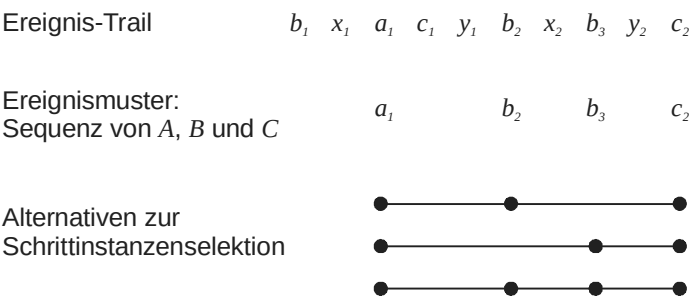


Abb. 5-3. Ereignismuster und Schrittinstanzenselektion am Beispiel

Abb. 5-3 zeigt ebenfalls, dass es Alternativen für die Auswahl einer Ereignisinstanz vom Typ *B* gibt, die an den entsprechenden Schritt gebunden werden soll. Kandidaten für Instanzen vom Type E_1 sind die Schrittinstanzpaare $\{a_1\ b_2\ c_2\}$, $\{a_1\ b_3\ c_2\}$ und $\{a_1\ b_2\ b_3\ c_2\}$.
Frage 3 (Schrittinstanzkonsum): Sollen bereits eingetretene Schritte einer (partiellen) komplexen Ereignisinstanz *nur mit dem ersten* passenden Auftreten eines folgenden Schrittes korreliert werden oder *auch mit allen weiteren* Instanzen des Folgeschrittes? Wird der durch bereits eingetretene Schritte einer komplexen Ereignisinstanz etablierte Zustand durch die erste Instanz des Folgeschrittes konsumiert oder werden auch alle weiteren Instanzen des Folgeschrittes in diesem Zustand betrachtet?

Angenommen in Beispiel 5-1 werden Schrittinstanzen vom Typ *A* nicht durch Schrittinstanzen vom Typ *B* konsumiert. In diesem Fall können durch die gegebene Ereignis-Trail die beiden Instanzen $\{a_1\ b_2\ c_2\}$ und $\{a_1\ b_3\ c_2\}$ ausgelöst werden. Sind hingegen alle Schritte von E_1 *konsumierend* (*consuming*), so kann nur die Instanz $\{a_1\ b_2\ c_2\}$ erzeugt werden.

Beispiel 5-2:
Wir betrachten hier Aktionen von Betriebssystemprozessen in der Solaris-Beispielumgebung, wie sie beispielsweise bei der Durchführung der PATH-Attacke (vgl. Abschn. 4.2.2) auftreten. Sei E_2 eine Ereignisbeschreibung, die jede Sequenz der folgenden Ereignisse identifiziert:

1. ein Prozess *p* beginnt die Ausführung eines Programms (Ereignistyp *A*),
2. Prozess *p* erzeugt einen Kindprozess (Ereignistyp *B*),
3. der Kindprozess beginnt die Ausführung eines bestimmten Programms (Ereignistyp *C*).

Da ein Prozess eine beliebige Zahl Kindprozesse erzeugen kann, unterscheiden sich Ereignisse vom Typ *B* in diesem Szenario von Ereignissen der Typen *A* und *C* in einer bestimmten Eigenschaft. Nachdem ein Ereignis a_1 aufgetreten ist, müssen alle folgenden Ereignisse der Typen *B* und *C* mit a_1 korreliert werden. Ursache dafür ist, dass die Ausführung eines Programms durch einen Prozess (Ereignistyp *A*) einen Zustand etabliert, der durch die Erzeugung eines Kindprozesses nicht konsumiert wird. Stattdessen können weitere Kindprozesse erzeugt werden. Ereignisse vom Typ *B* sind in diesem Szenario *nicht-konsumierend* (*non-consuming*), das heißt Ereignis a_1 wird niemals durch ein Ereignis vom Typ *B* konsumiert. Eine Ereignis-Trail $t_2 = \{a_1\ b_1\ c_1\ b_2\ c_2\}$ löst somit die beiden Instanzen $\{a_1\ b_1\ c_1\}$ and $\{a_1\ b_2\ c_2\}$ von E_2 aus.

Die genannten Fragen charakterisieren verschiedene semantische Dimensionen, die für die Spezifikation von Signaturen relevant sind. In den folgenden Abschnitten werden die einzelnen Aspekte der Dimensionen detailliert erörtert und an Beispielen demonstriert.

5.4 Ereignismuster

Das Ereignismuster einer Signatur beschreibt auf einer abstrakten Ebene das Suchmuster, das Instanzen der Signatur in einer Ereignis-Trail identifiziert. Ein Ereignismuster berücksichtigt die fünf Aspekte, die in den nächsten Abschnitten erörtert werden:

- Typ und Reihenfolge,
- Häufigkeit,
- Kontinuität,
- Nebenläufigkeit und
- Kontextbedingungen.

5.4.1 Typ und Reihenfolge

Das Grundgerüst einer Signatur wird durch die zugrunde liegenden Schritte und deren Reihenfolge geformt. Schritte einer Signatur können von einem komplexen Ereignistyp oder einem Basisereignistyp sein. Die Reihenfolge der Schritte wird unter Verwendung der Infix-Operatoren

- Sequenzoperator (;),
- Disjunktionsoperator (OR),
- Konjunktionsoperator (AND),
- Simultanoperator (||),

sowie dem Präfix-Operator

- Negationsoperator (NOT)

definiert.

Der Sequenzoperator ist wahrscheinlich der am meisten benutzte Operator. Schritte müssen in der Reihenfolge auftreten, die durch den Sequenzoperator festgelegt ist. Eine Instanz des Ereignistyps *(A; B; C; D)* tritt ein, wenn die Schritte der Typen *A*, *B*, *C* und *D* in der vorgegebenen Reihenfolge auftreten.

Der Disjunktionsoperator erlaubt die Beschreibung von Attackenmustern, die durch alternative Aktionen komplettiert werden können. Instanzen des Typs *(A OR B OR C)* treten ein, wenn ein Schritt von einem der angegebenen Typen auftritt. Anstatt einer Beschreibung aller möglichen Sequenzen alternativer Aktionen erlaubt die Verwendung des Disjunktionsoperators eine kompaktere und übersichtlichere Beschreibung von Signaturen.

Verschiedene Attacken enthalten Aktionssequenzen, die in beliebiger Reihenfolge ausgeführt werden können. Statt einer Beschreibung aller möglichen Kombinationen dieser Sequenzen ermöglicht der Konjunktionsoperator eine explizite und kompakte Darstellung dieses nebenläufigen Verhaltens. Eine Instanz des Typs *(A AND B AND C)* tritt ein, wenn Schritte der Typen *A*, *B* und *C* aufgetreten sind, ungeachtet Ihrer Reihenfolge.

Ereignisse vom Typ *(A || B || C)* werden ausgelöst, wenn alle Schritte aufgetreten sind und die Eintrittszeitpunkte der Schritte identisch sind. Der Simultanoperator ist nützlich zur Korrelation von Ereignissen von verschiedenen verteilten Quellen, die Aktionen simultan durchführen. Des Weiteren kann eine Instanz, deren Typ als finaler Schritt in Signaturen spezifiziert ist, mehrere Ereignisinstanzen von möglicherweise verschiedenen komplexen Eltern-Ereignistypen auslösen. Da komplexe Ereignisse ihren Eintrittszeitpunkt von dem des finalen Schritts ableiten, treten diese

verschiedenen Instanzen simultan auf. Mit Hilfe des Simultanoperators ist es möglich, Ereignisse hinsichtlich ihres simultanen Auftretens zu korrelieren.

Die Verwendung des Negationsoperators ist nur im Zusammenhang mit einem Zeitinterval sinnvoll, der als Sequenz von mindestens zwei Ereignissen spezifiziert wird. Aus diesem Grund wurde der Definition des Negationsoperators als n-stelligem Präfixoperator der Vorzug gegenüber einer Definition als unärem Operator gegeben (vgl. [Zim98]). Ereignisse des Typs *NOT(A, (B; C; ...; D))* treten ein, wenn sequentiell die Schritte der Typen *B*, *C*, ... und *D* eingetreten sind und zwischen den Eintrittszeitpunkten der Schritte vom Typ *B* und *D* kein Ereignis vom Typ *A* aufgetreten ist. Die Hauptanwendung des Negationsoperators liegt in der Spezifikation von Abbruchschritten einer Signatur.

Beispiel 5-3:
Angenommen eine Attacke besteht aus der Erzeugung eines bestimmten Prozesses (Ereignistyp *A*), der drei spezifische Aktionen in festgelegter Reihenfolge ausführt (Sequenz von Ereignissen der Typen *B*, *C* und *D*). Die Beendigung des Prozesses (Ereignistyp *E*) repräsentiert einen Abbruchschritt für diese Attacke. Der Ereignistyp *NOT(E, (A; B; C; D))* beschreibt eine Signatur für diese Attacke.

5.4.2 Häufigkeit

Ereignismuster, die das mehrfache Auftreten eines Schrittes fordern, werden als *Schrittketten* bezeichnet. Für Schrittketten kann die Zahl der Schrittinstanzen explizit spezifiziert werden. Dazu muss ein *Begrenzer* für jeden Schritt spezifiziert werden. Der Begrenzer kann eine Zahl darstellen, die angibt wie viele Schrittinstanzen *genau* (*n*), *mindestens* (*n*-) oder *höchstens* (-*n*) durch das komplexe Ereignismuster gefordert werden. Ein Begrenzer der Form *mindestens n* und *höchstens m*, wobei $n < m$, ist ebenfalls möglich (*n*-*m*). Ist für einen Schritt kein Begrenzer angegeben, so wird als Standardbegrenzung *mindestens 1* angenommen.

Die Semantik des Begrenzers für einen Schritt hängt davon ab, ob ein Schritt innerhalb eines Zeitintervalls auftreten muss oder nicht. Ein solches Zeitinterval wird durch die Vorgänger- und Nachfolgerschritte eines Schritts definiert.

Ein Schritt mit einem Höchstens-Begrenzer ist bereits eingetreten, wenn keine Schrittinstanz vorliegt, so dass seine Verwendung außerhalb eines Zeitintervals wenig sinnvoll ist. Die Semantik eines Mindestens-Begrenzers außerhalb eines Zeitintervals ist identisch mit der Semantik eines Ge-

nau-Begrenzers. Eine Instanz des Ereignistyps *((n-) B)* tritt ein, wenn die *n*-te Instanz vom Typ *B* auftritt.

Begrenzer für Schritte innerhalb eines Zeitintervals werden häufiger verwendet. Ein Genau-Begrenzer erlaubt die Spezifikation einer bestimmen Anzahl von Wiederholungen einer Aktion. Ein Ereignis des Typs *(A; (n) B; C)* tritt ein, wenn nach einer Instanz vom Typ *A n* Instanzen vom Typ *B* auftraten und darauf eine Instanz vom Typ *C* eingetreten ist. Tritt eine *n+1*-te Instanz vom Typ *B* auf bevor eine Instanz vom Typ *C* auftrat, so kann die partielle Instanz des komplexen Ereignisses nicht mehr vervollständigt werden. Die *n+1*-te Instanz vom Typ *B* stellt für diese Signatur einen impliziten Abbruchschritt dar.

Höchstens- und Mindestens-Begrenzer ermöglichen die Beschreibung von Attacken, die sich in einer Anzahl von Wiederholungen einer Aktion manifestieren, die einen bestimmten Schwellwert überschreitet. Beispiel 5-4 diskutiert eine solche Signatur.

Beispiel 5-4:
Die in Abschn. 4.2.1 eingeführte Login-Attacke besteht aus einer Anzahl wiederholter Anmeldeversuche. Ein Angreifer kann diese Prozedur verwenden, um ein Zugangspasswort durch Probieren zu ermitteln. Als Indikator für die Präsenz eines solchen Angriffs wird hier das Auftreten von mindesten sechs fehlerhaften Anmeldeversuchen innerhalb von 30 Sekunden verwendet. Die Zeitspanne zwischen dem ersten und dem letzten Anmeldeversuch wird durch eine Bedingung auf maximal 30 Sekunden beschränkt. Unter der Annahme, dass fehlgeschlagene Anmeldeversuche durch Ereignisse des Typs *A* dokumentiert werden, kann die entsprechende Signatur unter Verwendung eines Genau-Begrenzers wie folgt beschrieben werden:

(A; (4) A; A).

Mit Hilfe von expliziten Zeitereignissen und einem Mindestens-Begrenzer kann die Signatur wie folgt spezifiziert werden: $(T_0; (6\text{-}) A; T_{30})$. Instanzen dieses Typs treten ein, wenn T_{30} eintritt und zuvor mindestens sechs fehlgeschlagene Anmeldeversuche nach T_0 auftraten. Für dieses Beispiel ist die Ereignisbeschreibung $NOT(T_{31}, (T_0; (6) A))$ jedoch unter Umständen geeigneter, da Ereignisse dieses Typs bereits eintreten, wenn der sechste Anmeldeversuch auftritt.

Wie bereits erwähnt erfüllen Schritte mit einem Höchstens-Begrenzer bereits das Ereignismuster, wenn keine Schrittinstanz aufgetreten ist. Höchstens-Begrenzer modellieren demnach nicht das Auftreten von Ereignissen, sondern die Abwesenheit von Ereignissen bzw. die Abwesenheit einer genügenden Anzahl von Ereignissen. Eine Anwendung gibt das folgende Beispiel.

Beispiel 5-5:
Die Sicherheitspolitik einer Institution fordert, dass kryptographische Schlüssel, die zur Absicherung von Kommunikationsverbindungen verwendet werden, täglich dreimal gewechselt werden müssen. Um in der Lage zu sein, Verletzungen der Sicherheitspolitik zu erkennen, werden Schlüsselwechsel durch die Generierung eines Audit-Records (Ereignistyp *C*) dokumentiert. Folgende Signatur erkennt Verstöße gegen diese Sicherheitspolitik: *(T_1; (-2) C; T_2)*. Instanzen dieses Typs werden ausgelöst, wenn zwei oder weniger Ereignisse vom Typ *C* (Schlüsselwechsel) in der durch T_1 (00.00 Uhr) und T_2 (23.59 Uhr) definierten Zeitspanne eintreten. Eine alternative Beschreibung dieser Signatur ist *NOT((3-) C, (T_1; T_2)*. Offensichtlich kann jede Ereignisbeschreibung der Form *(A; (-n) B; C)* in die Form *NOT(((n+1)-) B, (A; C))* transformiert werden.

Schritte mit einem Mindestens- oder Genau-Begrenzer beschreiben eine Art Schleife. Der begrenzte Schritt stellt den Schleifenkörper dar und im Fall eines Genau-Begrenzers spezifiziert der Begrenzer die Schleifenabbruchbedingung. Das durch die Schleifenbedingung geprüfte Kriterium ist die Anzahl der aufgetretenen Schrittinstanzen. Im Fall eines Mindestens-Begrenzers stellen sowohl die Anzahl der aufgetretenen Schrittinstanzen als auch das Auftreten des Nachfolgerschrittes die Kriterien für die Abbruchbedingung der Schleife dar.

Zur Beschreibung verschiedener Situationen ist es hilfreich, in der Lage zu sein, weitere Schleifenabbruchbedingungen spezifizieren zu können. Zu diesem Zweck kann als eine Erweiterung des Semantikmodells die Möglichkeit eingeführt werden, Schleifenabbruchbedingungen auf der Basis von aggregierten Merkmalen des Schrittes im Schleifenkörper zu spezifizieren. Diese Erweiterung kann verwendet werden, um Ereignisse, wie das Lesen einer bestimmten Anzahl von Bytes aus einem Socket, zu beschreiben. Die Anzahl Bytes kann durch eine variable Anzahl von Read-Systemrufen gelesen werden. Anstatt des Zählens der Read-Systemrufe ist hier eine auf der Summe gelesener Bytes basierende Schleifenabbruchbedingung erforderlich. Eine entsprechende Ereignisbeschreibung könnte wie folgt aussehen: *((until sum(R.bytes) > 1000) R)*. Ereignisse vom Typ *R* repräsentieren hierbei Audit-Records, die Read-Systemrufe dokumentieren und das Merkmal *bytes* enthält die Zahl jeweils gelesener Bytes. Eine Instanz des beschriebenen komplexen Ereignisses tritt ein, wenn mehr als 1000 Bytes durch einen oder mehrere Read-Systemrufe gelesen wurden.

5.4.3 Kontinuität

Der Aspekt Kontinuität legt eine von zwei möglichen Interpretationen von Ereignisbeschreibungen fest. Zum Beispiel kann die Ereignisbeschreibung $E_2 := (A; B; C)$ wie folgt interpretiert werden:

1. Mindestens ein Schritt vom Typ *A* wird von mindestens einem Schritt vom Typ *B* gefolgt, auf den mindestens ein Schritt vom Typ *C* folgt.
2. wie 1., wobei kein Schritt vom Typ *C* zwischen den Schritten der Typen *A* und *B* und kein Ereignis vom Typ *A* zwischen den Schritten der Typen *B* und *C* auftreten darf.

Um die Bedeutung eines solchen Musters festlegen zu können, werden die Kontinuitätsmodi *nicht-kontinuierlich* (*non-continuous*) und *kontinuierlich* (*continuous*) unterschieden. Kontinuierlich (der Standardmodus) legt für E_2 die unter 1. und nicht-kontinuierlich die unter 2. genannte Bedeutung fest. Für die Missbrauchserkennung ist hauptsächlich der Modus kontinuierlich von Nutzen. Alle in diesem Buch diskutierten Beispielsignaturen verwenden diesen Modus.

5.4.4 Nebenläufigkeit

Die Nebenläufigkeitsmodi *überlappend* (*overlap*, Standardmodus) und *nicht-überlappend* (*non-overlap*) bestimmen, ob sich die mit den Schritten einer Signatur assoziierten Zeitintervalle überlappen dürfen. Seien beispielsweise folgende Ereignistypen gegeben: $X := (A; B; C)$, $Y := (D; E; F)$, $Z_1 := non\text{-}overlap\ (X; Y)$ und $Z_2 := overlap\ (X; Y)$. Das Ereignismuster von Z_1 verlangt, dass die Instanzen der Typen *A*, *B*, *C*, *D*, *E*, und *F* in dieser Reihenfolge auftreten. Hingegen umfasst Z_2 auch Ereignisse, bei denen die Folge der Ereignisse der Typen *A*, *B* und *C* und die Sequenz der Ereignisse der Typen *D* und *E* nebenläufig auftreten kann.

Verdeutlicht sei an dieser Stelle auch der Unterschied zwischen Z_2 und $Z_3 := (X\ AND\ Y)$ (vgl. Abschn. 5.4.1). Das Ereignismuster von Z_2 fordert, dass der Schritt vom Typ *F* in Ereignistyp *Y* nach dem Schritt vom Typ *C* in Ereignistyp *X* eintritt. Im Gegensatz dazu erlaubt Z_3, dass die Schritte der Ereignistypen *X* und *Y* nebenläufig auftreten. Abb. 5-4 illustriert die verschiedenen Bedeutungen.

5.4.5 Kontextbedingungen

Kontextbedingungen spezifizieren Einschränkungen bzgl. des Kontextes in dem Schritte eines Ereignisses auftreten. Für jeden Schritt kann eine Menge von *Intra-Ereignis-Bedingungen* definiert werden. Sie können beispielsweise dazu verwendet werden, um Schritte auf Aktionen zu beschränken, die durch einen bestimmten Benutzer oder auf einem festgelegten Rechner ausgeführt wurden. Der Typ eines Ereignisses ist ein Spezialfall für eine Intra-Ereignis-Bedingung. Weitere Einschränkungen können durch *Inter-Ereignis-Bedingungen* festgelegt werden. Diese erlauben es beispielsweise zwei oder mehrere Schritte auf Aktionen einzuschränken, die durch den gleichen Betriebssystemprozess ausgelöst wurden oder auf das gleiche Objekt zugreifen. Bei der Missbrauchserkennung werden Inter-Ereignis-Bedingungen hauptsächlich dazu verwendet, Ereignisse einer Attackeninstanz anhand instanzspezifischer Merkmale, z. B. der Prozess-ID (vgl. PATH-Attacke in Abschn. 4.2.2) oder dem Benutzernamen (vgl. Login-Attacke in Abschn. 4.2.1), der entsprechenden Signaturinstanz zuzuordnen.

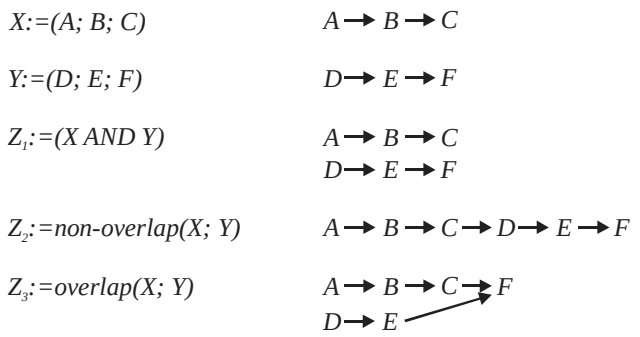


Abb. 5-4. Unterschiede der Nebenläufigkeitsoperatoren

5.5 Selektion der Schrittinstanzen

Bevor komplexe Ereignisse ausgelöst werden können, müssen Ereignisinstanzen an die Schritte des komplexen Ereignisses gebunden werden. An Schritte gebundene Ereignisinstanzen bilden die Grundlage zur weiteren Korrelation des komplexen Ereignisses und zur Durchführung von Aktionen, die auf Merkmale der Schritte Bezug nehmen. Da hinsichtlich der Auswahl, der an einen Schritt zu bindenden Ereignisinstanz, Alternativen bestehen können, muss die Ereignisbeschreibung Festlegungen zur Selek-

tion der Instanzen enthalten. Für jeden eingetretenen Schritt ist mindestens eine Ereignisinstanz auszuwählen, die Zuordnung mehrerer Instanzen zu einem Schritt kann jedoch ebenfalls sinnvoll sein. Im Kontext der Missbrauchserkennung ist diese Entscheidung von großer Bedeutung, da typischerweise nicht nur der Fakt, dass eine komplexe Ereignisinstanz (bzw. Attacke) aufgetreten ist, von Interesse ist, sondern außerdem die auslösenden Schritte (Aktionen) zu dokumentieren sind, um weitere Korrelationen und Gegenmaßnahmen zu ermöglichen. Relevante Selektionsalternativen werden im Kontext von Beispiel 5-6 diskutiert.

Beispiel 5-6:
Gegeben sei Ereignistyp $E_3 := (A; B; C)$, in dem Ereignisse des Typs *B* Audit-Ereignisse repräsentieren, die regelmäßig Sensorwerte, z. B. die CPU-Auslastung oder die Netzwerklast, dokumentieren. Gegeben sei weiterhin die Ereignis-Trail $t_3 := \{a_1\ b_1\ b_2\ b_3\ c_1\}$. Hier stellt sich die Frage, welche Instanzen vom Typ *B* selektiert werden. Es werden die drei Selektionsmodi *erste* (*first*, Standardmodus), *letzte* (*last*) und *alle* (*all*) unterschieden. Wird der Modus *erste* für Schritte des Typs *B* definiert, so wird durch t_3 die Instanz $\{a_1\ b_1\ c_1\}$ vom Typ E_3 erstellt. Bei Verwendung des Modus *letzte* würde t_3 stattdessen die Instanz $\{a_1\ b_3\ c_1\}$ erzeugen. Der Modus *alle* resultiert in der Erzeugung der Instanz $\{a_1\ b_1\ b_2\ b_3\ c_1\}$, die alle Ereignisse vom Typ *B* an den entsprechenden Schritt bindet. Abb. 5-5 veranschaulicht die verschiedenen Schrittinstanzselektionsmodi.

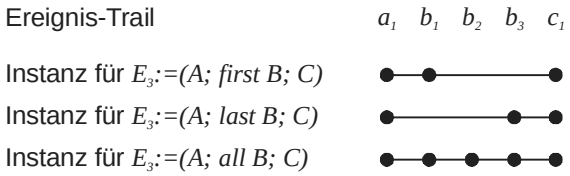


Abb. 5-5. Selektion der Schrittinstanzen

Alle drei Modi sind bei der Beschreibung von Attackensignaturen relevant. Der Modus *erste* kann verwendet werden, wenn der Schritt vom Typ *B* beschreibt, dass der Sensorwert einen bestimmten Schwellwert überschritten hat, und die erste Überschreitung des Schwellwerts in der durch die Ereignistypen *A* und *B* definierten Zeitspanne mit dem komplexen Ereignis dokumentiert werden soll. Soll stattdessen der letzte und damit aktuelle Wert eines Sensors im komplexen Ereignis dokumentiert werden, dann kann dies durch Verwendung des Modus *letzter* für den Schritt des Typs *B* erreicht werden. Der Modus *alle* für den Schritt vom Typ *B* be-

wirkt, dass alle Schwellwertüberschreitungen in der definierten Zeitspanne im komplexen Ereignis festgehalten werden.

5.6 Konsum von Schrittinstanzen

Die Dimension Schrittinstanzkonsum definiert, welche Auswirkungen das Eintreten einer Instanz eines Schrittes für die weitere Korrelation des komplexen Ereignisses hat. Sie legt fest, ob die bisherige partielle Instanz des komplexen Ereignisses nach Auftreten der Schrittinstanz zur Korrelation mit weiteren Instanzen dieses Schrittes verfügbar ist oder nicht.

Folgt für ein komplexes Ereignis *E:=(A; B; C)* nach dem Auftreten der Instanz a_1 eine Instanz b_1, dann wird die partielle Instanz vom Typ *E* erweitert zu $\{a_1\ b_1\ \ldots\}$. Darauf folgend tritt eine weitere Instanz b_2 vom Typ *B* ein. Die Frage, die durch Schrittinstanzkonsum beantwortet wird, lässt sich an diesem Beispiel wie folgt formulieren: Bilden a_1 und b_2 eine weitere partielle Instanz $\{a_1\ b_2\ \ldots\}$ vom Typ *E*, oder nicht?

Die Antwort auf diese Frage wird in der Ereignisbeschreibung durch Verwendung der Schrittinstanzkonsummodi *konsumierend* (*consuming*, Standardmodus) und *nicht-konsumierend* (*non-consuming*) festgelegt. Wurde für den Schritt vom Typ *B* der Modus *konsumierend* spezifiziert, dann wird die Schrittinstanz a_1 durch b_1 konsumiert. In diesem Fall kann Ereignis b_2 mit a_1 keine weitere partielle Instanz von Typ *E* bilden. Wurde Schritt *B* hingegen als *nicht-konsumierend* definiert, dann bildet jede Instanz vom Typ *B* mit jeder zuvor eingetretenen Instanz vom Typ *A* eine eigene partielle Instanz vom Typ *E*.

Abb. 5-6 illustriert die partiellen Instanzen der Ereignistypen E_4*:=(A; B; C)* und E_5*:=(A; non-consuming B; C)*, die durch die Ereignis-Trail $t_4:=\{a_1, b_1, b_2, a_2, b_3\}$ erzeugt werden.

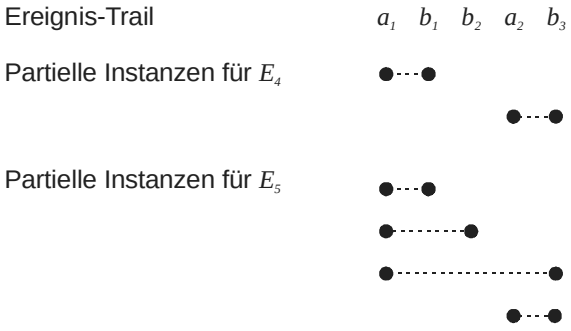


Abb. 5-6. Schrittinstanzkonsum

Der Instanzkonsummodus für einen Schritt ergibt sich einerseits aus der Rolle, die dieser Schritt in der Ereignisbeschreibung spielt. Andererseits werden die Konsummodi einzelner Schritte durch die mit der Signatur zu charakterisierende Situation vorgeben. In den folgenden Abschnitten wird anhand von Beispielen für verschiedene Sachverhalte dargelegt, wodurch jeweils der Konsummodus einzelner Schritte bestimmt ist.

5.6.1 Aktionsfolgen und Aktionssemantik

Beschreibt ein komplexes Ereignis eine bestimmte Abfolge von Aktionen, so leitet sich der Instanzkonsummodus eines Schrittes typischerweise aus der Bedeutung der durch diesen Schritt repräsentierten Systemaktion und deren Auswirkungen auf Systemobjekte ab. Wenn eine Signatur bspw. die Sequenz der Systemrufe eines Prozesses verfolgt, dann können die Instanzkonsummodi der Schritte aus der Bedeutung des jeweiligen Systemrufs in der Sequenz gefolgert werden. Zum Beispiel (vgl. Beispiel 5-2) werden Fork-Systemrufe (zur Erzeugung eines Kindprozesses) typischerweise durch nicht-konsumierende Schritte repräsentiert, da ein Prozess mehrere Kindprozesse erzeugen kann, die jeweils die Sequenz komplettieren können. Im Allgemeinen werden Aktionen zur Konstruktion von Systemobjekten (z. B. Erzeugung einer Datei oder eines Prozesses) durch nicht-konsumierende Schritte dargestellt. Der Modus nicht-konsumierend wird häufig auch für finale Schritte verwendet, deren wiederholtes Auftreten jeweils eine Instanz des komplexen Ereignisses auslösen soll.

Die Beendigung eines Prozesses wird hingegen durch einen konsumierenden Schritt abgebildet, da ein Prozess nur einmal beendet werden kann. Konsumierende Schritte werden in der Regel zur Beschreibung der Destruktion von Systemobjekten (Löschen einer Datei, Prozessterminierung etc.) oder der Änderung von Objekteigenschaften (Umbenennung einer Datei, Änderung der Zugriffsrechte einer Datei etc.) verwendet. Außerdem werden konsumierende Schritte zum Zählen gleichartiger Ereignisinstanzen in Schrittketten (vgl. Login-Attacke in Abschn. 4.2.1) oder zur Aggregation von Schrittmerkmalen in Schleifen verwendet. Abbruchschritte sind immer konsumierend, da sie Aktionen repräsentieren, die den Systemzustand derart verändert, dass eine weitere Verfolgung der Aktionsfolge zum Zweck der Angriffserkennung nicht mehr erforderlich ist.

5.6.2 Auswahl von Schrittkombinationen

Ereignisse dienen neben der Dokumentation von Aktionen auch zur Beschreibung von Zuständen des überwachten Systems. Einerseits können

Ereignisse, die Aktionen dokumentieren, Merkmale enthalten, die implizit einen zum Zeitpunkt der Aktion aktuellen Teilzustand des Systems repräsentieren. Zum Beispiel wird in einem Audit-Record zur Dokumentation der Ausführung eines Programms auch die aktuelle Belegung von Umgebungsvariablen festgehalten. Andererseits können durch eine Audit-Funktion explizit Audit-Records bzw. Ereignisse erzeugt werden, um Teilzustände des überwachten Systems, z. B. den aktuellen Wert eines Sensors (z. B. Prozessor- oder Netzwerkauslastung), zu dokumentieren. Auf der Grundlage solcher Zustandsbeschreibungen können u. a. Schwellwertüberschreitungen oder bestimmte Veränderungen in den dokumentierten Zuständen erkannt werden. Die interessanten Veränderungen werden durch eine entsprechende Bedingung charakterisiert. Durch diese Bedingung wird definiert, welche verschiedenen Instanzen (Sensorwerte bzw. Zustandsbeschreibungen) miteinander zu korrelieren sind, wie das folgende Beispiel verdeutlicht.

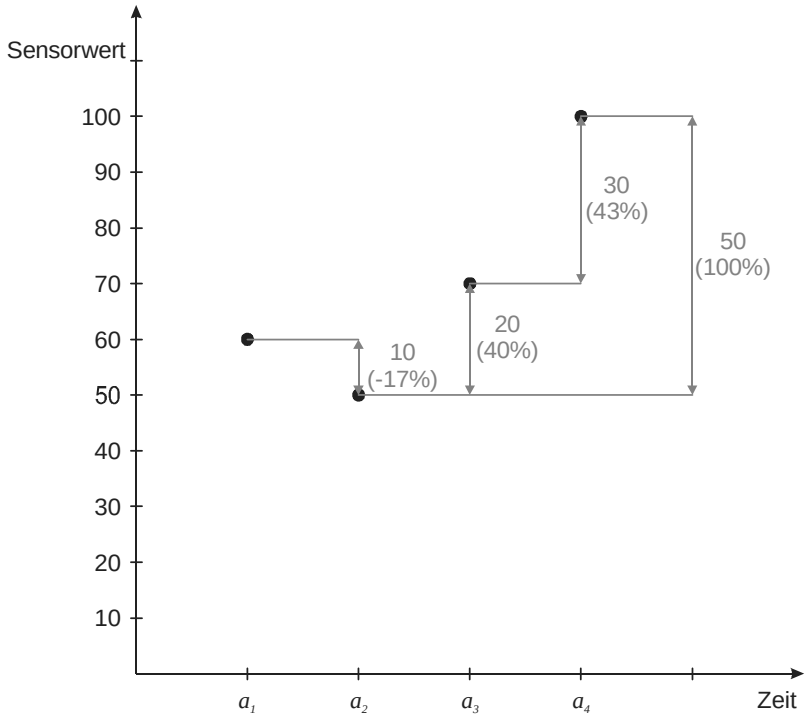


Abb. 5-7. Beispiel für die Korrelation von Sensorwerten

Beispiel 5-7:
Angenommen in einem IT-System wird die Netzwerkauslastung regelmäßig, z. B. alle fünf Minuten, durch Generierung eines Audit-Records dokumentiert. Als Indikator für eine Sicherheitsverletzung wird angesehen, wenn sich die Netzwerkauslastung innerhalb einer bestimmten Zeitspanne um mehr als 70% verändert. Der Anfang der Zeitspanne sei durch ein Ereignis vom Typ *B* und das Ende der Zeitspanne durch ein Ereignis vom Typ *C* definiert. Ereignisse vom Typ *A* dokumentieren die jeweils aktuelle Netzauslastung. Für eine solche Zeitspanne sei eine Ereignis-Trail $t_5:=\{b_1\ a_1\ a_2\ a_3\ a_4\ c_1\}$ erzeugt worden. Um die entsprechende Veränderung der Netzauslastung zu erkennen, ist es in diesem Beispiel nicht ausreichend, jeweils die beiden aufeinander folgenden Ereignisse a_n und $a_{(n+1)}$ miteinander zu vergleichen. Stattdessen müssen alle innerhalb der Zeitspanne generierten Ereignisse vom Typ *A* miteinander korreliert werden, wie Abb. 5-7 verdeutlicht. Dies kann durch Verwendung des Instanzkonsummodus *nicht-konsumierend* für die Schritte vom Typ *A* erreicht werden. Die Ereignisbeschreibung einer entsprechenden Signatur könnte wie folgt aussehen: E_6*:=NOT(C, (B; non-consuming A; non-consuming A))*. Darüber hinaus beschreibt die Kontextbedingung von E_6, dass die mit dem zweiten Schritt vom Typ *A* dokumentierte Netzauslastung um mehr als 70% über oder unter der des ersten Schrittes vom Typ *A* liegt.

In diesem Beispiel ist der Schrittinstanzkonsummodus für die Schritte vom Typ *A* durch die zu überprüfende Bedingung bestimmt, die hier eine Korrelation aller Kombinationen von zwei Schritten vom Typ *A* erforderlich macht, von denen einzelne anhand der Bedingung ausgewählt werden.

5.6.3 Schrittkombinationen

Der vorige Abschnitt betrachtete eine Situation, in der zunächst alle relevanten Schrittinstanzkombinationen geprüft werden, von denen dann anhand einer Bedingung einzelne ausgewählt werden, für die ein komplexes Ereignis auszulösen ist. In anderen Situationen stellt sich allein die Frage, für welche Schrittinstanzkombinationen komplexe Ereignisse auszulösen sind, wie im Folgenden diskutiert wird. Zur Erörterung wird eine Variante der Login-Attacke (vgl. Abschn. 4.2.1 und Beispiel 5-4 in Abschn. 5.4.2) verwendet.

Für die Login-Attacke sei folgende Anforderung zu erfüllen:

> *Es ist ein Alarm zu generieren, wenn sich ein Benutzer innerhalb von 30 Sekunden dreimal falsch angemeldet hat.*

Für den Fall, dass innerhalb von 30 Sekunden mehr als drei fehlerhafte Anmeldungen auftreten, lässt diese Aufgabenbeschreibung offen,

a) ob für verschiedene Kombinationen von drei fehlerhaften Anmeldungen innerhalb 30 Sekunden jeweils ein Alarm bzw. ein komplexes Ereignis ausgelöst werden soll oder
b) ob für die Tatsache, dass drei fehlerhafte Anmeldungen innerhalb von 30 Sekunden ein Alarm bzw. ein komplexes Ereignis ausgelöst werden soll.

Die nachfolgende Diskussion untersucht verschiedene konkretisierende Ereignisbeschreibungen. Die entsprechende Signatur sucht nach drei fehlerhaften Anmeldeversuchen, die durch Ereignisse des Typs *A* dargestellt werden. Die Überprüfung der Zeitspanne von 30 Sekunden, kann durch eine Kontextbedingung realisiert werden, die den Zeitstempel des ersten Ereignisses vom Typ *A* mit dem aller weiteren vergleicht. Zur Vereinfachung der Darstellung wird auf die Angabe dieser Bedingungen im Folgenden verzichtet. Dafür wird ein dedizierter Ereignistyp *B* verwendet, der das Eintreten eines beliebigen Ereignisses, einschließlich Ereignissen vom Typ *A*, zu einem Zeitpunkt, der mehr als 30 Sekunden nach dem ersten Ereignis vom Typ *A* liegt, repräsentiert. Der Ereignistyp *B* stellt somit Abbruchschritte für die Signaturen dar.

Diskutiert wird die Semantik der komplexen Ereignisse $E_7 := NOT(B, (non\text{-}consuming\ A;\ non\text{-}consuming\ A;\ non\text{-}consuming\ A))$ und $E_8 := NOT(B, (non\text{-}consuming\ A;\ consuming\ A;\ consuming\ A))$ bei der Abarbeitung der Ereignis-Trail $t_6 := \{a_1\ a_2\ a_3\ a_4\ a_5\ a_6\ b_1\}$.

E_7 beschreibt mit einer Schrittkette (vgl. Abschn. 5.4.2) alle Kombinationen (ohne Wiederholungen)[1] von drei Instanzen des Typs *A*. Die Anzahl der Instanzen vom Typ E_7, die durch eine Ereignis-Trail mit *n* Instanzen des Typs *A* ausgelöst werden, ist durch den Binomialkoeffizienten *n über 3* bestimmt, der im Fall von t_6 20 ist. Ein nicht-konsumierender Initialschritt ist erforderlich, damit nach fehlerhaften Anmeldungen eines Nutzers weitere partielle Instanzen für fehlerhafte Anmeldeversuche anderer Nutzer erzeugt werden. Abb. 5-8 (links) zeigt welche Instanzen aus t_6 durch E_7 miteinander korreliert werden.

Zunächst wird a_1 an eine neue partielle Instanz gebunden. Ereignis a_2 wird einerseits mit existierenden partiellen Instanzen korreliert, wobei die ursprüngliche partielle Instanz nicht konsumiert wird. Andererseits wird eine neue partielle Instanz erzeugt, an deren Initialschritt a_2 gebunden wird. Ereignis a_3 wird nicht-konsumierend an alle partiellen Instanzen gebunden, komplettiert die erste Instanz des komplexen Ereignisses und legt eine neue partielle Instanz an. Jede partielle Instanz bindet alle folgenden

[1] Im Sinne der Kombinatorik

Instanzen vom Typ *A* in einer neuen Instanz. Durch b_1 werden schließlich alle partiellen Instanzen, die a_1 gebunden haben, abgebrochen.

Da der initiale Schritt von E_8 vom Typ *A* und nicht-konsumierend ist, erzeugt jede Ereignisinstanz a_n aus t_6 eine neue partielle Instanz von E_8. Im Unterschied zu E_7 werden die partiellen Instanzen bei Bindung weiterer Schritte konsumiert. Insgesamt werden durch t_6 die folgenden vier Instanzen des Typs E_8 ausgelöst: $\{a_1\ a_2\ a_3\}$, $\{a_2\ a_3\ a_4\}$, $\{a_3\ a_4\ a_5\}$ und $\{a_4\ a_5\ a_6\}$. Außerdem verbleiben nach Abarbeitung von t_6 die partiellen Instanzen $\{a_5\ a_6\ \ldots\}$ und $\{a_6\ \ldots\}$. Abb. 5-8 (rechts) illustriert, welche Ereignisinstanzen aus t_6 durch E_8 miteinander korreliert werden. E_8 löst für jede mögliche Kombinationen (ohne Wiederholungen) von drei *aufeinander folgenden* Instanzen des Typs *A* eine komplexe Ereignisinstanz aus.

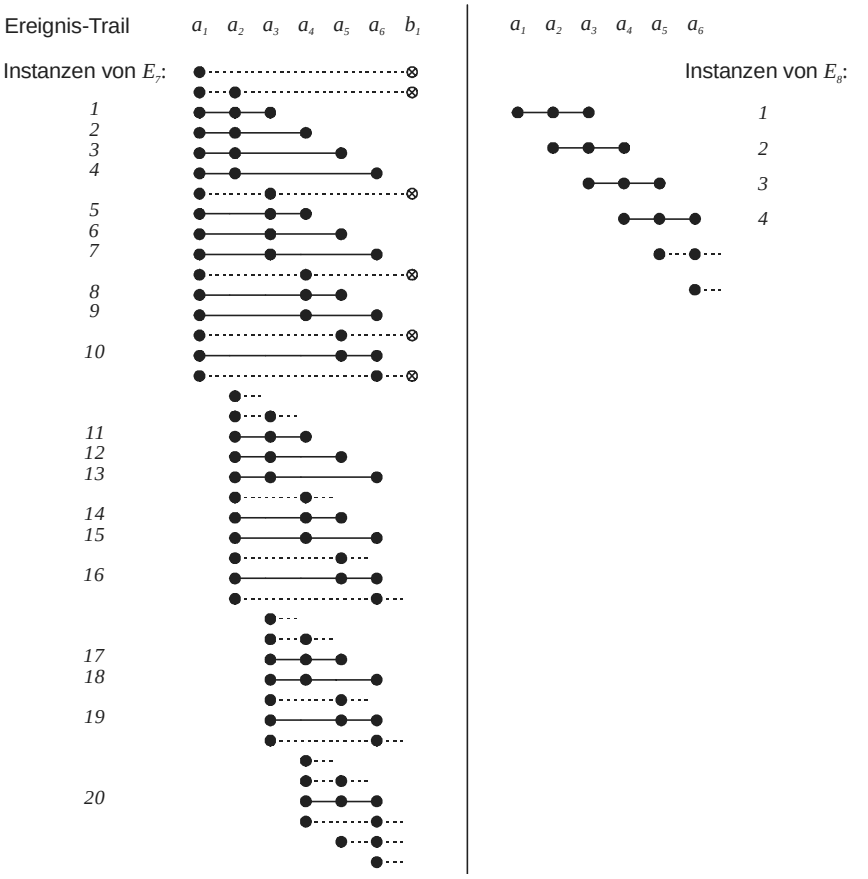


Abb. 5-8. Instanzen vom Typ E_7 (links) und E_8 (rechts)

5.7 Zusammenfassung

In diesem Kapitel wurde ausgehend von Erkenntnissen in der Theorie aktiver Datenbanken ein grundlegendes Modell für die Semantik von Angriffssignaturen entwickelt, das die semantischen Aspekte von Signaturen in die drei Dimensionen Ereignismuster, Schrittinstanzselektion und Schrittinstanzkonsum unterteilt. Während das Ereignismuster beschreibt, wann komplexe Ereignisse eintreten, definiert Schrittinstanzselektion, welche Ereignisse an ein komplexes Ereignis zu binden sind. Der Schrittinstanzkonsum steuert die Auswirkungen des Eintretens eines Schrittes auf die weitere Korrelation der bisherigen partiellen Instanz des komplexen Ereignisses. Tabelle 5-2 fasst die betrachten Ausprägungen der verschiedene semantischen Dimensionen bzw. ihrer Unteraspekte zusammen.

Tabelle 5-2. Übersicht der Aspekte des Semantikmodells

Dimensionen	Aspekte	Ausprägungen
Ereignis-muster	Typ und Reihenfolge	Sequenz
		Disjunktion
		Konjunktion
		Simultan
		Negation
	Häufigkeit	Genau
		Mindestens
		Höchstens
	Kontinuität	Kontinuierlich
		Nicht-kontinuierlich
	Neben-läufigkeit	Überlappend
		Nicht-überlappend
	Kontext-bedingungen	Intra-Ereignis-Bedingungen
		Inter-Ereignis-Bedingungen
Schrittinstanz-selektion		Erste
		Letzte
		Alle
Schrittinstanz-konsum		Konsumierend
		Nicht-konsumierend

Das vorgestellte Modell semantischer Aspekte ist nicht minimal. Beispielsweise können simultane Ereignisse auch durch Konjunktion der Ereignisse und Kontextbedingungen hinsichtlich der Zeitstempel der Ereignisse beschrieben werden. Die Ausprägung *nicht-kontinuierlich* des

Aspekts Kontinuität ist durch den Modus *kontinuierlich* und die Einführung entsprechender Negationen simulierbar. Zugunsten einer übersichtlichen Systematik und um die verschiedenen semantischen Charakteristika explizit zu betrachten, wurde hier auf Minimalität verzichtet.

Durch das entwickelte Modell werden systematisch verschiedene Aspekte von Angriffssignaturen betrachtet und gleichzeitig Anforderungen an Sprachen zur Beschreibung von Signaturen definiert. Darüber hinaus stellt es einen Kriterienkatalog für einen Vergleich der Ausdrucksstärke existierender Signaturbeschreibungssprachen hinsichtlich einzelner semantischer Aspekte dar. Es bildet die Grundlage für die in diesem Buch vorgestellte Modellierungs- und Beschreibungsmethode sowie die durchgeführten Analysen existierender Beschreibungsansätze.

6 Modell für Angriffssignaturen

Während sich Kapitel 5 der Frage widmet, *was* relevante Aspekte von Angriffsignaturen sind, fokussiert dieses und das darauf folgende Kapitel 7 die Frage, *wie* diese Aspekte von Signaturen erfasst und dargestellt werden können. In diesem Kapitel wird ein Ansatz zur Modellierung von Signaturen vorgestellt.

Im Folgenden wird zunächst das grundlegende Modellierungskonzept der Signaturnetze [FleMei02] vorgestellt und an einem Beispiel demonstriert. Eine formale Definition von Signaturnetzen schließt sich an. Darauf folgt eine detaillierte Darstellung, mit welchen Modellierungsmitteln die verschiedenen Aspekte von Angriffssignaturen dargestellt werden können. Verwandte Ansätze zur Modellierung von Signaturen werden diskutiert.

6.1 Signaturnetze – Das allgemeine Modell

Mit Signaturnetzen werden die bei der Durchführung einer Attacke erzeugten Ereignisse, auf deren Grundlage eine Erkennung erfolgen kann, modelliert. Die Modellierung von Signaturen umfasst die Charakterisierung von komplexen Ereignissen, insbesondere den Zusammenhängen zwischen den beobachteten Teilereignissen und Beziehungen zwischen Attributen der Teilereignisse. Im Zuge des sequentiellen oder nebenläufigen Auftretens von Teilereignissen einer komplexen Signatur durchläuft die entsprechende Signaturinstanz verschiedene Zustände. Sowohl die Zustände als auch Teilereignisse einer Signatur werden bei der Modellierung mit Signaturnetzen explizit erfasst.

Signaturnetze sind ein für den Anwendungsfall dedizierter Modellierungs-ansatz, der an Petrinetze [Bau90] und eine Reihe von Erweiterungen angelehnt ist, jedoch signifikante Unterschiede aufweist. Petrinetze erlauben aufgrund ihrer Zweiteilung des Netzes in Plätze und Transitionen die Modellierung komplexer kausaler Abhängigkeitssituationen [Nie03, Bau90]. Gleichzeitig erlaubt das Tokenspiel in Petrinetzen die Simulation und Analyse dynamischer Abläufe. Vergleichbar zu nicht-autonomen Petrinetzen [Da+92] wird in Signaturnetzen das Schalten von Transitionen mit

externen Ereignissen gekoppelt. Wird durch das Schalten von Transitionen mit einer Menge externer Ereignisse ein definierter Endzustand eines Signaturnetzes erreicht, dann dokumentiert diese Ereignismenge eine dem Netz entsprechende Sicherheitsverletzung. Eine solche Ereignismenge erfüllt das durch das Signaturnetz beschriebene Muster. Zur Modellierung und Simulation mit Signaturnetzen werden die vier Grundelemente Platz, Transition, Kante und Token verwendet.

Plätze beschreiben Teilzustände von Signaturen bzw. den korrespondieren Attacken. *Transitionen* spezifizieren die beobachtbaren Teilereignisse und mittels gerichteten *Kanten*, die entweder Plätze mit Transitionen oder Transitionen mit Plätzen verbinden, werden die kausalen Beziehungen zwischen den Teilereignissen ausgedrückt. Abb. 6-1 stellt beispielsweise ein vereinfachtes Modell einer Signatur dar, die eine Sequenz von zwei Ereignissen beschreibt.

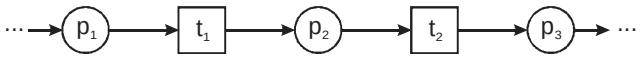


Abb. 6-1. Vereinfachtes Signaturnetz

Charakteristisch für die durch Signaturnetze beschriebenen Ereignismuster ist, dass der spezifizierten Ereignisreihenfolge eine *„follows“* anstatt einer *„immediately follows“* Semantik zugrunde liegt (vgl. [Ku95]). Das bedeutet, dass die in Abb. 6-1 spezifizierte Folge von Ereignissen der Typen *A* (Transition t_1) und *B* (Transition t_2) nicht nur durch direkt aufeinander folgende Ereignisse der Typen *A* und *B* erfüllt wird. Folgt auf ein Ereignis vom Typ *A* zunächst eine beliebige Zahl anderer Ereignisse bevor ein Ereignis vom Typ *B* folgt, so erfüllt diese Ereignismenge ebenfalls das in Abb. 6-1 beschriebene Muster.

Token erlauben die Repräsentation verschiedener Instanzen einer Signatur. Entsprechende Kontextinformationen werden analog gefärbten Petrinetzen [Jen92] in Tokenvariablen abgelegt. Token dokumentieren den Zustand einer solchen Instanz und enthalten außerdem Informationen über die bisher eingetretenen Teilereignisse, die zur weiteren Korrelation erforderlich sind.

Durch ein Signaturnetz wird ein Akzeptor beschrieben, dessen Anfangsplatz mit einem Token initialisiert wird. Durch das Tokenspiel wird die Menge der modellierten Muster beschrieben. Alle möglichen Ereignismengen, die Schaltfolgen auslösen, durch die Token einen definierten Finalplatz erreichen, bilden die akzeptierte Sprache des Netzes. Worte der Sprache eines Signaturnetzes entsprechen Manifestierungen der modellierten Sicherheitsverletzungen.

Es sei ausdrücklich darauf hingewiesen, dass trotz grundlegender Ähnlichkeiten zwischen Signaturnetzen und Petrinetzen, die Semantik beider Ansätze signifikant verschieden ist. Signaturnetze sollen eine möglichst intuitive Darstellung komplexer Ereignismuster sowie entsprechender Erkennungsabläufe erlauben. Aufgrund der Spezifik dieser Anwendung, bei der bspw. Nicht-Determinismus kontra-intuitiv ist, wird für Signaturnetze eine eigene Schaltregel eingeführt.

6.2 Modellierungselemente im Detail

Ein Platz repräsentiert einen Teilzustand von Signaturen bzw. der korrespondieren Attacken. Plätze beschreiben die relevanten Teilzustände des überwachten Systems, die zur Erkennung einer entsprechenden Sicherheitsverletzung anhand der beobachteten Ereignisse rekonstruiert werden. Kausale Beziehungen zwischen den modellierten Teilereignissen werden durch gerichtete Kanten zwischen Plätzen und Transitionen beschrieben. Um Merkmale von zwei Teilereignissen in Beziehung setzen zu können, können beim Schalten von Transitionen Variablen in den Token mit Werten z. B. Merkmalen des aufgetretenen Ereignisses belegt werden, die in Bedingungen der zweiten Transition referenziert werden. Im Folgenden werden die einzelnen Modellierungselemente von Signaturnetzen sowie die verwendete Schaltregel detailliert erläutert.

6.2.1 Plätze

Plätze, von denen eine Kante zu einer Transition führt, werden als *Vorplätze* dieser Transition bezeichnet. Entsprechend sind die *Nachplätze* einer Transition die Plätze zu denen eine Kante von der Transition führt. Es werden vier Platzarten unterschieden: *Initial-*, *Abbruch-*, *Final-* und *Interiorplatz*. Abb. 6-2 stellt die graphischen Symbole für die verschiedenen Platzarten dar. Jedes Signaturnetz besitzt eine nicht-leere Menge von Initialplätzen, die initial genau ein Token enthalten.

Abb. 6-2. Platzsymbole

Ein Signaturnetz, das genau eine Signatur modelliert, besitzt genau einen Finalplatz. Dieser repräsentiert, dass die beschriebene Attacke vollständig aufgetreten ist. Finalplätze sind Nachplätze von Transitionen, die finale Ereignisse (vgl. Definition 5-5) modellieren, und besitzen keine ausgehenden Kanten. Verschiedene Signaturen, insbesondere solche, die Varianten einer Attacke beschreiben, können identische Ereignisfolgen enthalten. Um die wiederholte Modellierung identischer Teile zu vermeiden, können die Signaturen in einem Signaturnetz modelliert werden. In diesem Fall kann das Netz mehrere Finalplätze besitzen. Token, die einen Finalplatz erreichen, werden verworfen.

Abbruchplätze beschreiben den Zustand, dass für eine Signaturinstanz ein abbrechendes Ereignis (vgl. Definition 5-5) eingetreten ist. Abbruchplätze besitzen keine ausgehenden Kanten. Prinzipiell ist in einem Signaturmodell maximal ein Abbruchplatz erforderlich. Um eine übersichtlichere graphische Modellierung zu ermöglichen, können jedoch mehrere Abbruchplätze modelliert werden. Token, die einen Abbruchplatz erreichen, werden verworfen.

Alle Plätze die weder Initial-, Final- noch Abbruchplätze sind, sind Interiorplätze. Sie besitzen mindesten eine eingehende Kante und mindestens eine ausgehende Kante.

Typischerweise besitzen Initialplätze keine eingehenden Kanten. Es sind jedoch Signaturnetze vorstellbar, in denen Eingangskanten von Initialplätzen verwendet werden. Beispielsweise kann dadurch gesteuert werden, dass Initialplätze bei Auftreten bestimmter Ereignisse (erneut) mit einem Token markiert werden.

6.2.2 Transitionen

Transitionen charakterisieren die einzelnen Teilereignisse einer Signatur. Eine Transition besitzt eine Beschriftung mit den folgenden Elementen:

- Ereignistyp,
- Intra-Ereignis-Bedingungen,
- Inter-Ereignis-Bedingungen,
- Variablenbindungen und
- Aktionen.

In Abb. 6-3 sind links die Beschriftungselemente veranschaulicht und rechts eine Beispielbeschriftung dargestellt. Der Ereignistyp, im Beispiel *FailedLogin* für eine fehlerhafte Anmeldung, identifiziert den Typ des Ereignisses, den diese Transition modelliert. Die Intra-Ereignis-Bedingungen

spezifizieren zusätzliche Einschränkungen bzgl. des modellierten Ereignisses, indem sie bestimmte Werte für Ereignismerkmale fordern (vgl. Abschn. 5.4.5). Im Beispiel wird für das Ereignismerkmal *user* der Wert *root* gefordert. Inter-Ereignisbedingungen setzen Merkmale verschiedener Teilereignisse in Beziehung. Diese Bedingungen enthalten Atome, die Merkmale des durch die Transition modellierten Ereignisses oder Tokenvariablen sein können. Im Beispiel wird die Übereinstimmung der Tokenvariable *h* mit dem Ereignismerkmal *hostname* gefordert. Darüber hinaus können Inter-Ereignis-Bedingungen Tokenvariablen und Konstanten in Beziehung setzen.

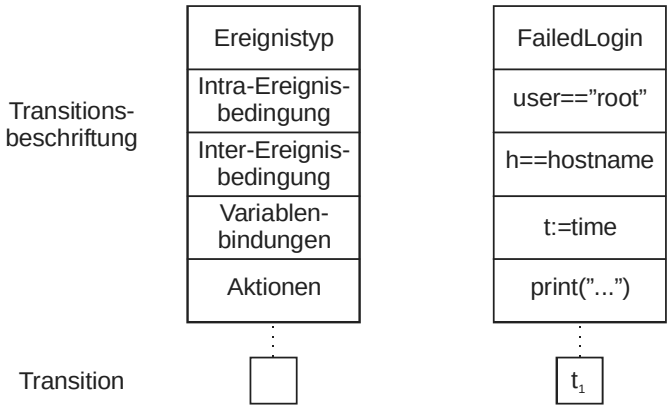


Abb. 6-3. Transitionssymbol, -beschriftung (links) und Beispiel (rechts)

Durch Variablenbindungen werden Merkmale des Ereignisses festgelegt, die beim Schalten der Transition in entsprechenden Variablen des Tokens aggregiert werden, um für Inter-Ereignis-Bedingungen und Aktionen anderer Transitionen zur Verfügung zu stehen. Die an Variablen zu bindenden Werte werden durch Angabe einer Funktion spezifiziert, die mit Konstanten, Ereignismerkmalen oder Tokenvariablen parametrisiert werden kann. Durch die Variablenbindung im Beispiel in Abb. 6-3 wird der Tokenvariablen *t* der Wert des Ereignismerkmals *time* zugewiesen. Wertebelegungen von Tokenvariablen können durch erneute Variablenbindung überschrieben werden.

Die einer Transition zugeordneten Aktionen, im Beispiel *print(“...“)*, enthalten Funktionen, die beim Schalten der Transition ausgeführt werden. Die Funktionen können mit Ereignismerkmalen, Tokenvariablen und Konstanten parametrisiert werden.

Um Konjunktionen von Ereignissen (vgl. Abschn. 5.4.1) ausdrücken zu können, wurden *spontane Transitionen* eingeführt, die unabhängig von ex-

ternen Ereignissen schalten können. Sie sind mit einem speziellen Ereignistyp (ε) gekennzeichnet und besitzen keine Intra-Ereignis-Bedingungen. Zur Abgrenzung wird im Folgenden zwischen spontanen und *regulären (nicht-spontanen) Transitionen* unterschieden.

6.2.3 Kanten

Durch gerichtete Kanten wird modelliert, welche Teilzustände Voraussetzung für Teilereignisse sind und welche Teilzustände durch das Auftreten eines Teilereignisses erreicht werden. Es wird zwischen Eingangs- und Ausgangskanten unterschieden. *Eingangskanten* führen von Plätzen zu Transitionen während *Ausgangskanten* von Transitionen zu Plätzen führen.

Alle Eingangskanten sind zusätzlich durch eines der Attribute konsumierend oder nicht-konsumierend gekennzeichnet, das den Konsum-Modus (vgl. Abschn. 5.6) des Transitionsereignisses bzgl. des entsprechenden Vorzustandes beschreibt. Konsumierende Kanten entsprechen den Standardkanten in Petrinetzen. Nicht-konsumierende Kanten sind mit Testkanten [Chri+93] in Petrinetzen vergleichbar. Abb. 6-4 stellt eine Transition t_1 dar, die mit ihrem Vorplatz p_1 über eine konsumierende Kante und mit Vorplatz p_2 über eine nicht-konsumierende Kante verbunden ist.

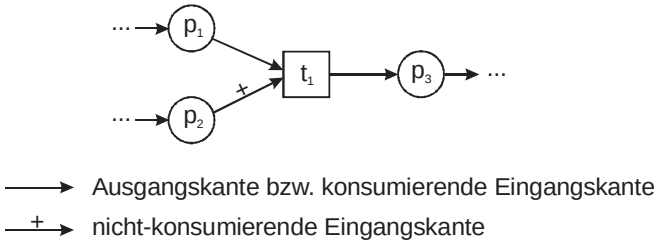


Abb. 6-4. Kantensymbole

6.2.4 Token

Ein Netz, das nur aus Plätzen, Transitionen und Kanten besteht, kann nur kausale Zusammenhänge einer Ereignismenge und deren Struktur beschreiben. Um Zustandsänderungen darstellen zu können, muss das Netz mit weiteren Elementen erweitert werden. Dies wird durch Markierung der Netze mit Token und Tokenfluss nach bestimmten Regeln erreicht.

Eine *Markierung* eines Netzes ordnet den Plätzen des Netzes eine natürliche Zahl von Token zu. Eine besondere Markierung ist die Anfangs-

markierung, die den Anfangszustand der modellierten Signatur beschreibt und nur den Initialplätzen je ein Token zuordnet.

Der Zustand einer Attacke kann durch mehrere Token im Signaturnetz repräsentiert werden. Dementsprechend beschreiben einzelne Token Teilzustände von Signaturinstanzen. Diese werden zum einen durch den Platz charakterisiert, auf dem sich ein Token befindet, und zum anderen durch Tokenvariablen, denen beim Passieren von Transitionen Werte zugewiesen werden können. In Analogie zu gefärbten Petrinetzen definieren die Variablenbelegungen von Tokenvariablen die Farbe eines Tokens.

In Abb. 6-5 ist ein Teilnetz mit verschiedenen Token und ein Ereignis dargestellt. An die Variablen des Tokens auf dem Initialplatz wurden noch keine Werte gebunden[1]. Die Transition t_1 ist mit dem Ereignistyp e_1 assoziiert. Sie fordert keine weiteren Einschränkungen bezüglich Ereignissen. Entsprechend enthalten die Intra- bzw. Inter-Ereignisbedingungen nur die Boolesche Konstante *true*. Beim Schalten der Transition wird die Variable *v* jedes passierenden Tokens mit dem Ereignismerkmal *oid* belegt. Der Transition sind keine Aktionen zugeordnet. Auf dem Nachplatz von t_1 befinden sich drei Token deren Variablen *v* mit verschiedenen Werten belegt wurden. Außerdem ist ein Ereignis vom Typ e_2 dargestellt, dessen Merkmal *oid* mit dem Wert *5* ausgeprägt ist. Transition t_2 ist mit dem Ereignistyp e_2 assoziiert und enthält als Intra-Ereignis-Bedingung die Konstante *true*. Die Inter-Ereignisbedingung der Transition t_2 fordert, dass die Werte des Ereignismerkmals *oid* und der durch t_1 belegten Tokenvariable *v* übereinstimmen. Dementsprechend erfüllt für das dargestellte Ereignis ein Token auf dem Vorplatz von t_2 diese Bedingung. Das heißt, eine Signaturinstanz schreitet durch das Ereignis fort. Der Transition t_2 sind keine Variablenbindungen und Aktionen zugeordnet.

Im vorliegenden Anwendungsfall von Petrinetzen machen mehrere identische Token auf einem Platz keinen Sinn. Dies entspräche der mehrfachen Repräsentation einer Ereignisfolge, die der Signatur entspricht. Alle identischen Token auf einem Platz würden identisch weiterfließen. Um diese Inkonsistenz zu vermeiden, werden identische Token auf einem Platz zu einem Token zusammengefasst. Dies findet seine natürliche Entsprechung darin, dass einem Platz eine *Menge* von Token zugeordnet ist.

Das Eintreffen mehrerer identischer Token auf einem Platz stellt gleichzeitig ein Indiz dafür dar, dass ein Signaturnetz reduziert werden kann und unnötige Elemente enthält. In Abb. 6-6 a) ist ein Beispiel für einen reduzierbaren Netzausschnitt skizziert. Hier erreichen für eine Folge von Ereignissen der Typen e_1 und e_2 zwei identische Token den Platz p_3. Ursache

[1] Tokenvariablen, denen noch keine Werte zugewiesen wurden, werden in den graphischen Darstellungen nicht wiedergegeben.

dafür ist, dass durch die Transitionen t_1 und t_2 sowie t_4 und t_5 identische Ereignisfolgen modelliert werden. Die reduzierte Variante des Netzausschnittes ist in Abb. 6-6 b) dargestellt.

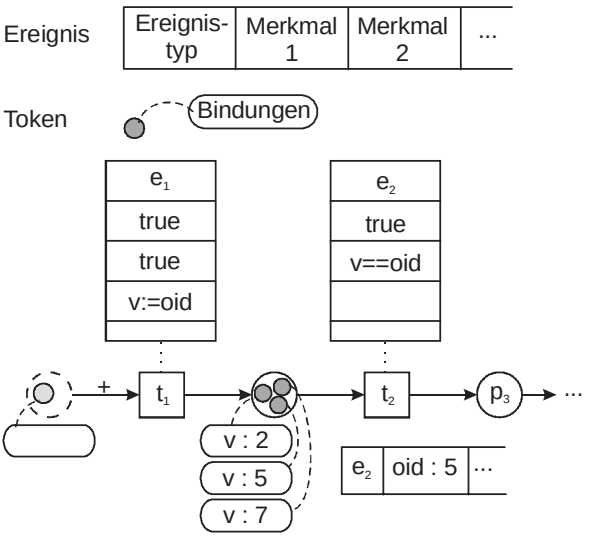


Abb. 6-5. Symbole für Token, Variablenbindungen und Ereignisse

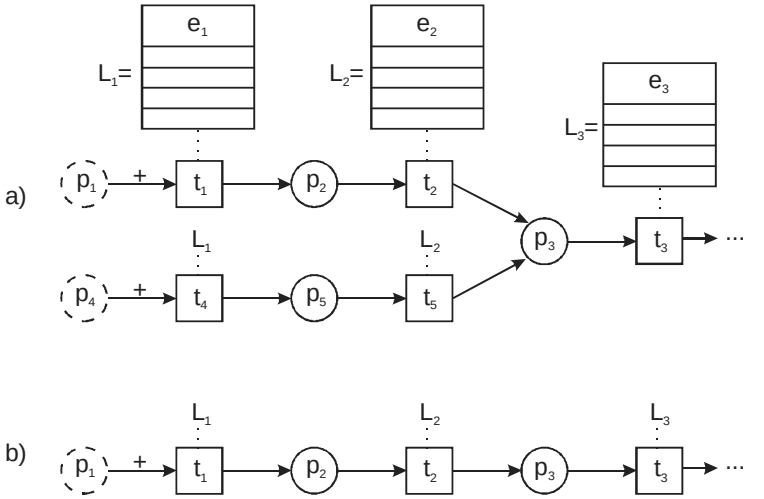


Abb. 6-6. Reduzierbares und reduziertes Teilnetz

6.2.5 Schaltregel

Transitionen von Signaturnetzen unterliegen einer Muss-Schaltregel mit maximaler Nebenläufigkeit. Das heißt alle schaltbereiten Transitionen schalten. Situationen, in denen Transitionen um Token auf einem gemeinsamen Vorplatz konkurrieren, werden deterministisch behandelt, indem ggf. Token dupliziert werden und alle konkurrierenden Transitionen schalten. Um die Schaltregel leichter formulieren zu können, werden verschiedene Begriffe definiert.

Eine *aktivierende Tokenmenge* einer Transition, ist eine Menge von Token, die genau ein Token in jedem Vorplatz der Transition enthält. Das heißt aktivierende Tokenmengen sind minimal. Wird ein Token aus einer aktivierenden Tokenmenge entfernt, dann ist sie nicht mehr aktivierend. Diese Festlegung auf genau ein Token in einem Vorplatz ist darin begründet, dass mehrere Token in einem Platz verschiedene Signaturinstanzen repräsentieren, die jeweils in einer separaten aktivierenden Tokenmenge behandelt werden.

Einem Token zugeordnete Variablen sind *initialisiert* und besitzen einen Wert, oder sie sind *uninitialisiert*. Eine Tokenmenge heißt *unifizierbar*, wenn sie keine zwei Token enthält, die der gleichen initialisierten Variablen verschiedene Werte zuordnen.

Jede aktivierende und unifzierbare Tokenmenge einer Transition bildet ein *unifiziertes Token*. Das unifizierte Token enthält alle Variablen aller Token der Tokenmenge. Alle in den Token initialisierten Variablen behalten ihre Wertebelegung. Alle anderen Variablen des unifizierten Tokens bleiben uninitialisiert.

Ein unifiziertes Token, das die Transitionsbedingungen, bei regulären Transitionen im Zusammenhang mit dem externen Ereignis erfüllt, heißt *auslösendes unifiziertes Token*. Es ist möglich, dass mehrere auslösende unifizierte Token für eine Transition *t* existieren, die jeweils einer Tokenmenge entsprechen. Die Vereinigung aller dieser Tokenmengen bildet die Menge der *Input-Token* von *t*, die im Folgenden zur Beschreibung des Tokenverbrauchs beim Schalten von Transitionen verwendet wird.

Eine konkrete Zuordnung von Token zu den Plätzen eines Signaturnetzes wird als *Markierung* des Signaturnetzes bezeichnet. Eine Markierung eines Signaturnetzes beschreibt zusammen mit dem Signaturnetz einen *Zustand*.

Schaltbereitschaft

Die Definition der Schaltbereitschaft unterscheidet sich für spontane und reguläre Transitionen. Im Unterschied zu regulären Transitionen sind spontane Transitionen unabhängig vom Vorliegen eines Ereignisses schalt-

bereit. Zur Charakterisierung dieses Unterschieds wird zwischen stabilen und instabilen Zuständen eines Signaturnetzes unterschieden. Solange spontane Transitionen eines Signaturnetzes schaltbereit sind, ist das Signaturnetz in einem *instabilen* Zustand. Entsprechend ist ein Signaturnetz bei einer Markierung *m* in einem *stabilen* Zustand, wenn keine spontanen Transitionen bei *m* schaltbereit sind.

Eine spontane Transition *t* eines Signaturnetzes *Sig* ist bei Markierung *m* schaltbereit, wenn

- mindestens ein auslösendes unifiziertes Token für die Transition *t* existiert.

Eine reguläre Transition *t* eines Signaturnetzes *Sig* ist bei Markierung *m* mit dem externen Ereignis *e* schaltbereit, wenn

- Signaturnetz *Sig* bei Markierung *m* in einem stabilen Zustand ist,
- der Ereignistyp von Transition *t* gleich dem Typ des externen Ereignis *e* ist,
- das externe Ereignis *e* die Intra-Ereignis-Bedingungen von Transitionen *t* erfüllt und
- mindestens ein auslösendes unifiziertes Token für die Transition *t* mit dem externen Ereignis *e* existiert.

Durch diese Festlegungen wird dem Schalten spontaner Transitionen gegenüber dem Schalten regulärer Transitionen Priorität gegeben. Wenn sich ein Signaturnetz in einem instabilen Zustand befindet, schalten spontane Transitionen solange, bis ein stabiler Zustand erreicht ist.

Schaltvorgang

Für den Schaltvorgang von Transitionen in Signaturnetzen gelten folgende Festlegungen:

- Jede schaltbereite (reguläre oder spontane) Transition *t* schaltet für alle auslösenden unifizierten Token.
- Tokenverbrauch: Jedes Inputtoken von Transition *t*, das auf einem Vorplatz von *t* liegt, der mit einer konsumierenden Kante mit *t* verbunden ist, wird gelöscht.
- Tokenerzeugung: Jedes auslösende unifizierte Token wird um die Variablenbindungen der Transition *t* erweitert, wobei uninitialisierte Tokenvariablen mit Werten belegt oder initialisierte Tokenvariablen mit neuen Werten belegt werden können. Auf jedem Nachplatz der Transition *t* wird eine Kopie von jedem erweiterten auslösenden unifizierten Token erzeugt.

- Die Aktionen von *t* werden ausgeführt.
- Alle bei einer Markierung *m* mit einem Ereignis *e* schaltbereiten regulären Transitionen schalten gleichzeitig und die durch Schalten erreichte Markierung *m'* wird für das Schalten regulärer Transitionen mit dem gleichen Ereignis *e* nicht berücksichtigt. Das heißt, dass Token, die durch Schalten regulärer Transitionen *t* mit Ereignis *e* erzeugt werden, für das Schalten von *t* oder anderer regulärer Transitionen *t'* beim gleichen Ereignis *e* nicht berücksichtigt werden. Entsprechend schaltet von einer Kette aufeinander folgender identischer Transitionen für ein Ereignis *e* maximal eine reguläre Transition[1].
- Wird durch das Schalten von spontanen Transitionen *t* bei *m* eine Markierung *m'* erreicht, bei der *t* oder andere spontane Transitionen *t'* schaltbereit sind, dann schalten *t* bzw. *t'* auch bei *m'*.

An dem in Abb. 6-7 dargestellten Beispiel werden die verschiedenen Konzepte und der Schaltvorgang verdeutlicht. Die Abbildung stellt ein Teilnetz in einem stabilen Zustand und dessen Markierung vor dem Schaltvorgang mit Ereignis e_1 dar. Für die Transition t_1 existieren die aktivierenden Tokenmengen $s_1=\{1, 3\}$, $s_2=\{1, 4\}$, $s_3=\{2, 3\}$ und $s_4=\{2, 4\}$, von denen s_1 und s_3 unifizierbar sind. Die Tokenmengen s_2 und s_4 sind nicht unifizierbar, da die enthaltenen Token der Variable *a* verschiedene Werte zuordnen. Des Weiteren existieren für die Transition t_4 die aktivierenden und unifizierbaren Tokenmengen $s_5=\{3\}$ und $s_6=\{4\}$. Das der Tokenmenge s_1 entsprechende unifizierte Token u_1 besitzt folgende Variablenbelegungen: *a : 1*, *b : 3*, *c : 1*. Das aus s_3 gebildete unifizierte Token u_2 besitzt die Variablenbelegungen *a : 2*, *b : 2*, *c : 1*. Die Tokenmengen s_5 und s_6 bilden die unifizierten Token u_3 und u_4 für die Transition t_4. Mit dem in der Abbildung dargestellten Ereignis erfüllen sowohl u_1 als auch u_2 die Transitionsbedingungen von t_1, so dass beide auslösende unifizierte Token darstellen. Von den unifizierten Token u_3 (s_5) und u_4 (s_6) erfüllt nur u_3 die Bedingungen der Transition t_4, so dass nur u_3 ein auslösendes unifiziertes Token darstellt. Die Menge der Inputtoken von t_1 ergibt sich aus der Vereinigung der Mengen s_1 und s_3, die Menge der Inputtoken von t_4 entspricht der Menge s_5. Da außerdem sowohl die Ereignistypen von t_1 als auch t_4 und dem dargestellten Ereignis übereinstimmen und das Netz in einem stabilen Zustand ist, da keine schaltbereite spontane Transition existiert, sind sowohl t_1 als auch t_4 schaltbereit.

[1] Dadurch wird u. a. für Schrittketten (vgl. Abschn. 5.4.2) ein kaskadiertes Schalten von Transitionen vermieden, das in einem folgenden Kapitel noch genauer betrachtet wird.

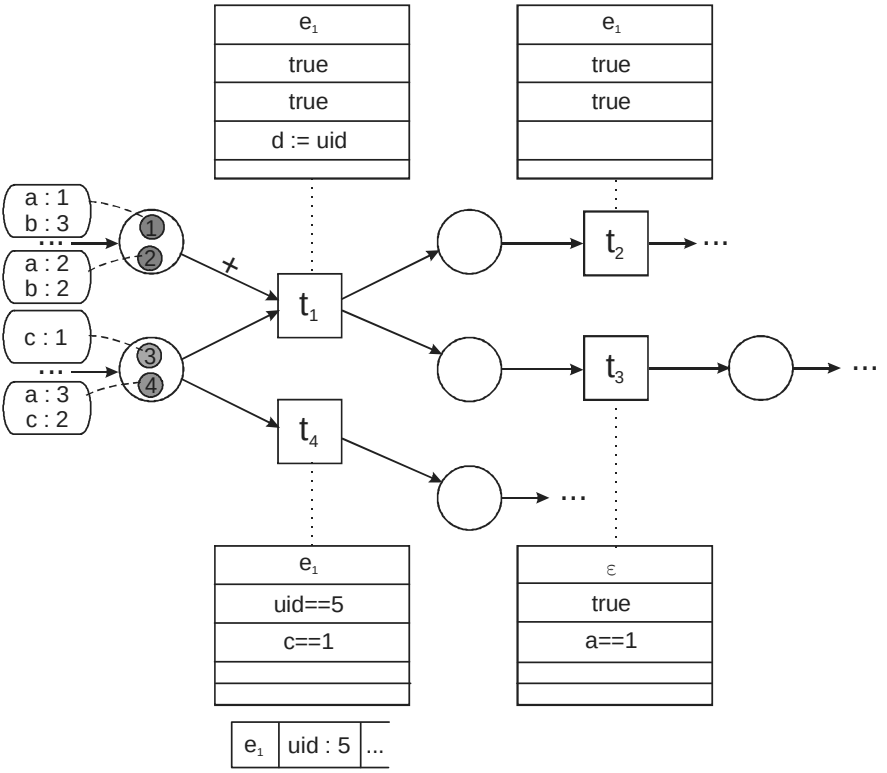


Abb. 6-7. Markiertes Teilnetz vor dem Schalten mit Ereignis e_1

Aufgrund der deterministischen Muss-Schaltregel mit maximaler Nebenläufigkeiten schaltet Transition t_1 für die beiden unifizierten Token u_1 und u_2 und Transition t_4 für u_3. Die Token u_1 und u_2 werden um die Variablenbindung *d : 5* erweitert. Auf jedem Nachplatz von t_1 wird eine Kopie der erweiterten Token u_1 und u_2, auf dem Nachplatz von t_4 eine Kopie von Token u_3 erstellt (vgl. Token *5*, *6*, *7*, *8* und *9* in Abb. 6-8). Die Menge der beim Tokenverbrauch zu betrachtenden Inputtoken von t_1 enthält die Token *1*, *2* und *3*. Token *3* ist gleichzeitig einziges Inputtoken von t_4. Da Token *3* auf einem Vorplatz von t_1 und t_4 liegt, der über konsumierende Eingangskanten mit t_1 und t_4 verbunden ist, wird Token *3* gelöscht. Es sei darauf hingewiesen, dass auch bei einer konsumierenden Kante zu der einen und einer nicht-konsumierenden Kante zu der anderen geschalteten Transition Token *3* gelöscht wird.

Nach dem Schalten von t_1 und t_4 existieren für die spontane Transition t_3 die aktivierenden und unifizierbaren Tokenmengen s_5*={7}* und s_6*={8}*. Da nur das der Menge s_5 entsprechende unifizierte Token die Transitionsbedingungen von t_3 (*a==1*) erfüllt, stellt nur dieses ein auslösendes unifi-

ziertes Token dar. Damit ist t_3 schaltbereit und schaltet. Da t_3 keine Variablenbindungen enthält, wird eine Kopie des durch s_5 repräsentierten unifizierten Tokens auf dem Nachplatz von t_3 erstellt (Token 9 in Abb. 6-9). Da das Inputtoken 7 von t_3 auf einem Platz liegt der mit t_3 über eine konsumierende Kante verbunden ist, wird das Token gelöscht.

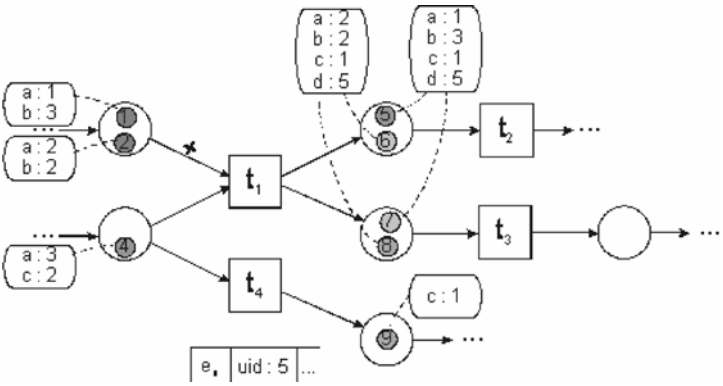


Abb. 6-8. Markiertes Teilnetz nach dem Schalten von t_1 und t_4

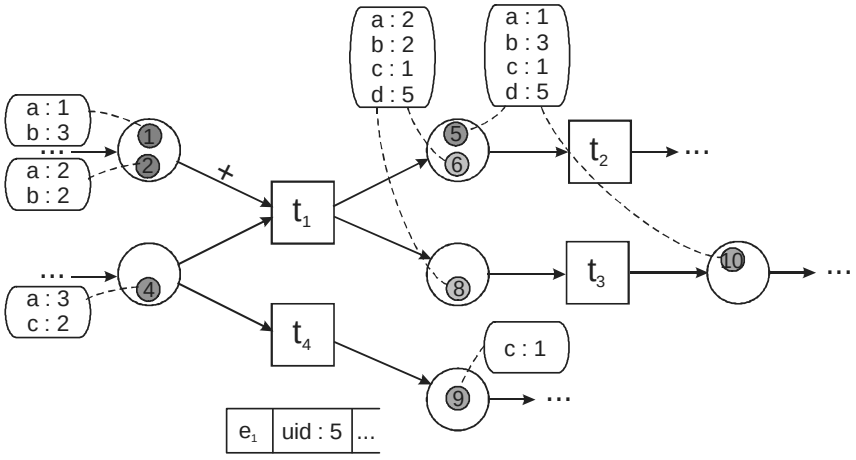


Abb. 6-9. Markiertes Teilnetz nach dem Schalten mit Ereignis e_1

Transition t_2 schaltet nicht. Zwar aktivieren die erstellten Token *5* und *6* die Transition und für e_1 sind alle Transitionsbedingungen erfüllt, jedoch wurden die aktivierenden Token erst durch das Schalten einer Transition

mit e_1 erzeugt[1]. Abb. 6-9 stellt den Netzausschnitt nach dem Schaltvorgang mit Ereignis e_1 dar.

6.2.6 Charakteristische Netztopologien

Unter Verwendung bestimmter struktureller Anordnungen von Netzknoten können charakteristische Eigenschaften von Ereignismengen repräsentiert werden. Im Folgenden werden verschiedene charakteristische Netztopologien diskutiert: Fork- und Join-Transitionen, Konflikte bzw. Alternativen, Rückwärtsalternativen und Schlingen. Außerdem wird die Verwendung der Strukturen in Netzen, die mehrere teilweise identische Signaturen zusammenfassen, dargestellt.

Modelle einzelner Signaturen

Transitionen mit mehr als einem Nachplatz, z. B. t_1 in Abb. 6-10, heißen *Fork-Transitionen*. Transitionen mit mehr als einem Vorplatz, z. B. t_4 in Abb. 6-10, werden *Join-Transitionen* genannt. Fork- und Join-Transitionen werden verwendet, um das nebenläufige Auftreten von Ereignissen innerhalb von Ereignismengen zu modellieren. Das Teilnetz in Abb. 6-10 beschreibt das Auftreten von den durch die Transitionen t_1, t_2, t_3 und t_4 repräsentierten Ereignissen. Die Fork-Transition t_1 modelliert in diesem Netz das letzte in fester Reihenfolge auftretende Ereignis vor den nebenläufigen durch t_2 und t_3 modellierten Ereignissen. Das durch t_1 repräsentierte Ereignis muss vor den Ereignissen von t_2 und t_3 auftreten. Hingegen sind die Ereignisse von t_2 und t_3 unabhängig voneinander (nebenläufig). Die Join-Transition t_4 repräsentiert das erste in fester zeitlicher Reihenfolge auftretende Ereignis nach den modellierten nebenläufigen Ereignissen. Das Ereignis von t_4 muss nach den Ereignissen von t_2 und t_3 auftreten. Ein Spezialfall sind spontane Join-Transitionen. Sie werden benutzt um die Konjunktion kausal unabhängiger Ereignisse zu beschreiben. Der Nachplatz der spontanen Transition t_4 beschreibt in diesem Fall den Zustand, dass die Ereignisse von t_2 und t_3 aufgetreten sind.

Plätze, zu denen Kanten von mehr als einer Transition führen, z. B. p_2 in Abb. 6-11, beschreiben Zustände, die durch alternative Ereignisfolgen erreicht werden können. Der durch p_2 repräsentierte Zustand kann sowohl durch Auftreten des Ereignisses von Transition t_4 als auch durch Auftreten des Ereignisses von t_5 erreicht werden.

---

[1] Ein Schalten von t_2 in dieser Situation würde einem kaskadierten Schalten von Transitionen entsprechen, das durch die Schaltregel ausgeschlossen wurde.

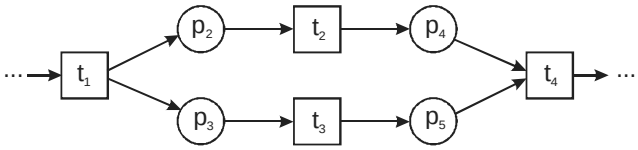


Abb. 6-10. Teilnetz mit Fork- und Join-Transition

In Analogie zur Terminologie der Petrinetze [Bau90] werden Transitionen, die einen gemeinsamen Vorplatz besitzen, als *Konflikt* oder *Alternative* bezeichnet. Transitionen in Signaturnetzen konkurrieren nur dann um Token auf gemeinsamen Vorplätzen, wenn die durch die Transitionsbeschriftungen spezifizierten Bedingungen für ein Ereignis von den gleichen Vorplatztoken erfüllt werden. Während Konflikte in klassischen Petrinetzen Nicht-Determinismus beschreiben, ist im Kontext der hier zugrunde liegenden Mustererkennung nur eine deterministische Interpretation sinnvoll.

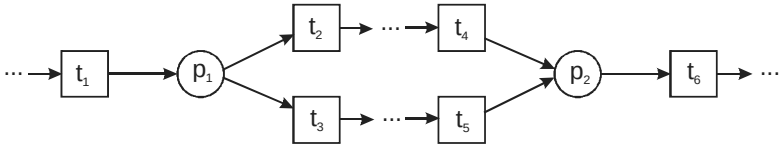


Abb. 6-11. Teilnetz mit Konflikt/Alternative und Rückwärtsalternative

Verwendung findet die Konfliktstruktur in Signaturnetzen um alternative Ereignisfolgen zu beschreiben. Beispielsweise repräsentiert das in Abb. 6-11 skizzierte Signaturnetz die durch die Schaltfolgen t_1, t_2, t_4, t_6 oder t_1, t_3, t_5, t_6 repräsentierten Ereignisfolgen. Es wird die Disjunktion beider Ereignisfolgen beschrieben, so dass auch Ereignisfolgen, die durch beide Transitionsfolgen dargestellt werden, möglich sind. In diesem Fall tritt ein Konflikt der Transitionen t_2 und t_3 um Token auf dem Platz p_1 auf. Die beschriebene Disjunktionssemantik, d. h., das Schalten beider Transitionen in diesem Fall, wird durch die vorgestellte Schaltregel sichergestellt. Unter der Voraussetzung, dass die durch die Transitionsbeschriftungen von t_1 und t_2 in Abb. 6-12 spezifizierten Bedingungen für ein Ereignis von den gleichen Vorplatztoken erfüllt werden, sind die in Abb. 6-12 dargestellten Teilnetze äquivalent.

Signaturen fordern häufig, dass eine bestimmte Anzahl von Ereignissen eines Typs aufgetreten sein muss. Die Anzahl der Ereignisse kann hierbei von konkreten Werten der Ereignismerkmale abhängen. Abb. 6-13 skizziert das Signaturnetz einer entsprechenden Signatur. Im Beispiel müssen vor dem Ereignis vom Typ e_2 so viele Ereignisse vom Typ e_1 aufgetreten sein, dass die kumulierten Werte des Merkmals *bits* den Wert *1000* über-

steigen. Derartige Forderungen können in Signaturnetzen über *Schlingen*, wie p_1 und t_2 in Abb. 6-13, und entsprechende Variablenbindungen an den Transitionen modelliert werden.

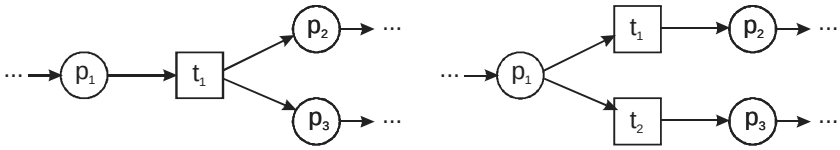


Abb. 6-12. Fork-Transition und potentiell äquivalente Konfliktstruktur

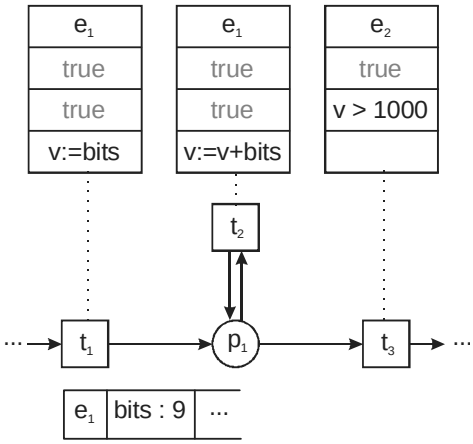


Abb. 6-13. Teilnetz mit Schlinge

Modelle mehrerer Signaturen

Existieren mehrere Signaturen, die in Teilen identisch sind, so können diese Signaturen durch ein Signaturnetz modelliert werden, wobei identische Teil nur einmal dargestellt werden. In einem solchen Multimodell treten Konflikte und Fork-Transitionen mit spezieller Funktion auf, wie im Folgenden erläutert wird.

Abb. 6-14 a) stellt zwei Signaturnetze für die Signaturen *S1* und *S2* dar, die zwei Transitionen (t_2 und t_6) mit identischer Beschriftung enthalten. In Abb. 6-14 b) ist das Signaturnetz dargestellt, in dem diese Transitionen und deren Nachplätze zusammengefasst wurden. Um die Semantik zu erhalten, müssen in diesem Netz die Schaltfolgen t_1, t_2, t_7 und t_5, t_2, t_3 ausgeschlossen werden. Dies wird erreicht, indem in einer Tokenvariable der Name der Signatur vermerkt und an den entsprechenden Transitionen überprüft wird.

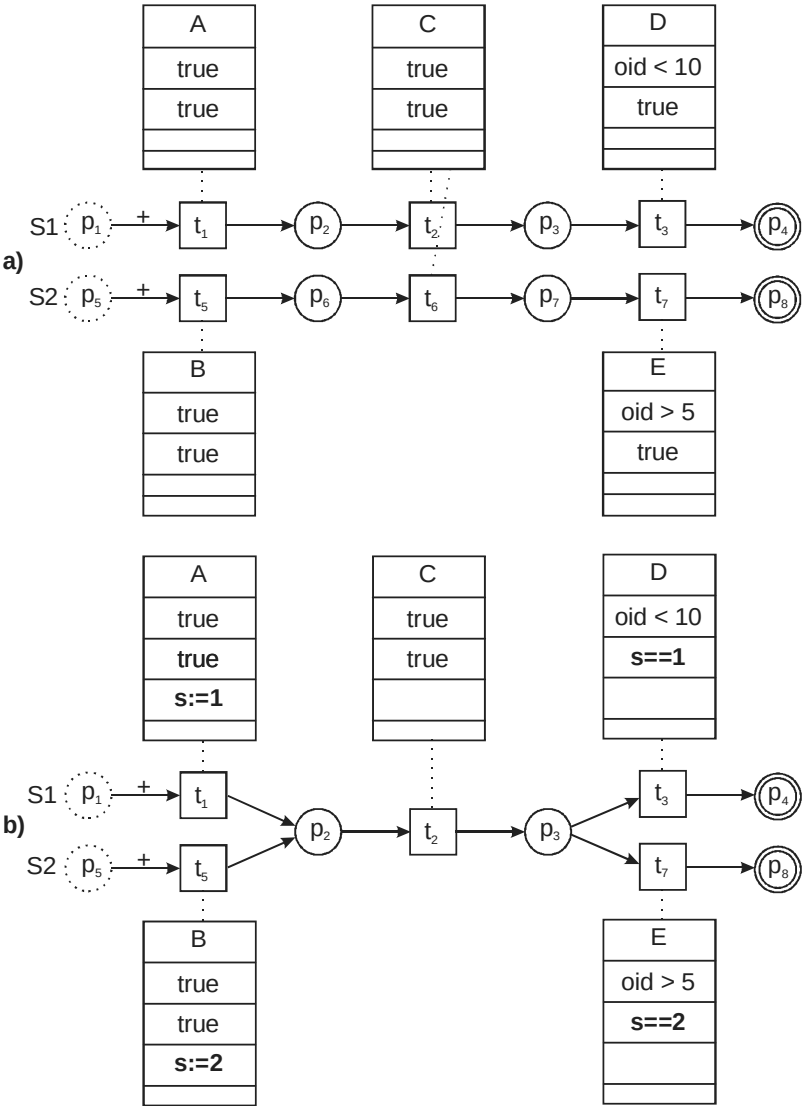


Abb. 6-14. Multimodell-Beispiel 1 (gleiche Teile)

Charakteristisch für das Signaturnetz in Abb. 6-14 b) ist, dass ein Token auf dem Platz p_3 genau eine Instanz entweder der Signatur *S1* oder *S2* darstellt. Entsprechend erfolgt die Verzweigung von Platz p_3 zu den signaturspezifischen Transitionen mit einer Konfliktstruktur. Ein echter Konflikt tritt hier aufgrund der Variablenbindung und Bedingungen nicht auf.

Abb. 6-15 a) stellt Signaturnetze für zwei weitere Signaturen *S1* und *S2* dar, deren letzte Transitionen t_3 und t_7 identisch beschriftet sind. Das Sig-

naturnetz für beide Signaturen in dem die identischen Elemente zusammengefasst wurden, ist in Abb. 6-15 b) dargestellt. Ein Token auf Platz p_3 repräsentiert wiederum genau eine Instanz entweder der Signatur *S1* oder *S2*.

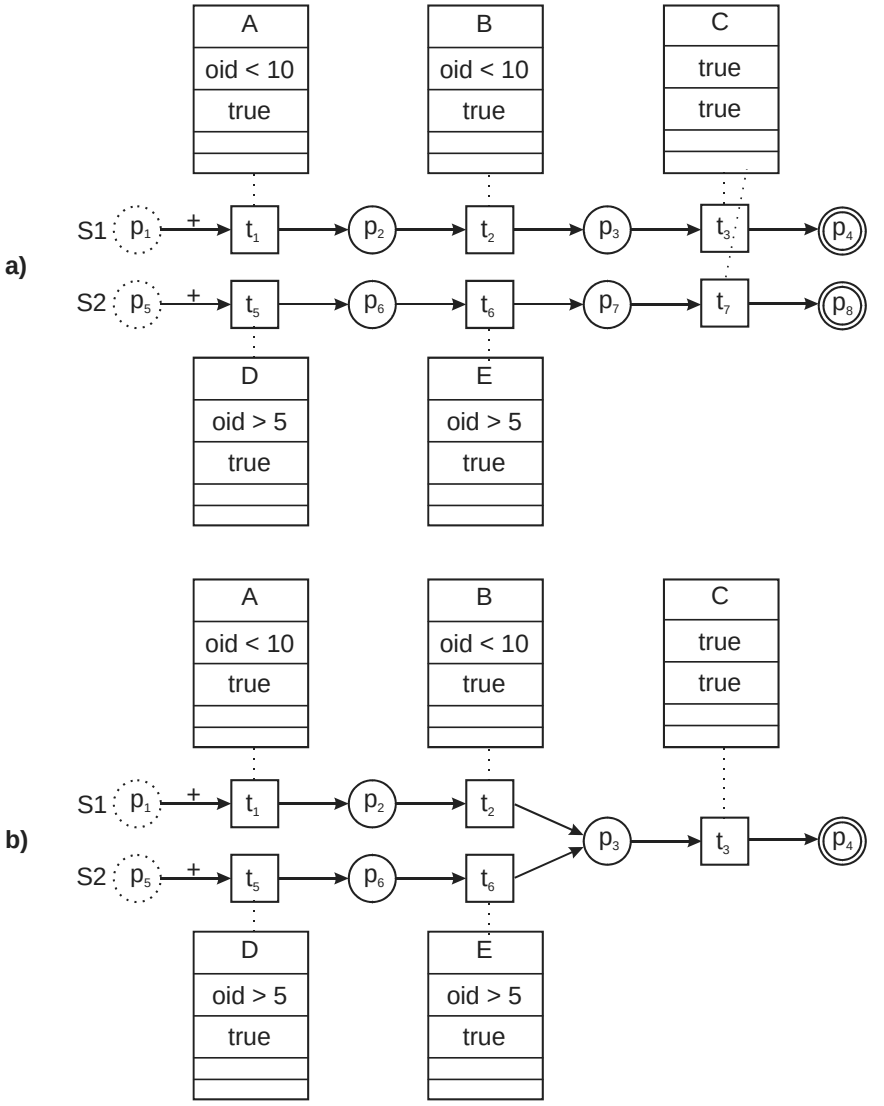


Abb. 6-15. Multimodell-Beispiel 2 (gleiche Enden)

In diesen beiden Multimodell-Beispielen wurden gleiche Transitionen verschiedener Signaturnetze einschließlich ihrer Vor- und Nachplätze zu-

sammengefasst. Anders verhält es sich in dem in Abb. 6-16 dargestellten Beispiel.

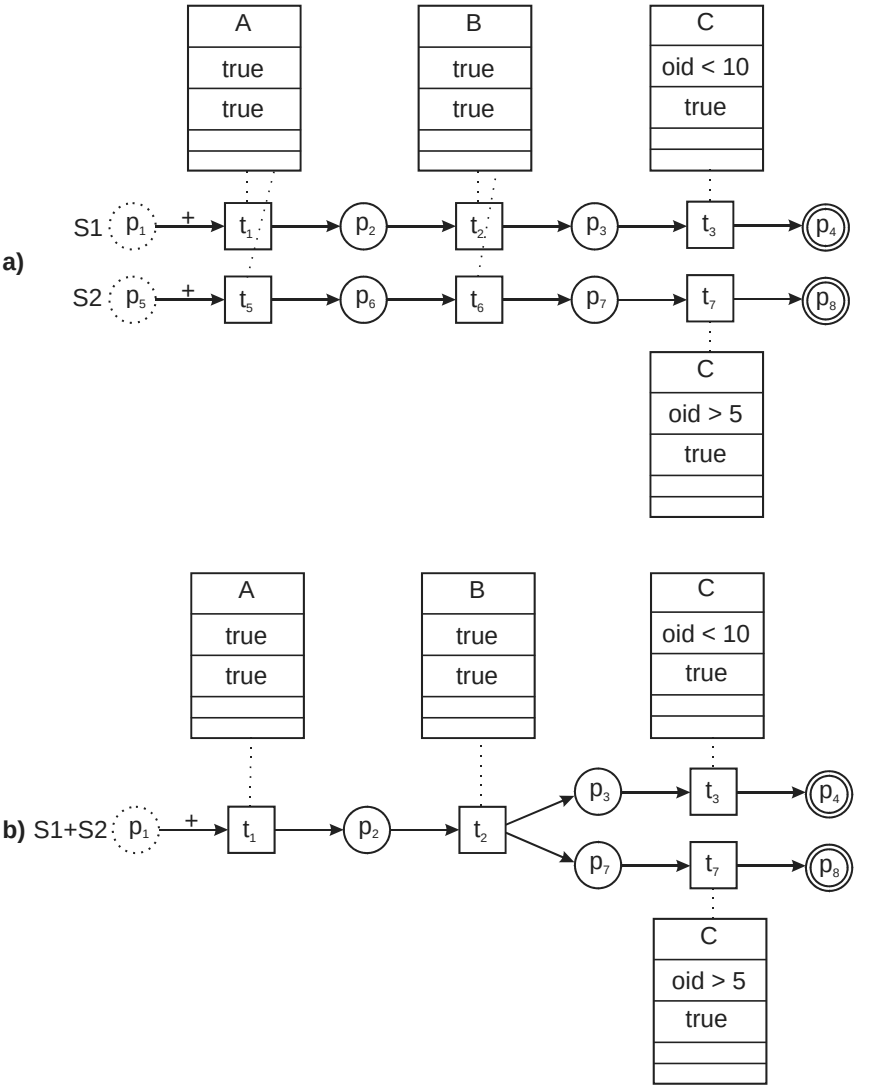


Abb. 6-16. Multimodell-Beispiel 3 (gleiche Anfänge)

Die Abb. 6-16 a) zeigt zwei Signaturnetze, deren Anfänge (p_1, t_1, p_2, t_2, p_3 und p_5, t_5, p_6, t_6, p_7) die gleichen Ereignisfolgen beschreiben. In Abb. 6-16 b) ist das zusammengefasste Signaturnetz für diese Signaturen dargestellt. Da hier die Anfänge der Signaturen zusammengefasst wurden, rep-

räsentiert ein Token auf dem Platz p_2 jeweils eine Signaturinstanz für *S1* und *S2*. Da beide Instanzen durch die signaturspezifischen Netzknoten (t_3, p_4 bzw. t_7, p_8) kausal unabhängig voneinander weiterverfolgt werden sollen, erfolgt die Verzweigung hier durch eine Fork-Transition, durch deren Schalten für jede Instanz ein separates Token erstellt wird. Entsprechend werden die Nachplätze der letzten gemeinsamen Transition t_2 bzw. t_6 nicht zusammengefasst.

6.3 Eine Beispielsimulation

Neben der Modellierung statischer Aspekte von Signaturen erlauben Signaturnetze auch die Simulation von Analyseabläufen zur Veranschaulichung dynamischer Abläufe. Im Folgenden wird beispielhaft eine Signatur für die Login-Attacke (vgl. Abschn. 4.2) modelliert und der Ablauf der Analyse einer Ereignismenge dargestellt. Die Signatur stellt Situationen dar, in denen drei fehlerhafte Anmeldeversuche, dokumentiert durch Ereignisse des Typs *fl* (failed login), eines Benutzers innerhalb einer Zeitspanne von weniger als 30 Sekunden aufgetreten sind. Im Beispiel dokumentieren Ereignisse des Typs *fl* u. a. den Namen des betroffenen Benutzerkontos (*uid*) sowie den Zeitpunkt des Anmeldeversuchs (*time*, in Sekunden). Das Signaturnetz mit der Anfangsmarkierung für diese Signatur ist in Abb. 6-17 dargestellt.

Durch die nicht-konsumierende Eingangskante von t_1 wird ausgedrückt, dass jede fehlerhafte Anmeldung eine neue Instanz der Signatur erzeugt. Über Bedingungen an den entsprechenden Transitionen t_2 und t_3 wird spezifiziert, dass alle drei Anmeldeversuche den gleichen Benutzernamen betreffen und innerhalb von 30 Sekunden auftreten. Beim Schalten von Transition t_3 wird die spezifizierte Aktion ausgelöst. Die Transitionen t_4 und t_5 verbrauchen Signaturinstanzen, die aufgrund einer Überschreitung der Zeitspanne die Signatur nicht mehr erfüllen können.

Abb. 6-18 stellt das Signaturereignisnetz sowie existierende Signaturinstanzen nach der Verarbeitung der drei dargestellten Ereignisse dar. Die drei Ereignisse dokumentieren fehlerhafte Anmeldeversuche von drei verschiedenen Benutzern. Die resultierenden Signaturinstanzen werden durch entsprechende Token auf Platz p_1 repräsentiert, die in Variablen sowohl den Benutzernamen (Variable *u*) als auch den Zeitpunkt der ersten fehlerhaften Anmeldung (Variable *s*) speichern.

Durch Verarbeitung des vierten Ereignisses (vgl. Abb. 6-19, unveränderte Variablenbelegungen sind grau dargestellt) wird eine vierte Signaturinstanz und damit ein neues Token auf Platz p_2 erzeugt. Gleichzeitig

entwickelt sich eine bereits existierende Signaturinstanz weiter und das entsprechende Token wird von p_2 durch Schalten von t_2 auf p_3 bewegt.

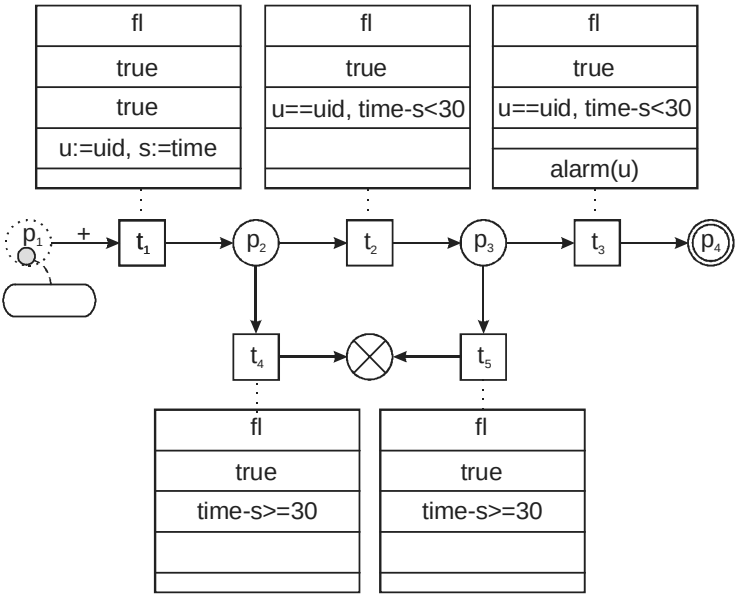


Abb. 6-17. Initial markiertes Signaturnetz

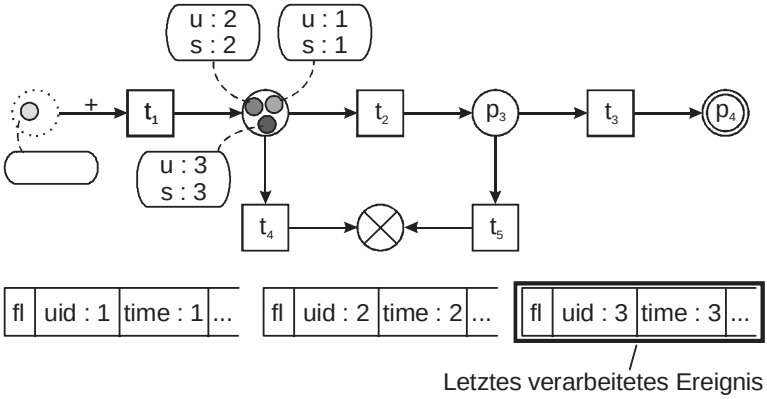


Abb. 6-18. Markiertes Signaturnetz nach drei Ereignissen

Bei Abarbeitung des fünften Ereignisses (vgl. Abb. 6-20) wiederholen sich die Abläufe, die bei Verarbeitung des vierten Ereignisses auftraten. Zusätzlich erreicht eine Signaturinstanz den Finalzustand der Signatur. Das zuvor auf Platz p_3 befindliche Token wird durch Schalten der Transition t_3 auf den Finalplatz des Netzes bewegt und von dort entfernt.

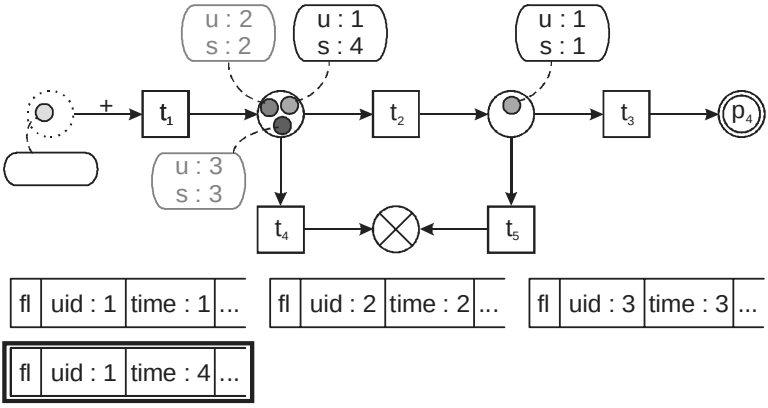


Abb. 6-19. Markiertes Signaturnetz nach vier Ereignissen

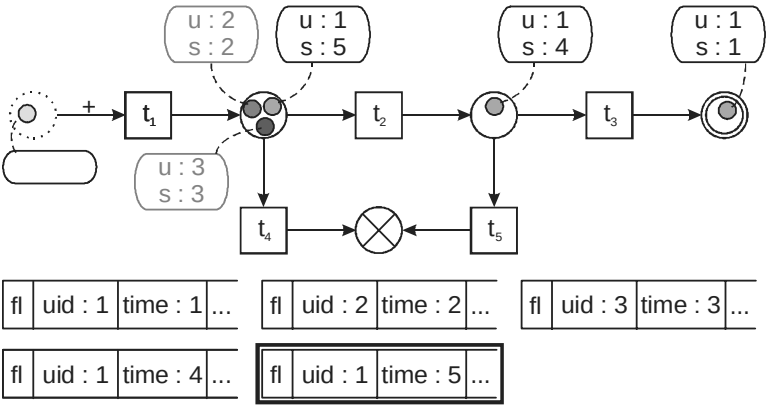


Abb. 6-20. Markiertes Signaturnetz nach fünf Ereignissen

Durch das sechste Ereignis wird wiederum, wie in Abb. 6-21 dargestellt, durch Schalten von Transition t_1 eine neue Signaturinstanz erstellt. Da der sechste Anmeldeversuch mehr als 30 Sekunden nach den vorhergehenden erfolgte, werden alle zuvor erstellten Signaturinstanzen abgebrochen. Das Schalten der entsprechenden Transitionen t_4 und t_5 bewegt die Token auf den Abbruchplatz von dem sie entfernt werden.

Obige Darstellung zeigte die Modellierung einer Beispielattacke als Signaturnetz und verdeutlichte wie mittels des Tokenspiels entsprechende Signaturmodelle getestet werden können. Durch eine solche Simulation der Analyseabläufe wird beispielsweise die Entwicklung der Tokenanzahl während der Analyse deutlich, die Rückschlüsse auf die zu erwartenden Speicherkosten zur Organisation der Instanzen der Signatur erlauben.

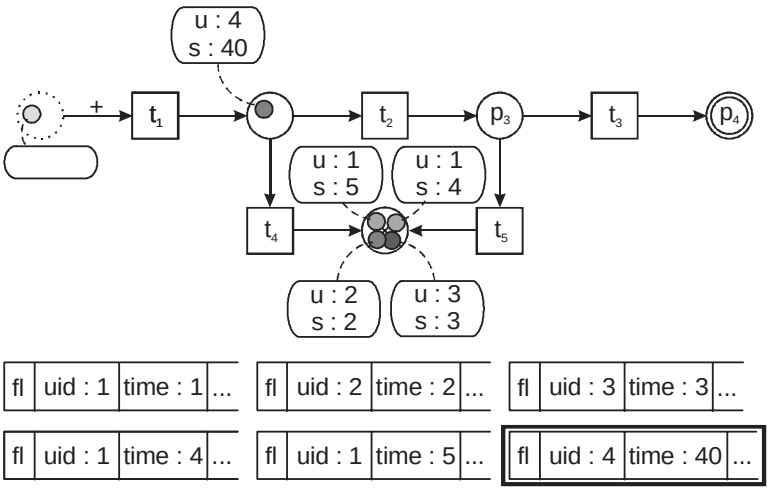


Abb. 6-21. Markiertes Signaturnetz nach sechs Ereignissen

6.4 Formale Definition eines Signaturnetzes

In diesem Abschnitt werden Signaturnetze, die in den vorigen Abschnitten informal beschrieben wurden, formal definiert. Dabei wird der Begriff *Situation* eingeführt, der ein Tripel bestehend aus einem Signaturnetz, dessen Markierung und einer Ereignis-Trail beschreibt. Definiert wird die operationale Semantik hinsichtlich der Änderung einer Situation bei der Verarbeitung der Ereignis-Trail, durch eine entsprechende Übergangsfunktion.

Ereignistypen, Merkmale und Ereignisse

F ist die Menge der Ereignismerkmale und W die Menge der Werte von Ereignismerkmalen einschließlich undef. $\mathrm{W}^{\mathrm{F}} = \{\mathrm{f} | \mathrm{f} : \mathrm{F} \to \mathrm{W}\}$ ist die Menge der totalen Funktionen von F nach W und repräsentiert die Menge der Belegungen der Merkmale. Ein $\mathrm{fb} \in \mathrm{W}^{\mathrm{F}}$ wird auch als *Merkmalsbelegung* bezeichnet. Jedes konkrete Ereignis e beschreibt eine Merkmalsbelegung $\mathrm{fb} \in \mathrm{W}^{\mathrm{F}}$ (e interpretiert die Merkmale). Σ sei die endliche Menge von Ereignistypen. Jedem Ereignistyp σ aus Σ ist eine feste Menge von Merkmalen $\mathrm{F}_\sigma \subseteq \mathrm{W}^{\mathrm{F}}$ zugeordnet, wobei F_σ nur solche Merkmale belegt, die zum Typ σ gehören. Dem speziellen Ereignistyp ε sind keine Merkmale zugeordnet, d. h., $\mathrm{F}_\varepsilon = \varnothing$.

Die Ereignis-Trail E ist eine durch $<_{\mathrm{trail}}$ (vgl. Abschn. 5.2) total geordnete Menge von Ereignissen. Für jedes Ereignis $\mathrm{e} \in \mathrm{E}$ bezeichne $\mathrm{Type(e)}$ den Typ des Ereignisses. V bezeichne die Menge aller Tokenvariablen. $\mathrm{W^V} = \{\mathrm{f} | \mathrm{f} : \mathrm{V} \rightarrow \mathrm{W}\}$ repräsentiert die Menge der Belegungen der Variablen. Ein $\mathrm{s} \in \mathrm{W^V}$ wird auch als *Speicherbelegung* bezeichnet.

Ausdrücke und Aktionen

$\mathrm{BExpr(F)}$ bezeichnet die Menge aller Booleschen Ausdrücke, die Merkmale der Menge F referenzieren können, d. h., entsprechende Merkmale enthalten können. Analog bezeichne $\mathrm{BExpr(F,V)}$ die Menge aller Booleschen Ausdrücke, die Merkmale aus F und Variablen aus V verwenden können. Für einen Ausdruck $\mathrm{b} \in \mathrm{BExpr(F)}$ und ein Ereignis $\mathrm{e} \in \mathrm{E}$ beschreibt $[\![\mathrm{b,e}]\!]$ die Bedeutung von b unter Interpretation e, also den Wert, der durch Auswertung des Ausdrucks b unter Verwendung der durch e gegebenen Merkmalsbelegung erhalten wird. Für einen Ausdruck $\mathrm{c} \in \mathrm{BExpr(F,V)}$, ein Ereignis e und eine Variablenbelegung $\mathrm{vb} \in \mathrm{W^V}$ repräsentiert $[\![\mathrm{c,e,vb}]\!]$ die Bedeutung des Ausdrucks c unter Interpretation e und vb.

$\mathrm{Expr(F,V)}$ bezeichnet die Menge von Ausdrücken die Variablen aus V und Merkmale aus F enthalten können. $\mathrm{AExpr(F,V)} = \mathrm{V} \times \mathrm{Expr(F,V)}$ ist die Menge von Zuweisungsausdrücken für Variablen $\mathrm{v} \in \mathrm{V}$. Die Auswertung eines Ausdrucks $\mathrm{ae} \in \mathrm{AExpr(F,V)}$ ist definiert durch die Funktion $\mathrm{eval} : \mathrm{Expr(F,V)} \times \mathrm{W^F} \times \mathrm{W^V} \rightarrow \mathrm{W}$. Für eine Speicherbelegung $\mathrm{s} \in \mathrm{W^V}$ beschreibt $\mathrm{s(v)}$ die Speicherbelegung an der Stelle v bzw. die Belegung der Variablen v bei Speicherbelegung s. Durch Auswertung eines Zuweisungsausdrucks wird eine Speicherbelegung $\mathrm{s} \in \mathrm{W^V}$ in eine Speicherbelegung $\mathrm{s'} \in \mathrm{W^V}$ überführt. Für ein Ereignis e beschreibt $\mathrm{s\langle ae,e\rangle(v)}$ die durch s gegebene Speicherbelegung der Variable $\mathrm{v} \in \mathrm{V}$ nach Auswertung des Zuweisungsausdrucks $\mathrm{ae} \in \mathrm{AExpr(F,V)}$ mit der durch ein Ereignis e gegebenen Merkmalsbelegung und ist wie folgt definiert:

$$\mathrm{s\langle ae,e\rangle(x)} = \begin{cases} \mathrm{eval(expr,e,s)} & \text{, falls } \mathrm{ae} = (\mathrm{y,expr}) \text{ und } \mathrm{x} = \mathrm{y} \\ \mathrm{s(x)} & \text{, sonst} \end{cases}$$

Die aus der Anwendung eines Zuweisungsausdrucks ae bei der durch ein Ereignis e gegebenen Merkmalsbelegung und der Speicherbelegung s resultierende Speicherbelegung $\mathrm{s'}$ wird beschrieben durch

$$\mathrm{s'} = \mathrm{s}\langle \mathrm{ae}, \mathrm{e} \rangle .$$

Die aus der Anwendung einer Folge von Zuweisungsausdrücken $\mathrm{AE} =< \mathrm{ae}_1, \mathrm{ae}_2, ... >$ bei der durch ein Ereignis e gegebenen Merkmalsbelegung und der Speicherbelegung s resultierende Speicherbelegung $\mathrm{s'}$ wird beschrieben durch

$$\mathrm{s'} = \mathrm{s}\dot{\langle}\mathrm{AE}, \mathrm{e}\dot{\rangle} \text{ mit}$$

$$\mathrm{s}\dot{\langle}\mathrm{AE}, \mathrm{e}\dot{\rangle} \coloneqq (\mathrm{s}\langle \mathrm{ae}_1, \mathrm{e} \rangle)\dot{\langle} < \mathrm{ae}_2, \mathrm{ae}_3, ... >, \mathrm{e}\dot{\rangle} \text{ und}$$

$$\mathrm{s}\dot{\langle} <>, \mathrm{e}\dot{\rangle} \coloneqq \mathrm{s}.$$

$\mathrm{Actions}$ bezeichne die Menge möglicher (Re-)Aktionen, die beim Schalten von Transitionen ausgelöst werden können.

Netz

Ein *Netz* ist ein endlicher bipartiter gerichteter Graph ohne isolierte Teilgraphen. Das Tupel $\mathrm{N} = [\mathrm{P}, \mathrm{T}, \mathrm{A}]$ wird Netz genannt, wenn gilt

- P ist eine endliche Menge von Plätzen,
- T ist eine endliche Menge von Transitionen,
- $\mathrm{P} \cap \mathrm{T} = \varnothing$ und $\mathrm{P}, \mathrm{T} \neq \varnothing$,
- A ist eine Kantenrelation, $\mathrm{A} \subseteq (\mathrm{P} \times \mathrm{T}) \cup (\mathrm{T} \times \mathrm{P})$ und
- es gibt keine isolierten Teilgraphen, also

$$\forall \mathrm{k}_1, \mathrm{k}_2 \in \mathrm{P} \cup \mathrm{T} : \left(\begin{array}{l} \left\{ \mathrm{k} \in \mathrm{P} \cup \mathrm{T} \middle| (\mathrm{k}, \mathrm{k}_1) \in \mathrm{A}^+ \vee (\mathrm{k}_1, \mathrm{k}) \in \mathrm{A}^+ \right\} \cap \\ \left\{ \mathrm{k} \in \mathrm{P} \cup \mathrm{T} \middle| (\mathrm{k}, \mathrm{k}_2) \in \mathrm{A}^+ \vee (\mathrm{k}_2, \mathrm{k}) \in \mathrm{A}^+ \right\} \neq \varnothing \end{array} \right),$$

wobei A^+ die transitive Hülle der Relation A bezeichne.

Ein Netz heißt *platzberandet*, wenn gilt:

$$\forall \mathrm{t} \in \mathrm{T} \exists \mathrm{p}_1, \mathrm{p}_2 \in \mathrm{P} : ((\mathrm{p}_1, \mathrm{t}) \in \mathrm{A} \wedge (\mathrm{t}, \mathrm{p}_2) \in \mathrm{A}) \wedge$$

$$\exists \mathrm{p} \in \mathrm{P} \forall \mathrm{t} \in \mathrm{T} : ((\mathrm{p}, \mathrm{t}) \notin \mathrm{A} \vee (\mathrm{t}, \mathrm{p}) \notin \mathrm{A}) \quad .$$

Signaturnetz

Ein *Signaturnetz* Sig für eine Menge E von Ereignissen der Typen aus $\sum$ ist ein 14-Tupel

$$\mathrm{Sig} = \begin{pmatrix} \mathrm{P}, \mathrm{P_I}, \mathrm{P_A}, \mathrm{P_F}, \\ \mathrm{T}, \mathrm{Type}, \mathrm{Bindings}, \mathrm{IntraCond}, \mathrm{InterCond}, \mathrm{Act}, \\ \mathrm{V}, \\ \mathrm{A}, \mathrm{Mode}, \\ \mathrm{m_0} \end{pmatrix}.$$

mit folgenden Eigenschaften:

- Das 3-Tupel $(\mathrm{P}, \mathrm{T}, \mathrm{A})$ ist ein platzberandetes Netz.
- $\mathrm{P_I} \subset \mathrm{P}, \mathrm{P_I} \neq \varnothing$ ist die nichtleere Menge der Initialplätze.
- $\mathrm{P_A} \subset \mathrm{P}$ ist die Menge der Abbruchplätze.
- $\mathrm{P_F} \subset \mathrm{P}, \mathrm{P_F} \neq \varnothing$ ist die nichtleere Menge der Finalplätze.
- $\mathrm{P_F} \cap \mathrm{P_A} = \varnothing$, $\mathrm{P_F} \cap \mathrm{P_I} = \varnothing$, $\mathrm{P_A} \cap \mathrm{P_I} = \varnothing$.
- Abbruch- und Finalplätze, und nur diese, besitzen keine ausgehenden Kanten: $\mathrm{P_F} \cup \mathrm{P_A} = \{\mathrm{p} \in \mathrm{P} \mid \forall \mathrm{t} \in \mathrm{T} : (\mathrm{p}, \mathrm{t}) \notin \mathrm{A}\}$.
- Type ist eine Funktion $\mathrm{Type} : \mathrm{T} \rightarrow \Sigma$, die jeder Transition einen Ereignistyp $\sigma \in \Sigma$ zuordnet.
- $\mathrm{Bindings}$ ist eine Funktion, die jeder Transition eine möglicherweise leere Folge von Zuweisungsausdrücken $< \mathrm{ae_1}, \mathrm{ae_2}, ... >$, $\mathrm{ae_i} \in \mathrm{AExpr(F, V)}$ zuordnet: $\mathrm{Bindings} : \mathrm{T} \rightarrow \mathrm{AExpr(F, V)}^*$

  $\mathrm{AExpr(F, V)}^*$ bezeichnet hierbei die Kleenesche Hülle über der Menge der Zuweisungsausdrücke, also die Menge aller endlichen Folgen von Ausdrücken, die sich durch beliebige Anordnung von Zuweisungsausdrücken ergeben.
- $\mathrm{IntraCond}$ ist eine Funktion, die jeder Transition eine Menge Boolescher Ausdrücke zuordnet, die die Intra-Ereignis-Bedingungen beschreiben: $\mathrm{IntraCond} : \mathrm{T} \rightarrow \wp(\mathrm{BExpr(F)})$.

  Das Symbol $\wp$ steht hierbei für die Potenzmenge.
- $\mathrm{InterCond}$ ist eine Funktion, die jeder Transition eine Menge Boolescher Ausdrücke zuweist, die die Inter-Ereignis-Bedingungen beschreiben: $\mathrm{InterCond} : \mathrm{T} \rightarrow \wp(\mathrm{BExpr\,(F, V)})$.
- Act ist eine Funktion, die einer Transition eine Menge von Aktionen zuordnet, die beim Schalten der Transition ausgeführt werden: $\mathrm{Act} : \mathrm{T} \rightarrow \wp(\mathrm{Actions})$.
- V ist eine endliche Menge von Variablen.

- Für die Menge der Kanten A sind die Teilmengen der Eingangskanten A_I und Ausgangskanten A_O wie folgt definiert: $A = A_I \cup A_O$, $A_I = \{a \in A | a \in (P \times T)\}$, $A_O = \{a \in A | a \in (T \times P)\}$
- $Mode$ ist eine Funktion, die jeder Eingangskante einen Konsummodus zuordnet: $Mode : A_I \rightarrow \{consuming, non\text{-}consuming\}$.
- Ein Token ist eine Variablenbelegung: $Token : V \rightarrow W$. Die Menge aller Token wird mit $TokenSet = W^V$ bezeichnet.
- m_0 – ist eine feste Anfangsmarkierung.
  - Eine Markierung m ist eine Relation, die Plätzen $p \in P$ Token zuordnet: $m \subseteq P \times TokenSet$. Eine Markierung beschreibt also eine Menge von Paaren $\{(p,k) | p \in P, k \in TokenSet)\}$. Das heißt u. a., dass identische Token $k \in TokenSet$ auf einem Platz $p \in P$ zusammengefasst werden. $M = \wp(P \times TokenSet)$ bezeichnet die Menge aller Markierungen.
  - $m_0 : P_I \rightarrow W^V$ ist die feste Anfangsmarkierung, die jedem Initialplatz $p \in P_I$ genau ein Token zuordnet, das alle Variablen mit dem Wert $undef$ belegt. Allen anderen Plätzen sind durch m_0 keine Token zugeordnet:
    $$m_0 = \{(p,k) | p \in P_I, k = \{(v, undef) \mid v \in V\}\}.$$

SIG bezeichne die Menge aller Signaturnetze. Zur Illustration der formalen Definition von Signaturnetzen ist in Abb. 6-22 dargestellt, wie das Signaturnetz aus Abb. 6-17 als Tupel repräsentiert werden kann.

Situation

Während für ein Signaturnetz Sig durch eine Markierung m, dessen Zustand (Sig, m) beschrieben wird, ist eine Situation $(Sig, m, e\omega)$ zusätzlich durch die vorliegende Ereignis-Trail gekennzeichnet. Eine Situation ist ein Tripel $(Sig, m, e\omega)$. Dabei ist e das nächste Ereignis in der Trail und ω die Folge der darauf folgenden Ereignisse.

Schaltbereitschaft von Transitionen in Situation (Sig, m, e)

$T_{spon} = \{t \in T : Type(t) = \varepsilon\}$ bezeichne die Menge aller spontanen Transitionen und $T_{reg} = \{t \in T : Type(t) \neq \varepsilon\}$ die Menge alle regulären Transitionen. $Pre(t) = \{p \in P | (p,t) \in A\}$ beschreibt die Menge der

Vorplätze und $\mathrm{Post}(t) = \{p \in P | (t,p) \in A\}$ die Menge der Nachplätze einer Transition t.

$$
\begin{aligned}
\Sigma &= \{\mathrm{fl}\} \qquad F = \{\mathrm{uid}, \mathrm{time}\} \\
P &= \{p_1, p_2, p_3, p_4, p_5\} \\
P_I &= \{p_1\} \qquad P_A = \{p_5\} \qquad P_F = \{p_4\} \\
T &= \{t_1, t_2, t_3, t_4, t_5\} \\
\mathrm{Type}(t) &= \mathrm{fl} \quad \text{für alle } t \in T \\
\mathrm{Bindings}(t_1) &= \{(u, \mathrm{uid}), (s, \mathrm{time})\} \\
\mathrm{Bindings}(t) &= \varnothing \qquad \text{für alle } t \neq t_1 \\
\mathrm{IntraCond}(t) &= \{\mathrm{true}\} \text{ für alle } t \in T \\
\mathrm{InterCond}(t_1) &= \{\mathrm{true}\} \\
\mathrm{InterCond}(t_2) &= \{u == \mathrm{uid}, \mathrm{time} - s < 30\} \\
\mathrm{InterCond}(t_3) &= \{u == \mathrm{uid}, \mathrm{time} - s < 30\} \\
\mathrm{InterCond}(t_4) &= \{\mathrm{time} - s >= 30\} \\
\mathrm{InterCond}(t_5) &= \{\mathrm{time} - s >= 30\} \\
\mathrm{Act}(t) &= \begin{cases} \{\mathrm{alarm}(u)\} & \text{für } t = t_3 \\ \varnothing & \text{sonst} \end{cases} \\
V &= \{u, s\} \\
A_I &= \{(p_1,t_1),(p_2,t_2),(p_2,t_4),(p_3,t_3),(p_3,t_5)\} \\
A_O &= \{(t_1,p_2),(t_2,p_3),(t_4,p_5),(t_5,p_5),(t_3,p_4)\} \\
A &= A_O \cup A_I \\
\mathrm{Mode}(a) &= \begin{cases} \mathrm{consuming} & \text{für } a \in \left\{ \begin{matrix} (p_2,t_2),(p_2,t_4), \\ (p_3,t_3),(p_3,t_5) \end{matrix} \right\} \\ \mathrm{non\text{-}consuming} & \text{für } a \in \{(p_1,t_1)\} \end{cases} \\
m_0 &= \{(p_1, \{(u, \mathrm{undef}), (s, \mathrm{undef})\})\}
\end{aligned}
$$

Abb. 6-22. Signaturnetz aus Abb. 6-17 als Tupel

Die Boolesche Funktion $\mathrm{unify} : \wp(\mathrm{TokenSet}) \rightarrow \mathrm{bool}$ prüft, ob eine gegebenen Tokenmenge unifizierbar ist, und ist wie folgt definiert:

$$\mathrm{unify}(\mathrm{tok}) = \begin{cases} \mathrm{true} \;\; \mathrm{falls} & \left(\begin{array}{l} \forall \mathrm{v} \in \mathrm{V}\, \forall \mathrm{k_a}, \mathrm{k_b} \in \mathrm{tok} \\ \left(\begin{array}{l} (\mathrm{v}, \mathrm{w_1}) \in \mathrm{k_a} \wedge (\mathrm{v}, \mathrm{w_2}) \in \mathrm{k_b} \Rightarrow \\ \mathrm{w_1} = \mathrm{w_2} \vee \mathrm{w_1} = \mathrm{undef} \vee \mathrm{w_2} = \mathrm{undef} \end{array} \right) \end{array} \right) \\ \mathrm{false} \;\; \mathrm{sonst} \end{cases}$$

Außerdem sei die Funktion $\mathrm{Unify} : \wp(\mathrm{TokenSet}) \rightarrow \mathrm{TokenSet}$, die eine unifizierbare Tokenmenge zu einem unifizierten Token zusammenfasst, wie folgt definiert:

$$\begin{aligned} \mathrm{Unify}(\mathrm{tok}) = \quad & \{(\mathrm{v}, \mathrm{w}) | \exists \mathrm{k} \in \mathrm{tok} : (\mathrm{v}, \mathrm{w}) \in \mathrm{k} \wedge \mathrm{w} \neq \mathrm{undef}\} \cup \\ & \{(\mathrm{v}, \mathrm{w}) | \forall \mathrm{k} \in \mathrm{tok} : (\mathrm{v}, \mathrm{w}) \in \mathrm{k} \wedge \mathrm{w} = \mathrm{undef}\} \end{aligned}$$

$\mathrm{FRT}^{\mathrm{TS}}_{(\mathrm{Sig},\mathrm{m},\mathrm{e})}$ *(Fire Ready Tokens)* ist die Menge von Tokenmengen, für die Transitionen aus der Menge TS in Situation $(\mathrm{Sig}, \mathrm{m}, \mathrm{e})$ schaltbereit sind:

$$\mathrm{FRT}^{\mathrm{TS}}_{(\mathrm{Sig},\mathrm{m},\mathrm{e})} =$$

$$\left\{ \mathrm{tok} \subseteq \mathrm{TokenSet} \left| \begin{array}{l} \exists \mathrm{t} \in \mathrm{TS} \\ \left(\begin{array}{l} (\mathrm{Type}(\mathrm{t}) = \mathrm{Type}(\mathrm{e}) \vee \mathrm{Type}(\mathrm{t}) = \varepsilon) \wedge \\ \forall \mathrm{k_1} \in \mathrm{tok}\, \exists \mathrm{p_1} \in \mathrm{Pre}(\mathrm{t})((\mathrm{p_1}, \mathrm{k_1}) \in \mathrm{m}) \wedge \\ \forall \mathrm{p_2} \in \mathrm{Pre}(\mathrm{t}) \exists \mathrm{k_2} \in \mathrm{tok}((\mathrm{p_2}, \mathrm{k_2}) \in \mathrm{m}) \wedge \\ \forall \mathrm{k_1}, \mathrm{k_2} \in \mathrm{tok}\, \forall \mathrm{p} \in \mathrm{P} \\ ((\mathrm{p}, \mathrm{k_1}) \in \mathrm{m} \wedge (\mathrm{p}, \mathrm{k_2}) \in \mathrm{m} \Rightarrow \mathrm{k_1} = \mathrm{k_2}) \wedge \\ \mathrm{unify}(\mathrm{tok}) \wedge \\ \forall \mathrm{b} \in \mathrm{IntraCond}(\mathrm{t})([\![\mathrm{b}, \mathrm{e}]\!]) \wedge \\ \forall \mathrm{c} \in \mathrm{InterCond}(\mathrm{t})([\![\mathrm{c}, \mathrm{e}, \mathrm{Unify}(\mathrm{tok})]\!]) \end{array} \right) \end{array} \right. \right\}$$

Eine spontane Transition $\mathrm{t} \in \mathrm{T_{spon}}$ des Signaturnetzes Sig ist in Situation $(\mathrm{Sig}, \mathrm{m}, \mathrm{e})$ schaltbereit wenn $\mathrm{FRT}^{\{\mathrm{t}\}}_{(\mathrm{Sig},\mathrm{m},\mathrm{e})} \neq \varnothing$.

Eine reguläre Transition $\mathrm{t} \in \mathrm{T_{reg}}$ des Signaturnetzes Sig ist in Situation $(\mathrm{Sig}, \mathrm{m}, \mathrm{e})$ schaltbereit wenn $\mathrm{FRT}^{\{\mathrm{t}\}}_{(\mathrm{Sig},\mathrm{m},\mathrm{e})} \neq \varnothing \wedge \mathrm{FRT}^{\mathrm{T_{spon}}}_{(\mathrm{Sig},\mathrm{m},\mathrm{e})} = \varnothing$

Schaltvorgang, Transitionsfunktionen und Situationsübergang

Die Aktualisierung der durch einen Token k gegebenen Speicherbelegung $\mathrm{k} \in \mathrm{W}^{\mathrm{V}}$ durch einen Zuweisungsausdruck $\mathrm{ae} \in \mathrm{AExpr}$ wird durch $\mathrm{k}\langle \mathrm{ae} \rangle$ beschrieben.

Tokenverbrauch

Die Menge $\mathrm{ConsumedMark}^{\mathrm{TS}}_{(\mathrm{Sig},\mathrm{m},\mathrm{e})}$ beschreibt die Menge der durch das Feuern von Transitionen aus der Menge TS des Signaturnetzes Sig bei Situation $(\mathrm{Sig},\mathrm{m},\mathrm{e})$ entfernten (Vorplatz-)Markierungselemente.

$$\mathrm{ConsumedMark}^{\mathrm{TS}}_{(\mathrm{Sig},\mathrm{m},\mathrm{e}\omega)} = \left\{ (\mathrm{p},\mathrm{k}) \;\middle|\; \exists \mathrm{t} \in \mathrm{TS}\, \exists \mathrm{tok} \in \mathrm{FRT}^{\{\mathrm{t}\}}_{(\mathrm{Sig},\mathrm{m},\mathrm{e}\omega)} \left(\begin{array}{c} \mathrm{k} \in \mathrm{tok} \quad \wedge \\ (\mathrm{p},\mathrm{k}) \in \mathrm{m} \quad \wedge \\ \mathrm{Mode}(\mathrm{p},\mathrm{t}) = \mathrm{consuming} \end{array} \right) \right\}$$

Tokenerzeugung

Die Menge $\mathrm{NewMark}^{\mathrm{TS}}_{(\mathrm{Sig},\mathrm{m},\mathrm{e})}$ beschreibt die Menge der durch das Schalten von Transitionen aus der Menge TS des Signaturnetzes Sig in Situation $(\mathrm{Sig},\mathrm{m},\mathrm{e})$ erstellten (Nachplatz-)Markierungselemente. Dies sind um die Bindungen der Transitionen erweiterte unifizierte Token, die auf den Nachplätzen der Transitionen erzeugt werden.

$$\mathrm{NewMark}^{\mathrm{TS}}_{(\mathrm{Sig},\mathrm{m},\mathrm{e})} =$$

$$\left\{ (\mathrm{p}, \mathrm{Unify}(\mathrm{tok}) \dot{\langle} \mathrm{Bindings}(\mathrm{t}),\mathrm{e} \dot{\rangle}) \;\middle|\; \begin{array}{c} \exists \mathrm{t} \in \mathrm{TS} \\ \exists \mathrm{tok} \in \mathrm{FRT}^{\{\mathrm{t}\}}_{(\mathrm{Sig},\mathrm{m},\mathrm{e})} \\ \left(\begin{array}{c} \mathrm{p} \in \mathrm{Post}(\mathrm{t}) \wedge \\ \mathrm{p} \notin \mathrm{P_A} \end{array} \right) \end{array} \right\}$$

In der Definition von $\mathrm{NewMark}^{\mathrm{TS}}_{(\mathrm{Sig},\mathrm{m},\mathrm{e})}$ werden durch $\mathrm{Unify}(\mathrm{tok})$ die bereits belegten Tokenvariablen des unifizierten Tokens übernommen. Auf die dadurch gegebene Speicherbelegung wird mit $\dot{\langle} \mathrm{Bindings}(\mathrm{t}),\mathrm{e} \dot{\rangle}$ die Folge von Zuweisungsausdrücken bei der durch e gegebenen Merkmalsbelegung angewendet.

Transitionsfunktion/Situationsübergang

Der Schaltvorgang von Transitionen des Signaturnetzes Sig bei Situation $(\mathrm{Sig}, \mathrm{m}, \mathrm{e}\omega)$ wird beschrieben durch den Situationsübergang $(\mathrm{Sig}, \mathrm{m}, \mathrm{e}\omega) \rightarrow (\mathrm{Sig}, \mathrm{m}', \omega)$ der unter Verwendung der Transitionsfunktionen $\mathrm{tf}^{\mathrm{Sig}} : \mathrm{M} \times \wp(\mathrm{F}) \rightarrow \mathrm{M}$ beschrieben wird. Ein Situationsübergang ist wie folgt definiert: $(\mathrm{Sig}, \mathrm{m}, \mathrm{e}\omega) \rightarrow (\mathrm{Sig}, \mathrm{m}', \omega) \Leftrightarrow \mathrm{m}' = \mathrm{tf}^{\mathrm{Sig}}(\mathrm{m}, \mathrm{e})$.

Das Schalten spontaner Transitionen wird durch die Funktion $\mathrm{tf}^{\mathrm{Sig}}_{\mathrm{spon}} : \mathrm{M} \times \wp(\mathrm{F}) \rightarrow \mathrm{M}$, das Schalten regulärer Transitionen durch die Funktion $\mathrm{tf}^{\mathrm{Sig}}_{\mathrm{reg}} : \mathrm{M} \times \wp(\mathrm{F}) \rightarrow \mathrm{M}$ definiert:

$$\mathrm{tf}^{\mathrm{Sig}}_{\mathrm{reg}}(\mathrm{m}, \mathrm{e}) = \left(\mathrm{m} - \mathrm{ConsumedMark}^{\mathrm{T_{reg}}}_{(\mathrm{Sig},\mathrm{m},\mathrm{e})}\right) \cup \mathrm{NewMark}^{\mathrm{T_{reg}}}_{(\mathrm{Sig},\mathrm{m},\mathrm{e})}$$

$$\mathrm{tf}^{\mathrm{Sig}}_{\mathrm{spon}}(\mathrm{m}, \mathrm{e}) = \begin{cases} \mathrm{m} & \text{falls } \mathrm{FRT}^{\mathrm{T_{spon}}}_{(\mathrm{Sig},\mathrm{m},\mathrm{e})} = \varnothing \\ \mathrm{tf}^{\mathrm{Sig}}_{\mathrm{spon}}\left(\left(\begin{array}{c}\left(\mathrm{m} - \mathrm{ConsumedMark}^{\mathrm{T_{spon}}}_{(\mathrm{Sig},\mathrm{m},\mathrm{e})}\right) \cup \\ \mathrm{NewMark}^{\mathrm{T_{spon}}}_{(\mathrm{Sig},\mathrm{m},\mathrm{e})}\end{array}\right), \mathrm{e}\right) & \text{sonst} \end{cases}$$

$$\mathrm{tf}^{\mathrm{Sig}}(\mathrm{m}, \mathrm{e}) = \begin{cases} \mathrm{tf}^{\mathrm{Sig}}_{\mathrm{spon}}(\mathrm{tf}^{\mathrm{Sig}}_{\mathrm{reg}}(\mathrm{tf}^{\mathrm{Sig}}_{\mathrm{spon}}(\mathrm{m}, \mathrm{e}), \mathrm{e}), \mathrm{e}) & \text{falls } \mathrm{m} = \mathrm{m}_0 \\ \mathrm{tf}^{\mathrm{Sig}}_{\mathrm{spon}}(\mathrm{tf}^{\mathrm{Sig}}_{\mathrm{reg}}(\mathrm{m}, \mathrm{e}), \mathrm{e}) & \text{sonst} \end{cases}$$

Für eine Markierung m und ein Ereignis-Trail $\mathrm{E} = \mathrm{e}\omega$ ist die Transitionsfunktion $\mathrm{TF}^{\mathrm{Sig}}_{\mathrm{i}}$ wie folgt definiert:

$$\mathrm{TF}^{\mathrm{Sig}}_{\mathrm{i}}(\mathrm{m}, \mathrm{e}\omega) = \begin{cases} \mathrm{TF}^{\mathrm{Sig}}_{\mathrm{i}-1}(\mathrm{tf}^{\mathrm{Sig}}(\mathrm{m}, \mathrm{e}), \omega) & \text{für alle } \mathrm{i} > 0 \\ \mathrm{m} & \text{für } \mathrm{i} = 0 \end{cases}$$

$\mathrm{M_f}$ bezeichne die Menge der Markierungen aus M, die mindestens einem Finalplatz mindestens ein Token zuordnen und ist wie folgt definiert:

$$\mathrm{M_f} = \left\{\mathrm{m} \in \mathrm{M} \middle| \exists \mathrm{p} \in \mathrm{P_f}, \mathrm{k} \in \mathrm{TokenSet}((\mathrm{p}, \mathrm{k}) \in \mathrm{m})\right\}$$

Die Menge der Situationsübergange von Markierung m_0 in eine Markierung $\mathrm{m_f} \in \mathrm{M_f}$ mit einer Ereignis-Trail ω wird beschrieben durch

$$(\mathrm{Sig}, \mathrm{m}_0, \omega) \xrightarrow{*} (\mathrm{Sig}, \mathrm{m_f}, \omega')$$

wobei ω' ein Suffix von ω ist.

Entsprechend ergibt sich die Bedeutung eines Signaturnetzes durch $[\![\mathrm{Sig}]\!] = \left\{\mathrm{E} | ((\mathrm{Sig}, \mathrm{m}_0, \mathrm{E}), (\mathrm{Sig}, \mathrm{m_f}, \omega)) \in \xrightarrow{*}\right\}$.

6.5 Ausdrucksstärke

In diesem Abschnitt wird die Ausdrucksstärke von Signaturnetzen hinsichtlich der in Kapitel 5 betrachteten semantischen Aspekte von Angriffssignaturen untersucht. Dazu wird diskutiert, wie die im Semantikmodell geforderten Aspekte mit Signaturnetzen charakterisiert werden können. Zum Zweck einer kompakteren Darstellung werden für die im Folgenden dargestellten Beispielsignaturnetze die in Abb. 6-23 dargestellten Transitionsbeschriftungen vereinbart.

6.5.1 Ereignismuster

Typ und Reihenfolge

Der Typ eines Ereignisses wird durch den Typ der Transition, die dieses Ereignis repräsentiert, spezifiziert. Zeitliche Zusammenhänge mehrerer Ereignisse werden durch die strukturelle Anordnung der Transitionen und Plätze im Signaturnetz charakterisiert. Eine Sequenz, wie sie durch das Ereignismuster *(A; B; C)* beschrieben wird, kann durch das in Abb. 6-24 dargestellte Signaturnetz beschrieben werden.

Abb. 6-23. Transitionsbeschriftungen

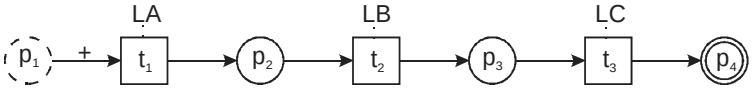


Abb. 6-24. Signaturnetz für das Ereignismuster *(A; B; C)* falls $<_{\text{time}} = <_{\text{trail}}$

Zu beachten ist hierbei, dass die strukturelle Anordnung der Knoten von Signaturnetzen tatsächlich die Reihenfolge von Ereignissen in der Ereignis-Trail (gegeben durch $<_{\text{trail}}$, vgl. Abschn. 5.2) spezifiziert und nicht die Eintrittsreihenfolge der Ereignisse (gegeben durch $<_{\text{time}}$). Falls $<_{\text{trail}}$ für die Menge der für ein Signaturnetz relevanten Ereignisse (in Abb. 6-24 alle Ereignisse der Typen *A*, *B* und *C*) eine Verschärfung von $<_{\text{time}}$ ist, dann wird die zeitliche Ordnung von Ereignissen durch die strukturelle Anord-

nung der Knoten nicht korrekt beschrieben. Beispielsweise werden durch Sequentialisierung der Ereignismenge *E2* in Abb. 5-1 auf Seite 48 die Ereignisse a_1 und e_1 in Relation ($<_{\text{trail}}$) gesetzt, die zeitlich nicht in Relation ($<_{\text{time}}$) stehen. In diesem Fall muss zusätzlich durch Transitionsbedingungen bezüglich der Ereigniszeitstempel die zeitliche Ordnung der Ereignisse spezifiziert werden. Abb. 6-25 zeigt das entsprechende Signaturnetz für die Sequenz *(A; B; C)*. Kann jedoch zum Beispiel aufgrund des Charakters der Ereignisse davon ausgegangen werden, dass auf der Menge der für ein Signaturnetz relevanten Ereignisse $<_{\text{time}}=<_{\text{trail}}$ gilt, dann wird die zeitliche Ordnung durch $<_{\text{trail}}$ korrekt wiedergegeben und kann allein durch strukturelle Anordnung von Signaturnetzknoten beschrieben werden.

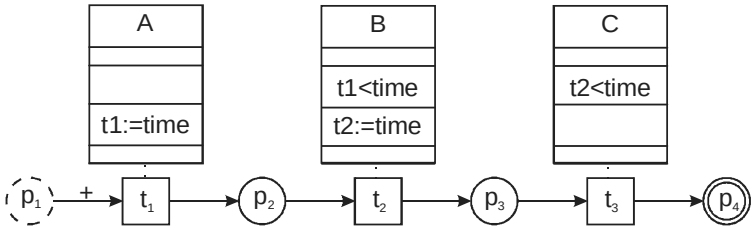


Abb. 6-25. Signaturnetz für das Ereignismuster *(A; B; C)*

Die Disjunktion von Ereignissen kann allein durch die strukturelle Anordnung der entsprechenden Transitionen im Signaturnetz ausgedrückt werden. Unter Verwendung von (Rückwärts-)Alternativen bzw. Konflikten beschreibt das in Abb. 6-26 dargestellte Signaturnetz das Ereignismuster *(A OR B OR C)*.

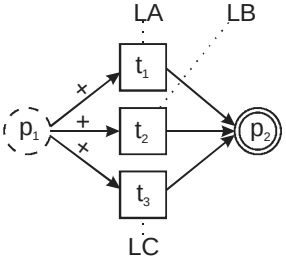


Abb. 6-26. Signaturnetz für Ereignismuster *(A OR B OR C)*

Mittels Fork- und Join-Transitionen erlauben Signaturnetze auf kompakte Weise die Spezifikation von Konjunktionen von Ereignissen. Unter Verwendung von spontanen Fork- und Join-Transitionen können kausal unabhängige Ereignismengen beschrieben werden. Die Abb. 6-27 zeigt

zwei verschiedene Signaturnetze für ein entsprechendes Ereignismuster *(A AND B AND C)*.

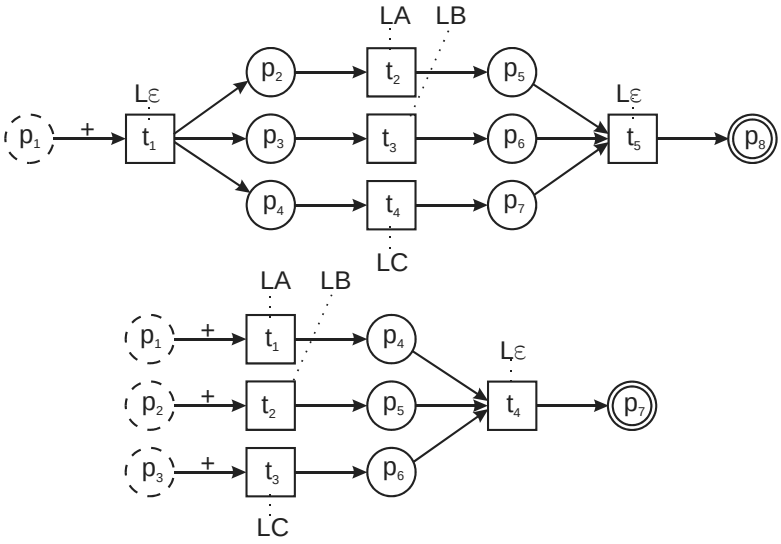


Abb. 6-27. Signaturnetze für Ereignismuster *(A AND B AND C)*

Alternativ kann dieses Ereignismuster durch eine Menge von Ereignissequenzen für alle möglichen Reihenfolgekombinationen der konjugierten Ereignisse beschrieben werden. Abb. 6-28 zeigt die zum Ereignismuster *(A AND B AND C)* äquivalente Menge von Signaturnetzen für Ereignissequenzen. Enthalten die in Abb. 6-27 verwendeten spontanen Join-Transition Inter-Ereignis-Bedingungen, Variablenbindungen oder Aktionen, dann sind diese der jeweils letzten Transition aller Ereignissequenzen in Abb. 6-28 hinzuzufügen.

Das simultane Auftreten von Ereignissen kann ebenfalls durch Join-Transitionen modelliert werden, die mittels einer Transitionsbedingung die Übereinstimmung der Eintrittszeitpunkte der Ereignisse fordern. Abb. 6-29 stellt ein entsprechendes Signaturnetz für das Ereignismuster *(A | B)* dar. In dem Signaturnetz sind zusätzlich die mit einem beliebigen Ereignis *C* assoziierten Transitionen t_4 und t_5 enthalten. Diese sorgen dafür, dass ein nach Ereignissen der Typen *A* oder *B* eintretendes Ereignis vom Typ *C* Token von den Plätzen p_1 bzw. p_2 verbraucht, die nicht-simultan eingetretene Ereignisse der Typen *A* und *B* repräsentieren.

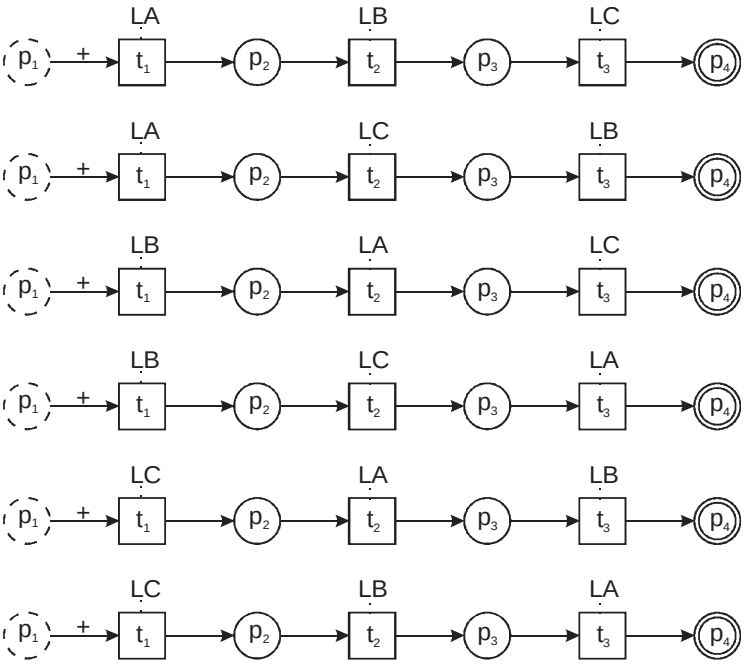


Abb. 6-28. Menge von Signaturnetzen für Ereignismuster *(A AND B AND C)*

Das simultane Auftreten von Ereignissen kann ebenfalls durch Join-Transitionen modelliert werden, die mittels einer Transitionsbedingung die Übereinstimmung der Eintrittszeitpunkte der Ereignisse fordern. Abb. 6-29 stellt ein entsprechendes Signaturnetz für das Ereignismuster *(A | B)* dar. In dem Signaturnetz sind zusätzlich die mit einem beliebigen Ereignis *C* assoziierten Transitionen t_4 und t_5 enthalten. Diese sorgen dafür, dass ein nach Ereignissen der Typen *A* oder *B* eintretendes Ereignis vom Typ *C* Token von den Plätzen p_1 bzw. p_2 verbraucht, die nicht-simultan eingetretene Ereignisse der Typen *A* und *B* repräsentieren.

Der Negationsoperator für Ereignismuster ist nur im Zusammenhang mit einem Zeitinterval sinnvoll (vgl. Abschn. 5.4.1). Wird der Zeitinterval durch eine Sequenz von Ereignissen definiert, muss bei der Modellierung dieser Sequenz berücksichtigt werden, ob $<_{\text{trail}}$ auf der Menge der involvierten Ereignisse eine Verschärfung von $<_{\text{time}}$ ist. Falls $<_{\text{time}} = <_{\text{trail}}$ gilt, dann stellt Abb. 6-30 ein korrektes Signaturnetz für das Ereignismuster *NOT(A, (B; C))* dar. Hierbei wird das Nicht-Auftreten eines Ereignisses *A* durch eine abrechende Transitionen modelliert.

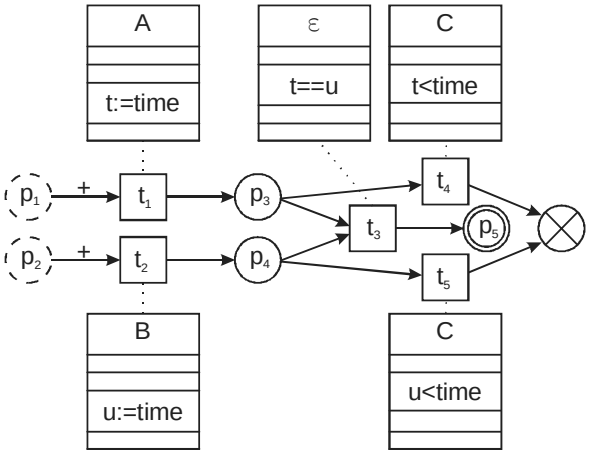


Abb. 6-29. Signaturnetz für Ereignismuster *(A | B)*

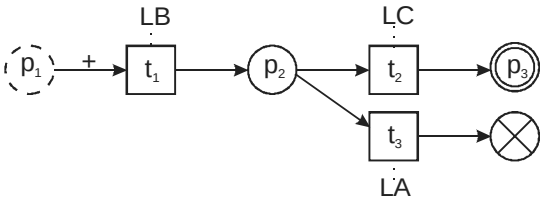


Abb. 6-30. Signaturnetz für Ereignismuster *NOT(A, (B; C))* falls $<_{\text{time}}=<_{\text{trail}}$

Für den allgemeinen Fall muss die zeitliche Reihenfolge der Ereignisse zusätzlich durch Transitionsbedingungen überprüft werden. Abb. 6-31 zeigt das entsprechende Signaturnetz für *NOT(A, (B; C))*. Eine Ereignisfolge *b, c* wird durch das Signaturnetz nicht akzeptiert, wenn die Ereignisse gleichzeitig eingetreten sind.

Im Folgenden wird davon ausgegangen, dass auf der Menge der jeweils betrachteten Ereignisse $<_{\text{time}}=<_{\text{trail}}$ gilt.

Häufigkeit

Die Häufigkeit von Ereignissen in einem Ereignismuster kann mittels Schlingen in Signaturnetzen modelliert werden. Ein Genau-Begrenzer kann durch eine Schlinge und eine Tokenvariable modelliert werden, in der die Anzahl der Ereignisse organisiert wird. Abb. 6-32 veranschaulicht das Signaturnetz für Ereignismuster *(A; (3) B; C)*. Die Variablenbindungen von Transition t_1 initialisieren die Zählvariable *n*, die durch Schalten von t_2 jeweils um eins erhöht wird. Die Transitionsbedingung von t_2 beschreibt, dass t_2 nur dann schaltet, wenn die Tokenvariable noch nicht den Wert *3*

erreicht hat. Mit der Bedingung von Transition t_3 wird gefordert, dass n den Wert *3* erreicht hat und somit drei Ereignisse vom Typ *B* aufgetreten sind. Folgt nach drei Ereignissen vom Typ *B* ein viertes, so verbraucht das Schalten von t_4 entsprechende Token.

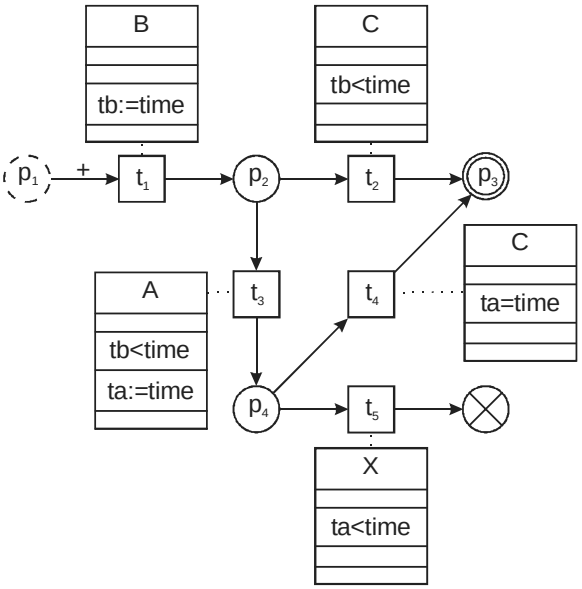


Abb. 6-31. Signaturnetz für Ereignismuster *NOT(A, (B; C))*

Die Modellierung der Mindestens-Begrenzung der Anzahl eines Ereignistyps ist in Abb. 6-33 exemplarisch für das Ereignismuster *(A; (3-)B; C)* dargestellt. Ein das Ereignismuster *(A; (-3)B; C)* modellierendes Signaturnetz ist in Abb. 6-34 dargestellt. Das mehr als dreimalige Auftreten von Ereignissen *B* wird durch die Abbruchtransition t_4 modelliert.

Kontinuität

Die Ausprägungen des Aspektes Kontinuität von Signaturen können mit Signaturnetzen dargestellt werden. Abb. 6-35 a) stellt das Signaturnetz für das Ereignismuster *(continuous A; B; C)* dar. Durch Ergänzung entsprechender Abbruchtransitionen kann auch das Ereignismuster *(non-continuous A; B; C)* modelliert werden (vgl. Abb. 6-35 b)).

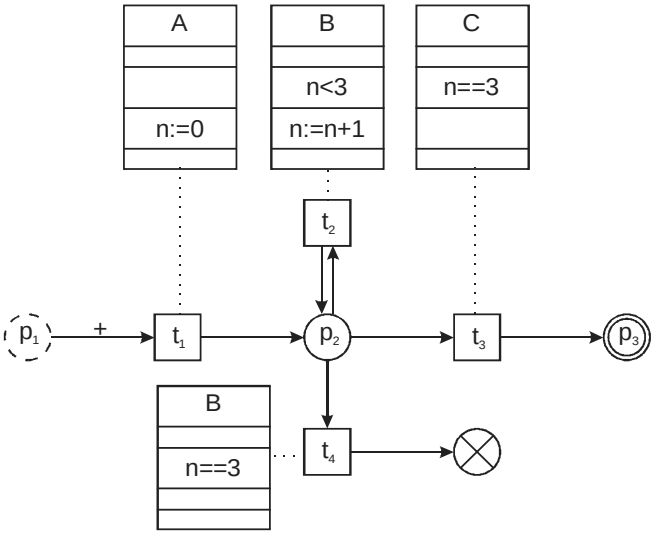


Abb. 6-32. Signaturnetz für Ereignismuster *(A; (3)B; C)*

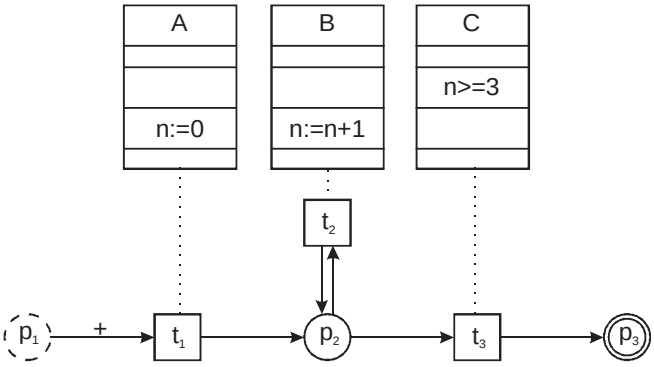


Abb. 6-33. Signaturnetz für Ereignismuster *(A; (3-)B; C)*

Nebenläufigkeit

Die Nebenläufigkeitsmodi *overlap* und *non-overlap* charakterisieren die Bedeutung von Ereignismustern, die komplexe (nicht Basis-) Ereignisse in Beziehung setzen. Signaturnetze werden zur Modellierung von Zusammenhängen zwischen Basisereignissen verwendet. Dementsprechend ist eine Unterscheidung der Nebenläufigkeitsmodi nicht erforderlich. In Kapitel 7 wird eine auf Signaturnetzen basierende Sprache zur Beschreibung von Signaturen vorgestellt, die Modularisierung unterstützt. Für eine Beschreibung von Zusammenhängen zwischen Modulen werden entspre-

chende Sprachmittel eingeführt, die eine Interpretation der Nebenläufigkeit festlegen.

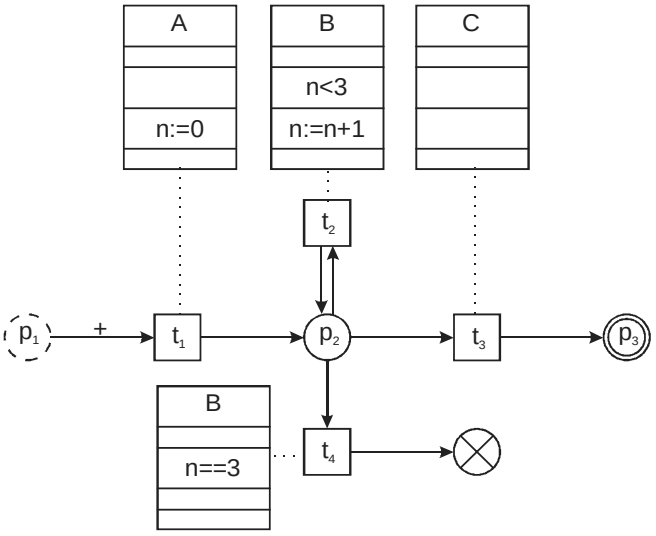


Abb. 6-34. Signaturnetz für Ereignismuster *(A; (-3)B; C)*

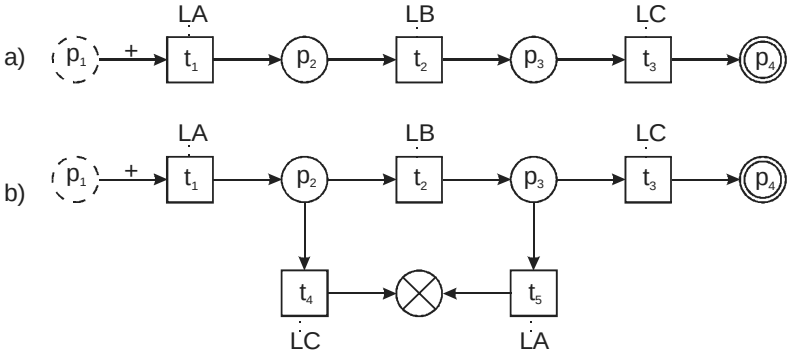


Abb. 6-35. Kontinuitätsmodi in Signaturnetzen

Kontextbedingungen

Einschränkungen für aufgetretene Ereignisse hinsichtlich des Kontextes können unter Verwendung von Variablenbindungen und Bedingungen an Transitionen spezifiziert werden. Zur Beschreibung einer Inter-Ereignis-Bedingung können Merkmale eines Ereignisses durch Variablenbindungen der entsprechenden Transition in einer Tokenvariablen aggregiert und in

den Bedingungen von Transitionen anderer Ereignisse referenziert werden. Beispielsweise wurden diese Mechanismen in Abb. 6-25 verwendet, um ein sequenzielles Eintreten der Ereignisse zu spezifizieren.

6.5.2 Instanzselektion

Die Festlegung, welche Merkmale welchen Ereignisses an eine Signaturinstanz gebunden werden, kann durch die Definition der Variablenbindungen getroffen werden. Der Selektionsmodus *first* für ein Ereignis vom Typ *A* wird durch eine entsprechende Transition festgelegt, deren Variablenbindungen die Merkmale des auslösenden Ereignisses in Tokenvariablen festhalten. Um ein Ereignis mit dem Selektionsmodus *last* modellieren zu können, kann eine Schlinge verwendet werden, die eine Transition enthält, durch deren Variablenbindungen die Tokenvariablen bei jedem Schalten die Merkmale des aktuellen Ereignisses zugewiesen werden. In Abb. 6-36 ist ein entsprechendes Signaturnetz für die Ereignisbeschreibung *(A; last B; C)* dargestellt.

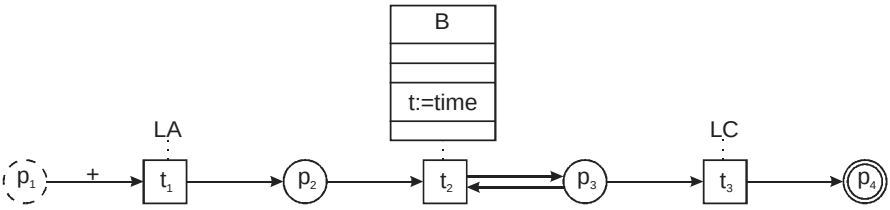


Abb. 6-36. Signaturnetz für Ereignisbeschreibung *(A; last B; C)*

Die Modellierung eines Ereignisses mit dem Selektionsmodus *all* macht eine Erweiterung der Tokenvariablen und Variablenbindungen erforderlich. Sie erfolgt wiederum durch eine entsprechende Schlinge (vgl. Abb. 6-37). Beim Schalten der Schlingentransition dürfen jedoch keine Werte von Tokenvariablen überschrieben werden. Dies kann erreicht werden, indem die Werte von Tokenvariablen in dynamischen Listen organisiert werden. Beim Schalten der Transition wird der Liste einer Tokenvariable der entsprechende Merkmalswert des aktuellen Ereignisses hinzugefügt.

6.5.3 Instanzkonsum

In vielen Fällen kann der Instanzkonsummodus eines Ereignisses aus der Semantik des Ereignisses im System abgeleitet werden. Die Konsumeigenschaft bezieht sich jedoch nicht nur auf das Ereignis selbst, sondern auf das

Ereignis und einen erforderlichen Systemzustand. Dementsprechend wird der Konsummodus nicht als Transitionseigenschaft, sondern als Eigenschaft der Eingangskanten von Transitionen modelliert. Das Signaturnetz in Abb. 6-38 enthält eine Transition t_2, die mit dem Vorplatz p_2 über eine nicht-konsumierende und mit p_4 über eine konsumierende Kante verbunden ist. Die Transition modelliert ein Ereignis *C*, dessen Eintreten den durch Ereignisse *B* erreichten Zustand konsumiert, nicht jedoch den durch Ereignisse *A* erreichten Zustand. Das folgende Beispiel diskutiert eine Anwendung für derartige Netze.

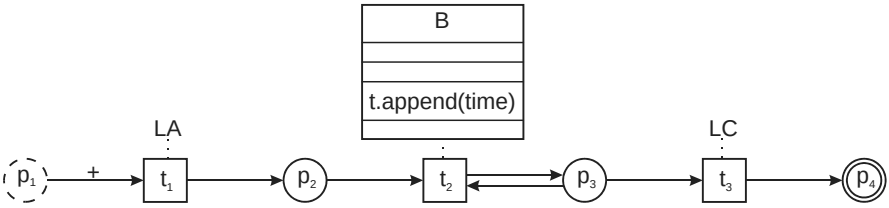


Abb. 6-37. Signaturnetz für Ereignisbeschreibung *(A; all B; C)*

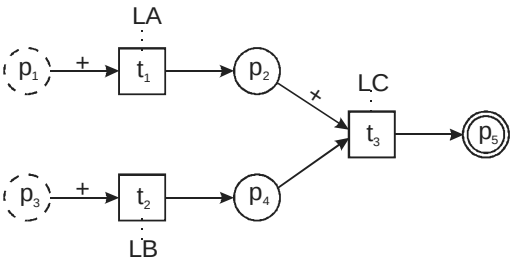


Abb. 6-38. Konsummodi in Signaturnetzen

Beispiel: Das Beispiel bezieht sich auf Betriebssystemaktionen. Entsprechende Ereignisse werden durch die Audit-Funktion generiert. Ein Ereignis vom Typ *A* modelliert die Erzeugung eines Prozesses mit privilegierten Zugriffsrechten (root). Ereignis *B* modelliert die Erzeugung einer Datei (Shell-Skript). Ereignis *C* stellt die Veränderung der Attribute der erstellten Datei durch den erzeugten Prozess dar (Eigentümer wird root und die Zugriffsrechte SUID). Durch die Verwendung der entsprechenden Konsummodi berücksichtigt das Signaturnetz in Abb. 6-38, dass mehrere Dateien angelegt und durch einen Prozess mit Zugriffsrechten versehen werden können.

6.6 Verwandte Ansätze

Dieser Abschnitt stellt verwandte Ansätze auf dem Gebiet der Modellierung von Signaturen vor und dient der Einordnung des vorgestellten Modellierungsansatzes in die aktuelle Forschung. Die diskutierten Ansätze können in automatenbasierte, graphenbasierte und netzbasierte Modellierungsansätze unterschieden werden.

6.6.1 Automatenbasierte Signaturmodellierung

Sobirey et al. [So+96] modellieren Signaturen unter Verwendung endlicher deterministischer Automaten, wobei Zustandsübergänge mit Ereignissen assoziiert werden. Die Modellierung erfolgt auf hohem Abstraktionsniveau und dient hauptsächlich einer groben graphische Veranschaulichung von Signaturen. Informal werden Automaten um Kontextbedingungen an Zuständen erweitert, die Merkmale von Ereignissen referenzieren können. Der Modellierungssatz erlaubt eine eingeschränkte Charakterisierung der Reihenfolge von Ereignissen einer Signatur. Unterstützt werden Sequenzen und Disjunktionen von Ereignissen. Durch die Verwendung von Abbruchzuständen und entsprechenden Zustandsübergängen kann die Negation von Ereignissen ausgedrückt werden. Konjunktionen von Ereignissen können nicht explizit dargestellt werden, so dass zur Beschreibung entsprechender Signaturen die Modellierung aller möglichen Sequentialisierungen dieser Ereignisse erforderlich ist. Der Ansatz verwendet keine Variablen, so dass auch unter Verwendung von Schlingen, keine Beschreibung der unterschiedlichen Häufigkeits- und Selektionsmodi möglich ist. Des Weiteren ist es nicht möglich Konsummodi für Ereignisse zu spezifizieren.

Die STAT-Toolsuite [Vi+00] basiert auf der State Transition Analysis Technique. Zur Beschreibung von Signaturen für diese Systeme wird die Beschreibungssprache *STATL* [Eck+00a, Eck+02] verwendet, der implizit eine Modellierung der Signaturen als Zustands-Transitions-Diagramm zugrunde liegt, die zunächst keine Unterschiede zu den von Sobirey et al. verwendeten Automaten aufweist. Entsprechend können auch hier nur zeitliche Beziehungen zwischen Ereignissen modelliert werden, die eine Sequenz oder Disjunktion von Ereignissen darstellen. Konjunktionen können nur durch Beschreibung aller möglichen Sequentialisierungen dargestellt werden. Die Negation von Ereignissen wird eingeschränkt durch dedizierte „unwinding“ Zustandsübergänge unterstützt. Bei der konkreten Semantik von Unwinding-Transitionen (vgl. [Eck+00b]) erweist sich jedoch als problematisch, dass ein „unwinding“ einer Signaturinstanz in verschiede-

nen Fällen Auswirkungen auf andere Signaturinstanzen besitzt. Kontextbedingungen können sowohl Zuständen als auch Transitionen zugeordnet werden. Die Beschreibungssprache STATL erlaubt außerdem die Verwendung von Variablen, so dass durch Schlingen verschiedene Häufigkeits- und Selektionsmodi abgebildet werden können. Konsumeigenschaften werden ebenfalls unterstützt, indem jeder Transition einer der Modi consuming oder non-consuming zugewiesen wird. Konsum wird allerdings als reine Transitionseigenschaft modelliert, die für alle Vorzustände gleich ist. Da Konjunktionen nicht unterstützt werden und demnach jede Transition nur einen Vorzustand besitzt, stellt dies keine zusätzliche Beschränkung dar. Eine Modellierung von Signaturen wie bspw. der aus Abb. 6-38 ist bereits aufgrund der fehlenden Unterstützung von Konjunktionen nicht möglich.

Pouzol et al. [Pou+01] entwickelten zur Beschreibung von Signaturen die deklarative Sprache *Sutekh*, deren operationale Semantik durch Abbildung auf Regeln für Expertensysteme definiert ist. Zur Abbildung auf Regeln werden Signaturen zunächst in eine als *SigGraph* bezeichnete Zwischenrepräsentation überführt, die Signaturen als Automaten modelliert. Entsprechend erfolgt die Modellierung statischer Signatureigenschaften hier hauptsächlich zu Transformationszwecken und Entwurfs- und Analyseaspekte stehen im Hintergrund. Transitionen eines SigGraphen sind Ereignisse, Bedingungen und Variablenbindungen zugeordnet. In Sutekh beschriebene zeitliche Zusammenhänge zwischen Ereignissen resultieren in einer entsprechenden strukturellen Anordnung von Zuständen und Transitionen im entsprechenden SigGraphen, wobei Konjunktionen von Ereignissen durch die Modellierung aller möglichen Sequentialisierungen dargestellt werden. Die Beschreibung wiederholten Auftretens und die Selektion von Ereignissen werden durch Sutekh nicht unterstützt. Eine Darstellung von Schleifen als SigGraph erscheint über Schlingen und entsprechende Variablenbelegungen jedoch möglich. Ohne dass diese in Sutekh spezifiziert werden, werden Konsumeigenschaften in SigGraphen modelliert. Diese werden als Eigenschaft von Zuständen dargestellt, die ausgehend von einer Heuristik auf der Basis von Variablenbindungen und Variablenverwendungen ermittelt werden. Da Konsum von den Sutekh-Autoren nicht als dedizierte zustandsspezifische Eigenschaft von Ereignissen in einer Signatur erkannt wurde, können sowohl Sutekh-Beschreibungen als auch SigGraph-Modellierungen nicht in allen Fällen die vollständige Semantik von Signaturen erfassen.

6.6.2 Graphenbasierte Signaturmodellierung

Einziger Vertreter graphenbasierter Signaturmodellierungsansätze sind die von Ning et al. [Li+98, Ni+01] vorgestellten *MuSigs (Misuse Signatures)*, die wie automatenbasierte Ansätze nur die Repräsentation von statischen Signaturcharakteristika unterstützen und keine Modellierungselemente zur Abbildung von Signaturinstanzen enthalten. Die Autoren führen Systemsichten als Abstraktion für Ereignisse einer Ereignis-Trail ein, auf deren Grundlage Signaturen beschrieben werden können. Zur Darstellung von Signaturen werden Graphen verwendet, deren Knoten Systemsichten, Variablenbindungen und Bedingungen zugeordnet werden. Variablenbindungen aggregieren Merkmale von Systemsichten und können in Bedingungen verwendet werden. Kanten zwischen den Knoten werden qualifizierte zeitliche Beziehungen, z. B. *jünger* oder *gleichzeitig*, zugeordnet, die zusätzlich in Bedingungen für Zeitstempel zwischen den entsprechenden Systemsichten quantifiziert werden müssen (vgl. [Ni+01]). Die Modellierung der Negation von Ereignissen wird explizit durch die Möglichkeit unterstützt, Knoten zu beschreiben, die negative Ereignisse bzw. Systemsichten repräsentieren. Die Verwendung von Schlingen in MuSig-Graphen wurde von den Autoren nicht vorgesehen, so dass keine Möglichkeit besteht, verschiedenen Häufigkeits- oder Selektionsmodi zu beschreiben. Die Charakterisierung der Konsumeigenschaften von Ereignissen wird durch MuSig-Graphen nicht unterstützt.

6.6.3 Netzbasierte Signaturmodellierung

Im Unterschied zu Automaten und Graphen werden in Netzen nicht nur Zustandsknoten und (Transitions-)Kanten als Modellierungselemente eingesetzt sondern zusätzlich Transitionsknoten und Token verwendet. Durch die Verwendung von Token werden insbesondere die Simulation von Signaturanalyseabläufen, die Repräsentation von Signaturinstanzen und somit dynamischen Zusammenhängen ermöglicht.

Einen netzbasierten Ansatz zur Modellierung von Signaturen stellen die von Kumar et al. [Ku+94a, Ku+94b] vorgestellten *Coloured Petri Net Automatons (CPAs)* dar. Analog Signaturnetzen erlauben auch CPAs die Modellierung vielfältiger zeitlicher Zusammenhänge von Ereignissen, die durch Transitionen repräsentiert werden. Im Unterschied zu Signaturnetzen kennen CPAs keine Abbruchplätze. Sie unterstützen zwar die Modellierung der Negation von Ereignissen, jedoch auf eine weniger kompakte Art und Weise. CPAs verwenden dazu Invarianten, die auch als konjugierte negative Spezifikationen bezeichnet werden (vgl. [Ku95]). Diese

werden als separates Netz spezifiziert. Für jedes Token, das einen CPA durchläuft, existiert ein Token im Invariantennetz. Erreicht ein Token im CPA einen neuen Platz, so wird die entsprechende Tokenkopie im Invariantennetz auf den Anfangsplatz zurückgesetzt. Erreicht eine Tokenkopie im Invariantennetz einen Endzustand, dann wird das korrespondierende Token im CPA entfernt.

CPAs verwenden Unifikation zur Bindung von Werten an Tokenvariablen (vgl. [Ku95]) mit der Konsequenz, dass einmal gebundene Variablen nicht mit anderen Werten aktualisiert werden können. Dadurch ist auch mittels Schlingen keine Modellierung aller Häufigkeitsmodi und Selektionsmodi möglich.

CPAs unterstützen ein dem Konsum ähnliches Konzept. Dazu werden alle Plätze vom Modellierer mit einem der Attribute *dup* oder *nodup* versehen. Token, die einen Platz mit dem Attribut dup erreichen, schalten nicht weiter und verlassen diesen Platz nur durch das Schalten des Invariantennetzes. Stattdessen schalten Kopien dieser Token weiter, wodurch teilweise ähnliche Effekte wie mit nicht-konsumierenden Transitionen in Signaturnetzen erreicht werden können. Allerdings wird diese Konsumeigenschaft in CPAs Plätzen zugeordnet und wirkt somit für alle Nachtransitionen eines Platzes. Dadurch ist es nur eingeschränkt möglich die Konsumeigenschaften von Ereignissen zu modellieren. Darüber hinaus erwähnen die CPA-Autoren in [Ku95] zwar, dass nodup-Plätze selten auftreten, versäumen es jedoch zu diskutieren, von welchen Fakten ausgehend der Signaturmodellierer entscheidet, ob Plätze als dup oder nodup zu spezifizieren sind.

6.7 Zusammenfassung

In diesem Kapitel wurde mit den Signaturnetzen ein Modellierungsansatz für Angriffssignaturen eingeführt und formal definiert. Mit einem konstruktiven Ansatz wurde gezeigt, dass alle in Kapitel 5 aufgeführten semantischen Aspekte von Signaturen mit Signaturnetzen ausgedrückt werden können und alle in existierenden Modellierungen vorliegenden Signaturen erfasst werden können. Gegenüber anderen Ansätzen besitzen Signaturnetze eine Reihe von Vorteilen. Zum einen erlauben sie die Modellierung von partieller Ordnung auf kompakte Weise. Zum anderen ermöglichen sie unter Verwendung von Token die Simulation und Betrachtung von Analyseabläufen. Außerdem unterstützen sie eine graphische Modellierung und verfügen über eine hohe Ausdrucksstärke.

Auch in Spezialbereichen der Missbauchserkennung haben sich Signaturnetze bewährt. Sie wurden als Modell für Signaturen verwendet, um die erforderlichen Informationsflüsse bei der Signaturanalyse zu untersuchen und hinsichtlich personenbezogener Daten mittels Pseudonymisierung zu minimieren [FleMei02, Fle06].

7 Beschreibung von Angriffssignaturen

Nachdem wir vorigen Kapitel die Frage betrachtet haben, wie Signaturen modelliert, analysiert und simuliert werden können, steht nun die Frage im Mittelpunkt, wie und mit welchen Werkzeugen Signaturen in der Praxis konkret notiert werden. Hierbei stehen Aspekte der Modellierung, Analyse und Simulation im Hintergrund. Die Ausdrucksstärke der Beschreibungsmittel spielt zwar weiterhin eine wichtige Rolle, jedoch rücken zusätzlich Aspekte wie Benutzbarkeit und Einfachheit in den Vordergrund. Im Folgenden werden daher zunächst Aspekte der Praxis der Signaturentwicklung beleuchtet sowie existierende Beschreibungssprachen diskutiert. Die derzeit hauptsächlich verwendeten regelbasierten Beschreibungsansätze sowie Probleme und eigene Erfahrungen in diesem Bereich werden dargestellt. Danach werden mit SHEDEL und EDL zwei konkrete nacheinander entwickelte Beschreibungswerkzeuge vorgestellt, die sich in Ausdrucksstärke und Einfachheit unterscheiden. Die Sprache SHEDEL wurde entwickelt, um Konzepte zur Vereinfachung der Beschreibung von Signaturen zu evaluieren. Sie bildet eine Vorstufe der Sprache EDL, in deren Entwicklung die gewonnenen Erkenntnisse einflossen. Das Kapitel schließt mit einer vergleichenden Zusammenfassung der existierenden und der vorgestellten Beschreibungsansätze.

7.1 Signaturentwicklung

Eines der Hauptprobleme beim Einsatz von Systemen zur Signaturanalyse ist die Entwicklung und Verwaltung einer großen Anzahl von Signaturen (vgl. Abschn. 3.2). Der Entwicklungsprozess ist zeitaufwändig und verlangt detaillierte Kenntnisse über

- existierende Verwundbarkeiten der zu schützenden IT-Systeme und Möglichkeiten zu deren Ausnutzung,
- die eingesetzten Audit-Funktionen und
- häufig auch über die verwendeten Analysetechniken.

Signaturentwicklung ist kein einmaliger Prozess. Der Betrieb eines IDS erfordert eine kontinuierliche Pflege der Signaturbasis. Nach jeder Veränderung der Systemkonfiguration, Aktualisierung der Software oder der Entdeckung einer neuen Verwundbarkeit, sind Anpassungen der Signaturbasis erforderlich, um die Wirksamkeit der Überwachung sicherzustellen.

Daher stellt die Entwicklung und Pflege von Signaturen eine der wichtigsten und aufwendigsten Aufgaben beim Betrieb von IDS dar. Eine nicht zu unterschätzende Rolle kommt dabei den verwendeten Sprachen zur Beschreibung der Signaturen zu. In der Klassifikation von Vigna et al. [Vig+00] (vgl. Abschn. 3.1) werden diese Sprachen als *Erkennungssprachen (Detection Languages)* bezeichnet. Als wünschenswerte Eigenschaften von Erkennungssprachen werden für gewöhnlich Ausdrucksstärke, Exaktheit, Einfachheit, Erweiterbarkeit, Portierbarkeit und Heterogenität herausgestellt [Eck+02].

Eine Reihe verschiedener Sprachen wurde bereits entwickelt. Typischerweise verwendet jedes Intrusion-Detection-System eine eigene Sprache, die speziell auf die jeweils verwendeten Analysemechanismen zugeschnitten ist. Zu diesen gehören die Sprache von *P-BEST* (*Production-Based Expert System Toolset* [Li+99, Li+01]), das in dem IDS EMERALD [Po+97, Neu+99] eingesetzt wird, die Sprache *RUSSEL* (*RUle-baSed Sequence Evaluation Language*) für das System *ASAX* (*Advanced Security audit trail Analyzer on uniX*) [Ha+92, Mou97] und die Sprachen der Expertensysteme *CLIPS (C Language Integrated Production System)* [Gi+94] und RTworks [Tal95]. Hierbei handelt es sich um regelbasierte Sprachen, in denen Signaturen durch Regeln beschrieben werden. Die regelbasierte Signaturbeschreibung stellt den derzeit meist verwendeten Ansatz dar und wird im nächsten Abschnitt detailliert betrachtet.

Weitere Sprachen sind STATL [Eck+00a] für die STAT-Toolsuite [Vi+00], die Sprache CPA[1] des IDIOT-IDS [Ku95] sowie die Sprachen Sutekh [Pou+01] und MuSig [Ni+01]. Von diesen zeichnet sich Sutekh durch eine klare Trennung der Signaturbeschreibung von den Erkennungsabläufen aus (vgl. auch [Pou+02]). Verschiedene Beschränkungen und Defizite dieser Sprachen wurden bereits im Zusammenhang mit der ihnen zugrunde liegenden Modellierung von Signaturen in Abschn. 6.6 aufgezeigt. Keine dieser Sprachen bietet geeignete Konzepte zur Einführung von Abstraktionen und Modularisierung. STATL bietet mit *Named Signature Actions* zwar die Möglichkeit mehrfach verwendete Ereignisbeschreibungen einmal zu definieren und dann über einen Namen mehrfach zu referenzieren, allerdings kann dieser Mechanismus nur für einzelne und

[1] Die Sprache des IDIOT-IDS wird entsprechend dem zugrunde liegenden Modellierungsansatz (vgl. Abschn. 6.6.3) im Weiteren als CPA bezeichnet.

nicht für komplexe Ereignisse verwendet werden. Des Weiteren ist es mit STATL möglich zu beschreiben, dass bei Erreichen eines Zustandes so genannte *Synthetische Ereignisse* erstellt und ausgelöst werden. Dadurch ist eine Vorwärtsverkettung von Signaturbeschreibungen möglich, der jedoch zwangsläufig der Nebenläufigkeitsmodus *überlappend* (vgl. Abschn. 5.4.4) zugrunde liegt.

LAMBDA [Cu+00] und ADeLe [Mi+01] sind weitere Sprachen zur Beschreibung von Signaturen. Beide wurden parallel innerhalb des gleichen Projekts entwickelt und sind keine reinen Erkennungssprachen. Sie wurden vielmehr zur Beschreibung der verschiedenen Aspekte von Attacken entwickelt. Die Autoren der Sprachen versuchen alle verfügbaren Informationen über eine Attacke in einer Beschreibung zu kombinieren. Beispielsweise enthalten Beschreibungen in diesen Sprachen neben der Signatur auch eine Beschreibung, wie die entsprechende Attacke durchgeführt wird. Beschreibungen in LAMBDA enthalten zusätzlich eine Spezifikation, wie der Erfolg einer Attacke überprüft werden kann. Hinsichtlich der Charakterisierung der semantischen Aspekte von Signaturen beschränken sich beide Sprachen auf die Aspekte *Typ und Reihenfolge* sowie *Kontextbedingungen* der Dimension Ereignismuster. Eine Unterscheidung verschiedener Selektions- und Konsummodi ist nicht möglich.

Obwohl die Einsetzbarkeit vieler der genannten Sprachen nachgewiesen wurde, erweist sich die Nutzung dieser Sprachen in der Regel als nicht einfach. Da, wie obige Diskussion zeigt (vgl. auch Abschn. 3.1), Signaturbeschreibungssprachen hauptsächlich von IDS-Administratoren mit unterschiedlicher Expertise eingesetzt werden, sind Einfachheit und Benutzbarkeit signifikante Charakteristika einer *geeigneten* Beschreibungssprache. Es besteht ein zunehmender Bedarf an Methoden und Techniken zur Vereinfachung der Signaturentwicklung.

Die meisten existierenden Sprachen fordern nicht nur die Beschreibung der Signaturen, sondern es sind zusätzlich Details zur Steuerung des Erkennungsprozesses anzugeben. Dadurch wird die Entwicklung und Verwaltung von Signaturen noch komplexer und fehleranfälliger. Eigene Erfahrungen aus der Entwicklung und dem Einsatz regelbasierter IDS unterstreichen diese Aussage.

7.2 Regelbasierte Signaturbeschreibung

Die regelbasierte Signaturbeschreibung stellt den derzeit am häufigsten verwendeten Ansatz dar. Auf diesem Ansatz basierende Systeme verwenden Expertensysteme als Analysekomponente und sind dadurch mit gerin-

gem Aufwand realisierbar. Im Folgenden werden Expertensysteme grundlegend eingeführt und ihre Anwendung zur Missbrauchserkennung dargestellt bevor Probleme des Ansatzes insbesondere hinsichtlich der Beschreibung von Signaturen diskutiert werden.

7.2.1 Expertensysteme

Charakteristisch für Expertensysteme ist die Trennung zwischen Anwendungswissen und allgemeinen Problemlösungsstrategien. Anwendungswissen wird in Form von Regeln beschrieben, die auf Fakten im Arbeitsspeicher des Expertensystems angewendet werden. Die Problemlösung obliegt vorgegebenen Mechanismen. Aufgrund dieser Trennung sind Expertensysteme flexibel einsetzbar und erweiterbar (vgl. [Pu91]) und wurden zur Realisierung verschiedener IDS herangezogen. Beispiele hierfür sind die IDS *MIDAS (Multics Intrusion Detection and Alerting System)* [Se88], EMERALD [Po+97], AID [So+96] und CLIPS-IDS [Krau04] und CMDS [Pro94]. Während MIDAS und EMERALD auf dem Expertensystem P-Best basieren, verwenden CMDS und CLIPS-IDS das Expertensystem CLIPS. AID wurde unter Verwendung der kommerziellen Expertensystemshell RTworks [Tal95] implementiert. Vorteil der Verwendung von Expertensystemen ist die einfache Implementierung und die Verwendung optimierter Analysealgorithmen.

Regelbasierte Expertensysteme, die auch Produktionssysteme genannt werden [Pu91, Ja99], automatisieren das Überprüfen von Aussagen über Faktenwissen und das Ziehen von Schlussfolgerungen, was ggf. in der Ableitung neuer Fakten resultiert. Dazu müssen Aussagenüberprüfungen und Schlussfolgerungen durch Bedingungs-Aktions-Regeln (IF-THEN-Regeln) dargestellt werden. Ein Expertensystem setzt sich im Wesentlichen aus den folgenden Komponenten zusammen (vgl. Abb. 7-1):

- einer Faktenbasis, die die gültigen Fakten enthält,
- den Regeln zur Herleitung neuer Fakten und
- dem Regelinterpreter zur Steuerung des Herleitungsprozesses.

Der Herleitungsprozess besteht aus zwei Phasen. In der ersten Phase (Überprüfungsphase) werden durch Überprüfung der Regelbedingungen alle anwendbaren Regeln ermittelt und in einer als *Agenda* bezeichneten Menge abgelegt. In der zweiten Phase (Auswahlphase) wird eine Regel aus der Agenda ausgewählt und ausgeführt. Die Regelausführung kann sich auf die Faktenbasis und somit auf die Anwendbarkeit von Regeln auswirken. Deshalb folgt darauf eine Aktualisierung der Agenda. Es wird über-

prüft, ob durch die Änderungen der Faktenbasis Regeln anwendbar oder nicht mehr anwendbar sind. Danach wird die nächste Regel aus der Agenda ausgewählt und ausgeführt. Dieser Ablauf wiederholt sich bis die Agenda leer ist und der Regelinterpreter terminiert.

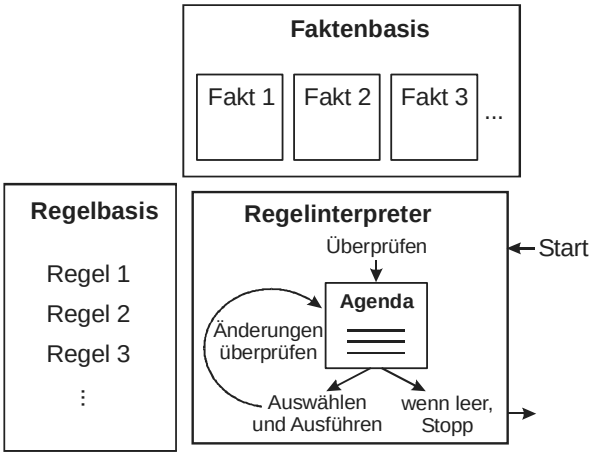


Abb. 7-1. Komponenten eines Expertensystems

Werden in der Überprüfungsphase mehrere Regeln in der Agenda abgelegt, dann existieren alternative Ausführungsreihenfolgen für diese Regeln. Da die Ausführung einer Regel Auswirkungen auf die Faktenbasis und somit die Agenda haben kann, können unterschiedliche Ausführungsreihenfolgen zu unterschiedlichen Zuständen der Faktenbasis führen. In diesem Zusammenhang wird eine Menge von Regeln als *konfluent* bezeichnet, wenn der Zustand der Faktenbasis nicht von der Ausführungsreihenfolge dieser Regeln beeinflusst wird, also jede Reihenfolge das gleiche Ergebnis liefert. In der Auswahlphase, die auch als Konfliktlösung bezeichnet wird, erfolgt die Festlegung einer Ausführungsreihenfolge. Durch Festlegung einer vorgegebenen Konfliktlösungsstrategie für den Regelinterpreter kann dieser Vorgang gesteuert werden. Typischerweise können Regeln ausgehend von definierten Prioritäten oder der Spezifik ihrer Bedingungen priorisiert werden.

7.2.2 Expertensystembasierte Missbrauchserkennung

Die regelbasierte Umsetzung von Signaturen kann durch eine Abbildung von Signaturnetzen (vgl. Kapitel 6) auf Regeln und Fakten verdeutlicht werden (vgl. [Krau04, Ko04]). Die Fakten der Faktenbasis repräsentieren

dabei das aktuell zu analysierende Ereignis sowie die existierenden Token. Transitionen werden auf Regeln abgebildet, deren Bedingungsteil sämtliche Transitionsbedingungen umfasst. Der Ereignistyp einer Transition wird ebenfalls auf eine Regelbedingung abgebildet. Fakten, die Token repräsentieren, erhalten zusätzlich zu den Tokenvariablen ein Attribut *Platzname*, das angibt, auf welchem Platz sich das Token befindet. Entsprechend erfolgt die Festlegung der Vorplätze einer Transition, indem der entsprechenden Regel Bedingungen hinzugefügt werden, die fordern, dass Tokenfakten existieren, die im Attribut *Platzname* mit den Namen der Transitionsvorplätze übereinstimmen. Beim Schalten einer Transitionsregel kommt ihr Aktionsteil zur Ausführung. Dieser erstellt Tokenfakten, die mit den Namen der Transitionsnachplätze sowie den Variablenbindungen der Transition belegt werden und führt die Transitionsaktion aus. Der Konsum von Token, wird durch das Löschen von Tokenfakten im Aktionsteil von Regeln realisiert. Abb. 7-2. veranschaulicht die Abbildung von Signaturnetzen auf Regeln, indem es den einzelnen Signaturnetzelementen die entsprechenden Regelbestandteile zuordnet.

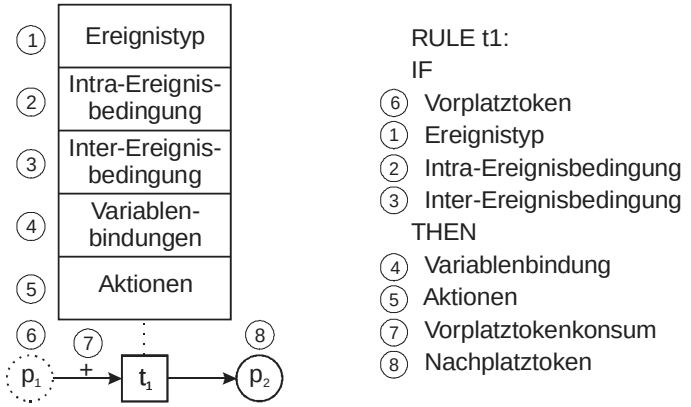


Abb. 7-2. Abbildung von Signaturnetzelementen auf Regelbestandteile

Der Analysezyklus eines solchen Systems sieht vereinfacht wie folgt aus: Nach dem Einfügen des aktuellen Ereignisses als Fakt in die Faktenbasis wird der Regelinterpreter gestartet, der alle erfüllten Regeln ermittelt und eine davon ausführt. Anschließend wird überprüft, ob durch die Aktualisierung der Faktenbasis Regeln anwendbar oder nicht mehr anwendbar geworden sind. Der Vorgang wiederholt sich bis keine Regel mehr erfüllt ist. Ist dieser Zustand erreicht, terminiert der Regelinterpreter. Danach wird der Ereignisfakt durch das nächste Ereignis aus der Ereignis-Trail er-

setzt und der Regelinterpreter erneut gestartet. Abb. 7-3 skizziert diese Umsetzung der Missbrauchserkennung mittels eines Expertensystems.

7.2.3 Probleme expertensystembasierter Missbrauchserkennung

Aufgrund verschiedener Besonderheiten der Missbrauchserkennung erweist sich die Verwendung von Standardexpertensystemen als Analyseverfahren als ungeeignet. Mit den Inferenzmechanismen der Regelinterpreter ist keine exakte Simulation der Schaltregel von Signaturnetzen möglich. Die von der Schaltregel geforderte gleichzeitige Ausführung aller schaltbereiten Transitionen (Regeln) kann durch die Regelinterpreter nicht ohne weiteres umgesetzt werden. Daraus resultieren schwerwiegende Probleme hinsichtlich der Zuverlässigkeit expertensystembasierter Systeme zur Missbrauchserkennung, die in der Literatur bisher nicht diskutiert werden. Auch werden keine Annahmen getroffen (z. B. konfluente Regelmengen), die ihr Auftreten ausschließen. Die als *kaskadierte Regelausführung* und *nicht-deterministische Regelauswahl* bezeichneten Probleme werden im Folgenden diskutiert und Lösungsmöglichkeiten werden erörtert.

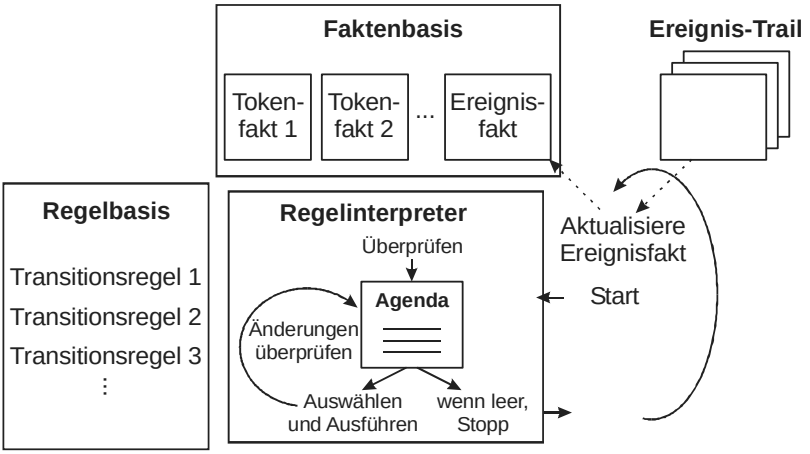


Abb. 7-3. Expertensystembasierte Missbrauchserkennung

Kaskadierte Regelausführung

Kaskadiertes Schalten von Transitionen wurde bereits im Zusammenhang mit der Schaltregel von Signaturnetzen erwähnt (vgl. Abschn. 6.2.5). Das Problem soll hier anhand einer Signatur für die Login-Attacke verdeutlicht

werden, durch die wiederholte fehlerhafte Anmeldeversuche erkannt werden (vgl. Abschn. 4.2). Charakteristisch für diese Signatur ist eine Sequenz gleichartiger Ereignisse. Das entsprechende Signaturnetz wurde in Abschn. 6.3 dargestellt. Jede Transition wird auf eine Regel abgebildet, wobei hier nur die Transitionen t_1, t_2 und t_3 betrachtet werden (vgl. Abb. 7-4). Die Umsetzung des Signaturnetzes durch Regeln und Fakten erfolgt wie in Abschn. 7.2.2 beschrieben. Zur weiteren Diskussion werden die Abläufe beim Einfügen eines Ereignisfakts für eine fehlerhafte Anmeldung und Starten des Regelinterpreters betrachtet. Dabei werden folgende Schritte ausgeführt:

- Ein Ereignisfakt für eine fehlerhafte Anmeldung wird in die Faktenbasis eingefügt.
- Der Regelinterpreter wird gestartet.
- Anwendbare Regeln werden ermittelt: *Regel t_1 ist anwendbar*.
  - Eine anwendbare Regel wird ausgewählt: *t_1 wird ausgewählt*.
  - Die ausgewählte Regel wird ausgeführt: t_1 wird ausgeführt und es wird ein Fakt erzeugt, der ein Token auf Platz p_2 repräsentiert.
- Es wird überprüft, ob durch die Änderungen der Faktenbasis Regeln anwendbar oder nicht mehr anwendbar sind: *Regel t_2 ist jetzt anwendbar*.
  - Eine anwendbare Regel wird ausgewählt: t_2 wird ausgewählt.
  - Die ausgewählte Regel wird ausgeführt: t_2 wird ausgeführt und es wird ein Fakt erzeugt, der ein Token auf p_3 repräsentiert. Der Fakt für ein Token auf p_2 wird gelöscht (konsumiert).
- Es wird überprüft, ob durch die Änderungen der Faktenbasis Regeln anwendbar oder nicht mehr anwendbar sind: *Regel t_3 ist jetzt anwendbar*.
  - Eine anwendbare Regel wird ausgewählt: t_3 wird ausgewählt.
  - Die ausgewählte Regel wird ausgeführt: t_3 wird ausgeführt und der Fakt für ein Token auf p_3 wird gelöscht (konsumiert).
- Es wird überprüft, ob durch die Änderungen der Faktenbasis Regeln anwendbar oder nicht mehr anwendbar sind: *keine Regel ist anwendbar*.
- Der Regelinterpreter terminiert.

In dem beschriebenen Beispiel führt der Standard-Inferenzmechanismus dazu, dass durch ein einziges Ereignis alle drei aufeinander folgenden Transitionsregeln der Signatur feuern, was im Widerspruch zur Schaltregel für Signaturnetze steht. Ursache dafür ist das wiederholte Überprüfen von Regeln nach Änderungen an der Faktenbasis, ohne dass ein neuer Ereignisfakt vorliegt.

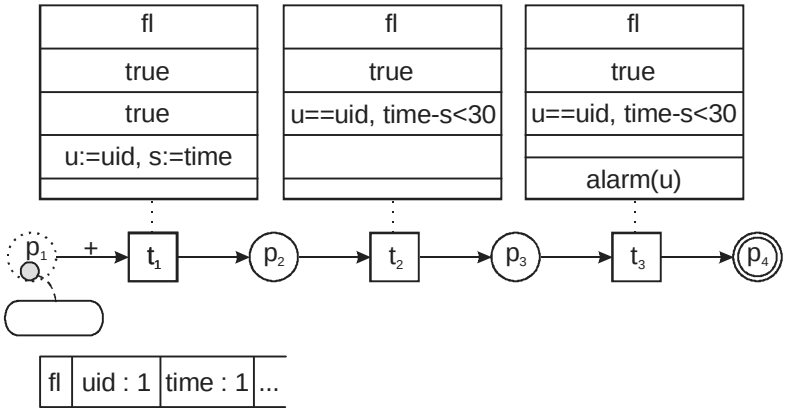


Abb. 7-4. Signaturnetz für die Login-Attacke (Auszug)

Um diesen Effekt auszuschließen ohne den Regelinterpreter zu modifizieren, müssen Ereignisfakten fortlaufend nummeriert werden. Beim Feuern einer Transitionsregel wird die Ereignisnummer in alle Tokenfakten aufgenommen, die beim Feuern erzeugt werden. Außerdem prüfen alle Transitionsregeln, ob die Nummer des aktuellen Ereignisfakts von der Ereignisnummer der Vorplatz-Tokenfakten verschieden ist. Abb. 7-5 hebt die erforderlichen Erweiterungen an einem Signaturnetz hervor. Durch dieses Vorgehen wird ausgeschlossen, dass Tokenfakten, die beim Schalten einer Regel für einen Ereignisfakt erstellt wurden, weitere Regeln mit diesem Ereignisfakt aktivieren. Nachteilig an dieser Lösung ist, dass zusätzliche Anweisungen in die Regeln aufgenommen werden müssen sowie die Prüfung zusätzlicher Bedingungen erforderlich ist.

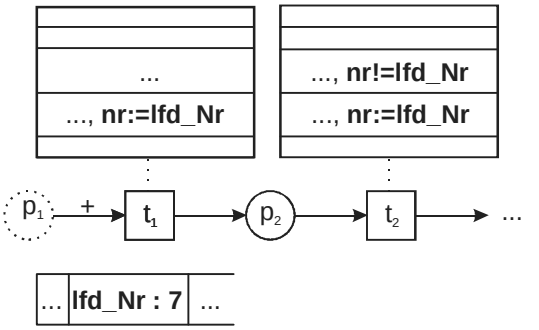


Abb. 7-5. Erweiterungen der Transitionsbeschriftungen

Nicht-deterministische Regelauswahl

Dieses Problem ergibt sich bei der regelbasierten Umsetzung von Signaturnetzen, die Konfliktstrukturen (vgl. Abschn. 6.2.6) verwenden. Hier sind Fälle möglich, in denen konsumierende Transitionen mit gemeinsamen Vorplätzen gleichzeitig schalten, auch wenn jeweils nur ein Token auf den gemeinsamen Vorplätzen vorhanden ist. Abb. 7-6 skizziert ein Signaturnetz in einer solchen Situation. Hier können beide Transitionen für ein Ereignis schalten. Bei der Umsetzung der Signatur mittels Regeln entspricht dies der Situation, dass beide Regeln in der Überprüfungsphase in die Agenda gelegt werden. In der Auswahlphase wird eine der Regeln ausgewählt und danach ausgeführt, wobei der den Vorplatztoken repräsentierende Fakt entfernt wird. Dadurch ist die andere Regel nicht mehr anwendbar und kommt nicht zur Ausführung, was im Widerspruch zur Schaltregel für Signaturnetze steht.

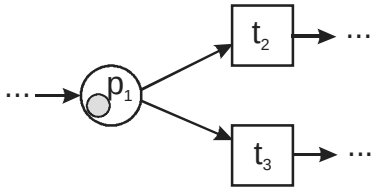


Abb. 7-6. Signaturnetz mit Konflikt (Auszug)

Eine Lösung für dieses Problem kann durch eine Verzögerung der Aktualisierungen der Faktenbasis erreicht werden, indem alle Aktualisierungen erst vorgenommen werden, nach dem die Agenda leer ist. Dazu können Expertensysteme um Zusatzfunktionen erweitert werden, die in den Regeln verwendet werden, um Faktenveränderungen in einer Datenstruktur vorzumerken. Nachdem der Regelinterpreter die Agenda abgearbeitet und terminiert hat, werden die vorgemerkten Faktenänderungen umgesetzt, bevor der nächste Ereignisfakt eingefügt wird. Wird dieses Vorgehen auch für das Erzeugen von Tokenfakten umgesetzt, dann wird dadurch das gleichzeitige Schalten von Transitionen simuliert und auch das Problem der kaskadierten Regelausführung beseitigt. Nachteile dieser Lösung sind die erforderlichen Erweiterungen des Regelinterpreters um die Zusatzfunktionen zur Organisation von Faktenänderungen. Der Hauptvorteil der Verwendung von Expertensystemen, der Einsatz vorgefertigter Verfahren, geht dadurch verloren.

7.2.4 Regelbasierte Signaturbeschreibung

Ausgehend von obiger Diskussion sowie basierend auf eigenen Erfahrungen aus der Entwicklung und dem Einsatz der IDS AID [So+96, Ri+99] und CLIPS-IDS [Krau04] wird die regelbasierte Beschreibung von Signaturen aus verschiedenen Gründen als problematisch angesehen. Zum einen erfolgt die Beschreibung fern von der Konzeptwelt der Missbrauchserkennung, was für die Beherrschung der Komplexität der Signaturentwicklung nicht förderlich ist. Zum anderen bieten Regelsprachen von Expertensystemen keine Konstrukte zur Einführung von Abstraktionen. Außerdem wird bei diesem Ansatz die Beschreibung einer Signatur über eine Menge von Regeln verstreut. Daraus resultiert ein Mangel an Übersichtlichkeit, der die Entwicklung und Verwaltung von Signaturen verkompliziert und das Potential für Fehler erhöht. Ein weiterer Nachteil des Ansatzes besteht darin, dass nicht nur die Beschreibung der Signaturen erforderlich ist, sondern auch verschiedene operationale Details bzgl. des eingesetzten Analyseverfahrens spezifiziert werden müssen. Beispielsweise ist es in jedem Falle notwendig, innerhalb des THEN-Teils einer Regel neue Fakten zu erzeugen oder existierende explizit zu verändern, um Zustandsänderungen oder die Erzeugung einer neuen Signaturinstanz zu implementieren. Des Weiteren ist eine korrekte Signaturbeschreibung ohne Anpassung der Regelinterpreter nur unter einschränkenden Annahmen über die Signaturen möglich (s. Abschn. 7.2.3). Können diese Annahmen nicht getroffen werden, sind zur korrekten Abbildung der Signaturen Erweiterungen der Regelinterpreter erforderlich und zusätzliche Anweisungen in den Regeln erforderlich. Diese Probleme lassen den regelbasierten Ansatz für eine systematische Beschreibung von Signaturen als ungeeignet erscheinen. Deshalb werden alternative Ansätze untersucht.

7.3 SHEDEL – Eine einfache ereignisbasierte Beschreibungssprache

Mit der Entwicklung der ereignisbasierten Beschreibungssprache *SHEDEL* (*Simple Hierarchical Event DEscription Language*) [Mei+02, HoMei+02a] wurde ein erster alternativer Ansatz entwickelt. SHEDEL verfolgt insbesondere das Ziel die Entwicklung und Wartung von Signaturen zu vereinfachen. Wir geben hier zunächst eine kurze Beschreibung der wesentlichen Sprachelemente, bevor einige Beispiele erörtert werden. Dem schließt sich eine Diskussion und Bewertung der Sprache an.

7.3.1 Beschreibungselemente von SHEDEL

Die meisten existierenden Erkennungssprachen sind vollständig oder teilweise imperativ. Signaturen in diesen Sprachen stellen im Wesentlichen prozedurale Beschreibungen der jeweiligen Erkennungsabläufe dar. Im Unterschied dazu erfolgt die Signaturbeschreibung in SHEDEL getrennt von den Analyseabläufen im IDS. Es werden nur die Signaturen an sich notiert. Es sind keinerlei Angaben notwendig, wie die Analyse der Audit-Daten durchgeführt wird und welche Zustandsinformationen dabei gespeichert werden müssen. SHEDEL beschreibt nur, *was* die Signatur ist und nicht *wie* die Analyse der Audit-Daten abläuft.

SHEDEL verwendet nur eine einzige zentrale Abstraktion: das *Ereignis*. Sowohl Audit-Datensätze als auch Teil-Signaturen und Signaturen stellen Ereignisse dar. Ein Ereignis ist durch seinen *Namen*, der gleichzeitig den *Typ* eines Ereignisses bestimmt, und eine Menge von *Merkmalen* gekennzeichnet. Die Signatur für eine Attacke wird in SHEDEL als Ereignis des entsprechenden Attacken-Typs dargestellt und als Kombination von *Schritten*, die ebenfalls Ereignisse sind, beschrieben. Dabei können sowohl *zeitliche Beziehungen* zwischen den Schritten als auch *Bedingungen* bzgl. der Merkmale der Schritte angegeben werden. Durch Bedingungen können Beschränkungen hinsichtlich der Merkmale eines Schrittes (Intra-Ereignis-Bedingungen) sowie Beziehungen zwischen den Merkmalen mehrerer Schritte (Inter-Ereignis-Bedingungen) festgelegt werden. Die zeitliche Reihenfolge wird dabei durch die Spezifikation der für einen Schritt erforderlichen Vorgängerschritte festgelegt. Die in einer Ereignisbeschreibung angegebene Schrittfolge enthält mindestens einen *Eingangsschritt* und einen *Ausgangsschritt*. Des Weiteren können *Abbruchschritte* spezifiziert werden.

Abb. 7-7 veranschaulicht diese Zusammenhänge beispielhaft für die Beschreibung eines Ereignisses *E*. Dieses Ereignis besteht aus zwei Schritten, einem Ereignis vom Typ *A*, das den Eingangsschritt darstellt, und einem Ereignis vom Typ *B*, dem Ausgangsschritt, die in dieser zeitlichen Reihenfolge auftreten müssen. Ereignisse vom Typ *A* und *B* bestehen ihrerseits wiederum aus mehreren Schritten. Die Merkmale des Ereignisses *E* werden definiert, indem ihnen jeweils ein Merkmal eines Schrittes zugeordnet wird. Beispielsweise wird das Merkmal *start* durch das Merkmal *time* des Schrittes vom Typ *A* bestimmt. Abb. 7-8 zeigt die resultierende Ereignishierarchie.

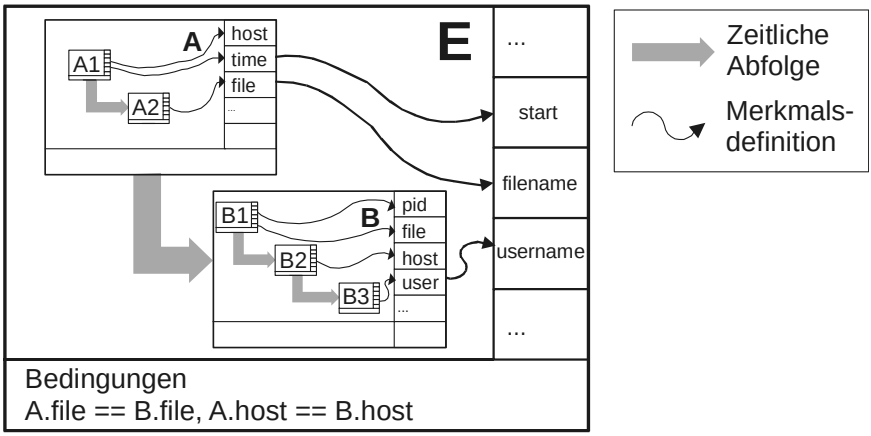


Abb. 7-7. Ereignisbeschreibung in SHEDEL

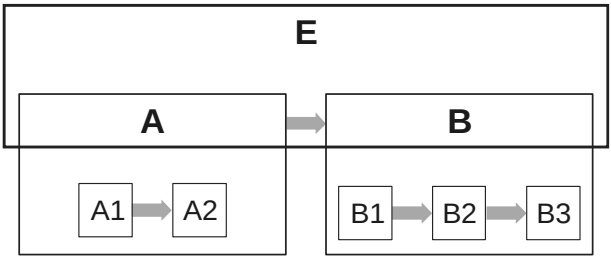


Abb. 7-8. Ereignishierarchie in SHEDEL

Die zugehörige textuelle SHEDEL-Beschreibung ist in Abb. 7-9 angegeben. Sie verdeutlicht die getrennte Beschreibung von Intra- und Inter-Ereignis-Bedingungen. Die Intra-Ereignis-Bedingung von Schritt *step2*, die Übereinstimmung des Ereignismerkmals *user* mit dem Wert *„root“* fordert, wird bei der Schrittbeschreibung spezifiziert. Inter-Ereignis-Bedingungen werden hingegen in einem separaten durch das Schlüsselwort *CONDITIONS* eingeleiteten Abschnitt angegeben. Im letzten Teil einer SHEDEL-Beschreibung, der mit dem Schlüsselwort *FEATURES* beginnt, werden die Merkmale des Ereignistyps *E* sowie ihre Belegung durch Schrittmerkmale festgelegt.

Den Schritten der Ereignisbeschreibung können Aktionen zugeordnet werden, die zur Ausführung kommen, wenn ein dem Typ des Schrittes entsprechendes Ereignis auftritt und die spezifizierten Bedingungen erfüllt sind. Sind alle mit der beschriebenen Schrittfolge assoziierten Ereignisse in der entsprechenden zeitlichen Folge aufgetreten und die angegebenen Bedingungen erfüllt, werden die Merkmale des Ereignisses entsprechend der Beschreibung belegt und das Ereignis wird ausgelöst. In obigem Bei-

spiel bedeutet dies, folgt nach einem Ereignis *A* ein Ereignis *B* und sind die spezifizierten Bedingungen erfüllt, so impliziert dies ein Ereignis vom Typ *E*. Eine ausführlichere Diskussion der Sprachmerkmale ist in [Mei+02] enthalten.

```
EVENT E {
  STEP step1
    INITIAL  // Eingangsschritt
    TYPE A
  STEP step2
    EXIT     // Ausgangsschritt
    TYPE B
      user =”root”
    REQUIRES step1
    ACTIONS
      ...
  CONDTIONS
    step1.file == step2.file,
    step1.host == step2.host
  FEATURES
    start = step1.time,
    filename = step1.file,
    username = step2.user
    ...
}
```

Abb. 7-9. Textuelle Ereignisbeschreibung

Zur Verarbeitung und Analyse von Signaturen in SHEDEL wurde das Werkzeug *StraFER* (*Straight Forward Event Recognition*) [HoMei+02a, 02b] entwickelt, dessen Vorgehen im Folgenden skizziert wird. Eine detaillierte Beschreibung des Verfahrens ist in [Bi01] enthalten. StraFER kompiliert SHEDEL-Beschreibungen zunächst in ein internes Format. Während der Verarbeitung durch StraFER werden eingehende Audit-Datensätze auf entsprechende Ereignisse abgebildet. Nachdem das Eintreten eines Ereignisses signalisiert wurde, werden alle vorliegenden Ereignisbeschreibungen, die entsprechende Ereignisse bzw. Schritte dieses Typs verwenden, über das Eintreten des Ereignisses informiert. Sind alle durch eine Ereignisbeschreibung spezifizierten Bedingungen erfüllt, so wird das Eintreten eines Ereignisses dieser Ereignisbeschreibung signalisiert. Nachdem alle als eingetreten angezeigten Ereignisse verarbeitet wurden, wird das nächste eingehende Audit-Record als Ereignis angezeigt und die Prozedur beginnt von vorn.

7.3.2 Beispiele

Durch die in SHEDEL gewählte Form der Beschreibung wird die Definition einer Ereignishierarchie möglich, die es gestattet, Abstraktionen einzuführen, die zur Vereinfachung der Signaturbeschreibung verwendet werden können. Zum einen erlaubt dieser Mechanismus komplexe Signaturen in einfachere Teilsignaturen zu zerlegen und als Komposition dieser zu beschreiben. Zum anderen ist es dadurch leicht möglich, Signaturen für so genannte Meta-Attacken darzustellen, deren Teilschritte selbst Signaturen von Attacken darstellen. Außerdem kann dieser Mechanismus zur Verallgemeinerung oder Spezialisierung von Ereignissen verwendet werden, wie die folgenden Beispiele verdeutlichen.

Verallgemeinerung von Ereignissen

Eine Ursache für die Kompliziertheit von Signaturbeschreibungen ist die teilweise ungeeignete Kodierung von Informationen in Audit-Daten [Ka01] (vgl. auch Abschn. 3.2). Beispielsweise müssen durch eine Signatur, die nach fehlerhaften Anmeldeversuchen in einer Windows NT/2000/XP-Umgebung sucht, zehn verschiedene Typen von Audit-Einträgen betrachtet werden (die Ereignistypen mit den Nummern 529 bis 537 und 539). In einer Signatur für die Login-Attacke müssen alle zehn Ereignistypen berücksichtigt werden. Das kann vereinfacht werden, indem eine Abstraktion für diese Ereignisse eingeführt wird. Diese neue Ereignisbeschreibung *FailedLogin* kann als Verallgemeinerung für fehlerhafte Anmeldeversuche in allen Signaturen verwendet werden, die nach derartigen Ereignissen suchen. Die SHEDEL-Beschreibung für das neue *FailedLogin*-Ereignis ist in Abb. 7-10 skizziert. Der Vorteil von SHEDEL ist hier, dass ohne großen Aufwand bestimmte Ereignisse in einem Ereignistyp zusammengefasst werden können. Dadurch kann der Signaturprogrammierer Abstraktionen konstruieren, mit denen die Konzepte von Sicherheitsverletzungen bzw. Signaturen einfacher und übersichtlicher beschrieben werden können, als auf der Grundlage einzelner Audit-Ereignisse.

Spezialisierung von Ereignissen

Die Spezialisierung von Ereignissen ist ein anderer Weg zur Vereinfachung von komplexen Signaturen. Beispielsweise verfolgen viele Signaturen Systemaktionen die Nutzern spezielle Privilegien zuordnen. Typischerweise existiert eine Reihe von Signaturen, die nach fehlerhaften Anmeldungen des Administrator-Kontos suchen. In diesem Fall erweist es sich als sinnvoll, ein Ereignis einzuführen, das diese Situation beschreibt.

Ein solches Ereignis *AdministratorFailedLogin* kann durch Spezialisierung der obigen Ereignisbeschreibung *FailedLogin*, wie in Abb. 7-11 skizziert, realisiert werden. Dieses Beispiel unterstreicht nochmals den Vorteil von SHEDEL, ohne großen Aufwand „maßgeschneiderte“ Abstraktionen zur Beschreibung von Signaturen erstellen zu können.

```
EVENT FailedLogin {
  // Jeder auftretende Einzelschritt löst eine
  // FailedLogin-Ereignis aus.
  STEP failure529  INITIAL EXIT
  TYPE EVENT_NO_529  // Anmeldeereignis 529

  STEP failure530  INITIAL EXIT
  TYPE EVENT_NO_530  // Anmeldeereignis 530

  ...

  STEP failure539  INITIAL EXIT
  TYPE EVENT_NO_539  // Anmeldeereignis 539

  FEATURES
    hostname = (failure529.host | ... | failure539.host)
    username = (failure529.user | ... | failure539.user)
  ...
}
```

Abb. 7-10. SHEDEL-Beschreibung *FailedLogin* (Skizze)

```
EVENT AdministratorFailedLogin {
  STEP failure
  INITIAL EXIT
  TYPE FailedLogin
  CONDITIONS
    // nur fehlerhafte Anmeldungen des Administrators
    failure.username == ”Administrator”
  FEATURES
    ...
}
```

Abb. 7-11. SHEDEL-Beschreibung *AdministratorFailedLogin* (Skizze)

SHEDEL-Signatur für die Link-Attacke

Als komplexeres Beispiel wird in diesem Abschnitt die SHEDEL-Beschreibung einer Signatur zur Erkennung der Link-Attacke (vgl. Abschn. 4.2.3) dargestellt. Zur Durchführung dieser Attacke erstellt ein Angreifer

einen Link auf ein SUID-Skript, wobei der Name des Links mit einem Bindestrich beginnt. Durch Ausführung des Links gelingt es dem Angreifer eine interaktive Shell mit den Rechten und Privilegien des Skripteigentümers zu erhalten. Eine Illustration der SHEDEL-Beschreibung der Signatur zur Erkennung dieser Attacke ist in Abb. 7-12 dargestellt. Die Abbildung stellt die Schritte der Beschreibung sowie die zeitlichen Beziehungen zwischen den Schritten durch Pfeile dar.

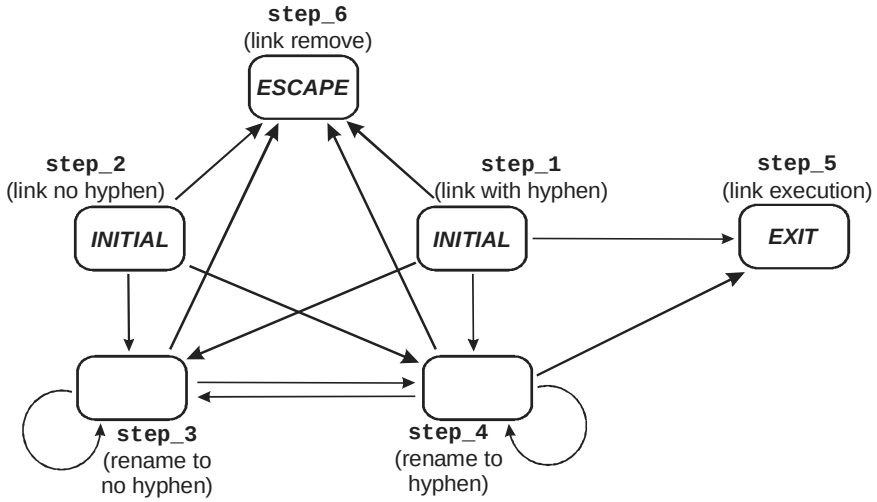


Abb. 7-12. Schema der SHEDEL-Beschreibung einer Beispielsignatur

Eine verkürzte textuelle Beschreibung dieser Signatur ist in Abb. 7-13 dargestellt. Die Schritte *step_1* und *step_2* sind initiale Schritte. Sie suchen nach der Erzeugung von Links, die auf ein SUID-Skript verweisen. Die dazu verwendeten Kriterien sind in dem Ereignistyp *create_link_to_shell_-script* gekapselt. Während *step_1* nach Links mit führendem Bindestrich sucht, verfolgt *step_2* Links ohne diesen Namensteil. Die Umbenennung eines verdächtigen Links wird durch die Schritte *step_3* und *step_4* nachvollzogen. Der finale Schritt *step_5* überprüft die Ausführung eines Links mit einem Bindestrich am Namensanfang. Durch den Abbruchschritt *step_6* wird das Löschen verdächtiger Links verfolgt. Die Korrelation verschiedener Schritte anhand des Linknamens spiegelt sich in den Inter-Ereignis-Bedingungen wider.

```
EVENT linkAttack {
  STEP step_1 // Link mit '-' erzeugt
    INITIAL
    TYPE  create_link_to_shellscript
      NAME=="-*"  // Link-Name beginnt mit '-'
  STEP step_2 // Link ohne '-' erzeugt
    INITIAL
    TYPE  create_link_to_shellscript
      NAME!="-*"  // Link-Name beginnt nicht mit '-'
  STEP step_3 // Umbenennung des Links zu ohne '-'
    TYPE  rename
      NEWNAME!="-*"  // Neuer Name beginnt nicht mit '-'
    REQUIRES step_1 OR step_2 OR step_3 OR step_4
  STEP step_4 // Umbenennung des Links zu '-'
    TYPE  rename
      NEWNAME=="-*"  // Neuer Name beginnt mit '-'
    REQUIRES step_1 OR step_2 OR step_3 OR step_4
  STEP step_5 // Ausführen des Links
    EXIT
    TYPE  execution
    REQUIRES step_2 OR step_4
    ACTIONS ...
  STEP step_6 // Löschen des Links
    ESCAPE
    TYPE  remove
    REQUIRES step_1 OR step_2 OR step_3 OR step_4
  CONDITIONS
    // Der umbenannte, gelöschte oder ausgeführte Link und
    // der in step_1 oder step_2 erzeugte Link besitzen
    // den gleichen Namen
    step_1.linkname == step_3.oldname,
    step_1.linkname == step_4.oldname,
    ...
    // Der Name des umzubennenden Links stimmt mit dem
    // Ergebnis der letzten Umbennung überein
    step_3.newname == step_4.oldname,
    step_4.newname == step_3.oldname,
    [step_3].newname == step_3.oldname,
    ...
  FEATURES
    ...
}
```

Abb. 7-13. SHEDEL-Signatur für die Link-Attacke (Skizze)

7.3.3 Diskussion

Mit SHEDEL wurde ein erster Ansatz zur Vereinfachung der Signaturbeschreibung vorgestellt. Im Unterschied zu den existierenden Sprachen STATL, Sutekh, CPA und MuSig bietet SHEDEL Sprachmittel zur Einführung von Abstraktionen, die eine Strukturierung von Signaturbeschreibungen ermöglichen.

Auch bezüglich der Ausdrucksstärke hinsichtlich der Charakterisierung semantischer Aspekte von Signaturen (vgl. Kapitel 5) ist SHEDEL den genannten Sprachen überlegen. Lediglich die mit CPA und MuSig beschreibbaren Konjunktionen und simultanen Ereignisse werden von SHEDEL nicht unterstützt.

Zur Beschreibung von Ereignismustern von Signaturen unterstützt SHEDEL Sequenzen und Disjunktionen. Unter Verwendung von Abbruchschritten können darüber hinaus Negationen ausgedrückt werden. Der Aspekt Häufigkeit von Ereignissen wird in der Ausprägung *mindestens* direkt unterstützt. Die Modi *genau* und *höchstens* können durch die zusätzliche Verwendung von Abbruchschritten umgesetzt werden. Von den existierenden Sprachen bietet nur STATL eine Unterstützung der drei Modi.

Der Instanzselektionsmodus *erste* wird von SHEDEL direkt realisiert, die Modi *letzte* und *alle* können durch Schleifenkonstrukte und entsprechende Merkmalsbelegungen umgesetzt werden. Analoge Beschreibungsmöglichkeiten bietet die Sprache STATL, wohingegen CPA, MuSig und Sutekh keine Differenzierung der Instanzselektionsmodi erlauben. Für die mit SHEDEL beschreibbaren Ereignismuster werden die Konsummodi *konsumierend* und *nicht-konsumierend* unterstützt. STATL unterstützt diese Modi ebenfalls. Alle anderen Sprachen bieten keine adäquate Unterstützung von Instanzkonsum.

SHEDEL behebt nicht alle Defizite existierender Sprachen. Verschiedene der in Kapitel 5 diskutierten semantischen Aspekte von Signaturen, z. B. Ereignismuster und Nebenläufigkeit, können nicht vollständig mit SHEDEL charakterisiert werden. Ursache dafür ist, dass die Entwicklung von SHEDEL der Systematisierung der semantischen Aspekte von Signaturen vorausging und die Entwicklung des in Kapitel 5 vorgestellten Modells auch von Erfahrungen aus der Entwicklung und Verwendung von SHEDEL getrieben wurde. Ereignismuster, die Konjunktionen und Simultanität verwenden, können in SHEDEL nicht dargestellt werden. Dadurch ist eine kompakte Beschreibung von Signaturen, die nach nebenläufigen Aktionsfolgen suchen, nicht möglich. Stattdessen müssen alle möglichen Aktionsfolgen beschrieben werden. Des Weiteren wird der Nebenläufigkeitsmodus *nicht-überlappend*, d.h. die sequentielle Komposition von komplexen Ereignissen, durch SHEDEL nicht unterstützt. Eine Dekompo-

sition komplexer Signaturen in eine Sequenz von Teilfolgen (Teilereignissen) ist nur eingeschränkt möglich. Weitere Erfahrungswerte aus der Entwicklung und Verwendung von SHEDEL sind, dass die strukturelle Trennung der Inter-Ereignis-Bedingungen und Merkmalsbelegungen von den involvierten Schritten in einzelnen Fällen zu unübersichtlichen Beschreibungen führen kann.

Zusammenfassend kann festgehalten werden, dass SHEDEL für eine Reihe von Signaturen, insbesondere Sequenzen, eine sehr einfache und strukturierte Beschreibung erlaubt und existierenden Sprachen überlegen ist. In ihrer Ausdrucksstärke ist die Sprache jedoch ebenfalls eingeschränkt und sie kann nicht alle Aspekte von Signaturen charakterisieren. Es ist eine Sprache notwendig, die eine vollständige Beschreibung von Signaturen erlaubt.

7.4 EDL

Ausgehend von den Erfahrungen mit SHEDEL wurde die Sprache *EDL* (*Event Description Language*) [Mei+05a, Mei+05b] konzipiert. Sie wurde mit dem Ziel entwickelt, alle Aspekte des semantischen Modells von Signaturen beschreiben zu können und gleichzeitig die mit SHEDEL erreichte einfache Beschreibung beizubehalten.

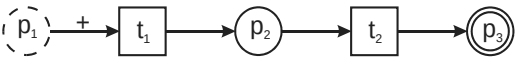
Die in Kapitel 6 eingeführten Signaturnetze erlauben eine Modellierung aller Aspekte des semantischen Modells von Signaturen (vgl. Abschn. 6.5). Um diese Ausdrucksstärke zu erreichen, orientiert sich der Sprachentwurf von EDL an den Konzepten dieses Modellierungsansatzes. Nachfolgend werden die grundlegenden Konzepte der Sprache vorgestellt und die wesentlichen Unterschiede zu Signaturnetzen diskutiert.

7.4.1 Basiskonzepte

Die Sprache EDL bietet, dem Modellierungsansatz von Signaturnetzen entsprechend, Konstrukte zur Beschreibung von Plätzen und Transitionen, die hier grundlegend beschrieben werden. Eine ausführliche Beschreibung der Sprache ist in [Schme04, Mei+05b] enthalten.

Die EDL-Beschreibung einer Signatur ist grundsätzlich als eine Liste von benannten Plätzen, gefolgt von einer Liste von Transitionen organisiert, wie Abb. 7-14 zeigt. Die Abbildung skizziert auf der linken Seite ein Signaturnetz einer Beispielsignatur und stellt auf der rechten Seite das Gerüst der entsprechenden EDL-Beschreibung dar. Die Beschreibungsmittel von EDL zur Charakterisierung von Plätzen und Transitionen sowie zur

Strukturierung von Signaturbeschreibungen werden nachfolgend dargestellt.



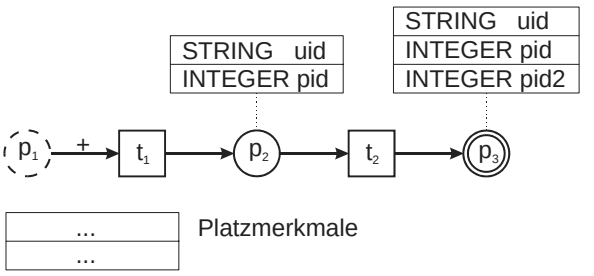

```
EVENT sigName
{
  PLACES
    p1
    { ... }
    p2
    { ... }
    p3
    { ... }
  TRANSITIONS
    // T1
    p1(+) p2
    { ... }

    // T2
    p2(-) p3
    { ... }
}
```

Abb. 7-14. Schema einer EDL-Beschreibung

Plätze

Analog zu Signaturnetzen unterscheidet EDL die vier Platzarten *Initial*-, *Interior*-, *Final*- und *Abbruch*-Platz. Jede EDL-Beschreibung einer Signatur besitzt mindestens einen Initial- und genau einen Abbruchplatz. Jeder Platz besitzt einen eindeutigen Namen. Interior- und Final-Plätze sind außerdem durch Merkmale gekennzeichnet. Die *Merkmale* eines Platzes definieren, welche Variablen der Token auf diesem Platz belegt sind. Alle Token auf einem Platz belegen dieselbe Menge von Merkmalen unabhängig davon, ob sie den Platz auf alternativen Pfaden erreicht haben. Für Merkmale werden die Datentypen *Bool*, *Integer*, *Float* und *String* unterstützt. Platzmerkmale können in einer EDL-Beschreibung durch Angabe des Platz- und des Merkmalsnamens referenziert werden. Abb. 7-15 veranschaulicht links die Platzmerkmale an einer Skizze eines Signaturnetzes und stellt rechts die EDL-Beschreibung zur Deklaration der entsprechenden Plätze dar.

Platz p_1 ist Initialplatz und besitzt als solcher keine Merkmale. Dem Interiorplatz p_2 sind die Merkmale *uid* vom Datentyp *String* und *pid* vom Datentyp *Integer* zugeordnet. Der Exit-Platz p_3 besitzt ein Merkmal *uid* vom Typ *String* sowie die *Integer*-Merkmale *pid* und *pid2*.

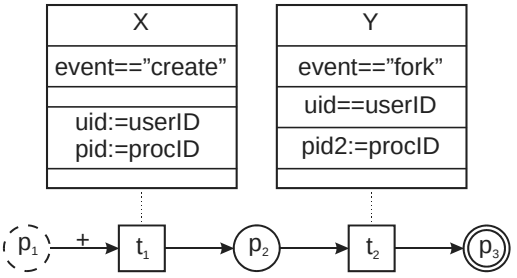



```
...
PLACES
  p1
  {
  TYPE INITIAL
  }
  p2
  {
  FEATURES
    STRING  uid,
    INTEGER pid
  }
  p3
  {
  TYPE EXIT
  FEATURES
    STRING  uid
    INTEGER pid,
    INTEGER pid2
  }
...
```

Abb. 7-15. Beschreibung von Plätzen in EDL

Transitionen

Transitionen einer EDL-Beschreibung besitzen alle Eigenschaften von Transitionen in Signaturnetzen, nämlich einen Ereignistyp, über den Transitionen auch als spontan deklariert werden können, sowie Bedingungen, Merkmalsbindungen und Aktionen, die sowohl Platz- als auch Ereignismerkmale referenzieren können. Durch die Merkmalsbindungen einer Transition wird festgelegt, wie die Merkmale der Nachplätze belegt werden. Zusätzlich wird für jede Transition eine Liste von Vorplatznamen und Nachplatznamen angegeben, durch die die Eingangs- und Ausgangskanten der Transition spezifiziert werden. Jeder Vorplatzname wird dabei mit dem Zeichen + oder – gekennzeichnet, um die entsprechende Eingangskante als *konsumierend* oder *nicht-konsumierend* zu deklarieren. Abb. 7-16 stellt links die Transitionsbeschriftungen eines Signaturnetzes und rechts die entsprechende EDL-Beschreibung der Transitionen dar.




```
...
TRANSITIONS
  // T1
  p1(+) p2
  {
  TYPE X
  CONDITIONS
  event==”create”
  MAPPINGS
    [p2].uid=userID,
    [p2].pid=procID
  ACTIONS
    ...
  }
  // T2
  p2(-) p3
  {
  TYPE Y
  CONDITIONS
    event==”fork”,
    p2.uid==userID
  MAPPINGS
    [p3].uid=p2.uid,
    [p3].pid=p2.pid,
    [p3].pid2=procID
  ACTIONS
...
}
...
```

Abb. 7-16. Beschreibung von Transitionen in EDL

Die erste Transition ist über eine nicht-konsumierende Kante mit dem Vorplatz p_1 verbunden und besitzt p_2 als Nachplatz. Der Transition ist der Ereignistyp *X* sowie eine Intra-Ereignis-Bedingung zugeordnet, die das Merkmal *event* des assoziierten Ereignisses auf Übereinstimmung mit der Konstante „*create*“ überprüft. Außerdem werden durch die Transition t_1 des Signaturnetzes zwei Tokenvariablen gebunden. Dies entspricht in der EDL-Beschreibung der Belegung von Platzmerkmalen der Nachplätze. Durch die Transition werden die Merkmale *uid* und *pid* des Nachplatzes p_2 mit den Ereignismerkmalen *userID* respektive *procID* belegt. Die zweite Transition besitzt den Vorplatz p_2, mit dem sie über eine konsumierende Kante verbunden ist, und den Nachplatz p_3. Sie ist mit dem Ereignistyp *Y* assoziiert und beschreibt zwei Bedingungen, von denen eine Übereinstimmung des Ereignismerkmals *event* mit dem Wert „*fork*“ fordert. Die Inter-Ereignis-Bedingung der Transition im Signaturnetz fordert Übereinstimmung der Tokenvariablen *uid* mit dem Ereignismerkmal *userID*. Die Ent-

sprechung in der EDL-Beschreibung spezifiziert eine Bedingung bzgl. des entsprechenden Vorplatzmerkmals *p2.uid* und dem Ereignismerkmal *user-ID*. Die Transition t_2 im Signaturnetz beschreibt außerdem die Belegung der Tokenvariable *pid2* mit dem Ereignismerkmal *procID*, die auf eine entsprechende Belegung eines Nachplatzmerkmals in der EDL-Beschreibung abgebildet wird. Des Weiteren enthält die EDL-Beschreibung Belegungen von Nachplatzmerkmalen (*p3.uid* und *p3.pid*) mit Merkmalen des Vorplatzes (*p2.uid* bzw. *p2.pid*). Im Unterschied zu Signaturnetzen, bei denen gebundene Tokenvariablen von Transition zu Transition aufkumuliert werden, ist in EDL-Beschreibungen explizit zu spezifizieren, welche Vorplatzmerkmale auf welche Nachplatzmerkmale übernommen werden.

Modulkonzepte

EDL übernimmt das bewährte Hierarchiekonzept aus SHEDEL. Eine EDL-Beschreibung einer Signatur definiert einen Ereignistyp mit dem Namen der Signatur. In anderen Signaturbeschreibungen können Transitionen mit diesem Ereignistyp assoziiert werden. Die Merkmale eines Ereignisses dieses Typs werden durch die Merkmale des Finalplatzes der EDL-Beschreibung definiert. Abb. 7-17 veranschaulicht dieses Prinzip. Damit sind die von SHEDEL bekannten Abstraktionsmöglichkeiten auch für EDL-Beschreibungen realisierbar.

Der Verwendung einer solchen Ereignishierarchie liegt eine nebenläufige Komposition von komplexen Ereignissen zugrunde, die bzgl. des Aspekts Nebenläufigkeit der Semantik von Signaturen (vgl. Abschn. 5.4.4) dem Modus *überlappend* entspricht. Dementsprechend ist das Ereignismuster des Ereignisses *E* aus Abb. 7-17 äquivalent zum Ereignismuster des in Abb. 7-18 dargestellten Signaturnetzes.

Unter dem Gesichtspunkt der Wiederverwendung von Teilsignaturen ist häufig auch die sequentielle Komposition, also der Nebenläufigkeitsmodus *nicht-überlappend*, erforderlich. Zur Umsetzung dieses Modus wurden in EDL als weiteres Modulkonzept Makros integriert. Ein *Makro* entspricht einem platzberandeten Teilsignaturnetz und kann mehrere Eingangs- und Ausgangsplätze besitzen. Definierte Makros können wie Ereignistypen verwendet werden. Die Verwendung eines Makros erfolgt durch *Makrotransitionen*, die mit einem Makro als Ereignistyp assoziiert sind. Die Vor- bzw. Nachplätze einer Makrotransition werden als Makrovor- bzw. Makronachplätze bezeichnet. Abb. 7-19 veranschaulicht die Definition eines Makros *B* und dessen Verwendung in der Beschreibung des Ereignisses *E*.

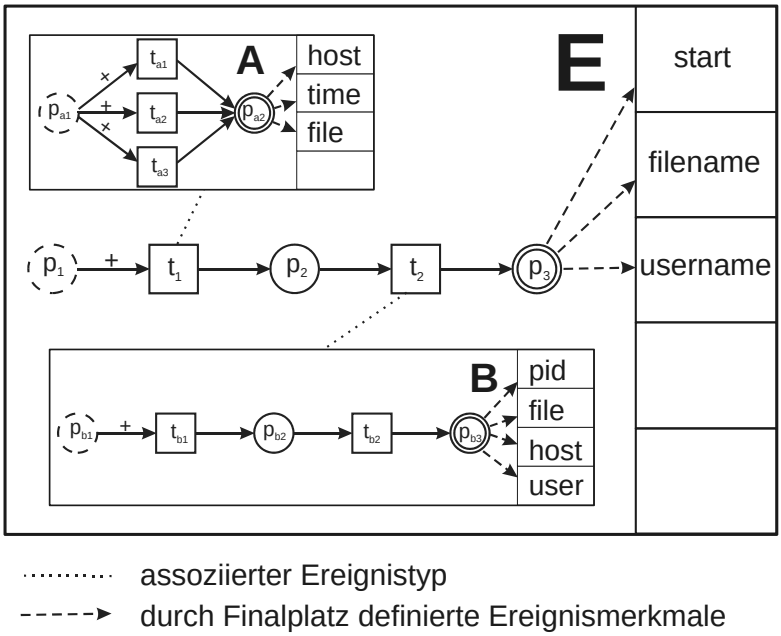


Abb. 7-17. Definition von Ereignismerkmalen in EDL

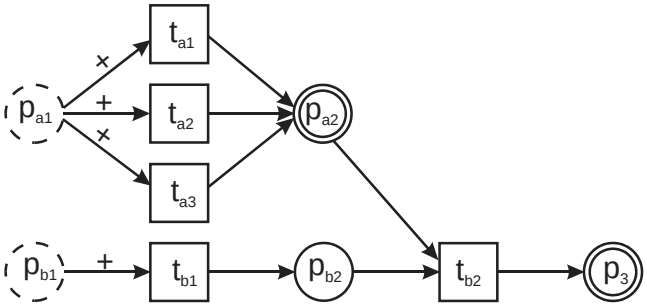


Abb. 7-18. Äquivalentes Ereignismuster zu Abb. 7-17

Bei der Verwendung von Makrotransitionen ist die Abbildung der Vorplätze auf die Eingangsplätze und der Ausgangsplätze auf die Nachplätze erforderlich. Außerdem ist zu spezifizieren, welche Merkmale der Vorplätze den Eingangsplätzen und welche Merkmale der Ausgangsplätze den Nachplätzen zugeordnet werden. Diese Form der Parametrisierbarkeit erleichtert die Entwicklung von wiederverwendbaren Makros, unabhängig von konkreten Einsatzkontexten. Außerdem kann die direkte Abbildung von Makrovorplatzmerkmalen auf Makronachplatzmerkmale definiert werden.

Zur Spezifikation dieser Abbildungen enthält die EDL-Beschreibung einer Makrotransition zusätzliche Abschnitte, die jeweils mit den Schlüsselwörtern

- *PLACE_MAPPINGS,*
- *MACRO_IN_MAPPINGS,*
- *MACRO_OUT_MAPPINGS und*
- *MACRO_THROUGH_MAPPINGS*

eingeleitet werden. Abb. 7-21 veranschaulicht beispielhaft eine EDL-Beschreibung für die Makrotransition aus Abb. 7-19.

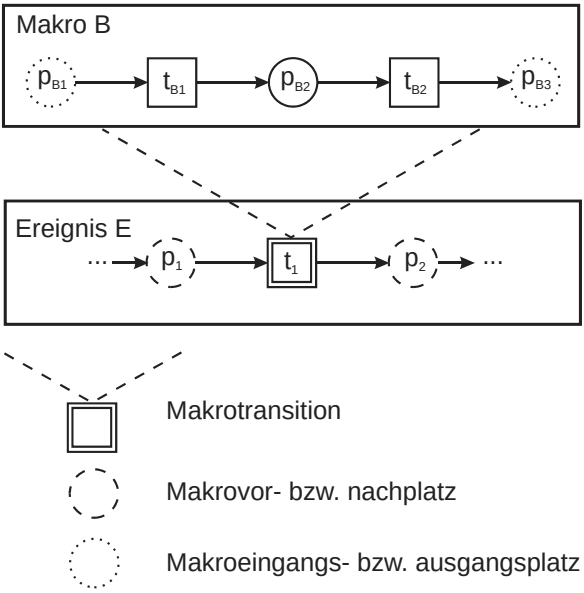


Abb. 7-19. Makrodefinition und -verwendung

Die Bedeutung einer Makroverwendung ergibt sich aus dem Ersetzen der Makrotransition durch die Makrodefinition, wobei Makroeingangs- und Makroausgangsplätze entsprechend der Beschreibung der Makrotransition durch Makrovor- bzw. Makronachplätze ersetzt werden. Abb. 7-20 veranschaulicht eine äquivalente Beschreibung des Ereignisses *E* aus Abb. 7-19 nach Auflösung bzw. Ersetzung der Makrotransition t_1 mit Makro *B*.

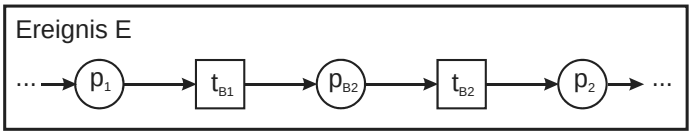


Abb. 7-20. Ereignis E nach Makroersetzung

7.4.2 Beispiel

Zur Veranschaulichung der Beschreibungsmöglichkeiten von EDL betrachten wir eine Signatur für die nebenläufige Link-Attacke (vgl. Abschn. 4.2.3). Hierbei handelt es sich um eine Variante der Link-Attacke, bei der ein Skript nicht von vornherein mit dem SUID-Bit versehen ist, sondern dieses nebenläufig zu anderen Aktionen der Attacke gesetzt wird.

Um diese nebenläufigen Aktionen mit einer Signatur zu verfolgen, müssen in Sprachen wie SHEDEL, STATL oder Sutekh alle möglichen Reihenfolgekombinationen der entsprechenden Aktionen beschrieben werden, was die Handhabbarkeit und Überschaubarkeit der Signaturen erschwert. Hingegen bietet EDL Beschreibungskonstrukte, die eine kompakte Beschreibung der nebenläufigen Aktionen ermöglichen. Abb. 7-22 skizziert das entsprechende Signaturnetz für diese Signatur, die im Folgenden kurz erörtert wird. Die vollständige textuelle EDL-Beschreibung der Signatur findet sich in Anhang A.

Da die Erstellung eines Links auf ein Shell-Skript und das Setzen des SUID-Bits zeitlich unabhängig voneinander ausgeführt werden können, werden die jeweiligen Ereignisse im Signaturnetz durch nebenläufige Transitionen modelliert. Die Transitionen t_1 und t_2 verfolgen die Erstellung von Links auf Shell-Skripte. Transition t_3 überprüft die Erstellung von SUID-Skripten. Token auf den Plätzen p_4 und p_5 beschreiben die Vorbedingungen für die erfolgreiche Durchführung der Attacke durch Ausführen eines Links. Ein Token auf p_4 beschreibt die Existenz eines entsprechend benannten Links auf ein Shell-Skript. Token auf p_5 repräsentieren erstellte Shell-Skripte mit gesetztem SUID-Bit. Bei Auftreten der Ausführung eines Objekts (Transition t_{11}) wird überprüft, ob auf Platz p_4 ein Token für das ausgeführte Objekt liegt (Ereignismerkmal Objektname und Tokenvariable Linkname stimmen überein). In diesem Fall handelt es sich bei dem Objekt um einen Link. Außerdem wird überprüft, ob Paare von Token auf den Plätzen p_4 und p_5 existieren, die sich auf das gleiche Skript beziehen. In diesem Fall existiert ein Link auf ein Skript mit SUID-Bit. Alle weiteren Transitionen des Signaturnetzes dienen der Verfolgung von Aktionen, die

eine Umbenennung des Links darstellen oder eine erfolgreiche Vervollständigung von Attackeninstanzen unmöglich machen.

```
MACRO B {
  PLACES
    pb1 // Makrostartplatz
    { ... }
    pb2
    { ... }
    pb3 // Makroendplatz
    { ... }
  TRANSITIONS
    // tb1
    pb1(-) pb2
    { ... }
    // tb2
    pb2(-) pb3
    { ... }
}

EVENT E
{
  PLACES
    ...
    p1
    { ... }
    p2
    { ... }
    ...

  TRANSITIONS
    p1(-) p2
    {
      TYPE B // Makrotransition
      PLACE_MAPPINGS
        pb1 = p1 // Startplatz = Vorplatz
        pb3 = p2 // Endplatz = Nachplatz
      MACRO_IN_MAPPINGS
        [pb1].m1 = p1.m1
      MACRO_OUT_MAPPINGS
        [p2].m2 = pb3.m2
      MACRO_THROUGH_MAPPINGS
        [p2].m3 = p1.m3
    }
  ...
}
```

Abb. 7-21. Makrodefinition und -verwendung in EDL

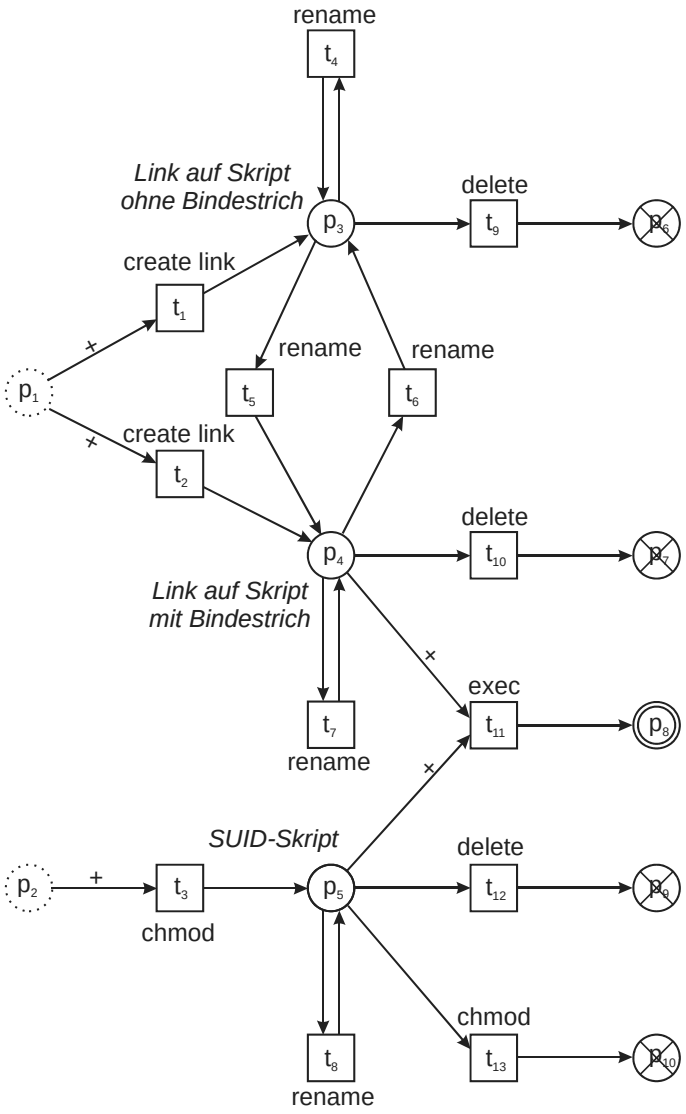


Abb. 7-22. Signaturnetz für die nebenläufige Link-Attacke (Skizze)

7.4.3 Diskussion

Da sich die Beschreibungssprache EDL weitgehend an den Signaturnetzen orientiert, besitzt EDL die gleiche Ausdrucksstärke wie Signaturnetze hinsichtlich der Charakterisierung semantischer Aspekte von Signaturen und

ist damit existierenden Beschreibungssprachen überlegen. Zusätzlich bietet die Sprache EDL beschreibungstechnische Mittel zur Modularisierung. Es können hierarchische Ereignisse und Makros definiert und verwendet werden. Durch entsprechende Verwendung dieser Modularisierungskonzepte unterstützt EDL auch die beiden Ausprägungen *überlappend* (hierarchisches Ereignis) und *nicht-überlappend* (Makro) des Semantikaspekts Nebenläufigkeit.

Die Kombination aus Platzname und Platzmerkmalen bildet die Entsprechung für Tokenvariablen in Signaturnetzen. Da Platznamen eindeutig sind, ist für jede Menge von Token von verschiedenen Plätzen der Durchschnitt aller Tokenvariablen leer und somit ist diese Tokenmenge unifizierbar. Das bedeutet, dass das Unifikationskonzept von Signaturnetzen in EDL-Beschreibungen nicht zur Formulierung von Gleichheitsbedingungen führt. Dadurch wird verhindert, dass durch eine zufällige oder fehlerhafte Gleichbenennung von Merkmalen von Vorplätzen einer Transition eine Übereinstimmung entsprechender Tokenvariablen gefordert wird. Sämtliche Bedingungen sind in EDL explizit als solche zu beschreiben.

Durch unterstützende graphische Werkzeuge wird eine deutliche Erleichterung der Signaturbeschreibung erwartet. Es liegen positive Erfahrungen mit einen prototypischen Werkzeug zur graphischen Beschreibung von SHEDEL-Signaturen vor [Da02]. Mit Signaturnetzen existiert bereits eine graphische Notation für Signaturen, die von einem entsprechenden Werkzeug zur Beschreibung von EDL-Signaturen übernommen werden kann. Entsprechende Arbeiten, bei denen ausgehend von dem generischen Graph-Werkzeug SNOOPY [Fie04] das Werkzeug SEG (Snoopy-based EDL-GUI) zur teilweise graphischen Entwicklung von EDL-Signaturen entwickelt werden soll, sind derzeit im Gange [Ro06]. Eine Vorabversion dieses Werkzeugs ist auf der beiliegenden CD-ROM enthalten.

7.5 Alternative Beschreibungszugänge

Die Beschreibung von Signaturen umfasst die Charakterisierung der semantischen Aspekte wie *Typ und Reihenfolge* und *Kontextbedingungen* (vgl. Kapitel 5) der Teilereignisse sowie die Zuordnung von Aktionen. Dabei können grundlegend zwei Vorgehensweisen unterschieden werden. Bei der *aspektorientierten* Beschreibung wird jeder Aspekt für alle Ereignisse beschrieben. Im Unterschied dazu werden bei der *ereignisorientierten* Beschreibung für jedes Ereignis alle Aspekte beschrieben. Die aus den Vorgehensweisen resultierenden Strukturen der Beschreibung sind in Abb. 7-23 dargestellt.

Eine aspektorientierte Beschreibung würde bspw. mittels eines algebraischen Ausdrucks zunächst separat die Reihenfolge der Teilereignisse spezifizieren. In weiteren Beschreibungsabschnitten müssten dann die anderen Aspekte wie Bedingungen, Aktionen usw. für die Teilereignisse beschrieben werden. Dadurch würde die Spezifikationen eines Teilereignisses auseinander gezogen und in einem Dokument verteilt.

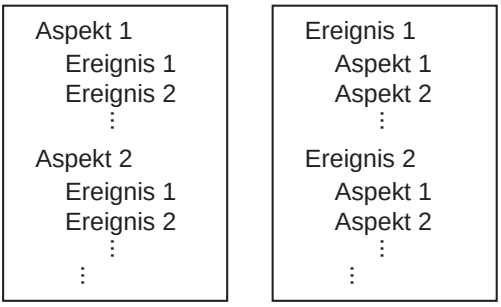


Abb. 7-23. Aspektorientierte (links) vs. ereignisorientierte (rechts) Beschreibung

Für die Beispiele bei der Einführung des Semantikmodells in Kapitel 5 wurde die Reihenfolge der Teilereignisse durch algebraische Ausdrücke dargestellt. Die Integration weiterer Ereignisaspekte in die algebraischen Ausdrücke führte jedoch bereits dort zu Unübersichtlichkeit. Die Formulierung von Bedingungen oder Aktionen innerhalb der Ausdrücke erscheint nicht sinnvoll möglich. Sie müsste in separate Beschreibungsabschnitte ausgelagert werden.

Aufgrund der zentralen Rolle des Ereigniskonzepts bei der Signaturbeschreibung und um die genannte Unübersichtlichkeit zu vermeiden, wurde SHEDEL eine ereignisorientierte Vorgehensweise zugrunde gelegt. In SHEDEL werden durch die Schritte alle Teilereignisse des komplexen Ereignisses beschrieben, wobei eine Schrittbeschreibung fast alle diesen Schritt betreffenden Aspekte charakterisiert. Eine Ausnahme bilden Inter-Ereignis-Bedingungen, die ausgelagert am Ende einer SHEDEL-Beschreibung folgen. Wie bereits in Abschn. 7.3.3 diskutiert, wird die aus dieser Trennung resultierende Unübersichtlichkeit als nachteilig empfunden.

Aus denselben Gründen folgt auch EDL weitgehend einer integrierten ereignisorientierten Beschreibung. Zwar werden in einer EDL-Beschreibung zunächst Plätze deklariert, die Beschreibung der zu betrachtenden Teilereignisse erfolgt jedoch vollständig durch die entsprechenden Transitionen, die jeweils die Aspekte eines Ereignisses charakterisieren. Hier besteht zwar der Nachteil, dass die temporale Struktur der Ereignisse bzw. Transitionen einer EDL-Beschreibung weniger offensichtlich entnommen

werden kann, die Erfahrung zeigt jedoch, dass diese Sicht vorrangig bei der Modellierung relevant ist, während sich bei der Beschreibung eine konzentrierte Darstellung der einzelnen Ereignisse als vorteilhaft erweist.

7.6 Zusammenfassung

In diesem Kapitel wurden zunächst verschiedene Aspekte, Probleme und Erfahrungen aus dem Bereich der Signaturentwicklung betrachtet bevor verschiedene Ansätze zur Beschreibung von Signaturen vorgestellt wurden. Die regelbasierte Beschreibung stellt den derzeitigen Stand der Praxis dar. Verschiedene Probleme dieses Ansatzes wurden diskutiert. Des Weiteren wurde mit SHEDEL eine Sprache vorgestellt, die insbesondere im Hinblick auf eine Vereinfachung der Signaturbeschreibung entwickelt wurde. Erfahrungen aus ihrem Einsatz flossen in die Entwicklung der vorgestellten Sprache EDL ein, die sich zum einen durch geeignete Modulkonzepte zur Einführung von Abstraktionen und Strukturierung von Signaturen und zum anderen durch ihre Ausdrucksstärke hinsichtlich der Semantik von Signaturen auszeichnet.

Des Weiteren wurden alternative Beschreibungszugänge erörtert. Ein zusammenfassender Vergleich verschiedener Erkennungssprachen bzgl. ihrer Ausdrucksstärke ist in Tabelle 7-1 dargestellt. Bei diesem Vergleich werden nur reine Erkennungssprachen betrachtet. Regelbasierte Ansätze werden aufgrund der in Abschn. 7.2.3 beschriebenen Probleme nicht dargestellt. Bei Lösung dieser Probleme, zum Beispiel durch einen der in Abschn. 7.2.3 vorgestellten Ansätze, kann mit regelbasierten Ansätzen die gleiche Ausdrucksstärke wie mit EDL bzw. mit Signaturnetzen erreicht werden, wie aus der in Abschn. 7.2.2 vorgestellten Abbildung von Signaturnetzen auf Regeln geschlossen werden kann. Allerdings geht damit entweder eine deutliche Verkomplizierung der Signaturbeschreibung einher oder der Hauptvorteil, die Verwendung existierender Regelinterpreter, verloren.

Tabelle 7-1 listet die verschiedenen semantischen Aspekte von Signaturen auf und stellt die betrachteten Signaturbeschreibungssprachen bezüglich ihrer Unterstützung der Aspekte gegenüber. Hinsichtlich des Aspekts Typ und Reihenfolge von Ereignissen unterscheiden sich die betrachteten Sprachen insbesondere bei der Unterstützung von Konjunktionen und simultanen Ereignissen. Während CPA, MuSig und EDL beides unterstützen, können entsprechende Ereignismuster in STATL, Sutekh und SHEDEL nicht beschrieben werden. Die zur Unterstützung der Negation in STATL vorhandenen Unwinding-Transitionen sind problematisch (vgl.

Abschn. 6.6.1). Die in CPA vorhandenen Invarianten-Netze erweisen sich zur Beschreibung der Negation von Ereignissen als vergleichsweise umständlich.

Tabelle 7-1. Vergleich der Ausdrucksstärke von Erkennungssprachen

Dimensionen/ Aspekte		**Ausprägungen**	**STATL**	**CPA**	**Sutekh**	**MuSig**	**SHEDEL**	**EDL**
Ereignismuster	Typ und Reihenfolge	Sequenz	✓	✓	✓	✓	✓	✓
		Disjunktion	✓	✓	✓	✓	✓	✓
		Konjunktion	○	✓	○	✓[1]	○	✓
		Simultan	○	✓	○	✓[1]	○	✓
		Negation	✓[2]	✓[3]	✓	✓	✓	✓
	Häufigkeit	Genau	✓	○	○	○	✓	✓
		Mindestens	✓	✓[4]	✓	✓	✓	✓
		Höchstens	✓	○	○	○	✓	✓
	Kontinuität	Kontinuierlich	✓	✓	✓	✓	✓	✓
		Nicht-kontinuierlich	✓[5]	✓[5]	✓[5]	✓[5]	✓[5]	✓[5]
	Nebenläufigkeit	Überlappend	○[6]	○[7]	○[7]	○[7]	○[6]	✓
		Nicht-überlappend	○[6]	○[7]	○[7]	○[7]	○[6]	✓
	Kontextbedingungen	Intra-EB	✓	✓	✓	✓	✓	✓
		Inter-EB	✓	✓	✓	✓	✓	✓
Schrittinstanzselektion		Erste	✓	✓	✓	✓	✓	✓
		Letzte	✓	○	○	○	✓	✓
		Alle	✓	○	○	○	✓	✓
Schrittinstanzkonsum		Konsumierend	✓[8]	○[9]	○[10]	○[11]	✓[8]	✓
		Nicht-konsumierend	✓[8]	○[9]	○[10]	○[11]	✓[8]	✓

Legende

✓	unterstützt	○	nicht unterstützt

[1] Mittels qualifizierter zeitlicher Beziehungen, z.B. „gleichzeitig“ (vgl. Abschn. 6.6.2);
[2] Mit „Unwinding“-Transitionen, deren Semantik problematisch ist (vgl. Abschn. 6.6.1);
[3] Unter Verwendung separater Invarianten-Netze (vgl. Abschn. 6.6.3);
[4] Mittels Sequenzen für eine entsprechende Anzahl Ereignisse (vgl. Abschn. 6.6.3);
[5] Mittels Negationen kann der Modus nicht-kontinuierlich umgesetzt werden.
[6] Nebenläufigkeitsmodi werden nicht unterstützt. Überlappend ist der Standardmodus.
[7] Module und Nebenläufigkeitsmodi werden nicht unterstützt.
[8] Nur als reine Transitionseigenschaft, was hier jedoch keine zusätzliche Einschränkung darstellt (vgl. Abschn. 6.6.1).
[9] Nur begrenzt als Zustandseigenschaft unterstützt (vgl. Abschn. 6.6.3).
[10] Nur eingeschränkt als Zustandseigenschaft unterstützt. Kann nicht spezifiziert werden, sondern wird anhand von Variablenverwendungen abgeleitet (vgl. Abschn. 6.6.1).
[11] Konsummodi werden nicht unterstützt. Konsumierend ist der Standardmodus.

Mit den Sprachen CPA, Sutekh und MuSig ist eine Spezifikation der verschiedenen Häufigkeitsmodi nicht möglich. Selbst die Spezifikation von Mindest-Häufigkeiten ist in CPA nur sehr umständlich realisierbar. Eine volle Unterstützung zur Beschreibung von Ereignishäufigkeiten bieten nur die Sprachen STATL, SHEDEL und EDL. Die Aspekte Kontinuität und Kontextbedingungen werden von allen betrachteten Sprachen gleichermaßen unterstützt. Hingegen ist EDL die einzige Sprache, die Modulkonzepte bietet und beide Nebenläufigkeitsmodi unterstützt. Eine Differenzierung der verschiedenen Instanzselektionsmodi ist nur mit den Sprachen STATL, SHEDEL und EDL möglich. Diese drei Sprachen erlauben außerdem eine Festlegung der Instanzkonsummodi für alle jeweils beschreibbaren Ereignismuster.

Dieser Vergleich zeigt, dass eine volle Unterstützung aller Aspekte nur mit der Sprache EDL möglich ist. Demnach kann festgehalten werden, dass EDL zum einen durch die unterstützten Konzepte zur Strukturierung und Vereinfachung von Signaturbeschreibungen und zum anderen hinsichtlich der Ausdrucksstärke, bisherigen Beschreibungsansätzen überlegen ist.

8 Analyseverfahren

Eine der Herausforderungen, der sich alle IDS gegenübersehen, ist die wachsende Leistungsfähigkeit sowohl der Netzwerke als auch der Endsysteme, mit der enorme Steigerungen des Aufkommens an Protokolldaten einhergehen. Darüber hinaus führt die wachsende Komplexität der IT-Systeme zu neuen Verwundbarkeiten und Angriffsmöglichkeiten, so dass die Zahl zu analysierender Signaturen sprunghaft zunimmt. Damit gewinnt die Effizienz der von IDS eingesetzten Analyseverfahren an Bedeutung. Bereits heute werden in Lastsituationen Protokolldaten von IDS verworfen oder die Erkennung von Sicherheitsverletzungen wird signifikant verzögert, so dass Gegenmaßnahmen nicht mehr oder nur noch eingeschränkt möglich sind. Um dieser Herausforderung gerecht zu werden, sind effiziente Analyseverfahren erforderlich. In der Forschung wurde der Effizienzoptimierung von Signaturanalyseverfahren bisher nur wenig Aufmerksamkeit gewidmet (vgl. Abschn. 3.2).

Kapitel 5 widmete sich der Frage, was relevante Aspekte von Angriffssignaturen sind, und es wurde ein Modell entwickelt, das eine systematische Betrachtung dieser Aspekte erlaubt. Mit den in Kapitel 6 vorgestellten Signaturnetzen wurde ein Ansatz zur Modellierung und Analyse von Signaturen eingeführt, der eine Charakterisierung aller Aspekte unterstützt, und darüber hinaus eine Simulation von Analyseabläufen ermöglicht. In Kapitel 7 wurde die Sprache EDL als geeignetes Werkzeug zur Beschreibung von Signaturen eingeführt. Mit diesen Hilfsmitteln ist es uns jetzt möglich eine dedizierte Betrachtung der Abläufe bei der Signaturanalyse durchzuführen und charakteristische Strukturen zu identifizieren, die zur Optimierung der Analyseeffizienz ausgenutzt werden können.

In diesem Kapitel werden Analyseverfahren zur Missbrauchserkennung hinsichtlich ihrer Effizienz untersucht. Zunächst werden aktuelle Verfahren vorgestellt und diskutiert. Danach werden ausgehend von ihrer Modellierung mit Signaturnetzen Charakteristika von Signaturen reflektiert, die eine Ableitung von Heuristiken zur Steigerung der Analyseeffizienz erlauben. Verschiedene Optimierungsstrategien, die strukturelle Eigenschaften von Signaturen zur Steigerung der Analyseeffizienz ausnutzen, werden vorgestellt. Zur experimentellen Evaluierung der Optimierungsstrategien wurden diese im Prototypen *SAM* (*Signature Analysis Module*) umgesetzt.

Es wurden Experimente durchgeführt, in denen die Effizienz des Prototyps mit der anderer Verfahren verglichen wurde. Die Ergebnisse der Experimente werden dargestellt und diskutiert.

8.1 Stand der Technik

Charakteristisch für die Missbrauchserkennung ist die Analyse eines kontinuierlichen Ereignisstroms auf eine Menge von in Signaturen beschriebenen Mustern. Existierende Analyseverfahren zur Missbrauchserkennung lassen sich grob in zwei Kategorien unterteilen. Verfahren der ersten Kategorie übersetzen jede Signaturbeschreibung in ein separates Programm-Modul. Die zweite Kategorie von Systemen setzt Standardexpertensysteme zur Analyse ein. Beide Ansätze werden im Folgenden diskutiert.

8.1.1 Abbildung in separate Programm-Module

Beispiele für Vertreter dieser Kategorie sind die STAT Toolsuite [Vi+00] und das IDIOT-IDS [Ku95, Cro+96]. Während STAT die zustandstransitionsbasierte Signaturbeschreibungssprache STATL [Eck+02] (vgl. auch Abschn. 7.1) zur Beschreibung von Signaturen verwendet, werden Signaturen für das IDIOT-IDS mit Hilfe gefärbter Petrinetze spezifiziert (vgl. auch Abschn. 7.1). Beide Systeme verfügen über einen entsprechenden Compiler, der die Signaturspezifikationen in C++-Klassenmodule übersetzt. Dabei wird für jede Signatur eine separate Klasse angelegt und in jeweils eine Shared Library übersetzt. Die Systeme verwalten intern eine Liste verfügbarer Signaturbibliotheken. Für verschiedene Instanzen einer Signatur wird zur Laufzeit jeweils eine Instanz des entsprechenden Klassenmoduls erzeugt. Während der Analyse werden die Audit-Ereignisse nacheinander an die einzelnen Signaturbibliotheken und die erzeugten Klasseninstanzen übergeben, welche das aktuelle Ereignis entsprechend der spezifizierten Kriterien analysieren.

Zur Verdeutlichung dieses Ansatzes wird das Teilsignaturnetz in Abb. 8-1 betrachtet. Für eine entsprechende Signatur wird durch die Compiler der Systeme ein Klassenmodul generiert, das in Abb. 8-3 als C++-Quelltextfragment veranschaulicht ist. Die Überprüfung des Ereignistyps sowie der Bedingungen von Transition t_1 wurden in der Methode `checkTransition1()` gekapselt. Das Schalten der Transition wird durch die Methode `executeTransition1()` realisiert. Beide erhalten das zu überprüfende Audit-Ereignis als Parameter. Zur Analyselaufzeit existiert für jede Instanz dieser Signatur ein Objekt der Klasse `Signa-`

`ture1`. Für jedes zu überprüfende Audit-Ereignis wird die Methode `analyseEvent()` aufgerufen, die in Abhängigkeit des Zustandes der Instanzen die entsprechenden Transitionen überprüft und ggf. schalten lässt.

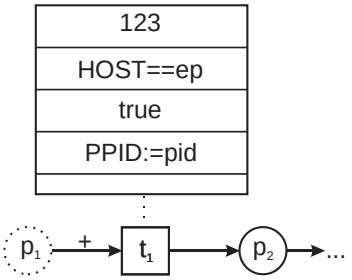


Abb. 8-1. Teilsignaturnetz einer Beispielsignatur

Der Gesamtablauf der Analyse ist in Abb. 8-2 veranschaulicht. Bereits bei Betrachtung des grundlegenden Analyseablaufs kann festgestellt werden, dass verschiedene Signaturen und Signaturinstanzen jeweils unabhängig voneinander analysiert werden. Damit können Redundanzen nicht vermieden werden, was die Effizienz des Ansatzes infrage stellt.

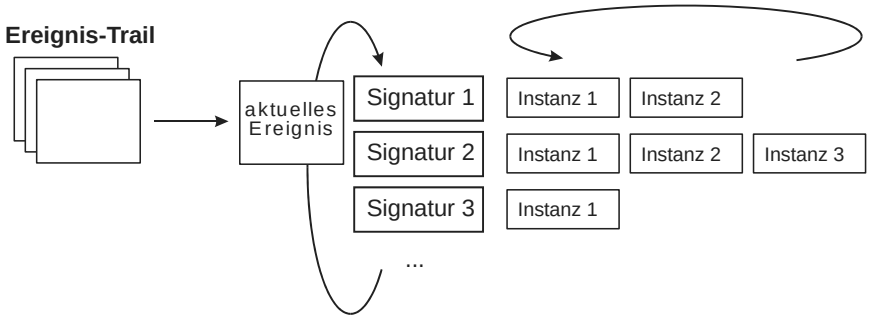


Abb. 8-2. Signaturanalyse durch einzelne Programm-Module

```
extern class InstanceStore gStore; // global instance store
class Signature1
{
private:
  int State; // State / Place of token
  int PPID;  // Token variables
  Signature1(int lState, int lPPID, ...) // Constructor
  {
    State = lState;
    PPID  = lPPID;
  }
  bool checkTransition1(Event* e)
  {
    return(
      (e->TYPE == 123)   && // Event type check
      (e->HOST == "ep")  && // Intra event condition check
      (true)                // Inter event condition check
    );
  }
  int executeTransition1(Event* e)
  {
    // Variable bindings
    PPID = e->pid;
    ...
    // create new instance / non-consuming
    gStore->registerInstance(new signature1(2, PPID, ...));
    // Execute actions
    ...
    return 1;
  }
  ...
public:
  Signature1(); // Constructor
  {
    State = 1; // Initial Place
    PPID  = -1;
  }
  int analyzeEvent(Event* e)
  {
    switch(State)
    {
    case 1:
      if(checkTransition1(e))
      {
          return executeTransition1(e);
      };
    ...
    }
    return 0;
  }
} // class Signature1
```

Abb. 8-3. Pseudo-C++-Quelltext eines Signatur-Moduls

8.1.2 Expertensystembasierte Analysen

Die Verwendung von Expertensystemen zur Missbrauchserkennung wurde bereits in Abschn. 7.2.2 diskutiert. Die eigentliche Analyse obliegt dem Regelinterpreter. Ein naiver Regelinterpreter iteriert über alle Regeln und Fakten, um anwendbare Regeln zu ermitteln. Die Optimierung dieser Verfahren stand bereits frühzeitig im Zentrum der Expertensystemforschung und Untersuchungen zeigten, dass 90 Prozent der Laufzeit eines Regelinterpreters zur Überprüfung der Bedingungen von Regeln benötigt wurden. Aus diesem Grunde konzentrieren sich Optimierungen auf diese Phase. Als Ergebnis der Forschung wurden optimierte Algorithmen [Mi87, Mi+90] entwickelt, von denen der Rete-Algorithmus [For82] der bekannteste und meist verwendete ist.

Die zur Realisierung von IDS eingesetzten Expertensysteme P-Best [Li+99] und CLIPS [Gi+94, Ri05] basieren auf dem Rete-Algorithmus, weshalb der Einsatz dieser Expertensysteme häufig auch mit der Analyseeffizienz begründet wird. Der Rete-Algorithmus und seine Optimierungen sowie ihre Wirkung bei der Verwendung des Verfahrens zur Missbrauchserkennung werden nachfolgend diskutiert.

Der Rete-Algorithmus

Dem Rete-Algorithmus liegen zwei Optimierungsideen zugrunde:

1. Vermeidung des Iterierens über die Regeln und
2. Vermeidung des Iterierens über die Fakten.

Die erste Idee geht von der Beobachtung aus, dass der größte Teil der Laufzeit von Expertensystemen zur Überprüfung der Regelbedingungen benötigt wird. Zur Optimierung werden die einzelnen Regelbedingungen in einem Datenflussgraphen, dem so genannten Rete-Netzwerk, organisiert. Dadurch wird erreicht, dass Bedingungen, die in mehreren Regeln auftreten, zusammengefasst und nicht mehrfach evaluiert werden. Diese Technik ist vergleichbar mit der aus der Compilertechnik bekannten *Common Subexpression Elimination* [Aho+88].

Die zweite Optimierungsidee basiert auf der Annahme, dass sich die Fakten in der Faktenbasis nur relativ selten ändern. Dies wird ausgenutzt, indem Fakten im Rete-Netzwerk an den Bedingungsknoten gespeichert werden, deren Bedingung sie erfüllen. Fortlaufend müssen dann nur die Fakten überprüft werden, die sich geändert haben.

Zur Verdeutlichung wird das Verfahren beispielhaft für ein typisches Regelsystem skizziert. Eine ausführliche Beschreibung erfolgt in [For82,

Gi+94]. Das diskutierte Beispiel verwendet die zwei Fakttypen *Ausdruck* und *Ziel*. Ein Fakt vom Typ *Ausdruck* beschreibt einen arithmetischen Ausdruck, der durch einen Namen, zwei Argumente und einen Operator gekennzeichnet ist. Fakten vom Typ *Ziel* beschreiben Transformationen wie bspw. *Auswertung* oder *Vereinfachung*, die auf Ausdrücke angewendet werden können. Sie sind durch den Typ der Transformation und den Namen eines Ausdrucks gekennzeichnet. Die Umsetzung der Transformationen wird durch Regeln angegeben. Abb. 8-4 stellt die Bedingungsteile von zwei Regeln zur Vereinfachung von Ausdrücken dar. Die im Arbeitsspeicher des Expertensystems vorhandenen Fakten sind in Tabelle 8-1 und Tabelle 8-2 beschrieben.

```
Regel 1
// Multiplikation mit 0
IF
 Ziel.Typ==Vereinfachung AND
 Ausdruck.Arg1==0 AND
 Ausdruck.Operator==* AND
 Ausdruck.Name==Ziel.Objekt
THEN
 ...
```

```
Regel 2
// Addition mit 0
IF
 Ziel.Typ==Vereinfachung AND
 Ausdruck.Arg1==0 AND
 Ausdruck.Operator==+ AND
 Ausdruck.Name==Ziel.Objekt
THEN
 ...
```

Abb. 8-4. Beispielregeln zur Vereinfachung von Ausdrücken

Tabelle 8-1. Beispielfakten vom Typ *Ausdruck*

Fakt-Nr.	Name	Arg1	Arg2	Op
1	a	0	11	+
2	b	0	12	+
3	c	2	13	+
4	d	0	14	*
5	e	0	15	*
6	f	5	16	*

Tabelle 8-2. Beispielfakten vom Typ *Ziel*

Fakt-Nr.	Typ	Objekt
7	Vereinfachung	a
8	Vereinfachung	b
9	Vereinfachung	c
10	Vereinfachung	d
11	Vereinfachung	e
12	Auswertung	f

Das den Regeln aus Abb. 8-4 entsprechende Rete-Netzwerk ist in Abb. 8-5 dargestellt. Die nummerierten Knoten des Netzwerks repräsentieren die Bedingungen der Regeln. Die obersten Knoten 1 und 2 prüfen den Typ der zu betrachtenden Fakten. Sie besitzen keine eingehenden Kanten und werden als *Wurzelknoten* bezeichnet. Die Knoten 3, 4, 5 und 6 repräsentieren die *Intra-Fakt-Bedingungen*, die sich auf einzelne Fakten beziehen. Diese Knoten werden als *Alpha-Knoten* bezeichnet und besitzen genau eine eingehende Kante. Die *Inter-Fakt-Bedingungen*, die Fakten der Typen *Ziel* und *Ausdruck* in Beziehung setzen, werden durch die *Beta-Knoten* 7 und 8 repräsentiert. Beta-Knoten besitzen genau zwei eingehende Kanten. Für jede eingehende Kante besitzt ein Beta-Knoten einen Speicher, also einen *linken* und einen *rechten Speicher*.

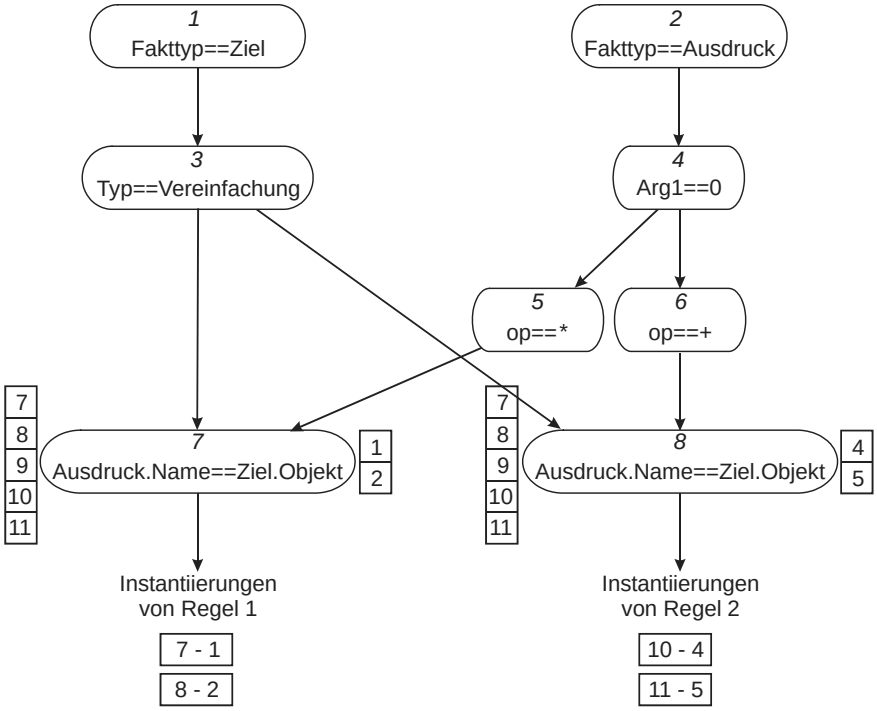


Abb. 8-5. Rete-Netzwerk für die Beispielregeln und -fakten

Beim Einfügen eines Fakts in die Faktenbasis wird dieser Fakt an die Wurzelknoten des Rete-Netzwerks übergeben. Fakten, die die mit dem Knoten assoziierte Bedingung erfüllen, werden an die Nachfolgerknoten weitergereicht. Ein Fakt, der einen Beta-Knoten erreicht, wird entsprechend der Eingangskante im linken (bzw. rechten) Speicher des Beta-

Knotens abgelegt. Außerdem wird überprüft, ob im rechten (bzw. linken) Speicher des Knotens Fakten existieren, so dass Faktenpaare die durch den Knoten repräsentierte Bedingung erfüllen.

In Abb. 8-5 ist dargestellt, welche der Fakten aus Tabelle 8-1 und Tabelle 8-2 in welchen Speichern der Beta-Knoten gehalten werden. Die Fakten mit den Nummern 7 und 1 bilden beispielsweise ein Paar, das der Bedingung von Knoten 7 genügt. Die Fakten dieser Paare werden zusammengefasst und an die Nachfolgerknoten weitergereicht. Die einzelnen Fakten verbleiben in den Speichern des Beta-Knotens. Dadurch wird festgehalten, dass diese Fakten allen Bedingungen der besuchten Vorgängerknoten genügen. Knoten, die keine Nachfolgerknoten besitzen, werden als *Terminalknoten* bezeichnet. Jeder Pfad von den Wurzelknoten zu einem Terminalknoten repräsentiert die Bedingungen einer Regel. Fakten, die einen Terminalknoten passieren, erfüllen alle Bedingungen der entsprechenden Regel und bilden eine Instantiierung der Regel. Beispielsweise bildet der aus den Fakten 1 und 7 zusammengesetzte Fakt eine Instantiierung der Regel 1.

Da Fakten, die den Bedingungen einer Regel genügen, in den Speichern der Beta-Knoten gespeichert werden, müssen diese Bedingungen für diese Fakten so lange nicht mehr überprüft werden, wie sich diese Fakten nicht ändern. Entsprechend müssen nur für neue bzw. veränderte Fakten Bedingungen überprüft werden. Die Änderung eines Fakts wird durch das Löschen des originalen Fakts und das Einfügen des geänderten Fakts realisiert.

Werden Fakten aus der Faktenbasis gelöscht, müssen sie auch aus dem Rete-Netzwerk gelöscht werden. Dazu muss ermittelt werden, in welchen Speichern sich der zu löschende Fakt befindet. Zu diesem Zweck wird der zu löschende Fakt erneut an die Wurzelknoten übergeben und die komplette Einfügeprozedur durchgeführt, wobei der Fakt aus den Speichern der Beta-Knoten entfernt anstatt eingefügt wird. Instantiierungen von Regeln, die zu löschende Fakten umfassen, werden entsprechend gelöscht.

Durch das Rete-Verfahren soll vermieden werden, dass in jedem Zyklus des Regelinterpreters alle Fakten geprüft werden. Zur weiteren Diskussion können hinsichtlich der Bedingungsprüfung für Fakten folgende Ansätze des Rete-Verfahrens festgehalten werden:

- **A1**: Intra-Fakt-Bedingungen werden nur einmal beim Einfügen eines Fakts in die Faktenbasis geprüft. Es wird gespeichert, welche Intra-Fakt-Bedingungen ein Fakt erfüllt.
- **A2**: Durch Beta-Knoten repräsentierte Inter-Fakt-Bedingungen werden nur einmal pro Fakten-Paar geprüft. Es wird gespeichert, welche Bedingungen ein Fakten-Paar erfüllt. Nur wenn ein neuer Fakt einen Beta-Knoten erreicht und im linken (bzw. rechten) Speicher abgelegt wird,

wird dieser Fakt hinsichtlich der Bedingung mit jedem Fakt im rechten (bzw. linken) Speicher des Knotens in Beziehung gesetzt.

Ausgehend von diesen Ansätzen untersucht der folgende Abschnitt die Wirkung der Optimierungen des Rete-Verfahrens bei der Verwendung zur Missbrauchserkennung.

Effizienz von Rete bei der Missbrauchserkennung

Zur Diskussion des Rete-Algorithmus bei Verwendung zur Missbrauchserkennung wird das Verfahren anhand des markierten Signaturnetzes in Abb. 8-6 und der Verarbeitung des dargestellten Ereignisses veranschaulicht. Entsprechend dem in Abschn. 7.2.2 dargestellten Verfahren wird dieses Signaturnetz auf Regeln und die Token sowie das Ereignis auf Fakten abgebildet. Abb. 8-7 skizziert die entsprechenden Regeln.

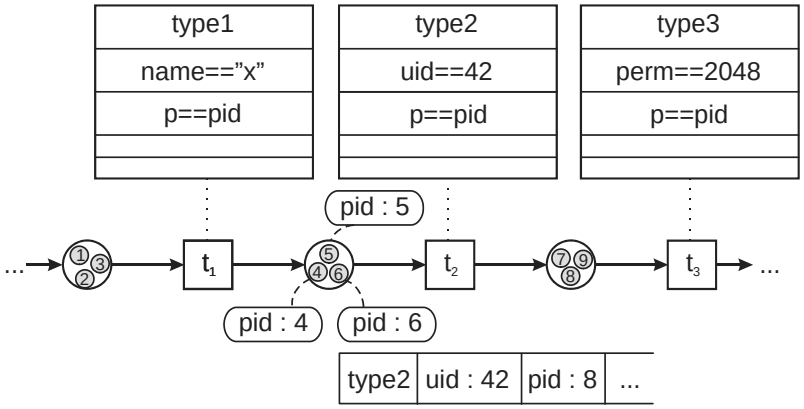


Abb. 8-6. Beispielsignaturnetz

```
Regel 1
IF
 Platz==p1 AND
 Typ==type1 AND
 name==”x” AND
 p==pid
THEN
  ...
```

```
Regel 2
IF
 Platz==p2 AND
 Typ==type2 AND
 uid==42 AND
 p==pid
THEN
  ...
```

```
Regel 3
IF
 Platz==p3 AND
 Typ==type3 AND
 perm==2048 AND
 p==pid
THEN
  ...
```

Abb. 8-7. Bedingungen der Transitionsregeln für das Beispielsignaturnetz

Abb. 8-8 stellt das diesen Regeln entsprechende Rete-Netzwerk dar, in dem auch die Token und das Ereignis aus Abb. 8-6 als Fakten wiedergegeben werden. Die Tokennummern wurden dabei als Faktnummern verwendet und der Ereignisfakt ist mit *e* benannt.

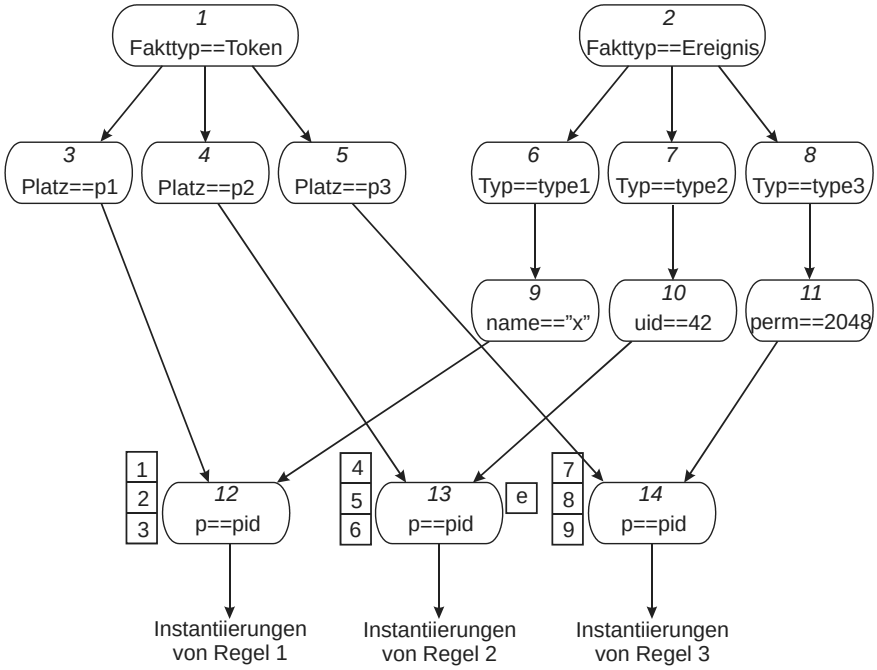


Abb. 8-8. Rete-Netzwerk für Beispielsignatur

Die Wurzelknoten (1 und 2) prüfen zunächst den Typ der zu verarbeitenden Fakten. Die Alpha-Knoten 3 bis 11 repräsentieren Intra-Fakt-Bedingungen. Jede Intra-Ereignis-Bedingung der betrachteten Transitionen des Signaturnetzes wird durch eine Intra-Fakt-Bedingung (die Alpha-Knoten 6 bis 11) repräsentiert. In Signaturnetzen existieren keine *Intra-Token-Bedingungen*, die Tokenvariablen mit Konstanten vergleichen[1]. Dementsprechend ist die Überprüfung des Platzes auf dem sich Token befinden die einzige Intra-Fakt-Bedingung für Tokenfakten (Alpha-Knoten 3 bis 5). Für jeden Platz im betrachteten Signaturnetz existiert ein entsprechender Alpha-Knoten. Weiter gilt für alle vom Token-Wurzelknoten

[1] Derartige Vergleiche würden durch Intra-Ereignis-Bedingungen an der Transition umgesetzt, an der die entsprechende Tokenvariable belegt wird.

(Knoten 1) ausgehenden Pfade im Rete-Netzwerk, dass sie genau einen Alpha-Knoten[1] enthalten.

Beta-Knoten im Rete-Netzwerk repräsentieren die Inter-Ereignis-Bedingungen der Transitionen des Signaturnetzes, also Vergleiche zwischen Variablen von Token auf verschiedenen Plätzen oder zwischen Ereignismerkmalen und Tokenvariablen. Entsprechend können im Rete-Netzwerk Beta-Knoten auftreten, die verschiedene Tokenfakten in Beziehung setzen (*TT-Beta-Knoten*, im Beispiel nicht enthalten) oder Vergleiche des Ereignisfakts mit Tokenfakten beschreiben (*ET-Beta-Knoten*, Knoten 12 bis 14). Durch die Knoten 12 bis 14 wird überprüft, ob die Tokenvariable *p* mit dem Ereignismerkmal *pid* übereinstimmt.

Da die betrachteten Regeln Transitionen beschreiben, die mit Ereignissen assoziiert sind[2], gilt, dass jede Regel mindestens eine Bedingung enthält, die Bezug auf den Ereignisfakt nimmt. Weiter gilt, dass jede Transition[3] eine Interereignisbedingung enthält, die überprüft, ob Signaturinstanzen (Token) existieren, die durch das Transitionsereignis fortschreiten. Derartige Bedingungen setzen ein instanzspezifisches Merkmal des Ereignisses (im Beispiel *pid*) mit einer entsprechenden Tokenvariable (im Beispiel *p*) in Beziehung. Dementsprechend enthält jeder Pfad von einem Wurzelknoten zu einem Terminalknoten mindestens einen ET-Beta-Knoten, der Tokenfakten mit dem Ereignisfakt in Beziehung setzt.

Außerdem gilt, dass sich der Ereignisfakt kontinuierlich ändert. Die Situation, dass der Ereignisfakt mehrere Zyklen des Regelinterpreters unverändert bleibt sich jedoch Tokenfakten ändern und daraus neue Regelinstantiierungen entstehen, tritt nicht auf. Diese Situation entspricht gerade dem Problemfall kaskadierter Regelausführung (vgl. Abschnitte 6.2.5 und 7.2.3), der ausgeschlossen werden muss. Dementsprechend wird ein Ereignisfakt durch die Bedingungen von ET-Beta-Knoten maximal einmal geprüft und danach durch einen neuen Fakt ausgetauscht.

Die vom Rete-Verfahren realisierten Ansätze zeigen damit folgende Auswirkungen:

- **A1**: Intra-Fakt-Bedingungen werden nur einmal bei Einfügen geprüft.
  - Der Ereignisfakt ändert sich in jedem Zyklus und wird grundsätzlich nur einmal überprüft und danach durch einen neuen ausgetauscht. Das Speichern des Ereignisfakts an Bedingungsknoten, um das Erfülltsein der Intra-Ereignisfakt-Bedingungen festzuhalten, ist hier

[1] Wurzelknoten werden nicht als Alpha-Knoten betrachtet.
[2] Spontane Transitionen (vgl. Abschn. 6.2.2) werden hier vernachlässigt.
[3] Eine Ausnahme sind initiale Transitionen, die hier vernachlässigt werden.

kontraproduktiv, da dadurch das Löschen des Fakts aus dem Netzwerk erforderlich wird.
  - Tokenfakten können über mehrere Zyklen unverändert bleiben. Die einzigen Intra-Fakt-Bedingungen für Tokenfakten sind die Überprüfungen der Plätze auf denen sich Token befinden. Die Anzahl der für einen Tokenfakten zu überprüfenden Platzbedingungen hängt von der Zahl der Plätze ab. Hier stellt das Speichern der Tokenfakten an den Bedingungsknoten eine Optimierung dar.
- **A2**: Inter-Fakt-Bedingungen von Beta-Knoten werden nur einmal pro Faktenpaar überprüft.
  - Für Token-Token-Bedingungen stellt dies eine Optimierung dar, da Tokenfakten und entsprechend Tokenfakten-Paare über mehrere Zyklen unverändert bleiben können.
  - Für Ereignis-Token-Bedingungen stellt dies keine Optimierung dar, da durch die kontinuierliche Veränderung des Ereignisfakts, ein Paar aus Token- und Ereignisfakt nur einen Zyklus unverändert bleibt. Das Speichern entsprechender Fakten an Beta-Knoten bzw. das Speichern entsprechender Regelinstantiierungen ist kontraproduktiv, da hierdurch ein Löschen der Einträge aus dem Rete-Netzwerk erforderlich wird.

Zusammenfassend kann festgehalten werden, dass unter den Besonderheiten der Missbrauchserkennung die Optimierungen des Rete-Verfahrens nur zum Teil wirken und teilweise kontraproduktiv sind. Dies wirft die Frage auf, ob der Nutzen des Verfahrens seine Kosten, z. B. die erforderlichen Löschoperationen, übersteigt. Bisher sind jedoch weder analytische noch experimentelle Untersuchungen bekannt, die sich dieser Frage widmen.

Durch Modifikationen des Rete-Verfahrens, die eine spezielle Behandlung des Ereignisfakts umsetzten, sind Effizienzverbesserungen zu erwarten. Allerdings laufen diese anwendungsspezifischen Anpassungen dem Grundgedanken von Expertensystemen zu wider, allgemeine Problemlösungsstrategien zu verwenden (vgl. Abschn. 7.2). Des Weiteren geht dadurch einer der Hauptvorteile der Verwendung von Expertensystemen, die Verwendung vorhandener Analyseverfahren, verloren.

Diese Situation war Motivation dafür, über dedizierte Optimierungen für die Signaturanalyse nachzudenken und entsprechende Verfahren zu entwickeln. Durch experimentelle Untersuchungen soll die Effizienz der Verfahren bewertet und verglichen werden. Die Ergebnisse dieser Arbeiten werden in den folgenden Abschnitten betrachtet.

8.2 Optimierungsstrategien

Der Effizienzoptimierung von Signaturanalyseverfahren wurde trotz der zunehmenden Bedeutung der Thematik bisher kaum Aufmerksamkeit durch die Forschung gewidmet (vgl. Abschnitte 3.2 und 8.1). Um diese Lücke zu schließen werden in diesem Abschnitt spezifische Merkmale der Signaturanalyse auf der Grundlage der Modellierung von Signaturen mit Signaturnetzen (vgl. Kapitel 6) betrachtet. Es wird ein Analyseverfahren diskutiert, das die Schaltregel für Signaturnetze umsetzt. Davon ausgehend werden verschiedene Optimierungsstrategien für dieses Verfahren vorgeschlagen und durch Simulation der Signaturnetze veranschaulicht.

Ein naives Analyseverfahren zur Umsetzung der Schaltregel von Signaturnetzen prüft für jedes eingehende Ereignis *X* alle Transitionen aller Signaturnetze. Beim Transitionstest wird geprüft, ob der Typ des Ereignisses *X* dem assoziierten Typ der Transition entspricht. Anschließend werden für jede Tokenkombination auf den Vorplätzen die Bedingungen der Transition im Zusammenhang mit dem Ereignis *X* evaluiert.

Typischerweise nimmt die Anzahl der Token in einem Signaturnetz bis zu einer bestimmten Größenordnung zu[1]. Für Signaturen, die beispielsweise Aktionen von bestimmten Prozessen verfolgen, existiert für jeden laufenden Prozess ein Token im entsprechenden Signaturnetz. Daraus ergibt sich eine entsprechende Anzahl von Tokenkombinationen, die im laufenden Betrieb zu testen sind. Für das naive Analyseverfahren erhöht sich der Aufwand zur Analyse eines einzelnen Ereignisses mit der Zahl vorhandener Token.

Zur Optimierung des naiven Verfahrens werden im Folgenden verschiedene kombinierbare Optimierungsstrategien vorgestellt. Sie nutzen strukturelle Eigenschaften und Redundanzen der Signaturen sowie Charakteristika der Analyse aus, um die Zahl zu prüfender Transitionen zu verringern, effizient die relevanten Transitionen zu selektieren und den Aufwand zur Überprüfung von Transitionsbedingungen zu reduzieren. Eine vergleichende Diskussion der Verfahren schließt sich an.

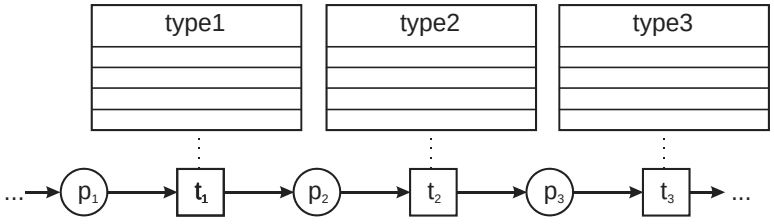
8.2.1 Strategie 1: Ereignistypbasierte Transitionsindizierung

Beim Auftreten eines Ereignisses prüft das naive Analyseverfahren für alle in Signaturnetzen spezifizierten Transitionen, ob das aufgetretene Ereignis

[1] Die durchschnittliche/maximale Anzahl von Signaturinstanzen richtet sich nach den Durchschnittswerten/Maxima der instanzidentifizierenden Merkmale, z.B. Prozess-IDs, Dateipfade, Benutzerkennungen.

dem Ereignistyp der Transition entspricht. Diese Überprüfungen können vermieden werden, indem eine Ereignis-Subskriptions-Tabelle verwendet wird. Durch diese Tabelle wird jedem Ereignistyp die Menge der Transitionen zugeordnet, die mit diesem Ereignistyp assoziiert sind. Statt einer Überprüfung aller Transitionen kann so durch einen Tabellenzugriff die Menge der Transitionen ermittelt werden, für die das aktuelle Ereignis relevant ist. Die Tabellenzugriffe können effizient durch Hash-Verfahren realisiert werden.

Zur Verdeutlichung des Verfahrens sind in Abb. 8-9 ein Signaturnetz sowie die entsprechende Ereignis-Subskriptions-Tabelle dargestellt. Bei Auftreten des dargestellten Ereignisses vom Typ *type2*, kann durch einen Tabellenzugriff anhand des Ereignistyps die Transition t_2 selektiert werden, für die das Ereignis relevant ist.



type2	uid : 42	pid : 8	...

Ereignis-Subskriptions-Tabelle	
Ereignistyp	Transitionen
type1	t_1
type2	**t_2**
type3	t_3

Abb. 8-9. Ereignis-Subskriptions-Tabelle

Die Erstellung der Ereignis-Subskriptions-Tabelle erfordert nur statische Informationen über die Transitionen in den Signaturnetzen. Diese können einmalig vor dem Beginn der Analyse erhoben werden. Eine Umsetzung dieser Strategie erfordert zur Laufzeit keinen weiteren Zeitaufwand.

8.2.2 Strategie 2: Tokenunabhängige Prüfung von Intra-Ereignis-Bedingungen

Damit eine Transition schalten kann, müssen alle der Transition zugeordneten Bedingungen erfüllt sein (s. Abb. 8-10). Diese Bedingungen werden in Intra- und Inter-Ereignis-Bedingungen unterschieden (vgl. Abschn.

6.2.2). Während Inter-Ereignis-Bedingungen entweder Merkmale des Ereignisses mit Variablen von Token in Beziehung setzen (zweite Inter-Ereignis-Bedingung in Abb. 8-10) oder Variablen von Token verschiedener Vorplätze vergleichen (erste Inter-Ereignis-Bedingung in Abb. 8-10), beziehen sich Intra-Ereignis-Bedingungen lediglich auf die Merkmalsbelegungen des auslösenden Ereignisses. Dementsprechend können Intra-Ereignis-Bedingungen unabhängig von den Token auf den Vorplätzen der Transition überprüft werden. Deshalb genügt es, Intra-Ereignis-Bedingungen für das aktuelle Ereignis einmal zu überprüfen. Nur wenn diese erfüllt sind, werden die Inter-Ereignis-Bedingungen für alle Kombinationen von Token auf den Vorplätzen ausgewertet.

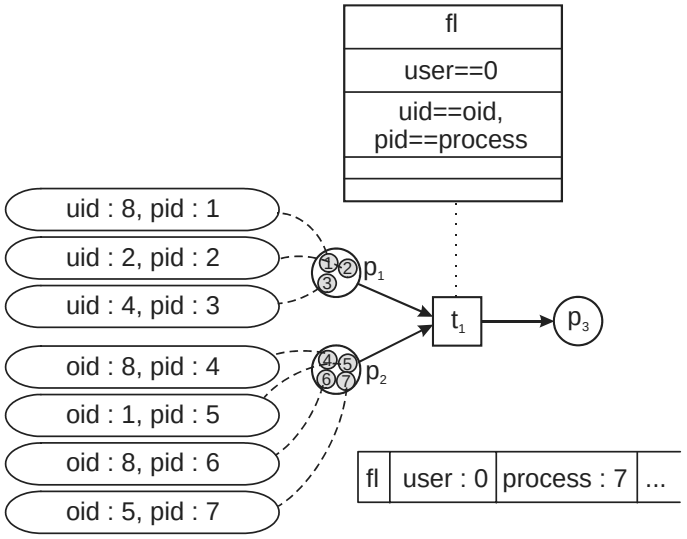


Abb. 8-10. Tokenunabhängige Prüfung der Intra-Ereignis-Bedingungen

Für die in Abb. 8-10 dargestellte Transition genügt es dementsprechend, die Intra-Ereignis-Bedingung *user==0* einmal für das aktuelle Ereignis zu überprüfen. Nur wenn diese Bedingung erfüllt ist, wird eine Überprüfung der Inter-Ereingis-Bedingungen für alle zwölf Tokenkombinationen mit dem aktuellen Ereignis erforderlich.

8.2.3 Strategie 3: Wertebasierte Tokenindizierung

Das naive Analyseverfahren prüft für Transitionen mit dem Ereignistyp des aktuellen Ereignisses und erfüllter Intra-Ereignis-Bedingung alle Kombinationen von Token auf den Vorplätzen, um die Kombinationen zu er-

mitteln, die alle Inter-Ereignis-Bedingungen erfüllen. Durch die Verwendung von Wertetabellen können Vergleichsoperationen der Inter-Ereignis-Bedingungen einer Transition dazu verwendet werden, die Anzahl der zu testenden Tokenkombinationen zu reduzieren.

Die Inter-Ereignis-Bedingungen einer Transition referenzieren Variablen von Token auf Vorplätzen der Transition. Für jeden Vorplatz wird ermittelt, welche Tokenvariablen referenziert werden. Für jede der Variablen wird eine separate Wertetabelle verwaltet, die möglichen Werten der Tokenvariable die Token zuordnet, die diese Variable mit dem entsprechenden Wert belegen. Durch Verwendung dieser Wertetabellen kann durch einen Tabellenzugriff ermittelt werden, welche Token auf einem Platz eine Variable mit einem bestimmten Wert belegen.

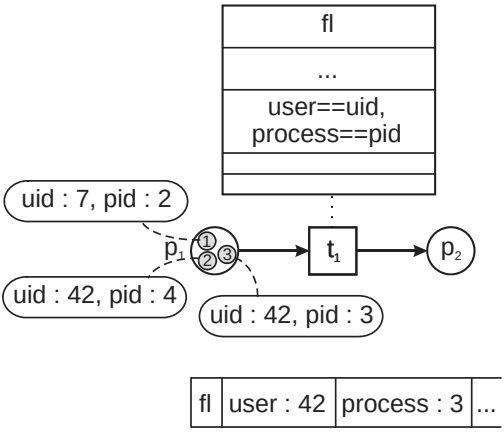
Zur Verdeutlichung dieser Strategie sind in Abb. 8-11 ein mit Token markiertes Beispielsignaturnetz sowie die entsprechenden Wertetabellen dargestellt. Die erste Bedingung im Beispiel fordert die Gleichheit der Wertebelegung vom Merkmal *user* des auslösenden Ereignisses und der Variablen *uid* eines Tokens auf einem Vorplatz p_1. Durch einen einzelnen Zugriff mit dem Wert von *user* (*42*) in die Wertetabelle für die Variable *uid* des Platzes p_1 kann die Menge $X=\{2, 3\}$ der Token auf p_1 selektiert werden, die diese Bedingung erfüllen. Für die zweite Gleichheitsbedingung zwischen dem auslösenden Ereignis und den Merkmalen auf Platz p_1 wird analog durch einen Zugriff eine zweite Tokenmenge $Y=\{3\}$ selektiert. Der Schnitt der Mengen X und Y enthält die Token von Platz p_1, die beide Bedingungen erfüllen. Entsprechend ist Transition t_1 für Token *3* mit dem aktuellen Ereignis schaltbereit. Durch dieses Vorgehen kann anhand von Gleichheitsbedingungen effizient die Menge der Token ermittelt werden, für die Transitionen schaltbereit sind.

Dieses Vorgehen kann auch bei Vergleichsbedingungen zwischen Tokenvariablen verschiedener Vorplätze einer Transition realisiert werden. In dem Beispiel in Abb. 8-12 wird gefordert, dass die Variable *uid* von Token auf dem Platz p_1 und die Variable *oid* von Token auf p_2 die gleichen Werte besitzen. Außerdem sollen die Werte der Variable *source* von Token auf Platz p_1 und der Variable *host* von Token auf p_2 übereinstimmen.

Abb. 8-12 stellt auch die Variablenbelegung der Token und die entsprechenden Wertetabellen dar. Für die erste Bedingung wird mit den Werten aus der Tabelle $p_1.uid$ auf die Tabelle $p_2.oid$ zugegriffen[1]. Für jede erfolg-

[1] Prinzipiell könnte auch umgekehrt mit den Werten aus der Tabelle $p_2.oid$ auf die Tabelle $p_1.uid$ zugegriffen werden. Um die Anzahl der Zugriffe zu minimieren, wird immer mit den Werten aus der kleineren Wertetabelle auf die größere Wertetabelle zugegriffen, wobei die Größe einer Wertetabelle durch die Anzahl ihrer Zeilen bestimmt ist.

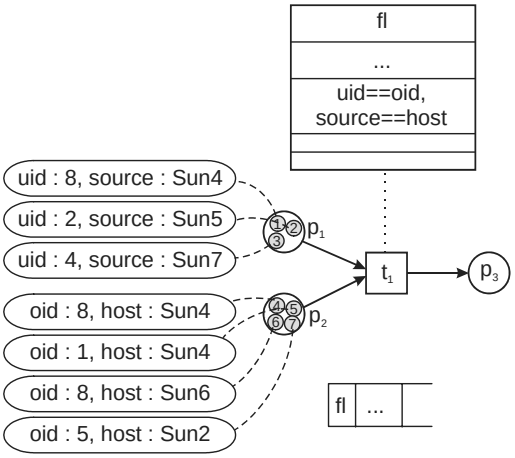
reiche Übereinstimmung werden platzweise Tokenmengen selektiert. Für Platz p_1 ergeben sich im Beispiel die Mengen *{1}* und *{3}* und für Platz p_2 die Menge *{4, 6}* und *{7}*.



p_1.uid	
Wert	Token-Nr.
7	1
42	2, 3

p_1.pid	
Wert	Token-Nr.
2	1
3	3
4	2

Abb. 8-11. Wertetabellen für Ereignis-Token-Vergleiche



p_1.uid	
Wert	Token-Nr.
8	1
2	2
4	3

p_2.oid	
Wert	Token-Nr.
8	4, 6
1	5
4	7

p_1.source	
Wert	Token-Nr.
Sun4	1
Sun5	2
Sun7	3

p_2.host	
Wert	Token-Nr.
Sun4	4, 5
Sun6	6
Sun2	7

Abb. 8-12. Wertetabellen für Token-Token-Vergleiche

Aus der platzweisen Vereinigung dieser Mengen resultiert die Menge der Token, die die erste Bedingung erfüllen. Für Platz p_1 ergibt sich die Menge p_{1a}*={1, 3}* und für p_2 die Menge p_{2a}*={4, 6, 7}*. Für die zweite Bedingung wird analog vorgegangen. Daraus resultiert für Platz p_1 die Menge p_{1b}*={1}* und für p_2 die Menge p_{2b}*={4, 5}*.

Der platzweise Schnitt, der aus den Bedingungen resultierenden Tokenmengen, liefert die Token, die beide Bedingungen erfüllen: $p_{1a} \cap p_{1b} = \{1\}$ und $p_{2a} \cap p_{2b} = \{4\}$, d. h., Transition t_1 ist mit Token *1* auf p_1 und Token *4* auf p_2 schaltbereit.

Das beschriebene Vorgehen kann nicht nur für Gleichheitsbedingungen angewendet werden, sondern ist für andere Vergleichsoperationen wie kleiner oder größer realisierbar. Dies erfordert eine sortierte Ablage der Schlüsselwerte in den Wertetabellen und effiziente Mechanismen zur Selektion von Wertebereichen in den Tabellen, z. B. allen Werten kleiner einem gegebenen Wert.

Durch die beschriebene Strategie wird das Iterieren über alle möglichen Tokenkombinationen vermieden. Stattdessen wird ausgehend von spezifizierten Vergleichsbedingungen effizient die Menge passender Tokenkombinationen ermittelt, die bei weiteren Bedingungsprüfungen zu berücksichtigen sind. Zur Anwendung der vorgeschlagenen Strategie ist die Pflege von Wertetabellen erforderlich. Die Anzahl und der Aufbau der Tabellen hängen allein von der statischen Struktur der betrachteten Signaturen ab. Die Einträge der Tabellen sind jedoch dynamisch bei Tokenfluss anzupassen. Des Weiteren sind verschiedene Mengenoperationen erforderlich, für die jedoch effiziente Realisierungen bekannt und verfügbar sind (vgl. [Jo99]).

8.2.4 Strategie 4: Gemeinsame Ausdrücke

Bei der Analyse eines eintretenden Ereignisses werden an den Transitionen spezifizierte Bedingungen auf Erfüllung getestet. Unterschiedliche Transitionen können in ihrem Bedingungsblock gleiche Ausdrücke bzw. Teilausdrücke enthalten. Beispielsweise suchen viele Signaturen nach Aktionen von privilegierten Benutzern (root) oder Zugriffen auf sicherheitskritische Systemressourcen (z. B. die Passwortdatei). Dadurch sind entsprechende Bedingungen in mehreren Signaturen vorzufinden. Um eine mehrfache Analyse dieser Ausdrücke mit den gleichen Parametern zu vermeiden, werden die Bedingungen aller Transitionen einmalig vor dem Beginn der Analyse auf gemeinsame Teilausdrücke untersucht.

Es werden gemeinsame Ausdrücke in den Intra-Ereignis-Bedingungen aller Transitionen identifiziert. Gemeinsame Ausdrücke in Inter-Ereignis-Bedingungen werden hingegen pro Transition ermittelt. Ergebniswerte der gemeinsamen Ausdrücke werden gespeichert und bei einer erneuten Analyse des gleichen Ausdrucks mit den gleichen Parametern wieder verwendet. Bei Intra-Ereignis-Bedingungen sind die gespeicherten Werte für den Zeitraum der Abarbeitung eines Ereignisses gültig, da erst durch ein neues

Ereignis eine Veränderung der Parameter eintritt. Dagegen ist der Gültigkeitszeitraum bei Inter-Ereignis-Bedingungen auf eine Tokenkombination beschränkt, da andere Tokenkombinationen andere Wertebelegungen repräsentieren.

Zur Identifizierung von gleichen Ausdrücken innerhalb der Bedingungen müssen die Ausdrücke ggf. semantisch äquivalent umgeformt werden. Dies ist nötig, da die gemeinsamen Ausdrücke in Bedingungen vom Modellierer bzw. Signaturentwickler meist nur inhaltlich, aber nicht syntaktisch identisch verwendet werden. Standardtechniken zur Identifizierung von gemeinsamen Ausdrücken werden in [Aho+88] diskutiert. Bei einer Umsetzung dieser Strategie wird durch die Verwaltung der Ergebnisse zur Laufzeit zusätzlicher Aufwand erforderlich.

8.2.5 Strategie 5: Kostenbasierte Bedingungspriorisierung

Da von der Vielzahl der zu analysierenden Ereignisse typischerweise nur ein geringer Teil eine Transition auslöst, werden die Transitionsbedingungen hauptsächlich negativ ausgewertet. Dementsprechend sollte die Reihenfolge der Bedingungsüberprüfung auf Misserfolg optimiert erfolgen. Nachdem eine Teilbedingung als nicht zutreffend evaluiert wurde, ist die Auswertung weiterer Teilbedingungen überflüssig. Demnach ist es von Vorteil, Bedingungen die häufig nicht zutreffen, zuerst zu überprüfen. Zur weiteren Diskussion wird für diesen Zusammenhang der Begriff der *Selektivität* von Bedingungen verwendet, wobei Bedingungen, die selten erfüllt sind, eine hohe Selektivität aufweisen.

Des Weiteren sind zur Auswertung verschiedener Ausdrücke unterschiedliche Zeitaufwände erforderlich. Beispielsweise können Zahlenvergleiche schneller ausgewertet werden als Zeichenkettenvergleiche. Teilbedingungen mit geringen Laufzeiten sollten zuerst analysiert werden, da dadurch ggf. die Auswertung der aufwendigeren Ausdrücke vermieden werden kann.

Die entsprechende Priorisierung bzw. Reihung der Bedingungen kann sowohl statisch als auch dynamisch erfolgen. Bei der statischen Priorisierung werden nur die Zeitaufwände zur Bedingungsauswertung berücksichtigt und die Bedingungen auf der Basis von Laufzeitabschätzungen der verwendeten Operationen kategorisiert. So wird bspw. zwischen Zahlen- und Zeichenkettenvergleichen unterschieden.

Die dynamische Bedingungspriorisierung erfolgt auf der Grundlage der Selektivität sowie der Zeitaufwände zur Evaluierung von Bedingungen. Sie berücksichtigt außerdem, dass die zu analysierenden Ereignisse von den Nutzeraktivitäten auf dem überwachten IT-System abhängen. Dabei

beeinflussen die Nutzeraktivitäten den Typ und die Wertebereiche der Merkmalsbelegungen der Ereignisse. Somit sind in Abhängigkeit von den aktuell vorherrschenden Aktivitäten bestimmte Bedingungen wahrscheinlicher erfüllt als andere. Weiterhin beeinflussen die Wertebelegungen der auftretenden Ereignisse die Evaluierungszeit von Ausdrücken (z. B. bei Zeichenkettenvergleichen). Um die Analyse diesen Umständen anzupassen, werden die realen Laufzeiten und als Maß für die Selektivität die Negativquote der Bedingungen regelmäßig erhoben. Der Quotient von Laufzeit und Negativquote einer Bedingung ergibt einen Wert, der angibt, wie effizient die Bedingung eine Ereignis- bzw. Tokenkombination verwirft und zur Priorisierung der Bedingungen verwendet wird. Durch die regelmäßige Aktualisierung des Maßes wird die Analyse an das aktuelle Systemverhalten angepasst. Für die in Tabelle 8-3 dargestellten Bedingungen wird dieses Vorgehen beispielhaft veranschaulicht. Die Tabelle stellt die kumulierten Laufzeiten sowie die Anzahl negativer Auswertungen der Bedingungen in einem Erhebungsintervall dar. Außerdem ist der Quotient der beiden Werte dargestellt. Entsprechend dieser Werte verwirft die Bedingung C_3 am effizientesten Ereignisse bzw. Tokenkombinationen.

Tabelle 8-3. Dynamische Priorisierung von Beispielbedingungen

Bedingung	Kumulierte Laufzeit in ms	Anzahl negativer Auswertungen[1]	Quotient
C_1	5	1	5
C_2	10	5	2
C_3	5	5	1
C_4	10	1	10

Die statische Variante der Bedingungspriorisierung kommt ohne zusätzliche Laufzeitaufwände aus. Hingegen beansprucht die dynamische Variante Laufzeit zur Erhebung der Messwerte sowie zur Anpassung der Auswertungsreihenfolge der Bedingungen.

8.2.6 Diskussion

Bevor die vorgestellten Optimierungsstrategien in den nächsten Abschnitten experimentell evaluiert und mit anderen Analyseansätzen verglichen werden, erfolgt in diesem Abschnitt ein zusammenfassender Vergleich der

[1] Für Bedingungen, die innerhalb des Erhebungsintervalls nicht negativ ausgewertet wurden, wird der Quotient auf einen großen festen Wert gesetzt, um der Bedingung geringe Priorität zu geben.

Strategien mit den von Rete realisierten Optimierungen. Erwartete Effizienzverbesserungen werden abgeleitet.

Mittels Strategie 1 (ereignistypbasierte Transitionsindizierung) werden effizient durch einen Tabellenzugriff die Transitionen selektiert, für die das aktuelle Ereignis relevant ist. Im Unterschied dazu wird der Ereignisfakt beim Rete-Verfahren an alle Alpha-Knoten weitergereicht, die einen Typvergleich für den Ereignisfakt beschreiben. Diese prüfen den Ereignisfakt entsprechend, um die Pfade im Rete-Netzwerk zu ermitteln, die Regeln für Transitionen mit diesem Ereignistyp beschreiben. Im Beispiel-Rete-Netzwerk in Abb. 8-8 in Abschn. 8.1.2 wird der Ereignisfakt jeweils an die Alpha-Knoten 6 bis 8 weitergereicht und durch diese ausgewertet.

Durch Strategie 2 (tokenunabhängige Prüfung von Intra-Ereignis-Bedingungen) wird erreicht, dass unabhängig von der Anzahl Token auf den Vorplätzen einer Transition, die Intra-Ereignis-Bedingungen der Transitionen für ein Ereignis nur einmal geprüft werden. Den gleichen Effekt erreicht auch das Rete-Verfahren.

Die von Strategie 3 realisierte wertebasierte Tokenindizierung führt dazu, dass mittels eines Tabellenzugriffs effizient die Token selektiert werden, die einer Inter-Ereignis-Bedingung genügen. Beim Rete-Verfahren entspricht dies bspw. der Situation, dass zu einem Ereignisfakt im rechten Speicher eines Beta-Knotens die Tokenfakten im linken Speicher des Beta-Knotens ermittelt werden, deren Variablen mit einem Merkmal des Ereignisses übereinstimmen. Knoten 13 in Abb. 8-8 ist ein Beispiel für diese Situation. Hier prüft das Rete-Verfahren für jedes Paar aus Token- und Ereignisfakten aus den Speichern des Beta-Knotens, ob die Bedingung erfüllt ist. Wie effizient diese Operation durchgeführt wird, hängt von der konkreten Implementierung des Rete-Verfahrens ab. Prinzipiell kann diese ebenfalls effizient mittels der von Strategie 3 verwendeten Tabellentechnik realisiert werden.

Die von Strategie 4 umgesetzte Zusammenfassung gemeinsamer Ausdrücke wird beim Rete-Verfahren ebenfalls umgesetzt. Dazu werden Knoten, die identische Bedingungen beschreiben, im Rete-Netzwerk zusammengefasst.

Eine Priorisierung einzelner Bedingungen, wie sie Strategie 5 umsetzt, wird beim Rete-Verfahren nicht umgesetzt. Die Reihenfolge der Alpha- bzw. Beta-Knoten im Rete-Netzwerk richtet sich nach der Anordnung der Bedingungen in den Regeln.

Von einem Analysewerkzeug, dass die vorgeschlagenen Optimierungsstrategien kombiniert wird eine höhere Effizienz als vom Rete-Verfahren erwartet, da zusätzlich zu den Optimierungen des Rete-Verfahrens Verbesserungen durch ereignistypspezifische Transitionsselektion und Priorisierung der Bedingungen erreicht werden. Darüber hinaus kann auf die über-

flüssige Zwischenspeicherung und dadurch notwendige Löschung von Ereignisfakten, wie sie beim Rete-Verfahren auftritt, verzichtet werden.

8.3 Das Analysewerkzeug SAM

Um eine Bewertung der im vorigen Abschnitt vorgestellten Optimierungsstrategien vornehmen zu können, wurde das Werkzeug SAM (Signature Analysis Module) entwickelt [Mei+05a, Mei+05b], das die vorgestellten Verfahren umsetzt. Es dient der Analyse von EDL-Signaturen und bietet die Grundlage für experimentelle Leistungsvergleiche mit anderen Analyseverfahren. Um die verschiedenen Optimierungsstrategien untersuchen und ihre Effekte vergleichen zu können, wurden verschiedene Ausbaustufen von SAM realisiert, die unterschiedliche Kombinationen der Optimierungsstrategien umsetzen. Dieser Abschnitt gibt einen Überblick über die Implementierung und führt die verschiedenen Ausbaustufen ein. Eine ausführliche Beschreibung der Implementierung erfolgt in [Schme04].

SAM wurde unter Verwendung der objektorientierten Programmiersprache C++ [Stro97, ISO98] implementiert. Es besteht aus folgenden Komponenten (s. Abb. 8-13):

- Parser-Einheit
- Signaturspeicher
- Eingabeeinheit
- Transformationseinheit
- Funktionsmanager.

Die *Parsereinheit* ist für das Einlesen von EDL-Beschreibungen zuständig. Sie wurde unter Verwendung der Werkzeuge Flex [FSF99] und Bison [FSF05] implementiert. Beim Einlesen werden die Beschreibungen auf syntaktische und semantische Korrektheit überprüft. Gleichzeitig werden die internen Darstellungen der EDL-Beschreibungen im Signaturspeicher generiert.

Der *Signaturspeicher* verwaltet die internen Darstellungen der Transitionen und Plätze der eingelesenen EDL-Beschreibungen. Außerdem ist er für die Speicherung der Token mit entsprechenden Variablenbelegungen verantwortlich. Des Weiteren enthält er Beschreibungen für Eingangsereignisse, die Finalplätzen mit entsprechenden Merkmalen entsprechen, auf die zu analysierende Ereignisse abgebildet werden. Zur Organisation der Daten werden die Datenstrukturen Hash-Map und Set der C++ Standard-Bibliothek [Jo99] verwendet.

Die *Eingabeeinheit* realisiert die Anbindung verschiedener Sensoren. Sie nimmt zu analysierende Eingangsereignisse entgegen und pflegt sie in den Signaturspeicher ein.

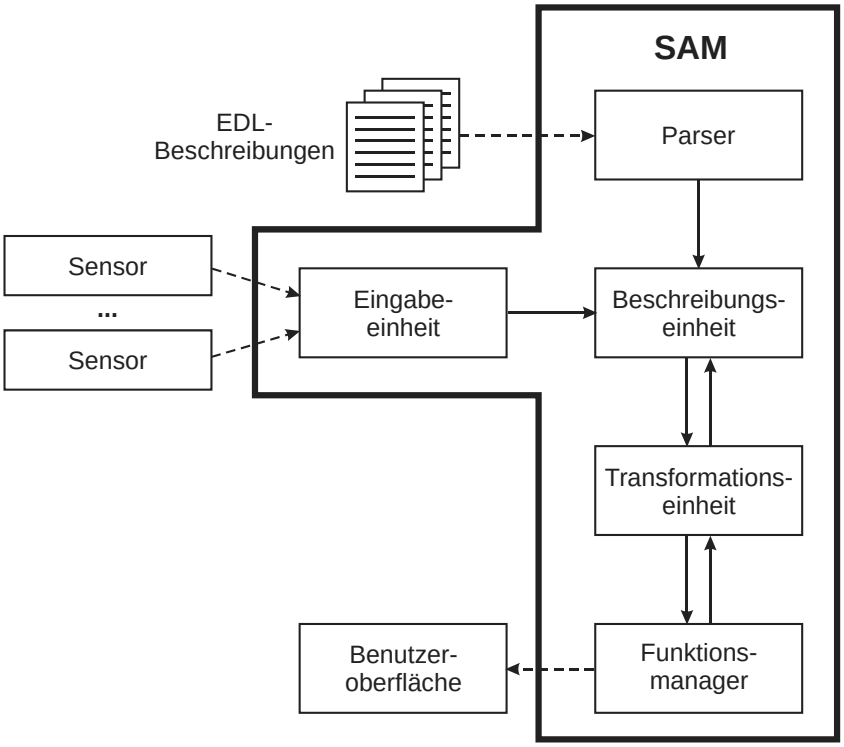


Abb. 8-13. Struktur von SAM

Funktionen zur Transformation interner Beschreibungen sowie Analysefunktionen sind in der *Transformationseinheit* realisiert. Durch Transformationen, wie das Auflösen von Makros oder das Identifizieren gemeinsamer Ausdrücke, werden die Beschreibungen im Signaturspeicher angepasst. Zu den Analysefunktionen zählen das Evaluieren von Bedingungen, die Belegung von Ereignismerkmalen sowie das Auslösen von Aktionen. Durch diese Funktionen werden die Tokenbelegungen von Plätzen und die Variablen von Token im Signaturspeicher verändert.

Der *Funktionsmanager* verwaltet eine erweiterbare Menge von Funktionen. Diese Funktionen realisieren Prädikate und Operationen, die in den Bedingungen der Transitionen verwendet werden. Außerdem sind hier Operationen realisiert, die bei der Belegung von Tokenvariablen und als Aktionen verwendet werden können.

Es wurden sechs verschiedene Ausbaustufen des SAM-Prototyps realisiert, die unterschiedliche Kombinationen der in Abschn. 8.2 vorgestellten Optimierungsstrategien umsetzen. In Tabelle 8-4 ist dargestellt, welche Ausbaustufe, welche Strategiekombination umsetzt. Die aufgeführten sechs Prototypen wurden zur experimentellen Evaluierung der Optimierungsstrategien verwendet, deren Ergebnisse im folgenden Abschnitt vorgestellt und diskutiert werden.

Tabelle 8-4. SAM-Ausbaustufen und umgesetzte Strategien

Ausbaustufe	Umgesetzte Strategie
SAM_1	1
SAM_2	1, 2
SAM_3	1, 2, 4
SAM_4	1, 2, 3, 4
SAM_5	1, 2, 3, 4, 5 (statisch)
SAM_6	1, 2, 3, 4, 5 (dynamisch)

8.4 Experimentelle Evaluierung

In der Literatur sind kaum experimentelle Untersuchungen der Effizienz von Signaturanalysesystemen beschrieben und vergleichende Betrachtungen sind nicht bekannt. Um entsprechende Einsichten gewinnen zu können, wurden verschiedene Experimente durchgeführt. Diese zielten zum einen darauf ab, die durch die vorgestellten Optimierungsstrategien erreichten Leistungsverbesserungen zu quantifizieren. Zum anderen dienten sie dazu, die entwickelten Verfahren mit dem Stand der Technik zu vergleichen, und die erreichten Effizienzsteigerungen aufzuzeigen.

In den Experimenten wurden die sechs Ausbaustufen von SAM sowie jeweils ein Vertreter für die beiden existierenden Analyseansätze (vgl. Abschn. 8.1) untersucht. Kriterien für die Auswahl der Systeme waren die ausreichende Dokumentation der Systeme sowie die Verfügbarkeit des Quelltextes, um erforderliche Anpassungen an die Testumgebung vornehmen zu können.

Das an der University of California at Santa Barbara entwickelte System STAT [Vi+00] realisiert den Analyseansatz, bei dem für jede Signatur ein separates Programmmodul erstellt wird. Für jede in der Beschreibungssprache STATL [Eck+00a, Eck+02] (vgl. auch Abschn. 7.1) vorliegende Signatur wird durch das System eine C++-Klasse erstellt, die die Analyse entsprechend der spezifizierten Kriterien durchführt. Eine ausführliche Beschreibung des Systems findet sich in [Vi+00]. STAT wurde für die expe-

rimentellen Untersuchungen herangezogen, da es der einzige im Quelltext verfügbare Vertreter dieses Analyseansatzes ist. Außerdem ist es eines der bekanntesten Systeme im akademischen Bereich und wurde in einer Reihe von Publikationen beschrieben. Eine Weiterentwicklung des Systems ist vorgesehen [Vi05].

Da kein offen verfügbarer Vertreter expertensystembasierter Signaturanalysesysteme (vgl. Abschnitte 7.2.2 und 8.1.2) bekannt ist, wurde mit CLIPS-IDS ein entsprechendes System entwickelt. Eine ausführliche Beschreibung dieses Prototyps findet sich in [Krau04]. Er wurde unter Verwendung des Rete-basierten Expertensystems CLIPS [Gi+94, Ri05] realisiert. CLIPS ist das bekannteste offen und frei verfügbare Expertensystem, das zudem kontinuierlich weiterentwickelt wird.

Im Folgenden wird zunächst das während der Experimente verwendete Testszenario beschrieben. Die verwendeten Signaturen und Protokolldaten werden dargestellt. Außerdem wird der Analyseablauf erläutert, der den folgenden Leistungsvergleichen zugrunde liegt. Eine Erläuterung der verwendeten Vorgehensweise zur Erhebung von Messwerten sowie der zugrunde liegenden Messumgebungen schließt sich an. Danach werden die ermittelten Messwerte präsentiert und diskutiert.

8.4.1 Testszenario

Bei den durchgeführten Tests wurden die drei Signaturen für die Link-Attacke, die Login-Attacke und die PATH-Attacke (vgl. Abschn. 4.2) verwendet und entsprechende Signaturbeschreibungen in den Sprachen STATL, EDL sowie der Regelsprache von CLIPS entwickelt.

Zur Vereinfachung der Beschreibung der Signaturen in STATL und der CLIPS-Regelsprache sowie des Vergleichs der Systeme wurde von einer homogenen Typisierung der Audit-Ereignisse ausgegangen. Das heißt sowohl alle Audit-Ereignisse als auch alle Transitionen der EDL-Signaturen sind mit dem gleichen Basisereignistyp SOLARIS_AUDIT_EVENT assoziiert.

In den Tests wurde der benötigte Analyseaufwand der Systeme in Lastsituationen bei steigender Zahl von Attacken- bzw. Signaturinstanzen untersucht. Dazu wurde eine Test-Audit-Trail erstellt, die Manifestierungen mehrerer Attackeninstanzen der drei betrachteten Attacken enthält, wobei die Anzahl laufender Attackeninstanzen kontinuierlich zunimmt. Zur Verdeutlichung dieses Szenarios werden die Zusammensetzung der Test-Audit-Trail sowie die Analyseabläufe während der Tests im Folgenden unter Verwendung von Signaturnetzen (vgl. Kapitel 6) detailliert diskutiert.

Die Test-Audit-Trail enthält Manifestierungen von zehn zeitlich überlappenden Instanzen der Link-Attacke, d.h., die ersten zehn Audit-Records

dokumentieren jeweils die erste Aktion einer Link-Attacke. Darauf folgen zehn Audit-Ereignisse, die jeweils die zweite Aktion der Link-Attacken dokumentieren, usw. Durch die ersten 30 Audit-Ereignisse werden zehn erfolgreiche Instanzen der Link-Attacke beschrieben. Abb. 8-14 veranschaulicht die Anordnung der Ereignisse der verschiedenen Instanzen der Link-Attacke in der Test-Audit-Trail.

Des Weiteren enthält die Test-Audit-Trail zehn (zeitlich) überlappende Instanzen der Login-Attacke, deren Manifestierung jeweils drei Audit-Ereignisse umfasst. Entsprechend dokumentieren die Ereignisse 31 bis 60 (vgl. Abb. 8-14) zehn erfolgreiche Login-Attacken.

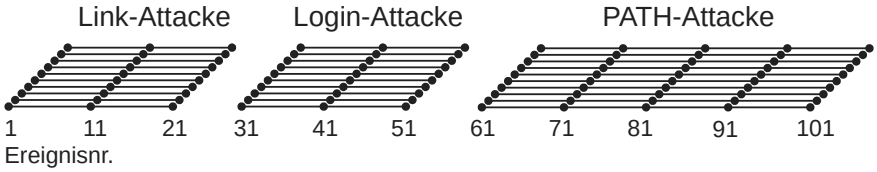


Abb. 8-14. Anordnung der Ereignisse in der Test-Audit-Trail

Die Ereignisse 61 bis 110 (vgl. Abb. 8-14) der Test-Audit-Trail beschreiben zehn sich zeitlich überlappende Instanzen der PATH-Attacke. Eine erfolgreiche Attacke wird durch drei Ereignisse dokumentiert. Zusätzlich enthält die Audit-Trail für jede Instanz zwei Ereignisse, die die Beendigung der in die Attacke involvierten Prozesse beschreiben und damit ein Terminierung der Attacken- bzw. Signaturinstanzen bewirken.

Die beschriebenen 110 Ereignisse der Test-Audit-Trail wiederholen sich 1.000 Mal in der Test-Audit-Trail. Dabei enthalten die wiederholten Ereignisse teilweise andere Parameter. Die daraus resultierenden Analyseabläufe und die Entwicklung der Zahl der Signaturinstanzen werden nachfolgend für die einzelnen Signaturen dargestellt.

Link-Attacke

Abb. 8-15 stellt ein Signaturnetz für die Link-Attacke dar. Die durch die Test-Audit-Trail ausgelösten Transitionsfolgen sind hervorgehoben. Tabelle 8-5 gibt an, welche Ereignisse welche Transition auslösen und wie viele Token sich danach auf den entsprechenden Plätzen befinden. Die Token auf Initial-, Abbruch- und Final-Plätzen werden nicht dargestellt. Interiorplätze, die während der Analyse leer bleiben, werden ebenfalls nicht dargestellt.

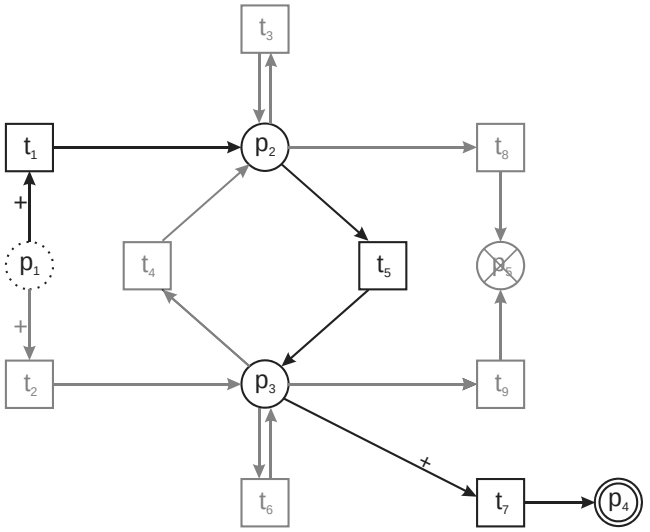


Abb. 8-15. Signaturnetz für die Link-Attacke (Skizze)

Die erste Spalte in Tabelle 8-5 gibt die Nummern der Ereignisse an, die in einer Zeile betrachtet werden. In der zweiten Spalte sind die Transitionen angegeben, die mit den Ereignissen schalten. Außerdem enthält die Tabelle eine Spalte für jeden der betrachteten Plätze. In diesen Spalten ist angegeben, wie viele Token sich auf dem entsprechenden Platz befinden, nachdem die Transitionen mit den betrachteten Ereignissen geschaltet haben. In der letzten Spalte der Tabelle ist dargestellt, wie viele offene Signaturinstanzen bzw. Token nach Abarbeitung der zehn Ereignisse vorliegen. Beispielsweise schaltet für die Ereignisse 1 bis 10 jeweils die Transition t_1. Danach befinden sich zehn Token auf Platz p_2 und Platz p_3 ist leer. Die Transition t_5 schaltet für die Ereignisse 11 bis 20, wonach sich auf Platz p_2 keine und auf Platz p_3 zehn Token befinden.

Entsprechend Tabelle 8-5 verbleiben zehn offene Signaturinstanzen (Token auf Platz p_3) nach Abarbeitung der ersten zehn Instanzen der Link-Attacke (Ereignisse 1-30). Die zweiten zehn Instanzen der Link-Attacke (Ereignisse 111-140) unterscheiden sich von den ersten zehn, indem andere Subjekte (Nutzer) und Objekte (Links) verwendet wurden. Dementsprechend entwickeln sich die verbliebenen zehn offenen Instanzen nicht weiter. Stattdessen wiederholen sich die Abläufe und es verbleiben zehn weitere offene Instanzen, so dass nach Abarbeitung der ersten 140 Ereignisse 20 offene Instanzen für diese Signatur vorliegen. Mit jeder Wiederholung der ersten 110 Ereignisse steigt die Zahl offener Instanzen dieser Signatur um zehn, so dass nach allen 1.000 Wiederholungen 10.000 offene Instanzen dieser Signatur vorliegen.

Tabelle 8-5. Analyseabläufe für die Link-Attacke

Link-Attacke				
Ereignis-nummern	Mit den Ereignissen schaltende Transitionen	Anzahl Token nach dem Schalten der Transitionen auf den Plätzen		Anzahl offene Instanzen
		p_2	p_3	
1-10	t_1	10		10
11-20	t_5		10	10
21-30	t_7		10	10
31-110	-		10	10
111-120	t_1	10		20
121-130	t_5		20	20
131-140	t_7		20	20
141-110.000	...		10.000	10.000

Login-Attacke

In Abb. 8-16 ist ein Signaturnetz für die Login-Attacke skizziert und die bei Verarbeitung der Test-Audit-Trail durchlaufenen Transitionsfolgen sind hervorgehoben. Dem Schema von Tabelle 8-5 folgend stellt Tabelle 8-6 die Abläufe bei der Verarbeitung der ersten 170 Audit-Ereignisse dar.

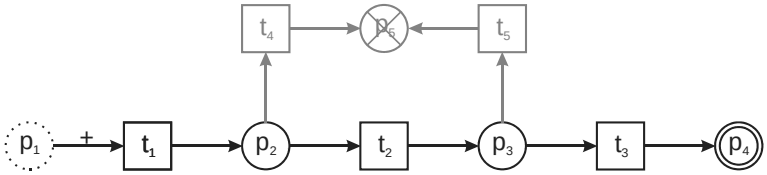


Abb. 8-16. Signaturnetz der Login-Attacke (Skizze)

Wie Tabelle 8-6 zu entnehmen ist, verbleiben nach der Abarbeitung der ersten zehn Manifestierungen der Login-Attacke (Ereignisse 31-60) 20 offene Signaturinstanzen. Es verbleiben jeweils zehn Token auf den Plätzen p_2 und p_3. Im Unterschied zur Link-Attacke unterscheiden sich die ersten und die zweiten (und alle folgenden) zehn Instanzen der Login-Attacke nicht in instanzspezifischen Parametern zu denen hier die Benutzerkennung zählt. Das heißt die zweiten zehn Instanzen dokumentieren Attacken auf die gleichen Nutzerkonten. Dadurch entwickeln sich die verbliebenen offenen Signaturinstanzen weiter. Demzufolge erhöht sich die Anzahl der offenen Instanzen hier nicht mit der Zahl der Wiederholungen der ersten 110 Ereignisse, sondern ist auf 20 beschränkt.

Tabelle 8-6. Analyseabläufe für die Login-Attacke

Login-Attacke				
Ereignis-nummern	Mit den Ereignissen schaltende Transitionen	Anzahl Token nach dem Schalten der Transitionen auf den Plätzen		Anzahl offener Instanzen
		p_2	p_3	
31-40	t_1	10		10
41-50	t_1, t_2	10	10	20
51-60	t_1, t_2, t_3	10	10	20
61-140	-	10	10	20
141-150	t_1, t_2, t_3	10	10	20
151-160	t_1, t_2, t_3	10	10	20
161-170	t_1, t_2, t_3	10	10	20
170-110.000	...	10	10	20

PATH-Attacke

Ein Signaturnetz für die PATH-Attacke ist in Abb. 8-17 skizziert. Bei der Verarbeitung der Test-Audit-Trail schalten alle Transitionen des Signaturnetzes. Die Ereignisse veranlassen zunächst das Schalten der Transitionen t_1, t_2 und t_3, was einem Vervollständigen der Attacke entspricht. Danach folgen Ereignisse, die das Schalten der Transitionen t_4 und t_5 verursachen, wie in Tabelle 8-7 dargestellt. Dies entspricht hier einer Beendigung der an einer Attacke beteiligten Prozesse, so dass prinzipiell mögliche alternative Vervollständigungen der offenen Signaturinstanzen unmöglich werden. Dementsprechend verbleiben nach Abarbeitung der ersten 110 Ereignisse der Test-Audit-Trail keine offenen Instanzen dieser Signatur. Dadurch ist die Zahl der Instanzen dieser Signatur unabhängig von der Anzahl der Wiederholungen der ersten 110 Ereignisse auf zehn beschränkt.

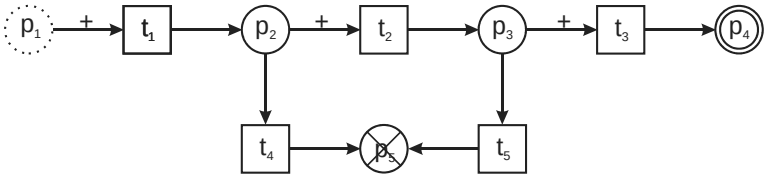


Abb. 8-17. Signaturnetz für die PATH-Attacke (Skizze)

Entsprechend obiger Darstellung enthält die gesamte Test-Audit-Trail 110.000 Ereignisse. Während der Abarbeitung dieser Ereignisse mit den drei genannten Signaturen existieren gleichzeitig bis zu 10.030 Signaturinstanzen.

8.4.2 Vorgehensweise und Messumgebungen

Mit Hilfe der im Folgenden beschriebenen Messungen sollte das Leistungsverhalten der drei Analysemethoden bewertet und verglichen werden. Dazu wurden zwei Experimente durchgeführt:

Experiment I: Vergleich der SAM-Versionen

Im ersten Experiment wurden die Effekte der verschiedenen in Abschn. 8.2 vorgestellten Optimierungsstrategien untersucht. Für die verschiedenen SAM-Versionen wurde gemessen, wie viel Prozessorzeit zur Analyse von jeweils 1.000 Ereignissen benötigt wurde, also die Prozessorzeiten zur Analyse der Ereignisse 1 – 1.000, 1.001 – 2.000, 2.001 – 3.000 usw. Außerdem wurde ermittelt wie viele Bedingungen bei der Analyse der ersten 20.000 Audit-Ereignisse der Test-Audit-Trail überprüft wurden. Dabei wurde zwischen Intra- und Inter-Ereignis-Bedingungen differenziert. Außerdem wurde ermittelt, wie viel Prozessorzeit zur Überprüfung der Bedingungen benötigt wurde.

Tabelle 8-7. Analyseabläufe für die PATH-Attacke

PATH-Attacke				
Ereignis-nummern	Mit den Ereignissen schaltende Transitionen	Anzahl Token nach dem Schalten der Transitionen auf den Plätzen		Anzahl offene Instanzen
		p_2	p_3	
61-70	t_1	10		10
71-80	t_2	10	10	20
81-90	t_3	10	10	20
91-100	t_4		10	10
101-110	t_5			0
111-170	-			0
171-180	t_1	10		10
181-190	t_2	10	10	20
191-200	t_3	10	10	20
201-210	t_4		10	10
211-220	t_5			0
221-110.000	...			0

Experiment II: Vergleich von SAM_6, STAT und CLIPS-IDS

Das zweite Experiment diente einer vergleichenden Untersuchung der Laufzeiten der Systeme SAM_6, STAT und CLIPS-IDS bei der Analyse

der Test-Audit-Trail. Zu diesem Zweck wurde bei jedem Testkandidaten die benötigte Prozessorzeit zur Analyse von jeweils 1.000 Ereignissen ermittelt. Die erhobenen Laufzeiten zur Verarbeitung von jeweils 1.000 Ereignissen wurden zur Beurteilung der Veränderung der Laufzeit in Abhängigkeit von der Anzahl bereits analysierter Ereignisse und damit steigender Anzahl offener Signaturinstanzen herangezogen. Für den Vergleich der drei Systeme wurde die Veränderung der Laufzeit zur Verarbeitung von 1.000 Ereignissen gegenüber der Laufzeit zur Verarbeitung der ersten 1.000 Ereignisse bestimmt. Außerdem wurde der Spitzenspeicherverbrauch der Systeme ermittelt.

Aus verschiedenen technischen Gründen wurden die Experimente mit den Systemen auf zwei verschiedenen Rechnern durchgeführt. Beispielsweise konnte STAT ohne aufwendige Portierungsarbeiten nur auf einem SPARC-Solaris-System getestet werden. Andererseits bieten Pentium-Prozessoren bessere Möglichkeiten um kurze CPU-Zeiten zu vermessen. Die SAM-Versionen und CLIPS-IDS wurden auf Rechner 1, STAT auf Rechner 2 getestet. Die Rechner weisen im Einzelnen folgenden Eigenschaften auf:

- **Rechner 1:** Pentium III 800 MHZ CPU, 320 MB Hauptspeicher, Windows 2000;
- **Rechner 2:** UltraSPARC III+ 900 MHZ CPU, 4 GB Hauptspeicher, Solaris 9.

Die Laufzeiten der Testkandidaten zur Verarbeitung von 1.000 Ereignissen wurden auf Rechner 1 unter Verwendung der Funktion `getProcessTimes()` [Ri99b] ermittelt. Die entsprechenden Messungen auf Rechner 2 wurden mittels der Funktion `getrusage()` [Ste93] durchgeführt.

Da die Auflösung der von der Funktion `getProcessTimes()` verwendeten Uhr zu grob zur Vermessung einzelner Bedingungen ist, wurde zu diesem Zweck die *Read Time Stamp Counter*-Instruktion [She96] des Pentium-Prozessors verwendet. Mit dieser Instruktion wurde die Anzahl benötigter CPU-Ticks zur Auswertung einzelner Bedingungen ermittelt und entsprechend aufkumuliert. Zur Ermittlung des Spitzenspeicherverbrauchs der Testkandidaten wurden die Werkzeuge `taskmanager` auf Rechner 1 und `top` auf Rechner 2 verwendet.

8.4.3 Messergebnisse und Diskussion

Die durchgeführten Experimente führten zu folgenden Ergebnissen.

Experiment I: Vergleich der SAM-Versionen

Zunächst erfolgt eine Diskussion der ermittelten absoluten Laufzeiten der Systeme. Eine Betrachtung der Anzahl geprüfter Bedingungen und der dazu erforderlichen Laufzeit schließt sich an.

Absolute Laufzeiten

In Experiment I wurden Laufzeiten zur Analyse von jeweils 1.000 Ereignissen erhoben. Durch Aufsummieren der einzelnen Werte wurden die absoluten Laufzeiten zur Analyse einer bestimmten Anzahl von Ereignissen ermittelt. Das Diagramm in Abb. 8-18 stellt die absoluten Laufzeiten der verschiedenen SAM-Versionen dar.

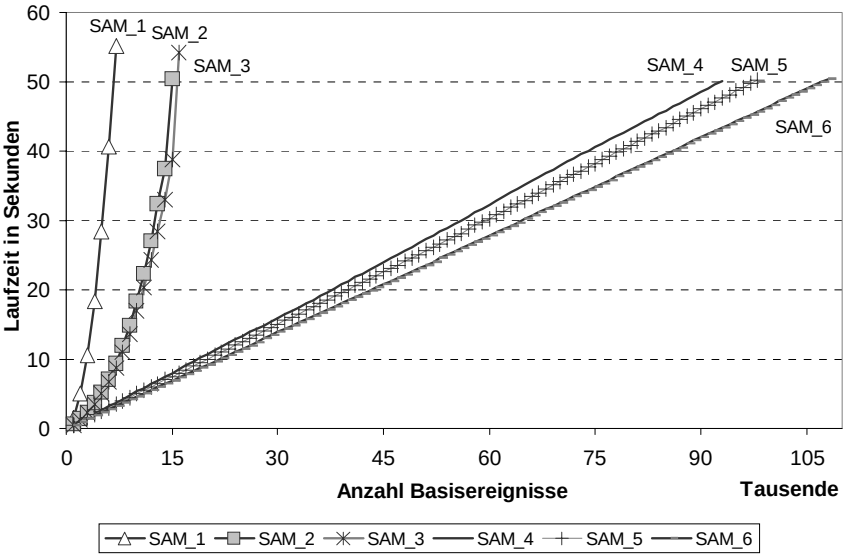


Abb. 8-18. Laufzeiten der SAM-Versionen

SAM_1 kann in dem durchgeführten Test einem naiven Analyseverfahren gleichgestellt werden. Zwar realisiert diese SAM-Version die in Abschn. 8.2.1 vorgestellte Strategie 1 (typbasierte Indizierung der Transitionen), jedoch wurde für den Test eine homogene Typisierung der Ereignisse und Transitionen verwendet (vgl. Abschn. 8.4.1). Somit werden für alle Ereignisse jeweils alle Transitionen ausgewählt, weshalb durch diese Strategie keine Laufzeitverbesserungen gegenüber einem naiven Analyseverfahren erreicht werden können.

Durch die zusätzliche Umsetzung von Strategie 2 (tokenunabhängige Prüfung von Intra-Ereignis-Bedingungen) konnten leichte Laufzeitverbesserungen gegenüber SAM_1 erreicht werden. Diese können durch eine

Reduktion der durchzuführenden Bedingungsprüfungen erklärt werden (vgl. auch Tabelle 8-8 unten), da Intra-Ereignis-Bedingungen nur einmal pro Transition bzw. Ereignis geprüft werden.

SAM_3 erreicht durch zusätzliche Anwendung von Strategie 4 (Gemeinsame Ausdrücke) eine weitere Laufzeitverbesserung. Ein Großteil der mehrfachen Auswertungen identischer Teilausdrücke ist auf wiederholte Überprüfungen von Intra-Ereignis-Bedingungen zurückzuführen. Die Leistungssteigerung von SAM_3 gegenüber SAM_2 fällt allerdings gering aus, da bereits durch Strategie 2 (tokenunabhängige Prüfung von Intra-Ereignis-Bedingungen) redundante Überprüfungen von Intra-Ereignis-Bedingungen vermieden werden.

Aufgrund der Anwendung von Strategie 3 (wertebasierte Tokenindizierung) zeigt SAM_4 eine lineare Abhängigkeit der Laufzeit von der Anzahl zu analysierender Ereignisse. Im vorliegenden Testfall kann mit dieser Strategie effizient das passende Token auf dem Eingangsplatz[1] einer Transition selektiert werden, anstatt über alle vorhandenen Token zu iterieren.

Durch die zusätzliche Umsetzung statischer Bedingungspriorisierung in SAM_5 wird die Leistungsfähigkeit gegenüber SAM_4 nochmals verbessert. Obwohl die Umsetzung der dynamischen Bedingungspriorisierung zusätzlichen Aufwand zur Laufzeit verursacht, konnten mit ihrer Umsetzung in SAM_6 weitere Laufzeitverbesserungen erreicht werden.

Anzahl und Laufzeit durchgeführter Bedingungsauswertungen

Weitere während des Experiments erhobene Messwerte erlauben zusätzliche Einschätzungen der Effizienz der verschiedenen Strategien. Tabelle 8-8 stellt die Anzahl der von den SAM-Versionen durchgeführten Intra- bzw. Inter-Ereignis-Bedingungsüberprüfungen bei der Analyse der ersten 20.000 Audit-Ereignisse dar. Tabelle 8-9 gibt die dabei verbrauchten Prozessorzeiten wieder.

SAM_1 realisiert nur die ereignistypbasierte Transitionsindizierung und trennt nicht zwischen Intra- und Inter-Ereignis-Bedingungen. Dementsprechend werden alle Bedingungen als Inter-Ereignis-Bedingungen behandelt. Aufgrund der Trennung zwischen Intra- und Inter-Ereignis-Bedingungen erreicht SAM_2 eine Reduktion der Anzahl zu überprüfender Bedingungen um circa 90 Prozent. Durch Strategie 4 (gemeinsame Ausdrücke, SAM_3) kann keine weitere Reduktion der Anzahl zu prüfender Bedingungen erreicht werden. Jedoch wird die Auswertungszeit insbesondere der Intra-Ereignis-Bedingungen verringert. Aufgrund von Strategie 2 (tokenunabhängige Prüfung von Intra-Ereignis-Bedingungen) überprüft SAM_3 die Intra-Ereignis-Bedingungen einer Transition nur einmal für ein Ereig-

[1] Im allgemeinen Fall werden Kombinationen von Token auf den Eingansplätzen selektiert.

nis. Identische Intra-Ereignis-Bedingungen anderer Transitionen werden erneut geprüft. Mittels Strategie 4 kann SAM_3 diese Bedingungen jedoch effizient auswerten, indem auf zwischengespeicherte Ergebniswerte zurückgegriffen wird.

Tabelle 8-8. Anzahl geprüfter Bedingungen

	Absolute Anzahl geprüfter Bedingungen	Anzahl geprüfter Intra-Ereignis-Bedingungen	Anzahl geprüfter Inter-Ereignis-bedingungen
SAM_1	123.536.189	-	123.536.189
SAM_2	12.111.503	304.539	11.806.964
SAM_3	12.111.503	304.539	11.806.964
SAM_4	340.868	304.539	36.329
SAM_5	511.191	474.862	36.329
SAM_6	346.370	310.041	36.329

Tabelle 8-9. Laufzeiten zur Prüfung der Intra- und Inter-Ereignis-Bedingungen

	absolute Laufzeit in Sek.	Laufzeit zur Prüfung der Intra-EB in Ticks	Laufzeit zur Prüfung der Inter-EB in Ticks
SAM_1	528,19	-	39.973.974.649
SAM_2	74,64	1.166.863.394	10.707.403.379
SAM_3	74,80	586.452.012	10.153.721.838
SAM_4	9,12	445.889.711	75.830.533
SAM_5	8,98	428.167.261	75.837.252
SAM_6	7,88	388.641.204	75.867.025

Durch Strategie 3 (wertebasierte Tokenindizierung) wird die Anzahl der zu prüfenden Token verringert. Dementsprechend sinkt die Zahl der zu überprüfenden Inter-Ereignis-Bedingungen von 11.806.964 bei SAM_3 auf 36.329 bei SAM_4. Ein umgekehrter Effekt kann bei der Anwendung statischer Bedingungspriorisierung aufgrund einer veränderten Auswertungsreihenfolge der Bedingungen beobachtet werden. Hier steigt die Zahl zu überprüfender Bedingungen von 340.868 bei SAM_4 auf 511.191 bei SAM_5 an. Die erforderliche Laufzeit zur Überprüfung der Bedingungen sinkt jedoch von 9,12 auf 8,98 Sekunden.

Durch Anwendung dynamischer Bedingungspriorisierung wird mit SAM_6 eine weitere Reduktion der Auswertungszeiten von Intra-Ereignis-Bedingungen erreicht. Ursache dafür ist, dass SAM_6 die realen Ausführungszeiten der Bedingungsüberprüfungen und damit auch die Effekte von Strategie 4 (Gemeinsame Ausdrücke) berücksichtigt. Die Messergebnisse zeigen weiter, dass sowohl die statische als auch die dynamische Bedin-

gungspriorisierung keine Verbesserungen hinsichtlich zu überprüfender Inter-Ereignis-Bedingungen bewirkt. Dies ist auf die Spezifik des Testszenarios zurückzuführen, dass die von diesen Strategien ausgenutzten Strukturen in Inter-Ereignis-Bedingungen nicht aufweist. Stattdessen liegen als Ergebnis von Strategie 3 im vorliegenden Testfall nur Token vor, die alle Inter-Ereignis-Bedingungen erfüllen. Da keine dieser Bedingungen negativ ausgewertet wird, hat die Reihenfolge ihrer Auswertung keinen Effekt auf die erforderliche Laufzeit.

Experiment II: Vergleich von SAM_6, STAT und CLIPS-IDS

Zuerst werden die ermittelten Laufzeiten dargestellt und diskutiert bevor der Speicherverbrauch der Systeme betrachtet wird.

Laufzeiten

Die bei Experiment II erhobenen Werte wurden verwendet, um die Veränderung der Laufzeit der Systeme zur Analyse von 1.000 Ereignissen gegenüber der Laufzeit zur Analyse der ersten 1.000 Ereignisse zu ermitteln. Die entsprechenden Werte für die Systeme STAT, CLIPS-IDS und SAM_6 sind in dem Diagramm in Abb. 8-19 dargestellt. Unter den Namen der Systeme im Diagramm sind jeweils die absoluten Laufzeiten zur Analyse der gesamten Test-Audit-Trail notiert.

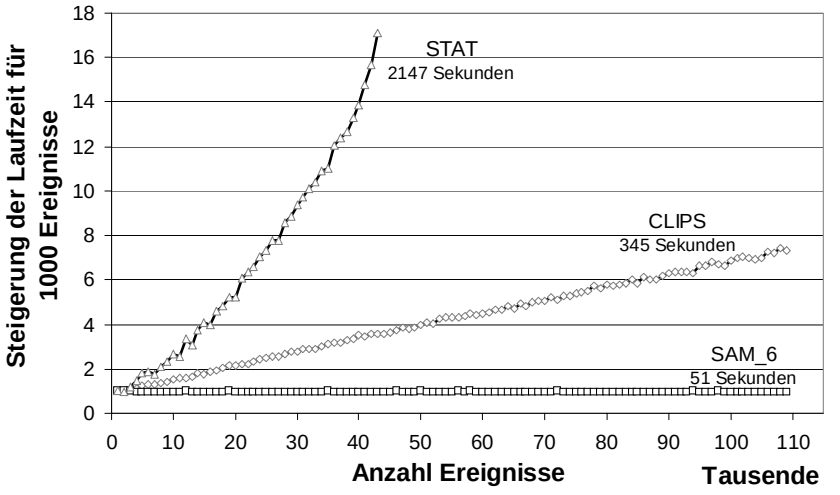


Abb. 8-19. Laufzeitveränderungen von STAT, CLIPS-IDS und SAM_6

Aus dem Diagramm ist ersichtlich, dass STAT zur Analyse der Ereignisse 20.001 – 21.000 ungefähr das Fünffache der Zeit benötigt, die zur

Analyse der Ereignisse 1 – 1.000 verbraucht wurde. Die Veränderung der Laufzeit von STAT hängt polynomiell von der Anzahl bereits analysierter Ereignisse bzw. offener Signaturinstanzen ab. Ursache für dieses Verhalten ist die instanzenunabhängige Analysemethodik von STAT (vgl. Abschn. 8.1.1). STAT nutzt weder strukturelle Charakteristika von Signaturen noch Beziehungen zwischen verschiedenen Instanzen einer Signatur bei der Analyse aus.

Im Gegensatz dazu zeigt CLIPS-IDS ein lineares Wachstum der Veränderung der Analysezeiten bei steigender Anzahl von Signaturinstanzen. Dies ist das Resultat der Anwendung des Rete-Algorithmus (vgl. Abschn. 8.1.2). Zwar werden Instanzen auch hier unabhängig voneinander analysiert, jedoch werden identische Bedingungen und Bedingungsteile nur einmal analysiert.

Im Unterschied zu STAT und CLIPS-IDS zeigt SAM_6 auch bei wachsender Anzahl von Instanzen keine Veränderung der Laufzeit zur Analyse von 1000 Ereignissen. Dies ist auf die verschiedenen in Abschn. 8.2 vorgestellten Optimierungsstrategien zurückzuführen.

Speicherverbrauch der Systeme

Die von SAM_6 umgesetzten Optimierungsstrategien pflegen zur Laufzeit Wertetabellen und speichern Ergebnisse von Bedingungsauswertungen. Um zu untersuchen, inwieweit die Laufzeitverbesserungen von SAM_6 durch einen erhöhten Speicherverbrauch erkauft werden, wurde während des Experiments jeweils der Spitzenspeicherverbrauch der Testkandidaten ermittelt. Tabelle 8-10 stellt die ermittelten Werte dar, die aufzeigen, dass der Speicherverbrauch von SAM_6 deutlich unter den Werten der anderen Testkandidaten liegt.

Tabelle 8-10. Speicherplatzverbrauch der verglichenen Systeme

Testkandidat	Spitzenspeicherverbrauch
SAM_6	36 MB
CLIPS-IDS	58 MB
STAT	638 MB

8.5 Zusammenfassung

In diesem Kapitel wurden existierende Analyseverfahren von Missbrauchserkennungssystemen hinsichtlich ihrer Effizienz untersucht. Dabei stellte sich die Verwendung separater Programm-Module als ineffizient heraus. Für expertensystembasierte Analysen wurde aufgezeigt, dass die Optimie-

rungen des Rete-Algorithmus bei der Missbrauchserkennung nicht ihre volle Wirkung zeigen und teilweise unnötigen Mehraufwand verursachen, wodurch die Effizienz des Ansatzes fragwürdig ist. Deshalb wurden in diesem Kapitel Optimierungsstrategien entwickelt, die strukturelle Eigenschaften und Redundanzen von Signaturen sowie Charakteristika der Analyse zur Effizienzsteigerung ausnutzen. Des Weiteren wurden die entwickelten Optimierungen experimentell untersucht und mit existierenden Verfahren verglichen. Als Ergebnis konnten deutliche Effizienzsteigerungen aufgezeigt werden.

9 Zusammenfassung und Ausblick

In diesem Buch wurden verschiedene Ansätze zur Steigerung der Wirksamkeit der Missbrauchserkennung vorgestellt. Ausgehend von den Problemen existierender Ansätze, wie hohen Fehlalarmraten, wurde insbesondere der Problembereich der Signaturmodellierung und -beschreibung betrachtet. Dabei wurden ungenaue Signaturen als eine Hauptursache der Probleme identifiziert. Zur Überwindung der bestehenden Schwierigkeiten wurde zunächst untersucht, welche Aspekte von Signaturen einer Charakterisierung bedürfen. Als Ergebnis wurde in Kapitel 5 ein Modell semantischer Aspekte von Angriffssignaturen entwickelt, das systematisch Anforderungen an Modellierungs- und Beschreibungsansätze hinsichtlich einer exakten und vollständigen Beschreibung von Signaturen definiert.

Mit den Signaturnetzen wurde in Kapitel 6 ein an Petrinetzen angelehnter Ansatz zur Modellierung und Veranschaulichung von Signaturen eingeführt, der allen ermittelten semantischen Anforderungen gerecht wird. Darüber hinaus, erlauben Signaturnetze die Simulation von Erkennungsabläufen anhand graphischer Darstellungen, welche die Veranschaulichung von Zwischenzuständen ermöglicht, und so ein Nachvollziehen und Testen von Signatureigenschaften erlaubt.

In Kapitel 7 wurden mit den Sprachen SHEDEL und EDL konkrete Werkzeuge zur Beschreibung von Signaturen vorgestellt. Beide Sprachen wurden anhand der in Kapitel 5 eingeführten Systematik mit existierenden Sprachen verglichen. Für die an Signaturnetzen angelehnte Sprache EDL wurde das Werkzeug SEG entwickelt, das eine graphische Erstellung von Signaturen sowie die Simulation und schrittweise Ausführung entsprechender Erkennungsabläufe ermöglicht. Eine Vorabversion dieses Werkzeugs und verschiedene Beispielsignaturen sind auf der diesem Buch beiliegenden CD-ROM enthalten.

Ausgehend von der steigenden Leistungsfähigkeit moderner IT-Systeme sowie der aus der zunehmenden Komplexität resultierenden wachsenden Zahl von Verwundbarkeiten wurde die Erkennungseffizienz von Missrauchserkennungssystemen als weitere Herausforderung identifiziert. In Kapitel 8 wurden existierende Erkennungsverfahren zur Missbrauchserkennung vorgestellt und insbesondere hinsichtlich der Erkennungseffizienz diskutiert. Von gewonnenen Einsichten in spezifische Eigenschaften von

Signaturen und deren Analyse ausgehend, wurden verschiedene Optimierungsstrategien zur Steigerung der Erkennungseffizienz vorgestellt. Das in Kapitel 8 vorgestellte Analysewerkzeug SAM implementiert die entwickelten Optimierungen und wurde für verschiedene Experimente verwendet. Durch einen experimentellen Vergleich von SAM und eine Auswahl existierender Erkennungsverfahren wurden die durch die Optimierungen erreichbaren Leistungssteigerungen aufgezeigt.

Die Leistungsfähigkeit von IT-Systemen wird weiter zunehmen. Daher besteht weiterhin Bedarf an der Leistungssteigerung von IDS. Neben algorithmischen und verfahrenstechnischen Optimierungen spielen hier auch architekturelle Überlegungen eine Rolle. Dazu gehört die Platzierung bzw. Anordnung der Aufzeichnungs- und Analysekomponenten in einem möglicherweise verteilten IDS sowie die Organisation des Datenaustauschs. Hier besteht weiterer Forschungsbedarf beispielsweise hinsichtlich einer Lastverteilung zwischen Analysekomponenten. Eine Verteilung von Audit-Daten oder Signaturen an verschiedene Analysekomponenten ist ebenso vorstellbar, wie die Verteilung von Teilsignaturen oder eine Parallelisierung der Analyse. Die Realisierbarkeit und die erreichbaren Leistungssteigerungen bedürfen weiterer Untersuchungen.

Hinsichtlich der Exaktheit und Vollständigkeit der Beschreibung von Signaturen konnte durch die in den Kapiteln 5 bis 7 dargestellten Erkenntnisse eine Systematisierung erreicht werden. Gleichzeitig wurde mit EDL eine Sprache vorgestellt, die sich neben ihrer Ausdrucksstärke durch einfache und intuitive Beschreibungsmöglichkeiten auszeichnet, und so eine effektive Signaturentwicklung unterstützt. Zur effektiven und effizienten Entwicklung von Signaturen sind jedoch auch geeignete Verfahren zur Ermittlung der charakteristischen Kriterien zur Erkennung einer Attacke erforderlich. Derzeit sind weder manuelle noch automatische Verfahren bekannt, die für eine systematische Ableitung von Signaturen beispielsweise aus vorliegenden Schadprogrammen sog. Exploits verwendet werden können. Ihre Ableitung erfolgt zumeist empirisch auf der Grundlage jahrelanger Erfahrungen von Sicherheitsadministratoren. Entsprechend besteht weiterer Forschungsbedarf, um den empirischen Anteil in der Signaturentwicklung durch Entwicklung systematischer Vorgehensweisen zu reduzieren, so dass exakte Signaturen in kürzeren Entwicklungszeiten erstellt werden können. Erste Ideen zur Entwicklung von Ableitungsmethoden wurden bereits vorgestellt [Schme+06], bedürfen jedoch weiterer Untersuchung. In diesem Zusammenhang sind außerdem Methoden zur Validierung von Signaturen von großem Interesse.

10 Anhang

10.1 Signatur der nebenläufigen Link-Attacke in EDL

In Abschnitt 7.4.2 wurde die EDL-Signatur für die nebenläufige Link-Attacke (vgl. Abschnitt 4.2.4) diskutiert und skizziert. Das Signaturnetz für diese Signatur ist in Abb. 7-22 auf Seite 141 abgebildet. Nachfolgend ist die vollständige Signatur in EDL dargestellt.

```
EVENT ConcurrentLinkAttack
{
PLACES
  p1
  {
    TYPE INITIAL
  }
  p2
  {
    TYPE INITIAL
  }
  p3
  {
    FEATURES
      STRING mLinkName,
      STRING mScriptName,
      STRING mScriptOwner
  }
  p4
  {
    FEATURES
      STRING mLinkName,
      STRING mScriptName,
      INT    mScriptOwner
  }
  p5
  {
    FEATURES
      STRING mScriptName,
  }
```

```
  p6 // Abbruchplatz; steht auch für p7, p9, p10 im
     // Signaturnetz
  {
    TYPE ESCAPE
  }
  p8
  {
    TYPE EXIT
    FEATURES
      STRING mLinkName,
      STRING mScriptName,
      INT    mScriptOwner,
      INT    mExecutorID
  }

TRANSITIONS

  // t1 -- create link ohne "-"
  p1(+) p3
  {
    TYPE SOLARIS_AUDIT_EVENT
    CONDITIONS
      (EVENT==5) OR (EVENT==21),  // Systemruf link()
      ROWNER != AUDIT_ID,   // Link auf eine fremde Datei
      RSCRIPT != "",        // Link auf ein Script
      NOT(REGCMP(".*-.*", RRNAME)),  // Linkname ohne "-"
      PERM(9, RPERM)      // Skript ist für andere ausführbar
    MAPPINGS
      [p3].mLinkName   = RRNAME,
      [p3].mScriptName = RNAME,
      [p3].ScriptOwner = ROWNER
  }

  // t2 –- create link mit "-"
  p1(+) p4
  {
    TYPE SOLARIS_AUDIT_EVENT
    CONDITIONS
      (EVENT==5) OR (EVENT==21),  // Systemruf link()
      ROWNER != AUDIT_ID,  // Link auf eine fremde Datei
      RSCRIPT != "",       // Link auf ein Script
      REGCMP(".*-.*", RRNAME),  // Linkname besitzt "-"
      PERM(9, RPERM)  // Skript ist für andere ausführbar
    MAPPINGS
      [p3].mLinkName   = RRNAME,
      [p3].mScriptName = RNAME,
      [p3].ScriptOwner = ROWNER
    ACTIONS
      WARNLN("Link-Attacke: Verdächtiger Link: " + RRNAME)
  }
```

```
// t3 – chmod
p2(+) p5
{
  TYPE SOLARIS_AUDIT_EVENT
  CONDITIONS
    (EVENT==10) OR (EVENT==39), // Systemruf chmod() oder
                                // fchmod()
    RSCRIPT != "",              // ein Script
    PERM(2048, STATUS)          // SUID gesetzt
  MAPPINGS
    [p5].mScriptName = RNAME
  ACTIONS
    WARNLN("Link-Attacke: SUID-Skript erstellt: " + RNAME)
}

// t4 –- rename zu ohne "-"
p3(-) p3
{
  TYPE SOLARIS_AUDIT_EVENT
  CONDITIONS
    (EVENT==42),                    // Systemruf rename()
    p3.mLinkName == RNAME,          // verfolgter Linkname
    NOT(REGCMP(".*-.*", RRNAME)),   // neuer Linkname
                                    // besitzt kein "-"
  MAPPINGS
    [p3].mLinkName = RRNAME,
    [p3].mScriptName = p3.mScriptName,
    [p3].mScriptOwner = p3.mScriptOwner
}

// t5 -- rename zu "-"
p3(-) p4
{
  TYPE SOLARIS_AUDIT_EVENT
  CONDITIONS
    (EVENT==42),              // Systemruf rename()
    p3.mLinkName == RNAME,    // verfolgter Linkname
    REGCMP(".*-.*", RRNAME),  // neuer Linkname besitzt "-"
  MAPPINGS
    [p4].mLinkName = RRNAME,
    [p4].mScriptName = p3.mScriptName,
    [p4].mScriptOwner = p3.mScriptOwner
  ACTIONS
    WARNLN("Link-Attacke: Verdächtiger Link: " + RRNAME)
}
```

```
// t6 –- rename zu ohne "-"
p4(-) p3
{
  TYPE SOLARIS_AUDIT_EVENT
  CONDITIONS
    (EVENT==42),                      // Systemruf rename()
    p4.mLinkName == RNAME,            // verfolgter Linkname
    NOT(REGCMP(".*-.*", RRNAME)),     // neuer Linkname
                                      // ohne "-"
  MAPPINGS
    [p3].mLinkName = RRNAME,
    [p3].mScriptName = p4.mScriptName,
    [p3].mScriptOwner = p4.mScriptOwner
}

// t7 -- rename zu "-"
p4(-) p4
{
  TYPE SOLARIS_AUDIT_EVENT
  CONDITIONS
    (EVENT==42),                    // Systemruf rename()
    p4.mLinkName == RNAME,          // verfolgter Linkname
    NOT(REGCMP(".*-.*", RRNAME)), // neuer Linkname mit "-"
  MAPPINGS
    [p4].mLinkName = RRNAME,
    [p4].mScriptName = p4.mScriptName,
    [p4].mScriptOwner = p4.mScriptOwner
}

// t8 -- rename
p5(-) p5
{
  TYPE SOLARIS_AUDIT_EVENT
  CONDITIONS
    (EVENT==42),           // Systemruf rename()
    p5.mLinkName == RNAME  // verfolgter Skriptname
  MAPPINGS
    [p5].mScriptName = RRNAME
}

// t9 -- delete Link
p3(-) p6
{
  TYPE SOLARIS_AUDIT_EVENT
  CONDITIONS
    (EVENT==6),            // Systemruf unlink() (delete)
    p3.mLinkName == RNAME  // verfolgter Linkname
}
```

```
  // t10 -- delete Link
  p4(-) p7
  {
    TYPE SOLARIS_AUDIT_EVENT
    CONDITIONS
      (EVENT==6),            // Systemruf unlink() (delete)
      p4.mLinkName == RNAME  // verfolgter Linkname
  }

  // t11 -- execute
  p4(+), p5(+) p8
  {
    TYPE SOLARIS_AUDIT_EVENT
    CONDITIONS
      (EVENT==7) OR (EVENT==23),  // Systemruf exec() oder
                                  // execve()
      p4.mLinkName == RNAME,      // verfolgter Linkname
      p4.mScriptName == p5.mScriptName,  // verlinktes Skript
                                         // ist SUID
      EUID!=AUDIT_ID              // Subjekt agiert unter
                                  // fremder Identität
    MAPPINGS
      [p8].mLinkName = p4.mLinkName,
      [p8].mScriptName = p4.mScriptName,
      [p8].mScriptOwner = p4.mScriptOwner,
      [p8].executor = AUDIT_ID
    ACTIONS
      WARNLN("Link-Attacke erkannt!")
  }

  // t12 -- delete Skript
  p5(-) p9
  {
    TYPE SOLARIS_AUDIT_EVENT
    CONDITIONS
      (EVENT==6),              // Systemruf unlink() (delete)
      p5.mScriptName == RNAME  // verfolgter Skriptname
  }

  // t13 -- chmod
  p5(-) p10
  {
    TYPE SOLARIS_AUDIT_EVENT
    CONDITIONS
      (EVENT==10) OR (EVENT==39), // Systemruf chmod() oder
                                  // fchmod()
      p4.mScriptName == RNAME     // verfolgter Skriptname
      NOT(PERM(2048, STATUS))     // kein SUID gesetzt
  }
} // End of Event
```

Index

Literatur

[Aho+88] Aho, A. V.; Sethi, R.; Ullman, J. D.: Compilers, Techniques and Tools. Addison-Wesley, 1988.

[An+03] Anagnostakis, K. G.: Markatos, E. P.; Antonatos, S.; Polychronakis, M.: E2xB: A domainspecific string matching algorithm for intrusion detection. In: Proceedings of the 18th IFIP International Information Security Conference (SEC 2003), S. 217–228, Kluwer, 2003.

[Ax00] Axelsson, S.: The base-rate fallacy and the difficulty of intrusion detection. In: ACM Transactions on Information and System Security. Volume 3, Issue 3, August 2000, S. 186–205, 2000.

[Ax98] Axelsson, S.: Research in Intrusion-Detection systems: A Survey. Chalmers University of Technology, Department of Computer Engineering, Technical Report 98-17, Göteborg, 1998.

[Ba00] Bace, R. G.: Intrusion Detection. ISBN: 1-57870-185-6, Macmillan Technical Publishing, 2000.

[Bau90] Baumgarten, B.: Petri-Netze: Grundlagen und Anwendungen. ISBN: 3-411-14291-X, BI-Wissenschaftsverlag, 1990.

[Be01] Betser, J.: The Challenge of Creating Productive Collaborating, Information Assurance Communities via Internet Research and Standards; invited postition paper IEEE Symposium on Reliable Distributed Systems (SRDS), 2001.
http://csdl2.computer.org/comp/proceedings/srds/2001/1366/00/13660070.pdf

[Be91] Berndtsson, M.: ACOOD: An approach to an Active Object-Oriented DBMS. Master thesis, University of Skövde, Department of Computer Science, Skövde, Sweden, 1991.

[Bi01] Bischof, N.: Spezifikation und Identifizierung von IT-Sicherheitsverletzungen. Diplomarbeit, Brandenburgische Technische Universität Cottbus, Lehrstuhl Rechnernetze und Kommunikationssysteme, 2001.

[Bi03] Bishop, M.: Computer Security – Art and Science. ISBN: 0-201-44099-7, Pearson Education, 2003.

[Brü+04] Brügge, B.; Dutoit, A. H.: Objektorientierte Softwaretechnik. ISBN: 3-8273-7082-5, Pearson Studium, 2004.

[Bü+99] Büschkes, R.; Borning, M.; Kesdogan, D.: Transaction-based Anomaly Detection. In: Proceedings of the Workshop on Intrusion Detection and Network Monitoring, S. 129–140, USENIX, 1999.

[Bü01] Büschkes, R.: Angriffserkennung in Kommunikationsnetzen. Dissertation, Fakultät für Mathematik, Informatik und Naturwissenschaften, Rheinisch-Westfälische Technische Hochschule Aachen, 2001.

[CE05] CERT Coordination Center: CERT/CC Statistics 1988 – 2005. Carnegie Mellon University, 8. August, 2005. http://www.cert.org/stats/cert_stats.html

[Ch+94] Chakravarthy, S.; Krishnaprasad, V; Anwar, E.; Kim, S.: Composite Events for Active Databases: Semantics, Contexts and Detection. In: Proceedings of the 20th International Conference on Very Large Databases, S. 606–617, Morgan Kaufmann, 1994.

[Chri+93] Christensen, S.; Hansen, N. D.: Coloured Petri Nets Extended with Place Capacities, Test Arcs and Inhibitor Arcs. In: Proceedings of the 14th International Conference on Application and Theory of Petri Nets, LNCS 691, S. 186–205, Springer Verlag, 1993.

[Ci02] Cisco Systems, Inc.: NetFlow Services and Applications. White Paper, July 15, 2002. http://www.cisco.com/warp/public/cc/pd/iosw/ioft/neflct/tech/napps_wp.htm

[Co+96] Collet, C.; Coupaye, T.: Composite Events in NAOS. In: Prococeeding of the 7th International Conference on Database and Expert Systems Applications, LNCS 1134, S. 475–481, Springer Verlag, 1996.

[Cro+96] Crosbie, M.; Dole, B.; Ellis, T.; Krsul, I.; Spafford, E.: IDIOT – User Guide. Department of Computer Science, Purdue University, West Lafayette, Indiana, Technical Report TR 99-050, 1996.

[Cu+00] Cuppens, F.; Ortalo, R.: LAMBDA: A Language to Model a Database for Detection of Attacks. In: Proceedings of the Third International Workshop on Recent Advances in Intrusion Detection, LNCS 1907, S. 197–216, Springer Verlag, 2000.

[Da+92] David, R.; Alla, H.: Petri nets and Grafcet: tools for modeling discrete event systems. ISBN: 0-13-327537-X, Prentice Hall, 1992.

[Da+96] Dayal, U.; Buchmann, A.; Chakravarthy, S.: The HiPAC Project. In: Active Database Systems. ISBN: 1-55860-304-2, Morgan Kaufmann, 1996.

[Da02] Damm, M.: Entwicklung eines Prototypen zum benutzerfreundlichen Management von Signaturen in SHEDEL-Notation. Studienarbeit, Brandenburgische Technische Universität Cottbus, Lehrstuhl Rechnernetze und Kommunikationssysteme, 2002.

[De+01] Debar, H.; Wespi, A.: Aggregation and Correlation of Intrusion-Detection Alerts. In: Proceedings of the 4th Symposium on Recent Advances in Intrusion Detection (RAID 2001), LNCS 2212, S. 85–103, Springer Verlag, 2001.

[De+02] Debar, H.; Morin, B.: Evaluation of the Diagnostic Capabilities of Commercial Intrusion Detection Systems. In: Proceedings of the Fifth International Symposium on Recent Advances in Intrusion Detection (RAID 2002), LNCS 2516, S. 177–198, Springer Verlag, 2002.

[De+05] Debar, H.; Curry, D.; Feinstein, B.: The Intrusion Detection Message Exchange Format (draft-ietf-idwg-idmef-xml-14). Internet Draft, Work in Progress, The Internet Society, 2005.
http://www.ietf.org/internet-drafts/draft-ietf-idwg-idmef-xml-14.txt

[De+92] Debar, H.; Becker, M.; Siboni, D.: A neural network component for an intrusion detection system. In: Proceedings of the IEEE Computer Society Symposium on Research in Security and Privacy, S. 240–250, IEEE Computer Society Press, 1992.

[Die04] Dierstein, R.: Sicherheit in der Informationstechnik – Der Begriff IT-Sicherheit. In: Informatik-Spektrum, Band 27, Nr. 4, S. 343–353, Springer Verlag, 2004.

[Eck+00a] Eckmann, S. T.; Vigna, G.; Kemmerer, R. A.: STATL: An Attack Language for State-based Intrusion Detection. In: Proceedings of the 1st ACM Workshop on Intrusion Detection, 2000.

[Eck+00b] Eckmann, S.T.; Vigna, G.; Kemmerer, R. A.: STATL Syntax and Semantics. Computer Science Department, University of California, Santa Barbara, Technical Report TRCS20-19, 2000.

[Eck+02] Eckmann, S. T.; Vigna, G.; Kemmerer, R. A.: STATL: An Attack Language for State-based Intrusion Detection. In: Journal of Computer Security. Vol. 10 (2002), Nr. 1–2, S. 71–104, IOS Press, 2002.

[Eck02] Eckert, C.: IT-Sicherheit: Konzepte – Verfahren – Protokolle. ISBN: 3486272055, Oldenbourg Verlag, 2002.

[Fie04] Fieber, M.: Entwurf und Implementierung eines generischen, adaptiven Werkzeugs zur Arbeit mit Graphen. Diplomarbeit, Brandenburgische Technische Universität Cottbus, Lehrstuhl Datenstrukturen und Softwarezuverlässigkeit, 2004.

[Fle06] Pseudonymizing Audit Data for Privacy Respecting Misuse Detection. Dissertation, Fachbereich Informatik, Universität Dortmund, 2006.

[FleMei02] Flegel, U.; Meier, M.: Towards a scalable approach to tailoring the disclosure of pseudonymous audit data to misuse detection signatures. Internes Diskussionspapier, 2002.

[Fo+96] Forrest, S.; Hofmeyr, S. A.; Somayaji, A.; Longstaff, T. A.: A sense of self for Unix processes. In: Proceedings of the IEEE Computer Society Symposium on Research in Security and Privacy, S. 120–128, IEEE Computer Society Press, 1996.

[For82] Forgy, C. L.: Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem. In: Artificial Intelligence, 19 (1982) 10, S. 17–37, 1982.

[FSF05] Free Software Foundation: Bison. http://www.gnu.org/software/bison/bison.html, July 2nd, 2005.

[FSF99] Free Software Foundation: Flex. http://www.gnu.org/software/flex/, April 29th, 1999.

[Gi+94] Giarratano, J.; Riley, G.: Expert Systems – Principles and Programming. 2nd Edition. ISBN: 0-534-93744-6, PWS Publishing Company, 1994.

[Go99] Gollmann, D.: Computer Security. ISBN: 0-471-97844-2, Wiley & Sons, 1999.

[Gri05] Grisby, D.: omniORB – Free High Performance ORB. http://omniorb.sourceforge.net/

[Ha+92] Habra, N.; Le Charlier, B.; Mounji, A.; Mathieu, I.: ASAX: Software architecture and rule-based language for universal audit trail analysis. In: Proceedings of the 2nd European Symposium on Research in Computer Security (ESORICS' 92), LNCS 648, S. 435–450, Springer Verlag, 1992.

[Ho99] Holz, T.: Der AID-Monitoring-Agent für Windows NT - Teil 1. Technischer Bericht E/I11S/X0034/Q5123 - II.d, Brandenburgische Technische Universität Cottbus, Lehrstuhl Rechnernetze und Kommunikationssysteme, 1999.

[HoMei+02a] Holz, T.; Meier, M.; Koenig, H.: High-Efficient Intrusion Detection Infrastructure. In: Proceedings of the NATO Symposium "Real Time Intrusion Detection", ISBN: 92-837-0032-5, NATO report RTO-MP-101, 2002.

[HoMei+02b] Holz, T.; Meier, M.; Koenig, H.: Bausteine für effiziente Intrusion Detection Systeme. In PIK 25 (2002) 3, S. 144–157, K. G.. Saur Verlag, 2002.

[Il93] Ilgun, K.: USTAT: A Real-Time Intrusion Detection System for UNIX. In Proceedings of the IEEE Symposium on Security and Privacy, S. 16–28, IEEE Computer Society Press, 1993.

[ISO98] International Organization for Standardization (ISO): Programming Languages – C++. International Standard, ISO/IEC 14882, 1998.

[Ja+89] Jacobson, V.; Leres, C.; McCanne, S.: The Tcpdump Manual Page. Lawrence Berkeley Laboratory, Berkeley, CA, June 1989. ftp://ftp.ee.lbl.gov/tcpdump.tar.Z

[Ja+91] Javitz, H. S.; Valdes, A.: The SRI IDES statistical anomaly detector. In: Proceedings of the IEEE Computer Society Symposium on Research in Security and Privacy, S. 316–326, IEEE Computer Society Press, 1991.

[Ja99] Jackson, P.: Introduction to Expert Systems. 3rd Edition, Addison Wesley Longman, Ltd., Harlow, 1999.

[Jen92] Jensen, K.: Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use – Volume 1. ISBN: 3-540-55597-8, Springer Verlag, 1992.

[Jo99] Josuttis, N. M.: The C++ Standard Library. ISBN: 0-201-37926-0, Addison Wesley Longman, 1999.

[Ju+98] Jumes, J. G.; Cooper, N. F.; Chamoun, P.; Feinman, T. M.: Microsoft Windows NT 4.0 Security, Audit, and Control. ISBN: 157231818X, Microsoft Press, 1998.

[Ju00] Julisch, K.: Dealing with False Positives in Intrusion Detection. Präsentation auf dem Symposium on Recent Advances in Intrusion Detection (RAID 2000), 2000, http://www.raid-symposium.org/raid2000/Materials/Abstracts/ 50/Julisch_foils_RAID2000.pdf http://www.raid-symposium.org/raid2000/Materials/Abstracts/ 50/50.pdf

[Ju03] Julisch, K.: Using Root Cause Analysis to Handle Intrusion Detection Alarms. Dissertation, Fachbereich Informatik, Universität Dortmund, 2003.

[Ka+98] Kahn, C.; Porras, P.; Staniford-Chen, S.; Tung, B.: A Common Intrusion Detection Framework. Submitted to the Journal of Computer Security, 1998, http://www.isi.edu/gost/cidf/papers/cidf-jcs.ps

[Ka01] Kannegieser, J.: Audit-basierte Einbruchserkennung mit Windows 2000 - Möglichkeiten und Grenzen. Diplomarbeit, Brandenburgische Technische Universität Cottbus, Lehrstuhl Rechnernetze und Kommunikationssysteme, 2001.

[Ko+94] Ko, C.; Levitt, K.: Automated Detection of Vulnerabilities in Privileged Programs by Execution Monitoring. In: Proceedings of the 10th Annual Computer Security Applications Conference, S. 134–144, IEEE Computer Society Press, 1994.

[Ko04] Koalick, S.: Entwurf und Implementierung von Übersetzungswerkzeugen zur Transformation von Attacken-Signaturen zwischen verschiedenen Beschreibungssprachen. Bachelorarbeit, Brandenburgische Technische Universität Cottbus, Lehrstuhl Rechnernetze und Kommunikationssysteme, 2004.

[Ko96] Ko, C.: Execution Monitoring of Security-Critical Programs in a Distributed System: A Specification-Based Approach. Ph.D. Thesis, University of California at Davis, 1996.

[Krau04] Krauz, R.: Implementierung eines auf dem Expertensystem-Tool CLIPS basierenden Intrusion Detection Systems. Studienarbeit, Brandenburgische technische Universität Cottbus, Lehrstuhl Rechnernetze und Kommunikationssysteme, 2004.

[Krü+03] Krügel, C.; Toth, T.: Using Decision Trees to Improve Signature-based Intrusion Detection. In: Proceedings of the 6th Symposium on Recent Advances in Intrusion Detection (RAID 2003), LNCS 2820, S. 173–191, Springer Verlag, 2003.

[Krü02] Krügel, C.: Network Alertness – Towards an adaptive, collaborating Intrusion Detection System. Dissertation, Fakultät für Technische Naturwissenschaften und Informatik, Technische Universität Wien, 2002.

[Ku+94a] Kumar, S.; Spafford, E.: An Application of Pattern Matching in Intrusion Detection. Department of Computer Science, Purdue University, West Lafayette, Indiana, Technical Report TR 94-013, 1994.

[Ku+94b] Kumar, S.; Spafford, E.: A Pattern Matching Model For Misuse Intrusion Detection. In: Proceedings of the 17th National Computer Security Conference, S. 11–21, 1994.

[Ku95] Kumar, S.: Classification and Detection of Computer Intrusions. Ph.D. Thesis, Purdue University, West Lafayette, Indiana, USA, 1995.

[La+04] Laskov, P.; Schäfer, C.; Kotenko, I.; Müller, K.-R.: Intrusion Detection in Unlabeled Data with Quater-Sphere Support Vector Machines. In: Praxis der Informationstechnik (PIK), Jahrgang 27, Nr. 4, S. 228–236. K.G. Saur Verlag, 2004.

[Le+99] Lee, W.; Stolfo, S. J.; Mok, K. W.: A data mining framework for building intrusion detection models. In: Proceedings of the IEEE Computer Society Symposium on Research in Security and Privacy, S. 120–132, IEEE Computer Society Press, 1999.

[Li+00a] Lippmann, R.; Fried, D. J.; Graf, I.; Haines, J. W.; Kendall, K. R.; McClung, D.; Weber, D.; Webster, S. H.; Wyschograd, D.; Cunningham, R. K.; Zissman, M. A.: Evaluating Intrusion Detection Systems: The 1998 DARPA Off-Line Intrusion De-tection Evaluation. In: Proceedings of DISCEX 2000, S. 12–26, IEEE Computer Society Press, 2000.

[Li+00b] Lippmann, R.; Haines, J. W.; Fried, D. J.; Korba, J.; Das, K.: The 1999 DARAP Off-Line Intrusion Detection System Evaluation. In Proceedings of the Third International Workshop on Recent Advances in Intrusion Detection (RAID 2000), LNCS 1907, S. 162–182, Springer Verlag, 2000.

[Li+01] Lindqvist, U.; Porras, P. A.: eXpert-BSM: A Host-based Intrusion Detection Solution for Sun Solaris. In: Proceedings of the 17th Annual Computer Security Applications Conference (ACSAC 2001), S. 240–251, IEEE Computer Society, 2001.

[Li+02] Lippmann, R.; Webster, S.; Stetson, D.: The Effect of Identifying Vulnerabilities and Patching Software on the Utility of Network Intrusion Detection. In: Proceedings of the Fifth International Symposium on Recent Advances in Intrusion Detection (RAID 2002), LNCS 2516, S. 307–326, Springer Verlag, 2002.

[Li+98] Lin, J.; Wang, X. S.; Jajodia, S.: Abstraction-Based Misuse Detection: High-Level Specifications and Adaptable Strategies. In: Proceedings of the 11th IEEE Computer Security Foundations Workshop, S. 190–202, IEEE Computer Society, 1998.

[Li+99] Lindqvist, U.; Porras, P. A.: Detecting Computer and Network Misuse with the Production-Based Expert System Toolset (P-BEST). In: Proceedings of the IEEE Symposium on Security and Privacy, S. 146–161, IEEE Computer Society Press, 1999.

[Mc01] McHugh, J.: Intrusion and intrusion detection. In: International Journal of Information Security, Volume 1, No. 1 (August 2001), S. 14–35, Springer Verlag, 2001.

[Mc04] McHugh, J.: Set, Bags and Rock and Roll – Analyzing Large Datasets of Network Data. In: Proceedings of the 9th European Symposium on Research in Computer Security (ESORICS 2004), LNCS 3193, S. 407–422, Springer Verlag, 2004.

[Mé96] Mé, L.: Genetic Algorithms, a Biologically Inspired Approach for Security Audit Trails Analysis. Short paper presented at the 1996 IEEE Computer Society Symposium on Security and Privacy, 1996.

[Mei+02] Meier, M.; Bischof, N.; Holz, T.: SHEDEL - A Simple Hierarchical Event Description Language for Specifying Attack Signatures. In: Proceedings of the 17th International Conference on Information Security. S. 559–571, Kluwer, 2002.

[Mei+05a] Meier, M.; Schmerl, S.: Effiziente Analyseverfahren für Intrusion-Detection-Systeme. In: Proceedings of the Second GI Conference on "Sicherheit - Schutz und Zuverlässigkeit", LNI P-62 S. 209–220, Köllen Verlag, 2005.

[Mei+05b] Meier, M.; Schmerl, S.; Koenig, H.: Improving the Efficiency of Misuse Detection. In: Proceedings of the Second Conference on "Detection of Intrusions & Malware and Vulnerability Assessment" (DIMVA 2005), LNCS 3548, S. 188–205, Springer Verlag, 2005.

[Mei+99] Meier, M.; Richter, B.: Penetration von Solaris 2.x. Technischer Bericht E/I11S/X0034/Q5123 III.a, Brandenburgische Technische Universität Cottbus, Lehrstuhl Rechnernetze und Kommunikationssysteme, 1999.

[Mei04a] Meier, M.: A Model for the Semantics of Attack Signatures in Misuse Detection Systems. In: Proceedings of the 7th Information Security Conference, S. 158–169, LNCS 3225, Springer Verlag, 2004.

[Mei04b] Meier, M.: Intrusion Detection Systems List and Bibliography. 8.12.2004, http://www-rnks.informatik.tu-cottbus.de/en/security/ids.html

[Mi+01] Michel, C.; Mé, L.: ADeLe: an Attack Description Language for Knowledge-based Intrusion Detection. In: Proceedings of the International Conference on Information Security, S. 353–368, Kluwer, 2001.

[Mi+90] Miranker, D. P.; Brant, V.; Lofaso, B. J.; Gadbois, D.: On the performance of lazy matching in production systems. In: Proceedings of the 8th National Conference on Artificial Intelligence, S. 685–692, AAAI Press, 1990.

[Mi87] Miranker, D. P.: Treat: A better match algorithm for AI production systems. In: Proceedings of the Sixth National Conference on Artificial Intelligence, S. 42–47, AAAI Press, 1987.

[Mou97] Mounji, A.: Languages and Tools for Rule-Based Distributed Intrusion Detection. Ph.D. Thesis, Computer Security Institute, University of Namur, Belgium, 1997.

[Neu+99] Neumann, P. G.; Porras, P. A.: Experience with EMERALD to Date. In: Proceedings of the First USENIX Workshop on Intrusion Detection and Network Monitoring. S. 73–80, 1999.

[Ni+01] Ning, P.; Jajodia, S.; Wang, X. S.: Abstraction-Based Intrusion Detection In Distributed Environment. In: ACM Transactions on Information and System Security, Vol. 4 (2001), Nr. 4, S. 407–452, ACM Press, 2001.

[Nie03] Niebert, P.: Petrinetze – Ein anschaulicher Formalismus der Nebenläufigkeit. In: at – Automatisierungstechnik, Vol. 51 (2003), Nr. 3 und 4, S. A5–A12, Oldenbourg Wissenschaftsverlag, 2003.

[Pa+99] Paton, N. W.; Diaz, O.: Active Database Systems. In: ACM Computing Surveys, Volume 1, Number 31, S. 63–103, 1999.

[Pa98] Paton, N. W.: Active Rules in Database Systems. ISBN: 0-201-87686-8, Springer Verlag, 1998.

[Pi04] Pink, M.: Entwurf und Implementierung eines Webbasierten Managementsystems für Intrusion Detection Systeme. Studienarbeit, Brandenburgische Technische Universität Cottbus, Lehrstuhl Rechnernetze und Kommunikationssysteme, 2004.

[Po+97] Porras, P. A.; Neumann, P. G.: EMERALD: Event Monitoring Enabling Responses to Anomalous Live Disturbances. In: Proceedings of the 20th National Information Systems Security Conference, S. 353–365, National Institute of Standards and Technology, 1997.

[Pou+01] Pouzol, J.-P.; Ducassé, M.: From Decalarative Signatures to Misuse IDS. In: Proceedings of the 4th International Symposium on Recent Advances in Intrusion Detection (RAID), LNCS 2212, S. 1–21, Springer Verlag, 2001.

[Pou+02] Pouzol, J. P.; Ducasse, M.: Formal Specification of Intrusion Signatures and Detection Rules. In: Proceedings of the 15th IEEE Computer Security Foundations Workshop (CSFW'02), S. 64–76, IEEE Press, 2002.

[Pri97] Price, K. E.: Host-Based Misuse Detection and Conventional Operating Systems' Audit Data Collection. Master's thesis, Purdue University, West Lafayette, Indiana, USA, 1997.

[Pro94] Proctor, P. E.: Audit reduction and misuse detection in heterogeneous environments: Framework and application. In: Proceedings of the 10th Annual Computer Security Applications Conference, S. 117–125, 1994.

[Pta+98] Ptacek T. H.; Newsham, T. N.: Insertion, Evasion, and Denial of Service: Eluding Network Intrusion Detection. Technical report, Secure Networks, Inc., 1998.

[Pu91] Puppe, F.: Einführung in Expertensysteme. ISBN: 3-540-54023-7, Springer Verlag, 1991.

[Ra02] Ranum, M. J.: Challenges for the Future of Intrusion Detection. Invited Talk, Fifth International Symposium on Recent Advances in Intrusion Detection (RAID 2002), Zürich, 2002.

[Ri+99] Richter, B.; Sobirey, M.: Funktionsbeschreibung des Intrusion-Detection-Systems AID. Technischer Bericht E/I11S/X0034/ Q5123 - I.a, Brandenburgische Technische Universität Cottbus, Lehrstuhl Rechnernetze und Kommunikationssysteme, 1999.

[Ri05] Riley, G..: CLIPS: A Tool for Building Expert Systems. http://www.ghg.net/clips/CLIPS.html, Feburary 3rd, 2005.

[Ri99a] Richter, B.: Die Auditfunktion des Betriebssystems Solaris 2.x. Technischer Bericht E/I11S/X0034/Q5123 - III.b. Brandenburgische Technische Universität Cottbus, Lehrstuhl Rechnernetze und Kommunikationssysteme, 1999.

[Ri99b] Richter, J.: Programming Applications for Microsoft Windows. 4th Edition, ISBN: 1-57261-996-8, Microsoft Press, 1999.

[Ro06] Rohr, C.: Graphische Beschreibung und Simulation von Signaturnetzen. Diplomarbeit (in Bearbeitung), Brandenburgische Technische Universität Cottbus, Lehrstuhl Rechnernetze und Kommunikationssysteme, 2006.

[Roe99] Roesch, M.: Snort – Lightweight Intrusion Detection for Networks. In: Proceedings the 13th Systems Administration Conference (LISA'99), S. 229–238, 1999.

[Sa90] Savory, S. E.: Grundlagen von Expertensystemen. ISBN: 3-486-21524-8, Oldenbourg Verlag, 1990.

[Schae+03] Schaelicke, L.; Slabach, T.; Moore, B.; Freeland, C.: Characterizing the Performance of Network Intrusion Detection Sensors. In: Proceedings of the 6th Symposium on Recent Advances in Intrusion Detection (RAID), LNCS 2820, S. 155–172, Springer Verlag, 2003.

[Schme04] Schmerl, S.: Entwurf und Implementierung einer effizienten Analyseeinheit für Intrusion-Detection-Systeme. Diplomarbeit, Brandenburgische Technische Universität Cottbus, Lehrstuhl Rechnernetze und Kommunikationssysteme, 2004.

[Schme+06] Schmerl, S.; König, H.; Flegel, U.; Meier, M.: Simplifying Signature Derivation by Reuse. In Proceedings of the International Conference on Emerging Trends in Information and Communication Security (ETRICS 2006), LNCS 3995, S. 436–450, Springer Verlag, 2006.

[Se88] Sebring, M. M.; Sellhouse, E.; Hanna, M. E.; Whitehurst, R. A.: Expert system in intrusion detection: A case study. In: Proceedings of the 11th National Computer Security Conference, 1988, S. 74–81, 1988.

[She96] Shemitz, J.: Using RDTSC for benchmarking code on Pentium computers. June 17th, 1996. http://www.midnightbeach.com/jon/pubs/rdtsc.htm, June

[So+02] Sommer, R.; Feldmann, A.: NetFlow: Information Loss or Win? In: Proceedings of the 2nd ACM SIGCOMM and USENIX Internet Measurement Workshop (IMW2002), 2002.

[So+96] Sobirey, M.; Richter, B.; König, H.: The Intrusion Detection System AID – Architecture, and experiences in automated audit analysis. In: Proceedings of the IFIP TC6/TC11 International Conference on Communications and Multimedia Security, S. 278–290, Chapman & Hall, 1996.

[So99] Sobirey, M.: Datenschutzorientiertes Intrusion Detection. ISBN: 3-528-05704-1, Vieweg & Sohn, 1999, zugl. Dissertation, Fakultät Mathematik, Naturwissenschaften und Informatik, Brandenburgischen Technische Universität Cottbus, 1999.

[Sta03] Stallings, W.: Cryptography and and Network Security. Third Edition, ISBN: 0-13-111502-2, Pearson Education, 2003.

[Ste93] Stevens, W. R.: Advanced Programming in the UNIX Environment. ISBN: 0-201-56317-7, Addison Wesley, 1993.

[Stro97] Stroustrup, B.: Die C++-Programmiersprache. ISBN: 3-82731-296-5, Addison-Wesley, 1997.

[Sun02] Sun Microsystems, Inc.: Solaris 9 System Administration Guide: Security Services. In: Sun Microsystems, Inc.: Solaris 9 4/03 System Administrator Collection. Santa Clara, CA, 2002.
http://docs.sun.com/app/docs/doc/816-4883

[Sun97] Sun Microsystems, Inc.: SunSHIELD Basic Security Module Guide. In: Sun Microsystems, Inc.: Solaris 2.6 System Administrator Collection Vol. 1. Mountain View, CA, 1997, http://docs.sun.com/db/doc/802-5757/

[Tal95] Talarian, Corporation: RTie Inference Engine. In: Talarian Corporation (eds.): RTworks 3.5. Mountain View, CA, 1995.

[Tu99] Tung, B.: Common Intrusion Detection Framework. 1999. http://www.isi.edu/gost/cidf/

[Vi+00] Vigna, G.; Eckmann, S. T.; Kemmerer, R. A.: The STAT Tool Suite. In: Proceedings of DARPA Information Survivability Conference and Exposition (DISCEX) 2000, Vol. 2, S. 46–55, IEEE Press, 2000.

[Vi05] Persönliche Konversation mit Giovanni Vigna. Wien, Juli, 2005.

[Vig+00] Vigna, G.; Eckmann, S. T.; Kemmerer, R. A.: Attack Languages. In: Proceedings of the IEEE Information Survivability Workshop, Boston, MA, October 2000.

[Wo+00] Wolf, G.; Pfitzmann, A.: Charakteristika von Schutzzielen und Konsequenzen für Benutzungsschnittstellen. In: Informatik-Spektrum, Band 23, Nr. 3, S. 171–193, Springer Verlag, 2000.

[Zim+99] Zimmer, D.; Unland, R.: On the Semantics of Complex Events in Active Database Management Systems. In: Proceedings of the 15th International Conference on Data Engineering, S. 392–399, IEEE Computer Society Press, 1999.

[Zim98] Zimmer, D.: Ein Meta-Modell für die Definition der Semantik von komplexen Ereignissen in aktiven Datenbankmanagementsystemen. Dissertation, Universität Paderborn, ISBN: 3-8265-3744-0, Shaker-Verlag, Aachen, 1998.

Rücknahme oder Umtausch
nur mit ungeöffneter Datenträgerverpackung