Sibel Mutlu • Axel Schnell • Emine Yüksel

Intrusion Detection als ergänzender Sicherheitsmechanismus am Beispiel von UNIX

Diplomarbeit
Vorgelegt zur Begutachtung durch
Prof. Dr. Klaus Brunnstein
Prof. Dr. Simone Fischer-Hübner

12. März 2001

UNIVERSITÄT HAMBURG
Fachbereich Informatik
Arbeitsbereich Anwendungen der Informatik
in Geistes- und Naturwissenschaften

Hiermit erklären wir, dass wir die Diplomarbeit selbständig durchgeführt haben. Die einzelnen Abschnitte der Arbeit wurden jeweils von den in Anhang A genannten Autor(en) angefertigt. Für diese Arbeit haben wir keine anderen, als die angegebenen, Quellen und Hilfsmittel benutzt.

Hamburg, den 10. April 2001

Sibel Mutlu — Emine Yüksel — Axel Schnell

Inhaltsverzeichnis

0 EINLEITUNG **8**

1 ZIELE VON SICHERHEIT **12**

1.1 Sicherheitsanforderungen 12
1.1.1 Vertraulichkeit (Confidentiality) 13
1.1.2 Integrität (Integrity) 13
1.1.3 Verfügbarkeit(Availability) 14
1.2 Methoden zur Durchsetzung der Sicherheitsanforderungen 14
1.2.1 Mechanismen zur Durchsetzung der Vertraulichkeit 14
1.2.1.1 Zugriffskontrollen 15
1.2.1.2 Zugriffsmatrix 16
1.2.1.3 Authentifikation und Identifikation 21
1.2.2 Mechanismen zur Durchsetzung der Integrität 23
1.2.3 Mechanismen zur Durchsetzung von Verfügbarkeit 23
1.3 Sicherheitspolitik und Sicherheitsmodelle 24
1.3.1 Bell-LaPadula-Modell 26
1.3.2 Modellbeschreibung 26
1.4 Sicherheitsarchitekturen 28
1.4.1 Sicherheitskern und der Referenzmonitor 29
1.4.2 Trusted Computing Base (TCB) 31
1.4.2.1 Protokollierung (Auditing) 33
1.4.3 Kerberos 33
1.4.3.1 Kerberos-Architektur 35
1.4.3.2 Schwächen von Kerberos 37
1.4.4 Gesamtübersicht der behandelten Sicherheitsbegriffe 38
1.5 Sicherheitskriterien-Kataloge 39
1.5.1 TCSEC - Das *Orange Book* 39
1.5.1.1 Klasse-D-Systeme (Minimaler Schutz) 39
1.5.1.2 Klasse-C-Systeme: (Diskreter Schutz) 40
1.5.1.3 Klasse-B-Systeme (Mandatorischer Schutz) . . . 40
1.5.1.4 Klasse-A-Systeme (Verifiziertes Design) 44
1.5.2 TNI - *Red Book* 45
1.5.3 Die deutschen IT-Sicherheitskriterien-*Green Book* 47

1.5.4 ITSEC - *Die europäischen Kriterien* 52
1.5.5 CCITSE - Common Criteria 54

2 UNIX **57**
2.1 Historische Entwicklung von UNIX 57
2.2 Das UNIX-System 60
2.2.1 Der Kernel 61
2.2.2 Dateisystem in UNIX 62
2.2.2.1 Verzeichnisse 63
2.2.3 Die Komponenten des Dateisystems 64
2.2.4 Prozess 66
2.2.5 Benutzer-Identifikation 68
2.2.6 Zugangs- und Zugriffskontrollen 69
2.2.6.1 UNIX Login 69
2.2.6.2 UNIX Password Mechanism 70
2.2.6.3 ACLs und Capabilities in UNIX 70
2.3 Entwicklung von TCP/IP 75
2.3.1 Einführung in TCP/IP 76
2.3.2 TCP/IP-Grundlagen 80
2.3.2.1 IP-Internet Protocol 80
2.3.2.2 ICMP-Protokoll 87
2.3.2.3 TCP (Transmission Control Protocol) 90
2.3.2.4 UDP-User Datagram Protocol 93
2.4 Angriffe auf Computer und Netze 95
2.4.1 Hacker und Cracker 95
2.4.2 Kategorien von Systemschwächen 96
2.4.3 Kategorien von Angriffsschäden 97
2.4.4 Angriffsarten 98
2.4.5 Angriffsplanung 100
2.4.5.1 TCP Port Scan 100
2.4.5.2 UDP Port Scans 102
2.4.5.3 Suche nach verwertbarer Information 103
2.4.6 Angriffe auf der unteren Dienstebene 104
2.4.6.1 Angriffe auf ARP-Ebene 104
2.4.6.2 Angriffe auf IP-Ebene 105
2.4.6.3 Angriffe, die auf ICMP basieren 106
2.4.6.4 Spoofing 107
2.4.6.5 Source Routing : 109
2.4.6.6 Session Hijacking 109
2.4.6.7 Überfluten (Flooding) 110
2.4.7 Angriffe auf der mittleren Ebene 111
2.4.7.1 DNS-Angriffe 111
2.4.7.2 NIS 113

2.4.7.3 FTP 113
2.4.7.4 HTTP 114
2.4.7.5 X11 114
2.4.7.6 NFS-Network File System 115
2.4.8 Angriffe auf der oberen Ebene 116
2.4.8.1 Sendmail 116
2.4.8.2 Mail Spoofing 117
2.4.8.3 Finger Flooding 117
2.4.8.4 Verteilte, koordinierte Angriffe 117
2.4.9 Angriffe auf das Authentisierungssystem 118
2.4.9.1 Lauschangriff-Sniffing 118
2.4.9.2 Erraten von Passwörtern 118
2.4.10 Import von kompromittierendem Code 119
2.4.10.1 Viren 119
2.4.10.2 Trojanische Pferde 119
2.4.10.3 Würmer 120
2.4.11 Probleme im Zusammmenhang mit dem WWW 120
2.4.11.1 Java 121
2.4.11.2 Cookies 123
2.4.11.3 CGI 124
2.4.11.4 MIME 124
2.4.11.5 WWW Spoofing 125

3 WAS IST INTRUSION DETECTION 132
3.1 Entwicklung von Intrusion Detection Systemen 132
3.1.1 Projekte zu Intrusion Detection bei SRI 134
3.1.1.1 Studien zur Algorithmengewinnung 135
3.2 Charakterisierung von Intrusion Detection Systemen 137
3.2.1 Network-Based Intrusion Detection (NIDS) 140
3.2.1.1 Paket Sniffer und Netzwerk Monitore 140
3.2.1.2 Paket Sniffing und wechselnder Modus 141
3.2.1.3 Netzwerkbasierter Intrusion Detection als eine Weiterentwicklung des Paket Sniffers 141
3.2.2 Host-Based Intrusion Detection (HIDS) 143
3.2.2.1 Überwachung hereinkommender Verbindungen 144
3.2.2.2 Überwachung von Login-Aktivitäten 144
3.2.2.3 Überwachung von Root-Aktivitäten 144
3.2.2.4 Überwachung des Dateisystems 145
3.2.2.5 Software bei hostbasierter Intrusion Detection 146
3.2.3 Anomaly Detection (Behaviour-Based ID) 148
3.2.4 Misuse Detection (Knowledge-Based ID) 149
3.3 Dorothy Denning‘s Intrusion Detection Modell 151
3.3.1 Modellbeschreibung 151

3.3.2 Grundbeziehung zwischen Angriffstypen und Verhaltensanomalien . . . 157
3.3.3 Analyse von Verhaltens-Anomalien . . . 158
3.3.3.1 Merkmalräume (Metrics) . . . 158
3.3.3.2 Statistische Modelle (statistical models) . . . 159
3.3.4 Profil Charakterisierung (Profile characterization) . . . 161
3.3.4.1 Login- und Sitzungsaktivitäts-Profile . . . 161
3.3.4.2 Befehls- und Programmausführungs-Profile . . . 164
3.3.4.3 Dateizugriffs-Profile . . . 166
3.3.4.4 Datenbankzugriffs-Profile . . . 168
3.3.4.5 Profil-Modelle und Aktivitäts-Profil-Management 168
3.4 IDES (Intrusion Detection Expert System) . . . 169
3.4.1 Überblick . . . 169
3.4.1.1 Unterschiede zwischen dem IDES-Prototypen und älteren Versionen . . . 170
3.4.2 IDES Design Modell . . . 175
3.4.2.1 Zielsystem Domain . . . 176
3.4.2.2 Realm-Schnittstelle . . . 176
3.4.2.3 IDES Prozessor . . . 176
3.4.2.4 Benutzer-Schnittstelle . . . 177
3.4.3 Entwurf der statistischen Komponente . . . 178
3.4.3.1 Die funktionelle Architektur . . . 178
3.5 MIDAS - The Multics Intrusion Detection and Alerting System . 182
3.5.1 Regeln . . . 182
3.5.2 MIDAS Verfahren . . . 183
3.6 ID-Systeme der neueren Generation . . . 185
3.6.1 NIDES . . . 185
3.6.2 NIDES Beta-Version . . . 186
3.6.2.1 Erweiterungen der aktuallisierten Beta-Version . 187
3.6.3 NIDES - Architektur . . . 188
3.6.3.1 Agend, Agen und Arpool . . . 189
3.6.3.2 Statistische Analyse . . . 189
3.6.3.3 Regelbasierte Analyse . . . 190
3.6.3.4 Der Resolver . . . 190
3.6.4 IDA (Intrusion Detection and Avoidance System) . . . 192
3.6.4.1 Die Architektur von IDA . . . 194
3.6.5 Intrusion Detection In Our Time (IDIOT) . . . 198
3.6.5.1 Das Modell im Überblick . . . 199
3.6.5.2 Ein Beispiel . . . 200
3.6.6 EMERALD . . . 200
3.6.6.1 Der EMERALD-Monitor . . . 203
3.6.6.2 Anomalieerkennung . . . 203
3.6.6.3 Anomalieerkennung mit EMERALD . . . 205

3.6.6.4 Signaturbasierte Analyse 207
3.6.6.5 Die Wissensbasis 208
3.6.6.6 Der Einfluss früherer Systeme auf EMERALD . . 209
3.6.6.7 Die Beobachtung von TCP/IP Strömen durch EMERALD 210
3.6.7 ASAX 211
3.6.7.1 Die Geschichte von ASAX 211
3.6.7.2 Überblick 211
3.6.7.3 ASAX und UNIX 213
3.6.7.4 Die Instantiierung von ASAX 220
3.6.7.5 Die Benutzung von ASAX 222

4 ANFORDERUNGEN AN EIN ID-SYSTEM 224
4.1 Allgemeine Anforderungen 224
4.2 Ein Klassifikationsschema für vorhandene ID-Systeme 226
4.2.0.6 Untersuchungskriterien 227
4.2.1 EMERALD 235
4.2.2 IDES 238
4.2.3 NIDES 241
4.3 Ein Prototyp für eine ID-Funktionalitätsklasse 244
4.3.1 ID-F1 Sicherheitsaudit 245
4.3.2 ID-F2 Wartungsunterstützung 246
4.3.3 ID-F3 Identifizierung und Authentifizierung 246
4.3.4 ID-F4 Schutz der Sicherheitsfunktionen 247
4.3.5 ID-F5 Vertraulichkeit (Privacy) 247
4.3.6 ID-F6 Verfügbarkeit der Dienste 248
4.3.7 Vertrauenswürdigkeit (Assurance) 248
4.3.8 Bemerkungen 248

5 ENTWICKLUNGSSPRACHEN FÜR IDS REGELN 250
5.1 Das Beobachten und Erkennen von Missbrauch durch Expertensysteme 250
5.1.1 Einführung in Expertensysteme 250
5.1.2 Mathematische Grundlagen der Inferenz 251
5.1.2.1 Eine kleine Einführung in die Logik 251
5.1.2.2 Modus-Ponens 252
5.1.2.3 Hornformeln und Hornklauseln 254
5.1.3 Evaluation 254
5.1.3.1 Die Leistung 255
5.1.3.2 Die Konsistenz der Regelbasis 255
5.1.3.3 Die Falschpositiv- und die Falschnegativrate . . . 258
5.1.4 Backwardchaining 258
5.1.5 Regeln als Petrinetze 259

5.1.5.1 Formale Definition von Netzen 259
5.1.5.2 Die Konstruktion eines Regelnetzes 261
5.1.5.3 Gefärbte Petrinetze 264
5.2 Production-Based Expert System Toolset (P-BEST) 266
5.2.1 Die Bestandtteile von P-BEST im Überblick 266
5.2.2 Die P-BEST Sprache 266
5.2.2.1 Der Mustertyp (ptype) 267
5.2.2.2 Regeln oder Produktionen 268
5.2.2.3 Benutzerdefinierte Funktionen und Variablen. . . 273
5.3 Die Sprache von ASAX: RUSSEL 274
5.3.1 Die wichtigsten Merkmale der Sprache RUSSEL 274
5.3.2 Module 274
5.3.3 Datentypen 275
5.3.4 Deklarationen 276
5.3.4.1 Die Deklaration globaler Variablen 276
5.3.4.2 Regeldeklaration 276
5.3.4.3 Resumee 281
5.4 Vergleich von P-BEST und RUSSEL 282
5.4.1 Die Unterschiede 282
5.4.1.1 Einleitung 282
5.4.1.2 Fakten 282
5.4.1.3 Die Anweisungen 283
5.4.1.4 Regeln 284
5.4.1.5 Konzepte 284
5.4.2 Die Gemeinsamkeiten 285
5.4.2.1 Die Mächtigkeit beider Sprachen 285
5.4.2.2 Weitere Gemeinsamkeiten 286

6 UNIX-NETZWERKSICHERHEITSARCHITEKTUR 287
6.1 Überblick 287
6.1.1 Ein Sicherheitsschichtenmodell 287
6.1.2 Ergänzung des Modells um IDS-Komponenten 289
6.1.2.1 IDS in Schicht 5 290
6.1.2.2 IDS in Schicht 4 291
6.1.2.3 IDS in Schicht 3 292
6.1.2.4 IDS in Schicht 2 292

A Zuordnung der Autoren zu den einzelnen Abschnitten 294

Kapitel 0

EINLEITUNG

Das Schlagwort “Informationsgesellschaft” drückt aus, dass neben den traditionellen Wirtschaftsgütern die *Information* als eine neue Art von Wirtschaftsgut eine wachsende Bedeutung hat. *Information* muss erst einmal gewonnen, dann aufbewahrt, und bei Bedarf transportiert werden. *Information* gewinnt, wie ein traditionelles Wirtschaftsgut, bei jedem Verarbeitungsschritt an Wert. Damit der Wert der *Information* erhalten bleibt, muss die *Information* geschützt werden. In [dD82] wird *Information* folgendermaßen definiert :

Definition 1 (Information) *Information ist eine als räumliche oder zeitliche Folge physikalischer Signale,* die mit bestimmten Wahrscheinlichkeiten oder Häufigkeiten auftreten*, sich zusammensetzende Mitteilung, die beim Empfänger ein bestimmtes [Denk]verhalten bewirkt.*

Die Entwicklung der Informationstechnologien, insbesondere die Expansion des Internets, hat dazu beigetragen, dass *Information* inzwischen ein Gut darstellt, mit dem in allen Bereichen der Gesellschaft, vor allem der Wirtschaft, gehandelt wird. Von dem Schutz der Information hängt somit auch der Schutz vieler Wirtschaftsbereiche ab. Die Entwicklung von Technologien, die dem Schutz der Information dienen, fand nicht proportional zum Wertegewinn der *Information* statt. Vielmehr wird versucht durch nachträgliche Maßnahmen diese Differenz auszugleichen, was in vielen Fällen keine ausreichende Lösung bietet. Festzustellen ist, dass „Sicherheit“ schon beim Entwurf der Systeme oder Programme, als ein Entwurfsziel, festgelegt werden muss, um den Schutz der *Information* zu gewährleisten. Dies stellt aber bei der Entwicklung vieler Produkte nicht den Normalfall dar. Sicherheitsaspekte gewinnen erst nach Auftreten von sicherheitsrelevanten Problemen an Bedeutung. Diese Nachlässigkeit ist die Basis eines Marktes, in dem Unternehmen Sicherheitsprodukte anbieten, die für den nachhaltigen Schutz sorgen sollen. Firewalls gehören zu solchen Software-Lösungen. Sie stellen aber passive Komponenten dar, die im Bereich der Netzwerksicherheit dazu eingesetzt werden, festgelegte Zugriffskontrollen durchzusetzen. In vielen Forschungsinstituten begann man in den 80’ern mit der Entwicklung von aktiven

Komponenten, sog. Intrusion Detection-Systemen, die über die Fähigkeiten der Firewalls hinausgehend, versuchen sollten, durch Schlussfolgerungsfähigkeit, Einbrüche zu entdecken, wenn möglich in Echtzeit. Hauptziel dieser Diplomarbeit ist es, diese Systeme vorzustellen, zu untersuchen und ihre Einsatzmöglichkeiten aufzuzeigen.

In **Kapitel 1** werden zunächst Ziele von Sicherheit behandelt. In Abschnitt 1.1 werden Sicherheitsanforderungen, die zum Schutz der *Information* erfüllt werden müssen, vorgestellt. Zur Erfüllung dieser Anforderungen existieren verschiedene traditionelle Mechanismen. In Abschnitt 1.2 werden diese Mechanismen, die meist auf Zugriffskontrollen basieren, ausführlich behandelt. *Information* lässt sich in verschiedene Schutzebenen einteilen. In einem Unternehmen muss z. B. entschieden werden, welche Art von Informationen einem höheren Schutzbedarf unterliegen und welche nicht. Es muss auch festgelegt werden, wie der Schutz, sowohl technisch als auch organisatorisch, tragbar und machbar ist. Dies macht die Erstellung von Sicherheitspolitiken und Sicherheitsmodellen notwendig, für die im Abschnitt 1.3 Beispiele gezeigt werden. In Abschnitt 1.4 werden Sicherheitsarchitekturen vorgestellt, die für den Bau der sicherheitsrelevanten Komponenten der Systeme aufgestellt werden, und die für die Durchsetzung der Sicherheitspolitik und Implementierung der Sicherheitsmodelle den nötigen Rahmen bilden. Als traditionelle Konzepte für monolithische Systeme wurden in Abschnitt 1.4.1 und in 1.4.2 der Referenzmonitor, die Trusted Computing Base (TCB) und der Sicherheitskern vorgestellt. Im Abschnitt 1.4.3 wird schließlich auf die Sicherheitsarchitektur Kerberos eingegangen, die den Anforderungen von verteilten Systemen Rechnung trägt. Um die Sicherheit von Informationssystemen beurteilen zu können, wurden verschiedene nationale und internationale Richtlinien entwickelt. Diese sollen

a) den Erwerbern eines Produktes, bei dessen Beurteilung in Bezug auf Sicherheit, Hilfe leisten,

b) den Entwicklern einen Rahmen für den Entwurf und die Entwicklung der sicherheitsrelevanten Komponenten eines Produktes bieten.

In Abschnitt 1.5 werden verschiedene Sicherheitskriterien-Kataloge, angefangen beim *Orange Book*, vorgestellt.

Bei der Entwicklung vieler Intrusion Detection Systeme spielte das Betriebssystem UNIX eine entscheidende Rolle. Dies lässt sich darauf zurückführen, dass UNIX im Bereich der Forschung ein weitverbreitetes Betriebssystem war und ist. Viele Entwicklungen im Bereich der Kommunikationstechnologien wurden auf UNIX aufgebaut. Die Vernetzung der Systeme, und die dazu benötigten Kommunikationsprotokolle, wurden auf UNIX-Systemen erprobt und etablierten sich später als quasi Standard. In **Kapitel 2** wird daher ausführlich auf dieses Betriebssystem eingegangen. Abschnitt 2.1 zeigt die historische Entwicklung von UNIX. Diesem folgt in Abschnitt 2.2 der Aufbau der einzelnen Komponenten eines UNIX-Systems, mit Schwerpunkt auf sicherheitsrelevanten Aspekten, wie Zugangs- und Zugriffskontrolle.

Der Informationsaustausch spielt, wegen der verteilten Aufbewahrung der Informationen in Netzwerken, beim Verarbeitungsprozess der Information, eine große Rolle. Die Kommunikationsprotokolle spielen daher eine wichtige Rolle im Hinblick auf den Schutz der zu übertragenden Information. Daher wurde dem gängigsten Kommunikationsprotokoll TCP/IP Abschnitt 2.3 gewidmet.

Bevor man zeigen kann, **wie** Schutz gewährleistet wird, muss dargestellt werden, **wovor** Systeme geschützt werden müssen. In Abschnitt 2.4 werden daher die gängigsten Angriffe und Angriffsmethoden, einschließlich der sie begünstigenden Systemschwächen und die aus den Angriffen resultierenden Systemschäden, vorgestellt. Der Bezug der Angriffe zu dem Betriebssystem UNIX besteht darin, dass die meisten Protokolle und Dienste der UNIX-Welt entstammen.

Die Vielseitigkeit der in **Kapitel 2** dargestellten Angriffe und Angriffsmethoden zeigt die Notwendigkeit von Vorsorge-Werkzeugen, die neben den üblichen Schutzvorkehrungen eingesetzt werden können. **Kapitel 3** behandelt einbruchsentdeckende Systeme, angefangen mit der historischen Entwicklung dieser Systeme, dargelegt in Abschnitt 3.1. In Abschnitt 3.2 werden ID-Systeme, ihren Eigenschaften entsprechend, charakterisiert und verschiedene Methoden zur Einbruchsentdeckung erläutert. Ein Intrusion Detection Modell, das zwischen 1985 und 1987 von Dorothy Denning beim SRI entwickelt wurde, beeinflusste viele nachfolgende ID-Systeme. In Abschnitt 3.3 wird dieses Modell ausführlich behandelt. Diesem folgt, in Abschnitt 3.4, die Vorstellung des Systems IDES. In Abschnitt 3.6.1 werden ID-Systeme der „neueren“ Generation beschrieben. NIDES zählt zu diesen Systemen und baut weitgehend auf IDES auf. Die Besonderheiten von einem NIDES-Prototypen, einschließlich der Veränderungen gegenüber IDES und älteren NIDES-Prototypen, wird in Abschnitt 3.6.2 behandelt. Ein weiteres ID-System, das im Gegensatz zu den vorher erwähnten Systemen, einen modellbasierten Ansatz bei der Missbrauchserkennung benutzt, ist IDIOT. Es wird in Abschnitt 3.6.5 behandelt. Das ID-System EMERALD vereinigt die Eigenschaften von IDES und NIDES und erweitert diese um die Fähigkeit der verteilten Analyse. Bei EMERALD wurde der Subjektbegriff allgemeiner gefasst, woraus eine größere Flexibilität, in Bezug auf die zu überwachenden Ziele, resultierte. EMERALD wird in Abschnitt 3.6.6 ausführlich behandelt. Die bisher erwähnten ID-Systeme entstammen, bis auf IDIOT, Forschungsprojekten, die beim SRI durchgeführt wurden. Ein ID-System, das europäischen Ursprungs ist, und an der Universität Namur, unter Beteiligung der Universität Hamburg, entwickelt wurde, ist ASAX und wird in Abschnitt 3.6.7 vorgestellt.

Ein wichtiger Aspekt beim praktischen Einsatz von ID-Systemen, ist die Effektivität dieser Systeme. Damit die ID-Systeme die gewünschten Leistungen erbringen, müssen sie bestimmten Anforderungen genügen. Diese Anforderungen werden in **Kapitel 4** behandelt. Zunächst werden im Abschnitt 4.1 allgemeine Anforderungen gestellt und in 4.2 ein Klassifikationsschema für vorhandene ID-Systeme aufgestellt. Dieses Klassifikationsschema begründet sich auf die Untersuchungsergebnisse, die in [vHK98] dokumentiert wurden. In Betracht gezogen wurden in dieser Diplomarbeit nur die ID-Systeme IDES, NIDES und EMERALD. In Abschnitt 4.3 wird, in Anlehnung an [vHK98], der Vorschlag für eine ID-Funktionalitätsklasse gemacht.

In **Kapitel 3** wurden die Architektur der ID-Systeme vorgestellt, und die beiden Einbruchsentdeckungs-Methoden, Misuse Detection und Anomaly Detection, allgemein beschrieben. In **Kapitel 5** werden zunächst die Grundlagen der Inferenz Methode "Modus Ponens", die in Expertensystemen Anwendung findet, vorgestellt. Darauf aufbauend werden in 5.1.3 besondere Probleme bei der Erstellung von Regelbasen zur Missbrauchsentdeckung erläutert. In dem Abschnitt 5.1.5 wird aus den logischen Grundlagen der Inferenzmethode "Modus Ponens" ein Inferenzmodell mit Hilfe von B/E-Netzen, einer speziellen Art von Petrinetzen, entwickelt. Der Abschnitt 5.2 erklärt die Regelentwicklungssprache des Expertensystems P-BEST, welches in IDES, NIDES und EMERALD eingesetzt wird. Die Regelspezifizierungssprache RUSSEL, in der unter dem ID-System ASAX die Regelbasen erstellt werden, wird in Abschnitt 5.3 vorgestellt. Daran schließt sich ein Vergleich der beiden Sprachen P-BEST und RUSSEL in Abschnitt 5.4 an.

In **Kapitel 6** werden Vorschläge gemacht, **wo** in einem Netzwerk ein ID-System sinnvoll eingesetzt werden kann. Als Grundlage dazu dient ein Schichtenmodell, das in [Rei93] vorgeschlagen wurde. Dieses Modell teilt ein Netzwerk in verschiedene Abstraktionsebenen auf und besitzt als wesentliche Bestandteile, in Bezug auf Sicherheit, Firewalls. Dieses Schichtenmodell wurde von uns um ID-Systeme erweitert, um zeigen zu können, wie diese Systeme eine Firewall sinnvoll ergänzen können. Dies wird im Abschnitt 6.1.2 dargelegt.

Kapitel 1

ZIELE VON SICHERHEIT

1.1 Sicherheitsanforderungen

Die Abhängigkeit der Gesellschaft von Informationstechnologien bringt es mit sich, dass die Auswirkungen von Sicherheitseinbrüchen unüberschaubare Schadensausmaße annehmen können. Insbesondere ist mit großen finanziellen Verlusten bei den betroffenen Wirtschaftsunternehmen zu rechnen. Aus diesem Grund ist beim Einsatz von Rechensystemen in Unternehmen ein hohes Maß an Sicherheit erforderlich.
Hauptziel der Sicherheit ist dabei die Wahrung der

- **Vertraulichkeit**
- **Integrität**
- **Verfügbarkeit**

eines Systems. In [Kü96] heißt es weiterhin :

> „ *Die* **Sicherheit eines Systems** *ist seine Eigenschaft, Schutz vor* <u>*als relevant erachteten*</u> *Bedrohungen der Vertraulichkeit, Integrität und Verfügbarkeit der verarbeiteten Informationen zu gewährleisten.* “

Vertraulichkeit, Integrität und Verfügbarkeit können auch als Sicherheitsanforderungen bezeichnet werden. In folgenden Abschnitten wird eine detaillierte Beschreibung der Sicherheitsanforderungen und Methoden zu deren Durchsetzung vorgenommen. Diese Beschreibung wird die Vorstellung eines Sicherheitmodelles, unter dem Namen *Bell-LaPadula*, folgen, welches die Vertraulichkeit und Integrität eines Systems (formal) durchsetzt. Anschließend wird unter dem Aspekt der Sicherheitsarchitektur der *Referenzmonitor* und der *Trusted Computing Base (TCB)* vorgestellt.

1.1.1 Vertraulichkeit (Confidentiality)

Vertraulichkeit bedeutet, dass Informationen vor unberechtigtem Einblick geschützt werden. Im Zusammenhang mit Netzwerken, insbesondere dem Internet bedeutet dies Folgendes:

Über das Internet übertragene Daten, Programme oder Wissen dürfen nicht durch potentielle Angreifer lesbar bzw. interpretierbar sein. Ferner dürfen diese Daten nicht verloren gehen oder zerstört werden. Die Vertraulichkeit der Kommunikation kann für alle Daten (Nutz- und Protokolldaten), für einen Teil der Daten, für Informationen über die Datenmenge oder die Kommunikationsbeziehungen an sich, erforderlich sein.
Der Schutz durch die Sicherung des Kommunikationsweges reicht hierbei nicht aus; Auch der Schutz inhaltlicher Aspekte muss gewährleistet werden.

1.1.2 Integrität (Integrity)

Integrität bedeutet, dass die Daten

> „ die korrekte physikalische und semantische Repräsentation von Informationen enthalten“[FH92]

Im Bereich der Netzsicherheit bedeutet dies, dass die Unversehrtheit der Daten, bzw. Nachrichten, beim Vorgang des Übertragens gewährleistet werden muss. Eine Nachricht, die verschickt wird, muss beim Erreichen des Empfängers genau die Form aufweisen, die der Absender produziert hat, d.h., dass weder die unabsichtliche noch die mutwillige Veränderung erlaubt sein darf. Die zufällige Veränderung der Daten, z.B. durch fehlerhafte Datenträger oder durch Übertragungsfehler, ist ebenso zu vermeiden.
Kurz gefasst könnte man sagen, dass ein sicheres Computersystem die Integrität seiner Daten sicherstellen muss, indem es diese vor

- **Zerstörung**
- **Modifikation**
- **Manipulation**

schützt.

Integritätsverluste können z.B. durch Viren, Würmer oder Trojanische Pferde verursacht werden.

1.1.3 Verfügbarkeit(Availability)

Der Schutz vor unbefugten Versuchen zur Vorenthaltung von Informationen oder Rechnerressourcen fällt in den Bereich der Verfügbarkeit.
Die Hard- und Software des Systems spielen hierbei eine wichtige Rolle. Je sensibler die Daten sind, desto stabiler muss das Betriebssystem und die installierte Software sein.
Um die Verfügbarkeit von Daten bzw., Diensten, sicherzustellen, müssen gewisse Anforderungen erfüllt werden. Dazu zählen folgende Bedingungen:

- hohe Fehlertoleranz des Systems,
- problemloses Backup der Daten,
- die Möglichkeit auf Reservesysteme zurückgreifen zu können,
- schnelle Herstellung des alten Betriebszustandes nach einem Ausfall,
- sorgfältige Administration des Systems.

1.2 Methoden zur Durchsetzung der Sicherheitsanforderungen

Zur Gewährleistung der oben erwähnten Sicherheitsanforderungen können verschiedene Mechenismen eingesetzt werden. Ein *Sicherheitsmechanismus* ist nach [Kü96]

> *„ ein algorithmisch faßbares Lösungsprinzip, das zur Abdeckung von Sicherheitsanforderungen an ein IT-System gesetzt wird. “*

In folgenden Abschnitten werden verschiedene Beispiele für solche Mechanismen vorgestellt.

1.2.1 Mechanismen zur Durchsetzung der Vertraulichkeit

Zur Durchsetzung oder Gewährleistung der Sicherheitsanforderungen *Vertraulichkeit* und *Integrität* können Mechanismen oder Grundfunktionen verwendet werden, die meist auf Zugriffskontrollen basieren. Die für die Durchsetzung der Verfügbarkeit benutzten Grundfunktionen und Mechanismen unterscheiden sich hiervon.

1.2.1.1 Zugriffskontrollen

Die Zugriffskontrolle ist die traditionelle Methode unter den Sicherheitsmechanismen. Zugriffskontrollen überprüfen die Zugriffe von Subjekten auf Objekte, d.h., ein Zugriffskontrollsystem unterscheidet, ob ein Subjekt (Benutzer, Prozess) mit einer Operation (read,write, delete) auf ein Objekt (Daten, Programm) zugreifen kann oder nicht [Bru].

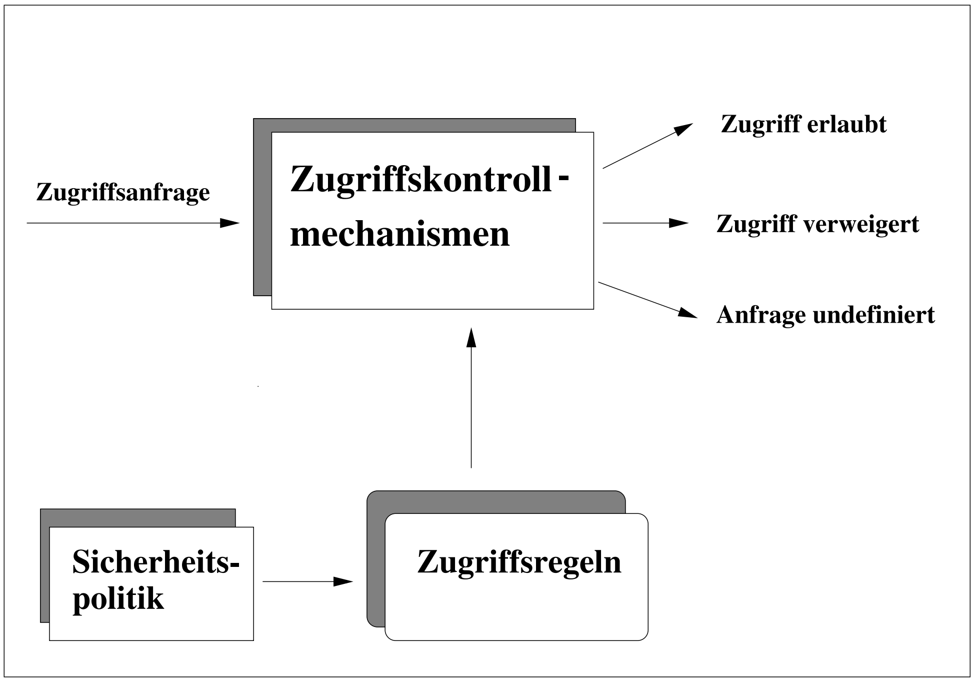

Abbildung 1.1: Ein allgemeines Zugriffskontrollsystem [Bru]

Ein Zugriffskontrollsystem besteht aus [Bru]:

- Zugriffsregeln, die die System-Sicherheitspolitik implementieren,
- Zugriffskontrollmechanismen, die jede Anfrage auf Einhaltung der Zugriffsregeln überprüfen.

Man kann zwei verschiedene Arten von Zugriffskontrolle unterscheiden :

- *Diskrete Zugriffskontrollen* (discretionary access controls, DAC),
- *Mandatorische Zugriffskontrollen* (mandatory access controls, MAC)

Da die Unterschiede zwischen den beiden Arten der Zugriffskontrolle für die nachfolgenden Ausführungen von Bedeutung sind, werden sie im Folgenden beschrieben.

Diskrete Zugriffskontrolle (DAC)

Nach [FH92] macht die diskrete Zugriffskontrolle, die Einteilung oder Verwehrung eines Zugriffs auf ein Objekt von der aktuellen Subjekt-ID, und nicht von den Sicherheitslevels abhängig (vgl. Abschnitt 1.3.1). Damit können Benutzer Zugriffsrechte an ihre Objekte weitergeben oder ihnen entziehen. DAC kann durch eine Zugriffsmatrix modelliert werden. Wegen der großen Bedeutung wird in 1.2.1.2 näher auf die Zugriffsmatrix eingegangen.

Mandatorische Zugriffskontrolle (MAC)

Nach [FH92] überprüfen mandatorische Zugriffskontrollen Zugriffe durch einen Vergleich von *Sicherheitsattributen*, die Subjekten und Objekten administrativ von einem Sicherheitsadministrator, oder automatisch, fest zugeordnet werden, und von Benutzern oder Benutzerprogrammen nicht verändert werden können. Sicherheitsattribute sind partiell geordnete Sicherheitslevels (stammen aus einer militärischen Sicherheitspolitik). Durch die Einhaltung dieser Sicherheitslevels können bestimmte Angriffe (z.B. trojanische Pferde) verhindert werden. In [FH92] wird weiterhin aufgeführt, dass die meisten mandatorischen Zugriffskontrollen zur Erlangung von *Multilevel Security* (MLS) das *Bell-LaPadula-Modell* umsetzen. Auf dieses Modell wird in 1.3.1 ausführlicher eingegangen.
Mandatorische Zugriffskontrollen sind in sämtlichen Betriebssystemen realisiert, die den *Sicherheitskern-Ansatz* (security kernel based systems) verfolgen [FH92].

1.2.1.2 Zugriffsmatrix

Die Zugriffsmatrix ist ein Konzept zur Formulierung von Zugriffsrechten. Der Eintrag *rw* in der Abb.1.2 bedeutet beispielsweise, dass das Subjekt **S2** auf das Objekt **O2** lesend und schreibend zugreifen kann. Eine Zugriffsmatrix kann durch *Capabilities* oder *Zugriffskontrolllisten (ACLs)* implementiert werden.

Die **Zugriffskontrollliste (ACL)** beschreibt die Rechte von Subjekten an Objekten aus der *Objektsicht*. Eine ACL ist genau mit einem Objekt assoziiert und beschreibt sämtliche Rechte von Subjekten bzgl. dieses assoziierten Objektes. Die ACL ist als eine Tabelle aufgebaut, in der ein Eintrag aus einem Tupel der Form

< *Subjekt-Identifikation Si, Rechte des Si* >

besteht. Eine Variante dazu wäre

< *GruppenId-Gi, Rechte von Gi* >

Bei der **Veränderung von Rechten** gibt es verschiedene Möglichkeiten

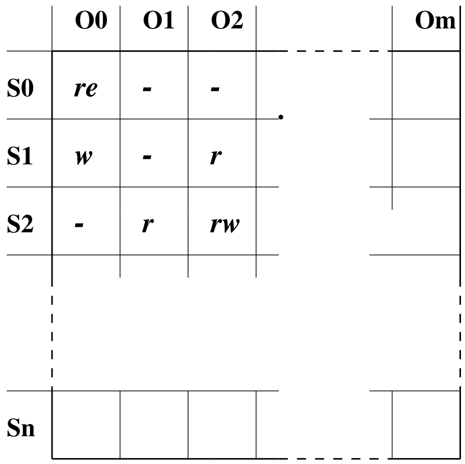

Abbildung 1.2: Zugriffsmatrix [Kü96]

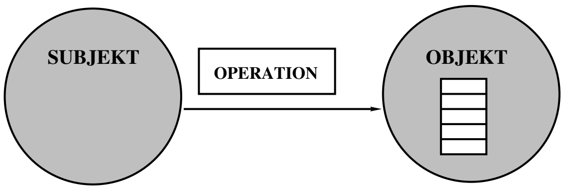

Abbildung 1.3: Abstrakte Modellierung des Zugriffes [Kü96]

[Kü96] :

- Recht auf Veränderung ist an den Eigentümer des Objektes (Erzeuger) gebunden.
- Recht auf Veränderung ist durch die ACL selbst definiert.
- Recht auf Veränderung ist Dritten vorbehalten (mandatorische Sicherheitspolitik)

Die **Rechteprüfung** wird bei jedem Zugriffsversuch eines Subjektes durchgeführt. Dabei werden die Subjektattribute *(Subjekt-Id, Gruppen-Id, neutrale Kennung)* in Bezug auf den gewünschten Zugriff geprüft. ACLs werden durch Referenzmonitore, auf die später eingegangen wird, implementiert.

Capabilities beschreiben Rechte von Subjekten an Objekten aus der *Subjektsicht.* Capabilities sind genau mit einem Subjekt assoziiert und beschreiben sämtliche Rechte dieses assoziierten Subjekts an Objekten. Capabilities sind, wie die ACLs, in Tabellen organisiert, wobei ein Eintrag ein Tupel der Form

< *Objekt-Id Oi, Rechte an Oi* >

aufweist.

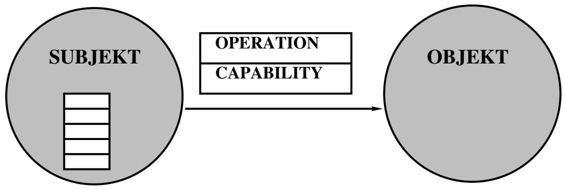

Abbildung 1.4: Abstrakte Modellierung des Zugriffes [Kü96]

Bei der **Veränderung von Rechten**, d.h. die Ausgabe (A), Weitergabe (W) und Rückruf (R) von Capabilities, gibt es verschiedene Möglichkeiten [Kü96] :

- A/W/R ist an den Eigentümer des Objektes (Erzeuger) gebunden.
- A/W/R ist durch Capability selbst gesteuert.
- A/W/R erfolgt durch dritte Instanz (mandatorische Sicherheitspolitik)

Die **Rechteprüfung** erfolgt bei jedem Zugriffsversuch eines Subjektes, wobei die Rechte in den Capabilities des Subjektes gegen den geforderten Zugriff geprüft werden. Die Implementierung der Capabilities erfolgt, genau wie bei ACLs, durch den Referenzmonitor, welcher ein Architekturprinzip bei der Konstruktion sicherer Betriebssysteme darstellt.

Schwächen von Capabilities und ACLs :

Capabilities und ACLs weisen einige Gefahren im Hinblick auf die Sicherheit auf. Nicht-vertrauenswürdige Subjekte können

a) ihre Capabilities modifizieren,

b) ihre Capabilities an Dritte weitergeben, die ihrerseits die Capabilities modifizieren können, oder

c) fremde Capabilities, unter Fälschung der eigenen Identifikation, verwenden.

Zur Vermeidung solcher Missbräuche könnten die Sicherheitsmechanismen bei *monolithischen Systemen* in vertrauenswürdiger Betriebssystemkerne gekapselt werden. Damit wäre nach [Kü96] die Modifikation und die Weitergabe der Capabilities, sowie die Identifikation von Subjekten, unter vertrauenswürdige Kontrolle.

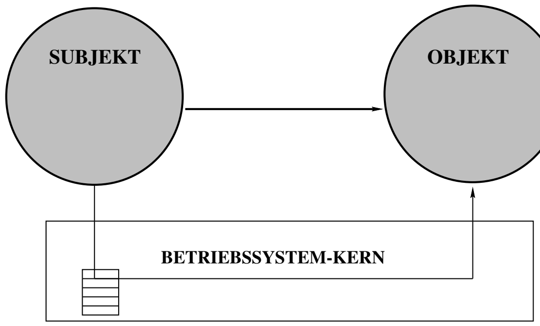

Abbildung 1.5: Kapselung der Sicherheitsmechanismen in den BS-Kern (In Bezug auf Capabilities). [Kü96]

Bei *verteilten Systemen* (räumliche und administrative Verteilung) ist die Kapselung der Sicherheitsmechanismen in zentraler Instanz i.A. nicht möglich. Missbrauchsgefahr entsteht hierbei durch die Architektur von verteilten Systemen. In [Kü96] wird dies durch die folgenden Feststellungen begründet :

- Die Aufbewahrung der Capabilities und die Identifikation von Subjekten wird auf dem Subjekt-System durchgeführt.
- Der Transport der Capability und der Subjektidentifikation findet über das Kommunikationssystem statt.
- Die Auswertung der Capabilities und der Subjektidentifikation wird durch das Objekt-System durchgeführt.

Diese Art der Aufgabenteilung, bedingt durch den Aufbau eines verteilten Systems, ermöglicht die, weiter oben erwähnten, Missbräuche (a), (b) und (c) durch ein nicht-vertrauenswürdiges Subjekt- und Kommunikationssystem.

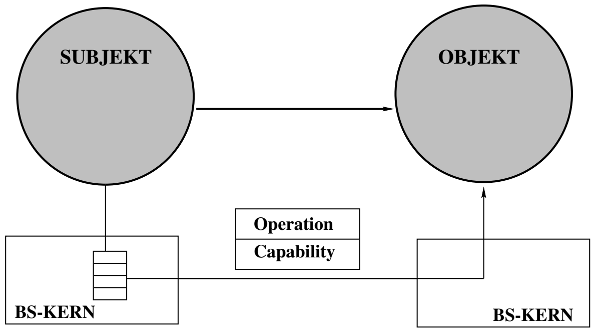

Abbildung 1.6: Sicherheitsmechanismen in verteilten Systemen (In Bezug auf Capabilities). [Kü96]

Zur Vermeidung dieser Missbräuche können die Capabilities kryptographisch versiegelt werden, so dass der Aussteller der Capability feststellbar und Veränderungen bemerkbar sind. Es können auch Authentifikations- und Autorisierungsarchitekturen eingesetzt werden, die im Gegensatz zu traditionellen Sicherheitsarchitekturen, wie z.B. TCB (vgl. Abschnitt 1.4.2), auf verteilte Systeme zugeschnitten sind. Ein Beispiel für eine solche Architektur ist Kerberos und wird in Abschnitt 1.4.3 vorgestellt.

Auch die ACLs weisen ähnliche Schwächen wie die Capabilities auf, die Gefahren der Fälschung mit dem Ziel in den Besitz von nicht zustehenden Rechten zu gelangen, ermöglichen (siehe [Kü96]) :

(a) Nicht-vertrauenswürdige Objekte modifizieren die ACL-Einträge von Subjekten

(b) Trifft nicht zu

(c) Verwendung fremder ACL-Einträge durch das Subjekt durch Fälschung der eigenen Identifikation

Genau wie bei den Capabilities können auch hier die Sicherheitsmechanismen in monolithischen Systemen in vertrauenswürdige Betriebssysteme gekapselt werden, wodurch, nach [Kü96], die

(a) Modifikation der ACLs und

(c) Identifikation von Subjekten

unter vertrauenswürdige Kontrolle gestellt wäre.

In verteilten Systemen existiert die vertrauenswürdige Kapselung der Sicherheitsmechanismen in zentraler Instanz i.A. nicht. In [Kü96] heißt es :

- Der Transport der Subjektidentifikation erfolgt über das Kommunikationssystem
- Die Auswertung der Subjektidentifikation wird durch das Objekt-System durchgeführt.

Die Gefahr hierbei ist, dass eine Fälschung gemäß (c) durch ein nicht-vertrauenswürdiges Subjekt- und Kommunikationssystem auftreten kann.

Maßnahmen zur Gefahrenvermeidung können durch den Einsatz von Authentifikationsarchitekturen erfolgen, die eine Instanz mit gemeinsamen Vertrauen für die Authentifikation von Subjekten darstellen. Nach [Kü96] benutzen Authentifikations- und Autorisierungsarchitekturen oftmals ACLs und Capabilities gemeinsam. Sie authentifizieren Subjekte und erteilen Capabilities auf der Basis lokaler ACLs (vgl. 1.4).

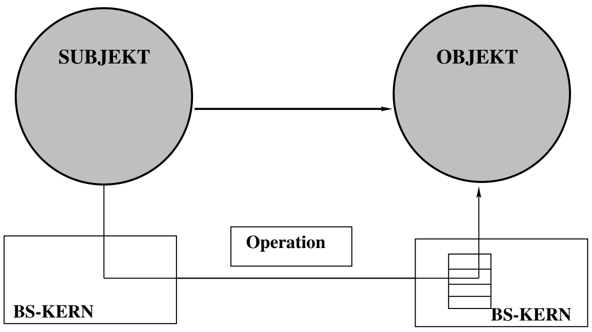

Abbildung 1.7: Sicherheitsmechanismen in verteilten Systemen (In Bezug auf ACLs). [Kü96]

1.2.1.3 Authentifikation und Identifikation

Authentifikation und Identifikation werden im Wesentlichen zur Durchsetzung der Vertraulichkeit benutzt. Es wird gefordert, dass das System den Benutzer identifizieren und authentisieren muss, bevor dieser das System benutzen und auf dem System Aktionen durchführen kann.
Die korrekte Identifizierung ist einerseits für die Festlegung nötig, ob der Benutzer überhaupt mit dem System in Kontakt treten darf, und andererseits, um dem Benutzer die ihm zustehenden Rechte zuzuweisen (Autorisierung). Wird z.B. eine Kommunikationsbeziehung in einem Netzwerk aufgebaut, so möchte jeder Teilnehmer sicherstellen, dass ausschließlich die gewünschten Partner daran beteiligt sind. Jeder Kommunikationspartner, der eine Angabe über seine Identität macht, muss auch einen Beweis der Richtigkeit der Authentizität übermitteln.
Diesen Prozess der Verifikation, ob jemand der ist, der er zu sein vorgibt, nennt man Authentisierung. Die Prüfung der Identität von Personen oder Instanzen ist unter anderem für ACLs und persönliche Capabilities nötig.
Eine Authentisierung kann u.a. durch folgende Mechanismen erfolgen:

- **Bei Personen :**
  - *Kennwortverfahren :* Nachweis der Identität durch Besitz eines (einfachen) Geheimnisses, z.B. Passwort, Personal Identification Number (PIN).
  - *Biometrische Verfahren :* Nachweis der Identität durch ein unfälschbares persönliches Merkmal, z.B. Fingerabdruck, Iris.
  - *Gegenstände :* Nachweis der Identität durch Besitz eines komplexen Geheimnisses, z.B. Magnetstreifen oder Chipkarten.
  - *Kryptographische Verfahren :* Asymmetrische Verfahren (z.B. RSA) oder symmetrische Verfahren (z.B. DES).
- **Bei Instanzen :** Kryptographische Verfahren

Autorisierung (Authorization) :
Mit Autorisierung wird der Vorgang bezeichnet, der notwendig ist, um herauszufinden, ob eine Anforderung für eine bestimmte Aktion zulässig ist. Dabei entscheidet die Autorität, die für die Verwaltung der Objekte (wie z.B. Dateien) zuständig ist , ob sie die Ausführung von gewünschten Aktionen gestattet oder nicht. Eine Autorität kann sein (vgl. [Bru]) :

- Der Systemadministrator d.h., die Autorisierung erfolgt zentral (z.B. bei mandatorischer Zugriffskontrolle).
- Verschiedene Benutzer, die Zugriffsrechte an ihre Objekte weitergeben oder entziehen d.h., die Autorisierung erfolgt dezentral (z.B. diskreter Zugriffskontrolle).
- Der Erzeuger eines Objektes wird zum Eigentümer des Objektes und kann Rechte am Objekt vergeben oder entziehen (Eigentümerprinzip)
- Eine zentrale Autorität vergibt Administratorrechte, z.B. für Teile eines Netzwerkes, an Sub-Administratoren d.h., die Autorisierung erfolgt hierarchisch dezentralisiert.
- Rechte werden nur in Zusammenarbeit einer Gruppe von Benutzern vergeben (4-Augen-Prinzip).

Um solche Autorisierungsentscheidungen korrekt treffen zu können, braucht die entsprechende Autorität ein gesichertes Wissen über die Identität des Anfragenstellers. Dieses Wissen wird der Autorität über die Authentisierung geliefert. Die Authentisierungsdaten sind dabei oft in einer Datenbank abgelegt, auf die die Autorität zugreifen kann.

Beweissicherung :
Bei der Beweissicherung geht es im Allgemeinen um den Nachweis einer stattgefundenen Kommunikation. Der Nachweis einer Kommunikation bezieht sich in der Regel auf das Senden, das Übertragen und den Empfang von Daten.

Man unterscheidet verschiedene Nachweisformen :

- Sendenachweis
- Einschreiben (der Sender weist nach, dass er Daten gesendet hat)
- Einschreiben mit Rückschein (der Sender kann nachweisen, dass der Empfänger die Daten unverändert erhalten hat)

Gleichzeitig können Mechanismen zur Datenintegrität, oder sogenannte Notariate, zum Kommunikationsnachweis herangezogen werden. Hierbei handelt es sich um einen vertrauenswürdigen Dritten, der alle Informationen besitzt, um die Datenintegrität und das Vorhandensein einer Kommunikationsbeziehung zu bestätigen.

1.2.2 Mechanismen zur Durchsetzung der Integrität

Zur Durchsetzung der Integrität gibt es verschiedene Verfahren, die man in statische und dynamische Verfahren unterteilen kann [FH92]:

Dynamische Verfahren : Die Einhaltung von Integritätsbedingungen wird bei jedem Zugriff eines Programms auf ein Objekt überprüft. Dies kann durch Zugriffskontrollen verwirklicht werden.

Statische Verfahren : Die Zugriffe eines Programms werden nur einmal, auf dessen Übereinstimmung mit Integritätsbedingungen, überprüft.

Zur dynamischen Durchsetzung von bestimmten Integritätsaspekten können gleiche oder ähnliche Mechanismen eingesetzt werden, wie jene, die zur Durchsetzung von Vertraulichkeit benutzt werden. Auch hierbei spielt die Zugriffskontrolle eine entscheidende Rolle. Für die Vertraulichkeit werden observierende Zugriffe, und für die Integrität modifizierende Zugriffe, kontrolliert.
Zur Durchsetzung der inneren und externen Konsistenz werden weitere, vor allem statische, Verfahren benötigt, die sich von den Verfahren zur Durchsetzung der Vertraulichkeit unterscheiden. Im Allgemeinen sind folgende Verfahren zur Durchsetzung von Integrität notwendig :

- Identifikation und Authentifikation,
- Zugriffskontrolle (diskrete und mandatorische),
- Protokollierung,
- Programm-Verifikation und Zertifizierung (für innere Bedrohungen).

1.2.3 Mechanismen zur Durchsetzung von Verfügbarkeit

Zur Durchsetzung der Verfügbarkeit lassen sich folgende Mechanismen und Grundfunktionen auflisten [FH92] :

- Mechanismen zur **Fehlerüberbrückung** (durch Hardware- und Softwareduplikate wird im Falle von Komponentenausfällen dafür gesorgt, dass der Rechenbetrieb weiterläuft).

- Mechanismen zur **Fehlerverhinderung** (dazu gehören auch Mechanismen zur Gewährleistung von Integrität). Zur Schnellerkennung von Fehlern kann beispielsweise fehlersensitive Hardware eingesetzt werden.

- Mechanismen zur **Verklemmungsvorsorge**, d.h., die Verklemmungsfreiheit von Systemen muss durch bestimmte Verfahren, wie *deadlock-prevention, deadlock-avoidance, deadlock-detection, deadlock-recovery*, gewährleistet werden.

1.3 Sicherheitspolitik und Sicherheitsmodelle

Um die Einhaltung der Sicherheitsanforderungen zu gewährleisten, sollte zunächst eine **Sicherheitspolitik** festgelegt werden. Eine Sicherheitspolitik hat nach [Kü96] folgende Ziele :

- Definition dessen, welche Aspekte der Sicherheit (Vertraulichkeit, Integrität, Verfügbarkeit) in einem gegebenen Kontext (Applikation, System, Organisation) relevant sind.

- Strategien, wie diese Sicherheit in dem gegebenen Kontext erreicht wird.

Sicherheitspolitik wird in [its91] (aus [Kü96]) folgendermaßen definiert :

- **Corporate Security Policy :** The set of laws, rules and practises that regulate how assets including sensitive information are managed, protected and distributed within a user organisation.

- **Technical Security Policy :** The set of laws, rules and practises regulating the processing of sensitive information and the use of resources by the hardware and software of an IT system or product.

- **System Security Policy :** The set of laws, rules and practises that regulate how sensitive information and other resources are managed, protected and distributed within a specific system.

Sicherheitspolitiken lassen sich klassifizieren in ([Kü96]),

- **mandatorische Politiken :** Zugriffsregeln, die für alle Subjekte eines Systems zwingend sind und i.d.R. auf allgemeinen Objektattributen basieren,

- **diskrete Politiken :** Zugriffsregeln, die durch beliebige, auch unprivilegierte, Benutzer aufgestellt werden können (z.B. dem Eigentümer eines Objektes), und i.d.R. auf den Identitäten der Subjekte und Objekte basieren.

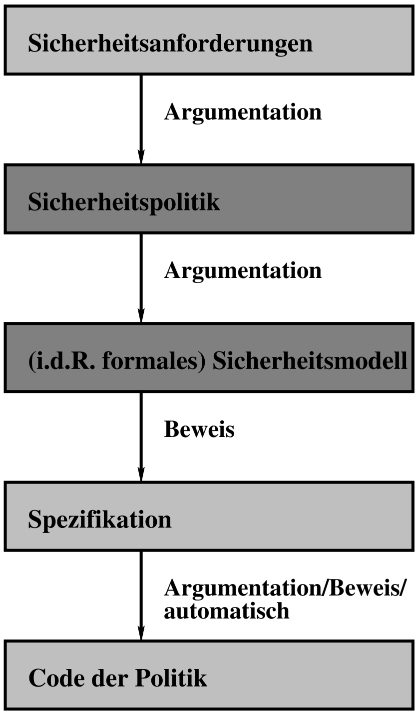

Abbildung 1.8: Von der Sicherheitspolitik zur Software-Realisierung [Kü96]

Der Erfolg bei der Durchsetzung einer hochgradigen Sicherheit in einem System, basiert auf einer großen Sorgfalt bei der Entwicklung und Implementierung der Sicherheitskontrollen. Diese Abhängigkeit zwischen erstrebter Sicherheit und den Kontrollen zu ihrer Einhaltung, führte zur Entwicklung von **Sicherheitsmodellen**, die Beschreibungsmittel darstellen, mit denen ein Bild der Realität widergespiegelt werden kann.
In [Kü96] wird Sicherheitsmodell folgendermaßen definiert :

> *„ Ein Sicherheitsmodell ist eine abstrakte Beschreibung der, nach der zugrundeliegenden Sicherheitspolitik für wesentlich gehaltenen, Aspekte der Sicherheit eines IT-Systems, wobei die als nicht sicherheitsrelevant geltenden Aspekte unterdrückt werden.“*

In [FH92] wird Sicherheitsmodell, iin Bezug auf die Entwicklung eines sicheren Betriebssystems, wie folgt definiert :

> *„Um ein Betriebssystem zu entwickeln, das Sicherheit gewähren soll, muss zunächst eine Sicherheitsstrategie(security policy) aufgestellt werden, die festlegt, welches bestimmte Subjekt welche Art von Zugriff auf ein bestimmtes Objekt haben soll. Ein Sicherheitsmodell ist ein (formaler oder semi-formaler) Mechanismus, der solch eine Sicherheitsstrategie realisieren soll. “*

Es existieren folgende Beschreibungsmittel, bzw. Modelltypen, mit denen ein Sicherheitsmodell erstellt werden kann [Kü96] :

- Endliche Automaten
- Zugriffsmatrizen
- Informationsflussmodelle
- Nicht-Beeinflussungsmodelle
- Kombinationen der obigen Beschreibungsmittel

Im nächsten Abschnitt wird ein konkretes Sicherheitsmodell beschrieben, welches als Basis bei der Aufstellung der meisten Sicherheitskriterien diente(vgl. 1.5).

1.3.1 Bell-LaPadula-Modell

Der Schwerpunkt der Sicherheitsforschung und Sicherheitsentwicklung auf Systemen liegt bei der Durchsetzung der Vertraulichkeit. Es gibt verschiedene *Sicherheitsmodelle*, die zur Durchsetzung von Vertraulichkeit und Integrität entwickelt worden sind. Eins dieser Sicherheitsmodelle ist das *Bell-LaPadula-Modell*, dass 1973 von D.Elliot Bell und L.LaPadula veröffentlicht und 1976 als Multics-Interpretation weiterentwickelt wurde.
Da die Kriterienkataloge, angefangen beim *Orange Book*, sich am *Bell-LaPadula-Modell* orientieren, soll es im Folgenden kurz erläutert werden.

1.3.2 Modellbeschreibung

Das Bell-LaPadula-Modell realisiert,

- das *least privilege*-Prinzip (Benutzer soll nicht unnötig hohe Privilegien erhalten),
- das *need to know*-Prinzip (Benutzer erhält nur Zugriff auf die Objekte, die er für seine Arbeit benötigt).

Die sicherheitsrelevanten Elemente und Komponenten (Mengen von Elementen) eines Systems werden in diesem Modell in abstrakter Form dargestellt. Dabei sind z.B. die (aktiven) Subjekte S_i und die (passiven) Objekte O_j, auf die die Subjekte zugreifen, Elemente.
Es gibt verschiedene Zugriffsarten, die durch Zugriffsattribute $\underline{\mathbf{x}}$ repräsentiert werden :

$\underline{\mathbf{e}}$ **:** (execute) weder schreiben noch lesen

$\underline{\mathbf{r}}$: (read) lesen und nicht schreiben

$\underline{\mathbf{a}}$: (append) schreiben und nicht lesen

$\underline{\mathbf{w}}$: (write) schreiben und lesen

Der Systemzustand wird durch ein 4-Tupel (**b,M,F,H**) beschrieben [FH92]:

1. **b** = Aktuelle Zugriffsmenge (current access set). Das Tripel (S_i, O_j, $\underline{\text{x}}$) beschreibt den aktuellen Zugriff $\underline{\mathbf{x}}$ eines Subjektes **i** auf das Objekt **j** in diesem Zustand. **b** ist die Menge aller Tripel, die aktuelle Zugriffe darstellen.

2. **M** = Die Zugriffsmatrix (access matrix). Die Einträge M_{ij} der Matrix, die Zugriffsatribute sind, spezifizieren die Zugriffsrechte von Subjekt S_i auf Objekt O_j.

3. **F** = Die Level-Funktion (level function). Jedes Subjekt und jedes Objekt erhält jeweils ein Sicherheitslevel. Dies ist ein Paar, bestehend aus einer Klassifikation (Menge der Klassifikationen ist dabei hierarchisch geordnet) und eine Menge von Kategorien. Die Levelfunktion F ist ein Tripel das aus drei Level-Zuordnungsfunktionen (f_S, f_O, f_C) besteht. Dabei ist

   (a) $f_S(S_i)$ = der maximale Sicherheitslevel eines Subjektes S_i,

   (b) $f_O(O_j)$ = der Sicherheitslevel eines Objektes O_j und

   (c) $f_C(S_i)$ = der aktuelle Sicherheitslevel eines Subjektes S_i.

   Mit den Klassifikationen wird das *least privilege*-Prinzip und mit den Kategoriemengen das *need to know*-Prinzip durchgesetzt.

4. **H** = Die Hierarchie (hierarchy). Diese Systemzustandskomponente wurde speziell für die Multix-Interpretation eingeführt. Es ist eine Objekt-Struktur, die eine Baum-Struktur auf den Objekten und auf isolierte Objekte zulässt.

Das System ist als eine Sequenz von Tripeln Anfragen (R_k), Entscheidungen (D_m), Zuständen (Z_n)) für einen Anfangszustand z_0 definiert, die eine bestimmte Relation für aufeinanderfolgende Zustände erfüllen.
Nach dem Bell-LaPadula-Modell werden unter Sicherheit drei Sicherheitscharakteristiken verstanden, die in jedem Zustand erfüllt sein müssen [FH92] :

1. **Simple Security-Property :** Falls (S, O, $\underline{\text{r}}$) ein aktueller Zugriff (d.h. Element von b) ist, so gilt :

$$f_S(S) \text{ dominiert } f_O(O)$$

   Hiermit soll verhindert werden, dass ein Subjekt ein Objekt mit höherer oder nicht vergleichbarer Sicherheitsklasse (read-up) lesen kann.

2. ***-Property**(Confinement Property) **:** Falls ein Subjekt gleichzeitig aktuellen Lese-Zugriff auf ein Objekt O_1 und aktuellen Schreib-Zugriff auf ein Objekt O_2 hat, so gilt :

   $$f_O(O_1) \text{ wird von } f_O(O_2) \text{ dominiert.}$$

   Es soll verhindert werden, dass ein Subjekt sensitive Daten lesen und in ein Objekt mit niedrigerer Sicherheitsstufe schreiben kann (write down). Es gelten folgende Gesetzmäßigkeiten : Falls (S, O, Attribut) ein aktueller Zugriff ist (Element aus b), so gilt

   $f_O(O)$ dominiert $f_C(S)$, falls Attribut = $\underline{\text{a}}$;
   $f_O(O)$ gleich $f_C(S)$, falls Attribut = $\underline{\text{w}}$;
   $f_O(O)$ wird von $f_C(S)$ dominiert, falls Attribut = $\underline{\text{r}}$;

3. **ds-Property**(discretionary security) **:** Falls (S_i, O_j, $\underline{\text{x}}$) in der aktuellen Zugriffsmenge b ist, dann ist das $\underline{\text{x}}$-Attribut im Matrix-Eintrag M_{ij}.

Simple Security- und ***-*Property*** sind mandatorisch, weil die Subjekte die Sicherheitslevels nicht verändern können. Die Subjekte können aber nach eigener Diskretion (durch die diskrete Sicherheit *ds*) Zugriffsrechte auf Objekte vergeben, sofern sie der nicht-diskreten Sicherheits-Strategie nicht widerprechen. Der Mechanismus, der *Simple Security-* und **-Property* realisiert, wird *Mandatory Access Control* (MAC) genannt, und der Mechanismus der *ds-Property* realisiert, wird *Discretionary Access Control* (DAC) genannt.

„Sicherheit“ kann folgendermaßen bewiesen werden [FH92]:

Sicherheits-Theorem :

Falls der Anfangszustand „sicher“ ist und bei jedem Zustandswechsel die *Simple-Security-*, die **-* und die *ds-* Eigenschaft erhalten bleibt, ist „Sicherheit“ systematisch beweisbar [FH92].

Zustandswechsel werden durch Regeln dargestellt, die jeder Anfrage r_k und jedem Zustand z_n eine Entscheidung d_m und einen Nachfolge-Zustand z_{n+1} zuordnen. Es genügt also zu gewährleisten, dass der Anfangszustand z_0 sicher ist und jeder Zustandswechsel (Regel) die Sicherheitseigenschaften erhält.

1.4 Sicherheitsarchitekturen

Eine Sicherheitsarchitektur beschreibt nach [Kü96]

> *„Prinzipien, Merkmale und Strukturen eines Systems, die dem Sicherheitsaspekt Rechnung tragen. Sie bildet einen Rahmen, der Sicherheitsmechanismen und Sicherheitsmodelle aufnimmt und für Applikationen zu einem homogenen Ganzen integriert. Sie liefert Paradigmen für den Entwurfsprozess eines sicheren Systems."*

Die Sicherheitsarchitektur kann als die Architektur der sicherheitsrelevanten Komponenten eines Systems aufgefasst werden. Im Zusammenhang mit Sicherheitsarchitekturen können für monolithische Systeme drei Begriffe angegeben werden, die traditionelle Konzepte darstellen :

- Der Referenzmonitor
- Die Trusted Computing Base (TCB)
- Der Sicherheitskern

Im Folgenden werden diese Architekturen beschrieben.

1.4.1 Sicherheitskern und der Referenzmonitor

Es gibt einige Konzepte, die der Realisierung einer sicheren Zugriffskontrolle dienen. Eines der wichtigsten wird als *Sicherheitskern-Systemarchitektur* bezeichnet. Eingeführt wurde dieses Konzept von *Roger Shell* und es realisiert das *Referenz-Monitor-Prinzip*. Der Referenzmonitor ist die Komponente eines Systems, die die Zugriffssteuerung implementiert.

Die unterste Betrachtungsebene eines Sicherheitsmodells kennt sog. Entitäten. Entitäten lassen sich in Subjekte und Objekte unterteilen. Subjekte greifen auf Objekte zu (subjects access objects). Eine Entität kann sowohl Subjekt als auch Objekt sein. Beispiele für Entitäten sind Benutzer, Festplatten, Dateien, Prozesse, der Betriebssystemkern usw. Die unterste Art von Zugriffen sind Lese- und Schreibzugriffe. Alle Aktionen, die innerhalb eines Computers stattfinden, können auf Lese- und Schreiboperationen zurückgeführt werden (vgl. Petrinetze [Val]).
Der Referenzmonitor wird als sog. *black box* aufgefasst, in der alles kontrolliert wird, was im System geschieht. Jede Aktion eines Subjektes besteht dabei aus einem lesenden oder schreibenden Zugriff auf ein Objekt. Der Referenzmonitor überprüft dabei die Zugriffsrechte (*access mode*) des Subjektes auf das Objekt. Dies geschieht mittels einer Datenbank (*authorization database*), die jedoch nicht Bestandteil des Referenzmonitors ist. Die Funktionen zur Überprüfung der Zugriffsrechte werden *reference functions* genannt. Der Referenzmonitor ist von der Autorisierungsdatenbank abhängig. Daher muss der Referenzmonitor Änderungen dieser Datenbank kontrollieren. Die hierfür zuständigen Funktionen werden als Autorisierungsfunktionen (*authorization functions*) bezeichnet.

Der Referenzmonitor stellt eine **abstrakte Maschine** dar, die den Zugriff von Subjekten auf Objekte mittels der Referenz- und Autorisierungsfunktionen kontrolliert. Gibt man beispielsweise bestimmte Dateien eines Dateisystems für irgendwelche Nutzer frei, so verändert man die Autorisierungsdatenbank. Dies wird dann von den Autorisierungsfunktionen des Referenzmonitors erledigt. Öffnet man mit einem Editorprogramm eine Textdatei, so überprüfen die Referenzfunktionen, ob dieser Zugriff gestattet ist. Drei Punkte sollte ein Referenzmonitor erfüllen:

1. Der Referenzmonitor muss geschützt sein. Es darf nicht möglich sein,dass er in seiner Zuverlässigkeit verändert wird (Schutz vor Manipulation).

2. Der Referenzmonitor muss vollständig sein. Es darf ihm keine Aktion verborgen bleiben (bei allen Zugriffen involviert).

3. Der Monitor muss verifizierbar sein (formal analysierbar, d.h, wohlstrukturiert und möglichst klein).

Ein Referenzmonitor, der diese Eigenschaften besitzt, enthält, nach [Kü96], folglich

- eine Zugriffsmatrix
- Operationen zur Rechteprüfung
- Operationen zur Rechteveränderung

Das Sicherheitsmodell benötigt zusätzlich zu den bisher besprochenen *Subjekten*, *Objekten* und dem Referenzmonitor noch drei weitere Komponenten :

1. Eine Identifikations- und Authentisierungskomponente (I&A)

2. Die schon angesprochene Autorisierungsdatenbank

3. Die *trusted computing base* (TCB)

Die I&A ist nicht auf das Login von Benutzern beschränkt, da auch Computer und Programme in einem Rechnernetz miteinander kommunizieren wollen, und somit auch solche Entitäten die I&A benötigen. Die TCP/IP Protokolle nutzen z.B. adressbasierte Authentifizierung. Da Adressen sich leicht fälschen lassen (*IP-Spoofing*), ist diese Art der Authentifizierung sehr schwach. Starke Authentisierungsverfahren in Rechnernetzen (*Distributed Computing* Environment) benutzen auf Kryptographie basierende Techniken.

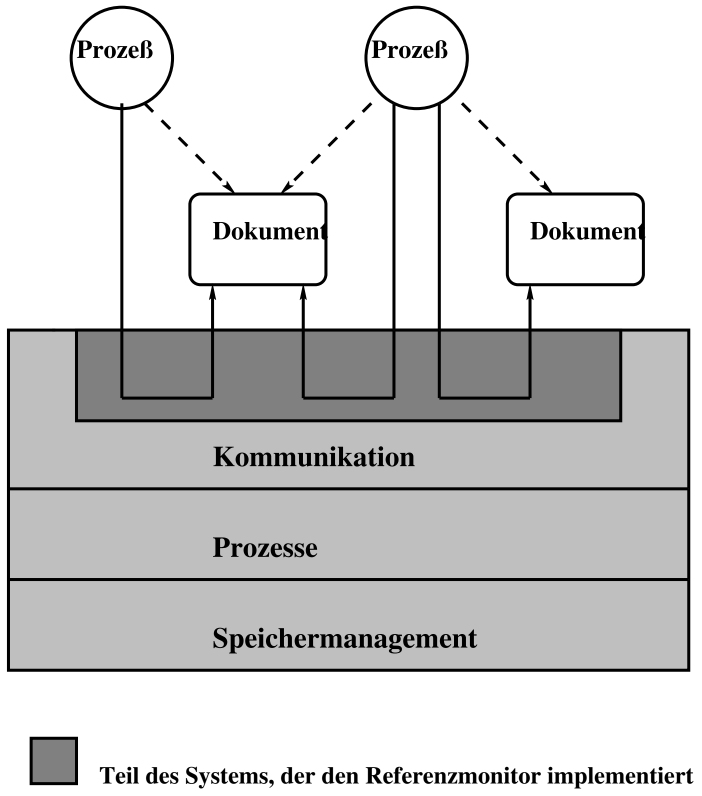

Abbildung 1.9: Referenzmonitor in Bezug auf das Gesamtsystem [Kü96]

1.4.2 Trusted Computing Base (TCB)

Die TCB ist die Gesamtheit aller Sicherheitsmechanismen eines Systems (Hardware, Firmware, Software), die in ihrer Kombination die Sicherheitspolitik durchsetzen (vgl. [Kü96]). Der Sicherheitskern seinerseits besteht aus den Teilen einer TCB, die das Referenzmonitorkonzept implementieren.

Die traditionellen Konzepte, Referenzmonitor, Sicherheitskern und TCB, weisen Probleme auf, die mit der heutigen Beschaffenheit von Rechnernetzen, den Sicherheitspolitiken und Anwendungen in Zusammenhang gebracht werden können.

Aus der TCB-Definition kann man ableiten, dass sie nur eine Sicherheitspolitik berücksichtigt. Administrativ verteilte Systeme besitzen aber mehrere Sicherheitspolitiken und sind anwendungsspezifisch, d.h. sie können ausgetauscht oder angepasst werden.

Der Referenzmonitor seinerseits beruht auf der Annahme, dass jede Sicherheitspolitik als eine Menge von Zugriffsregeln darstellbar ist. Gegenwärtig wird aber die Verfügbarkeit und Verbindlichkeit als Politikziel in den Vordergrund gestellt. Der Sicherheitskern stößt auf seine Grenzen auf Grund der Komplexität heutiger Sicherheitspolitiken. Dies wird in [Kü96] mit folgenden Feststellungen begründet

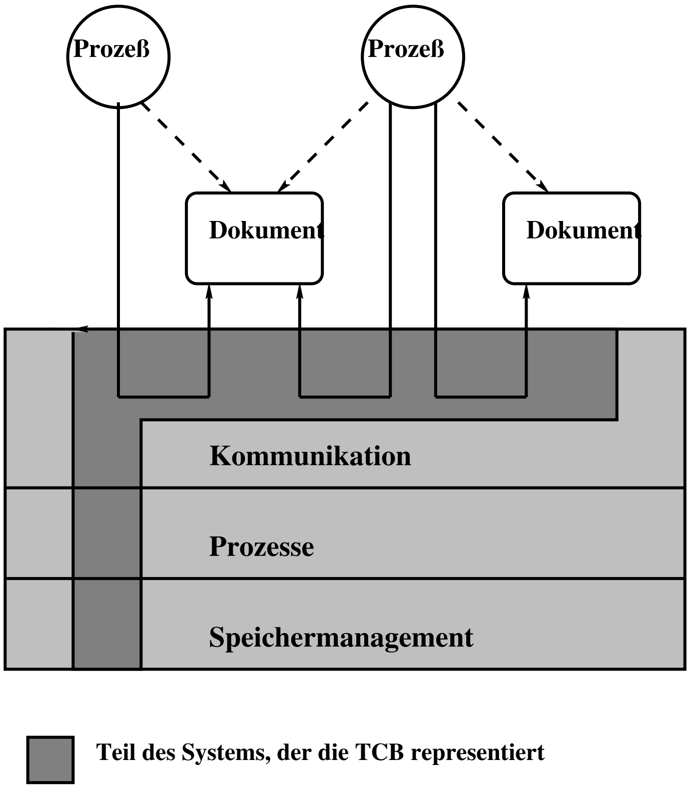

Abbildung 1.10: TCB in Bezug auf das Gesamtsystem [Kü96]

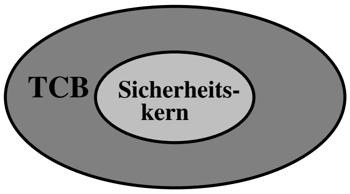

Abbildung 1.11: Beziehung von TCB und Sicherheitskern [Kü96]

:

- Die Darstellbarkeit durch das Belegen der Zugriffsmatrix ist limitiert.
- Die Politik ist abhängig von Systemvariablen, die nicht alle im Sicherheitskern liegen können.
- Die Politik enthält anwendungsspezifische Komponenten, die ausgetauscht oder angepasst werden können müssen.
- Die Integration und Koexistenz mehrerer Politiken.

Auf Grund der dargestellten Probleme wurden Sicherheitsarchitekturen entworfen, die den Anforderungen verteilter Systeme gerecht werden. Eine dieser Architekturen ist Kerberos und wird in Abschnitt 1.4.3 beschrieben.

1.4.2.1 Protokollierung (Auditing)

Eine weitere wichtige Komponente des Sicherheitssystems ist das Audit. Das Audit ist Teil der TCB, die der Referenzmonitor nutzt, um die Aktivitäten des Systems zu überwachen. In diesem Zusammenhang kann man sagen, dass Einbruchsentdeckung auf dem Audit basiert.

> **Intrusion Detection** *verbessert das traditionelle Sicherheitsmodell, indem es die Auditdaten auf Spuren und Muster von bekannten Angriffen durchsucht [Esc98].*

Protokollierung wurde in ihrer ursprünglichen Form zur nachträglichen Kontrolle der aufgezeichneten Informationen herangezogen. Beim Vorgang der Protokollierung werden alle Ereignisse, die für die Sicherheit relevant sind, in einer Protokoll-Datei (audittrail) festgehalten. Zu diesen Ereignissen zählen unter anderem : Die Benutzung des Identifikations- und Authentisierungsmechanismus, Hinzufügung von Objekten in den Adressraum des Benutzers (z.B. das Öffnen von Dateien), Löschen von Objekten, und Aktionen von Operateuren und Sicherheitsadministratoren. Bei vielen ID-Systemen, die einbruchsentdeckende Expertensysteme darstellen und auf Dorothy Denning's *Intrusion Detection Model* [Den87] basieren, wurde der Versuch unternommen, die Audit-Daten in Echtzeit auszuwerten. Die Vorgehensweise wird in [FH92] folgendermaßen aufgeführt :

Die Auswertung wird ausgeführt, indem die ID-Systeme statische Anomalien entdecken, oder mit Hilfe von Expertensystem-Regeln nach Aktionssequenzen suchen, die für Sicherheitsverstöße typisch sind. Solche Systeme können Sicherheitsverstöße mit einer bestimmten Wahrscheinlichkeit feststellen.

Eine ausführliche Beschreibung der Auswertungs- und Analysemethoden der ID-Systeme ist in **Kapitel 3** durchgeführt worden.

1.4.3 Kerberos

Kerberos ist eine Authentifikations- und Autorisierungsarchitektur für offene Rechnernetze. Das offene Netz besteht, grob gesehen, aus Arbeitsplatzrechnern, Servern und Sicherheitsserver.

Kerberos besitzt zwei grundlegende Konzepte, die, in Anlehnung an [Kü96], folgendermaßen beschrieben werden können :

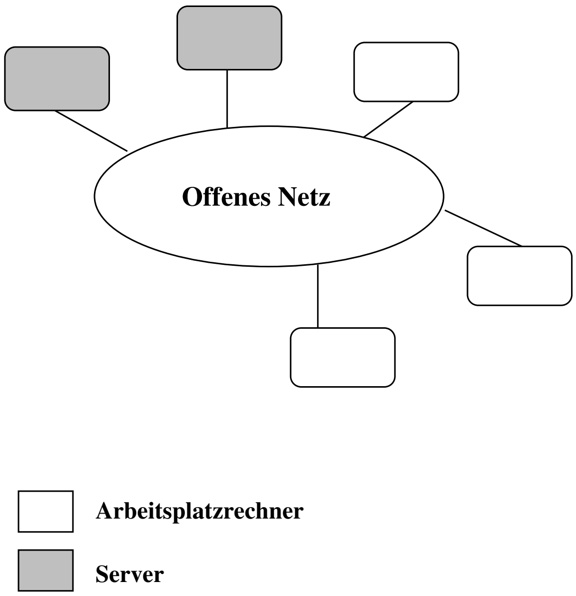

Abbildung 1.12: TCB Im Bezug auf das Gesamtsystem [Kü96]

Konzept (1) :

- Es gibt einen einzigen Punkt des gemeinsamen Vertrauens (single point of common trust) : Die *realm* (vgl. 3.4, 3.4.2). Eine realm ist im Allgemeinen eine Gruppe von ähnlichen Maschinen. Kerberos Realms reflektieren administrative Einheiten, innerhalb derer
  - einem Authentication Server,
  - einem TGS,
  - einer Kerberos Datenbank,

  vertraut wird, zwischen denen aber kein Vertrauen besteht.
- Es gibt *Benutzer* (Menschen) und *Instanzen*, die sich gegenüber Kerberos authentifizieren und Rechte an Dienstleistungen erwerben.
- Es gibt allgemeine *Server*, die diese Rechte prüfen und allgemeine Dienstleistungen (Namendienste, Fileserver, Mailserver) erbringen.
- Der *Kerberos-Authentification Server* ist die Authentifikationsinstanz. Sie authentifiziert Personen und Instanzen und versetzt sie damit in die Lage, sich selbst gegenüber allgemeinen Servern mittels *Authenticators* (dienen zum Nachweis der Identiät von Ticket-Vorzeigern beim Server) zu authentifizieren.
- Rechte und Dienstleistungen werden durch *tickets* kenntlich gemacht, die von dem *Kerberos Ticket Granting Server, TGS* (Kerberos Ticketausgabeserver), an einen Mensch oder eine Instanz vergeben werden.

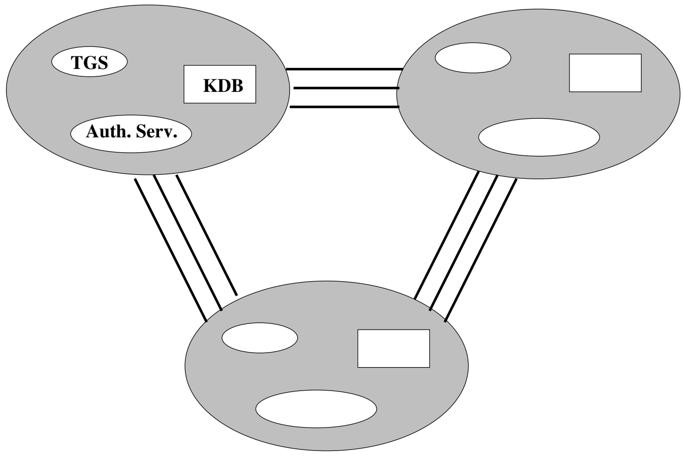

Abbildung 1.13: Die Kerberos Realms [Kü96]

- Bei jeder Benutzung eines Servers wird (a) ein Ticket für diesen Server und (b) ein Authenticator zum Nachweis der Identität verlangt.
- Benutzer, Instanzen und Server sind bei Kerberos angemeldet und die Kerberos-Daten werden in einer zentralen, replizierten Datenbank geführt.

Konzept (2) : Kerberos kennt drei *security levels* ([Kü96]) :

- Die Authentifikation zu Beginn einer Folge von Service-Benutzungen, trägt häufig zu beobachtenden Benutzungsmustern Rechnung. In Kerberos wird es z.B für Fileserver verwendet.
- Die Authentifikation erfolgt für jede Nachricht, aber nicht vertraulich. Durch digitale Signaturen, oder durch das Mitsenden von Authenticators, werden die Nachrichten authentisch.
- Authentifikation und Verschlüsselung werden für jede Nachricht verlangt, wenn z.B. ein Kerberos-Server Passwörter über das Netz versendet.

1.4.3.1 Kerberos-Architektur

Die Kerberos-Architektur besteht aus vier Komponenten (vgl. [Kü96]) :

1. **Authentication Server :** Der Authentication Server authentifiziert Menschen und Instanzen und erzeugt Sitzungsschlüssel und Ticket für TGS.
2. **Ticket Grantig Server (TGS) :** Der TGS gibt Tickets und Sitzungsschlüssel für spezifische Dienste aus.

3. **Kerberos Datenbank :** Die Datenbank enthält für jeden angemeldeten Benutzer und Instanz :

   - Benutzer- und Instanznamen
   - Passwort
   - privater Schlüssel (verschlüsseltes Passwort), geheimer Schlüssel

   Die Datenbank wird vom Kerberos Datenbank-Server verwaltet.

4. **Kerberos Datenbank-Server :** Der Kerberos Datenbank-Server trägt Benutzer ein, ändert Passwörter und Server-Schlüssel.

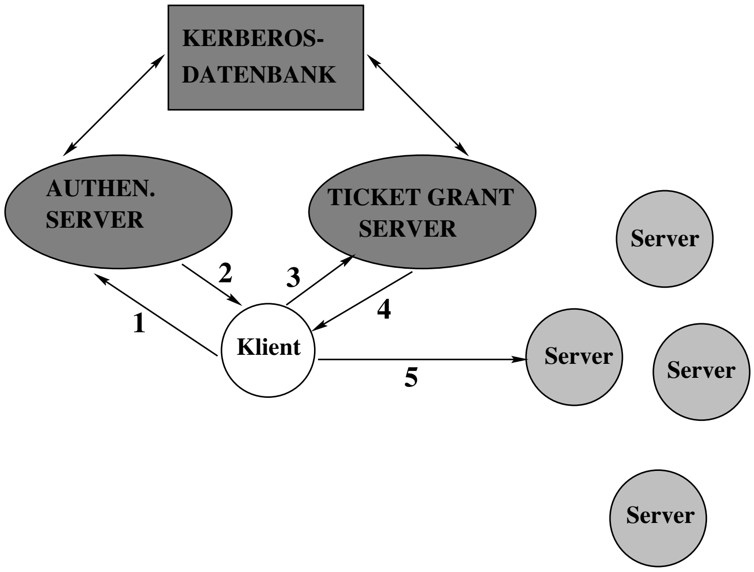

Abbildung 1.14: Die Kerberos Architektur [Kü96]

Abb.1.14 zeigt graphisch den Authentifizierungs- und Autorisierungsablauf bei Kerberos. In [Kü96] wird dieser Ablauf wie folgt dokumentiert :

1. Authentifikation; TGS-Ticket Anforderung

2. TGS-Ticket, TGS-Sitzungsschlüssel; gültig für ca. einen Tag

3. Server-Ticket Anforderung

4. Server-Ticket, Server Sitzungsschlüssel

5. Dienstleistungsanforderung : Server trifft Entscheidung basierend auf

   - Identität des Klienten
   - Ticket des TGS
   - lokalen ACLs

1.4.3.2 Schwächen von Kerberos

Eine der größten Schwächen von Kerberos ist die Tatsache, dass die Sicherheit auf der Geheimhaltung der Schlüssel beruht. Es ergibt sich aber die Frage nach dem Ort, wo die geheimen Schlüssel aufbewahrt werden sollen. Dafür gibt es mehrere Möglichkeiten, von denen keine eine endgültige, sichere Lösung bietet. In Anlehnung an [Kü96] gibt es folgende Möglichkeiten :

- **Schlüsselablage in Kerberos-Servern :** Jeder mit root-Rechten kann die geheimen Sitzungsschlüssel des Servers lesen. Da der Server jederzeit mit dem Netz verbunden ist, kann er angegriffen werden. Dies macht einen besonderen Schutz seiner sensitiven Funktionen erforderlich.

- **Schlüsselablage bei Mehrbenutzer-Maschinen :** Dabei sind die Sitzungsschlüssel auf den Klientrechnern abgelegt, auf die aber nur der root zugreifen kann. Die Gefahr liegt darin, dass im Falle von geglückten Angriffen auf das lokale Schutzsystem auch die Überwindung von Kerberos leicht wird.

- **Schlüsselablage bei Rechner ohne lokale Platte :** Schlüssel liegen in */tmp* (temporärer Datei bei UNIX) auf einer anderen Maschine. Wegen Netzwerkverkehr ist auch bei dieser Variante mit Sicherheitsgefahren zu rechnen. Bei neueren Lösungsansätzen liegen die Schlüssel im *shared memory*. Aber auch in diesem Fall droht Gefahr, z.B. durch *paging* übers Netz.

Neben dem Problem der Schlüsselablage weist Kerberos weitere Schwächen auf, wie z.B. Protokollschwächen. Außerdem muss auf insgesamt vier Instanzen vertraut werden, nämlich *Authentication Server, TGS, Kerberos Datenbank System, Time Server*. Auch das Erraten und Ausspionieren der Passwörter ist eine relevante Schwäche, sowie die Ticketschwächen, weil die feste Beziehung zwischen Client und Server die Delegation[1] umständlich macht.

[1] Die Übertragung von Rechten oder Aufgaben auf einen Repräsentanten ([Kü96])

1.4.4 Gesamtübersicht der behandelten Sicherheitsbegriffe

In diesem Abschnitt soll, graphisch, eine Gesamtübersicht, der bis jetzt vorgestellten und erwähnten, Begriffe wiedergegeben werden. Einige Ausführungen lehnen sich dabei an das, in [Kü96] vorgestellten Themengerüst. Die in der Abb. 1.15 hervorgehobenen Begriffe verdeutlichen die, in den Abschnitten 1.1 bis einschließlich 1.4, behandelten Themen. Die Ausführung der restlichen Themen würde den Rahmen dieser Diplomarbeit sprengen und wurde daher weggelassen.

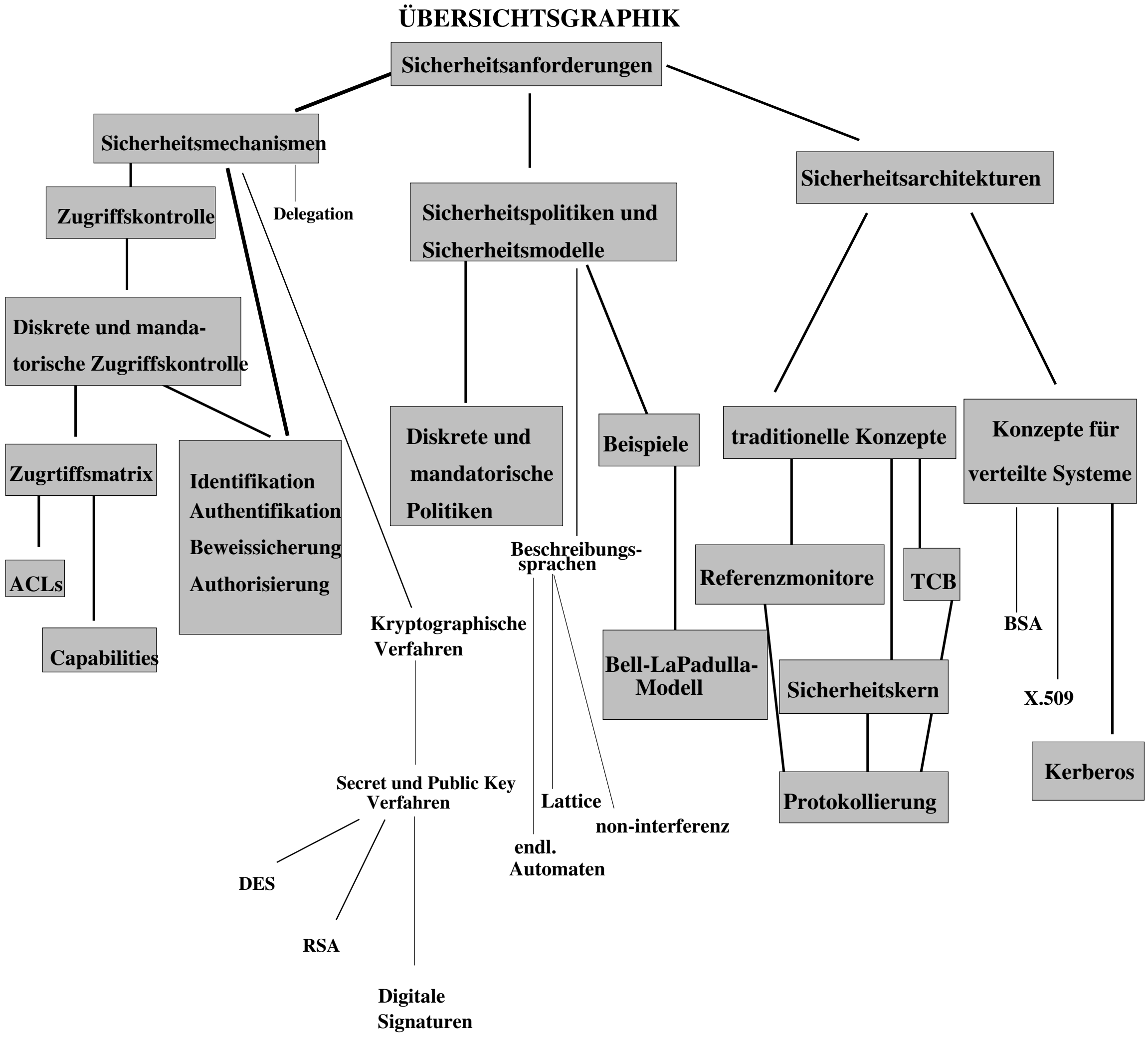

Abbildung 1.15: Gesamtübersicht - in Anlehnung an [Kü96]

1.5 Sicherheitskriterien-Kataloge

Um die Sicherheit von Informationssystemen objektiv und nach einheitlichen Kriterien beurteilen zu können, wurden nationale bzw. internationale Richtlinien geschaffen, nach denen sich Systeme in verschiedene Sicherheitsklassen einteilen lassen und nach denen sie von einer offiziellen Instanz evaluiert werden können. Die Kriterienkataloge behandeln weitgehend Einzelsysteme (ausgenommen das *Red Book* und *CC*) und sollen im Folgenden beschrieben werden.

1.5.1 TCSEC - Das *Orange Book*

Der erste bedeutende Klassifikationkatalog war das *Orange Book* (TCSEC - NCSC Trusted Computer System Evaluation Criteria), herausgegeben 1983 vom amerikanischen Verteidigungsministerium (DoD - Department of Defense). Darin werden IT-Systeme in vier Sicherheitsstufen, die teilweise weiter unterteilt sind, untergliedert [Kya98]:

- **D** (minimal security)
- **C** (1,2; discretionary protection)
- **B** (1,2,3; mandatory protection)
- **A** (verified protection)

Aus diesen vier Stufen ergeben sich sieben Klassen. Der Aufbau der Klassen ist hierarchisch, wobei die Reihenfolge A, B1, B2, B3, C1, C2, D den Sicherheitsstandard der Systeme angibt. Systeme der Klasse D besitzen den geringsten Sicherheitsstandard, Systeme der Klasse A den höchsten. Mit den Klassen wird in erster Linie die Vertraulichkeit von Systemen bewertet, indem für jede dieser Klassen zu erfüllende Kriterien für die Bereiche *Security Policy, Accoutability, Assurance,* und *Documentation* aufgestellt werden. Das NCSC (National Computer Security Center) kann auf Antrag Zertifikate für Systeme nach den Klassen des *Orange Book* vergeben.
im Folgenden erfolgt eine detaillierte Beschreibung der Kriterien für die einzelnen Sicherheitsklassen des *Orange Book* nach [Kya98] und [FH92].

1.5.1.1 Klasse-D-Systeme (Minimaler Schutz)

Diese Systeme stellen nach *Orange Book*, den geringsten Sicherheitsstandard dar, weil sie nicht die Anforderungen erfüllen, in eine der Sicherheitsklassen C1 bis A eingereiht zu werden.

1.5.1.2 Klasse-C-Systeme: (Diskreter Schutz)

Die Kriterien für die Sicherheitsklasse C erfordern das Vorhandensein von benutzerbestimmbaren Zugangsbeschränkungen. Bei diesen Systemen können vom Benutzer die Rechte für den Systemzugang bestimmt werden z.B. über Zugangslisten (vgl.1.2.1.1).

C1-Systeme (Diskreter Sicherheits-Schutz) : Bei C1-Systemen befinden sich die Benutzergruppen alle auf demselben Sicherheitsniveau. Gefordert nur die Trennung zwischen Benutzern und Daten. Ein Benutzer soll die Möglichkeit haben, den Zugriff auf seine Objekte, auf bestimmte Benutzer und/oder Objekte - Gruppen einzuschränken. Die meisten UNIX-Systeme erfüllen die Anforderungen von C1-Systemen.

C2-Systeme (Kontrollierbarer Zugriffsschutz) : In C2-Systemen müssen die benutzerbestimmbaren Beschränkungen so realisiert sein, dass die Operationen der einzelnen Benutzer überwacht und gespeichert werden können, z.B durch Login-Prozeduren, Protokollierung sicherheitsrelevanter Ereignisse (Auditing), Isolierung von Betriebsmitteln. Die einzelnen Benutzer müssen individuell identifizierbar, und die Überwachungsdaten vor nicht autorisierten Zugriffen geschützt sein ([Kya98]).
Folgende Operationen müssen überwacht werden können [Kya98]:

- Benutzung des Authentifikations-Mechanismus,
- Löschung von Objekten,
- die Eröffnung von neuen Objekten durch den Benutzer (Programmstart, Dateieröffnung etc.).

Nahezu alle UNIX-Systeme, wie auch Windows NT (nach TNI) erfüllen, entsprechend konfiguriert, die C2 Vorgaben.

1.5.1.3 Klasse-B-Systeme (Mandatorischer Schutz)

Die Klasse-B-Kriterien fordern hinausgehend über die Klasse-C-Voraussetzungen, die Implementation von festgelegten, regelbasierenden (z.B. basierend auf das Bell-LaPadula-Modell) Schutzmechanismen. In einem Die Klasse-B-System können die Benutzer nicht mehr Rechte vergeben. Die Rechtevergabe ist ausschließlich dem Systemadministrator vorbehalten und erfolgt nach definierten Regeln.

Diese Regeln basieren auf der Klassifikation der jeweiligen Informationen, nach dem Bell-LaPadula-Modell, in ([Kya98]):

- nicht vertraulich,

- vertraulich,
- geheim und
- streng geheim.

Jede Klasse gibt einen Geheimhaltungsgrad an. Jedem Objekt ist, über eine Markierung, einer der Geheimhaltungsgrade zugeordnet. Ob ein Benutzer berechtigt ist auf ein Objekt zuzugreifen, wird durch einen Vergleich von Benutzerberechtigung und dem Geheimhaltungsgrad des Objektes festgestellt.

B1-System (Zugriffsschutz unter Verwendung von Sicherheitslevels) : Subjekten und Objekten werden, zur Realisierung einer mandatorischen Zugriffskontrolle, Sicherheitslevel zugeordnet, die aus einer Menge von mehreren hierarchisch geordneten Sicherheitsklassen und einer Menge von nicht-hierarchischen Kategorien bestehen. Es wird eine informale Beschreibung des Sicherheitsmodells verlangt, wobei neben einer diskreten Sicherheitsstrategie, die mandatorische Sicherheitsstrategie des Bell-LaPadula-Modells vorausgesetzt wird (vgl. [FH92]). B1-Systeme müssen folgende Kriterien erfüllen ([Kya98]) :

- verbindliche Zugangskontrollen,
- Markierung aller Objekte durch Geheimhaltungsstufen,
- alle bekannten Möglichkeiten, in das System einzubrechen, müssen beseitigt sein,
- eine formelle oder informelle Beschreibung des Sicherheitsmodelles muss verfügbar sein,
- Ausdrücke von vertraulichen Daten müssen sichtbar als solche kenntlich sein,
- eine vollständige Dokumentation der Funktionen und Aufgaben des Systemverwalters muss verfügbar sein.

Die Sicherheitsklasse B1 kann mit erheblichen Anstrengungen (vor allem im Bereich der Dokumentation) von UNIX-Systemen noch erreicht werden. Nach Kyas [Kya98] bieten eine Reihe von Herstellern spezielle Versionen ihrer Betriebssysteme an, die B1-konform sind, z.B. :

- Von Amdahl Corparation,**UTS/MLS, Version 2.1.5+** (vom 07.01.1994)
- Von Digital Equipment Corporation, **SEVMS VAX Version 6.0** (vom 30.06.1994)

- Von Digital Equipment Corporation **SEVMS VAX Version 6.1**(vom 14.07.1995)
- Von Digital Equipment Corporation **SEVMS VAX and Alpha Version 6.1** (vom 24.10.1996)
- Von Digital Equipment Corporation **ULTRIX MLS+ Vers. 2.1 on VAX Station 3100** (vom 18.04.1996)
- Von Harris Computer Systems Corporation **CX/SX 6.2.1** (vom 18.09.1995)
- Von Hewlett Packard Corparation **HP-UX BLS release 8.4 und 9.0.9+** (vom 21.09.1993 und 01.12.94)
- Von Silicon Graphics Inc. **Trusted IRIX/B release 4.0.5** (vom 6.02.1995)
- Von Unisys Corporation **OS 1100 Security Release I**(vom 27.09.1989)
- Von Unisys Corporation **OS 1100/2200 Release SB3R6**(vom 05.04.1991)
- Von Unisys Corporation **OS 1100/2200 Release SB4R7**(20.04.1994)
- Von Computer Associates International **CA-ACF2 MVS mit CA-ACF2 MAC** (23.03.1998)

B2-Systeme (Strukturierter Schutz) : Die B2-Systeme müssen, über die B1-Klassifizierung hinausgehend folgende die Sicherheitskriterien erfüllen Bedingungen ([Kya98], [FH92]) :

- Die TCB (Trusted Computing Base), d.h der sicherheitsrelevante Teil des Systems, muss auf einem klar definierten und dokumentierten Sicherheitsmodell basieren.
- Die TCB muss in sicherheitsrelevante und nicht sicherheitsrelevante Module strukturiert werden
- Die TCB-Schnittstelle muss wohldefiniert sein und die TCB-Design und Implementation muss ein intensives Testen/Überprüfen ermöglichen
- Verbindliche Zugangskontrollen zu allen, direkt und indirekt vom IT-System erreichbaren Komponenten, müssen vorhanden sein.
- Der Kommunikationspfad zwischen Benutzer und IT-Systeme muss gesichert und vertrauenswürdig sein.
- Das IT-System muss gegen elektromagnetische Abstrahlung nach außen abgeschirmt sein.

- Operator und Systemverwaltungsfunktionen müssen getrennet sein. Die Aufgaben des Operators sind die Durchführung von Routine-Tätigkeiten wie Systemstart, Sicherungskopien oder Wartungsarbeiten. Der Systemverwalter hat die Berechtigung, neue Benutzerzugänge einzurichten und Sicherheitskontrollen durchzuführen.

- Alle Einrichtungen müssen mit ihren minimalen und maximalen Geheimhaltungsstufen markiert sein.

Einige mit B2 zertifizierten Systeme sind :

- Von Trusted Informations Systems, Inc., **Trusted XENIX 4.0**(vom 17.-09.1993)

- Von Wang Federal Inc., **Multics Version MR11.0** (vom 01.09.1995)

- Cryptek Secure Communications LLC, **VSLAN/VSLANE 5.1** (vom 11.-01.1994)

B3-Systeme (Sicherheitsbereich) : B3-konforme Systeme müssen in der Regel von Grund auf als solche konzipiert und entwickelt werden. Die TCB muss das Referenz-Monitor-Prinzip realisieren und so strukturiert sein, dass sie keinen Code enthält, der nicht zur Realisierung der Sicherheitsstrategie beiträgt. Der Auditing-Mechanismus muss sicherheitskritische Ereignisse signalisieren und selbstständig eingreifen können ([FH92]) . Ein UNIX-basierendes, B3-konformes System würde nach [Kya98] eine vollständige Neugestaltung des Kernels erfordern. Zu Testzwecken müssen alle nicht mit Sicherheitsfunktionen in Verbindung stehenden Software-Komponenten entfernt werden können. Neben den B2-Forderungen müssen im Einzelnen [Kya98]:

- Die Zugriffslisten auch Einträge der nicht zugriffsberechtigten Benutzer erhalten,

- Eine äußerst detaillierte Beschreibung von Funktion, Verhalten und Design des IT-Systems vorhanden sein,

- Ein automatisches Hilfsmittel zur Erfassung und Meldung von sicherheitsrelevanten Ereignissen vorhanden sein,

- das IT-System modular aufgebaut sein, bestimmte Funktionen sind in Hardware zu realisieren,

- Gesicherte Mechanismen für die Wiederherstellung des ursprünglichen Systemzustandes nach Systemfehlern implementiert sein.

Einige mit B3 zertifizierten Systeme sind :

- Von Wang Federal Inc., **XTS-300 STOP 5.2.E** (vom 30.04.2000)
- Von Wang Federal Inc., **XTS-300 STOP 4.4.2** (vom 31.03.1998)
- Von Wang Federal Inc., **XTS-300 STOP 4.1.a** (vom 31.10.1995)

1.5.1.4 Klasse-A-Systeme (Verifiziertes Design)

Ausser der geforderten, formalen Beschreibung des gesamten Software-Design-Modells erfordern Klasse-A-Systeme keinerlei Systemerweiterungen gegenüber der B3-Klassifikation. Auf der Grundlage dieser Beschreibung muss ein Nachweis der fehlerfreien Implementationen der Software erbracht werden können. Dazu muss die Konsistenz des benutzten Sicherheitsmodelles einer Überprüfung mit mathematischen Methoden standhalten. Schließlich müssen Mechanismen nachgewiesen werden, die die Auslieferung von Soft- und Hardware in der spezifizerten Form garantieren. Es gibt folgende Produkte, die mit A1 zertifiziert worden sind:

- Von Boeing Company, **MLS LAN Secure Network Server System** (vom 28.08.1991)
- Von Boeing Company, **MLS LAN Secure Network Component MDIA, Version 2**(vom 15.08.1994)
- Von Gemini Computers, Inc **Gemini Trusted Netwerk Processor**(vom 06.09.1994)

1.5.2 TNI - *Red Book*

Vom National Computer Security Center (NCSC) wurde 1987, ergänzend zum *Orange Book*, die Trusted Network Interpretation, auch *Red Book* genannt, herausgegeben [ncs87]. Das *Red Book* stellt eine Interpretation des *Orange Book* für vertrauenswürdige Netzwerksysteme und zusätzliche, für Netzwerke speziell erforderliche, Sicherheitsanforderungen auf.
Mit den TNI sollen nach [PH90] drei allgemeine Ziele erreicht werden :

1. Den Herstellern soll ein Standard zur Verfügung gestellt werden, der vorschreibt, welche Sicherheitsfunktionen ihre kommerziellen Netzwerkprodukte enthalten sollen und mit welcher Qualität diese Funktionen zu realisieren sind.
2. Es soll ein Maßstab zur Verfügung gestellt werden, mit dessen Hilfe der Grad des Vertrauens ermittelt wird, der in ein Rechensystem hinsichtlich der Behandlung sensitiver Informationen gesetzt werden kann.
3. Den Anwendern soll eine Grundlage zur Formulierung von Sicherheitsanforderungen in Pflichtenheften zur Verfügung gestellt werden.

In vernetzten Systemen sollen bestimmte Sicherheitsziele erreicht werden:

- Kontrolle des Zugriffs zur Vermeidung der unberechtigten Kenntnisnahme von Objekten (Unauthorized Disclosure),
- Vermeidung der unberechtigten Änderung von Nutz-, Anwender- und Systemdaten (Integrity),
- Vermeidung der Dienstleistungsverhinderung (Denial of Service).

Das *Red Book* besteht aus zwei Teilen und drei Anhängen, die im Folgenden kurz beschrieben werden [PH90] :

Teil I : Interpretation der grundlegenden Prinzipien der TCSEC mit Übertragung der Sicherheitsmaßnahmen, der Garantieforderungen und der Bewertungsstruktur auf Netzwerke, wie z.B. auf isolierte Local Area Netzwerke (LAN) bis hin zu Wide Area Netzwerk (WAN) Systemen.

Teil 2 : Beschreibung (specification and evaluation) der in Netzwerken relevanten Sicherheitsaspekte und -dienste der Kommunikationssicherheit (insbesondere Transmission Security und unterstützende Dienste wie Verschlüsselungsverfahren und Protokolle) und des Denial of Service. Die Beschreibung wird unter den folgenden drei Aspekten vorgenommen:

- Funktionalität,

- Stärke des Mechanismus und
- Garantie.

Weiterhin wird eine Bewertung der Sicherheitsdienste, hinsichtlich des Ziel-Erreichungsgrads dieser drei Aspekte, unterstützt. Sie unterscheidet sich von der hierarchischen Bewertung des TCSEC, weil sich der Anwender des *Red Book* sowohl für oder gegen die Nutzung einzelner Sicherheitsdienste entscheiden und auch den jeweilig gewünschten Qualitätsgrad festlegen kann. Folgende Sicherheitsdienste werden im *Red Book* aufgeführt :

- Kommunikationsintegrität (Authentifizierung, Kommunikationsintegrität, Nicht-Zurückweisung),
- Verweigern von Diensten (Aufrechterhaltung des Betriebs, Protokoll-basierte Schutzmechanismen, Netzmanagement),
- Schutz gegen Bloßstellung (Vertraulichkeit der Daten, Vertraulichkeit des Datenverkehrs, Wegewahl).

In den Anhängen A bis C wird die Bewertung von Netzkomponenten beschrieben, die Aufteilung der TCB (Trusted Computing Base) eines Netzes auf seine Komponenten begründet und die Bewertung komplexer Netze dargestellt [PH90].

Anhang A (Component Evaluation) : Bewertung von Netzwerkkomponenten. Das Netz wird nicht mehr aus ganzheitlicher Sicht betrachtet. Es wird eine komponentenorientierte Bewertung angeboten aufgrund der beiden folgenden Tatsachen :

1. Bestimmte Netzwerkkomponenten werden nur von bestimmten Herstellern angeboten;
2. Netzwerkkomponenten, die einmal für ein Netzwerk zertifiziert sind, können in anderen Netzwerken eingesetzt werden ohne erneut bewertet werden zu müssen. Dadurch wird die Anzahl der zu bewertenden Systeme und Komponenten verringert.

Grundsätzlich werden dabei die Bewertungsklassen des *Orange Book* (D, C1, C2, B1, B2, B3, A0) übernommen. Zusätzlich dazu wird im *Red Book* aufgeführt, welche Sicherheitspolitik durch die Komponenten unterstützt wird.

Anhang B (Rationale for the Partitioned NTCB Approach) : Grundlagen der verteilten NTCB. Begründung für die Aufteilung der TCB auf unterschiedliche Komponenten.

Anhang C (The Interconnected Accredited AIS View) : Der Ansatz zur Verknüpfung vernetzter und zertifizierter Systeme der IT. Verknüpft werden können, nach unterschiedlichen Klassen der TCSEC zertifizierte Systeme, ohne dass das Netzwerk einheitlich mit einer Klasse bewertet werden könnte.

Component Connections and the Interconnection Rule : Vernetzte Komponenten und die Verknüpfungsregel.
Die Verknüpfungsregel begrenzt den Bereich von Geheimhaltungsgraden und Kategorien beim Informationsaustausch zwischen vernetzten Systemen der IT :

- Bei Simplexverbindungen muss die empfangende Komponente im Vergleich zur sendenden Komponente für eine höhere oder dieselbe Sicherheitsstufe freigegeben sein;
- bei Dublexverbindungen müssen die Komponenten für dieselbe Sicherheitsstufe freigegeben sein.

The Global Network View : Gesamtbetrachtung des Netzwerkes.
Es können weitere Risiken auftreten, die nur durch eine detaillierte Betrachtung aller Komponenten beherrscht werden können. Dabei wird besonders auf das *Kaskadenproblem* hingewiesen : Die Möglichkeit der unberechtigten Kenntnisnahme von Daten aus einem Netzwerk, das insgesamt für ein größeres Intervall von Geheimhaltungsgraden und Kategorien freigegeben ist als jedes einzelne (vernetzte) System, aus dem unberechtigt Daten zur Kenntnis genommen werden könnten [PH90].

Kriterium für die Bewertung eines Netzwerkes mit den TNI ist dessen Komplexität. Daher können nicht alle Netzwerke von Systemen der IT mit den TNI bewertet werden.

1.5.3 Die deutschen IT-Sicherheitskriterien-*Green Book*

1989 wurde von der ZSI (Zentralstelle für Sicherheit in der Informationstechnik), im Auftrage der Bundesregierung, die erste Fassung eines Kriterienkatalogs, bekannt unter dem Namen *IT-Sicherheitskriterien*, herausgegeben. Die IT-Sicherheitskriterien wurden von der ZSI, aufbauend auf dem *Orange Book*, entwickelt und sind als eine deutsche Lösung zu betrachten. Ergänzt werden die IT-Sicherheitskriterien durch ein *IT-Evaluationshandbuch* [bsi89], das beschreibt, wie informationstechnische Systeme oder selbstständige Komponenten nach diesen Kriterien geprüft werden.

Im *Orange Book* beinhalten die Klassen eine genaue Beschreibung der Kriterien und mit der Zunahme der Sicherheitsstufe nimmt auch die detaillierte

Beschreibung der Kriterien zu. Dies hat zur Folge, dass die Funktionalität und die Qualität eng gekoppelt sind. Das Problem bei dieser Art von Unterteilung ist, dass eine Vielzahl von relevanten Systemen nicht in diese Klasseneinteilung hineinpassen und nach dem Kriterienkatalog nicht evaluierbar sind.
Bei der Erarbeitung der IT-Sicherheitskriterien wurde versucht, dieses Problem zu umgehen, indem die Funktionalitätsklassen und Qualitätsstufen voneinander getrennt und aufeinander abgestimmt wurden, d.h zu den einzelnen Funktionalitätsklassen wurde eine äquivalente Qualitätsstufe erarbeitet. Der Begriff *Funktionalitätsklasse* könnte in dem oben aufgeführten Zusammenhang folgendermaßen definiert werden:

Funktionalitätsklasse: Klasse, die bestimmte Mindestanforderungen, bezüglich der Funktionalität der Sicherheitsfunktionen, an ein IT-System stellt.

In [bun91] heißt es weiterhin:

> *„ Die ersten Funktionalitätsklassen der IT-Sicherheitskriterien sind so formuliert worden, dass sie weitgehend die Funktionalität abdecken, wie sie das Orange Book in seinen Klassen C1 bis A fordert. Die Qualitätsstufen sind vollkommen unabhängig vom Orange Book entstanden und beinhalten eine Vielzahl von Kriterien, die einen Einfluß auf die Qualität eines Software-Produktes haben, die jedoch im Orange Book nicht vorhanden sind. “*

Der Kriterienkatalog der *IT-Sicherheitskriterien* ist in Funktionalitätsklassen F1 bis F10 und in Qualitätsklassen Q0 bis Q7 aufgeteilt, wobei F1 und Q0 die niedrigste Stufe , F10 und Q7 die höchste Sicherheitsstufe kennzeichnen. Die Einteilung der Klassen ist keine abgeschlossene Aufteilung und kann, bei Bedarf, nach obenhin weiter definiert werden.
Als Grundfunktionen sicherer Systeme werden in den IT-Sicherheitskriterien folgende Funktionen aufgeführt [bsi89]:

Identifikation und Autorisierung:

In den Sicherheitsanforderungen müssen die Subjekte und Objekte spezifiziert sein, für die eine Identifikation bzw. eine Identifikation und Authentisierung durchgeführt werden muss. Eine Authentisierung kann nur dann unterbleiben, wenn eine fehlerhafte Identifikation praktisch ausgeschlossen werden kann. Diese Grundfunktion ist Voraussetzung für eine sinnvolle Anwendung anderer Grundfunktionen,wie z.B. *Rechteverwaltung, Rechteprüfung* und *Beweissicherung.*

Rechteverwaltung:

Identifizierbare Subjekte können Rechte bezüglich anderer identifizierbarer Objekte besitzen. Da diese Rechte im Allgemeinen nicht statisch sind, wird eine

Rechteverwaltung benötigt, die die in den Sicherheitsanforderungen festgelegten Rechte zur Modifikation von Regeln realisiert.
Vielfach bestehen auch Beschränkungen der Art, dass Rechte nur dann ausgeübt werden können, wenn das Subjekt ein bestimmtes Attribut besitzt. Dadurch können Rollen definiert sein, was dem Subjekt die Möglichkeit in den Besitz eines bestimmten Rechtes zu gelangen nimmt, falls das entsprechende Attribut nicht gesetzt ist.

Rechteprüfung:

Bei jedem Versuch eines identifizierbaren Subjektes, Rechte bezüglich eines anderen Subjektes oder Objektes auszuüben, besteht die Aufgabe der Rechteprüfung darin, nur solche Aktionen zu erlauben, die das identifizierbare Subjekt aufgrund seiner vorhandenen Rechte ausüben darf.

Beweissicherung:

Die Beweissicherung protokolliert Informationen über erfolgte oder versuchte Ausübung von Rechten. Dadurch kann nachträglich überprüft werden, ob ein Missbrauch von Rechten erfolgte oder versucht wurde. Andere Grundfunktionen, die hier nicht näher erläutert werden sollen sind : *Wiederaufbereitung, Fehlerüberbrückung, Gewährleistung der Funktionalität,* und *Übertragungssicherung.*

Die ersten fünf Funktionalitätsklassen (F1 bis F5) decken die Funktionalität ab, wie sie das *Orange Book* in seinen Klassen C1 bis B3 fordert.

Zusätzlich zu diesen wurden fünf neue Funktionalitätsklassen definiert für Systeme mit hohen Anforderungen an [FH92]:

- Integrität von Daten und Programmen (F6)
- Systemverfügbarkeit (F7)
- Vertraulichkeit von Daten bei der Datenübertragung (F9)
- Vertraulichkeit und Integrität der zu übertragenden Informationen bei Vernetzungen (F10)

Die Qualitätsstufen (Q0 bis Q7) stellen unterschiedliche Anforderungen an die Qualität der Sicherheitsanforderungen, Qualität der Spezifikation, Qualität der Abgrenzung zu nicht zu evaluierenden Systemteilen, Qualität des Herstellungsvorgangs, Betriebsqualität und Qualität der Dokumentation. Wie oben erwähnt sind die Qualitätsstufen unabhängig vom *Orange Book* entstanden und erhalten zusätzliche Qualitätskriterien, die nicht im *Orange Book* enhalten sind. Eine nähere Beschreibung des Kriterienkatalogs und der Qualitätsstufen sowie eine Gegenüberstellung zum *Orange Book*, kann den Tabellen 1.1, 1.2 und 1.3 entnommen werden.

Sicherheitsklassen	**Sicherheitsmerkmale**
F1:	-Identifikation und Autorisierung als erste Maßname -Verwaltung der Rechte nach Benutzer und oder Gruppen -Rechteüberprüfung bei jedem Zugriff
F2:-F1 und zusätzlich:	-Eindeutige Identifikation und Zuordnung von Aktionen zu dieser Identität -Rechtevergabe auf der Ebene der einzelnen Benutzer -Protokollierung und Überwachung sicherheitsrelevanter Aktionen -Löschen von Speicherobjekten vor der Wiedervergabe
F3: -F2 und zusätzlich:	-Einhaltung globaler Regeln bei der Rechtevergabe -Subjekte und Objekte haben einen 2-dimensionalen Attributraum (Attribute bleiben beim Im- und Export erhalten) -Informationsfluss kontrolle bei Rechteüberprüfung -Erweiterte Protokollierung
F4: -F3 und zusätzlich	-Identifikation und Authentisierung muss über einen vertrauenswürdigen Pfad erfolgen -Möglichkeit der Zuordnung von Rechten zu Rollen (mindestens Systemadministrator und Systemverwalter) -Definition von multilevel-Kanälen mit mini- und maximalem Level -Überwachung von Ereignissen, die als verdeckte Kanäle benutzt werden können
F5: -F4	-Rolle des Sicherheitsbeauftragten muß definiert sein -Für jedes Objekt können die Zugriffsberechtigungen einzeln angegeben werden -Aktuelle Überwachung sicherheitsrelevanter Aktionen und Benachrichtigung des Sicherheitsbeauftragten
F6:	-Systeme mit hohen Integritätsanforderungen (z.B.) Datenbanken
F7:	-Systeme mit hoher Verfügbarkeit (z.B. kritische Prozessteuerung)
F8:	-Hohe Anforderung an die Datenintegrität bei der Übertragung
F9:	-Hohe Anforderung an die Geheimhaltung bei der Übertragung
F10:	wie F8 und F9 zusammen

Tabelle 1.1: Klassifikation von Sicherheitsklassen nach funktionalen Kriterien [bsi89]

Sicherheitsklasse	Qualitätsmerkmal
Q0 :	-Unzureichende Qualität bzw. nicht geprüft
Q1 :	-Getestet, informelle Sicherheitspolitik -Spezifikation beschreibt das System allgemein
Q2 :	-Methodisch getestet
Q3 :	-Methodisch getestet und teilweise analysiert -Sicherheitspolitik ist informell und detailliert beschrieben
Q4 :	-Informell analysiert -Spezifikation und Sourcecode wird evaluiert und auf Konsistenz mit der Sicherheitspolitik überprüft
Q5 :	-Semiformal analysiert -Systemspezifikation in semiformaler Notation -Analyse mit semiformalen Methoden
Q6 :	-Formal analysiert -Sicherheitspolitik und Spezifikation sind formal gegeben und auf Konsistenz formal überprüft -Sourcecode wird sorgfältig auf Konsistenz mit Spezifikation getestet
Q7 :	-Formal verifiziert -Konsistenz des Sourcecodes mit der Spezifikation formal nachgewiesen

Tabelle 1.2: Klassifikation von Sicherheitsklassen nach Qualitätskriterien [bsi89]

IT-Sicherheitskriterien	Orange Book-Klasse
Q0	D
F1, Q2	C1
F2, Q2	C2
F3, Q3	B1
F4, Q4	B2
F5, Q5	B3
F5, Q6	A1
F5, Q7	beyond A1

Tabelle 1.3: Entsprechungen zwischen den Klassen der IT-Sicherheitskriterien und Klassen des *Orange Book* [bsi89]

1.5.4 ITSEC - *Die europäischen Kriterien*

Die Zusammenarbeit der europäischen Staaten im Bereich der IuK.-Technologien bedurfte auch einer gemeinsamen Lösung im Bereich der IT-Sicherheitskriterien, um eine Harmonisierung der Kriterienkataloge der verschiedenen Länder auf europäischer Ebene zu gewähren. Unter Beteiligung der Länder Deutschland, Frankreich, Niederlande und Großbritannien, wurde 1991 ein Kriterienkatalog, unter dem Namen *ITSEC*[2], erarbeitet.
Die Notwendigkeit der Harmonisierung wird in [its91] wie folgt begründet:

There were three reasons for harmonization:

1. *much experience had been accumulated in the various countries, and there would be much to gain by jointly building on that experience;*
2. *industry did not want different security criteria in the different countries;*
3. *the basic concepts and approaches were the same, across countries and even across commercial, government and defence applications.*

IT security means,

- **confidentiality -** prevention of the unauthorized disclosure of information
- **integrity -** prevention of the unauthorized modification of information
- **availability -** prevention of the unauthorized withholding of information or resources.

ITSEC weist viele Paralellen zu der deutschen Fassung auf, einige Aspekte wurden auch direkt übernommen. Das Wesentliche bei der Zusammenarbeit war, dass keines der beteiligten Länder größere Abweichungen zu der nationalen Fassung hinnehmen musste.
ITSEC unterscheidet zwischen Produkten und Systemen. Produkte sind Hardware- und/oder Software-Pakete, die von Herstellern angeboten werden und für verschiedene Anwendungsgebiete eingesetzt werden. Ein System besteht aus Hard- und Software-Kombinationen und ist auf eine bestimmte Anwendungsumgebung für einen bestimmten Benutzer zugeschnitten. Zur Bewertung der Sicherheit eines Systems ist die Anwendungsumgebung ausschlaggebend. Die Grundlage für die Evaluierung bilden Angaben über die Sicherheitsziele, Sicherheitsfunktionen und Sicherheitsmechanismen, sowie die gewünschte Evaluationsklasse [FH92].
Die ITSEC haben die zehn Funktionalitätsklassen der deutschen IT-Sicherheitsklassen übernommen. Sie werden in ITSEC folgendermaßen benannt : **F-C1,**

[2] Information Technology Security Evaluation Criteria

F-C2, F-B1, F-B2, F-B3, F-IN, F-AV, F-DI, F-DC, F-DX.
Zur Beschreibung und Strukturierung der Angaben über die Funktionalität eines Systems werden von ITSEC folgende Aspekte vorgeschlagen [FH92] : *Identification and Authentication, Access Control, Accountability, Audit, Object Reuse, Accuracy, Reliability of Service, Data Exchange.*
Zur Bewertung der Qualität wird geprüft, ob die Mechanismen, die zur Implementation der Funktionalität eingesetzt werden, korrekt und effektiv sind. Es gibt sieben hierarchisch geordnete *Korrektheitsklassen* E0 bis E6. Diese Klassen bauen aufeinander auf und verlangen folgende *Korrektheitskriterien* :

- **E1** ==> das Testen der Anforderungen,
- **E2** ==> die Konfigurationskontrolle, sowie einen kontrollierten Verteilungsprozess,
- **E3** ==> ein detailliertes Design und den Quellcode der sicherheitsrelevanten Funktionen,
- **E4** ==> Analyse der Verwundbarkeit des Systems,
- **E5** ==> Nachweis der Übereinstimmung des detaillierten Designs und des Quellcodes
- **E6** ==> ein formales Sicherheitsmodell, eine damit übereinstimmende formale Beschreibung der Sicherheitsarchitektur, Möglichkeit der Zuordnung von Objektcode zum Quellcode.

Zusätzlich zur Bewertung der Effektivität wird geprüft, ob die angegebenen Sicherheitsfunktionen die Sicherheitsziele erfüllen. Andernfalls erhält das System oder Produkt die Evaluations-Klasse E0 [FH92].

Die zwischen den ITSEC und TCSEC bestehenden Paralellen sind nach [its91] folgendermaßen zu sehen:

The intended correspondence between these criteria and the TCSEC classes is as follows [its91] :

These Criteria		TCSEC Class
EO	< ————— >	D
F-C1, E1	< ————— >	C1
F-C2, E2	< ————— >	C2
F-B1, E3	< ————— >	B1
F-B2, E4	< ————— >	B2
F-B3, E5	< ————— >	B3
F-B3, E6	< ————— >	A1

1.5.5 CCITSE - Common Criteria

Die europäischen Arbeitsgruppen schlossen sich 1993 mit der kanadischen (CTCPEC) und der nordamerikanischen Arbeitsgruppe (TCSEC) zusammen, und erarbeiteten die CCITSE-Spezifikation. Im Januar 1996 wurde dann ein einheitlicher, multinationaler Sicherheitsstandard verabschiedet, die *Common Criteria for Information Technology Security Evaluation* (CCITSE), auch *Common Criteria* genannt.

Die CC wurden verfaßt um sicherzustellen, dass die Prüfung und Bewertung den Bedürfnissen der Anwender entspricht :

> *Dies ist Hauptzweck und Rechfertigung des Evaluationsverfahrens* [cc99a].

Die Ergebnisse der Prüfung und Bewertung können von Anwendern dazu herangezogen werden festzustellen, ob ein evaluiertes Produkt oder System ihren Sicherheitsbedürfnissen genügt. Auch zu Vergleichszwecken von unterschiedlichen Produkten und Systemen können die Evaluationsergebnisse herangezogen werden.
Die CC beschreiben ferner Sicherheitsfunktionen, die ein Entwickler in einem **EVG** (Evaluationsgegenstand) verwenden kann. Die CC können nach [cc99a] dazu benutzt werden, die Zuständigkeiten und Handlungen bei der Erstellung der für die Prüfung und Bewertung des EVG benötigten Nachweise festzulegen, inklusive Inhalt und Form dieser Nachweise.
Die CC enthalten auch Kriterien, die von Evaluatoren zur Beurteilung der Übereinstimmung von EVG mit deren Sicherheitsanforderungen benutzt werden. Die CC beschreiben die Menge von allgemeinen Aktionen, die der Evaluator auszuführen hat, und die Sicherheitsfunktionen, mit denen die Aktionen auszuführen sind. Es wird aber kein Verfahren der Ausführung solcher Aktionen spezifiziert [cc99a].
Die CC selbst bestehen aus drei getrennten, aber inhaltlich zusammengehörigen, Teilen.

TEIL 1 - Einführung und allgemeines Modell - In Teil 1 wird zunächst eine Einführung in die CC selbst vorgenommen. Die Definition allgemeiner Konzepte und Prinzipien der IT-Sicherheitsevaluation und die Vorstellung des allgemeinen Modells der Prüfung und Bewertung stehen dabei im Vordergrund. Darüber hinaus werden Konstrukte dargestellt, die dazu dienen IT-Sicherheitsziele auszudrücken, IT-Sicherheitsanforderungen auszuwählen und zu definieren, Spezifikationen auf hoher Ebene für Systeme und Produkte zu verfassen.

TEIL 2 - Funktionale Sicherheitsanforderungen - Es wird eine Menge von funktionalen Komponenten als Mittel zur standardisierten Darstellung einheitlicher funktionaler Anforderungen an den EVG festgelegt. Teil 2 katalogisiert die Menge der funktionalen Komponenten, Familien und Klassen. Im Zusammenhang mit ID-Systemen ist Teil 2 von Bedeutung. In der Einführung von Teil 2 heißt es [cc99a]:

> „ *Funktionale Sicherheitskomponenten stellen Sicherheitsanforderungen dar, die dazu bestimmt sind, den Bedrohungen in der angenommenen Einsatzumgebung des EVG entgegenzuwirken und/oder die identifizierten organisatorischen Sicherheitspolitiken und Annahmen abzudecken.* “

Dabei ist EVG ein

> „*IT-Produkt oder- System, das über Betriebsmittel wie elektronische Speichermedien (z.B. Disketten), Peripheriegeräte (z.B. Drucker) und Rechenkapazität (z.B. CPU-Zeit) verfügt, das zum Verarbeiten und Speichern von Informationen benutzt werden kann und das Gegenstand einer Prüfung und Bewertung ist.* “

Als eine Sicherheitsanforderung könnte man z.B. die Einführung von ID-Systemen betrachten. Die Klasse *FAU-Sicherheitsprotokollierung* aus Teil 2, beschreibt im Allgemeinen Mechanismen von ID-Systemen zur Aufdeckung von Einbrüchen. Zur Sicherheitsprotokollierung gehört das Erkennen, Aufzeichnen, Speichern und Analysieren von Informationen im Zusammenhang mit sicherheitsrelevanten Aktivitäten. Die Protokollaufzeichnungen können analysiert werden, um zu ermitteln, welche sicherheitsrelevanten Aktivitäten ausgeführt wurden und welcher Benutzer dafür verantwortlich ist. Die Funktionen der Sicherheitsprotokollierung des EVG sind definiert, um sicherheitsrelevante Ereignisse zu überwachen und dienen als Abschreckung gegen Sicherheitsverletzungen. Die Anforderungen der Protokollierungsfamilien beziehen sich auf Funktionen wie *Schutz der Protokolldaten, Aufzeichnungsformat, Ereignisauswahl, Analysewerkzeuge, Verletzungsalarme* und *Echtzeitanalyse.*
Protokollierungsanforderungen in einer verteilten Umgebung wurden ebenso in Betracht gezogen, denn die Umsetzung der Protokollierungsanforderungen bei Netzen und andere große Systemen kann von der Umsetzung bei Einzelsystemen, abweichen [cc99b].

TEIL 3 - Vertrauenswürdigkeitsanforderungen - Hier wird eine Menge von Vertrauenswürdigkeitskomponenten, als Mittel zur standardisierten Darstellung einheitlicher Anforderungen an die Vertrauenswürdigkeit eines EVG, festgelegt. Unter anderem werden in Teil 3 die *Stufen* der Vertrauenswürdigkeit dargestellt, die die vordefinierte Skala der CC für die Wertung der Vertrauenswürdigkeit von EVG festlegen. Die Skala wird mit dem Ausdruck *Vertrauenswürdigkeitsstufen* (EAL) beschrieben. Es gibt acht unterschiedliche Stufen (EAL 0 bis EAL 7), die, im Unterschied zu *Orange Book* und ITSEC, in den CC folgendermaßen beschrieben worden sind [Kya98] :

- EAL 0 Keinerlei Sicherheitsstruktur

- EAL 1 Funktioneller Test:
  Sicherheitsstruktur vorhanden, definiert und funktionell getestet

- EAL 2 Struktureller Test:
  Sicherheitsstruktur mit funktioneller Spezifikation und definiertem Interface getestet

- EAL 3 Methodischer Test:
  Grey-Box-Analyse des getesteten Systems

- EAL 4 Methodisches Design und Test:
  Sicherheitsstruktur baut auf speziellem Design auf der untersten Ebene der Systemarchitektur auf

- EAL 5 Semiformales Design und Test:
  Die Analyse beinhaltet die gesamte Implementation. Die Sicherheit wird durch ein formales Modell und eine semiformale funktionelle Spezifikation gewährleistet

- EAL 6 Semiformelle Verifikation und Test:
  Modulares Sicherheitsdesign mit strukturierter Repräsentation der Implementation. Die Entwicklungsumgebung
  und das Konfigurationsmanagement entsprechen höchsten
  Ansprüchen

- EAL 7 Formelles Design und Test:
  Ein formales Modell mit einer formalen Repräsentation der funktionellen Spezifikation muss für das System vorhanden sein (White-Box-Analyse)

Kapitel 2

UNIX

2.1 Historische Entwicklung von UNIX

Ende der 60er Jahre wurden Betriebssysteme mit kurzen Antwortzeiten zum Entwicklungsziel vieler Forscher. Dies rührte aus der Tatsache, dass viele Betriebssysteme der dritten Generation, trotz ihres häufigen Einsatzes in vielen wissenschaftlichen Bereichen und in der kommerziellen Datenverarbeitung, im Kern Stapelverarbeitungssysteme waren. Der große Nachteil bei diesen Systemen war, dass mehrere Stunden zwischen dem Abschicken eines Jobs und dem Erhalten der Ausgabe, vergehen konnten, was auch die Entdeckung und Korrektur von aufgetretenen Programmierfehlern durch den Programmierer mühsam und zeitaufwendig machte. Als eine Lösung für dieses Problem wurden Timesharing-Systeme entwickelt, die als eine Variante des Multiprogrammings betrachtet werden können.

Bei Timesharing-Systemen hat jeder Benutzer on-line Zugang zum System über einen Terminal, d. h. viele Benutzer können gleichzeitig Zugang zum System haben, und die CPU wird abwechselnd ihren Jobs zugeteilt. Zu einem der ersten Timesharing-Systeme gehörte CTSS, das am MIT entwickelt wurde, aber sich nicht verbreiten konnte. Nach CTSS begann man am MIT, in Zusammenarbeit mit den Bell Labs und General Electric, mit der Entwicklung eines Systems, das Hunderte von gleichzeitigen Timesharing-Benutzern unterstützen sollte. Das System wurde unter dem Namen **MULTICS** (Multipexed Information and Computing System) bekannt. Der Bau von **MULTICS** stellte sich aber als schwerer heraus als anfänglich angenommen wurde, so dass Bell Labs aus dem Projekt ausstieg, und General Electric verließ das Rechnergeschäft gänzlich. Am **MIT** wurde weiter an **MULTICS** gearbeitet und es konnte in der Produktionsumgebung von **MIT** erfolgreich eingesetzt werden. Das anfängliche Konzept eines Rechnerwerkzeuges konnte aber nicht eingehalten werden.

Im Jahre 1969 begannen Ken Thomson, der an dem **MULTICS** -Projekt gearbeitet hatte, und Dennis Ritchie im AT&T Forschungszentrum Bell-Laboratories mit der Entwicklung eines neuen Betriebssystems. Ken Thomson benutzte einen PDP-7 Mini-Computer, um eine abgespeckte Einbenutzerversion von **MULTICS** zu schreiben. Aus diesem System entwickelte sich später das Betriebssystem UNIX, dass sich vor allem durch Mehrbenutzerfähigkeit (Multi-User) auszeichnete.
UNIX wurde in einer, von Thompson entwickelten, Sprache mit dem Namen B implementiert. Die Implementation mit B war nicht erfolgreich aufgrund einiger Schwächen dieser Sprache, insbesondere wegen des Fehlens von Strukturen. B wurde später von Dennis Ritchie zu einer Sprache namens C weiterentwickelt, für die auch ein sehr guter Compiler geschrieben werden konnte.
UNIX war günstig erhältlich und die Mitlieferung des Quellcodes sorgte für die Verbreitung dieses Betriebssystems in den Universitäten. Ein Grund für die günstige Erwerbung von UNIX war, dass AT&T (die Muttergesellschaft der Bell Labs) durch vertragliche Bindungen mit der US-Bundesregierung, die Vermarktung von UNIX untersagt wurde. Aus diesem Grund gab AT&T für Lehr- und Forschungszwecke Source-Lizenzen zu sehr günstigen Konditionen an die Universitäten weiter. Der Zugang zu den Quellcodes erlaubte den Anwendern, das System nach eigenen Wünschen abzuändern. Auf diese Weise entstand eine dynamische Entwicklung von UNIX, die auch von den Universitäten mit beeinflusst wurde. In der Folge entstand eine ganze Familie von ähnlichen Systemen. Die zwei bekanntesten Hauptstränge sind:

- **BSD :** Die an der Berkeley-Universität entstandene Berkeley-Standard-Distribution
- **System V :** Die von AT&T weitergepflegte Version

AT&T hatte im Jahre 1974 für die UNIX Version 6 auch kommerzielle Lizenzen verkauft, und ab 1984, nachdem ein weiteres Gerichtsurteil das Verbot der Vermarktung von UNIX-Software aufhob, wurde UNIX von AT&T als System V unter rein kommerziellen Gesichtspunkten verkauft. Durch die Verbreitung von UNIX an den Universitäten hat es sich auch im kommerziellen Sektor schnell durchgesetzt. Dies führte aber dazu, dass AT&T aus Konkurrenzgründen den Zugang zum Quellcode untersagte und tiefgehende Öffnungen auf der Systemebene abblockte. Die Eigendynamik der Entwicklung des Betriebssystems verlor durch die neue Lizenzpolitik von AT&T ihren Antrieb. Die heutige UNIX-Funktionalität entstand nicht nur durch die alleinige Arbeit der AT&T-Entwickler. Außerhalb von AT&T gab es viele Unix-Benutzer, die das Produkt benutzten, erweiterten und ihre Ergebnisse an AT&T weiterreichten. Für viele kleine Firmen und Universitäten waren aber die neuen Lizenzierungsrichtlinien für den Quellcode nicht mehr tragbar. Die Folge war, dass viele Gruppierungen UNIX-ähnliche Betriebssysteme ohne AT&T Quellcode entwickelten. Diese Entwicklung hatte den Nachteil, dass die verschiedenen UNIX-Derivate nicht mehr kompatibel zueinander waren. Abbildung 2.1 zeigt eine zeitliche Entwicklung der UNIX-Derivate.

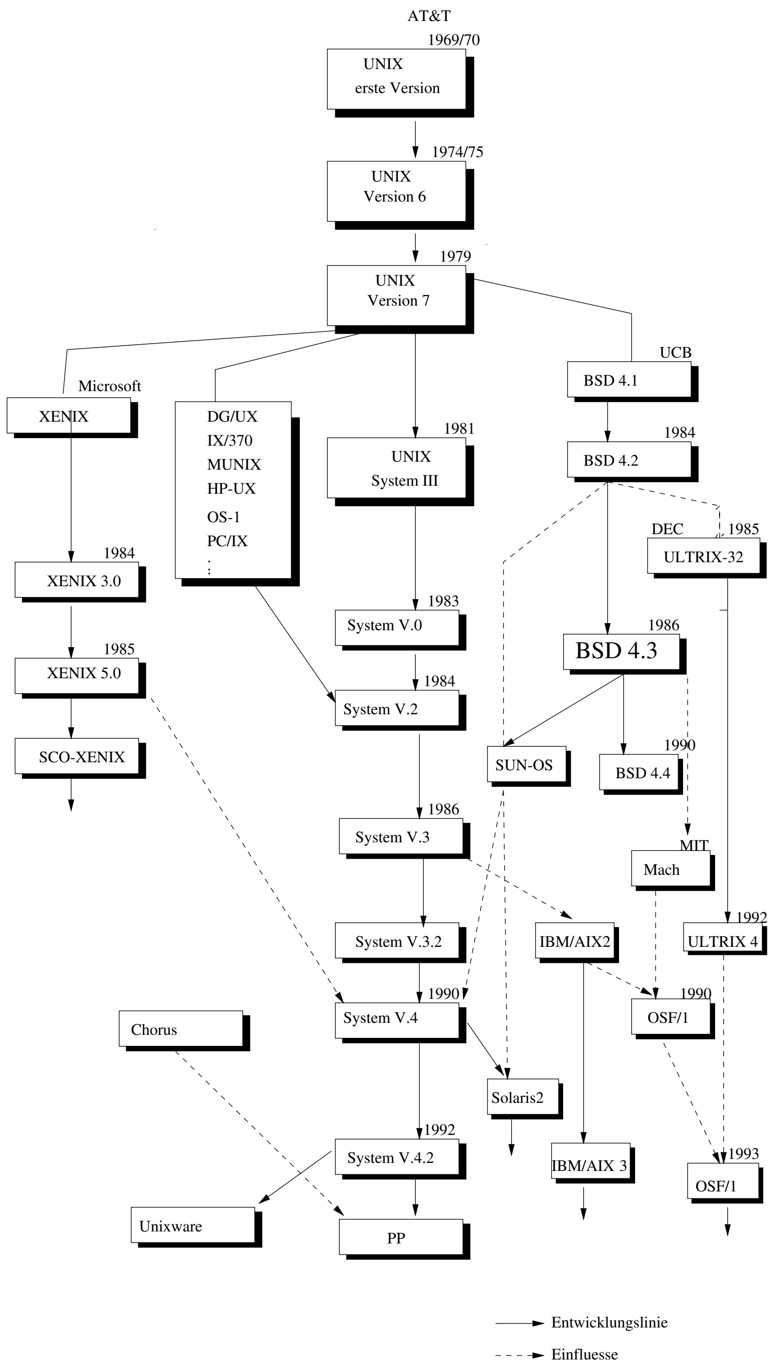

Abbildung 2.1: Entwicklungsrichtungen verschiedener UNIX-Systeme [J.G95]

2.2 Das UNIX-System

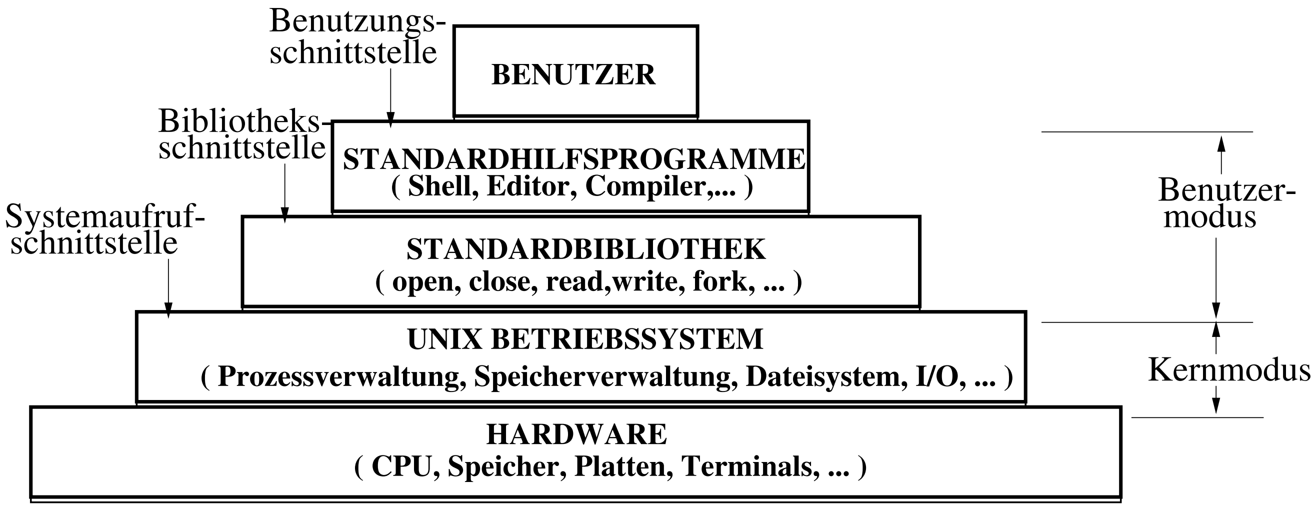

Abbildung 2.2: Die Schichten eines UNIX-Systems [Tan95]

Das UNIX-System kann nach [Tan95] als eine Pyramide dargestellt werden, die aus fünf Schichten besteht:

Schicht 1, die Hardware : Die Hardware besteht aus physikalischen Geräten wie CPU, Speicher, Platten, Terminals, Drucker usw.

Schicht 2, das Betriebssystem UNIX : Schicht zwei baut auf Schicht eins auf. In dieser Schicht werden Prozess- und Speicherverwaltung geregelt, sowie die Verwaltung des Dateisystems und der Ein- und Ausgabe. Das Betriebssystem steuert und kontrolliert somit die Hardware, und stellt allen Programmen eine Systemaufrufschnittstelle zur Verfügung.

Schicht 3, die Standardbibliothek : Hier werden Prozeduren für Systemaufrufe bereitgestellt (je eine Prozedur pro Systemaufruf), mit deren Hilfe der Trap-Befehl ausgeführt werden kann. Die Ausführung des Trap-Befehls ist bei Systemaufrufen für die Umschaltung aus dem Benutzermodus in den Kernmodus nötig, um die Funktionen des UNIX-Betriebssystems auszuführen. Die Programmiersprache C bietet aber keine Möglichkeit dafür, einen Trap-Befehl unmittelbar nach einem Systemaufruf aufzurufen. Da die Standardbibliotheksprozeduren in einer Assembler-Sprache geschrieben sind, können sie von C aus aufgerufen werden, um den Trap-Befehl auszuführen.

Schicht 4, die Standardhilfsprogramme : Hier sind der Kommando-Prozessor (Shell),der Compiler, die Editoren, die Textverarbeitungs- und Hilfsprogramme für die Verarbeitung von Dateien, angesiedelt. Diese Anwenderprogramme stellen die Benutzungsschnittstelle dar. Zu bemerken ist, dass die Menge der Standardprogramme nicht Teil des Betriebssystems sind und leicht ersetzt werden können.

Schicht 5, die Benutzer : Diese Schicht bezeichnet die Benutzer, die an ihren Terminals die Programme aufrufen.

2.2.1 Der Kernel

Die Hauptfunktionsblöcke von UNIX sind der Kernel, die Shell und das Dateisystem. Der Kernel ist der zentrale Programmteil von UNIX, der direkt mit der Hardware kommuniziert. Er ist für folgende Aufgaben zuständig :

- Steuerung der CPU-Aktivitäten,
- Steuerung der Ein- und Ausgabeaktivitäten,
- Steuerung der Netzwerkfunktionen,
- Überwachung der Zugriffsrechte auf das Dateisystem,
- Ausführung der Shell-Befehle.

Der Zugriff auf den Kernel erfolgt durch Systemaufrufe, die entweder von UNIX-Anwendungen oder direkt über die Shell ausgeführt werden.

Die interne Struktur des UNIX Kerns weist zwei Teile auf, die sich voneinander unterscheiden:

1. **Maschinenabhängiger Kern von UNIX :** Das ist die unterste Ebene des Kerns von UNIX. Der Code besteht nach [Tan95] aus:

    - Interrupt-Behandlung
    - Gerätetreiber des unteren Ein-/Ausgabesystems
    - Einem Teil der Speicherverwaltungs-Software

    Der größte Teil ist in C geschrieben, der aber neu implementiert werden muss, wenn UNIX auf eine neue Maschine portiert wird. Dies resultiert aus der Tatsache, dass mit diesem Teil des Kernels unmittelbar die Hardware betrieben wird.

2. **Maschinenunabhängiger Kern von UNIX :** Dieser Teil des Kernels hängt nicht unmittelbar von der speziellen Hardware ab, auf der er zur Ausführung kommt. Der Code enthält Anweisungen für:

- Die Behandlung der Systemaufrufe
- Prozeßverwaltung
- Scheduling
- Pipes
- Signale
- Paging
- Swapping
- Dateisystem und
- Den höheren Teil des Ein-/Ausgabesystems

2.2.2 Dateisystem in UNIX

Ein wesentlicher Bestandteil eines UNIX-Systems sind Dateien. Jeder Zugriff auf Programme und Daten ist als Dateizugriff realisiert, dies gilt auch für jede Form der Ein- oder Ausgabe. Wird ein Zeichen von der Tastatur gelesen, so ist dies der Lesezugriff auf eine Spezialdatei, z.B */dev/kbd* (keyboard), oder wenn etwas über den Drucker ausgegeben wird, so ist dies ein schreibender Dateizugriff auf die Spezialdatei, */dev/lp0* (line printer 0). Spezialdateien werden in Abschnitt 2.2.3, unter dem Aspekt *Dateiattribute*, näher behandelt .

Eine normale Datei in UNIX ist eine Folge von 0 oder mehr Bytes, die beliebige Informationen enthalten können. Das System macht dabei keinerlei Annahmen über die in den Bytes gepeicherten Informationen. Es wird auch kein Unterschied zwischen ASCII-Dateien, Binär-Dateien oder anderen Arten von Dateien gemacht. Dateien sind durch einen 9-Bit-Modus, der einer jeden Datei zugewiesen wird, geschützt.

Die ersten drei Bits legen die Zugriffsrechte des Eigentümers auf die Datei fest. Die nächsten drei Bits bezeichnen die Rechte für die Mitglieder einer Gruppe, in der sich der Eigentümer befindet. Die letzten drei Bits sind für alle anderen Benutzer reserviert. Abb. 2.3 zeigt ein Beispiel für gesetzte bzw. verwehrte Rechte (dargestellt als Balken). Das erste Feld in der Abbildung bezeichnet den Typ einer Datei auf den die gesetzten Rechte bezogen sind.

-	r	w	x	r	-	-	r	-	-
Typ	Eigentümer			Gruppe			Andere		

Abbildung 2.3: Zugriffsrechte

Die Zugriffsarten, die durch die Bits dargestellt werden, sind:

- Lesen (read)

- Schreiben (write)
- Ausführen (execute)

2.2.2.1 Verzeichnisse

Das Dateisystem ist als Baumstruktur aus Verzeichnissen (Gruppen von Dateien) und Dateien aufgebaut, die der besseren Übersicht dient. Die Verzeichnisse werden dabei selbst als Dateien gespeichert und weitgehend auch als solche behandelt, d. h. der Zugriff auf die Verzeichnisse wird ebenfalls mit denselben neun Bits geschützt. Anstelle die lesenden, schreibenden und ausführenden Zugriffe auf die Dateien zu kontrollieren, wird aber hierbei das Recht auf das Suchen in einem Verzeichnis erteilt oder verwehrt. Im folgenden Abschnitt werden die Komponenten des Dateisystems ausführlicher beschrieben und Abbildung 2.4 zeigt die Baumstruktur des Dateisystems.

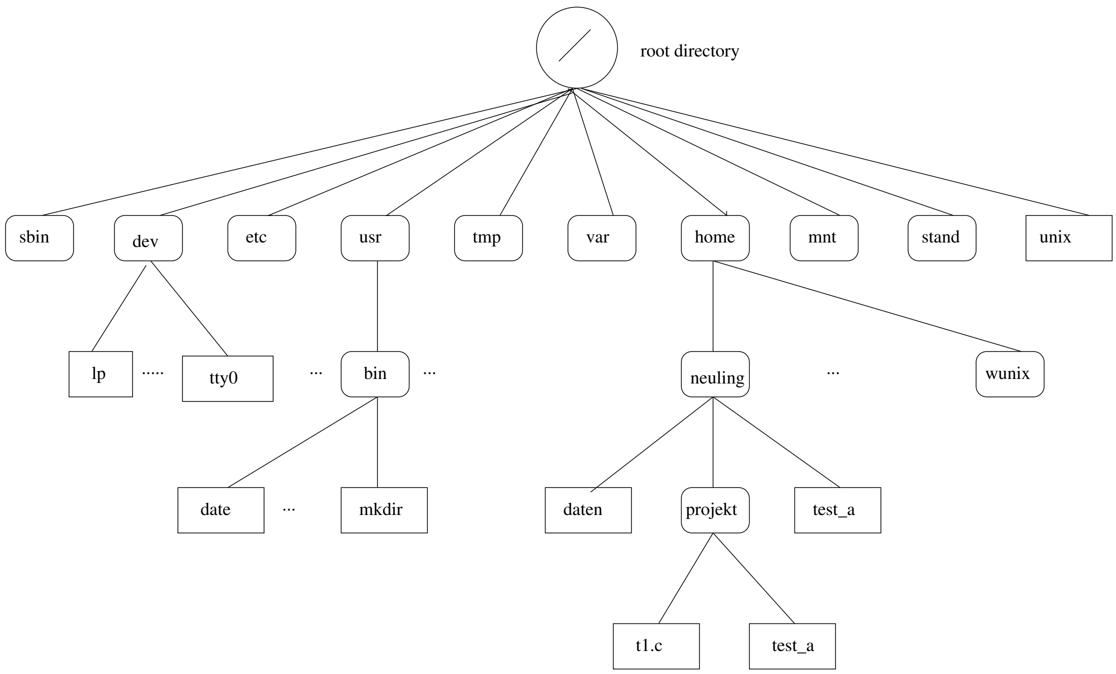

Abbildung 2.4: Beispiel für einen Dateibaum [J.G95]

2.2.3 Die Komponenten des Dateisystems

Die UNIX-Dateisysteme bestehen aus mehreren unterschiedlichen Komponente. Diese sind in der Abb. 2.5 aufgeführt.

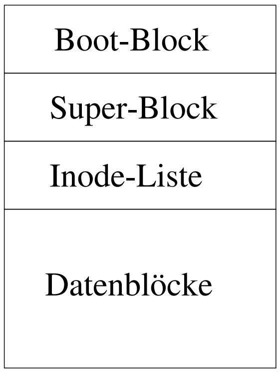

Abbildung 2.5: Aufbau des Dateisystems [J.G95]

Der Boot-Block :

Die erste Komponente eines Dateisystems ist als *boot block* reserviert und befindet sich am Anfang eines Dateisystems. Dieser enthält das Initialisierungsprogramm, welches beim Hochfahren des Systems das eigentliche UNIX-System in den Hauptspeicher lädt und startet.

Der Super-Block :

Der Super-Block des Dateisystems ist von zentraler Bedeutung, weil in diesem die gesamten Verwaltungsinformationen des Dateisystems stehen. Hierzu gehören die Angaben über die maximale Anzahl von Dateien, die das System auf dem Laufwerk speichern kann (Inode-Tabelle), die Größe des Dateisystems, die Anzahl der noch freien Inodes, die Größe und Position des freien Speicherplatzes (siehe 2.6).

Die Inode-Liste :

Beim Erstellen des Dateisystems wird die Inode-Liste aufgebaut und gespeichert. Sie stellt das Inhaltsverzeichnis dar. In ihr sind alle Dateien verzeichnet, die in dem Dateisystem existieren. Die Inodes sind Elemente der Inode-Liste. Jeder Datei wird genau ein Inode zugeordnet, der mit einer eindeutigen Nummer gekennzeichnet ist. Ein solcher Inode stellt den Dateikopf dar, der alle Attribute enthält, die einer Datei zugeordnet werden.

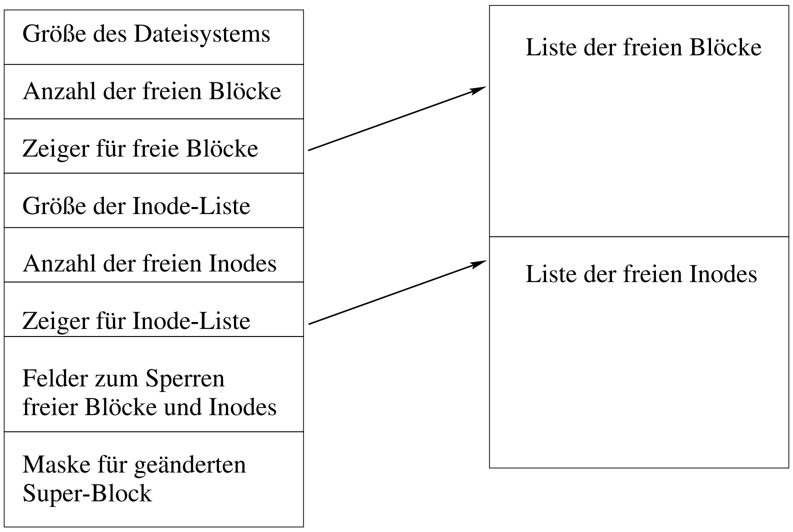

Abbildung 2.6: Der Super-Block [J.G95]

Die Datenblöcke :

Ein Datenblock stellt einen zusammenhängenden Adressbereich auf der Festplatte dar. Es können immer nur ganze Datenblöcke reserviert werden. Die Datenblockgröße ist im Allgemeinen implementationsabhängig und wird bei einigen Systemen im Super-Block angezeigt.

Die Dateiattribute :

Die Dateiattribute werden in [J.G95] folgendermaßen dargestellt:

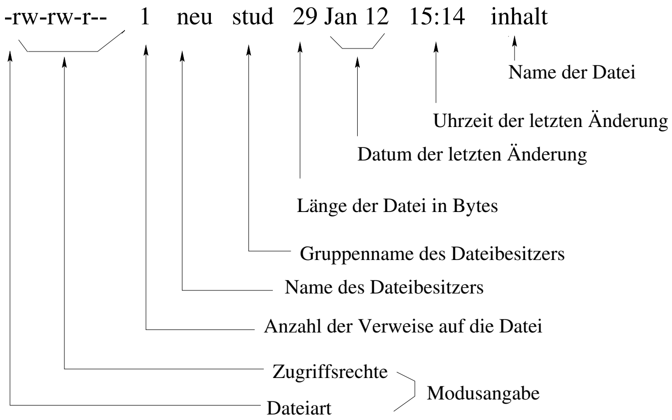

Abbildung 2.7: Dateiattribute von UNIX

Die Dateien unter UNIX kann man in drei Kategorien unterteilen :

1. **Normale Dateien :** Dateien, die lediglich als Byte-Folgen interpretiert werden.

2. **Verzeichnisse (directories) :** Das sind Dateien, die entweder leer sind, oder Verweise auf weitere Dateien enthalten. Die Verweisstruktur ist hierarchisch aufgebaut.

3. **Spezialdateien :** In UNIX gibt es zwei Kategorien von Spezialdateien:

   - *Blockorientierte Spezialdateien :* Diese Dateien bestehen aus einer Folge von numerierten Blöcken. Hauptmerkmal der blockorientierten Spezialdateien ist, dass jeder Block individuell adressiert und auf jeden individuell zugegriffen werden kann. Ein Programm kann eine solche Spezialdatei öffnen und beispielsweise auf den Block 88 zugreifen, ohne zuvor die Blöcke 0-87 gelesen zu haben. Blockorientierte Spezialdateien werden meist für Platten verwendet.
   - *Zeichenorientierte Spezialdateien :* Diese Dateien werden im Allgemeinen für Geräte verwendet, die einen Zeichenstrom ein- bzw. ausgeben. Beispiele für solche Geräte sind Terminals, Drucker, Netzwerke, Mäuse, Plotter. Auf zeichenorientierte Spezialdateien kann nicht wahlfrei zugegriffen werden, sie unterstehen aber einer häufigen Kontrolle und Steuerung, die es für blockorientierte Spezialdateien nicht gibt.

Bei UNIX werden alle Geräte als Spezialdateien betrachtet und sind somit in das Dateisystem integriert. Jedem Ein-/Ausgabegerät wird ein Pfadname zugeordnet. Geräte sind meist in dem Verzeichnis /dev untergebracht. Der Drucker kann z.B. als /dev/lp benannt werden. Vorteile dieser Lösung sind:

- Zugriff auf diese Spezialdateien kann auf die gleiche Art und Weise erfolgen, wie auf alle anderen Dateien

  ==> spezielle Kommandos oder Systemaufrufe sind nicht nötig.
  ==> spezieller Mechanismus für die Ein-/Ausgabe ist nicht nötig.

- Die allgemeinen Dateischutzregeln können auch automatisch auf die Ein-/Ausgabe-Geräte angewandt werden.

- Durch die Integration der Ein-/Ausgabe-Geräte in das Dateisystem, wird eine große Flexibilität beim Zugriff erreicht.

2.2.4 Prozess

In Unix-Systemen stellen die Prozesse die Handelnden, also die Subjekte, dar. Prozesse sind die einzigen aktiven Einheiten in UNIX-Systemen. Die Instruktionen eines Prozesses werden sequentiell ausgeführt. In UNIX-Systemen können mehrere, unabhängige Prozesse gleichzeitig in Ausführung sein. Jeder Prozess wird mit einer **PID** (Prozessidentifikator) versehen. Außerdem gehören zu einem Prozess noch die **GID** (Gruppenidentifikator) und die **UID** (Benutzeridentifikator) desjenigen (Benutzers oder Prozesses), der den Prozess gestartet hat.

Die Erzeugung eines Prozesses kann nach [Tan95] folgendermaßen beschrieben werden:

- Systemaufruf *FORK* erzeugt eine exakte Kopie des orginalen Prozesses. Der Systemaufruf, der *FORK* aufruft, heißt *Vaterprozess*.
- Der von dem Vaterprozess erzeugte Prozess wird als *Sohnprozess* bezeichnet.
- Vater und Sohn haben verschiedene private Speicherabbilder, d.h nachfolgende Veränderungen an den Variablen des Vaters sind für den Sohn nicht sichtbar (gilt auch für die Umkehrung).
- Geöffnete Dateien werden von Vater und Sohn gemeinsam benutzt.
- Veränderungen an diesen Dateien, die von einem der beiden gemacht werden, sind für den jeweils anderen sichtbar.
- Die Prozesse können Vater und Sohn voneinander unterscheiden, weil der Systemaufruf *FORK* dem Sohn eine 0 als Ausgabe liefert und dem Vater einen Wert ungleich 0, d.h., er ordnet ihm die PID zu.
- Ein Vater kann mehrere Söhne haben. Die Kinder können wiederum Kinder haben, so dass ein, als Ausgangspunkt betrachteter Prozess, einen ganzen Baum von Kindern, Kindeskindern bilden kann, der eine hierarchische Struktur aufweist. Hierin spiegelt sich die Funktionsweise von UNIX als Timesharing-System (zur näheren Beschreibung der obigen Verhältnisse soll an dieser Stelle auf [Tan95] verwiesen werden) wider.

UNIX-Prozesse kommunizieren miteinander über gewisse Kanäle, wobei die Kommunikation die Form eines Nachrichtenaustausches aufweist. Im Falle einer Kommunikation zwischen zwei Prozessen, schreibt einer der Prozesse in die erzeugten Kanäle einen Strom von Bytes, die der andere lesen kann. Die Kanäle werden als **Pipes** bezeichnet. Eine andere Möglichkeit der Kommunikation wird durch Softwareunterbrechungen ermöglicht. Dabei sendet ein Prozess ein sogenanntes **Signal** an einen anderen Prozess. Das System kann, auf Anweisung der Prozesse hin, mit den Signalen auf verschiedene Weise verfahren. Das System kann

- die Signale ignorieren
- sie abfangen
- oder das Signal den Prozess zerstören lassen.

2.2.5 Benutzer-Identifikation

Bei Unix unterscheidet die Identifikations- und Authentifikations-Einrichtung zwischen Gruppen (von Benutzern) und Benutzern. Jeder Benutzer wird in UNIX durch einen systemweit eindeutigen Namen und eine (nicht notwendige) Benutzerkennung **UID** (User Identification) identifiziert. Außerdem gibt es noch die Möglichkeit, Gruppen zu definieren. Jeder Gruppe ist eine ID, die sog. **GID** zugeordnet. Mehrere Benutzer können in einer Gruppe sein, und ein Benutzer kann mehreren Gruppen angehören. Benutzerspezifische Daten stehen in der für alle offen zugänglichen Datei **/etc/passwd**. Informationen über Gruppen stehen in der Datei **/etc/group**.

In allen Unixsystemen gibt es einen Benutzer, den man bei der Neueinrichtung eines Systems nicht selber anlegen muss: Den Benutzer *root*. Dieser Benutzer ist mit dem Privileg ausgestattet, alles im System (auch die Vernichtung des Systems selber) tun zu dürfen. Er besitzt die Benutzer-Identifikation 0. Ihm ist es erlaubt, alle Dateien zu lesen oder zu beschreiben. Das Anlegen von weiteren Benutzern fällt in den Zuständigkeitsbereich des *root*. Bei dem Einrichten von Benutzerkonten kann man den Benutzern Rechte einräumen, indem man sie Gruppen zuordnet. Jeder Benutzer hat dann während seiner Arbeit im System das Recht, die Zugriffsmöglichkeiten auf ihm gehörende Daten selber zu regeln, indem er selbst die Lese- und Schreibrechte der Dateien setzt. Prozesse mit der Identifikation 0 können eine kleine Anzahl von geschützten Systemaufrufen durchführen, die normalen Benutzern nicht erlaubt ist. Die Fähigkeiten des Systemadministrators können von Benutzern zu Angriffszwecken missbraucht werden.

Es gibt in UNIX die Möglichkeit, das Prozesse ihre UID ändern. Das Betriebssystem stellt hierfür eine Schnittstelle, in Form der Systemfunktion *setuid* , zur Verfügung. Diese darf immer dann von einem Prozess ausgeführt werden, wenn die Datei, von der aus der Prozess gestartet wurde, das **s**-Bit (set userid-Bit) in ihren Ausführungsrechten gesetzt bekommen hatte. Das **s**-Bit kann vom System gesetzt werden, wenn Benutzer z.B. Informationen über die Größe des freien Speicherplatzes einer Platte in Erfahrung bringen wollen. Üblicherweise ist es den Benutzern verboten, die Rohdaten unmittelbar zu lesen. Der Systemadministrator könnte unter Herausgabe seines Passwortes die Informationen herausfinden, dies hätte aber zur Folge, dass das gesamte Schutzsystem aufgehoben wäre. Zur Vermeidung dieser Gefahr, setzt der Systemadmin das **s**-Bit für Programme, mit deren Hilfe die Benutzer Lesezugriffe auf die Platte ausführen können. Aber das Setzen des **s**-Bits stellt ein Sicherheitsrisiko dar. Es ist möglich, dass ein fehlerhaft implementiertes Programm einem einfachen Benutzer plötzlich Systemadministratorrechte (*root privileges)* einräumt. Dies steht auch in Verbindung mit der UID und GID. Die UID und die GID dienen zwei Zwecken :

1. Der Zuordnung von Prozessen zu Benutzern oder Benutzergruppen.

2. Die Überprüfung der Zugriffsrechte.

Von Bedeutung ist in diesem Fall die Überprüfung der Zugriffsrechte.
Ein Prozess besitzt zwei Arten von Benutzer- und Gruppennummer:

- Die **effektive** Benutzer- und Gruppennummer und
- Die **reale** Benutzer- und Gruppennummer

Die **effektive** Benutzer- und Gruppennummer dient der Überprüfung der Dateizuriffsrechte, und die **reale** Benutzer- und Gruppennummer ist jeweils die Nummer, die der aufrufende Benutzer besitzt. Ruft ein Benutzer ein Programm auf, für das er selbst Ausführungsrechte besitzt, so wird beim Programmstart die **effektive** und die **reale** Benutzer- und Gruppennummer des aufrufenden Benutzers zugrunde gelegt. Ist aber an der Stelle des Ausführungsrechtes statt des **x** ein **s** (set user ID) gesetzt, dann wird zum Zeitpunkt der Programmausführung die Benutzernummer des Programmbesitzers als **effektive** Benutzernummer eingesetzt, und als **reale** die des Aufrufers. Auf diese Weise lassen sich Veränderungen an geschützten Dateien kontrolliert durchführen, ein Beispiel ist die Datei **/etc/passwd**.

2.2.6 Zugangs- und Zugriffskontrollen

2.2.6.1 UNIX Login

Während des Startvorganges des Betriebssystems (*boot sequence*) erzeugt der Betriebssystemkern (*kernel*) den ersten Prozess *init*, der die Wurzel in der baumartigen Prozessstruktur darstellt. *Init* startet unter anderem ein *getty*genanntes Programm, dass die Verbindung vom Betriebssystem mit einem Terminal aufbaut, oder auf eine solche Verbindung wartet. Außerdem druckt es den *login prompt* auf den Bildschirm. Die Prozesse *init* und *getty* haben *root*-Privilegien. Der Prozess *getty* hat die *root*-Privilegien dabei von dem *init*-Prozess geerbt. Diese Prozessvererbung (*process inheritance*) ist eines der Hauptmerkmale von UNIX-Betriebssystemen. Gibt ein Benutzer seinen Namen ein, so startet der *getty*-Prozess das Loginprogramm und übergibt ihm den eingetippten Namen als Zeichenkette. *Login* fragt daraufhin nach dem Passwort. Hat der Benutzer das richtige Passwort eingegeben, so aktualisiert *login* die Dateien **utmp** und **wtmp**. In der ersten Datei stehen die augenblicklich eingeloggten Benutzer, und in der zweiten stehen die Logingeschichten (login histories), also die Logins der jüngsten Vergangenheit. Nun ändert der *login*-Prozess, durch entsprechende Aufrufe von Routinen des Betriebssystemkerns, die *UID* und *GID* so, wie sie für den Benutzer vom Systemadministrator angelegt wurden. Die nötigen Informationen findet der *login*-Prozess in den schon erwähnten Dateien /etc/passwd, /etc/group. Am Ende ersetzt sich der *Login*-Prozess durch das sog. *shell*-Programm.

2.2.6.2 UNIX Password Mechanism

Die Gültigkeit eines Benutzerpasswortes muss bei jedem erneuten Einmelden (*login*) in das System durch das System bestätigt werden. Die Authentisierungsprozedur involviert zwei Parteien (*two-party authentication*), Benutzer und Rechner. Da der Rechner sich dem Benutzer gegenüber nicht authentisieren muss, ist es außerdem ein Einwegverfahren (*unilateral, oneway authentication*). Das Passwort stellt somit das Geheimnis dar, das Computer und Benutzer teilen. Anstelle des verschlüsselten Passwortes jedoch speichert das System einen Wert, der aus dem Passwort berechnet wird. Dieser Wert wird *cryptographic hash* genannt. Die Funktion zur Berechnung dieses *hash*-Wertes ist eine Einwegfunktion, d. h. sie ist surjektiv und nicht injektiv. Auf diese Weise ist gesichert, dass eine Rückrechnung vom *hash* zum Passwort nur sehr schwer, wenn nicht gar unmöglich ist. D. h. die Umkehrung der Berechnungsschritte ergibt eine Funktion, die eine Abbildung aus der Menge der *hash*-Werte in die Potenzmenge der zulässigen Passwörter und nicht in die Menge der Passwörter erlaubt. Dies beinhaltet natürlich die Schwäche, dass es vorkommen könnte, dass zwei verschiedene Passworteingaben den gleichen *hash*-Wert ergeben. Um dies auszuschließen, erweitert man die Menge der zulässigen Passwörter auf Werte, die nicht vom Benutzer eingegeben werden können. Man schafft so eine Abbildung, bei der es nahezu unmöglich ist, dass zwei verschiedene Eingaben den gleichen *hash*-Wert ergeben. UNIX benutzt den *Data Encryption Standard* (DES). Das Passwort wird also auf zwei Arten vor unerlaubten Zugriffen geschützt:

- Durch die Zugriffskontrolle des Systems: Nur *root* darf die Datei mit den *hash*-Werten lesen
- Durch die Benutzung eines Verschlüsselungsalgorithmus'.

Der Computer berechnet also bei jedem *login* aus dem eingegebenen Passwort einen Wert und vergleicht diesen mit einem Wert, der in einer nur dem privilegierten Superuser zugänglichen Datei gespeichert ist. Diese Datei ist eine Tabelle, in der Paare von Benutzernamen und *hash*-Werten stehen.

2.2.6.3 ACLs und Capabilities in UNIX

In Kapitel 1 wurden Zugriffskontrolllisten (ACLs) und Capabilities behandelt, die in Zusammenhang mit der Zugriffsmatrix stehen. Oft wird die Zugriffsmatrix nicht unmittelbar gespeichert, weil sie in der Regel groß und nicht ausreichend besetzt ist. Die Speicherung einer großen, aber im Wesentlichen leeren, Matrix würde viel Plattenplatz verbrauchen. Praktischer und effektiver ist, die zeilen- oder spaltenweise Speicherung der nicht leeren Felder. Die spaltenweise Speicherung wird durch die ACLs, und die zeilenweise Speicherung durch die Capabilities verwirklicht.

ACLs : ACLs werden in UNIX nicht in der Form benutzt, wie sie in Kapitel 1 beschrieben wurden. Zunächst soll aber von der Annahme ausgegangen werden, dass ACLs in UNIX benutzt werden, weil Zugriffskontrolllisten im Rollenmodell von UNIX konkret benutzt werden, und somit die Annahme nicht ganz abwägig ist. Zur Erläuterung wird folgend ein Beispiel, dass in [Tan95] benutzt wurde, in modifizierter Form vorgestellt.

Beispiel :

Das Beispiel sieht folgende vier Benutzer vor :

1. *Jan* : Gehört zu Gruppe *system*

2. *Sandra* : Gehört zu Gruppe *staff*

3. *Karl* : Gehört zu Gruppe *student*

4. *Ellen* : Gehört zu Gruppe *student*

Für eine Datei soll die folgende Zugriffskontrollliste gegeben sein :

File 0 :	(***Jan , * , r w x***)
File 1 :	(***Jan , system , r w x***)
File 2 :	(***Jan , * , r w -***) ,
	(***Sandra, staff, r - -***) ,
	(***Ellen , * , r w -***)
File 3 :	(****, student , r - -***)
File 4 :	(***Karl , * , - - -***)

)

Abbildung 2.8: Beispiel für eine Zugriffskontrollliste (in Anlehnung an [Tan95])

Es soll an dieser Stelle noch einmal daran erinnert werden, dass jeder Eintrag in der Zugriffskontrollliste als ein Tupel der Form

< *Subjekt-Identifikation Si, Rechte des Si* >

oder

$<$ *GruppenId-Gi, Rechte von Gi* $>$

dargestellt wurde (vgl. 1.2.1.2). Wir fassen hier diese beiden Tupel in einen Tripel der folgenden Form zusammen :

$<$ Subjekt-Identifikation Si, GruppenId-Gi, Rechte des Si und Gi $>$

Ein * in der obigen Tabelle steht als Platzhalter für alle Si's bzw. Gi's (bei UNIX UID bzw. GID genannt).
Man kann nun die Tabelle 2.9 wie folgt beschreiben :

File 0 :	***Kann von jedem Prozess gelesen, beschrieben oder ausgeführt werden, dessen Si = Jan ist, mit beliebiger Gruppenidentifikation.***
File 1 :	***Kann nur von Prozessen benutzt werden , deren Si = Jan und deren Gi = system ist. Das bedeutet, dass ein Prozess mit der Si = Jan und der Gi = staff auf File 0 zugreifen kann, aber nicht auf File 1.***
File 2 :	***Kann von jedem Prozess mit der Si = Jan und einem beliebigen Gi gelesen oder beschrieben, von Prozessen mit der Si = Sandra und der Gi = staff gelesen, von Prozessen mit der Si = Ellen und einer beliebigen Gi gelesen oder beschrieben werden.***
File 3 :	***Kann von allen Studenten gelesen werden***
File 4 :	***Kein Prozess mit der Si = Karl und aus einer beliebigen Gi darf auf File 4 zugreifen. Alle anderen Studenten dürfen diese Datei aber lesen.***

Abbildung 2.9: Beschreibung der Zugriffskontrollliste (in Anlehnung an [Tan95])

Interessant sind hier die gesetzten Rechte bei File 4. Man kann durch die Verwendung der ACLs einzelnen Benutzern oder Gruppen den Zugriff auf Dateien verwehren, während allen anderen der Zugriff erlaubt wird.

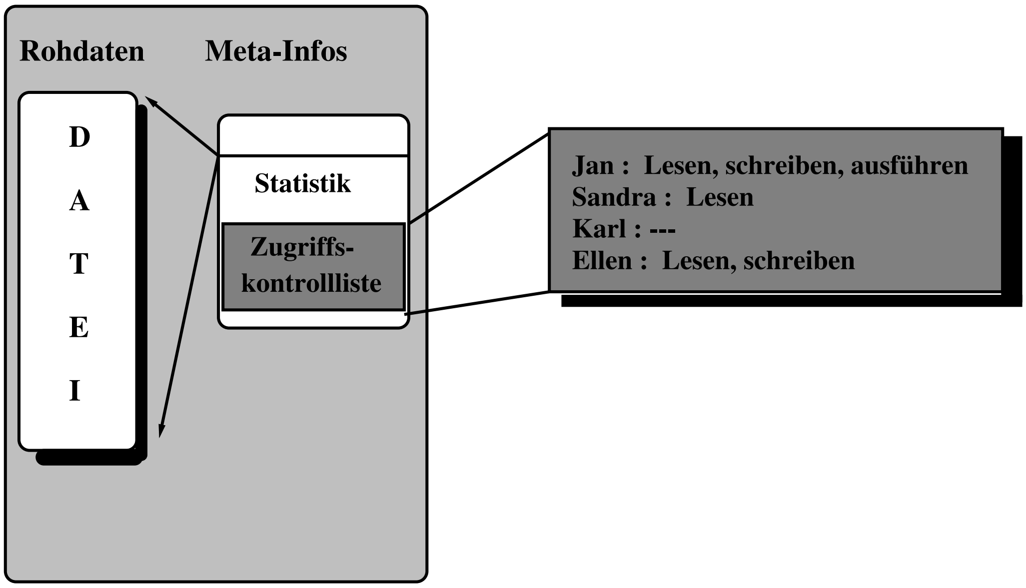

Abbildung 2.10: Implementierung der Zugriffskontrollliste am Beispiel einer Unix-Datei ([Kü96] (modifiziert)

Eigentümer	**r+**	**w+**	**x+**
Gruppe	**r+**	**w-**	**x-**
Rest d. Welt	**r-**	**w-**	**x-**

Implementierung : kodiert in 16-Bit-Wort

							+	+	+	+	-	-	-	-	-

Abbildung 2.11: Die Unix-Zugriffskontrollliste [Kü96] (modifiziert)

In UNIX wird die Zugriffskontrollliste etwas anders implementiert als oben angenommen und beschrieben. Es wurde zuvor erwähnt, dass UNIX für die Regelung der Zugriffsrechte einen 9-Bit-Modus verwendet, mit jeweils drei Bits, **rwx**, für Eigentümer, Gruppe des Eigentümers und alle anderen. Diese Art der Implementierung stellt ebenfalls eine Zugriffskontrollliste dar, aber komprimiert in 9 Bits. Die Liste gibt an, welches Subjekt auf welche Art auf ein Objekt zugreifen darf. Dieses Schema ist einfacher und kostengünstiger zu implementieren. Außerdem kann der Eigentümer eines Objektes die Zugriffskontrollliste, wenn gewünscht, ändern, so dass es einfach ist, einen zuvor erlaubten Zugriff aufzuheben. Ein Nachteil ist aber, dass die Änderung der ACL die Benutzer, die das Objekt bereits benutzen, nicht betrifft.

Capabilities : In Kapitel 1, Abschnitt 1.2.1.2, wurden Capabilities eingeführt und kurz beschrieben. Hier sollen sie etwas genauer, und in Zusammenhang mit UNIX erläutert werden .
Die Zugriffsmatrix wird bei dem Capability-Ansatz durch eine zeilenweise Speicherung aufgespalten. Jedem Prozess wird eine Liste mit den Objekten, auf die er zugreifen kann, zugeordnet. Auch die Art der zulässigen Operationen wird durch die Liste zugeordnet. Diese Liste ist die Capability-Liste und jedes einzelne Element dieser Liste wird Capability genannt.

	Typ	**Rechte**	**Objekte**
0	**Datei**	**R - -**	**Zeiger auf Datei 3**
1	**Datei**	**RWX**	**Zeiger auf Datei 4**
2	**Datei**	**RW -**	**Zeiger auf Datei 5**
3	**Drucker**	**- W -**	**Zeiger auf Drucker 1**

Abbildung 2.12: Eine Capability-Liste [Tan95]

Jede Capability einer Capability-Liste besitzt folgende drei Felder :

1. *Typ*-Feld : Hier wird die Art des Objektes angegeben
2. *Rechte*-Feld : Hält die zulässigen Operationen dieses Typs von Objekten fest
3. *Objekt*-Feld : Enthält einen Zeiger auf das Objekt selbst

Ein Prozess, dem eine Capability-Liste zugeordnet wurde wie sie in Abb. 2.12 dargestellt ist, kann z.B auf das Objekt *Datei 3*, vom Typ *Datei*, nur lesend oder auf das Objekt *Drucker 1*, vom Typ *Drucker*, nur schreibend zugreifen.

2.3 Entwicklung von TCP/IP

Mitte der 60er Jahre versuchte das amerikanische Verteidigungsministerium (DoD - Department of Defense) das Problem zu lösen, die militärische Kommunikation in einem eventuellen Kriegsfall aufrechtzuerhalten, selbst wenn große Teile des Telefon- und Kommunikationsnetzes ausfallen sollten. Eine Unterabteilung des amerikanischen Verteidigungsministeriums wurde daraufhin damit beauftragt ein neuartiges Computernetzwerk zu entwickeln, welches dieses Problem lösen sollte.
In den Projekten, die zu diesem Zweck gestartet wurden, versuchten die Entwickler ein Netzwerk zu entwerfen, das im Gegensatz zu den damals vorhandenen Systemen, keine direkte Verbindung zwischen Sender und Empfänger vorsah. Die Vorstellung dabei war, Nachrichten in einzelne Datenpakete aufzuteilen, jedes Paket mit einer Adresse zu versehen und es sich seinen Weg durchs Netz suchen zu lassen. Die Route, über die die einzelnen Teilpakete weitergereicht wurden, sollte vorher nicht feststehen. Jede Station (Knoten) sollte die Adresse eines ankommenden Teilpaketes analysieren und es in Richtung der Empfängerstation weitersenden. Falls ein Zwischenrechner ausfallen sollte, würde der Datenstrom einfach an einen anderen Knoten umgeleitet.
Um diese Konzepte umzusetzen, wurde 1969 das ARPANET-Projekt ins Leben gerufen. Die bei der Entwicklung von ARPANET erforderlichen Software-Komponenten wurden in den Bell-Laboratories von AT&T konzipiert, was zur Entwicklung der Ur-Version von UNIX und seiner zahlreichen Derivate beitrug. Das ARPANET basierte somit auf UNIX-Rechner .
Anfangs wurde die Kommunikation zwischen militärischen und wissenschaftlichen Einrichtungen sowie Datenverbindungen zu bedeutenden Behörden angestrebt. Dieses Vorhaben konnte aber aus technischen Gründen (geringe Leitungskapazitäten) nicht umgesetzt werden [Lie00]. Man begann stattdessen zunächst mit der Installation von vier Knoten : *University of Utah, Stanford Research Institute (SRI), University of Santa Barbara, University of Los Angeles.*

Da die Netze der Universitäten zum Teil aus verschiedenen Systemen aufgebaut waren, ergab sich ein Kommunikationsproblem zwischen ihnen. Zur Lösung dieses Problems mussten Regeln entwickelt werden, die, unabhängig von dem Typ der verschiedenen Systeme, den Datenaustausch zwischen ihnen beschrieben und festlegten. Solche Regeln werden Netzwerkprotokolle genannt. Im Rahmen des ARPANET-Projektes wurde ein erstes *Host-to-Host-Protokoll* entwickelt, um die logische Verbindung zwischen den einzelnen Rechnern und Routern zu gewährleisten. Dieses Protokoll wurde *Network Control Program (NCP)* genannt und stellt die Ur-Version von TELNET und FTP dar.
Als im Jahre 1972 ARPANET das erste Mal der Öffentlichkeit vorgestellt wurde, bestand es schon aus 20 Knotenrechnern bzw. Routern und 50 Hosts. Das ARPANET-Projekt wurde weiterentwickelt. Man verfolgte dabei nach [Lie00] vier grundsätzliche Ziele :

- Das Zusammenspiel heterogener Netzwerkkomponenten
- Die Gewährleistung einer optimalen Sicherheit bei End-to-End-Verbindungen
- Verbesserte Fehlertoleranz bei Leitungsstörungen
- Unabhängigkeit vom Übertragungsmedium

Um 1978 begann man mit der Weiterentwicklung des NCP-Protokolls. Entwicklungsergebnis war das Protokoll TCP/IP (Transmission Control Protocol/Internet Protocol). Bei der Entwicklung von TCP/IP wurde großer Wert auf Plattformunabhängigkeit gelegt, d.h., es sollte von verschiedenen Rechnertypen verstanden werden.
1983 hat sich ARPANET auf TCP/IP umgestellt und im Jahre 1984 wurde TCP/IP, in der Version 4.2, von Berkeley-Software Division Unix (BSD-Unix) integriert, darauf folgte bald die Integration in kommerzielle Versionen von UNIX, und TCP/IP etablierte sich als Internet-Standard. Mit der Benutzung des TCP/IP Protokolls konnten nun unterschiedliche Netzwerke in den USA und anderen Ländern miteinander in Verbindung tretten. Somit leitete das TCP/IP Protokoll die Entwicklung des Internets ein und war eine der wichtigsten, technischen Grundlagen für dessen rasante Expansion. Das TCP/IP wird im nächsten Abschnitt ausführlich behandelt. In der Tabelle 2.13 wird die parallele Entwicklung und die gegenseitige Beeinflussung von TCP/IP und UNIX gesondert dargestellt.

2.3.1 Einführung in TCP/IP

Die Kommunikation von verschiedenen Systemen im Netz wird über Protokolle geregelt. Protokolle sind Regeln, die den Austausch von Daten zwischen zwei oder mehreren Kommunikationspartnern, beschreiben und festlegen. Da diese Protokolle sehr komplex sein können, werden diese in Schichten aufgeteilt, wobei jede Teilschicht bestimmte Teilaufgaben übernimmt. Durch diese Aufteilung werden die Protokolle einfacher in der Handhabung. Jede Schicht nimmt die Dienstleistungen der jeweils niedrigeren Schicht in Anspruch, und gleichzeitig bietet sie der nächsthöheren Schicht ihre Dienste an. Das ISO/OSI-Referenzmodell[1] bietet eine einheitliche Betrachtungsweise der Datenkommunikation und stellt eine Lösung dar, um dem Bestreben nachzukommen, Netzwerkgeräte unterschiedlicher Hersteller in einem Netzwerk gemeinsam betreiben zu können. Ziel des Modells ist es, Vorgänge im Netzwerk weitgehend voneinander zu trennen und über definierte Schnittstellen miteinander zu verbinden. Das ISO/OSI-Modell wird als bekannt vorausgesetzt und soll an dieser Stelle nur durch die 2.14 dargestellt werden. Die Schichten des ISO/OSI-Modells treten in konkreten Anwendungen

[1]ISO/OSI= International Standard Organisation/Open System Interconnection

JAHR	TCP/IP	UNIX
1968	Das Information Processing Techniques Office (IPTO) beginnt mit einer Versuchsreihe zur Realisierung von Rechner-Rechner-Verbindungen	
1969	November: Geburtstunde des ARPANET und UR-Session des TELNET	Entwicklung der Ur-UNIX-Version in den Bell Laboratories von AT&T (Assembler)
1972	ARPANET wird der Öffentlichkeit vorgestellt	UNIX wird mit der Programmiersprache C neu umschrieben
1975	Die Projektleitung ARPANET wird von der Defense Communications Agency (DCA) übernommen	Verkauf der ersten UNIX-Version
1976	Die International Federation of Information Processing (IFIP) legt den Grundstein für TCP/IP	
1979		UNIX Version 7 erscheint
1980		Xenix und BSD 4.1 werden entwickelt
1981		UNIX System III wird eingeführt
1983		Umstellung des ARPANET auf TCP/IP
1984		BSD 4.2 wird vorgestellt
1985		Weitere UNIX-Versionen, wie Ultrix, Xenix 3.0 oder Sinix
1987	Optionale Unterstützung der TCP/IP-Protokollfamilie durch IBM	
1988	Überarbeitung des Simple-Gateway-Protokolls zum Simple Network Management Protocol (SNMP)	
1989	Management Information Base (MIB-I) wird entwickelt	UNIX System 5.4
1990	Unterstützung der TCP/IP-Protokollfamilie durch Digital Equipment, MIB-II kommt raus	
1991	Unterstützung der TCP/IP-Protokollfamilie durch NetWare 3.xx (optional)	
1992	Über 10.000 Firmen unterstützen weltweit Ethernet, Token Ring, FDDI und TCP/IP	
1994	Überlegungen zur Erweiterung der IP-Adress-Struktur in IPng (IP Next Generation)	
1999	IPng-Implementierung in Routern und UNIX-Workstations sind allgemein verbreitet und können zur Konfiguration von IPng-Netzwerken verwendet werden	LINUX hat sich als Freeware-Betriebssystem am Markt etabliert (IBM-Rechner werden alternativ zum eigenen UNIX-Derivat AIX auf Wunsch mit LINUX ausgeliefert!)

Abbildung 2.13: TCP/IP und UNIX im historischen Vergleich [Lie00]

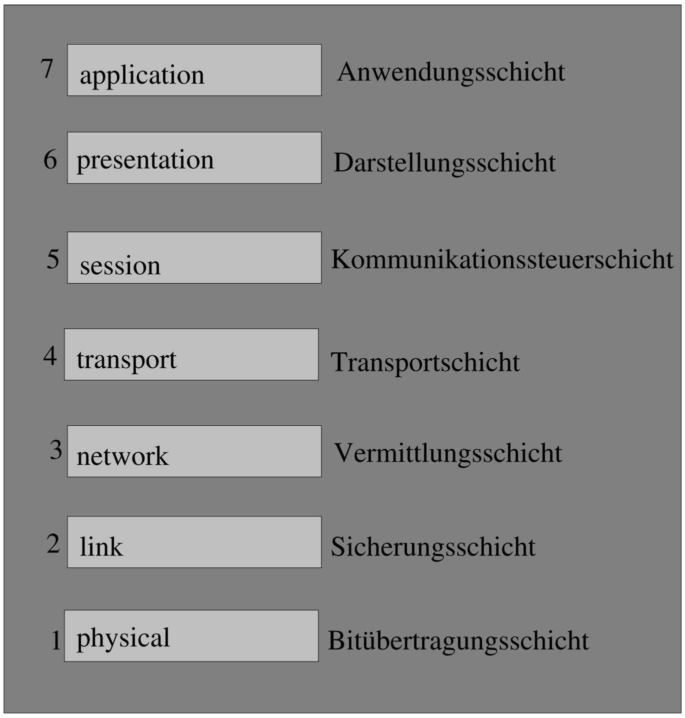

Abbildung 2.14: ISO/OSI-Schichtenmodell [Rö99]

oft nicht in reiner Form auf. Entweder werden die Dienste bestimmter Schichten nicht benötigt, oder mehrere Schichten zu einer zusammengefasst oder Schichten werden weiter unterteilt. Neben TCP/IP existieren noch andere Protokolle. In den folgenden Abschnitten wird nur die TCP/IP-Protokollfamilie betrachtet. Das TCP/IP-Modell hat eine gewisse Ähnlichkeit zu dem ISO/OSI-Modell, besteht jedoch nur aus vier Schichten. Die Unterschiede liegen in den Schichten 5-7 und den Schichten 1 und 2, die das TCP/IP-Modell als eine Einheit betrachtet (siehe Abbildung 2.15)

Sind die Daten wie in der Abbildung 2.16 verarbeitet und erreichen den Zielrechner oder das Zielnetzwerk, so wird der Prozess der Verarbeitung in umgekehrter Reihenfolge durchlaufen. Aber zu nächst sollen die einzelnen Schritte des Datenflusses in Anlehnung an [DH00] genauer betrachtet werden.

1. Eine Anwendung des Senders verschickt Datenpakete die über ein Socket-Interface an das Zielsystem übertragen werden sollen. Das Socket-Interface übermittelt die Daten von der Anwendungsschicht, die den Kommunikationspartner festlegt, an die Transportschicht.

2. Die Transportschicht (TCP) erhält diese Daten, hängt einen Transport-Header (TCP-Header) an die Daten an, und sendet diese Nutzdaten weiter

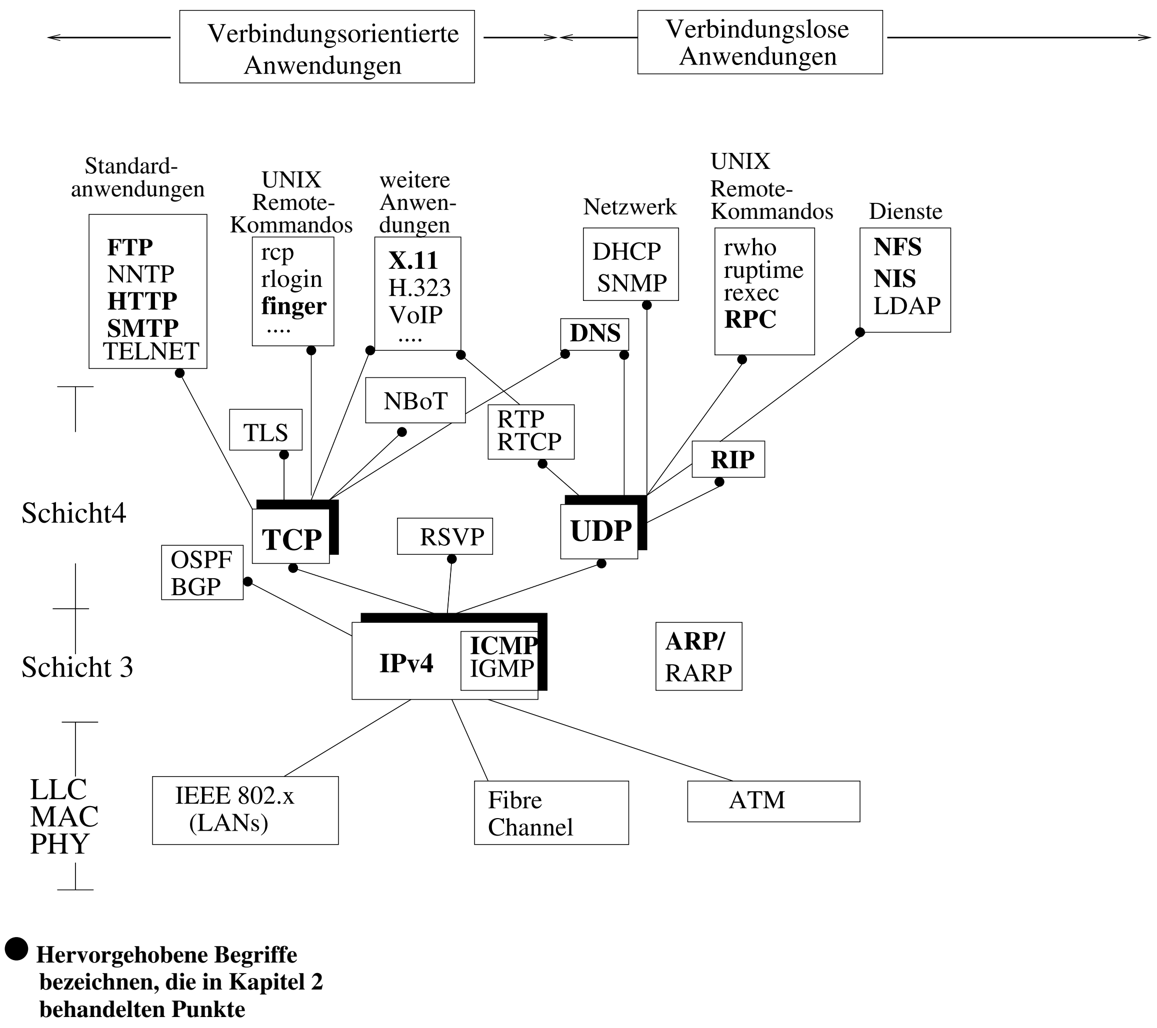

Abbildung 2.15: TCP/IP-Protokoll im Schichtenmodell [AB01]

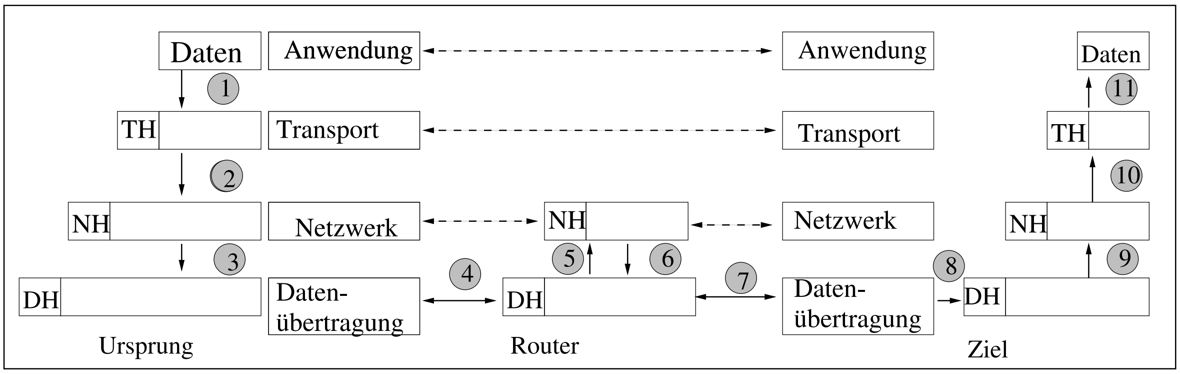

Abbildung 2.16: Der Datenfluss von der Quelle zum Zielort [DH00]

an die Netzwerkschicht. Der TCP-Header enthält differenzierte Informationen, um die angeforderten Dienste der Anwendung zu erbringen (vgl.2.3.2.3).

3. Die Netzwerkschicht erhält von der Transportschicht die Nutzdaten, die aus den Anwendungsdaten und dem TCP-Header bestehen und hängt ihrerseits an diese Daten einen Header (IP-Header) an, um sie dann an die Datenübertragungsschicht weiterzuleiten. In dieser Schicht wird auch die nächste Zwischenstation, die das Paket auf dem Weg zu seinem Zielort passieren soll, bestimmt.

4. An die Nutzdaten aus der Netzwerkschicht wird von der Datenübertragungsschicht ein Header (Übertragungs-Header) angehängt. Auf dieser Ebene wird die physikalische Adresse der nächsten Zwischenstation festgelegt und das Paket weitergeleitet.

5. Die Übertragungsschicht der Zwischenstation entfernt den Übertragungs-Header und leitet das Paket an seine Netzwerkschicht weiter.

6. Die Netzwerkschicht der Zwischenstation betrachtet den Netzwerk-Header und legt fest, welche die nächste Zwischenstation ist und leitet das Paket wieder zurück an die Übertragungsschicht.

7. Diese Übertragungsschicht hängt ebenfalls den Übertragungs-Header an die Nutzdaten aus der Netzwerkschicht und schickt das Paket los.

8. Die beschriebenen Prozesse in 6 und 7 werden so lange wiederholt, bis das Paket den Bestimmungsort erreicht.

9. Erreichen die Daten das Zielsystem, werden in den Schritten 9-11 (siehe Abb.2.16) die Vorgänge aus 1-3 in umgekehrter Reihenfolge durchgeführt, d. h., in den einzelnen Schichten werden die jeweiligen Header dieser Schicht entfernt und die Nutzdaten an die nächsthöhere Schicht weitergeleitet. Nach dem letzten Schritt liegen die Daten im Zielsystem so vor, wie sie von dem Ursprungsrechner versendet wurden.

In den nächsten Abschnitten werden die TCP/IP-Header ausführlicher betrachtet.

2.3.2 TCP/IP-Grundlagen

2.3.2.1 IP-Internet Protocol

Das IP ist das Kernstück der TCP/IP-Protokollfamilie. Das Internet Protokoll ist für die Übertragung von Datenpaketen zwischen Netzwerkschicht und der Transportschicht, sowie für den Transport von Datenpaketen über das Rechnernetz verantwortlich. Durch die Angabe der Quell- und der Zieladresse werden in jedem Datenpaket (Datagram) die Kommunikationspartner festgelegt. Das TCP- und UDP-Protokoll benutzen IP zur Übertragung von Datenpaketen.

Folgende Merkmale sind charakteristisch für das IP-Protokoll [Rö99] :

- Verbindungsloses Protokoll
- Datenfragmentierung in Pakete
- Maximale Paketgröße : 65.536 Bytes
- Adressierung im Internet (gegenwärtig 32-Bit-Adressen, in der neuen Version 128 Bit)
- 8-Bit-Transportprotokolladressen
- Kopfprüfsumme, keine Datenprüfsumme
- Endliche Lebensdauer von Paketen (Time to Live)

Für die folgenden Punkte hat das IP-Protokoll keine Funktion [Rö99] :

- Flusskontrolle
- End-to-End-Sicherung von Nachrichten

IP-Paketformat Ein IP-Paket besteht aus einem Header, der die gesamten Informationen zur Verarbeitung des IP-Paketes auf der Protokollebene enthält, und einem Datenanteil, wie in Abbildung 2.17 dargestellt. Der Datenanteil wird auf IP-Ebene nicht ausgewertet, weil auf der Protokollebene keine Aufgaben der Datensicherung übernommen werden. Dies geschieht auf übergeordneten Protokollschichten.

Der Protokollkopf (Header) eines IP-Paketes enthält folgende Informationen :

- **Version :** Dieses Feld enthält die IP-Versionsnummer. IPv4 ist das zur Zeit aktuelle Protokoll, welches durch die neue Version IPv6 abgelöst werden soll. Die Versionsangabe wird auch benutzt, um die Kompatibilität zu alten Versionen zu gewährleisten, wobei die Versionen vor IPv4 nicht mehr von Bedeutung sind.
- **IHL (Internet Header Length) :** Hier wird die Länge des Protokollkopfes (Header) in Bytes angezeigt.

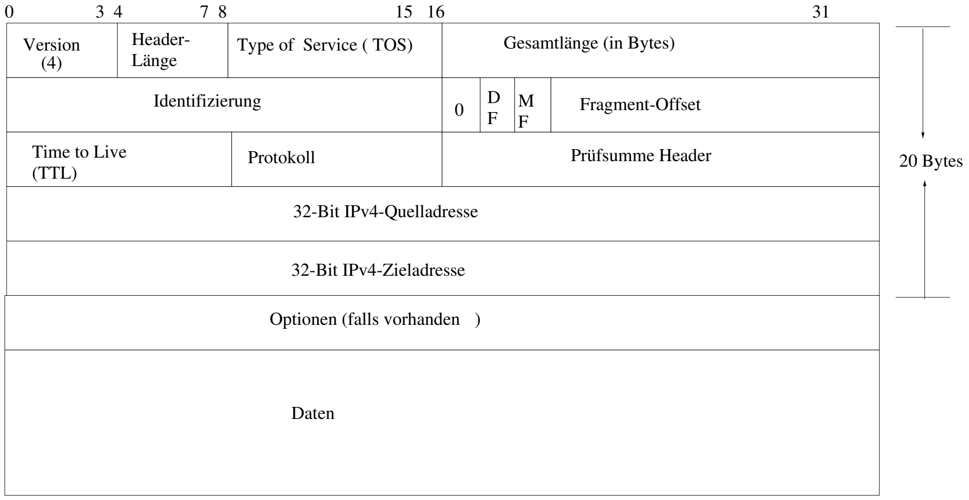

Abbildung 2.17: Das Format des IPv4-Headers [Ste00]

- **Type Of Service (TOS) :** Dieses Feld dient der Beschreibung der Qualität des angeforderten Services, die durch die Prioritätsvergabe bestimmt wird, d. h., durch die TOS-Angaben lässt sich erkennen, ob es sich um einen Netzkontrolldienst bzw. um ein mit Priorität versehenes Paket handelt. In diesem 8-Bit Feld erfolgt die Beschreibung.

Priority	**D**	**T**	**R**	**C**	

Die Prioritäten-Level werden durch die ersten drei Bits angegeben. Diese werden wie folgt verwendet [Lie00]:

  - 000 normal
  - 001 priority
  - 010 immediate
  - 011 flash
  - 100 flash override
  - 101 critical
  - 110 internet control
  - 111 network control

Sind die folgenden vier Bits gesetzt, sind die Funktionen, die sich dahinter befinden, aktiviert. Ansonsten stehen die Funktionen nicht zur Verfügung [Lie00] :

  - D-Bit Delay (Verbindung mit kurzer Verzörgerung wird angeford ert)
  - T-Bit Throughput (hoher Datendurchsatz wird gefordert)
  - R-Bit Reliability (hohe Sicherheit wird gefordert)
  - C-Bit Cost (Route zu niedrigeren Kosten wird gefordert)

- **Gesamtlänge (total length) :** Gibt die gesamte Länge eines IP-Paketes in Bytes an. Die Länge des Datenteils ergibt sich aus der Differenz zwischen dem Wert dieses Feldes und dem IHL-Wert. Die Feldlänge von 16 Bits bestimmt den maximalen Wert (65.535 Dezimal) für die Länge eines Datenpaketes. Da diese Größe von den meisten Netzwerken nicht unterstützt wird, mussen die Datenpakete fragmentiert werden.

- **Identifizierung :** Das 16-Bit-Identifikationsfeld dient der eindeutigen Bestimmung eines IP-Datenpaketes (Transport-Nutzdaten + IP-Header). Das Feld wird sowohl bei der Segmentierung von zu sendenden Dateien, als auch bei der Fragmentierung von großen IP-Paketen verwendet. Im Quellrechner wird jeder zu übertragenden Datei (Nachricht) eine eindeutige Identifikation vergegben. Diese Identifizierungsparameter, die zur Nummerierung der fragmentierten Datenpakete verwendet werden, benötigt man, um die IP-Pakete, beim Vorgang des Zusammensetzens, den richtigen Datenpaketen zuzuordnen.

- **Flags: 0, DF, MF :** Wird eine Fragmentierung vorgenommen, dient dieses Feld zur Steuerung der Fragmentierung. Das erste Bit *(high-order-bit)* wird nicht verwendet und erhält den Wert 0. Das **DF-Feld** steht für „*do-not-fragment-bit*“, das heißt, ist dieses Feld gesetzt, darf nicht fragmentiert werden.

  MF-Feld steht für „ *more-flag-bit*“ und gibt an, ob noch ein weiteres Fragment folgt, wenn ja, ist dieser Wert auf 1 gesetzt, ansonsten auf 0 und das letzte Fragment des Datagramms liegt vor.

- **Fragment Offset** (Fragmentabstand) : Hier folgen die Angaben, die für die korrekte Zusammenführung fragmentierter Datenpakete erforderlich sind, wie die Position der Daten des jeweiligen Datenpaket-Fragments innerhalb des Original-Datenpaketes vor der Fragmentierung.

- **Time to Live :** Dieses Feld bestimmt die Lebensdauer eines Datenpaketes, die von der sendenden Station vorgegeben wird. Der Wert der Lebenszeit jedes Datenpaketes wird beim Passieren eines Routers „um eins“ reduziert. Wird dieser Wert 0 bevor das Paket die Zieladresse erreicht, wird das Datenpaket verworfen und der Absender bekommt eine Fehlermeldung (ICMP-Meldung).

- **Protokoll :** Bestimmt das übergeordnete Transport-Protokoll, wie z.B. UDP, TCP, ICMP, EGP[2], GGP.[3]

- **Prüfsumme Header :** Dieses Feld enthält die Prüfsumme, die aus dem IP-Header[4] berechnet wird und die Integrität des IP-Headers gewährleisten soll. Da aber diese Prüfsumme keine kryptographische Prüfsumme ist, kann sie leicht gefälscht werden.

- **Quelladresse :** Dieses Feld beinhaltet die IP-Adresse des sendenden Rechners

- **Zieladresse :** Dieses Feld beinhaltet die IP-Adresse des empfangenden Rechners.

- **Optionen :** Dieses Feld ist kein Pflichtbestandteil eines IP-Datenpakets, aber ermöglicht eine erweiterte Nutzung des IP-Protokolls, z.B. Bereitstellung von Zeitmarken oder Unterstützung von Routing-Funktionen, d. h., der Absender kann festlegen auf welchem Weg das Paket den Zielrechner erreichen soll.

IP-Adressen Die Felder *Quelladresse* und *Zieladresse* im IP-Header dienen der Identifikation des Absenders und des Empfängers eines IP-Paketes. Die IP-Adresse stellt eine globale, eindeutige Identifizierung eines Rechners dar. D. h. jedes Endsystem (Rechner, Server, Router) im Netz wird durch eine logische IP-Adresse identifiziert, um die reibungslose Kommunikation zwischen den Endsystemen und Netzkomponenten zu gewährleisten. Jede IP-Adresse, sogenannte Unicast-Adresse, besteht im allgemeinen aus:

- **Netz-ID :** Bezeichnet das Teilnetz, in dem der Rechner mit der angegebenen Hostadresse zu finden ist.

- **HOST-ID :** Identifiziert ein beliebiges Endsystem im Netz, d. h., sie bezeichnet den Rechner in seinem Netz.

Der Netzanteil wird von nationalen Organisationen, in Deutschland vom DE-NEC[5], an die Provider oder End-Anwender vergeben (Netz-ID). Jedem Betreiber eines Netzwerkes wird ein Bereich von IP-Adressen zugewiesen, und innerhalb dieser Adressbereiche kann der Betreiber des Netzes die Adressen den einzelnen Rechnern frei zuordnen (HOST-ID). Der Netzanteil wird in fünf Klassen (A-E)

[2]Exterior Gateway Protocol

[3]Gateway-to-Gateway Protocol

[4]Nach dem Passieren eines Routers müssen die Werte für **TTL, Flag** und **Flag Offset** neu berechnet werden. Zur Vermeidung einer unnötigen Verlängerung der Transportzeit der Datenpakete, wird die Prüfsumme nur aus dem Header gebildet.

[5]NIC = Network Information Center

unterteilt, wobei die ersten Bits der Adresse die Zugehörigkeit zu einer Klasse bestimmen. Die Abbildung 2.18 gibt einen Überblick über die Aufteilung der Internet-Adressklassen. Die Klasse-A-Netze sind an einige große Organisationen, wie z.B. das Militär oder die Forschung, vergeben. Adressen der Klasse-B und C werden für alle übrigen Internet-Nutzer vergeben, wobei die Klasse-B-Netze über 16.000 Endstationen zulässt, und Klasse-C-Netze für kleinere Netze genutzt wird und 254 Endstationen zulässt. Multicast-Adressen, Adressen der Klasse-D, ermöglichen die Adressierung aller Rechner einer Multicast-Gruppe gleichzeitig. Die Klasse E stellt eine experimentelle Adresse dar und war ursprünglich für spezielle Einsätze reserviert, wird zur Zeit aber nicht verwendet.

<table>
<tr><th>Adreßklasse</th><th>Adressraum</th><th colspan="2">Bit 1 - 8</th><th>Bit 9- 17</th><th>Bit 18-24</th><th>Bit 25-32</th><th>Max. Anzahl der mögl. Netze</th><th>Anzahl der mögl. Rechner je Netz</th></tr>
<tr><td>A</td><td>0.0.0.0 - 127.255.255.255</td><td>0</td><td>Netz-anteil</td><td colspan="3">Rechneranteil</td><td>2^7</td><td>2^{24}</td></tr>
<tr><td>B</td><td>128.0.0.0 - 191.255.255.255</td><td colspan="3">1 0 | Netzanteil</td><td colspan="2">Rechneranteil</td><td>2^{14}</td><td>2^{16}</td></tr>
<tr><td>C</td><td>192.0.0.0 - 223.255.255.255</td><td colspan="4">1 1 0 | Netzanteil</td><td>Rechner-anteil</td><td>2^{21}</td><td>2^8</td></tr>
<tr><td>D</td><td>224.0.0.0 - 239.255.255.255</td><td colspan="5">1 1 1 0 | Multicast-Adressen</td><td colspan="2">2^{28}</td></tr>
<tr><td>E</td><td>240.255.255.255</td><td colspan="5">1 1 1 1 0 | Reserviert</td><td colspan="2">2^{27}</td></tr>
</table>

Abbildung 2.18: Einteilung der Internet-Adressklassen [Fuh00]

Fragmentierung Fragmentierung bezeichnet den Vorgang, bei dem ein IP-Paket in dem Quellrechner oder in einem Router in mehrere IP-Pakete zerlegt wird. Die einzelnen IP-Teilpakete (IP-Fragmente) werden unabhängig von den anderen übertragen und erst im Zielrechner wieder zusammengesetzt. Dieser Vorgang geschieht mit Hilfe der Flags, der Längen-Felder und der Fragmentierungs-Offset-Felder, die im IP-Header aufgeführt sind. Die Fragmentierung ist notwendig, um die Begrenzung der IP-Paket-Größen zu überwinden. Über welche Netzwerkstruktur ein IP-Paket übertragen wird, ist ausschlaggebend für die maximale IP-Paketlänge, die unterstützt wird. So erlaubt z.B. Ethernet eine maximale Paketgröße von 1500 Bytes, ein IP-Paket kann aber bis zu 65.535 Bytes groß sein. Bei der Fragmentierung wird nur der Datenanteil[6] des IP-Paketes berücksichtigt, weil jedes Teilpaket den Original IP-Header enthalten muss, denn in diesem

[6]IP-Paket = Nutzdaten + IP-Header

sind die Adressen (Routenwahl-Information) enthalten. Folgende Anpassungen im Header sind notwendig:

- Das Längenfeld wird auf die Größe des Fragments gesetzt
- Das Fragment-Offset gibt die Position der Fragmente innerhalb des ursprünglichen Paketes an.
- MF-Bit eines jeden Datagramms wird auf „1“ gesetzt, um zu bestimmen, wann das letzte Fragment (MF=0) vorliegt.

Im Zielrechner werden nach dem Empfang des letzten IP-Teilpaketes die einzelnen Fragmente mit Hilfe der Angaben des Identifikationsfeldes wieder zusammengesetzt. Nun kann mit Hilfe des Fragment-Offsets (Fragmentabstand) und der Längenfelder überprüft werden, ob die Fragmente vollständig sind. Sollten Fragmente fehlen, oder nicht alle Teilpakete das Zielsystem innerhalb des implementierungsabhängigen Zeitintervalls erreichen, werden auch die restlichen Pakete vom Empfänger verworfen und eine wiederholte Übertragung des IP-Paketes muss von TCP veranlasst werden. Über diesen Vorgang wird innerhalb von IP keine Fehlermeldung an den Absender verschickt. Dies übernimmt das ICMP-Protokoll.

Routing Die Bestimmung der Übertragungswege der IP-Pakete von einem Sender zum Empfänger wird als Routing bezeichnet. Wird ein IP-Paket innerhalb des gleichen Subnetzes verschickt, wird das Paket direkt über entsprechende Protokolle der Sicherungschicht dem Empfänger zugestellt, d.h., die Dienste des Routers werden nicht in Anspruch genommen (*direct-Routing*). Anderenfalls wird das Paket über Zwischenstationen (Hops) an den Empfänger übermittelt, d.h., der Zielrechner liegt außerhalb des eigenen IP-Netzwerkes und die Kommunikation wird indirekt über Verbindungsknoten (Router) geführt. Damit ein Router ein empfangenes Paket auf einen optimalen Weg in einem Verbund von mehreren Netzen weiterleiten kann, benötigt er Informationen bezüglich der Route. Diese sind in den sogenannten Routing-Tabellen festgelegt. Anhand dieser Einträge, die z.B. Ziel-Netzwerkadressen und Routenwahl enthalten, kann der Router erkennen, ob die betreffende IP-Adresse direkt erreichbar ist, oder über welche Zwischenstation das IP-Paket weiterzuleiten ist, oder ob eine IP-Adresse nicht erreichbar ist. Ein Paket, das nicht zugestellt werden kann, wird verworfen und der Absender wird benachricht. Routing-Tabellen Einträge können statisch oder dynamisch erstellt werden:

- Statisch: Der Administrator legt die Tabellen manuell an, d.h. jede Erweiterung oder Modifikation Aufgrund von Veränderung in der Netzwerktopologie muss per Hand vorgenommen werden. Es gibt hierbei drei Arten von Tabelleneinträgen :

1. **Netzrouten :** Dieser Eintrag gibt an wie ein komplettes Netz erreichbar ist.
2. **Hostrouten :** Dieser Eintrag definiert wie ein einzelner Rechner erreichbar ist.
3. **Default-Route :** Dieser Eintrag legt fest, wohin Pakete geschickt werden sollen deren Zieladresse über keine der angegebenen Routen erreicht werden kann.

- Dynamisch: Dies setzt das Zusammenwirken aller in einem Netzverbund installierten Router voraus, die über die Routing-Protokolle die nötigen Routing-Information untereinander austauschen (Broadcast).

Die Zwischenstation, die ein IP-Paket passieren soll, wird an dem lokalen Rechner oder Router, der das Paket aktuell bearbeitet, festgelegt. Es besteht aber auch die Möglichkeit, das durch die Aktivierung der Source-Routing-Option ein bestimmter Weg vorgeschrieben werden kann, um z.B. eine Route zu wählen, die durch eine Firewall geschützt ist. Eines der gebräuchlichsten Routing-Protokolle ist das RIP-Protokoll. Im Abschnitt 2.4.6 wird das Sicherheitsproblem, die durch den Einsatz von RIP-Protokoll entstehen, erläuert.

2.3.2.2 ICMP-Protokoll

Treten bei Übertragung von IP-Paketen Fehler auf, werden innerhalb von IP keine Fehlermeldungen an den Sender verschickt. Diese Aufgabe wird auf das ICMP-Protokoll ausgelagert. Das ICMP-Protokoll ist ein integraler Bestandteil von IP, wird aber auch von TCP und UDP genutzt, um Fehler und Diagnosemeldungen im Netz zu transportieren. ICMP-Nachrichten werden für die Übertragung, wie bei TCP und UDP, in IP-Pakete integriert, aber im Gegensatz zu diesen ist das ICMP-Protokoll nicht auf der Transportschicht angesiedelt, die Verarbeitung erfolgt auf der Schicht drei (IP-Ebene. vgl. Abbildung 2.15).
Eine Besonderheit von ICMP ist, dass Fehler bei der Verarbeitung von ICMP-Paketen keine neuen ICMP-Fehlermeldungen erzeugen, dies hilft Endlosschleifen zu vermeiden, die sonst das Netz mit ICMP-Meldungen überfluten würden.

Der Aufbau (siehe Abb. 2.19) eines ICMP-Header ist in allen Nachrichten gleich, nur die Bedeutung der einzelnen Felder variieren in Abhängigkeit von der zu erfüllenden Aufgabe. Die Bedeutung der einzelnen Felder (in Anlehnung an [Rö99] und [Lie00]):

Type: Nachrichtentyp, dient der Unterscheidung der Bedeutung von einzelnen ICMP-Nachrichten.

Code: Dieses Feld dient der weiteren Spezifikation der Nachricht innerhalb des Nachrichtentypes.

Prüfsumme: Dieses Feld beinhaltet eine Prüfsumme für die gesamte ICMP-Nachricht, die für die Überprüfung von ICMP-Daten benutzt werden kann.

Daten: Der IP-Header und die ersten 64 Bits des IP-Paketes, dass den Fehler verursacht hat, sind in diesem Feld enthalten.

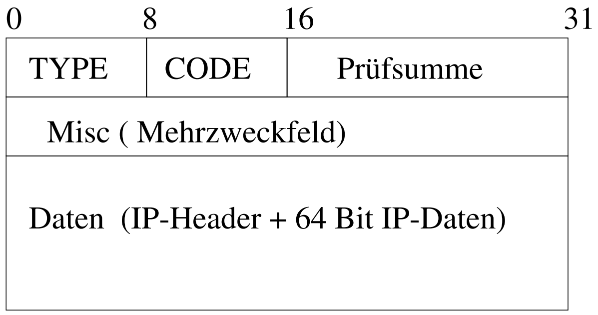

Abbildung 2.19: ICMP-Header [Rö99]

Um ICMP-Nachrichten kenntlich zu machen, wird das TOS-Feld im IP-Header auf „0“ gesetzt und das Protokoll-Feld auf „1“ (Protokoll-Code für ICMP). Empfänger von ICMP-Fehlermeldungen sind Prozesse der IP-, TCP- und UDP-Protokolle, die gleichzeitig auch die Auslöser dieser Meldungen sind. Ein Beispiel für eine Fehlermeldung ist:

- Destination Unreachable Nachricht (Typ 3, Code 1), d. h., ein IP-Paket kann nicht an den Bestimmungsort übermittelt werden, da der Empfänger (Zielhost) nicht erreichbar ist. Die Abbildung 2.1 und 2.2 geben ein Überblick über die Typen und Codes von ICMP-Nachrichten.

Außer den ICMP-Fehlermeldungen, die bei fehlerhaften Paketen generiert werden, gibt es auch ICMP-Diagnosemeldungen mit deren Hilfe sich Funktionen, wie die Betriebsbereitschaft oder Laufzeitmessungen, überprüfen lassen. Um dieses zu ermöglichen gibt es eine Reihe von ICMP-Meldungen, die für die Anfrage von Informationen und zur Beantwortung dieser Anfragen verwendet werden. Statt der üblichen IP-Header in der ICMP-Nachricht, werden Testdaten beliebiger Länge übermittelt, z.B. >>echo request<< , um die Erreichbarkeit von Rechnern zu überprüfen. Hierzu gehören:

- **Echo Request/Reply Message (Echo Funktion):** Das in allen TCP/IP -Implementierungen realisierte Ping-Kommando ist ein Beispiel für eine Anfragemeldung. Diese überprüft, ob ein Rechner oder Router erreichbar ist, zu diesem Zweck sendet das Ping-Programm ein Echo-Request und erwartet ein Echo-Reply, das geht natürlich nur, wenn das Zielsystem erreichbar ist.

TYP	**Funktion**	**Erläuterung**
0	echo reply	Antwort auf ein echo request
3	destination unreachable	Datenpaket nicht zustellbar
4	source quench	Empfänger signalisiert:Pufferkapazität erreicht(kann Paket nicht so schnell verarbeiten)
5	redirect message	Vorschlag:Routenwechsel zur Wegverkürzung
8	echo request	Von Netzknoten versendet, um Empfangsbereitschaft eines anderen Netzknotens zu ermitteln
11	time exceeded	An Absender:Lebenszeit des Paketes ist abgelaufen
12	parameter problem	Fehlerhafte Angabe im Paketheader-Paket muss verworfen werden
13	timestamp request	Ziel-Netzknoten soll seine aktuelle Uhrzeit senden
14	timestamp reply	Antwort auf timestamp request
15	information request	Von Netzknoten verwendet, um Net-ID des Netzwerkes zu ermitteln, an das sie angeschlossen sind
16	information reply	Antwort auf information request
17	address mask request	Von Host verwendet, um eigene Subnetzmaske zu ermitteln
18	address mask reply	Antwort auf address mask reply

Tabelle 2.1: ICMP-Pakettypen [Rö99]

CODE	**Bedeutung**
0	network unreachable
1	host unreachable
2	protocol unreachable
3	port unreachable
4	fragmentation needed and DF-Bit set
5	source route failed
6	destination network unknown
7	destination host unknown
8	source host isolated
9	communication with destination network prohibited
10	communication with destination host prohibited
11	network unreachable for type of service
12	host unreachable for type of service

Tabelle 2.2: Codes für Nachrichtentyp >>destination unreachable<< [Rö99]

- **Timestamp Request/Reply Message:** Anfrage, um eine Zeitmarke zu erhalten, die das aktuelle Datum und die Uhrzeit angibt.

- **Information Request/Reply Message:** Ermöglicht einem Rechner die eigene IP-Adresse (z.B. von einem Adress-Server) zu erfragen.

- **Address Mask Request/Response:** Abfrage der Subnetz-Maske

2.3.2.3 TCP (Transmission Control Protocol)

Das **T**ransmission **C**ontrol **P**rotocol setzt auf IP auf und wird als das zentrale Protokoll innerhalb der TCP/IP-Protokollfamilie betrachtet. Es bildet die Grundlage für weitere Anwendungsprotokolle, wie beispielsweise FTP[7], TELNET, SMTP[8] oder das X-Protokoll für XWindows/Motif-Anwendungen. Das TCP-Protokoll ist in OSI-Schicht 4 angesiedelt und ist im Gegensatz zu IP ein verbindungsorientiertes Protokoll. Da das IP-Protokoll keine Fehlerkontrolle bei der Übermittlung von IP-Paketen bietet, ist der sichere Datenaustausch nicht möglich. Fehler, wie Verluste bzw. Verfälschung von Daten, die während der Übermittlung auftauchen, werden von IP nicht erkannt. Dies übernimmt das TCP-Protokoll, dessen Aufgabe es ist, den sicheren Datenaustausch zwischen zwei kommunizierenden Rechnern zu gewährleisten, d.h., auftretende Fehler bei der Übertragung zu entdecken und Gegenmaßnahmen zu ergreifen. In diesem Zusammenhang bedeutet „ gesichert“ nur eine Absicherung gegen Übertragungsfehler des IP-Protokolls, d.h., TCP sorgt dafür, dass kein Datenpaket verlorengeht, dass alle Pakete in der richtigen Reihenfolge beim Empfänger ankommen und dass die von IP empfangenen Daten der richtigen Anwendung zugestellt werden.
Auf der TCP-Ebene erfolgt die Adressierung über sogenannte Sende- und Empfangsports. Da zwischen zwei Rechnern mehrere TCP-Verbindungen bestehen können, werden Portnummern vergeben, um die einzelnen Verbindungen auseinanderhalten zu können. Die Portnummer repräsentiert eine 16-Bit-Adresse, die der Identifikation eines eindeutigen Zugangspunktes dient, den das TCP-Protokoll braucht, um mit dem übergeordneten Anwendungsprotokoll (z.B. FTP) Daten austauschen zu können. D.h., nach dem Aufbau einer Verbindung wird erst zu dem betreffenden Dienstprogramm verzweigt, wenn die betreffende Portnummer angegeben wird. Die Portnummern werden mit jedem TCP-Paket, angegeben in den Feldern „Source Port“ (Sende-Portnummer) und „Destination Port“ (Empfänger-Portnummer), verschickt. Die Ports mit den Nummern 0 bis 1023 sind sogenannte *well known ports*, die weltweit eindeutig für Standarddienste, wie z.B. TELNET, FTP oder HTTP, von vornherein reserviert sind (z.B. 80 für WWW).

Funktion des Protokolls TCP Eine Anwendung veranlasst TCP zum Aufbau einer Verbindung, und über diese Verbindung werden die Daten in Form von festgelegten Datenblöcken (TCP-Pakete) ausgetauscht. Aufgabe von TCP ist es, den mitunter umfangreichen Datenstrom, der übertragen werden sollen, im

[7]File Transfer Protocol
[8]Simple Mail Transfer Protocol

Quellrechner in nummerierte Segmente zu teilen und jedes Segment mit einem eigenen Header zu versehen. Vor Beginn einer Übertragung müssen die maximalen Segmentgrößen zwischen den TCP-Instanzen der Sende- und Zielrechner vereinbart werden. Das IP-Protokoll überträgt die TCP-Pakete als zusammenhanglose IP-Pakete (Dtenpakete). Nachdem die Daten über das Netz übertragen wurden, erfolgt die Datenübergabe an das TCP, aber vorher entfernt IP seinen Header. TCP überprüft nun die Pakete auf Fehler, sortiert diese in die richtige Reihenfolge und entfernt ebenfalls den TCP-Header. Sollte ein Paket nicht den Zielrechner erreichen, veranlasst TCP die erneute Übertragung des entsprechenden Datensegmentes. Zu Letzt werden die Daten an die Ziel-Anwendung übergeben, und sobald die Anwendung entsprechende Anweisung erteilt, wird die logische Verbindung durch TCP beendet. Die Abbildung 2.20 zeigt den Aufbau des TCP-Headers

<table>
<tr><td colspan="8">Source Port</td><td>Destination Port</td></tr>
<tr><td colspan="9">Sequence Number</td></tr>
<tr><td colspan="9">Acknowledgement Number</td></tr>
<tr><td>Data Offset</td><td>Reserved</td><td>URG</td><td>ACK</td><td>PSH</td><td>RST</td><td>SYN</td><td>FIN</td><td>Window Size</td></tr>
<tr><td colspan="8">Checksum</td><td>Urgent Pointer</td></tr>
<tr><td colspan="9">Options/Padding</td></tr>
<tr><td colspan="9">····· Data ·····</td></tr>
</table>

Abbildung 2.20: TCP-Header [Ste00]

TCP-Format Im Folgenden sollen die Steuerungsinformationen im TCP genauer betrachtet werden:

- **Source Port** *(Quell-Port)* : Dieses Feld beinhaltet die Portnummer des Anwendungsprozesses im Quellrechner, der die Verbindung initialisiert.
- **Destination Port** *(Ziel-Port)* : Dieses Feld beinhaltet die Portnummer des Anwendungsprozesses im Zielrechner, an den die Daten adressiert sind.
- **Sequence Number** *(Sequenznummer) :* Der Sender generiert eine Sequenznummer und diese dient der Nummerierung der gesendeten Datensegmente. Die Sequenznummern werden gegenseitig ausgetauscht und gegenseitig bestätigt. Um ein TCP-Segment eindeutig zu identifizieren, muss gewährleistet sein, dass sich keine zwei gleichen Nummern, zum gleichen Zeitpunkt, im Netz befinden, daher darf sich die Sequenznummer innerhalb

einer festgelegten Lebensdauer für ein IP-Paket nicht wiederholen. Die Sequenznummer wird vom Sender immer jeweils um die Anzahl der bereits gesendeten Bytes erhöht.

- **Acknowledgement Number** *(Quittungsnummer) :* Dem Sender werden die empfangenen Daten durch die Acknowledgement Number bestätigt, d. h. diese Nummer wird vom Empfänger gesetzt, um dem Sender mitzuteilen, bis zu welchem Byte die Daten korrekt empfangen wurden.

- **Data Offset** *(Datenabstand) :* Dieses Feld gibt die Länge der TCP-Header in 32-Bit-Blöcken an und beschreibt damit die Stelle, ab der die Daten beginnen.

- **Reserved :** Dieses Feld wird zur Zeit nicht verwendet

- **Control Flags** *(Kontroll-Flags) :* Mit Hilfe der Control Flags können Aussagen über den Zustand der TCP-Verbindung gemacht werden.
  - **URG** *Urgent-Flag :* Der Urgent-Pointer (das ist ein Zeiger im Urgent-Feld) ist gültig.
  - **ACK :** Quittungsnummer, bestätigt empfangene Daten.
  - **PSH** *(Push-Funktion) :* Ist dieses Bit gesetzt, bedeutet es, dass die Daten so schnell wie möglich an die Anwendung weitergeleitet werden sollen.
  - **RST** *(Reset) :* Durch dieses Bit wird dem Empfänger mitgeteilt, dass die Verbindung aufgrund einer Fehlersituation beendet wird, z.B. wenn ein empfangenes Paket nicht zu der angegebenen Verbindung passt.
  - **SYN** *(Synchronization-Flag) :* Verbindungsaufbauwunsch, leitet den Synchronisationsvorgang ein (three way handshake)
  - **FIN** *(Final-Flag) :* Einseitiger Verbindungsabbau, d. h., durch dieses Bit wird das endgültige Verbindungsende eingeleitet, und haben beide Kommunikationspartner das FIN-Bit gesetzt, so ist die Verbindung ordnungsgemäß beendet.

- **Window Size** *(Fenstergröße) :* Mit dieser Angabe wird versucht, das Überlaufen des Buffers beim Empfänger während einer Verbindung zu vermeiden. Zu diesem Zweck wird dem Sender kontinuierlich mitgeteilt, wieviel Daten (in Byte) noch unbestätigt versendet werden können.

- **Checksum** *(Prüfsumme)*

- **Urgent Pointer** *(Urgent-Zeiger) :* Das URG-Bit zeigt an, dass dringende Daten im TCP-Paket enthalten sind. Der Verweis geschieht über den Urgent Pointer. Wird häufig für die Unterbrechung von Prozessen verwendet.

- **Options/Padding** *(Füllzeichen) :* Erlaubt Service-Optionen anzugeben, wie Optionstyp, Optionslänge oder Optionsdaten, z. B. kann man die maximale Segmentgröße, die zwischen zwei Kommunikationspartnern verarbeitet werden kann, festlegen. Das letzte Feld (Padding) wird, bis die Gesamtlänge des 32-Bit-Blocks erreicht ist, mit Nullen ergänzt.

TCP-Verbindungsaufbau Die Verbindungsphase bei TCP wird als *three-way-handshake* bezeichnet. Sie sorgt für die Synchronisation der Kommunikation und soll sicherstellen, dass die TCP-Verbindung korrekt initialisiert wird. Dieser Vorgang wird durch den Austausch von drei Datenpaketen (Kontrollpakete), mit denen sich die Kommunikationspartner über den Zustand der TCP-Verbindung einigen, realisiert. Der Verbindungsaufbau, z.B. zwischen einem *client A* und einem *server B*, soll in drei Schritten (S1-S3) erläutert werden :

S1: *Client A* sendet Server B ein Datenpaket mit gesetztem Synchronisations-Bit SYN und der Anfangssequenznummer $SEQ.NR_A$ der aufzubauenden TCP-Verbindung. Diese Sequenznummer wird vom Betriebssystem „zufällig" gewählt.

S2: *Server B* antwortet durch das Senden eines Datenpaketes, das zwei wesentliche Informationen enthält :

- Ein ACK-Flag und eine Bestätigungsnummer, die der Sequenznummer des empfangenen Paketes entspricht, also $ACK(SEQ.NR_A)$. Hiermit wird der Empfang des Paketes bestätigt.
- Die Anfangssequenznummer des Servers B, also $SEQ.NR_B$, und das gesetzte SYN-Flag. Hiermit wird Client A die Annahme der Verbindung bestätigt.

S3: *Client A* bestätigt den Empfang des zweiten Paketes durch das Senden eines Antwortpaketes mit folgendem Inhalt :

- Gesetzter ACK-Flag
- Bestätigungsnummer = $SEQ.NR_B$

2.3.2.4 UDP-User Datagram Protocol

Das User Datagram Protocol setzt, genau wie das TCP-Protokoll, auf das IP-Protokoll auf, d. h., es benutzt auch das IP-Protokoll zur Übertragung von Datenpaketen. UDP ist, wie TCP, auf der Transportschicht angesiedelt, aber im Gegensatz zu diesem ist UDP ein verbindungsloses Protokoll. Daher besitzt UDP weniger Funktionalität und ist auch einfacher aufgebaut als TCP.

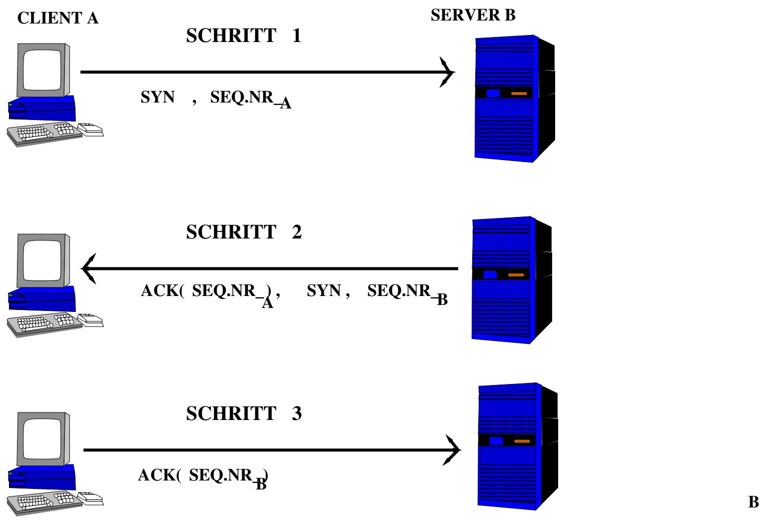

Abbildung 2.21: TCP-Verbindungsaufbau

Nachteil ist, dass UDP keinen Schutz gegen den Verlust von Datenpaketen bietet und auch keine Mechanismen, um ankommende Datenpakete in der richtigen Reihenfolge auszuliefern.
Der Vorteil im Vergleich zu TCP ist, dass UDP schneller ist und weniger Netzlast erzeugt. Daher eignet sich UDP besser für Hochgeschwindigkeitsanwendungen, wie z.B. für verteilte Dateisysteme (NFS), die in schnellen und sicheren LANs eingesetzt werden. Probleme durch ungesicherte Übertragung und fehlende Flusskontrolle entstehen erst, wenn UDP in ungesicherten Umgebungen (Internet) eingesetzt wird. Die Abbildung 2.22 zeigt den Aufbau eines UDP-Headers.

0	16 31
Sender-Portnummer	Empfänger-Portnummer
Länge	Prüfsumme
Daten	

Abbildung 2.22: Ein UDP-Header [Rö99]

Der UDP-Header eines Datenpaketes enthält folgende Informationen:

- **Sender-Portnummer** (Analog zu TCP)
- **Empfänger-Portnummer** (Analog zu TCP)
- **Länge :** Dieses Feld beinhaltet die gesamte Länge des Datenpakets, inklusive Header.

- **Prüfsumme:** Auch die Prüfsumme bezieht sich auf das gesamte Datenpaket, aber im Gegensatz zum TCP-Protokoll ist die Berechnung einer Prüfsumme nicht notwendigerweise erforderlich, d.h., es kann auch 0 sein, dann findet keine Überprüfung beim Empfänger des UDP-Paketes statt.
- **Daten:** Nutzdaten

2.4 Angriffe auf Computer und Netze

In diesem Abschnitt werden einige Sicherheitsprobleme von Systemen und ihrer Vernetzung betrachtet. Die Sicherheitsproblematik wird durch die Vernetzung der verschiedenen Systeme komplex. In der Praxis wird man heterogene Netzwerke vorfinden, wo PCs und Workstations mit Mainframerechnern zusammenwirken. Jede Art Rechner und Betriebssystem hat dabei ihre eigenen, spezifischen Sicherheitsprobleme. Es sei an dieser Stelle erwähnt, dass bei der Betrachtung der einzelnen Angriffsmethoden und den Systemschwächen, die diese Angriffe begünstigen, den Schwerpunkt auf UNIX Betriebssysteme gelegt haben, dies aber nicht immer explizit bei der Erklärung der einzelnen Angriffe angeben werden.

Zunächst werden die Urheber von Angriffen genauer betrachtet. Dabei wird zwischen *hacker* und *cracker* unterschieden.

2.4.1 Hacker und Cracker

Es gibt im Allgemeinen zwei Wörter, die einen Einbrecher (im IT-Bereich) beschreiben : *Hacker* und *cracker.*
In [LE99] wird als *hacker* eine Person bezeichnet, die

> *„....Gefallen an der intellektuellen Herausforderung hat, auf kreative Weise bestimmte Beschränkungen zu überwinden oder zu umgehen. Eine solche Person wird durchaus die Fähigkeit, nicht aber die Neigung besitzen, unbefugt in fremde Systeme einzudringen. Insbesondere wird ein „wahrer Hacker“ keinen Diebstahl, Vandalismus oder Verrat vertraulicher Daten begehen“.*

Der Begriff *cracker* wurde von den *hackern* selbst eingeführt. Dadurch sollte eine Differenzierung des Begriffs *hacker* vorgenommen werden. Die Notwendigkeit dazu entstand, weil allen *hackern*, meist seitens der Medien, böswillige Handlungen vorgehalten wurden. Der Begriff *cracker* beschreibt die Person, die mutwillig anderen Leuten Schaden zufügt, indem sie deren Systeme manipuliert,modifiziert oder zerstört. In [LE99] heißt es weiterhin :

> *Auch wenn sich die Vorgehensweise eines Hackers und eines Crackers technisch nicht grundlegend unterscheidet, so sind doch die Beweggründe deutlich andere , zumal Cracker dazu tendieren, nur in sehr*

kleinen Gruppen zusammenzuarbeiten und nicht bereit sind, die gewonnenen Informationen mit anderen zu teilen.

Wir gebrauchen in dieser Diplomarbeit den Ausdruck *Eindringling* oder *Einbrecher*, mit der wir jede Person bezeichnen, die versucht, unerlaubt in ein Fremd-System einzubrechen. Eindringlinge können grob in zwei Gruppen eingeteilt werden :

- **Outsiders :** Das sind Personen, die von außerhalb versuchen, unerlaubt in das System bzw. Netzwerk einzubrechen oder die äußere Präsenz des Netzwerkes anzugreifen (z.B. defensive Webserver). Äußere Eindringlinge können über das Internet, über dial-up Leitungen, durch physikalische break-ins oder aus dem Partner-Netzwerk (z.B. Kunden- oder Verkäufernetzwerk) heraus ihre Einbruchsversuche starten.

- **Insiders :** Es sind Personen, die dazu legitimiert sind, das interne Netzwerk zu benutzen, aber die Systemressourcen in einer unerlaubten Art und Weise nutzen. Dies schließt auch autorisierte Benutzer mit ein, die ihre Zugriffsprivilegien missbrauchen, sogenannte „Missbraucher“, oder jene, die höher privilegierte Benutzer imitieren (z.B. durch das Benutzen eines fremden Terminals).

2.4.2 Kategorien von Systemschwächen

In diesem Abschnitt wird eine Kategorisierung der Systemschwächen angeben, die für Angriffe ausgenutzt werden können, z.B. Designschwächen, Implemtierungsfehler und Konfigurationsfehler.

Die Aufteilung der Mengen der Systemschwächen wird, in Anlehnung an [vHK98], in sieben Kategorien (K1-K7) angeben.

- **K1 - Konfigurationsfehler:** Konfigurationsfehler können entstehen, wenn die Einstellung der Parameter und Optionen falsch oder nicht vollständig vorgenommen werden, mit denen ein Programm ausgeführt wird (z.B. falsche gesetzte Zugriffsrechte).

- **K2 - Implementierungsfehler:** Programmierungsfehler von Betriebssystemen oder Programmen, die dazu führen, dass die Sicherheitskriterien aus Abschnitt 1.1 verletzt werden.

- **K3 - Designfehler im Kommunikationsprotokoll:** Wird bei der Planung der Programme oder Protokolle nicht genügend über die notwendigen Sicherheitsanforderungen nachgedacht, können Sicherheitsrelevante Designfehler entstehen. Z. B. Authentisierungsschwächen bei einem Kommmunikationsprotokoll.

- **K4 - Designfehler in der Dienstspezifikation:** Dies sind Fehler, die bei der Planung von Diensten oder Programmen, die z. B. auf TCP oder UDP aufsetzen, gemacht werden und Verletzungen der Sicherheitsanforderungen zur Folge haben.

- **K5 - Designfehler in der Anwendung:** Der Fehler beruht auf einer Schwäche im Sicherheitsmechanismus der Anwendung, wie z.B. eine schwache Authentisierungsprozedur.

- **K6 - Fehlverhalten der Benutzer:** Diese Fehler entstehen durch Benutzer, die sich nicht an die Vorgaben der Sicherheitspolitik halten (z. B. von außen leicht zugängliche oder öffentlich sichtbare Paßwörter).

- **K7 - Andere:** Hierunter fällt die Ausnutzung von regulären Eigenschaften der angebotenen Dienste in einer Art und Weise, die den Sicherheitsanforderungen zuwiderläuft.

2.4.3 Kategorien von Angriffsschäden

Im Folgenden sollen auch die möglichen Schäden angegeben werden, die durch ein Angriff verursacht werden können. Diese werden ebenfalls in Anlehnung an [vHK98] in Kategorien (S1-S10) unterteilt:

- **S1 - Angreifer erhält Information über das anzugreifende Ziel:** Schadensfolge ist, dass der Angreifer Information erhält, die der Vorbereitung eines gezielten Angriffs dienen.

- **S2 - Zugriff auf Netzressourcen:** Angreifer kann Firewall-Komponenten umgehen und somit Zugriff zum Netz erhalten, dies ermöglicht ihm heimlich Pakete durch das Netz zu schicken.

- **S3 - Zugriff auf Rechnerressourcen:** Die Folgeschäden sind abhängig von dem Zugriff und die dadurch gewonnen Benutzerrechte (z.B. Root-Rechte). Mögliche Schäden sind: Verlust der Vertraulichkeit, Integritätsverlust und der Verlust lokaler Ressourcen. Das kompromittierte System kann auch für weitere Angriffe auf andere Rechner missbraucht werden.

- **S4 - Verlust der Verfügbarkeit:** Hierbei geht es um den Verlust der Verfügbarkeit von Diensten und Daten.

- **S5 - Datenintegritätsverlust während des Transportes:** Dem Angreifer gelingtes die Daten während des Transportes zu verfälschen (z. B. Modifikation des DATA-Eintrages und der Prüfsumme bei TCP/IP).

- **S6 - Datenvertraulichkeitsverlust während des Transportes:** Versendete Daten können abgehört und mitgeschnitten werden.

- **S7 - Datenintegritätsverlust auf Rechner:** Dem Angreifer gelingt es, auf einem Rechner die gespeicherten Daten zu modifizieren.
- **S8 - Datenvertraulichkeitsverlust auf Rechner:** Unerlaubter lesender Zugriff auf die Daten, die auf einem Rechner gespeichert sind.
- **S9 - Datenechtheitsverlust:** Dem Angreifer gelingt es eine andere Identität vorzutäuschen (Verlust der Authentizität).
- **S10 - Denial-of-Service:** Dienste werden lahm gelegt und sind, je nach Schadensfolge für eine gewisse Zeit nicht Verfügbar.

2.4.4 Angriffsarten

Bei der Umsetzung eines Sicherheitskonzeptes besteht das Ziel darin, die Sicherheitsanforderungen

- Verfügbarkeit
- Vertraulichkeit
- Integrität
- Authentizität

durchzusetzen (vgl. Kapitel 1). Die Abbildung 2.23 ist eine schematische Darstellung der Angriffe und der Verletzungen der Sicherheitsanforderungen.

Diese Angriffe werden, auf Grund ihrer Eigenschaften, in die folgenden zwei Klassen eingeteilt [Gel99]:

- Passive Angriffe: Abhören
- Aktive Angriffe: Unterbrechung, Modifikation, Fälschung

Die passiven Angriffe führen zu keinen direkten beobachtbaren Schäden des Systemes und sind daher schwer zu entdecken. Die wenigen Möglichkeiten, einen passiven Angriff zu entdecken, wären, den Täter inflagranti zu ertappen oder durch die indirekten Folgen des Abhörens (z.B. wenn sensitive Daten des Unternehmens öffentlich bekannt werden). Bei aktiven Angriffen dagegen haben die Folgen des Angriffes Einfluss auf die Funktionalität des betroffenen Systems. Eine der notwendigen Voraussetzungen sind gute Kenntnisse über das Zielsystem seitens des Angreifers.

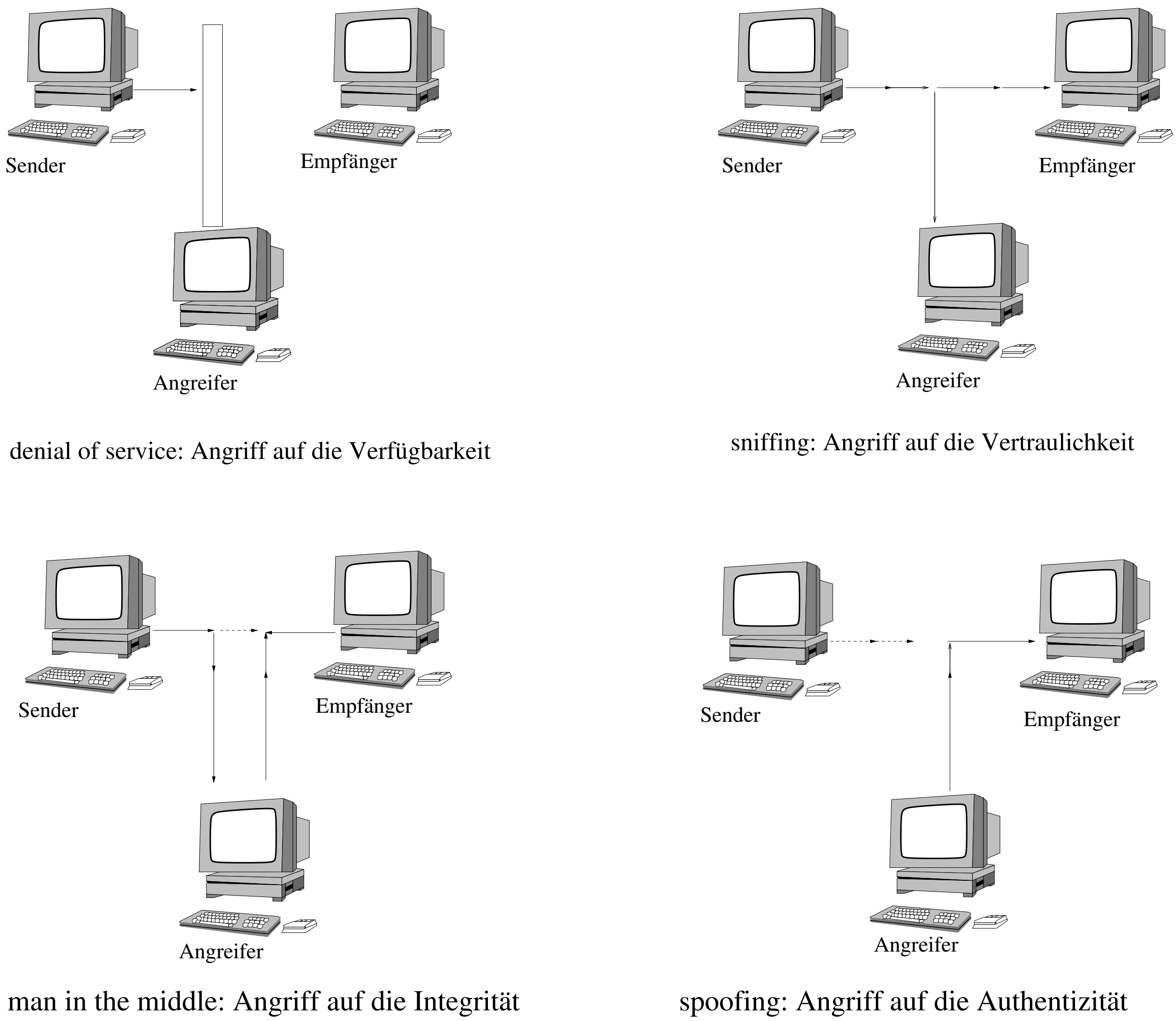

Abbildung 2.23: Verletzung gewünschter Systemeigenschaften [Gel99]

2.4.5 Angriffsplanung

Um einen systematischen Angriff zu Planen benötigt der Angreifer Informationen über das Zielsystem. Die Informationsbeschaffung an sich wird auch als ein Angriff bewertet, denn die Informationsbeschaffung kann einen Verlust der Vertraulichkeit beinhalten. Ein Angriff gliedert sich grob in zwei Schritte:

1. Angriffsplanung: Ausreichende Sammlung von Information zur erfolgreichen Durchführung des Angriffs.

2. Angriffsausführung.

Folgende Möglichkeiten stehen dem Angreifer für eine Angriffsplanung zur Verfügung [vHK98]:

1. TCP Port Scans

2. UDP Port Scans

3. Dienste, die Zusatzinformation liefern : finger, Identd etc.

Die in Punkt drei genannten Programme werden auch als "geschwätzige" Dienste bezeichnet. Neben den Techniken von Punkt 1 bis 3 zur Informationsbeschaffung, gibt es eine weitere Möglichkeit an Information zu gelangen:

"Social Engineering"

Hierbei wird die Gutgläubigkeit oder der Leichtsinn der Anwender ausgenutzt, um an Informationen zu kommen. Es wird angenommen, dass die meisten erfolgreichen Angriffe auf die Nachlässigkeit der Anwender zurückzuführen sind, da sie sich nicht an die vorgeschriebenen Maßnahmen der Sicherheitspolitik halten.

2.4.5.1 TCP Port Scan

Die Adressierung auf der Ebene von TCP erfolgt über sogenannte Sende- und Empfangsports. TCP-Ports adressieren die Dienstprogramme innerhalb eines Rechners. Wird für eine Anwendung eine TCP-Verbindung benötigt, muss die TCP-Portnummer bekannt sein, d.h., erst nach dem Aufbau der Verbindung unter Angabe der Portnummer, wird zum betreffenden Dienstprogramm verzweigt. Es gibt verschiedene Varianten des TCP Port Scans, die sich auf verschiedene Weise durchführen lassen.

Aktiver Port Scan : Hierbei wird versucht herauszubekommem, welche TCP-basierten Dienste ein Zielrechner anbietet. Dieses kann durch ein TCP Port Scan erreicht werden. Der aktive TCP Port Scan gründet sich auf den dreistufigen Startvorgang, dem sog. three-way-handshake, um eine TCP-Verbindung aufzubauen (siehe Abschnitt 2.3.2.3, Seite 90):

1. SYN-Paket wird an zu testenden Port des Zielsystems gesendet
2. Zielsystem schickt SYN/ACK-Paket zurück
3. Angreifer bestätigt mit ACK-Paket

Nachdem der Verbindungsaufbau erfolgreich war, ist für den Angreifer klar, dass der entsprechende Port des Zielsystems aktiv ist. Programme wie TCP-Wrapper können diese Art von Angriffen entdecken.

- **Angriff basiert auf :** K 7
- **Schadens-Kategorie :** S 1

Half Open Scan : Hierbei geht es darum, dass der Angreifer unentdeckt sein Vorhaben durchführt. Um dieses zu erreichen, werden nicht alle drei Stufen des TCP-Verbindungsaufbaus durchgeführt, sondern nur die ersten beiden. Der Angreifer weiß nach den beiden Schritten, dass ein Dienst auf dem entsprechenden Port wartet, aber da der letzte Schritt nicht ausgeführt wurde, war der Verbindungsaufbau nicht erfolgreich und wird auch nicht protokolliert. Wiederholt der Angreifer in kurzen Zeitintervallen die oben beschriebene Vorgehensweise, liegt SYN-Flooding vor (siehe 2.4.6.7). Um diesem entgegenzuwirken gibt es die Möglichkeit, Programme wie *tcplog* einzusetzen, die in der Lage sind auch fehlgeschlagene Versuche eines TCP-Verbindungsaufbaues zu protokollieren.

- **Angriff basiert auf :** K 7
- **Schadens-Kategorie :** S 1

Stealth Scanning : Ein Fin-Paket, das den einseitigen Verbindungsabbau einleitet, wird an das Zielsystem verschickt. Erhält der Angreifer ein RST-Paket als Antwort, kann er auf einen inaktiven Port schließen. Folgende Punkte, die Stealth Scanning auszeichnen, zeigen, dass die Entdeckung eines Angriffes dieser Art schwer ist, weil der Versuch eines Verbindungsaufbaues nicht stattfindet. Nach [vHK98] ist Stealth Scanning:

* Sehr schwer zu protokollieren
* Kann Firewall-Schutzmechanismen umgehen
* Ist nicht mit netstat zu erkennen
* Kein three-way-handshake

- **Angriff basiert auf :** K 3
- **Schadens-Kategorie :** S 1

RPC Stealth Scan : Ein Remote Procedure Call (RPC) ermöglicht die Ausführung von Prozeduren, die nicht auf dem lokalen Rechner vorliegen. Der Anwender, der eine gewünschte Prozedur innerhalb eines Programms aufrufen möchte, muss für diesen Zweck eine Anfrage an den Zielrechner schicken. Die Anfrage (*initial query*) enthält die benötigten Programm-, Prozedurnummern und entsprechende Argumente, mit denen die Prozedur ausgeführt werden kann. Die Besonderheit der Prozedurnummer 0 ist, dass zu dieser Nummer immer eine Prozedur existiert, so dass ein Angreifer nach aktiven Programmnummern auf einem Zielsystem suchen kann, indem er alle Programmnummern nach der Prozedur 0 abfragt. Diese Prozedur hat zwar keine Argumente und liefert auch keinen Wert, aber ein missglückter Aufruf übermittelt einen Fehlercode. Der Informationsgehalt dieses Fehlercodes ist:

- Programm ist inaktiv.

Auf diese Weise kann nach aktiven oder inaktiven Programmen gesucht werden. Vorher aber braucht der Angreifer die Portnummer, auf der ein RPC-Server läuft. Dies kann er durch einen Port Scan ermitteln werden. Nachdem ein aktiver Port gefunden wurde, kann der Angreifer sein Vorhaben starten und mit der Programmnummer j nach Prozedur 0 fragen. Falls die Anfrage positiv ausfällt, d.h., nicht abgewiesen wurde, weiß der Angreifer, dass hinter diesem Port sich ein RPC-Dienst mit Programmnummer j befindet.

- **Angriff basiert auf :** K 7
- **Schadens-Kategorie :** S 1

2.4.5.2 UDP Port Scans

Im Gegesatz zu TCP ist UDP ein verbindungsloses Protokoll und benötigt deshalb auch keine Verbindungsaufbauprozedur. Daher ist es viel schwerer nach aktiven UDP-Diensten zu suchen. Vielmehr wird hier die Möglichkeit ausgenutzt nach inaktiven UDP-Ports zu suchen. Ein Rechner, der auf einem bestimmten Port keinen UDP-Dienst anbietet, wird auf eine Anfrage an entsprechenden Port eine ICMP-Fehlermeldung „Port Unreachable“ erhalten. Der Angreifer kann nun von den inaktiven Ports auf die aktiven schließen. Mit dieser Information kann nun der Angreifer die gefundenen Ports gezielt daraufhin überprüfen, welche Programme und welche Versionen dieser Programme mit jedem dieser Ports angesprochen werden können.

- **Angriff basiert auf :** K 7
- **Schadens-Kategorie :** S 1

2.4.5.3 Suche nach verwertbarer Information

Viele Angriffe beginnen damit, dass die Angreifer sich Informationen über das anzugreifende System verschaffen, z. B. mit den oben dargestellten Methoden. Einige angebotene Dienste bieten zusätzliche Informationen.

Finger, rusers, whois: Diese Dienste liefern Informationen über den Rechner und die momentan aktiven Benutzer eines Systems. Sie gehören zu den schon weiter oben kurz erwähnten "geschwätzigen Diensten".

Netstat: Dieser Dienst liefert Angaben über den Netzstatus und ermöglicht dem Angreifer somit, Informationen über aktuell geöffnete TCP oder UDP-Verbindungen zu bekommen. Dem Angreifer wird dadurch die Möglichkeit geboten, nach aktiven Ports Ausschau zu halten.

- **Angriff basiert auf :** K 4
- **Schadens-Kategorie :** S 1

Rstatd, sysstat : Hier werden Informationen über den aktuellen Ressourcenverbrauch und über die momentan aktiven Prozesse einer bestimmten Maschine gesammelt.

Identd : Ist dieser Dienst auf ein Client installiert, bietet er die Möglichkeit, dass bei Bedarf Informationen darüber verschickt werden, welcher Benutzer welchen Port benutzt.

Rwho : Dieser Dienst liefert Informationen über die *remote users* im System.

- **Angriffe basieren auf :** K 4
- **Schadens-Kategorie :** S 1

In dem folgenden Abschnitt sollen die verschiedenen Angriffsmethoden dargestellt werden. Als Abstraktionsmodell wird auf das OSI-Referenzmodell verwiesen. Die verschiedenen Angriffe werden gemäß ihres Abstraktionsgrades in drei Ebenen unterteilt, diese werden auch als Dienstebenen bezeichnet :

- Angriffe auf der unterer Dienstebene (ARP,IP,ICMP etc.)
- Angriffe auf der mittleren Dienstebene (DNS,NFS,NIS etc.)
- Angriffe auf der oberen Dienstebene (sendmail, recursive finger etc.)

Diese Unterteilung der Angriffe, in verschiedenen Ebenen wird nicht notwendigerweise disjunkt sein, d. h., ein Angriff kann sich über mehrere Ebenen hinweg erstrecken. Dies liegt daran, das ein Angriff sich in mehrere Phasen teilt, die jeweils verschiedenen Ebenen zugeordnet sein können. In Abhängigkeit von den Abstraktionsgrad ist die Menge der Daten, die bei einem Angriff erzeugt wird, unterschiedlichem Umfangs. Bei den unteren Dienstebenen reichen „primitive'Äktionen aus, um den Angriff durchzuführen und die erzeugten Datenmengen sind geringer (Betriebssystem nah). Auf den oberen Dienstebene wird mit wenigen Befehlen sehr viel Aktivität (Handlungen) im System ausgelöst (d.h. es werden große Mengen von Daten erzeugt) und nutzen somit Schwächen in den höheren Abstraktionsebenen aus, d. h., diese Angriffe haben wenig mit den betriebssystemspezifischen Schwächen zu tun, sondern mit den Schwächen der Anwendungen selber.

2.4.6 Angriffe auf der unteren Dienstebene

Auf den unteren Protokollebenen des OSI-Referenzmodells werden viele Angriffe gestartet. Der Vorteil für Angreifer ist, dass auf dieser Ebene nur eine schwache Protokollierung, und somit auch nur eine schwache Sicherheitskontrolle. Sicherheitsmechanismen, wie Authentisierung, greifen erst in den höheren Ebenen.

2.4.6.1 Angriffe auf ARP-Ebene

Ethernet-Technologien werden in lokalen Netzen (LANs) eingesetzt. Bei Anbindung eines LANs an das Internet wird das ARP(Adress Resolution Protocol) benötigt. Das ARP dient in Ethernet-Netzwerken dazu, einer Internet-Adresse zugeordnete Ethernet-Hardware-Adresse zu finden, d.h., zu einer 32-Bit IP-Adresse die zugehörige 48-Bit Ethernet-Adresse (MAC-Adresse)[9]. Die Umsetzung findet folgendermaßen statt: Benötigt ein Rechner die Ethernet-Adresse zu einer gegebenen IP-Adresse, wird an alle angeschlossenen Rechner ein ARP-Paket in Form von *Ethernet Broadcast* gesendet, das die gefragte IP-Adresse und die Ethernet-Adresse des Absenders enthält (sog. Arp-Request). Nur der entsprechende Rechner meldet sich auf diese Anfrage, indem er seine eigene Ethernet-Adresse an den anfragenden Rechner sendet (Arp Reply). Damit nicht für jedes zu übertragende Paket eine ARP-Anfrage benötigt wird, speichert jeder Host in seinem lokalen Zwischenspeicher die bisherigen Antworten (ARP-Tabelle) für eine gewisse Zeitspanne (z. B. 20 min.).

ARP-Spoofing : Bei dieser Art von Angriffen wird der ARP-Cache verändert. Ein Angreifer kann in der Zeitspanne, bis die Tabelleneinträge gelöscht und aktualisiert werden, die ARP-Tabelle mit falschen Einträgen versorgen. Dies ist

[9]MAC-Adresse *Media Access Control* ist die eindeutige Hardwareadresse einer Ethernetkarte

deswegen möglich, weil auf ARP-Ebene keine Authentisierung stattfindet. Ein Angreifer kann auf eine Arp Request mit seine Ethernet-Adresse antworten und vorgeben die gesuchte IP-Adresse zu besitzen (Identität vortäuschen). Von diesem Zeitpunkt an werden alle Pakete vom Ziel an die Ethernet-Adresse des Angreifers geleitet.

- **Angriff basiert auf :** K 3
- **Schadens-Kategorie :** S 5, S 6, S 10

2.4.6.2 Angriffe auf IP-Ebene

IP Fragmentierung : Bei zu großen IP-Datenpaketen werden diese nicht im Ganzen übertragen und müssen daher fragmentiert werden. Diese Eigenschaft der TCP/IP-Implementierung kann von einem Angreifer ausgenutzt werden. Der Angreifer sorgt dafür, dass die IP-Pakete so fragmentiert werden, dass die TCP-Header-Informationen über zwei IP-Datenpakete verteilt sind. Das erste Fragment enthält den IP-Header und die ersten Bytes des TCP-Headers, mit Quell-, Sendeport- und Sequenznummer. Erst im zweiten Fragment folgen weitere Teile des TCP-Headers, die das, für das Filtern von TCP-Verbindungen nötige SYN-Bit und das ACK-Bit enthalten. Im ersten Fragment fehlt somit die entscheidende Filter-Information (SYN-Bit), und das erste Fragment des IP-Paketes kann nicht rausgefiltert werden und passiert somit die Firewall.

- **Angriff basiert auf :** K 3
- **Schadens-Kategorie :** S 2

Falsche IP-Parameter : Eine Reihe von IP-spezifischen Angriffen fallen unter diese Rubrik und stellen die eine Gefahr für viele System dar, z.B. wenn Quell- und Zieladresse identisch sind. Ein anderer Punkt ist die Wahl falscher IP-Paket-Längen. Ein IP-Paket kann maximal 65.535 Byte lang sein. Wird nun diese Länge überschritten und in fragmentierter Form übertragen, entstehen Probleme bei dem Versuch, die zusammengesetzten Fragmente zu verarbeiten, und es kommt zum Systemabsturz (ping of death). Fehlerhafte TCP/IP-Implementierungen sind Ursache für den Erfolg dieser Angriffe.

- **Angriff basiert auf :** K 2
- **Schadens-Kategorie:** S 10

SYN mit unerreichbarer Quelladresse : Wie bei SYN-Flooding täuscht der Angreifer einen TCP-Verbindungsaufbauwunsch vor, indem er ein SYN-Paket sendet. Mit dem Unterschied, dass das SYN-Paket gefälschte und unerreichbare IP-Quelladressen enthält. Das betreffende System reserviert Betriebsmittel und verschickt ein SYN/ACK-Paket und wartet auf eine Bestätigung, die nicht erfolgt. Gelingt es dem Angreifer, die obige Prozedur in kurzen Abständen zu wiederholen, werden zu viele Rechnerressourcen verbraucht und es kommt zur Überlastung oder Absturz des Systems.

- **Angriff basiert auf :** K 3
- **Schadens-Kategorie :** S 10

2.4.6.3 Angriffe, die auf ICMP basieren

Die Funktionsweise des *Internet Control Message Protocol* (ICMP) ist in Abschnitt 2.3.2.2 genauer beschrieben. Im Folgenden sollen ICMP-Nachrichten, die für Angriffe ausgenutzt werden können, betrachtet werden.

ICMP Redirect : Der ICMP-Befehl, der eigentlich dazu dienen soll, einem Rechner mitzuteilen, welche Wegewahl zum Zielrechner besser ist, kann für einen Angriff genutzt werden. Akzeptiert der Empfänger diesen ICMP-Befehl, dann wird der Eintrag in seiner Routing-Tabelle geändert. Dies hätte zur Folge, dass durch gefälschte Redirect-Pakete das anzugreifende System dazu veranlasst wird, zukünftig eine andere Route zu verwenden, die z.B. nicht durch eine Firewall geschützt ist.

- **Angriff basiert auf :** K 3
- **Schaden-Kategorie :** S 5, S 6

ICMP Destination Unreachable : Die ICMP-Nachrichten des Types „destination unreachable“ oder „Time to Live exceeded“ 2.3.2.2 können für Denial-of-Service Attacken missbraucht werden, indem der Angreifer einige Pakete mit falschen Adressen, die den Ausfall des Empfängers oder eines Teilnetzes vortäuschen, an ein System sendet. Der Sender nimmt folglich an, dass der Empfänger nicht erreichbar ist und baut keine Verbindung mehr mit dem betreffenden Rechner auf.

- **Angriff basiert auf :** K 3
- **Schadens-Kategorie :** S10

ICMP Echo Request (ping of death) : Der Angreifer nutzt hierbei die Tatsache aus, dass die meisten ICMP-Implementierungen nur mit Paketen umgehen können, die nicht von der, in der Spezifikation geforderten Maximalgröße, abweichen. Daher versendet der Angreifer ICMP-Echo Request Pakete mit einer kritischen Größe, was zur Folge hat, dass das Zielsystem beim Empfang dieser Pakete zusammenbricht und nicht mehr verfügbar ist.

- **Angriff basiert auf :** K 2
- **Schadens-Kategorie :** S 10

ICMP Tunnel : Das Vorhaben bei einem ICMP-Tunnel ist es, im Datenfeld eines *ICMP-Echo Request* Daten unterzubringen, die auf dem Zielsystem von einem Server interpretiert werden. Dieser Vorgang ist nicht ICMP-spezifisch, auch andere Protokolle weisen dieses Verhalten auf. Das Problem hierbei ist, dass seitens der Administratoren ein *ICMP-Echo Request* Paket eher unkritisch eingestuft wird und infolgedessen keine Filterung stattfindet. Dies hat zur Folge, dass trotz aktiver Firewall, unter Verwendung solcher Pakete, Daten übermittelt werden können. Voraussetzung für solch einen Angriff ist aber, dass dem Angreifer vorher gelungen ist, auf dem Zielsystem einen Server zu installieren, der diese Pakete entgegennimmt und interpretiert.

- **Angriff basiert auf :** K 1, K 4
- **Schadens-Kategorie :** S 3, S 4, S 7, S 8, S 10

2.4.6.4 Spoofing

Unter Spoofing versteht man die Vortäuschung falscher Identitäten, Rechnernamen oder ähnliches. Von Spoofing-Angriffen können verschiedene Dienste betroffen sein, z.B. ARP Spoofing, denn der Angreifer täuscht dem Router eine falsche Identität vor.

IP Spoofing : Der Fehler beruht darauf, dass bei vielen Protokollen der TCP/IP-Familie die Authentisierung der kommunizierenden Systeme nur über die IP-Adresse erfolgt, die aber leicht verfälscht werden kann. Somit fehlt dem Zielsystem die Möglichkeit, die Authentizität des Absenders zu überprüfen. Dies ermöglicht dem Angreifer, durch ein IP-Spoofing, Zugriff auf einen Rechners zu erhalten, indem dieser die Identität eines anderen Rechners, durch Übernahme dessen IP-Absender-Adresse, übernimmt.

- **Angriff basiert auf :** K 3
- **Schadens-Kategorie :** S 2

TCP Sequenznummernvorhersage : Diese Angriffe werden in Kombination mit IP-Spoofing ausgeführt. Der Angreifer nutzt hierbei aus, dass die von den Rechnern zur Synchronisation beim Aufbau einer TCP/IP-Verbindung verwendeten, Sequenznummern leicht zu erraten ist, weil diese Nummern aufgrund eines einfachen Algorithmus ermittelt werden. Daher ist es möglich, Pakete mit jeder beliebigen Absender Adresse zu verschicken, ohne je ein Antwortpaket von dem missbräuchlich benutzten Rechner zu erhalten (vgl. Abschnitt 2.3.2.3 und Abb. 2.21). Dieser Vorgang soll in drei Schritten (S1-S3) verdeutlicht werden :

S1: Ein Angreifer X sendet einem Zielsystem Z ein Datenpaket mit dem Inhalt :

X>>**Z** : gesetzter *SYN-Bit,* $SEQ.NR_X$

Das Zielsystem Z antwortet mit :

Z>>**X** : *SYN-Bit,* $SEQ.NR_Z$, $ACK(SEQ.NR_X,)$

In diesem Schritt hat der Angreifer die Sequenznummer ($SEQ.NR_Z$) des Zielsystems ermittelt.

S2: Der Angreifer baut eine neue Verbindung zu dem Zielsystem auf, täuscht dabei aber die Identität eines internen Systems A vor, dessen IP-Adresse er zuvor durch IP-Spoofing ermittelt hat :

X_A >>**Z** : gesetzter *SYN-Bit,* $SEQ.NR_X$

Die Antwort des Zielrechners geht aber an die interne Station A :

Z>>**A** : *SYN-Bit,* $SEQ.NR_Z+$, $ACK(SEQ.NR_X,)$

S3: Obwohl die Antwort des Zielsystems Z den Angreifer X nicht erreicht, kann der Angreifer, ausgehend von dem Wert $SEQ.NR_Z$ (aus Schritt 1), die Sequenznummer $SEQ.NR_Z+$ berechnen, die das Zielsystem Z an A übermittelt hat. Er sendet also dem Zielsystem folgendes Datenpaket:

$\mathbf{X}_A$ >>**Z** : $ACK(SEQ.NR_Z+)$

Die Berechnung von $SEQ.NR_Z+$ beruht auf der schon erwähnten Schwäche bei der Vergabe von Anfangssequenznummern. Bei der Berkeley-TCP-Implementierung wird z.B. die Sequenznummer, innerhalb einer Verbindung jede Sekunde um den Wert 128, und für jede neue Verbindung um den Wert 64, erhöht.

- **Angriff basiert auf :** K 3
- **Schadens-Kategorie :** S 3

DNS Spoofing : Siehe Abschnitt 2.4.7.1

RIP Spoofing : Aufgrund fehlender Authentisierung ist es möglich modifizierte RIP-Pakete an ein Router zuversenden, der diese in seine Tabelle aufnimmt. Somit kann der Angreifer einen „man-in-the-middel“ Angriff starten, d.h., durch die gefälschten Routing Informationen kann er die Kommunikationsverbindung über das eigene System umleiten. Die abgefangenen Pakete können ausgewertet und danach an das eigentliche Ziel weitergeleitet werden.

- **Angriff basiert auf :** K 3
- **Schadens-Kategorie :** S 5, S 6, S 10

2.4.6.5 Source Routing :

Das Internet-Optionsfeld stellt Routing-Funktionen zur Verfügung, die dem Absender erlauben, festzulegen, auf welchem Wege das Paket den Zielrechner erreichen soll. Gelingt es dem Angreifer die Identität (IP-Spoofing) eines vertrauenswürdigen Rechners anzunehmen und aktiviert er die Source-Routing Funktion über seine eigene IP-Adresse, kann er alle Rechte des vertrauenswürdigen Rechners aneignen.

- **Angriff basiert auf :** K 3
- **Schadens-Kategorie :** S 5, S 6,

2.4.6.6 Session Hijacking

Ein Angreifer versucht, eine bestehende Verbindung, die die nötigen Prozeduren der Authentisierung vollzogen hat, zu übernehmen. Die nötigen Informationen kann der Angreifer z.B. durch Sniffing einer Telnet Sitzung erhalten, d. h., unter anderem erhält er Passwörter oder auch aktuelle Sequenznummern. Diese werden benötigt, um eine Verbindung zwischen Client und Server zu synchronisieren. Gelingt es dem Angreifer an die Sequenznummern zu gelangen, die der Server und Client erwarten, kann er mit beiden kommunizieren, oder er versucht einen nicht-synchronisierten Zustand zu erreichen, um nur mit einer Partei zu kommunizieren. Dies kann er erreichen indem er modifizierte Pakete schickt, die die Sequenznummer des Opfers erhöhen.

- **Angriff basiert auf :** K 3
- **Schadens-Kategorie :** S 3

2.4.6.7 Überfluten (Flooding)

Unter Flooding versteht man das Überfluten von Systemen mit riesigen Datenmengen, die durch Konfigurationsfehler, oder auch Schwächen im Selbstschutz der Systemkomponenten, begünstigt werden.

- **Angriff basiert auf :** K 1, K3, K4, K5
- **Schadens-Kategorie :** S 10

Mail Flooding : Bei Mail-Flooding werden große Datenmengen per E-Mail an das Zielsystem verschickt. Dies führt dazu, dass das Spool-Verzeichnis überläuft oder andere Arten der Überlastung auftreten.

- **Angriff basiert auf :** K 1, K 7
- **Schadens-Kategorie :** S 10

SYN Flooding : Hierbei geht es, wie bei Half Open Scans, um einen Verbindungsaufbau, bei dem der letzte Schritt, der „three-way-handshake“, nicht ausgeführt wird. Der Angreifer versucht, diese Prozedur so lange auszuführen, bis das Zielsystem, unter der großen Anzahl der Verbindungswünsche, die nicht bestätigt werden, überlastet ist, weil das System zu viele Betriebsmittel für halboffene Verbindungen reserviert. Dadurch stehen die entsprechenden Dienste nicht mehr zur Verfügung.

- **Angriff basiert auf :** K 3
- **Schadens-Kategorie :** S 10

Data Flooding : Hierunter fallen alle Angriffstechniken, die durch Erzeugen und Versenden von riesigen Datenmengen das Zielsystem überlasten. Das Zielsystem kann diese Daten nicht mehr korrekt verarbeiten.

- **Angriff basiert auf :** K 7
- **Schadens-Kategorie :** S 10

System Log Flooding : Diese Vorgehensweise dient der Angriffsvorbereitung, d.h., der Angreifer versucht, vor dem eigentlichen Angriff, den Protokollmechanismus des Zielsystems außer Kraft zu setzen. Zu diesem Zweck führt er Aktionen durch, die große Protokolldatenmengen beim Zielsystem erzeugen. Dadurch läuft der Protokollspeicher über. Bei schlecht konfigurierten Systemen wird als Folge einer solchen Aktion der Protokollmechanismus abgeschaltet werden, somit können verdächtige Aktionen nicht aufgezeichnet werden.

- **Angriff basiert auf :** K 1, K 7
- **Schadens-Kategorie :** S 10

Open/Close Flooding : Das schnelle Öffen und Schließen von Verbindungen kann zur Überlastung des Zielsystems führen, so dass betreffende Dienste nicht mehr zur Verfügung stehen.

- **Angriff basiert auf :** K 7
- **Schadens-Kategorie:** S 10

2.4.7 Angriffe auf der mittleren Ebene

Hier werden Angriffe betrachtet, die auf Stufe 7 des OSI-Referenzmodells angesiedelt werden können.

2.4.7.1 DNS-Angriffe

Ein*domain-name-service* besteht aus:

- *Domain-Namens-Raum :* Beschreibt die Zuordnung von Rechnernamen zu Internetadressen und umgekehrt.
- *DNS-Server (Name Server):* Verwaltet die Domain-Bäume und/oder Teilbäume
- *Resolver :* Schnittstelle, die die Anfragen eines Clients an den DNS-Server weiterleitet, die Antworten interpretiert und an den Client übermittelt.

Der DNS (Domain-Name-Service) ist für die Zuordnung der Rechnernamen zu ihren numerischen Internetadresse zuständig (z.B. yahoo.com <> 204.71.177.72). Wird zu einem Rechnernamen die zugehörige Internetadresse benötigt, so wird ein DNS-Anfrage-Paket erzeugt und an den DNS-Server gesendet. Dieser überprüft, ob die nachgefragte Adresse sich auf der lokalen Domäne befindet. Bei einem negativen Ergebnis wird die Anfrage an den nächsten Server in der übergeordneten Domäne weitergeleitet.
Viele Angriffe auf DNS nutzen bestimmte Schwächen des DNS-Servers aus. Der DNS-Server kann nicht überprüfen, von welcher Gegenstelle die Antworten auf seine Namens- oder Adressanfragen kommen. Meist werden mit diesen Antworten zusätzliche Informationen an den DNS-Server gesendet, die er nicht explizit angefordet hat. Der DNS-Server ignoriert diese Informationen nicht und aktualisiert, entsprechend der Antwort und der Zusatzinformation, seine Datenbank. Auf diese Weise kann ein Angreifer die Datenbank mit falschen Informationen modifizieren und Angriffe starten. Meistens werden Angriffe dieser Art dazu verwendet, um die Identität eines vertrauenswürdigen Rechners oder Servers vorzutäuschen.

DNS-Spoofing : Beim DNS-Spoofing ändert der Angreifer beim DNS-Server die Tabellen zur Zuordnung von Rechnernamen und IP-Adressen. Diese Änderungen werden in die Übersetzungstabellen auf dem DNS-Server geschrieben. Wenn ein Client die zugehörige IP-Adresse zu einem Rechnernamen anfordert, erhält er eine gefälschte Adresse, die die IP-Adresse eines Rechners ist, der sich unter der Kontrolle des Angreifers befindet.

- **Angriff basiert auf :** K 4
- **Schadens-Kategorie :** S 6, S 9

DNS-Cache Verunreinigung : Der Domain-Namens-Raum (siehe oben) bietet spezielle Einträge, die die Verwendung von Alias-Namen (CNAME-Record) unterstützen. Rechnernamen, wie „www.sae16m.kompliziert.de", die schwerer zu merken sind, können mit leicht merkbaren Namen wie „ www.einfach.de" angesprochen werden. Wird das Alias-Record eines DNS-Datensatzes mit falschen Informationen versorgt, spricht man von Cache-Verunreinigung.

- **Angriff basiert auf :** K 4
- **Schadens-Kategorie :** S 8, S 9

Reverse Lookup : Reverse Lookup bietet einem System die Möglichkeit durch Komplementäre Anfragen die Konsistenz der DNS-Server-Antworten zu überprüfen:

Beispiel 1 (Reverse Lookup)
- ***System-Anfrage*** = *Welche IP-Adresse hat Rechner www.diplom.de ?*
- ***DNS-Server-Antwort*** = *Rechner www.diplom.de hat die IP-Adresse : 192.168.0.1*
- ***Komplementäre-System-Anfrage*** = *Wie heißt der Rechner mit der IP-Adresse 192.168.0.1 ?*
- ***DNS-Server-Antwort*** = *Der Rechner mit der IP-Adresse 192.168.0.1 heißt : www.diplom.de*

Der Angreifer macht sich zunutze, das einige Systeme diese Reverse-Lookup-Verfahren nich durchführen, so dass er seinem Opfer leichter eine gefälschte DNS-Antwort unterschieben kann.

- **Angriff basiert auf :** K 1
- **Schadens-Kategorie :** S 9

2.4.7.2 NIS

Das NIS (Network Information System) ist ein von der Firma Sun entwickeltes Protokoll, das auf RPC aufsetzt. Es dient der Übermittlung von wichtigen Dateien von zentralen NIS-Servern zu den NIS-Clients, dazu gehören unter anderem die Passwortdatei oder Host-Adresstabellen. NIS führt aber keine Authentisierung auf RPC-Ebene durch. Dieses ermöglicht einem Angreifer, mittels Sniffing in den Besitz der von NIS übertragenen Systemdateien zu kommen, oder mittels Spoofing dem NIS-Client einen NIS-Server vorzutäuschen. Der Spoofing-Angriff verläuft folgendermaßen :

- **N**IS-Client sendet eine RPC-Anfrage an den NIS-Server.
- **D**er Angreifer versucht vor dem NIS-Server dem NIS-Client eine gefälschte RPC-Antwort zukommen zu lassen. Diese gefälschte RPC-Antwort kann z.B.die NIS-Adresstabellen ändern. Dem Angreifer wird dadurch ermöglicht in der Datei *.rhost* des Opfers einen Rechnernamen durch seine eigene zu ersetzen und sich somit Zugriff auf den NIS-Client zu verschaffen (z.B. mittels *rlogin*).
- **Angriff basiert auf :** K 4
- **Schadens-Kategorie :** S 3

2.4.7.3 FTP

FTP (File Transfer Protocol) wird für den Austausch von Dateien zwischen entfernten Rechnern benutzt und ist einer der am häufigsten benutzten Internet Dienste. FTP baut auf TCP auf und braucht bei der Durchführung einer Datenübertragung zwei gleichzeitig aktive Verbindungen :

- **1. Verbindung:** Diese Verbindung wird von einem beliebigen Port des Clients zum Standard FTP-Steuer-Port 21 des FTP-Servers aufgebaut. Über diesen Port werden die Befehle übertragen. Der Client übermittelt, mit Hilfe des Port-Befehls, über diese Steuerleitung die eigene Portnummer.
- **2.Verbindung:** Der Server baut vom Standard-FTP-Datenport 20 aus eine TCP-Verbindung zum Client-Datenport auf. Port 20 dient also der Übertragung der Daten. Über diese Verbindung werden dem Client die angeforderten Daten übermittelt.

Sicherheitsrelevant ist die Tatsache, dass keine Überprüfung der Identitäten des Paares (Client,Server) aus Schritt 1 in dem nachfolgenden Schritt 2 stattfindet.

Man unterscheidet zwei Angriffsmethoden auf das FTP-Protokoll. Die eingesetzte Methode hängt davon ab, ob der FTP-Server im aktiven oder passiven Modus arbeitet. Demnach kann man auch die Angriffsmethoden nach aktiven oder passiven Angriffen unterscheiden.

Passiver Angriff: Angreifer versucht, die Portnummer auf der Server-Seite zu erraten. Wenn dem Angreifer dies gelungen ist, kann er eine Verbindung zu diesem Port aufnehmen. Der Server merkt nicht, dass über denselben Port zwei verschiedene Clients Daten übertragen. Das Erraten der Portnummer ist leicht, weil diese gewöhnlich in direkter, aufreihender Folge vergeben wird. Durch eine normale FTP-Verbindung, mit der man die Portnummer des Datenkanals auf dem Server in Erfahrung bringen kann, kann man die Portnummern von anderen Verbindungen zu anderen Clients berechnenen. Der Angriff beim aktiven Modus ist ähnlich aufgebaut.

Zu bemerken ist, dass der Server bei FTP-Verbindungen das privilegierte Port 20 benutzt und daher als „root“ gestartet wird. Dies stellt ein potentielles Sicherheitsproblem dar.

- **Angriff basiert auf :** K 4
- **Schadens-Kategorie :** S 8, S9

2.4.7.4 HTTP

Das Hypertext Transfer Protocol (HTTP) ist das Protokoll des World Wide Webs und wird für die Übermittlung von Daten zwischen HTTP-Server und HTTP-Client (Browser) verwendet. Es handelt sich um ein Protokoll der Anwendungsebene und benutzt die TCP/IP-Protokolle als Transportprotokoll. Das Problem der Klartext-Übertragung wurde durch Verschlüsselungsmethoden behoben. Das HTTP-Protokoll hat keine bekannten Schwächen und trotzdem gibt es Möglichkeiten den HTTP-Dämon zu missbrauchen. Z.B. kann der Passwort-Autorisierungsmechanismus durch zusätzliche „Slash’s“ in der URL überwunden werden.

- **Angriff basiert auf :** K 2, K 5
- **Schadens-Kategorie :** S 8

2.4.7.5 X11

Das X11-Protokoll ermöglicht einer Anwendung, die auf dem lokalen Rechner läuft, ihre Eingaben von einem entfernten Rechner zu erhalten. Jedes System, welches das X-Protokoll aktiviert hat, dient als X-Server. Der Einsatz von X-Servern bringt Risiken mit sich, die durch Schwachstellen bei der Authentisierung nicht-autorisierten Benutzern die Verbindung mit dem X-Display ermöglicht. Die Deaktivierung des xhost-Modus bewirkt das Abschalten der Zugangskontrolle, und jeder Nutzer im Netz kann eine Verbindung zum X-Server aufbauen. Besitzt ein Angreifer bereits einen Zugang zum Zielrechner, kann er trotz aktivierter Zugangskontrolle über den *xwd*-Befehl die Bildschirminhalte in einer Datei abspeichern und mit *xwud*-Befehl wieder auslesen. Wird die Authentifizierung über sogenannte Magic-Cookies realisiert, die an die Clients verteilt und in der *.Xauthority*-Datei gespeichert wird, ist der Schutz des Authentizierungsmechanismus abhängig von dem Schutz dieser Datei. Gelingt es dem Angreifer eine gültige Verbindung zum X-Server aufzubauen, kann er :

* Fenster zerstören (Bildschirm löschen oder mit beliebigen Zeichenfolgen füllen)
* Neue Fenster erzeugen
* Tastatureingaben und Passwörter abhören
* Jede mögliche Aktivität in der X-Umgebung ausführen

- **Angriff basiert auf :** K 1
- **Schadens-Kategorie:** S 8

2.4.7.6 NFS-Network File System

NFS ist ein von der Firma Sun entwickeltes Protokoll, welches den Zugriff von einem NFS-Client auf Dateien und Verzeichnisse, die sich auf einem NFS-Server befinden, erlaubt. Für die Zugriffe auf den entfernten Server wird das RPC-Protokoll eingesetzt als Transportprotokoll für die RPC-Anfragen wird UDP oder TCP verwendet. Die Zugriffe werden über zwei Protokolle realisiert:

mount: Hier wird bestimmt, welche Partition für welche Rechner verfügbar sein soll. Auch die Vergabe der Zugriffsrechte auf die Daten für die einzelnen Clients wird hier festgelegt. Der Client erhält zu diesem Zweck eine Zugriffsnummer (file-handle). Diese wird für alle weiteren NFS-Anfragen benutzt.

NFS: Das NFS-Protokoll regelt alle konkreten Zugriffe auf ein Dateisystem des Servers. Hier werden die notwendigen Zugriffsrechte auf ein Dateisystem, die bei Zugriffen auf Dateien benötigt werden, festgelegt (z. B. Dateien lesen, schreiben und Verzeichnisse anlegen.

Die Sicherheitsschwachstellen bestehen in der schwachen Authentisierung zwischen Client und Server und in der unverschlüsselten Übertragung der Daten. Folgende Sicherheitsprobleme, die den unautorisierten Zugang zum System und das Abhören der Daten ermöglichen, können auftreten:

→ File-handle kann geraten werden.

→ Missbrauch der file-handle. Besitzt ein Angreifer ein korrektes file-handle, kann er auf die exportierten Dateisysteme gemäß der UNIX-Zugriffsrechte zugreifen. Ein Problem ist auch, dass die file-handle Nummern solange ihre Gültigkeit behalten, bis das Dateisystem des Servers reinstalliert wird.

→ IP-Adressen werden für die Authentisierung während des mount-Vorganges verwendet. Das Problem der Authentisierung auf der Basis von IP-Adressen wurde in den vorherigen Abschnitten dargestellt.

→ Ist ein Dateisystem gemountet, kann jeder Rechner, der das korrekte file-handle besitzt, auf das Dateisystem zugreifen. Ganz kritisch wird es, wenn es sich um ein root-file-handle handelt, das dem Angreifer den uneingeschränkten Zugang zu dem root-Verzeichnis erlaubt.

- **Angriff basiert auf :** K 2, K 4, K 5
- **Schadens-Kategorie :** S 4, S 8, S 7

2.4.8 Angriffe auf der oberen Ebene

Implementierungsfehler in der Anwendungsebene lassen sich für viele Angriffe ausnutzen. Beipiele für diese Angriffstypen werden im Folgenden betrachtet.

2.4.8.1 Sendmail

Sendmail-Anwendung ist ein Bestandteil jedes UNIX-Systems und stellt den SMTP-Dienst zur Verfügung. Das Problem ist, dass aufgrund der Komplexität des Sendmail-Programms administrative Privilegien benötigt werden, um die Aufgaben zu erfüllen. Diese Tatsache und Implemtationsfehler, vor allem in älteren Versionen, ermöglichen potentiellen Angreifer root-Rechte zu erlangen. Ein Beispiel (Vorraussetzung ist, dass der Angreifer ein account auf dem Zielsystem besitzt) :

> `cp Programm`[10]/var/spool/mail/root (kopiert dieses Programm in die Datei root)

> `chmod +sxr /var/spool/mail/root` (setzt die Ausführungsrechte (setuid) für die Datei root)

> `touch X` (eine leere Datei wurde erzeugt)

> `mail root < X` (die leere Datei X wird verschickt und wird von sendmail in das Verzeichnis /var/spool/mail/root/X angehängt)

> `/var/spool/mail/root/X` (mit diesem Befehl wird eine root Shell eröffnet und der Angreifer kann im System als root agieren)

- **Angriff basiert auf :** K 2, K 5
- **Schadens-Kategorie :** S 3

[10] ein beliebiges Ausführbaresprogramm mit setuid

2.4.8.2 Mail Spoofing

Die Funktionalität von SMTP-Servern ähnelt der von Paket-Routern, nur werden diese ausschließlich für E-Mails benutzt. So kann es vorkommen, dass eine E-Mail mehrere SMTP-Gateways passieren muss, bevor das Ziel erreicht wird. An jeder Zwischenstation werden die E-Mail beurteilt und weitergeschickt.
Für die Kommunikation mit dem SMTP-Server braucht man nicht unbedingt einen speziellen Mail-Client, sondern kann auch über eine Telnet-Sitzung an Port 25 direkt mit dem SMTP-Server kommunizieren. Das Problem ist, dass der SMTP-Server per Voreinstellung, jedem Benutzer großes Vertrauen entgegenbringt. D. h., ein Anwender kann jede beliebige Absenderadresse angeben und der SMTP-Server wird diese Mail mit der falschen Adresse verarbeiten. Diesen Umstand kann der Angreifer nutzen um E-Mails unter anderen Namen zu verschicken oder für E-Mail-Spams.

- **Angriff basiert auf :** K 3
- **Schadens-Kategorie :** S 9

2.4.8.3 Finger Flooding

Einige Implementierungen des finger-Dienstes erlauben es, rekursive Anfragen zu stellen, die dazu führen können, dass das System zu viele Betriebsmittel verbraucht und es zum Einstellen der Dienste kommt (denial-of-service).

- **Angriff basiert auf :** K 4
- **Schadens-Kategorie :** S 10

2.4.8.4 Verteilte, koordinierte Angriffe

Hierunter versteht man Angriffe, die z.B. mit Hilfe einer modifizierten Web-Seite den Telnet-Port eines bestimmten Rechners angreifen, d.h., jeder Benutzer, der auf diese Seite zugreift hilfte dem Angreifer den betreffenden Rechner zu attackieren. Finden genügend Zugriffe in kurzer Zeit auf diese Web-Seite statt, kann es zur Einstellung des Dienstes des betreffenden Rechners kommen.

- **Angriff basiert auf :** K 4
- **Schadens-Kategorie :** S 10

2.4.9 Angriffe auf das Authentisierungssystem

Authentisierungsverfahren sollen den unerlaubten Zugang, z.B. zu einem Netz unterbinden. Dieses ist notwendig, um eine Kontrolle über die Nutzung z.B. von Internet-Diensten für einen Nutzer oder Rechner effektiv durchführen zu können. Es gibt verschiedene Verfahren, mit deren Hilfe eine Authentisierung ermöglicht wird, eines der am häufigsten verwendeten Verfahren ist die Authentisierung über ein Passwort. Ist ein Passwort die einzige Authentisierungsmaßnahme um den Zugang zu einem System zu schützen, ist es erforderlich, dass das Passwort gewisse Kriterien erfüllt um genügend Schutz zu bieten. D. h., alle Anwendungen, bei denen im Zuge der online-Authentisierung Passwörter im Klartext übertragen, werden eröffnen eine große Sicherheitslücke. Ein erfolgreicher Angriff, bei dem der Passwort-Schutzmechanismus überwunden werden konnte, gewährt dem Angreifer Zutritt zum System. Wenn es sich dabei um eine privilegiertes Passwort handelt, dann hat der Angreifer uneingeschränkten Zutritt zum System.

2.4.9.1 Lauschangriff-Sniffing

Mit Lauschangriffen (Sniffing) bezeichnet man Angriffe, bei denen durch das Mitlesen der Daten, z.B. bei einer TCP/IP-Datenübertragung, Informationen gewonnen werden. Eine der Ursachen für diese Angriffe ist, dass die Nachrichten im Klartext Übertragen werden. Handelt es sich bei der übertragenen Nachricht um Passwörter, so kann es fatale Folgen für die Systemsicherheit haben. Durch Belauschen wird die Vertraulichkeit des Systems gefährdet und mit den gewonnen Informationen können weitere Gefahren für das System entstehen. Lauschangriffe sind passive Angriffe, die sich jedem Protokollierungsmechanismus entziehen können und somit unentdeckt bleiben.

- **Angriff basiert auf :** K 3
- **Schadens-Kategorie :** S 3

2.4.9.2 Erraten von Passwörtern

Das Ziel von Angreifern sind Rechnersysteme, die ihre Authentisierungsmechanismen über Passwörter schützen. Die Methoden, die zum Einsatz kommen um an die Passworte zu gelangen sind unterschiedlich. Folgende Möglichkeiten sind bekannt:

- Einfaches Erraten der Passwörter durch rekursives Ausprobieren.
- Social Engineering, das die Nachsicht der Anwender ausnutzt, z.B. werden Passwörter, unter Angabe falscher Identität (Administrator)erfragt, oder dem Anwender wird bei der Eingabe über die Schulter geschaut.

- Systematisches Erraten der Passwörter unter Verwendung der Passwortdatei, z.B. passwd. Mit dem gleichen Verschlüsslungsalgorithmus verschlüsselte Wortbibliotheken werden mit den Einträgen in der Passwortdatei verglichen.
- Modifizierte Programme, die im Zielsystem installiert sind, können die Passwörter mitlesen und z.B. per E-Mail an den Angreifer verschicken.

- **Angriff basiert auf :** K 6
- **Schadens-Kategorie :** S 3

2.4.10 Import von kompromittierendem Code

Weitere Gefahren für die Systemsicherheit sind sogenannte *"bösartige Code"*. Unter *"bösartige Code"* versteht man, unautorisierte Code, die Funktionen ausführen, die dem Benutzer nicht bekannt, bzw. von ihm nicht erwünscht sind. Aber auch ein reguläres Programm, welches durch die Einbindung eines solchen Codes verändert wurde, kann Funktionen ausführen, die dem Benutzer nicht bekannt sind. Ein *"bösartiger Code"* kann auch ein unautorisierter Code sein, der zu dem Zweck entwickelt wurde, sich selbst im Verborgen zu halten und/oder die Zerstörung von Daten zu bewirken. Der Schaden, den ein kompromittierender Code verursacht, ist abhängig von der Funktionalität des Codes.

2.4.10.1 Viren

Ein Virus bezeichnet ein Programmstück, das in ein gültiges Programm (Wirtsprogramm) eingebettet wird, und das darüber andere Programme infiziert. Viren können sich selbst replizieren und somit im System ausbreiten. Zu diesem Zweck verändert das Virus andere Programme so, dass diese eine Kopie des Virus enthalten. Die anschließende Ausführung des befallenen Programme bewirkt auch die Ausführung des Virus.

- **Angriff basiert auf :** K 1, K 6
- **Schadens-Kategorie:** S 7, S 8, S 4, S 10

2.4.10.2 Trojanische Pferde

Trojanische Pferde - auch Trojaner genannt - sind Programme, die neben ihren eigentlichen Funktionen, die dem Benutzer bekannt sind, auch weitere Funktionen enthalten, die dem Benutzer verborgen bleiben und somit auch meistens nicht bemerkt werden. Diese Programme, die von „böswilligen Programmierern" verändert werden indem zusätzliche Code in das Programm eingebunden werden, können verborgene und unautorisierte Funktionen ausführen. Ein Beispiel hierfür ist, dass ein Angreifer die *login*-Prozedur durch eine neue, modifizierte *Login*-Prozedur ersetzt und somit dem Benutzer eine reguläre login- und Passworteingabe vortäuscht, d. h., dieser verhält sich genauso wie die orginale Prozedur, aber insgeheim werden die Passwörter in einer Datei gespeichert und zum späteren Missbrauch an einen Angreifer weitergeleitet.

- **Angriff basiert auf :** K 6, K 7
- **Schadens-Kategorie :** S 3

2.4.10.3 Würmer

Würmer sind selbstständige, selbstreproduzierende Programme, die sich in einem System, aber vor allem in Netzen, ausbreiten (vervielfältigen). Ein Angriff eines Wurms läuft wie folgt ab: Im ersten Schritt wird eine Systemschwäche, z.B. *sendmail-debug*, ausgenutzt, um ein Stück Programmcode ausführen zu lassen, welches dann z.B. die Übertragung des gesamten Wurmprogrammes bewirkt. Abhängig von dem Betriebssystem auf dem der Mail-Client läuft und von den Berechtigungen des Anwenders unter dem der verwundbare Mail-Client läuft, kann der Angreifer das gesamte System zum Absturz bringen oder sogar administrativen Zugriff auf das System bekommen, wenn ein privilegierter Anwender die Mail liest. Ist der Angriff auf einen Rechner erfolgreich gewesen, so kann dieser als Ausgangsbasis für weitere Angriffe auf andere angeschlossene Systeme ausgenutzt werden. Der erste Wurm ist 1988 in der UNIX-Welt aufgetaucht. Der Student R.T. Mooris hatte Fehler im Berkeley UNIX entdeckt und ausgenutzt, die ihm den nicht-autorisierten Zugriff zu Maschinen im gesamten Internet ermöglichten.

- **Angriff basiert auf :** K 5
- **Schadens-Kategorie :** S 3, S 7, S 8, S 10

2.4.11 Probleme im Zusammmenhang mit dem WWW

Das World Wide Web wird oftmals als Synonym für das Internet benutzt. Einer der Gründe dafür ist, dass WWW-Technologien zur Verbreitung der Internet-Nutzung sehr viel beigetragen haben. Mit Hilfe dieser Technologien sind auch die Anwendungsmöglichkeiten bei der Nutzung des Internets erheblich gestiegen. In die ursprünglichen, statischen HTML-Anwendungen wurden immer mehr dynamische, interaktive Programme integriert. Auch in Unternehmensnetzen (Intranets) vermehrte sich die Benutzung der WWW-Technologien, weil diese Netze meist an das Internet angeschlossen wurden.
Beim Design der WWW-Technologien standen Sicherheitsaspekte nicht im Vordergrund. Daher wurden nicht ausreichende Maßnahmen zur Verhinderung von

missbräuchlicher Nutzung berücksichtigt. Dieser Aspekt führte zu einer großen Anzahl von Sicherheitsrisiken, wie z.B. bei der Nutzung von Web-Browsern. Mit Hilfe von Web-Browsern kann man aus dem Internet Programme herunterladen und diese auf dem lokalen Rechner ablaufen lassen. Darüber hinaus kann auch das Laden von Webseiten Gefahren mit sich bringen, weil diese Seiten bösartige Inhalte in Form von Programmen (Java, JavaScript) enthalten können. Im Folgenden werden Angriffe beschrieben, die die Risiken der modernen Web-Technologien aufzeigen.

2.4.11.1 Java

Die Programmiersprache Java ist in den 80igern von der Firma Sun Microsystems als objektorientierte Programmiersprache entwickelt worden. Sie war ursprünglich einmal gedacht zur Programmierung von Geräten wie Videorekordern, Waschmaschinen usw.. Bei der Vielzahl von möglichen Zielsystemen war die **Plattformunabhängigkeit** ein wichtiges Designziel, das bei der Entwicklung von Java verfolgt wurde. Es folgt eine Liste der wichtigsten Ziele, die mit Java erreicht werden sollten:

Einfach: Der Befehlsumfang sollte so klein wie möglich gehalten werden. Gleichzeitig sollte auch die Syntax einfach sein, so dass ein Anfänger schnell Erfolge bei der Erstellung eigener Programme hat.

Objektorientiert: Objekte stehen bei der Programmentwicklung unter Java im Mittelpunkt der Betrachtung. Unter einem Objekt wird unter Java Folgendes verstanden:

Definition 2 (Objekt) *Ein* **Objekt** *ist ein Softwarebündel bestehend aus Variablen und dazu in Beziehung stehenden Methoden (Funktionen und Prozeduren)*

Es gibt verschiedene Arten von Objekten. Diese Arten werden Klassen genannt. Eine Klasse kann zu einer neuen Klasse erweitert werden. Diesen Mechanismus nennt man **Vererbung**. Dabei erbt die neue Klasse alle Merkmale der Mutterklasse.

Definition 3 (Klasse) *Eine Klasse ist ein Prototyp, der alle Variablen und Methoden definiert, die allen Objekten dieser Art gemeinsam sind.*

Die konkreten Datenobjekte, bzw. deren Repräsentation im Speicher der Systeme werden **Instanzen** genannt.

Architekturneutral: Anstelle des Begriffes der Plattformunabhängigkeit wird lieber gesagt, dass Java architekturneutral sei. Denn tatsächlich benötigen alle Java-Programme als Umgebung die Java-Plattform. Da diese Plattform aber auf vielen unterschiedlichen Systemen mit unterschiedlichen Architekturen verfügbar ist, wird Java das Prädika "architekturneutral" gegeben.

Verteilt: Java bietet die Möglichkeit, über die Schnittstelle "Remote Method Invocation Interface" entfernte Prozeduren (remote procedure calls) auf aufzurufen

Multithreaded: Java bietet die Möglichkeit, Programme mit verschiedenen, nebenläufigen Handlungssträngen zu erstellen.

Die Javaplattform. Ein Javaprogramm wird als eine für Menschen lesbare Textdatei eingegeben. Diese Datei ist auch für die Maschine lesbar, aber der Aufwand, den die Maschine betreiben muss, um die hochsprachlichen Java-Instruktionen in Maschinenbefehle umzusetzen, ist beträchtlich. Die Übersetzung der Java-Instruktionen durch einen Compiler in Maschinensprache zerstört die Architekturunabhängigkeit. Um dieses Dilemma zu lösen, wurde das Konzept der virtuellen Maschinen für die Realisierung von Java benutzt. Dies sind Programme, die sich wie ein Rechner mit Betriebssystem verhalten. Diese virtuellen Maschinen besitzen nun eine Maschinensprache, die sich leicht in die Maschineninstruktionen beliebiger Rechnerarchitekturen umsetzen lässt. D. h., die Interpretation des virtuellen Maschinencodes kann wesentlich schneller geschehen, als die Interpretation des Hochsprachencodes. Der virtuelle Maschinencode wird als **bytecode** oder **picode** bezeichnet.

Die Benutzung von bytecode war schon vor der Entwicklung von Java eine gängige Praxis. Beispielsweise übersetzen viele Lispinterpreter (z. B. GNU-Common-Lisp) den Hochsprachencode in einen Bytecode. Auch Scriptsprachen wie Python verwenden Bytecode.

Die Java-Plattform besteht aus einer **J**ava-**V**irtual-**M**achine (JVM) und einer Menge von Programmbibliotheken, die in objektorientierten Programmierumgebungen Klassenbibliotheken heißen. Diese bilden die Java-Laufzeitumgebung (**Java Runtime Environment** (JRE)). Die Art der Klassen, die in den Klassenbibliotheken einer normalen Java-Laufzeitumgebung vorhanden sind, ist ein fester Standard.

Die JRE kann in den verschiedensten Anwendungen implementiert sein. Bekanntestes Beispiel stellen die Java fähigen http-Clients dar. In ihnen ist eine JVM implementiert, die sich den Bytecode der Javaprogramme aus dem Netz holt und dann ausführt. Dies ermöglicht die Einbindung von Java-Programmen in HTML-Dokumente. Diese Art von in HTML-Dokumenten eingebetteten Programmen werden **Java Applets** genannt. Applets unterscheiden sich von "normalen" Java-

Anwendungen dadurch, dass ihnen bei dem Zugriff auf Systemresourcen Grenzen gesetzt sind. So darf ein Applet nicht auf die Festplatte des Systems, auf dem es gerade ausgeführt wird, zugreifen (lesen, schreiben). Es darf auch nicht direkt Systemfunktionen, wie `exit()` oder `fork()` ausführen.

Der Java Sandkasten. Die Möglichkeit, Anwendungen in ihrem Zugriff auf Systemressourcen zu beschränken, besteht darin, sie in einer virtuellen Laufzeitumgebung ablaufen zu lassen, die die Eigenschaften und Möglichkeiten des echten Systems überdeckt, bzw. versteckt. Diese Methode wird auch **sandboxing** genannt. Man sagt auch, die Programme laufen in einer einem Sandkasten ähnlichen Umgebung ab, in der sie keinen Schaden anrichten können. Die JRE kann auch so einen Sandkasten darstellen, wenn sie die Eigenschaften des zugrundeliegenden Systems in geeigneter Weise versteckt.

Einige mögliche Gefahren. Eine Gefahr, die von Applets ausgeht ist, dass sie einen Code enthalten, der die Java-Sandbox austrickst und dem Applet den Zugriff auf beliebige Systemressourcen ermöglicht. Ist dies einmal gelungen, so ist es denkbar, dass das Applet nun Viren, die in dem Bytecode der JVM geschrieben sind, in alle vorgefundenen Javaklassen einbringt.

In Java selbst Viren zu schreiben ist sehr schwer. Aber der einfacherere Bytecode der JVM ermöglicht das Schreiben von sich selbst replizierenden Programmen/Klassen, die sich in andere Wirtsklassen einnisten. Ein solches Virus kann auf jedem System, jeder Architektur, auf der die Java-Plattform implementiert wurde, ablaufen.

Eine andere Gefahr, die von bösartigen Applets ausgehen kann, ist eine Form einer denial-of-service Attacke. Ein Applet kann z. B. auf dem Zielsystem ein den gesamten Bildschirm überdeckendes Fenster erscheinen lassen. Dieses Fenster kann nun so sein, dass es alle Eingaben des Benutzers, also alle Tastatur- und Mausereignisse ignoriert. Der Anwender kann so vor dem Rest seines Systems ausgesperrt werden.

- **Angriff basiert auf :** K 2, K 5
- **Schadens-Kategorie :** S 3

2.4.11.2 Cookies

Cookies beinhalten Informationen über den Zustand einer HTTP-Verbindung zwischen einem Server und einem Client. Diese werden als Paar von Variablen und Werten in der Cookie-Datei gespeichert. Die Cookie-Datei dient dem Web-Browser im Wesentlichen zur Informationsspeicherung über die besuchten Web-

Server sowie über den Browser selber. D. h., wird eine Verbindung zu einer Web-Seite aufgebaut, können die Informationsanbieter die Daten ihres Servers in die Cookie-Datei des Benutzers eintragen, um später überprüfen zu können, ob dieser Benutzer vorher schon diese Web-Seite besucht hat. Die Cookie-Datei kann nur die Daten speichern, die der Server bei einer Anfrage vom Client (Browser) bekommt, diese können Informationen über IP-,TCP-, und HTTP-Header sein und die URL für die Daten, die abgefragt wurden. Cookies sind keine ausführbaren Programme und können daher nicht für die Verbreitung von Viren (bösartige Code) ausgenutzt werden, aber sie ermöglichen Profilbildungen über das Surfverhalten des Benutzers.

- **Angriff basiert auf :** K 7
- **Schadens-Kategorie :** S 1

2.4.11.3 CGI

Das **Common Gateway Interface** (CGI) ist ein Standard und spezifiziert die Schnittstelle für die Kommunikation zwischen einem HTTP-Server-Prozess und einer externen Applikation. Der Einsatz von CGI-Programmen ermöglicht dynamische Inhalte innerhalb von HTML-Seiten zu erzeugen. Auf ein CGI-Programm eines WWW-Servers wird, wie auf alle anderen WWW-Objekte, über die URL-Hyperlinks zugegriffen. Weist eine URL auf ein CGI-Programm, so wird vom betreffenden Server das Programm gestartet und läuft auf dem Server selber, nur das Ergebnis wird an den Client übertragen. Dies bringt auch Gefahren mit sich. Sollte es einem Angreifer gelingen, Zugriff auf eine CGI-Skript zu erlangen, so kann er das Skript modifizieren und mit der Aktivierung der betreffenden URL beliebige Befehle zum Ausführen bringen. Ein Grund dafür ist die unzureichende Prüfung der Benutzereingaben, die anschließend weiterverarbeitet werden sollen. So kann die Eingabe von Zeichenfolgen wie: *Eingabe; UNIX-Befehl* dazu führen, dass die Eingabefolge verarbeitet wird und danach kann der eingegebene Befehl zu Ausführung gebracht wird.

- **Angriff basiert auf :** K 5
- **Schadens-Kategorie :** S 3

2.4.11.4 MIME

Bei **MIME** handelt es sich um einen Standard für die Übertragung von mehrteiligen, multimedialen und binären Daten im Internet. Hierzu zählen Grafiken, Audiodaten, Dateien von Textverarbeitungsprogrammen, Programmdateien und einfache Texte, bei denen es wichtig ist, dass ihr Format bei der Übertragung nicht verändert wird. Daten, die im MIME-Format vorliegen, enthalten Zeilen im

Header der Nachricht, die unter anderem die Angaben über den *"content-type"*, d. h., das Format der zu Übertragenen Daten angeben. Einer der *"content-type"* ist der Message-Type (Subtype)" *external-body*", dieser ermöglicht statt eines Textes einen Verweis zu diesem zu übertragen, wobei verschiedene Methoden zum Zugriff auf diese Daten angegeben werden können (z.B. FTP, TFTP oder ANIN-FTP). D. h., eine MIME-Mail mit Message-Type "external-body" zeigt dem Mail-Client an , dass der Nachrichteninhalt nicht Bestandteil der E-Mail ist, sondern erst bei dessen Aufruf, mit Hilfe der unter *"access type"* spezifizierten Methode, geladen wird. Unter Sicherheitsgesichtspunkten birgt MIME-Nachrichten aber auch Gefahren, da diese Befehle enthalten, die bei der Interpretation durch MIME-fähige Mail-Clients unmittelbar zur Ausführung kommen, so kann die automatische Ausführung von MIME-codierten E-Mail zum Überschreiben von Systemdateien benutzt werden. Ein Beispiel [vHK98]:

- Content-Type:Message/External-body ;
- name = ".rhost" ;
- site = "ftp.visigoth.org" ;
- access = "anin-ftp" ;
- directory = "." ;
- Content-Type: text/plain

Diese Nachricht bewirkt einen anonymen Filetransfer der Datei ".rhost" vom Server "ftp.visigoth.org" und kopiert diesen in das aktuelle Verzeichnis des Empfängers der E-Mail. D. h., ist das Zielsystem nicht korrekt konfiguriert, ist es für einen Angreifer, unter Verwendung der oben beschriebenen MIME-Optionen, möglich, Dateien wie .rhost in das Empfängersystem einzuschleusen und über einen rlogin-Versuch in das System einzudringen.

- **Angriff basiert auf :** K 5
- **Schadens-Kategorie :** S 3

In diesem Beispiel stellt der "r-Dienst" das eigentliche Sicherheitsproblem dar.

2.4.11.5 WWW Spoofing

Unter WWW-Spoofing versteht man einen Vorgang beim dem einen Anwender auf modifizierte Web-Adressen zugreift ohne dieses zu bemerken, diese Seiten werden nicht von dem gewünschten Server im Internet geliefert, sondern von dem Server eines Angreifers. Zu diesem Zweck muss der Angreifer dafür sorgen, dass die Anfragen an einen bestimmten Server, dessen Inhalt vorgetäuscht werden soll, an ihm gerichtet werden. Hierfür stehen dem Angreifer die Möglichkeiten des DNS-Spoofing oder die Manipulation von Suchmaschinen, die auf eine Anfragen nach einem Stichpunkt die IP-Adresse des Servers von dem Angreifers liefern, zur Verfügung. Dieser Angriff fällt in den Bereich des sogenannten „man in the middle"-Angriffs, der die Integrität des betreffenden Web-Servers gefährdet. Dem Anwender ist die Möglichkeit gegeben, diese Manipulation leicht zu entdecken, indem er die Statusanzeige seines Browsers überprüft.

- **Angriff basiert auf :** K 4
- **Schadens-Kategorie :** S 5, S 6, S 9,

Angriffsarten	**K1-K7**	**S1-S10**
Aktiver Port Scan	$K7$	$S1$
Half Open Scan	$K7$	$S1$
Stealth Scanning	$K3$	$S1$
Remote Stealth Scan	$K7$	$S1$
UDP Port Scans	$K7$	$S1$
Finger, rusers	$K4$	$S1$
Netstat	$K4$	$S1$
Rstatd,sysstat	$K4$	$S1$
Identd, Rwho	$K4$	$S1$
ARP Spoofing	$K3$	$S5, S6, S10$
IP-Fragmentierung	$K3$	$S2$
Falsche IP-Parameter	$K2$	$S10$
SYN mit unerreichbarer Quelladresse	$K3$	$S10$
ICMP Redirect	$K3$	$S5, S6$
ICMP Destination Unreachable	$K3$	$S10$
ICMP Echo Request (ping of death)	$K2$	$S10$
ICMP Tunnel	$K1, K4$	$S3, S4, S7, S8, S10$
IP-Spoofing	$K3$	$S2$
TCP Sequenz-nummervorhersage	$K3$	$S3$
DNS Spoofing	$K4$	$S6, S9$
RIP Spoofing	$K3$	$S5, S6, S10$
Source Routing	$K3$	$S5, S6$
Session Hijacking	$K3$	$S3$
Flooding	$K1, K3, K4, K5$	$S10$
E-Mail-Flooding	$K7, K1$	$S10$

Tabelle 2.3: Angriffstypen

Angriffsarten	**K1-K7**	**S1-S10**
SYN-Flooding	$K3$	$S10$
System Log Flooding	$K7, K1$	$S10$
Data Flooding	$K7$	$S10$
Open/Close Flooding	$K7$	$S10$
DNS Cache Verunreinigung	$K4$	$S8, S9$
Reverse Lookup	$K1$	$S9$
NIS	$K4$	$S3$
FTP	$K4$	$S8, S9$
HTTP	$K2, K5$	$S8$
X11	$K1$	$S8$
NFS	$K2, K4, K5$	$S4, S7, S8$
Sendmail	$K2, K5$	$S3$
Mail Spoofing	$K3$	$S9$
Finger Flooding	$K4$	$S10$
Verteilte,koordinierte Angriffe	$K7, K1$	$S10$
Lauschangriffe	$K3$	$S3$
Erraten von Passwörtern	$K6, K7$	$S3$
Viren	$K1, K6$	$S4, S7, S8, S10$
Trojanische Pferde	$K6, K7$	$S3$
Würmer	$K5$	$S3, S7, S8, S10$
JAVA	$K2, K5$	$S3$
Cookies	$K7$	$S1$
CGI	$K5$	$S3$
MIME	$K5$	$S3$
WWW Spoofing	$K4$	$S5, S6, S9$

Tabelle 2.4: Angriffstypen

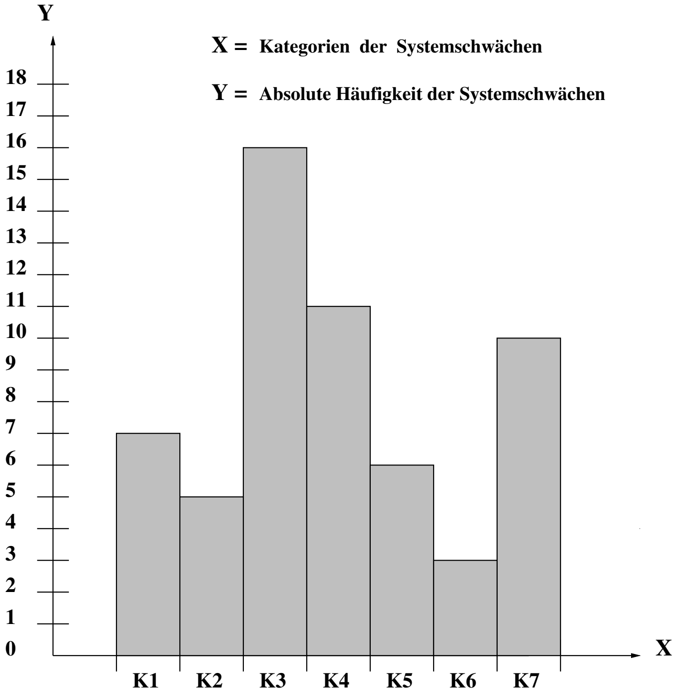

Abbildung 2.24: Balkendiagramm der Systemschwächen

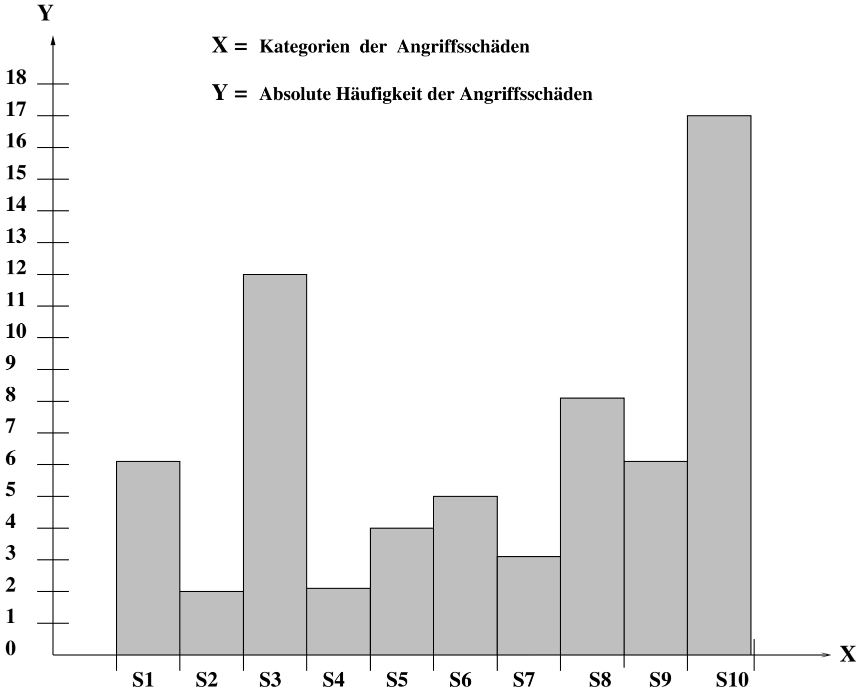

Abbildung 2.25: Balkendiagramm der Angriffsschäden

	K 1	K2	K3	K4	K5	K6	K7	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10
Aktive Port Scan							X	X									
Half Open Scan							X	X									
Stealth Scanning			X					X									
RPC Stealth Scan							X	X									
UDP Port Scan							X	X									
Finger, rusers, Netstat				X				X									
Rstatd, sysstat				X				X									
Identd, Rwho				X				X									X
ARP-Spoofing			X									X	X				X
IP-Fragmentierung			X						X								
Falsche IP-Parameter		X															X
SYN mit unerreichbarer SRC			X														X
ICMP Redirect			X									X	X				
ICMP Destination Unreachable			X														X
ICMP Echo Request		X															X

Abbildung 2.26: Angriffsarten

	K 1	K2	K3	K4	K5	K6	K7	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10
ICMP Tunnel	X			X						X	X			X	X		X
IP-Spoofing			X						X								
TCP- Sequenznummern-vorhersage			X							X							
DNS Spoofing				X									X			X	
RIP Spoofing			X									X	X				X
Source Routing			X									X	X				
Session Hijacking			X							X							
Flooding	X		X	X	X												X
Mail-Flooding	X						X										X
SYN-Flooding			X														X
System Log Flooding	X						X										X
Data-Flooding							X										X
Open/Close Flooding							X										X
DNS- Cache Verunreinigung				X											X	X	

Abbildung 2.27: Angriffsarten

	K 1	K2	K3	K4	K5	K6	K7	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10
Reverse Lookup	X															X	
NIS				X						X							
FTP				X											X	X	
HTTP		X			X										X		
X11	X														X		
NFS		X		X	X						X			X	X		
Sendmail		X			X					X							
Mail Spoofing			X													X	
Finger Flooding				X													X
Verteilter koordinierter Angriff				X													X
Lauschangriff			X							X							
Erraten von Passwort						X				X							
Viren	X					X					X			X	X		X
Trojanische Pferde						X				X							
Würmer					X					X				X	X		X

Abbildung 2.28: Angriffsarten

	K 1	K2	K3	K4	K5	K6	K7	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10
Java		X			X					X							
Cookies							X	X									
CGI					X					X							
MIME					X					X							
WWW- Spoofing				X								X	X			X	

Abbildung 2.29: Angriffsarten

Kapitel 3

WAS IST INTRUSION DETECTION

3.1 Entwicklung von Intrusion Detection Systemen

Eine der früh eingeführten Maßnahmen zur Gewährleistung der Sicherheit von Computersystemen waren **Zugriffsbeschränkungen**, die nur die Sicherheitsaspekte Vertraulichkeit und Integrität abdeckten. Außerdem wurde die Wirksamkeit dieser Sicherheitsmaßnahmen oft durch bestimmte Gegebenheiten eingeschrängt :

- menschliche Fehler bei der Umsetzung von Sicherheitsmaßnahmen,
- Schwierigkeiten bei der Findung eines Kompromisses zwischen widersprüchlichen Bedürfnissen (z.B. Abwägung von Sicherheit gegen Performanz, Sicherheit gegen Benutzerfreundlichkeit),
- unentdeckte Sicherheitslücken und der Missbrauch eingeräumter Rechte

Im Bereich der Rechnersicherheit begann man an neuen Ideen zu arbeiten, um effizientere Schutzmaßnahmen einzuführen, die weitgehend unberührt von den eben geschilderten Problemen blieben. Eine solche Sicherheitsmaßnahme war die Erzeugung und Analyse der Audit-Spuren von System- bzw. Benutzeraktivitäten. Diese Maßnahmen bilden die Funktionsgrundlage von ID-Systemen.

Die ersten ID-Systeme untersuchten lediglich die vom Betriebssystem erzeugten Protokolldaten im Hinblick auf mögliche Angriffe. im Allgemeinen wurden dabei die Benutzerverhaltenskontrollen im System durch das Verfolgen von Benutzeranfragen und deren Aufzeichnung in geeigneter Form (Protokollierung) in

Angriff genommen. Durch die Analyse dieser Aufzeichnungen sollte dann aufgedeckt werden, ob eine Menge von Benutzeranfragen als verdächtig angesehen werden konnte.

Audit-Kontrollen in „traditionellen Audit-Systemen" hatten aber den Nachteil sehr komplex und nicht rechtzeitig ausführbar zu sein. Die Komplexität rührte aus der Tatsache, dass die, während des normalen Systembetriebes anfallenden, Audit-Datenmengen sehr schnell groß und unhandlich wurden. Der manuelle Rückblick auf die langen Audit-Dateneinträge, die bewertet werden sollten, schränkte die Durchführbarkeit dieser Methode ein, was dazu führte, dass die Kontrollen nur in bestimmten Zeitabständen durchgeführt werden konnten. Folglich konnten mögliche Systemangriffe nicht rechtzeitig entdeckt werden oder sie wurden lange nach dem Ereignis festgestellt. Der Angreifer auch die Möglichkeit seine Spuren im System zu verwischen, indem er die Audit-Daten veränderte. Es gab zwei Möglichkeiten, mit solchen unhandlichen Datenmengen umzugehen [vHK98] :

- Verbesserung des Audit-Mechanismus, so dass nur relevante Handlungen protokolliert werden konnten.
- Bereitstellung von Vorsorgewerkzeugen und Systemen zur automatischen oder halb-automatischen Überprüfung von Audit-Daten, um Einbruchsversuche möglichst online, in Echtzeit, zu erkennen.

Werkzeuge zur automatischen Analyse von Audit-Daten wurden sowohl von Forschern im Bereich Rechnersicherheit, als auch von Systemadministratoren als propritäre Lösungen entworfen. Die Effektivität dieser Werkzeuge war eher gering und erfüllte nicht die Erwartungen. Anfangs lag der Schwerpunkt der Forschung bei der Herausfilterung nicht-relevanter Audit-Daten, um dem SSO die manuelle Analyse zu ermöglichen. Die Entwicklung der automatischen Werkzeuge wurde aber vorangetrieben. Aus diesen Überlegungen und Entwicklungsversuchen heraus entstand das Forschungsgebiet **Intrusion Detection**.

Ziel dieser Forschungen war es, die obigen Ansätze zu erweitern, indem man in den Audit-Daten nach bekannten Angriffsmustern und Evidenz für anomales Benutzerverhalten zu suchen begann. Dabei ergaben sich weitere Probleme :

- Der Mangel an verfügbaren Angriffsszenarien, die zum Vergleich herangezogen werden konnten
- Die große Zahl von Parametern, die möglicherweise mit den Audit-Daten assoziiert sein konnten.

Zur Lösung dieser Probleme wurden mit der Zeit Versuche unternommen Methoden anderer Bereiche, wie z.B. aus dem Gebiet der Künstlichen Intelligenz, Statistik und Informationstheorie, bei Einbruchsentdeckung einzusetzen. Diese

Methoden mussten aber an das Design und an die Bedienung der ID-Systeme angepasst werden.

Viele ID-Systeme wurden anfangs eher einfach gehalten, und der Erfolg eines Systems hing meistens entscheidend von der Wahl der richtigen Parameter ab. In dieser Hinsicht ähnelten bzw. ähneln viele ID-Systeme früheren Expertensystemen (wie z.B. MYCIN). Ein weiteres Problem war, dass ID-Systeme Eigenschaften benötigten, die nicht im direkten Zusammenhang mit der Entdeckung von Angriffen standen. Beispielsweise war es sehr wichtig, die Integrität der IDS-Konfigurationsdaten, oder auch die Echtheit einer Alarmmeldung zu garantieren, denn ein ID-System sollte sich nicht auf die Fähigkeit beschränken Angriffe zu erkennen.

In Anbetracht der obigen Entwicklungen kann man sagen, dass die, durch das Forschungsgebiet **Intrusion Detection** hevorgebrachten ID-Systeme, Programme sind, die folgende Arten der Analyse durchführen :

- Programme, die Benutzeraktivitäten mit Benutzerprofilen vergleichen (anomaly detection),
- Programme, die Benutzeraktivitäten mit bekannten Angriffsmethoden vergleichen (misuse detection),
- Programme, die Eigenschaften beider Typen besitzen.

3.1.1 Projekte zu Intrusion Detection bei SRI

Projekte zu Missbrauchs-Entdeckungs-Systemen wurden in verschiedenen Institutionen durchgeführt. Eine der führenden darunter war SRI (Standford Research Institute).

Bei SRI wurde mit verschiedenen Ansätzen experimentiert. Die durchgeführten Studien lieferten Informationen für weitere Entwicklungen. Eine der frühesten Studien über Intrusion Detection wurde von Jim Anderson durchgeführt, der nach Möglichkeiten für die Analyse der Audit-Daten von Computersystemen suchte. Seine Methoden benutzten Daten, die für andere Gründe (z.B. Leistungsanalyse) gesammelt und für die Stapelverarbeitung entworfen wurden. Es waren Audit-Daten, die an einem Tag anfielen und die am Ende des Tages analysiert wurden.

Nachfolgende Arbeiten konzentrierten sich auf die Entwicklung von Prozeduren und Algorithmen für die automatische offline Sicherheitsanalyse der Audit-Daten. Das Ziel war die Bereitstellung von automatischen Werkzeugen für den SSO, um ihn bei der Analyse/Bewertung der täglichen Systemaktivitäten zu unterstützen.

3.1.1.1 Studien zur Algorithmengewinnung

Zur Algorithmengewinnung wurden zwei wesentliche Projekte durchgeführt.
In dem ersten Projekt wurden existierende Audittrails benutzt um mögliche Ansätze zur Bildung von automatischen Werkzeugen für die Sicherheitsanalyse der Audittrails zu erforschen.

Ziel war die Entwicklung von analytischen Statistik-Techniken, um in den Systemaufzeichnungsdaten nach Benutzerverhalten zu suchen, das auf Einbrüche schließen lässt. Projektergebnis war die Entwicklung eines high-speed Algorithmus, der die Benutzer, auf Grund ihrer Verhaltensprofile, genau voneinander unterscheiden konnte.

Das zweite Projekt wurde von Theresa Lunt geleitet. Es wurden spezielle Sicherheits-Audittrails erzeugt und mögliche Ansätze für ihre automatische Analyse untersucht. Das Projekt lieferte den ersten experimentellen Hinweis dafür, dass Benutzer auf der Grundlage ihrer Systemnutzungsmuster voneinander zu unterscheiden sind und Benutzerverhaltens-Merkmale gefunden werden können, mit deren Hilfe zwischen normalem Benutzerverhalten und einer Anzahl von simulierten Einbrüchen unterschieden werden kann. Es stellte sich bei diesen Projekten heraus, dass es sehr problematisch war zwischen „normalem“ und „anormalem“ Benutzerverhalten zu unterscheiden. Zu Lösung des Problems machte man sich KI-Ansätze zu Nutze, zu denen auch die Expertensystem-Techniken zählen.

Studien mit Expertensystem Ansätzen Bei der Benutzung von Expertensystem-Techniken wurden folgende Schritte unternommen :

- Das Expertenwissen wurde in Form von Sicherheitsregeln (security rules) in einer Wissensdatenbank gespeichert.
- Die Regeln wurden meist in einer eigens für das jeweilige System entwickelten, formalen Sprache verfasst.
- Die Regeln aus der Wissensdatenbank wurden dazu benutzt, Auditdaten auf verdächtige Aktivitäten hin zu analysieren.

Alternativ dazu wurde mit einem **regelbasierte-Ansatz** experimentiert. Die Methode dabei war,

- die Codierung von Systemverwundbarkeiten und bekannten Angriffsszenarien in einer formalen Sprache,
- die Spezifikation dieser Sicherheitsschwachstellen und möglichen Angriffsszenarien in Form von “Produktion“ genannten Regeln,
- der Vergleich der Audit-Daten mit diesen Regeln.

Bei dem regelbasierte-Ansatz waren die Regeln unabhängig vom Verhalten der Benutzerund des Systems in der Vergangenheit (feste Regeln). Die von dem überwachten System abgeleiteten Audit-Daten wurden mit diesen Regeln verglichen, um verdächtiges Verhalten festzustellen. Ein Ereignis löste die Anwendung einer Regel aus, unabhängig davon, ob das Ereignis für den Benutzer als normal anzusehen war. Folglich konnten Einbruchsszenarien, die vielleicht in Bezug auf den Benutzer nicht anomal waren, von den geeigneten Regeln entdeckt werden. Die Einschränkung bei dem regelbasierten-Ansatz war, dass die Audit-Daten nach bekannten Angriffsmustern durchsucht wurden. Viele Einbrüche aber wurden gemäß unbekannten Einbruchsarten durchgeführt. Ein anderer Nachteil war, dass das Schreiben einerRegelbasis eine komplexe „Wissensorganisations“-Aufgabe (knowledge engineering) darstellte und die Aufrechterhaltung der Regelbasis ein ebenso schwieriges Unterfangen war (rule consistency maintenance, rule ranking, facts deletion).

Modellbasierter Ansatz : Ein weiterer Ansatz, der sich weitgehend von den Expertensystem Ansätzen unterschied, war das Benutzen von **modellbasierter Schlussfolgerung** (model-based reasoning). Dieser Ansatz basierte auf der Annahme, dass Angreifer typische Verfahren benutzen, um ein System anzugreifen, wie bei systematischen Password-Attacken oder Zugriffsversuchen auf privilegierte Dateien. Diese Methode entwickelte spezielle Modelle von vordefinierten, bekannten Angriffen. Das System suchte nach Einbrechern, indem es nach Aktionen „Ausschau hielt“, die zu den hypothetischen Angriffsszenarien passten. Die Szenarien variierten dabei in Abhängigkeit von dem Typ des Angriffs und des Systems. Dieser Ansatz wurde in den Forschungsstudien der SRI [TT91] ausgewertet. Eine Forderung, die dabei aufgestellt wurde, war, dass nach einem gegebenen Angriffs-Szenario das System in der Lage sein sollte, zu ermitteln, welche Aktionen ein Eindringling als nächstes ausführen würde. Es stellte sich heraus, dass es nützlich war Audit-Daten auszuwählen, die zusammengefasst und beobachtet werden konnten, um die nächste Stufe eines Einbruchsversuches voraussagen und die Hypothese eines möglichen Einbruches bewerten zu können. Ein Vorteil von modellbasierter Schlussfolgerung für Intrusion Detection war seine Selektivität bei den Audit-Daten: Die relevanten Daten wurden gebündelt und dadurch die Menge der zu prüfenden Daten reduziert. Weiterhin konnten vorbeugende Maßnahmen gegen Einbrüche unternommen werden, da das System in der Lage war Voraussagen zu treffen.
Genau wie bei Expertensystemen war auch dieser Ansatz beschränkt, weil hier nur nach bekannten Einbruchsszenarien gesucht wurde, viele Sicherheitslücken und Angriffe aber von unbekannter Art sein konnten. Die Forschung tendierte dazu modellbasierte Schlussfolgerung mit statistischer Anomalie-Entdeckung zu kombinieren. Andere Ansätze, die nicht auf „Verdächtigungen“ basierten, defi-

nierten „akzeptables Verhalten“ (acceptable behaviour).

Neuere Vorschläge (abgeleitet von den Studien beim SRI) nutzen das Potential von *neuronalen Netzwerken* aus, um auf Einbruchsattacken angemessen zu reagieren. Neuronale Netzwerke lösen das Problem von statistischen Methoden: sie benötigen nicht die Annahmen über die zugrundeliegende Verteilung von Benutzerverhalten Obwohl vielversprechend, ist dieser Ansatz nicht ausreichend genug entwickelt, um die statistischen Komponenten der erwähnten Prototypen zu ersetzen.

3.2 Charakterisierung von Intrusion Detection Systemen

Der historischen Entwicklung von ID-Systemen ist zu entnehmen, dass sich verschiedene Einbruchentdeckungsansätze herausgebildet haben.
Viele dieser Ansätze können miteinander kombiniert oder auch seperat benutzt werden. Bei den Nachforschungen zu dieser Diplomarbeit mussten wir leider immer wieder feststellen, dass keine eindeutige Klassifizierung der verschiedenen ID-Arten vorhanden war. Viele Quellen haben sich eingängig mit speziellen Intrusion Detection Methoden auseinander gesetzt, so dass ein Gesamtbild sehr schwer zu erstellen war. Wir haben, zum Zwecke eines besseren Überblicks, versucht eine Charakterisierung vorzunehmen, um dem Leser eine bessere Differenzierung zu ermöglichen. Die Charakterisierung der ID-Systeme kann basierend auf

1. **Datenquellen** (Data Source based IDS),
2. **Einbruchsmodellen** (Model based IDS)

erfolgen.

Die auf **Datenquellen** basierende Charakterisierung unterscheidet folgende ID-Systeme :

- *Hostbasierte ID-Systeme :* Zur Entdeckung von Einbrüchen werden die Audit-Daten von einem einzelnen Host herangezogen.
- *Multihostbasierte ID-Systeme :* Zur Entdeckung von Einbrüchen werden Audit-Daten von mehreren Hosts herangezogen.
- *Netzwerkbasierte ID-Systeme :* Zur Entdeckung von Einbrüchen werden Netzwerkverkehrs-Daten, zusammen mit den Audit-Daten, von einem oder mehrere Hosts herangezogen.

Bei der Charakterisierung von ID-Systemen, die auf **Einbruchsmodellen** basieren, werden allgemein zwei Modelltypen unterschieden :

- *Anomaly detection Models :* Das ID-System entdeckt Einbrüche durch das Suchen von Aktivitäten, die sich von dem normalen Benutzer- oder Systemverhaltensmuster unterscheiden.

- *Misuse detection Models :* Das ID-System entdeckt Einbrüche durch das Suchen nach Aktivitäten, die mit bekannten Einbruchstechniken oder Systemverwundbarkeiten übereinstimmen.

Anomaly detection wird in der Fachliteratur öfter auch als *behaviour-based intrusion detection* und Misuse detection als *knowledge-based intrusion detection* bezeichnet. Es soll noch einmal darauf hingewiesen werden, dass die oben vorgenommene Charakterisierung zwei verschiedene Betrachtungsweisen von ID-Systemen darstellt und nicht etwa zwei verschiedene Arten von ID-Systemen. Ein netzwerkbasiertes ID-System kann beispielsweise das Anomaly detection Modell oder auch beide Modelle gemeinsam implementieren. Durch diese Charakterisierung sollen die ID-Systeme lediglich danach unterschieden werden, woher sie ihre Audit-Daten bekommen und nach welchen Modellen sie diese bewerten.
Im nächsten Abschnitt wird eine detaillierte Beschreibung von einigen oben erwähnten Charakterisierungsmerkmalen vorgenommen.

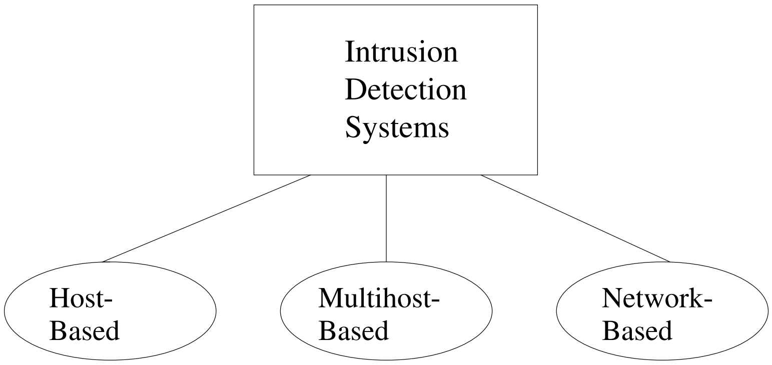

Abbildung 3.1: Characterization of Intrusion Detection Systems Based on Data Source

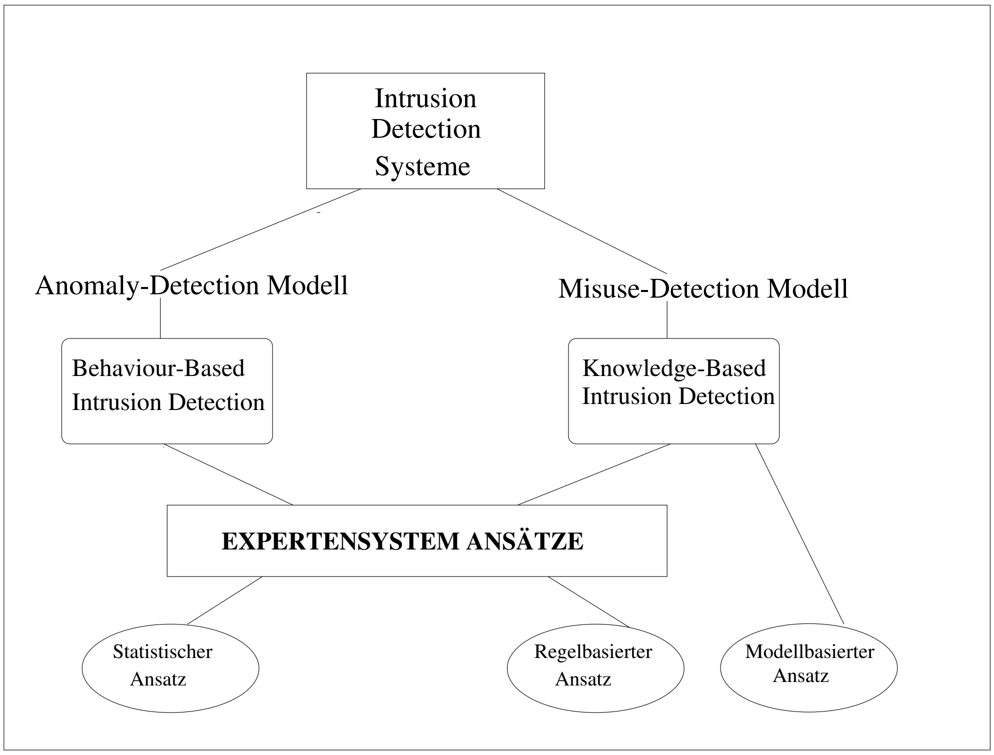

Abbildung 3.2: Characterization of Intrusion Detection Systems Based on Model of Intrusions

3.2.1 Network-Based Intrusion Detection (NIDS)

Netzwerkbasierte ID-Systeme überwachen den Verkehr auf dem ganzen Netzwerk-Segment, auf dem sie angesetzt sind. Sie überprüfen dabei den Netzwerkverkehr und entscheiden, ob er innerhalb annehmbarer Grenzen liegt. Die meisten gegenwärtigen Intrusion Detection Systeme sind netzwerkbasiert und werden auch als „first-generation-IDS" bezeichnet. Diese Systeme benutzen überwiegend eine passive Methode zur Datensammlung. Netzwerkbasierte ID-Systeme können gg als Neztwerksniffer auf IP-Ebene bezeichnet werden. Eine Netzwerk-Interface-Karte (*NIC, Network Interface Card*) kann in einem der zwei unten aufgeführten Modi arbeiten :

- *normalem Modus*, wo Pakete, die für einen Computer vorgesehen sind, bestimmt durch die Ethernet- oder MAC-Adresse (Media Access Control) des Paketes, an das Host-System durchgereicht werden.
- *wechselndem Modus*, wo alle Pakete, die auf dem Ethernet gesehen werden an das Host-System wiedergegeben werden

Eine Netzwerk-Interface-Karte kann normalerweise, durch das Benutzen von low-level Funktionen des Betriebssystems, zur direkten Kommunikation mit der Netzwerk-Karte, von *normalem Modus* zu *wechselndem Modus* (und umgekehrt) umgeschaltet werden. Netzwerkbasierte Intrusion Detection Systeme benötigen normalerweise eine Netzwerk-Interface-Karte in wechselndem Modus.

Im Follgenden sollen zwei Programme beschreiben werden, die in Zusammenhang mit netzwerkbasierter Intrusion Detection erwähnt werden müssen, *Paket Sniffer* und *Netzwerk Monitore.*

3.2.1.1 Paket Sniffer und Netzwerk Monitore

Paket Sniffer und Netzwerk Monitore wurden ursprünglich dazu entworfen, bei dem Überwachungsprozess des Ethernet-Netzwerkverkehrs Hilfe zu leisten. Die ersten zwei dieser Produkte sind *Novell LANalyser* und *Microsoft Netzwerk Monitor.* Diese Produkte erfassen grundsätzlich alle Pakete, die sie auf dem Netzwerk sehen. Wenn einmal die Pakete eingefangen werden, entstehen eine Menge von Möglichkeiten :

- Pakete können gezählt werden. Das Zählen der vorbeikommenden Pakete und das Zusammenaddieren ihrer gesamten Anzahl über eine Zeitperiode, liefern einen sehr guten Hinweis darüber, wie stark das Netzwerk belastet ist. Beide, LANalyser und Microsoft Netzwerk Monitor unterstützen Belastungs-Graphen oder Belastungs-Messer, um die relative Belastung des Netzwerkes anzuzeigen.

- Pakete können detailliert überprüft werden (es kann nötig sein, eine Menge von Paketen einzufangen, die einen Webserver erreichen, um einige Probleme mit dem Webserver zu untersuchen).

Paket Sniffing Produkte sind in den letzten Jahren immer fortschrittlicher geworden. Programme, wie *Ethereal* und neuere Versionen von *Netzwerk Monitor*, können den Inhalt von verschiedenen Paket-Typen zerlegen, um zu zeigen, welche Art von Kommunikation innerhalb dieser Pakete stattfindet.

Es muss aber bemerkt werden, dass alle Paket Sniffer, entgegen ihrer Einsatzbestimmung, zu Angriffszwecken missbraucht werden können. Paket Sniffing kann z.B. dazu benutzt werden, das Unix Passwort einer Person herauszufinden, indem Telnet-Pakete zu der Maschine, mit der diese Person in Verbindung steht, *erschnüffelt* werden. Wenn ein Angreifer das Netzwerk schon kompromittiert hat, kann einer seiner ersten Schritte, die er ausführt, darin bestehen, einen Paket Sniffer auf irgend einer Art und Weise zu installieren.

3.2.1.2 Paket Sniffing und wechselnder Modus

Alle Paket Sniffer erfordern eine Netzwerk-Interface-Karte in wechselndem Modus. Nur in wechselndem Modus wird jedes Paket, dass von dem NIC empfangen wird, an den Paket Sniffer selbst weitergereicht. Der Paket Sniffer benötigt normalerweise administrative Vorrechte auf der Maschine, auf der er läuft, damit die Hardware auf der Netzwerk-Karte manipuliert werden kann, um in den wechselnden Modus zu gelangen. Ein anderer Punkt, der berücksichtigt werden muss, ist das Benutzen von Schaltern (switches) in einem Netzwerk. Pakete, die von *einer* Schnittstelle des Schalters empfangen werden, werden nicht immer an andere Schnittstellen des Schalters gesendet. Aus diesem Grund wird eine mit sehr vielen Schaltern ausgestattete Umgebung das Benutzen von Paket Sniffern ablehnen.

3.2.1.3 Netzwerkbasierter Intrusion Detection als eine Weiterentwicklung des Paket Sniffers

Ein Paket Sniffer ist, vom Standpunkt der Sicherheit aus gesehen, nur begrenzt von Nutzen. Der Vorgang des Einfangens eines jeden Paketes aus dem Netz, und dessen manuelle Zerlegung, basierend auf dem Inhalt des Paketes, ist sehr zeitraubend. Netzwerkbasierte Intrusion Detection Systeme sind Software-Lösungen, die diese Aufgabe automatisch erfüllen. Zwei verfügbare Pakete, die diese Art von Intrusion Detection durchführen, sind *ISS RealSecure Engine* und *Network Flight Recorder*. Einige Beispiele für Intrusion Detection Methoden, die *RealSecure Engine* durchführen kann, sind folgende :

- Überprüfen von Paketen, die durch das Netzwerk laufen.

- Legitimierten Paketen das Durchlaufen erlauben (eventuell deren Aufzeichnung für zukünftige Analysen).
- Unterbrechung der Paketübertragung durch das Senden von TCP „connection closed“ oder ICMP „port unreachable“ Nachrichten zu dem Empfänger- und Sendersystem, wenn ein Paket die Sicherheit oder die Integrität des Zielsystems gefährdet.

Auf diese Weise kann *RealSecure* eine effektive, *zweite Ebene* Verteidigung für das Zielsystem ausführen, wenn es hinter einem Firewall platziert ist.Ein netzwerkbasiertes ID-System kann auch folgende Aufgaben ausführen :

- **Netzwerküberwachung zur Entdeckung von Port Scan's :**
  Bevor ein Cracker ein System kompromittiert wird er oft ein Port Scan des Systems durchführen, um festzustellen welche Verwundbarkeiten existieren könnten. Port Scan- Versuche, ausgehend von einem Host aus dem Internet, können ein Signal dafür sein, dass eine Person auf diesem Host beabsichtigt, das Netzwerk, bei dem er den Port Scan durchführt, zu beschädigen.
- **Netzwerküberwachung zur Entdeckung bekannte Angriffe :**
  Der Zugriff auf einem Webserver-Host, auf den Webserver-Port (80), kann als eine relativ harmlose Aktivität angesehen werden, aber einige Zugriffsversuche sind in der Tat absichtliche Angriffe oder Angriffsversuche. Ein Zugriff, der Form :
  GET/../../../etc/passwd HTTP/1.0,
  kann ein Angriffsversuch sein und sollte blockiert werden.
- **Identifizierung von IP Spoofing Versuchen auf verschiedene Arten :**
  Das ARP-Protokoll (Address Resolution Protocol), das benutzt wird um IP-Adressen zu MAC-Adressen (Media Access Control) zu konvertieren, ist oft ein Vorhaben für einen Angriff. Durch das Senden von gefälschten ARP Paketen über ein Ethernet, kann ein Einbrecher, der schon Zugriff auf ein System erlangt hat, vorgeben, er würde als ein anderes System operieren. Dies kann zu verschiedenen Arten von *Denial of service* Angriffen führen, so wie zu *System Hijacking* , wodurch ein wichtiger Server (wie der DNS-Server oder Authentication-Server) „reingelegt“ (spoofed) wäre. Cracker können diese Manipulation (spoofing) dazu benutzen, Pakete auf ihr eigenes System umzuleiten, und Angriffe vom Typ „man-in-the-middle“ auszuführen. Durch das Halten eines Registers von ARP-Paketen, kann ein netzwerkbasiertes ID-System die Herkunft (Ethernet-Adresse) eines kompromittierten Systems identifizieren und Cracker aufspüren.

Wenn unerwünschte Aktivitäten entdeckt werden, kann das netzwerkbasierte Intrusion Detection-System Maßnahmen ergreifen, einschließlich der Rekonfigurierung eines nahegelegenen Firewalls, um den gesamten Verkehr, der von dem

Computer oder Netzwerk des Einbrechers kommt, zu blockieren. Nach [Die00] weisen netzwerkbasierte ID-Systeme folgende Vor- und Nachteile auf :

==> **Vorteile :**

- Erkennung von IP-basierten Angriffen, wie z.B.
  - Angriffe gegen den Verbindungsaufbau
  - Denial-of-Service Angriffe mit fehlerhaften IP-Headern
- Keine negative Beeinflussung anderer Netzkomponenten

==> **Nachteile :**

- Frage des Standortes (die Frage nach dem Ort, wo das ID-System am besten platziert werden sollte)
- Schlechte bzw. schwierige Erkennung von Angriffen in
  - fragmentierten IP-Paketen
  - gekapselten Protokollen
- Schlechte Performanz bei Hochgeschwindigkeits-Netzen

3.2.2 Host-Based Intrusion Detection (HIDS)

Hostbasierte ID-Systeme laufen auf dem eigentlich beobachteten System und sind meist als eine dritte „Verteidigungslinie" hinter dem Firewall und dem Netzwerk-Monitor platziert. Das ID-System überprüft das System, um zu entscheiden oder festzulegen, ob die Aktivitäten, die beobachtet werden, akzeptabel sind.

Hostbasierte Intrusion Detection umfasst nicht nur das Überprüfen des Kommunikationsverkehrs inner- oder außerhalb eines einzelnen Computers, sondern auch die Integritätskontrolle der Systemdateien, einschließlich des „Ausschauhaltens" nach verdächtigen Prozessen. Hostbasierte Intrusion Detection tritt meist in verschiedenen Formen auf. Als zwei Haupttypen sind folgende gängig :

- *Netzwerk Monitore :* Diese überwachen hereinkommende Netzwerkverbindungen zu dem Host und versuchen zu bestimmen, ob irgendeine dieser Verbindungen eine Bedrohung darstellt. Netzwerkverbindungen, die einen Einbruchsversuch repräsentieren, lösen dann Gegenmaßnahmen aus. Es muss bemerkt werden, dass sich diese Art der Überwachung von netzwerkbasierter Intrusion Detection unterscheidet, weil hier nicht der gesamte Verkehr beobachtet wird, der das Netzwerk durchläuft, sondern nur der Teil, der den überwachten Host betrifft. Aus diesem Grunde werden auch keine dauernd wechselnden Betriebsmodi auf dem Netzwerk-Interface benötigt.

- *Host Monitore :* Diese überwachen Dateien, Dateisysteme, logs oder andere Teile des Hosts selbst, um nach besonderen Typen von verdächtigen Aktivitäten ausschau zu halten, die einen Einbruchsversuch repräsentieren könnten. Der Systemadministrator kann dann über jeden gefundenen Einbruchsverdacht informiert werden.

Im Folgenden wird genauer auf die Überwachung der Aktivitäten eingegangen, die durch die beiden oben erwähnten Monitortypen durchgeführt wird.

3.2.2.1 Überwachung hereinkommender Verbindungen

Es ist auf vielen Hosts möglich,, die Pakete, die Zugriffsversuche auf diesen Host ausführen, zu überwachen, bevor diese auf die *Netwerk-Schicht* gelangen. Dieser Mechanismus versucht, einen Host dadurch zu schützen, dass er Pakete , die für den Host bestimmt sind, abfängt, bevor sie irgendwelche Schäden anrichten können. Beispiele für einige Aktionen sind :

- Entdeckung von hereinkommenden, unautorisierten Verbindungsversuchen zu den TCP- oder UDP-Ports (z.B. Verbindungsversuche zu den Ports, hinter denen keine Dienste vorzufinden sind). Solche Aktivitäten weisen oft auf einen möglichen Cracker hin, der „herumstöbert", um einen Schwachpunkt ausfindig zu machen.

- Aufdeckung von hereinkommenden Port Scans, die Alarmierung einer Firewall oder die Modifizierung der localen IP-Konfiguration zur Vermeidung eines unerlaubten Zugriffsversuches von einem anderen Host.

3.2.2.2 Überwachung von Login-Aktivitäten

Trotz aller möglichen Anstrengungen des Netzwerkadministrators und der neuesten, eingesetzten und überprüften Intrusion Detection-Software, kann es passieren, dass ein Einbrecher „durchschlüpft" und dazu übergeht, sich in ein System einzuloggen, indem er einen, bis jetzt unbekannten, Angriffstyp ausführt. Ein Eindringling kann auch durch gewisse Mittel in den Besitz eines Netzwerk-Passwortes gekommen sein (z.B. durch Paket-Sniffing) und besitzt nun die Fähigkeit, sich von einem örtlich entfernten Rechner aus, in das System einzuloggen. Mit Hilfe einiger Software-Produkte (z.B. HostSentry) kann nach solchen ungewöhnlichen Aktivitäten im System gesucht werden. Diese Art Software überprüft Login- und Logoutversuche, und alarmiert den Systemadministrator bei Aktivitäten, die unüblich oder unerwartet ausgeführt werden.

3.2.2.3 Überwachung von Root-Aktivitäten

Die Absicht eines jeden Einbrechers ist es, Root- oder Administrator-Zugriffsrechte über das System zu erlangen, das er kompromittiert hat. Gut gewartete

und zuverlässige Systeme, die oft als Webserver und Datenbanken eingesetzt werden, weisen gewöhnlich wenige oder keine Super-User Aktivitäten auf, außer zu bestimmten Zeiten am Tag oder in der Nacht, wenn das System planmäßig gewartet wird. Cracker führen ihre Aktivitäten selten in Wartungszeiten aus und arbeiten oft zu unüblichen Zeiten am Tag. Es gibt nur eine Verteidigungsmöglichkeit, um diese Aktionen aufzudecken : Die Überwachung aller Aktivitäten durch den Root-User oder Systemadministrator.
Viele Unix-Systeme erlauben Login-Überwachung und auch andere Arten der Überwachung aller Aktivitäten durch den Root-User. Pakete, wie z.B. *Logcheck*, können dann diese Logs auf ungewöhnliche Aktivitäten hin *durchscannen* und den Systemadministrator über ihre Ergebnisse in Kenntnis setzen.
Administratoren von Open-Source Betriebssystemen haben als weitere Option die Möglichkeit der Veränderung des Kernels, um spezifische Aktivitätstypen zu protokollieren.

3.2.2.4 Überwachung des Dateisystems

Wenn einmal ein Einbrecher das System kompromittiert hat, wird er anfangen Dateien auf dem System zu verändern. Ein erfolgreicher Einbrecher könnte z.B. einen Paket-Sniffer oder einen Port Scan-Detektor installieren, einige System-Dateien oder Programme modifizieren, um gewisse Intrusion Detection Methoden zu unterbinden. Die Installation von Software in ein System bedeutet üblicherweise die Modifikation von einigen Teilen des Systems. Diese Modifikationen betreffen generell die Dateien oder Bibliotheken des Systems.
Programme wie *Tripwire, Fcheck* und *AIDE* sind dazu entworfen Dateiveränderungen auf dem System zu entdecken, und den Systemadministrator bei jeder Veränderung zu alarmieren. Folgende Methoden können dazu benutzt werden :

- Das Bilden von MD5 oder anderen kryptographischen Prüfsummen aller Systemdateien und deren Speicherung in einer Datenbank. Wenn eine Datei sich verändert, verändert sich auch ihre Prüfsumme.
- Das Speichern von Datum und Zeit der Anlegung oder Modifikation aller System-Dateien. Das Suchen nach Veränderungen in diesem Zeitstempel.
- Die Durchführung einer Aufzeichnung für jedes „run-as-root“ Programm auf dem System. Wenn jemand diese Aufzeichnungen verändert oder neue installiert oder löscht, kann man davon ausgehen, dass etwas Verdächtiges vorliegt.

Die Methoden, die bei Tripwire, Fcheck und AIDE eingesetzt werden, variieren geringfügig. Sie basieren aber alle auf den oben beschriebenen Mechanismen. Diese Programme stellen auch sicher, dass die Datenbank für die bekannten kryptographischen Prüfsummen nicht auf irgendeine Art und Weise selbst kompromittiert ist. Zum Beispiel könnte es für einen Eindringling, der Kenntnisse über das Betriebssystem und die Intrusion Detection-Software besitzt, möglich sein, die System-Dateien sowie die Datenbank für die kryptographischen Prüfsummen so zu modifizieren, dass es den Anschein erweckt als hätte sich nichts verändert.

3.2.2.5 Software bei hostbasierter Intrusion Detection

Um bei hostbasierter ID eine komplette Berichterstattung auf den eigenen Terminal zu bekommen, muss die ID-Software auf jeden Computer geladen werden. Es gibt zwei Hauptklassen von hostbasierter Intrusion detection Software :

- Host wrappers/personal firewalls
- Agent-based Software

Beide Lösungen sind wesentlich effektiver bei der Entdeckung von Angriffen durch „vertrauenswürdige-Insider“ (sogenannte anomale Aktivität), als die netzwerkbasierte Intrusion Detection, und sie sind relativ effektiv bei der Entdeckung von Angriffen von außen.
Host wrapper oder *personal firewalls* können so konfiguriert werden, dass alle Netzwerkpakete, Verbindungs- oder Login-Versuche zu der beobachteten Maschine überprüft werden können. Dies kann auch dial-in oder andere, nichtnetzwerkbezogene, Kommunikationsports einschließen.
Die bekanntesten Beispiele für Wrapper-Pakete sind *TCPWrappers* für UNIX und *Nuke Nabber* für WINDOWS. Personal Firewalls können auch solche Software auf dem Host entdecken, die versucht, eine Verbindung zum Netzwerk herzustellen (z.B WRQ‘s AtGuard).

Hostbasierte Agenten sind in der Lage Zugriffe, Veränderungen in kritischen Systemdateien und Benutzerprivilegien zu überwachen. Allgemein bekannte kommerzielle Versionen beinhalten Produkte von AXENT, CyberSafe, ISS und Tripwire. Darüber hinaus bietet auch UNIX eine große Menge an Software-Werkzeugen, um Einbruchsentdeckung vorzunehmen. Die Software kann an jeden einzelnen Computer, der überwacht werden soll, angepasst werden. Falls ein Computer nur eine kleine Anzahl von Benutzern besitzt, brauchen nur die Verbindungen von außen und die Integrität der Systemdateien überwacht zu werden. Im Gegensatz dazu braucht ein Rechner mit einer größeren Anzahl von Benutzern, und/oder mit Netzwerkverkehr, eine strengere Überwachung. Software-Typen, die zur Überwachung von Hosts dienen, umfassen :

- System- und Benutzer log-files (syslog);
- Verbindungs-Überwachung (TCPWrappers, lastlog);
- Prozess-Überwachung (lsof, process accounting);

- Speicher-Nutzungs-Überwachung (quotas);
- Sitzungs-Überwachung ;
- System-Kontrolle (Audit).

UNIX hostbasierte Einbruchsentdeckung ist nur so gut, wie das durchgeführte *logging*. Es können Programme geschrieben werden um log-files zu analysieren, die den Systemadministrator via e-mail oder pager, im Falle eines Verdachts, alarmieren. Logging-outputs können an einen entfernten Rechner gesendet oder modifiziert werden. Durch die Aufbewahrung der log-files an unüblichen Plätzen, werden die Eindringlinge daran gehindert, ihre Spuren zu verdecken. Wegen der Häufigkeit von Hacker-Scripten, kann diese „hausgemachte" Überwachung, zum Zwecke des Ausschau haltens nach spezifischen Instanzen von *break-ins*, umgerüstet werden.

Hostbasierte ID-Systeme werden auch als *Second-Generation ID-Systeme* bezeichnet. Immer mehr Nutzer, wie z.B. Firmen, bevorzugen hostbasierte ID-Systeme. hostbasierte ID-Systeme können weit aus mehr effiziente Intrusion Detection Techniken einsetzen (z.B. heuristische Regeln und Analysen) als netzwerkbasierte ID-Systeme. In Abhängigkeit von der technischen Ausgereiftheit der Sensoren, können diese Systeme auch „lernen" und Benutzerprofile als Teil ihrer Verhaltensdatenbank erstellen. Die Darstellung des normalen Verhaltens im Netzwerk kann über eine Beobachtungs-Zeitperiode realisiert werden.

In [Die00] sind folgende Vor- und Nachteile der hostbasierten ID-Systeme aufgeführt :

==> **Vorteile :**

- Analyse möglich für
  - Dienste, Benutzer, Objekte
  - Gekapselte Protokolle
  - Switched-Networks
- Geringe Last pro Client

==> **Nachteile :**

- Nachteile bei der Erkennung von IP-basierten Angriffen (DoS)
- Unterwanderung des IDS durch Betriebssystem Schwächen (Bugs)
- Hoher administrativer Aufwand
- Plattform-spezifische Software

Ein effizientes Intrusion Detection System wird aber beide, host- und netzwerkbasierte, ID-Typen implementieren.

3.2.3 Anomaly Detection (Behaviour-Based ID)

Verhaltensbasierte ID-Techniken gehen von der Annahme aus, dass ein Einbruch durch das Beobachten einer Abweichung vom normalen oder erwarteten Verhalten des Systems oder der Benutzer, entdeckt werden kann. Das Modell von normalem oder gültigem Verhalten wird abgeleitet von Referenz-Informationen, die durch verschiedene Hilfsmittel gesammelt werden. Das Intrusion Detection-System vergleicht später dieses Modell (z.B. mittels statistischer Methoden) mit der aktuellen Aktivität. Ein Einbruch in das System wird hier an seinen Auswirkungen auf die Parameter des Systemverhaltens, wie Auslastung, Netzwerkverkehr und Dateizugriffe, erkannt. Bei einer signifikanten Abweichung wird ein Alarm generiert. Mit *„signifikant"* wird eine Schwelle bezeichnet, die von einem bestimmten Modell oder von dem Sicherheitsbeauftragten selbst, gesetzt wird und die bei einem Angriff überschritten wird.
Bei verhaltensbasierter Intrusion Detection wird jede Aktivität, die nicht dem zuvor gelernten Verhalten entspricht, als „aufdringlich" gewertet. Dies lässt das Intrusion Detection System als „vollendet" erscheinen (im Allgemeinen können alle Angriffe abgefangen werden). Die Genauigkeit des Systems stellt aber ein schwieriges Problem dar, weil sehr viele Falsch-Alarme ausgelöst werden können (*vgl. 4.1 Allgemeine Anforderungen*). Diese Modelle werden in der Fachliteratur oft auch als „Paranoider Ansatz" bezeichnet, weil alles, was zuvor nicht schon mal gesehen wurde, als gefährlich eingestuft wird.

Vorteile des verhaltensbasierten Ansatzes sind,

- unvorhergesehene Verwundbarkeiten können leichter entdeckt werden
- unbekannte Angriffe sind leichter abzuwehren bzw. zu entdecken
- Angriffe, die durch den Missbrauch von Privilegien durchgeführt werden, können entdeckt werden (diese werden normalerweise nicht in die Auswertung von Sicherheitslücken einbezogen)
- weitgehende Unabhängigkeit von betriebssystemspezifischen Mechanismen

Nachteil des verhaltensbasierten Ansatzes sind die hohen Falsch-Alarm-Raten. Die Ursachen dafür können folgendermaßen aufgelistet werden :

- Während der Lernphase des ID-Systems kann nicht der ganze Verhaltensumfang eines Informationssystems abgedeckt werden.
- Das Verhalten der Benutzer oder des Systems kann sich mit der Zeit, verändern aber die Einführung einer periodischen online „Umgestaltung" des Verhaltens-Profils, würde weitere Probleme verursachen :
  - das ID-System wäre während der Lernphase nicht verfügbar,

  - bei einem während der Lernphase durchgeführten Angriff könnte aufdringliches Verhalten in das Verhaltens-Profil eingebaut werden.

In solchen Fällen würden zusätzliche Falsch-Alarme generiert werden.

3.2.4 Misuse Detection (Knowledge-Based ID)

Wissensbasierte Intrusion Detection-Techniken verwenden das Wissen, das über spezifische Angriffe und Systemverwundbarkeiten gespeichert bzw. angesammelt wurde. Das ID-System enthält Informationen über diese Verwundbarkeiten, sucht nach den ihm bekannten Angriffen und macht sich dabei das Wissen über die Verwundbarkeiten zu Nutze. Dieser Ansatz unterstützt den Vergleich zwischen Parametern der Benutzersitzungen (*user's session*) und bekannten Angriffstechniken, die von Einbrechern eingesetzt werden, um in ein System einzudringen. Wenn ein solcher Angriff entdeckt wird, wird ein Alarm ausgelöst.
In diesem Modell wird jede Aktion, die nicht explizit als Angriff wiedererkannt wird, als akzeptabel betrachtet. Die Genauigkeit von wissensbasierten ID-Systemen istrelativ gut. Dennoch ist ihre Vollständigkeit (d.h. die Fähigkeit, alle möglichen Angriffe entdecken zu können) von der regelmäßigen Aktualisierung der Wissensdatenbank über Angriffsszenarien abhängig.

Vorteile der wissensbasierten Modelle sind :

- Niedrige Falsch-Alarm-Raten
- Die detaillierte, kontextuelle Analyse, die durch das ID-System vorgeschlagen wird, und die damit verbundene leichte Bedienung des ID-Systems für den SSO, zur Durchführung von präventiven oder fehlerbehebenden Maßnahmen.

Die Nachteil dieses Ansatzes sind :

- Die Schwierigkeit der Erfassung von erforderlichen Informationen über bekannte Angriffe und die Aktualisierung der Wissensbasis mit neuen Verwundbarkeiten
- Die zeitraubende Verwaltung, bzw. Wartung der Wissensbasis des ID-Systems (jede Verwundbarkeit muss einer sorgfältigen Analyse unterzogen werden).
- Das Wissen über Angriffe ist meist abhängig von Betriebssystem, Version, Plattform und Anwendungen. Die daraus resultierenden ID-Werkzeuge sind daher eng an die gegebene Umgebung gebunden.
- Die Entdeckung von Insider-Angriffen, die meistens durch den Missbrauch von Privilegien ausgeführt werden, ist schwierig, weil der Angreifer dabei meist keine Verwundbarkeiten des Systems zu Nutze zieht.

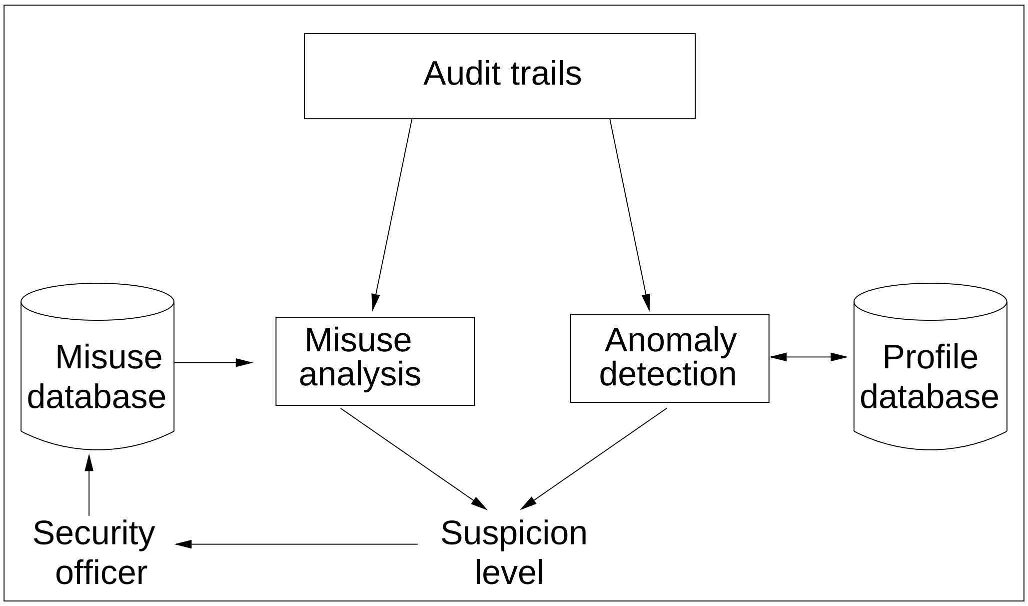

Abbildung 3.3: Structure of an IDS (Proceedings of the Workshop on Future Directions in Computer Misuse and Anomaly Detection, University of California, Davi, March 1992) [CFGP94]

Zu bemerken wäre noch, dass gegenwärtig nahezu alle IDS-Werkzeuge wissensbasiert sind. Oft implementieren ID-Systeme auch beide Ansätze, was in vielen Fällen die effizientere Lösung ist.

Die Grundstruktur eines ID-Systems mit Anomalie- und Missbrauchs-Entdeckung wird in Abb. 3.3 beschrieben.

3.3 Dorothy Denning's Intrusion Detection Modell

Die Ergebnisse der, in den frühen 80'er Jahren beim SRI durchgeführten Studien, bildeten die Basis für die Entwicklung einer ganzen Serie von IDS-Prototypen unter dem Namen IDES (Intrusion Detection Expert System). Dabei baute jeder Nachfolge-Prototyp auf den Vorgänger-Prototypen auf und erweiterte diesen um gewisse Funktionen und Eigenschaften.
Zwischen 1985 und 1987 wurde von Dorothy Denning und P.G. Neumann ein Intrusion Detection Modell entwickelt (ebenfalls beim SRI), auf dem die nachfolgenden IDES-Prototypen aufbauten. Der in dieser Diplomarbeit vorgestellte IDES-Prototyp stammt aus dem Jahr 1992 und basiert weitgehend auf diesem Modell.

Im folgenden Abschnitt wird dieses Modell näher beschreiben und uns anschließend der IDES-Prototyp vorgestellt.

Dorothy Denning's *Intrusion Detection Model* beschreibt ein Expertensystem, welches in der Lage ist, Einbrüche und Missbräuche von Computersystemen in Echtzeit zu erkennen. Das Modell basiert auf der Annahme, dass die Ausbeutung von Sicherheitsmängeln eines Systems zu einer anomalen Benutzung des Systems führt, was sich dann als anomales Muster der Systemnutzung abzeichnet. Es beinhaltet *Profile* zur Beschreibung des Verhaltens von Subjekten, in Bezug auf Objekte, in Form von Maßzahlen (metrics), statistischen Modellen und Regeln. Das Modell ist unabhängig von

- System,
- Anwendungsumgebung,
- Systemschwachstellen,
- Art des Einbruchs.

Evaluationsregeln für die Entdeckung von anomalem Verhalten sind ebenso definiert, wie die Aktionen, die überwacht werden müssen. Die Aktionen schliessen sowohl nicht-autorisierte, als auch autorisierte Anfragen mit ein (apparently legitimate).

3.3.1 Modellbeschreibung

Das IDES-Sicherheitsmodell umfasst folgende Elemente :

1. **Subjekte :** Subjekte sind die Initiatoren der Aktivitäten im überwachten System. Im Allgemeinen stellen die Benutzer die Subjekte dar. Ein Prozess

kann aber auch als ein Subjekt betrachtet werden, das im Namen von Benutzern oder Benutzergruppen, oder im Namen des Systems, operiert. Alle Aktivitäten werden von *Benutzerbefehlen* (user commands) initiiert. Subjekte können in Klassen gruppiert werden, um auch die Zugriffe, die von mehreren Benutzern auf die Objekte im System durchgeführt werden, zu berücksichtigen. Die für diesen Zweck formierten Benutzergruppen können sich überschneiden, d.h. Benutzer können zu mehr als einer Gruppe gehören.

2. **Objekte :** Ressourcen, die vom System unterhalten werden. Es sind die Entitäten, auf denen die Aktionen ausgeführt werden. Objekte sind : *Dateien, Programme, Nachrichten, Datensätze* (records), *Terminals,Drucker,* und *Strukturen,* die von Programmen oder Benutzern erzeugt werden.

   Die Subjekte, auf denen Aktionen ausgeführt werden können, können auch als Objekte betrachtet werden. Objekte können, ihrem Typ (z.B. Text- oder Programmdateien) oder anderen Beziehungen entsprechend, in Klassen gruppiert werden (z.B. können records in Dateien oder Datenbankrelationen und Dateien innerhalb von Verzeichnissen zusammengefasst werden). Die Granularität der kontrollierten Objekte hängt von dem überwachten System ab. Für einige Datenbankanwendungen ist z.B. die Betrachtung der Daten im *record-level* vorzuziehen, für andere Anwendungen ist eine Datei-/Verzeichnis- Granularität ausreichend.

3. **Audit-Datensätze (Audit-Records) :** Sie werden vom dem überwachten System, als Reaktion auf die von den Subjekten ausgelösten Aktionen, generiert. Deshalb repräsentieren *Audit-Datensätze*[1] Zugriffswünsche der Benutzer (Subjekte) auf die Objekte. Jeder Datensatz (Record) ist ein 6-Tupel, bestehend aus :

   $\langle Subjekt, Aktion, Objekt, Ausnahmebedingung, Ressourcennutzung, Zeit\rangle$
   wobei folgendes gilt :

   - **Subjekt :** Das Subjekt, das die Aktion anfordert;
   - **Aktion :** Eine Zugriffsoperation (z.B. login, logout, read, execute);
   - **Objekt :** das Objekt, auf dem die Aktion ausgeführt werden soll;
   - **Ausnahmebedingung :** Wurde durch die Aktion eine Ausnahmebedingung ausgelöst, so wird sie in dieser Komponente verzeichnet.
   - **Ressourcen-Benutzung :** Eine Liste von quantitativen Elementen, von denen jedes, die Benutzungshäufigkeit einer bestimmten Ressource zur Verfügung stellt, z.B. die Anzahl der gedruckten Zeilen oder Seiten, Anzahl der gelesenen/geschriebenen Records, die CPU oder I/O Benutzungszeit, die Dauer einer Sitzung.

[1] im Folgenden mit Audit-Record bezeichnet

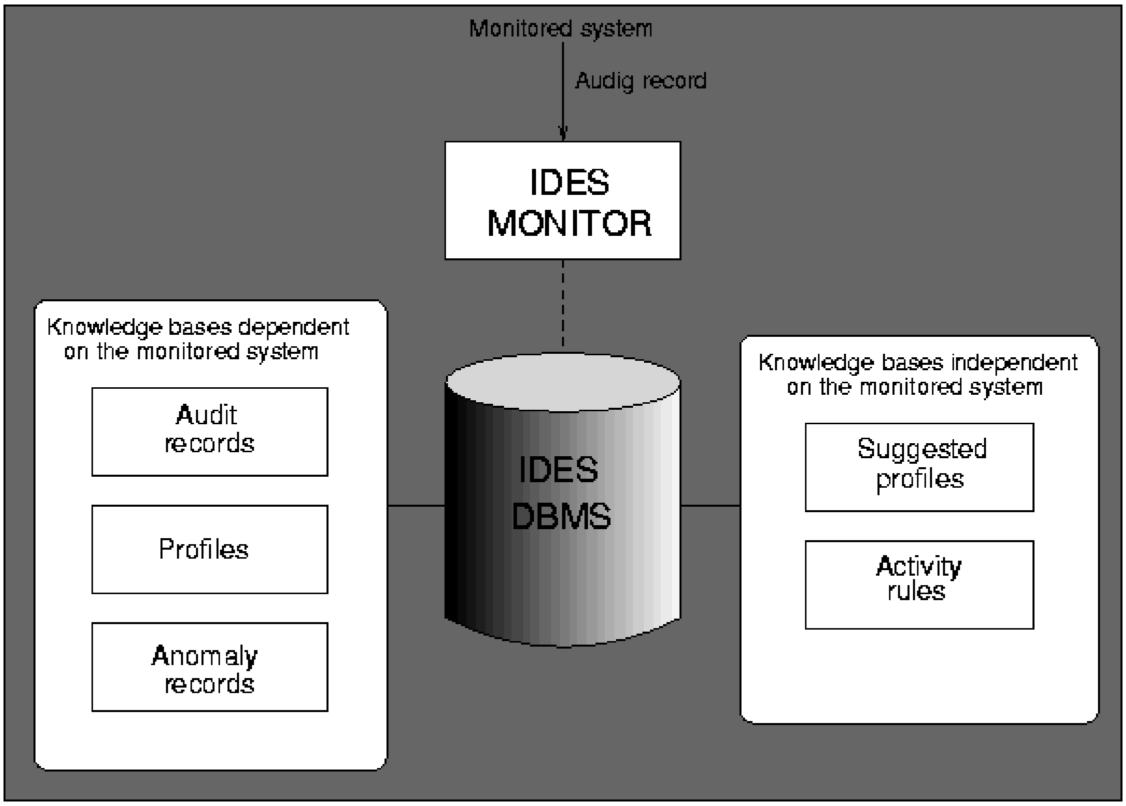

Abbildung 3.4: IDES Modell-Elemente [CFGP94]

- **Zeit :** Zeitpunkt der Aktionsausführung.

Viele Operationen in dem System schließen mehrere Objekte ein. Beim Kopieren einer Datei wird z.B. die Datei, die das Kopierprogramm beinhaltet, die Originaldatei und die Datei, die die Kopie beinhalten soll, mit in den Kopiervorgang eingeschlossen. Alle Aktionen und Aktivitäten sind in elementare Aktionen auf einzelne Objekte zerlegt. Das Kopieren einer Datei z.B. ist zerlegt in eine Ausführungsaktion auf dem Kopier-Programm (execution action), eine Leseaktion auf der Original-Datei (read action) und eine Schreibaktion auf der Datei, die die Kopie enthält. Die Zerlegung komplexer Aktionen hat den Vorteil, dass dadurch den Schutzmechanismen des Systems entsprochen wird. Durch die Zerlegung der Aktionen stimmen die Objekte mit den Systementitäten überein, die schutzbedürftig sind. Somit ist auch das Modell und dessen Anwendungen vereinfacht. Zuletzt ist die Zerlegung auch kompatibel mit dem Audit-System in der die Audit-Records, produziert von den zu überwachenden Systemen, sich gewöhnlich auf Aktionen beziehen, die auf jedem einzelnen Objekt ausgeführt wurden.

4. **Profile :** Es sind Strukturen, die das Verhalten von Subjekten in Bezug auf die Objekte, anhand von *metrischen Ausdrücken* (terms of metrics) und *statistischen Modellen* (statistical models) charakterisieren. Jedes Aktivitäts-Profil charakterisiert das normale Verhalten eines Subjektes oder einer Menge von Subjekten, in Bezug auf ein Objekt oder einer Menge von Objekten. Präziser ausgedrückt : Jedes Profil beschreibt einige besondere Verhaltensaspekte des betrachteten Subjektes.

   Einige Profile können beispielsweise Leseoperationen überwachen, um mögliche *Schlussfolgerungsbedrohungen* aufzuzeigen, während andere Schreiboperationen überwachen, um die Gefahr der falschen Modifikation von Informationen zu vermeiden. Das normale Verhalten, festgelegt von IDES durch das Beobachten der Benutzeraktivitäten, wird mit Merkmalräumen, bzw Meßräumen (metrics) und statistischen Modellen ausgedrückt. Informationen über die Merkmalräume und das statistische Modell, mit dem ein Profil definiert und überwacht wird, werden in demselben Profil angezeigt.

   Ein **Aktivitäts-Profil** wird von einem 10-Tupel beschrieben :

   ⟨ *Variablenname, Aktionsmuster, Ausnahmemuster, Ressourcen Benutzungsmuster, Periode, Variablentyp, Grenzwert, Subjektmuster, Objektmuster, Wert* ⟩

   Die einzelnen Elemente des Tupels können folgendermaßen erläutert werden:

   - *Variablenname :* Der Name einer Variablen.
   - *Aktionsmuster :* Entspricht keinem oder mehreren Aktionen in dem Audit-Record (z.B. login, read, execute, usw.).
   - *Ausnahmemuster :* Entspricht den Ausnahmebedingungsfeldern auf einem Audit-Record,
   - *Ressourcen-Benutzungsmuster :* Entspricht dem Ressourcen-Benutzungsfeld auf einem Audit-Record.
   - *Periode :* Beschreibt die Länge der Zeitperiode, auf welche sich die Beobachtung bezieht.
   - *Variablentyp :* Definiert den Merkmalraum (metric) und das statistische Modell, auf dem das Profil definiert ist, z.B. *Ereignis-Zähler* (event counter) und *Mittelwert/Standard-Abweichungsmodell* (average/standard deviation model).
   - *Grenzwert :* Ein Parameter, der den Grenzwert (oder Grenzwerte) definiert, der in dem statistischen Test zur Feststellung der Anomalie,

benutzt wird. Die Interpretation dieses Feldes hängt von dem benutzten statistischen Modell ab, angezeigt durch das Element *Variablentyp*. Für das operationale Modell beispielsweise (vgl. 3.3.3.2), repräsentiert dieses Feld eine Obergrenze (Untergrenze), mit dem eine Beobachtung verglichen werden muss. Für das *Mittelwert/Standard-Abweichungsmodell* beschreibt es die Anzahl der Standardabweichungen, die annehmbar hinsichtlich des Mittelwertes sind.

- *Subjektmuster :* Entspricht dem Subjekt-Feld auf einem Audit-Record.
- *Objektmuster :* Entspricht dem Objekt-Feld auf einem Audit-Record.
- *Wert :* Der Wert der aktuellsten Beobachtungen und der von dem statistischen Modell zur Darstellung der Verteilung vorheriger Werte benutzten Parameter. Für das *Mittelwert/Standard-Abweichungsmodell* z.B. bestehen die Parameter aus der Gesamtanzahl der Beobachtungen, der Summe der Beobachtungswerte und der Quadratsumme der Beobachtungswerte. Ein operationales Modell braucht keine Parameter, weil es nicht direkt von der letzten Beobachtung abhängt.

Die ersten sieben Felder des Aktivitäts-Profils sind von dem betrachteten Subjekt und Objekt unabhängig, während für die restlichen drei Felder dies nicht zutrifft. Ein Profil wird eindeutig durch den *Variablennamen*, die *Subjektmuster-* und *Objektmuster-Felder* definiert, welche den Typ des Profils (das Subjekt und das Objekt), für den das Profil definiert ist, anzeigen. Alle Profilkomponenten sind unveränderlich, außer dem *Werte-Feld*, welches jedesmal, wenn eine Aktion unter der Profilaufzeichnung ausgeführt wird, modifiziert wird.

Wenn IDES einen Audit-Record empfängt, die dem Muster *Variablenname* entspricht, führt es ein update der Variablenverteilung, und einige Kontrollen zur Entdeckung möglicher Anomalien durch. Aktivitäts-Profile können in Bezug auf einzelne Subjekte und Objekte (oder Gruppen von Subjekten/Objekten) definiert werden. Die Aktivitäts-Profile werden von IDES, bei der ersten Benutzung eines Objektes durch ein Subjekt, automatisch generiert. Auf diese Weise werden nur relevante Profile berücksichtigt, was einer realen Benutzung des Systems entspricht. Die Definition und das Management der Profile für alle möglichen Subjekt/Objekt-Kombinationen, die nie an einer Zugriffsanforderung beteiligt sind, wird auf diese Weise vermieden.

5. **Anomalie-Datensätze (anomaly records) :** Aufzeichnungen, die das unbefugte Verhalten der Benutzer beschreiben. Sie werden, hinsichtlich der definierten Verhaltens-Profile, aufgrund beobachteter Anomalien, generiert. Jedesmal, wenn ein Audit-Record erzeugt oder eine *Periode* beendet ist,

wird von IDES ein update der Aktivitäts-Profile und eine Suche nach möglichen Verhaltensanomalien, mit Hilfe der Aktivitäts-Regeln, durchgeführt. Falls während dieses Vorgangs ein anomales Verhalten entdeckt wird, wird ein *Anomalie-Datensatz*[2] generiert, die die entdeckte Verhaltensanomalie beschreibt. Die Anomalie-Records werden durch einen 3-Tupel der folgenden Form definiert :

$\langle Ereignis, Zeit, Profil \rangle$

- *Ereignis (event) :* Zeigt das Ereignis an, dass die Anomalie verursacht hat. Es nimmt den *Audit*-Wert an, wenn die Anomalie von einem einzelnen Audit-Record verursacht wurde (das ist der Fall bei einer Einzelaktion). Es nimmt den *Period*-Wert an, wenn die Anomalie einer Menge von Aktionen in einer gegebenen Zeitperiode entspringt.
- *Zeit (time) :* Wenn die Anomalie auf einen besonderen Audit-Record verweist, gibt dieses Attribut die im Record angezeigte Zeit an. Sonst zeigt es den Endzeitpunkt der Periode an, auf den sich die Anomalie bezieht.
- *Profil (profile) :* Es ist das Aktivitäts-Profil, bezogen auf die festgelegte Anomalie .

6. **Aktivitäts-Regeln :**

   Eine Aktivitäts-Regel spezifiziert die Aktion, die ausgeführt werden soll, wenn eine Audit- oder Anomalie-Aufzeichnung generiert wird, oder wenn eine Zeitperiode abläuft. Aktivitäts-Regeln bestehen aus einer *Bedingung*, welche, wenn erfüllt, die Regelausführung auslöst. Die Bedingung ist spezifiziert als ein Zusammenspiel zwischen einem Muster und ein Ereignis. Aktivitäts-Regeln können in vier Klassen gruppiert werden :

   - *Audit-Aufzeichnungsregeln :* Diese sind aktiviert, wenn ein neuer Audit-Record dem Aktivitäts-Profil entspricht. Sie aktualisieren das Profil, um die geprüfte Aktion widerzuspiegeln und kontrollieren mögliche Verhaltens-Anomalien. Ist eine Anomalie entdeckt worden, wird der entsprechende Anomalie-Record generiert.
   - *Regeln für die periodische Aktualisierung der Aktivität :* Diese Regeln sind bei Beendigung eines Intervalls, dessen Dauer der Perioden-Komponente eines Aktivitäts-Profils entspricht, aktiviert. Sie aktualisieren das Profil und suchen nach möglichen Verhaltensanomalien. Bei Anomalie-Entdeckung wird auch ein entsprechender Anomalie-Record generiert.

[2] im Folgenden mit Anomalie-Record bezeichnet

- *Anomalie-Aufzeichnungsregeln :* Diese sind bei Erzeugung eines Anomalie-Records aktiviert.

  Mit Hilfe diese Regeln wird der Sicherheitsbeauftragte auf die Anomalie aufmerksam gemacht, indem der Typ der vermuteten Verletzung angezeigt wird.

- *Regeln für die periodische Analyse der Anomalien :* Diese Regeln werden periodisch aktiviert, so dass Berichte über Anomalien generiert werden können, die in einer gegebenen Referenz-Periode entdeckt wurden.

3.3.2 Grundbeziehung zwischen Angriffstypen und Verhaltensanomalien

Auf Grund der gestellten Annahme, dass die Ausnutzung von Systemverwundbarkeiten zwecks eines Angriffes, zu einer feststellbaren, anomalen Benutzung des Systems führt, wurden in dem ID-Modell von Dorothy Denning folgende Grundbeziehungen zwischen Angriffstypen und Verhaltens-Anomalien aufgestellt :

- *Einbruchsversuch* : Viele Login-Versuche benutzen mehrere Passwörter unter der gleichen Account-ID, oder das gleiche Passwort unter verschiedenen Accounts;

- *Maskerade (masquerading)* : Einbrüche durch legitimierte Logins (korrekter Account und Passwort, möglicherweise gestohlen oder kopiert) und daraus folgende, abweichende Systemnutzungsmuster (z.B. in Verzeichnissen „herumschmökern" anstelle von üblichen Ausführungsaktionen, wie editieren, kompilieren, linken usw.).

- *Eindringen duch legitimierte Benutzer* : Autorisierte Benutzer, die versuchen die Sicherheitskontrollen zu überlisten (Nutzung von unüblichen Programmen)

- *Ausbreitung von Daten durch autorisierte Benutzer* : Ein Benutzer versucht Zugang zu sensitiven Daten zu erlangen durch das „Einloggen" in unüblichen Zeiten, durch das Schreiben in einer unüblichen Art, durch das Benutzen von räumlich weit entfernten Druckern oder durch das Erzeugen von mehr Kopien als üblich.

- *Schlussfolgerung durch autorisierte Benutzer* : Das Erhalten von vertraulichen Daten durch Aufsummierung oder Schlussfolgerung (gekennzeichnet durch eine unnormal große Anzahl von Datenbankanfragen).

- *Trojanische Pferde* : Ein trojanisches Pferd, eingesetzt in ein Programm, oder das Ersetzen des Programms durch das trojanische Pferd, kann die

Benutzungsraten von CPU, von Speicher, Geräten usw. modifizieren (z.B. kann die Anzahl der Zugriffe auf Verzeichnisse plötzlich wachsen).

- *Viren* : Diese können ein Anwachsen der *rewriting*-Häufigkeiten in ausführbaren Dateien, oder einen Anstieg der Ausführungshäufigkeiten einiger Programme hervorrufen.

- *Denial of Service* : Ein Benutzer, der eine Ressource blockieren kann, hat eine sehr hohe Zugriffsrate auf diese Ressource, im Vergleich zu anderen Benutzern.

Bei der Aufstellung dieser Beziehungen stellte sich das Problem, dass Verhaltens-Anomalien nicht zwingenderweise einen Einbruch darstellen. Abweichendes Verhalten eines Benutzers kann auch als Folge seines veränderten Aufgabenbereichs zustande kommen, oder eine erhöhte Zahl von schreibenden Zugriffen auf ausführbare Dateien kann das Zeichen eines Software-Updates sein. Das Optimierungsziel des Modells war daher, ein Maximum an Erkennungsrate bei gleichzeitigem Minimum der Falsch-Positiv Alarm-Rate zu erreichen.

3.3.3 Analyse von Verhaltens-Anomalien

Die Definition von einem Intrusion Detection-Modell verlangt nach Profilen und Regeln, die festgelegt sind. Insbesondere kann die Definition von Profilen in metrischen Ausdrücken (metric terms) und statistischen Modellen angegeben werden. Eine *metric* (Merkmalraum) ist eine Zufallsvariable X, welche ein quantitatives Mengenmaß repräsentiert, das über eine gegebene Beobachtungsperiode akkumuliert wird (entweder ein festes Zeit-Intervall oder das Intervall zwischen zwei, voneinander abhängigen, Ereignissen).

Die Beobachtungen (sample points) der Benutzeraktionen, welche Stichproben x_i von X sind, werden in einem statistischen Modell dazu benutzt, festzulegen, ob eine Beobachtung als normal in Erwägung zu ziehen ist. Das statistische Modell ist unabhängig von der Verteilung von X, daher kann das gesamte Wissen über X von den Beobachtungen abgeleitet werden.

In den folgenden Abschnitten werden die „Basis-Merkmalräume“ (basic metrics) und einige „statistische Modelle“ (statistical models) erklärt, die für die Analyse der Verhaltens-Anomalien benutzt werden können.

3.3.3.1 Merkmalräume (Metrics)

Merkmalräume beschreiben, wie einzelne Beobachtungen des Benutzerverhaltens zur Erzeugung eines Profils zusammengefasst werden können. Einige *Merkmalräume*, die benutzt werden können sind folgende:

- *Ereignis-Zähler (event counter)* : X ist die Anzahl der Ereignisse, die in einer bestimmten Periode aufgezeichnet werden und bestimmte Eigenschaften besitzen. X könnte z.B. die Anzahl von Logins innerhalb einer Stunde, die Anzahl der Zeiten, in der ein bestimmter Befehl bei einer Sitzung ausgeführt wurde, die Anzahl der Passwort-Misserfolge in einer Minute, oder die Anzahl der Zugriffsverletzungen auf Dateien an einem Tag, definieren.

- *Zeit-Intervall (time interval)* : X ist das Zeit-Intervall zwischen zwei sich bedingenden Ereignissen, z.B. das Intervall zwischen zwei (zeitlich) aufeinander folgenden Logins, die einem einzigen account zugeordnet sind.

- *Ressourcen-Zähler (resource measure)* : X ist die Anzahl von Ressourcen, die von einem System-Prozess, in einem gegebenen Zeitabschnitt benutzt werden, z.B. die gesamte Menge an Seiten, die von einem Benutzer an einem Tag ausgedruckt wurden, die gesamte CPU-Zeit eines laufenden Programms, oder die Anzahl der an einem Tag gelesenen Datensätze.
  Die Ressourcen-Belegung ist modelliert in Termen von Ereignis-Zählern oder Zeit-Intervallen. Die Anzahl der ausgedruckten Seiten ist z.B. ein Ereignis-Zähler.

3.3.3.2 Statistische Modelle (statistical models)

Zu einer ein-dimensionalen Zufallsvariablen X wurden eine Folge $x = (x_i)_1^n$ von Werten beobachtet. Die wichtigsten Modelle, mit deren Hilfe entschieden werden kann, ob die Beobachtung x_{n+1} eine Anomalie darstellt, sind:

1. **Operationales Modell**: Dieses Modell arbeitet mit Schwellwerten . Hier wird angenommen, das eine Beobachtung für die $|limX - x_{n+1}| > |limX - s|$ gilt, eine Anomalie darstellt. Der Wert s wird als Schwellwert bezeichnet und $limX$ gibt die erlaubte Schwankung um den Schwellwert an. Die Schwellwerte werden aus den Erfahrungen mit dem System in der Vergangenheit abgeleitet. Da dieses Modell keinerlei Annahmen hinsichtlich eines zugrunde liegenden stochastischen Modells macht, wird es als operationales Modell bezeichnet.

2. **Mittelwert und Standard-Abweichung**: Ist X ein bedingtes Merkmal, so lässt sich aus Folge x der Mittelwert $\bar{x}$ und die Standardabweichung

$$s_x = \sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

ermitteln. Aus s_x und einer natürlichen Zahl d wird nun

$$ki = d \times s_x$$

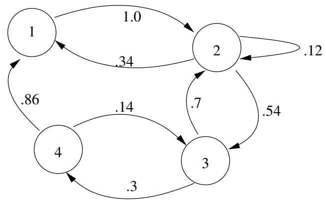

Abbildung 3.5: Beispiel einer Übergangsmatrix

das Konfidenzintervall gebildet. Gilt nun:

$$|x_{n+1} - \bar{x}| > ki$$

so ist x_{n+1} eine Anomalie. Die Wahrscheinlichkeit, mit der Werte außerhalb des Konfidenzintervalls beobachtet werden, sinkt mit $\frac{1}{d^2}$. Bei diesem Modell werden empirische Verteilungen benutzt.

3. **Multivariates Modell**: Dieses Modell ist eine Erweiterung des Mittelwertes und des Standard-Abweichungsmodells. Hier werden jedoch die Abhängigkeiten von Merkmalen untereinander mit einbezogen. Dieses Modell kommt besonders dann zur Geltung, wenn sich gezeigt hat, das Anomalien besser unter Berücksichtigung mehrerer Merkmale erkannt werden können.

4. **Markov Modell**: Gegeben sei ein endlicher, abzählbarer Zustandsraum $I = \{i_1, \ldots, i_q\}$ und die Folge $x = (x_i)_1^n \in I^n$ von Zuständen. Für die Übergangswahrscheinlichkeiten $P(x_k = i_h | x_{k+1} = i_j)$ existiert eine Übergangsmatrix (p_{ij}). Die Matrix lässt sich auch als Übergangsgraph veranschaulichen. Dieses Modell setzt voraus, dass die Wahrscheinlichkeit eines Zustandsüberganges nur vom aktuellen Zustand abhängt. D. h., die vorangegangen Übergänge beeinflussen die Übergangswahrscheinlichkeit nicht. Diese Art der Kopplung wird auch als Markov-Kopplung bezeichnet. Die Folge von Beobachtungen $x = (x_i)_1^n$ wird Markovkette genannt. Wenn die Übergangswahrscheinlichkeiten konstant sind, werden die Folgen x auch homogene Markovketten genannt. Die Beobachtung x_{n+1} ist dann eine Anomalie, wenn sie einen Zustandsübergang darstellt, der "*unwahrscheinlich*" war.

- **Zeit-Reihen Modell (Time series model)** : Dieses Modell zieht den Zustand und die Intervall-Zeiten, die zwischen den Beobachtungen liegen, einschließlich der beobachteten Werte, in Betracht. Eine neue Beobachtung wird als anomal definiert, wenn die Wahrscheinlichkeit ihres Auftretens, zum Zeitpunkt der Beobachtung, zu niedrig ist. Vorteil des Modells ist, dass es die zeitlichen Veränderungen beim Verhalten eines Benutzers nachvollziehen und messen kann.

3.3.4 Profil Charakterisierung (Profile characterization)

Einige Profile (Verhaltensaspekte), die dazu benutzt werden können das aktuelle Verhalten von Benutzern zu vergleichen, sind in dem IDES-Modell beschrieben worden. Für jedes ist ein *Merkmalraum* und ein *statistisches Modell* vorgeschlagen worden.

Die Profile werden hinsichtlich der Subjekte, die die Aktionen ausführen, die Objekte, auf denen die Aktionen ausgeführt werden, und den Typ der Aktionen, definiert.

Profile können für jedes Subjekt bezogen auf jedes Objekt, oder für Klassen von Subjekten und/oder Objekten definiert werden. In Abb. 3.6 sind alle möglichen Subjekt/Objekt Kombinationen aufgeführt, die überwacht werden können. Dabei bedeutet

- *System*, dass die Aktionen *irgendeines* Subjektes, ausgeführt auf *irgendeinem* Systemobjekt, in Betracht gezogen werden.
- *Subjekt*, dass die Aktionen eines *gegebenen* Subjektes, ausgeführt auf *irgendeinem* Systemobjekt, in Betracht gezogen werden
- *Objekt*, dass die Aktionen *irgendeines* Subjektes, ausgeführt auf ein *gegebenes* Objekt, in Betracht gezogen werden.

Andere Kombinationen haben eine direkte Interpretation.

3.3.4.1 Login- und Sitzungsaktivitäts-Profile

Login- und Sitzungsaktivitäts-Profile ziehen die Login-/Logout-Aktionen in Betracht, die von einem Subjekt (user) auf den Objekten, repräsentiert durch Terminals, Workstations, Netzwerke, entfernte Hosts usw., ausgeübt werden.
Diese Profile können durch folgende Ausdrücke charakterisiert werden:

- *Login-Häufigkeit (login frequency):* Diese überwacht die Login-Häufigkeit. Da das Benutzerverhalten sich, z.B. im Verlauf einer Woche, beträchtlich ändern kann, können Login-Vorgänge/Login-Ereignisse als ein Vektor von Ereignissen, parametrisiert mit Rücksicht auf den Wochentag (spezieller Tag, Wochenende usw.) und den Tagesabschnitt (Zeit/Turnus, morgens/-nachmittags, abends/nachts), repräsentiert werden. Login-basierte Profile können dazu herangezogen werden, Benutzer zu entdecken, die im Allgemeinen über einen nicht-autorisierten account, außerhalb der Arbeitszeiten, Zugang zum System herstellen wollen. Login-Profile können für einzelne Benutzer (oder für Benutzergruppen) und für Objektklassen definiert werden.

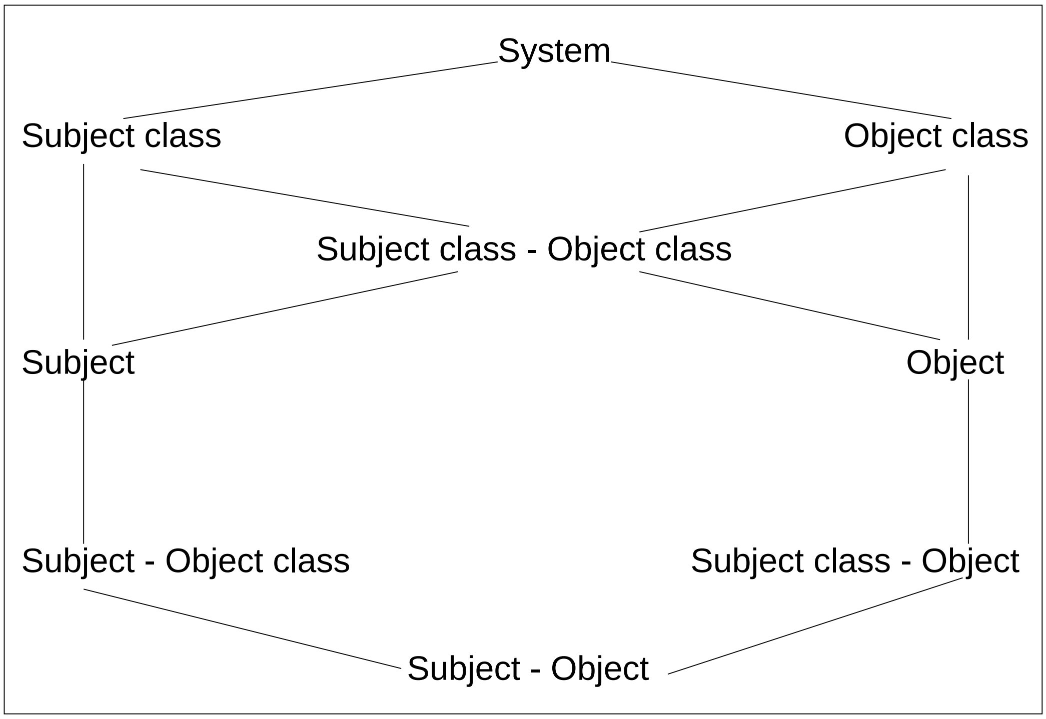

Abbildung 3.6: Hierarchie von Elementen, für die Verhaltens-Profile definiert werden können [CFGP94].

Benutzter Merkmalraum : Ereignis-Zähler
Benutztes Modell : Mittelwert/Standard-Abweichungsmodell

- *Standort-Häufigkeit (location frequency) :* Dieser Parameter überwacht die Login-Häufigkeit an verschiedenen Standorten. Einige Messungen, die diesem Profil ähneln, können separat betrachtet werden, bezogen auf den Wochentag und die Tageszeit, weil ein Benutzer die Verbindung zum System von verschiedenen Standorten aus herstellen kann, in Abhängigkeit davon, ob er dies inner- oder außerhalb der Arbeitszeiten durchführt. Die Analyse der *Standort-Häufigkeit* ist nützlich für die Feststellung, ob sich jemand mit einem bestimmten account von einem Standort aus einloggt, der durch legitime Benutzer nie in Anspruch genommen wird. Sie erlaubt auch die Identifizierung von Sicherheitsverletzungen durch legitime Benutzer. Es ist möglich, Verbindungen über „ privilegierte“ Terminals festzustellen, die durch Benutzer hergestellt werden, die gewöhnlich an „nicht privilegierten“ Terminals arbeiten.

  Benutzter Merkmalraum : Ereignis-Zähler
  Benutztes Modell : Mittelwert/Standard-Abweichungsmodell

- *Letztes Login (last login) :* Überwacht die verstrichene Zeit seit dem letzten Login. Dieses Profil kann für Einzelbenutzer und Standort-Klassen (location classes) definiert werden. Es ist nicht nötig, für jeden einzelnen Standort ein Profil zu definieren, weil der exakte Standort irrelevant ist. Das Profil kann für die Aufdeckung von Bedrohungen durch „tote", d.h. nicht mehr benutzte, accounts herangezogen werden.

  Benutzter Merkmalraum : Zeit-Intervall
  Benutztes Modell : Operationales Modell

- *Sitzungsdauer (session duration) :* Dieses Profil kann die Dauer einer Sitzung überwachen. Dieses Profil kann für einzelne Benutzer oder Gruppen definiert werden. Abweichungen vom Standardverhalten sind dabei symptomatisch für „maskierte" Angriffe.

  Benutzter Merkmalraum : Zeit-Intervall
  Benutztes Modell : Mittelwert/Standard-Abweichungsmodell

- *Sitzungsausgaben (session output) :* Dies überwacht die Menge der von einem Terminal produzierten *outputs* einer Sitzung oder eines Tages. Die Definition dieses Profils für individuelle Standorte oder Klassen von Standorten ist für die Bestimmung von Datenüberladungen (data overloads)von Bedeutung, die auf entfernte Standorte übertragen werden, was als ein Zeichen für die Übermittlung von sensitiven Daten gesehen werden kann.

  Benutzter Merkmalraum : Ressourcen-Zähler
  Benutztes Modell : Mittelwert/Standard-Abweichungsmodell

- *Ressourcen-Benutzung pro Sitzung :* Mit diesem Parameter wird die Benutzung von Ressourcen in einer Sitzung kontrolliert. Das Profil kann zur Aufdeckung von maskierten Angriffen benutzt werden.

  Benutzter Merkmalraum : Ressourcen-Zähler
  Benutztes Modell : Mittelwert/Standard-Abweichungsmodell

- *Passwort Mißerfolge (password failures) :* Dieses Profil überwacht die Anzahl der Passwort-Misserfolge in Bezug auf ein gegebenes Login. Der Ansatz hierbei ist die Annahme, dass ein Angriff, ausgeführt durch viele Passwort-Attacken von einem gegebenen account aus, zu einer ziemlich hohen und für einen einzelnen Benutzer ungewöhnlichen Anzahl von Misserfolgen führen muss, was als ein Zeichen für einen Angriff interpretiert werden kann.

Benutzter Merkmalraum : Ereignis-Zähler
Benutztes Modell : Operationales Modell

- *Standort Misserfolge (location failures) :* Dieses Profil überwacht die Login-Misserfolge und kann für einzelne Benutzer oder Gruppen von Standorten definiert werden. Standortgruppen werden deswegen in Betracht gezogen, weil gewöhnlich eine bestimmte Sitzung für Kontrollzwecke irrelevant ist, während die Tatsache, dass ein Standort nicht autorisiert ist, von großer Bedeutung sein kann. Das Profil kann zur Feststellung von Einbruchsversuchen benutzt werden, die über privilegierte Terminals ausgeführt werden.

  Benutzter Merkmalraum : Ereignis-Zähler
  Benutztes Modell : Operationales Modell

3.3.4.2 Befehls- und Programmausführungs-Profile

Die Befehls- und Programmausführungs-Profile kontrollieren die *execute*-Aktionen, die Benutzer auf Programmen ausführen. Die Befehle (commands) können auch als *Aufrufprogramme* (called programs) betrachtet werden. Durch das Definieren von Profilen können Programme klassifiziert und zusammengefasst werden, indem unterschieden wird zwischen :

- Privilegierten Programmen (ausführbar nur durch besondere Benutzer, die über einen privilegierten Modus Zugang zum System haben) und nicht-privilegierten Programmen,
- Systemprogrammen und Benutzerprogrammen oder,
- Programmen, die von erfahrenen Benutzern, und Programmen, die von unerfahrenen Benutzern genutzt werden.

Die Prüfung von Profilen in Bezug auf jedes einzelne Programm ist unnötig, obwohl es realisierbar wäre. Einige Programme jedoch können besondere Profile benötigen, um Maskierungen, Einbruchsversuche oder trojanische Pferde festzustellen. Diese Profile sollten in Bezug auf Benutzer zusammengefasst werden. Programme, die möglicherweise individuelle Kontrollen benötigen, sind privilegierte Programme, sicherheitsrelevante Programme, Programme, die *browsing*

Aktionen ausführen, und Programme, die häufig benutzt werden. Das *Programmausführungs-Profil* kann mit nachfolgenden Ausdrücken charakterisiert werden :

- *Ausführungs-Häufigkeit (execution frequency) :* Mit diesem Profil werden die Login-Misserfolge in einer Zeitperiode geprüft. Es kann für einzelne Benutzer oder für Benutzergruppen, für einzelne Programme oder für Programm-Klassen definiert werden. Ein Profil für einzelne Benutzer, in Bezug auf einzelne Programme, kann dazu benutzt werden, maskierte Angriffe, oder erfolgreiche Einbrüche, ausgeführt durch legitime Benutzer, aufzudecken. Dies ist deswegen möglich, weil ein Benutzer, der versucht, mit einem unüblichen account in das System einzudringen, im Allgemeinen Programme benutzt, die sich von jenen unterscheiden, mit denen sich legitime Benutzer beschäftigen. Bei Angriffsversuchen durch legitime Benutzer, können diese Zugang zu privilegierten Programmen erlangen, die nicht in ihren Profilen aufgeführt werden. Ein Profil, bezogen auf alle Benutzer für ein einzelnes Programm, kann zur Aufdeckung von trojanischen Pferden benutzt werden. Die Ausführungshäufigkeit des Original-Programms ändert sich infolge von Aktionen, die durch den versteckten Code (hidden code) initiiert werden. Ferner ist dieses Profil hilfreich bei der Erkennung von Viren, die während einer „viralen“ Aktion in einem Programmaufruf auftauchen.

  Benutzter Merkmalraum : Ereignis-Zähler
  Benutztes Modell : Mittelwert/Standard-Abweichungsmodell

- *Ressourcen-Benutzung :* Dieses Profil wird zur Überprüfung der Ressourcen-Benutzutzung bei einer Programmausführung benutzt. Das Profil kann für beide, Einzelbenutzer/Programme und Benutzer/Programm-Klassen, definiert werden. Ein unregelmäßiger Wert bei einer dieser Messungen, angewandt auf die ganze „Benutzer-Population“, suggeriert eine mögliche Einfügung von trojanischen Pferden oder Viren in die Original-Programme, die dann eine erhöhte CPU-Benutzungsrate oder I/O-Rate verursachen. Datenanhäufung, bezogen auf eine Zeitbasis (z.B. pro Tag) oder auf eine Ausführungsbasis, wäre weniger nützlich bei der Beendigung möglicher Einbrüche.

  Benutzter Merkmalraum : Ressourcen-Zähler
  Benutztes Modell : Mittelwert/Standard-Abweichungsmodell

- *Ausführungsverweigerung (denied executions) :* Überwacht wird hierbei die Anzahl der Versuche, die zur Ausführung unbefugter Programme unternommen werden, und die sich innerhalb einer bestimmten Zeitperiode ereignen (z.B. ein Tag). Dieses Profil, definiert für Einzelbenutzer, ist bei der Feststellung von Einbruchsversuchen hilfreich. Es kann auch für einzelne, hochsensitive, Programme definiert werden.

Benutzter Merkmalraum : Ereignis-Zähler
Benutztes Modell : Operationales Modell

- *Sättigung von Programm-Ressourcen (saturation of program resources) :* Es wird die Anzahl der Zeiten überwacht, in denen ein Programm, aufgrund von unzulänglichen Ressourcen, vorzeitig terminiert. Dieses Profil kann auf eine pro-Tag-Basis erstellt und für einzelne Programme oder Programm-Klassen, zu Überwachungszwecken abgebrochener Programme, definiert werden.

  Benutzter Merkmalraum : Ereignis-Zähler
  Benutztes Modell : Operationales Modell

3.3.4.3 Dateizugriffs-Profile

Dateizugriffs-Profile berücksichtigen *read, write, create, delete* und *append* Operationen, die auf Dateien ausgeführt werden. Dateien können, bezogen auf ihren Typ (Text-Dateien, ausführbare Programme, Verzeichnisse usw.), bezogen auf ihre Eigenschaft Benutzer- oder System-Dateien zu sein, oder nach anderen Eigenschaften, klassifiziert werden. Da ein Programm auch eine Datei ist, kann das Benutzen von Programmen, neben ihrer Überwachung hinsichtlich der Ausführung, auch durch die Überwachung der Dateizugriffsaktivitäten beobachtet werden. Die Anzahl der Dateien in einem System kann sehr hoch sein (Millionen), was die einzelne Kontrolle undurchführbar macht. Die Kontrolle von einzelnen, sicherheitsrelevanten Dateien ist realistischer (z.B. Passwort-Dateien, Dateien, die Autorisations-Informationen enthalten, Dateien, die Audit-Einträge oder Netzwerk Routing-Tabellen beinhalten). Dateizugriffs-Profile können hinsichtlich folgende Operationen definiert werden :

- *Read-, write-, create- und delete-Häufigkeit :* Diese kontrolliert die Zugriffszahlen und Typen der Zugriffsmodi. Zugriffshäufigkeits-Profile für read/write Operationen können für einzelne Benutzer und Dateien, oder für Benutzer-Klassen und Dateien, definiert werden. Profile für create/delete Operationen sind nur für Datei-Gruppen sinnvoll, weil jede Datei meist einmal erzeugt oder gelöscht wird. Anomalien bei read/write Zugriffen bei Einzelbenutzern können auf getarnte Angriffe hinweisen. Zusätzlich können sie einen erfolgreichen Einbruch kennzeichnen, weil ein Benutzer Zugriff auf Dateien bekommen kann, die gewöhnlich für ihn gesperrt sein sollten.

Benutzter Merkmalraum : Ereignis-Zähler
Benutztes Modell : Mittelwert/Standard-Abweichungsmodell

- *Read/written records :* Es wird die Anzahl der gelesenen/geschriebenen Datensätze überwacht. Dieses Profil kann bei jedem Zugriff oder auf einer pro-Tag-Basis gebildet werden. Es kann für einzelne Benutzer oder Dateien, oder für Benutzer- und Datei-Klassen, definiert werden. Anomalien können auf Versuche zur Erlangung von sensitiven Daten, durch Schlussfolgerungs- oder Gruppierungsprozesse, hindeuten (z.B. durch das Erbitten von langen, zusammenhängenden Datenmengen).

  Benutzter Merkmalraum : Ressourcen-Zähler
  Benutztes Modell : Mittelwert/Standard-Abweichungsmodell

- *Read-, write-, delete-, create-Misserfolge :* Überwacht wird die Anzahl der Zugriffs-Verletzungen in einem gegebenen Zeit-Intervall (z.B. ein Tag). Das Profil kann für einzelne Benutzer und Dateien, sowie für Benutzergruppen und Datei-Klassen, definiert werden. Einzelbenutzerprofile, die hinsichtlich einer Klasse alle Dateien enthalten, sind zur Entdeckung von Benutzern nützlich, die wiederholt versuchen, unberechtigten Zugang zu Dateien zu erlangen. Einzeldatei-Profile, hinsichtlich aller Benutzer, können zur Entdeckung unberechtigter Zugriffe auf hochsensitive Dateien herangezogen werden.

  Benutzter Merkmalraum : Ereignis-Zähler
  Benutztes Modell : Operationales Modell

- *Datei-Ressourcen Verbrauch :* Dieses Profil überwacht die Anzahl der erfolglosen Anfragen, die aufgrund von fehlendem Speicher entstehen. Definiert werden kann dieses Profil für einzelne Benutzer in Bezug auf alle Dateien. Eine Anomalie kann auf einen verdeckten Kanal hindeuten, in welchem der Signalierungsprozess den gesamten verfügbaren Speicherplatz verbraucht, um die flag-bits darin abzulegen.

  Benutzter Merkmalraum : Ereignis-Zähler
  Benutztes Modell : Operationales Modell

3.3.4.4 Datenbankzugriffs-Profile

Datenbankzugriffe können wie Dateizugriffe betrachtet werden. Da in relationalen Datenbank-Systemen jede Relation als eine separate Datei gespeichert wird, ist der Zugriff auf eine Relation analog zu Dateizugriffen. Dennoch sind die Operationen verschieden. In einer Datenbank muss der *retrieve*-, *update*-, *insert*- und *delete*-Zugriff auf die Datensätze (records) in der Relation bezogen werden und die *create/delete*-Modi auf die gesamte Relation.*Retrieve*-Operationen auf der Datenbank entsprechen *file-read*-Operationen; *update-, insert-, delete*-Operationen entsprechen *file-write*-Operationen. Wenn auditing-Funktionen auf der Relationsebene (relation level) ausgeführt werden, dann können Profile, die für das Überwachen der Datei-Aktivitäten benutzt werden, auch zur Überwachung der Datenbank-Aktivitäten herangezogen werden. Falls das auditing in einer niedrigeren Ebene ausgeführt wird (z.B. auf einzelnen Records), können dieselben Prinzipien, die auf Dateien angewandt werden, auch hierbei verwendet werden. Das *Datenbank Management System* (DBMS) muss aber, für die Erstellung von Audit-Daten in der Record-Ebene, unterstützt werden. Dennoch ist ein ID-System auf Record-Ebene, in Anbetracht der großen Datenvolumina, die höchstwahrscheinlich durch separate Prüfung der Zugriffe auf jedes Record generiert werden, nicht immer praktikabel (ein einfaches read könnte einen Zugriff auf Millionen von Records mit sich bringen, um den Durchschnitt ihrer Werte zu berechnen). Für einige Systeme ist es hinreichend auf dem Datenbank-level zu überwachen, ohne die Datenbank in ihre wesentlichen Datei-Bestandteile zerlegen zu müssen. In diesem Fall sind ebenfalls die gleichen Prinzipien, wie jene für die Dateien, anwendbar.

3.3.4.5 Profil-Modelle und Aktivitäts-Profil-Management

Profil-Modelle sind Muster, die für die Definition von neuen Aktivitäts-Profilen benutzt werden. Sie haben dieselbe Struktur wie die Aktivitäts-Profile, Subjekt- und Objektmuster sind jedoch anders definiert. Die Muster definieren die Audit-Records, denen die Profile entsprechen, und beschreiben, wie die Subjekt- und Objektmuster, die in den generierten Aktivitäts-Profilen hinzugefügt werden müssen, festgelegt werden können. Deshalb ist jedes Aktivitäts-Profil auf der Grundlage eines besonderen Profil-Modells kreiert, welches das Subjekt und das Objekt, auf das sich das Profil bezieht, spezifiziert. Wenn ein neuer Audit-Record empfangen wird, wird dieser mit den Aktivitäts-Profilen und dem Profil-Modell verglichen. Auf diese Weise werden alle schon existierenden Profile kontrolliert, sowie alle neuen, die von den Modellen generiert wurden. Ein, dem Audit-Record entsprechendes, neu generiertes Profil wird dann mit den schon existierenden verglichen. Falls dessen Werte gleich denen eines existierenden Profils sind, wird es verworfen, ansonsten eingefügt. Das Profil, das dem Audit-Record entspricht (generiert oder schon vorhanden), wird dann zur Auswertung vorgeschlagen.

3.4 IDES (Intrusion Detection Expert System)

3.4.1 Überblick

Der IDES-Prototyp, der in dieser Diplomarbeit beschrieben wird, wurde 1992 von SRI vorgestellt und basiert weitgehend auf Dorothy Denning's Intrusion Detection Modell (vgl. Abschnitt 3.3).
IDES wurde, als ein systemunabhängiger Mechanismus, für die Echtzeit-Entdeckung von sowohl externen als auch internen Sicherheitsverletzungen entworfen und entwickelt. IDES läuft selbstständig auf der eigenen Hardware. Dies bringt folgende Vorteile mit sich :

- *Performance :* Die Präsenz von IDES erhöht nicht die System-Antwortzeit,
- *Sicherheit :* IDES kann gegenüber dem überwachten System gesichert werden, damit ein Benutzer des kontrollierten Systems keinen Zugriff auf IDES bekommen kann, um es zu verändern oder Wissen über IDES auszuspähen. Daher können mögliche Fehler auf dem überwachten System die Integrität, die Verfügbarkeit und die Vertraulichkeit von IDES nicht beeinflussen.
- *Integration :* IDES kann leicht an verschiedene Umgebungen angepasst und mit mehreren Host-Systemtypen vereinheitlicht werden.

Der IDES-Prototyp verarbeitet Audit-Daten, die er von einem oder mehreren Zielsystemen empfängt. Jedes Zielsystem bietet eine Einrichtung, um die relevanten Audit-Daten zu sammeln und sie in das IDES Audit-Record-Format umzusetzen. Zu diesem Zweck wurde ein flexibles Format für die Audit-Records, die IDES von den Zielsystemen empfängt, und ein Protokoll für deren Übermittlung entwickelt. Das IDES Protokoll und das Audit-Record-Format sind systemunabhängig. Aus diesem Grund können verschiedene Systeme überwacht werden, ohne grundlegende Veränderungen bei IDES vornehmen zu müssen. Der IDES-Prototyp benutzt zur Entdeckung von Einbrüchen

- eine Anomaly Detection-Komponente und
- eine Expertensystem-Komponente

Eine Darstellung dieser Komponenten erfolgt in späteren Abschnitten.

3.4.1.1 Unterschiede zwischen dem IDES-Prototypen und älteren Versionen

Der IDES-Prototyp überwacht ein Netzwerk aus *Sun Workstations*. Die Nutzungsmuster bei diesen Workstations unterscheiden sich von den Nutzungsmustern einer zentralisierten Rechen-Ressource. Sun-Benutzer arbeiten meist an mehreren, gleichzeitig geöffneten Fenstern, von denen viele UNIX-Shells laufen lassen können. Im Gegensatz zu der TOPS-20 (eine Produktmarke von *Digital Equipment Corporation*) Umgebung, die frühere IDES-Prototypen überwacht haben, existiert hierbei keine Benutzer-Sitzung, weil die Benutzer UNIX-Shells oder andere Fenster öffnen bzw. schließen können, ohne sich ein- oder auszuloggen, und diese Fenster über längere Zeitperioden (Tage oder Wochen) aktiviert lassen können, ohne sich je auszuloggen. Ein Benutzer hat im Allgemeinen „Superuser-Privilegien" auf seiner eigenen Workstation und hat gewöhnlich keine solchen Privilegien auf anderen Workstations. Benutzer einer vernetzten Sun-Umgebung weisen gewöhnlich mehr Zugriffe, Prozeduraufrufe und Datei-Übertragungen von entfernten Rechnern auf, als die Benutzer von zentralisierten Rechnern. Somit können sich auch die Missbrauchsmuster der Sun-Netzwerk-Benutzer merklich von Benutzern anderer Netzwerkumgebungen unterscheiden. Bei dem vorliegenden IDES-Prototypen wurden aus diesen Gründen zusätzliche Maßregeln für das Verhalten der Sun-Netzwerk-Benutzer entwickelt und Modifikationen an IDES durchgeführt, um die Integration von Daten aus zahlreichen Quellen zu ermöglichen, was die Verfolgung der Spur eines Fernzugriffes auf die Rechner erlaubt.
Auf Grund der durchgeführten Experimente und Tests mit den früheren IDES--Versionen, die auf ein TOPS-20 Zielsystem aufgesetzt wurden, konnten Mängel bei den statistischen Algorithmen entdeckt werden. Dies führte zu einer Verbesserung und Erweiterung der Algorithmen. Als eine weitere Folge der Test-Ergebnisse wurde bei dem IDES-Prototypen eine komplette Rekonstruktion und Reimplementation der statistischen Komponente durchgeführt, um sie mit den neuen Algorithmen in Einklang zu bringen und sie leicht modifizierbar zu machen.
Ein anderer wichtiger Grund für die Rekonstruktion war das Vorhaben, IDES im höchsten Maße modular zu gestallten. Der modulare Entwurf

- erleichterte die Kommunikation unter den IDES-Mitarbeiten,
- reduzierte die Verantwortung eines Programmierers auf eine klar definierte System-Ebene oder System-Komponente,
- trennte die low-level Abhängigkeiten von der high-level Funktionalität von IDES,
- unterstützte ein Grundgerüst für die allmähliche Aufhebung der Abhängigkeiten von einem Datenbank Management System,

- erlaubte die Durchführung erprobte Software Engineering Praktiken.

Der vorliegende IDES-Prototyp ist, im Vergleich zu früheren Versionen, sehr komplex. Er schließt die Fähigkeit der simultanen Überwachung heterogener Zielmaschinen (z.B. UNIX- und IBM-Großrechner), sowie die Fähigkeit der simultanen Überwachung von mehreren homogenen Zielen, ein. Er ist auch in der Lage, verschiedene Audit-Daten-Quellen jeder einzelnen Maschine zu überwachen. Der IDES-Prototyp wurde als ein erweiterbares System entworfen, wodurch das Hinzufügen von zusätzlichen Analyse-Komponenten sehr leicht durchführbar wurde. Er benutzt robustere, statistische Algorithmen, eine Expertensystem-Komponente und eine verfeinerte Benutzer-Schnittstelle.

Frühere IDES-Versionen benutzten das relationale *Datenbank Management System* (DBMS) von ORACLE, das die Verwaltung aller IDES-Informationen übernahm. Die Implementationssprache von IDES war (ist) C. Datenbank-Aktualisierungs- und Profilvergleichs-Operationen wurden in SQL eingebettet (Oracle-Pro*C). IDES bestand aus Prozessen, die über das relationale ORACLE DBMS, welches die IDES-Datenbank bediente, miteinander wechselwirkten. Diese Datenbank enthielt :

- *Audit-Daten :* Diese waren zusammenhängend mit den Audit-Records, die IDES von dem überwachten System erhielt. Sie wurden in einer Tabelle gespeichert und untersucht, um die Aktiv-Daten und die Anomalie-Daten zu aktualisieren;

- *Aktiv-Daten :* Diese beinhalteten die Menge der Aktivitäten eines Benutzers, in Bezug auf eine bestimmte Variable, die gerade gemessen wurde. Eine Tabelle von berücksichtigten Einbrüchen wurde für jede Variable benutzt. Diese Daten wurden periodisch zu Aktualisierungszwecken der Benutzer-Profile, relativ zu einer gegebenen Variablen, eingesetzt.

- *Archiv-Daten :* Dies waren die geprüften Audit-Records, d.h. jene, die bereits für die Profil-Aktualisierung und Anomalie-Analyse berücksichtigt wurden. Sie wurden in einer Tabelle gespeichert und periodisch entfernt.

- *Profil-Daten :* Diese Daten definierten die normalen Verhaltensprofile eines jeden Benutzers oder einer Benutzer-Klasse. Sie wurden in eine Menge von Tabellen gespeichert (eine für die betrachteten Einbrüche, bezogen auf jede Variable).

- *Zeitplan-Daten :* Diese zeigten die Aktualisierungsperiode eines jeden Profils und das Datum der letzten Profil-Aktualisierung an. Sie wurden in einer Menge von Tabellen gespeichert (eine für die Einbrüche für jede gemessene Variable).

- *Anomalie-Daten :* Dies waren die Anomalie-Records, wie sie in Dorothy Denning's ID-Modell beschrieben wurden. Sie waren in einer Menge von Tabellen gespeichert (eine für jeden Anomalie-Typ).

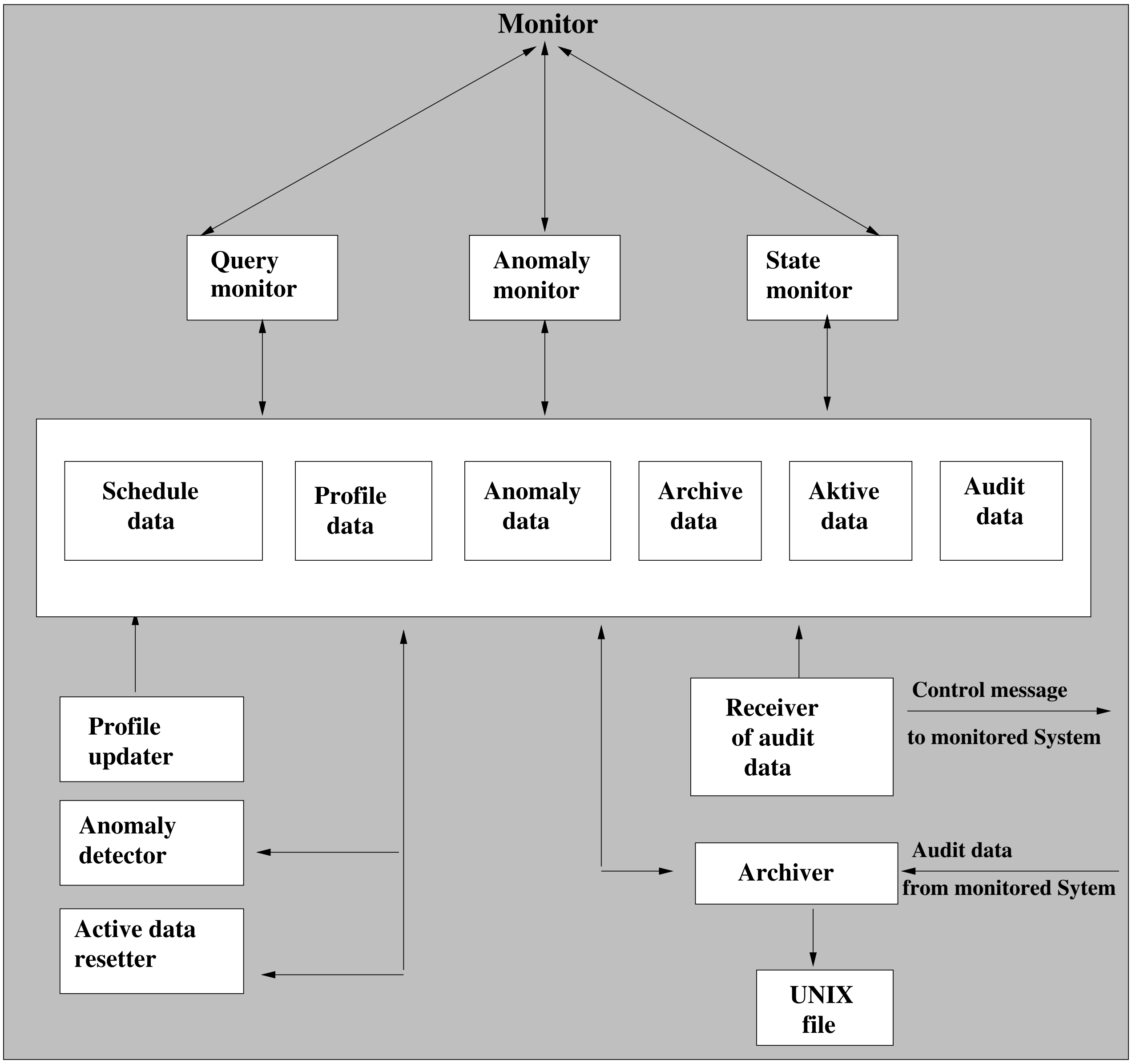

Abbildung 3.7: Struktur von IDES Prototyp [CFGP94]

Die *Prozesse*, die IDES implementierten und miteinander über die IDES-Datenbank kommunizierten, waren folgende :

- *Empfänger :* Der Empfänger (*Receiver*) implementierte das IDES-Protokoll. Er entzifferte, analysierte und validierte die Audit-Records, die von dem System kamen. Er fügte gültige Records in die *Audit-Daten* ein.

- *Anomalie-Detektor :* Der Anomalie-Detektor sondierte einen Record, der sich auf die *Audit-Daten* bezog, und aktualisierte die *Aktiv-Daten*. Wenn das Ergebnis der kontrollierten Messung für einen einzelnen Benutzer die *Annehmbarkeitsschwelle*, angezeigt durch die *Aktiv-Daten*, überstieg, wurden die *Anomalie-Daten* aktualisiert. Wenn ein Audit-Record überprüft wurde, konnte er aus den *Aktiv-Daten* entfernt und in die *Archiv-Daten* aufgenommen werden.

- *Archivierer :* Der Archivierer sicherte periodisch die Daten.

- *Profil-updater :* Dieser Prozess aktualisierte die Profile mit Hilfe der *Aktiv-Daten* am Ende einer Zeitperiode, die sich auf die betrachteten Profile bezog;

- *Aktiv-Daten resetter :* Dieser aktualisierte periodisch die *Aktiv-Daten*. Am Ende einer Zeitperiode, während der die Daten kontrolliert wurden, gingen die in dieser Periode gesammelten Aktivitäten in die globale Aktivität über, die Perioden-Nummer wurde inkrementiert und die Aktivität auf 0 gesetzt.

- *Administrator Interface :* Diese Schnittstelle unterstützte den IDES-Administrator mit einer zur Verfügung gestellten Gesamtbetrachtung des überwachten Systems. Das IDES-Interface, basierend auf einem Window-System, bot :

  - einen *Zustands-Monitor*, welcher den aktuellen Zustand von IDES anzeigte;
  - einen *Anomalie-Monitor*, welcher das abnormale Verhalten, festgelegt in dem System für eine Messung (oder für alle Messungen) und aufgetreten in der Vergangenheit oder Gegenwart, anzeigte;
  - einen *Abfrage-Monitor*, welches dem Administrator, durch Benutzung von vordefinierten Fragen oder SQL-Abfragen, den Zugriff auf die IDES-Datenbank erlaubte.

Bei der Konstruktion des hier beschriebenen IDES-Prototypen, wurde die Benutzung des ORACLE Datenbank Management Systems umgewertet. In Anbetracht des neuen Design-Ansatzes und der zahlreich durchgeführten Bewertungsexperimente, kam man zu dem Schluss, dass zwischen den Anforderungen des neuen Online-IDES-Prozessors und den Fähigkeiten von ORACLE Unvereinbarkeiten bestanden. Man entschied daraufhin, die Benutzung von ORACLE für den Online-IDES-Prozessor einzustellen. Die Hauptbegründung bei der Entscheidung gegen ORACLE für den Online-IDES-Prozessor war die Tatsache, dass ORACLE bei der Wiedergewinnung der Daten nicht schnell genug war. ORACLE besaß sehr viele Mechanismen, die für IDES nicht gebraucht wurden, die aber die Performanz des ID-Systems herabsetzten. Es stellte sich heraus, dass die Benutzung einer UNIX Datei-basierten Implementierung eine angemessene Performanz

unterstützte. Es ergaben sich aber auch Probleme, die durch den Verzicht auf ORACLE entstanden. Eine sehr wichtige Aufgabe, die gelöst werden musste, betraf die simultanen Zugriffe auf die Datenbank. ORACLE erlaubte den gleichzeitigen Zugriff durch Lese- und Schreib-Prozesse. Ursprünglich war geplant, diese Fähigkeit einzusetzen. Als aber entschieden wurde, ORACLE nicht zu benutzen, ergab sich die Frage, wieviel Aufwand nötig war, um einen neuen, eigenen Mechanismus zu implementieren. Durch die Analyse der Zugriffs-Muster wurde entdeckt, dass, im schlechtesten Fall, nur gleichzeitige lese-schreib Zugriffe und nie schreib-schreib Zugriffe auftauchen würden. Diese worst-case Situation würde nur einmal am Tag, während der Profil-Aktualisierungsperiode, auftreten. Die IDES-Entwickler entdeckten auch, dass die simultanen lese-lese Zugriffe sehr selten auftauchten (nur dann, wenn ein Sicherheitsbeauftragter oder ein Datenanalysierer ein Subjekt-Profil kurz überprüfen wollte). Basierend auf den obigen Ergebnissen wurde schließlich entschieden, einen Caching-Prozess, gekoppelt mit einem einfachen Cache-Update-Protokoll, für die Vermittlung der Zugriffe auf die Benutzer-Profil-Daten einzusetzen.
Wie schon erwähnt wurde, in den IDES-Prototypen auch ein Expertensystem-Werkzeug integriert und eine Experten-Regel-Basis entwickelt.

Das Werkzeug stellt eine Regel-Spezifikationssprache, einen Übersetzer, der den aktuellen Expertensystem-Code produziert, und run-time Bibliotheken, zur Unterstützung des Expertensystems, zur Verfügung.

Die Regel-Basis codiert bekannte Einbruchsszenarien und bekannte System-Verwundbarkeiten. Eine Regel ist analog zu einer Software-Prozedur oder einem Unterprogramm zu sehen und muss getestet werden, um sicherzustellen, dass sie sich unter verschiedenen Arten von Eingabe-Daten wie erwartet verhält. Eine *debugging*-Einrichtung half den Entwicklern der Regel-Basis, die in Frage kommendeRegeln zu testen. Regel-Basis wurde in Zusammenarbeit mit der FBI entwickelt und war an ihren Anforderungen angepasst. Zusätzlich wurden auch spezielle Regeln für Sun UNIX entwickelt.
Nachdem die Expertensystem-Komponente implementiert war, wurde es für die Analyse von FBI-Daten eingesetzt.

Für den IDES-Prototypen wurde eine neue Kern-Technologie entwickelt, die als GLU (Granular Lucid - Granulare Übersicht) bezeichnet wurde und wodurch IDES leicht skalierbar für viele Prozessoren wurde. Vom Standpunkt des Entwurfs aus gesehen, erlaubt GLU die Ausnutzung der natürlichen Parallelismen, die bei der Verarbeitung der Audit-Daten durch IDES gegeben sind, und unterstützt ein hochmodulares Design. GLU verleiht IDES eine erhöhte Fehlertoleranz, weil es automatisch zu einer anderen Prozessor-Aufgabe wechselt, die auf dem gestörten bzw. ausgefallenen Prozessor aktiv ist.

Nach der kurzen Darstellung der Erweiterungen und Erneuerungen bei dem IDES-Prototypen, gegenüber den Vorgänger Versionen, wird in den folgenden Abschnitten das Design Modell von IDES beschreiben.

3.4.2 IDES Design Modell

Die Orginal-Version von IDES wurde für die Unterstützung eines einzelnen Zielsystems (target host) entworfen und entwickelt. Für den hier beschriebenen IDES-Prototypen wurde die Architektur der originalen Version neu entworfen, so dass sie auf eine Menge von heterogenen Zielsystemen angewandt werden konnte und darüber hinaus eine Plattform für experimentelle Intrusion Detetection-Techniken und verschiedene Implementierungssmethoden unterstützen konnte. Das neue Design-Modell ist in vier Teile aufgegliedert:

- Zielsystem-Domain (target system domain)
- Realm-Schnittstelle (realm interface)
- Arbeitsmaschine (processing engine)
- Benutzer-Schnittstelle (User interface)

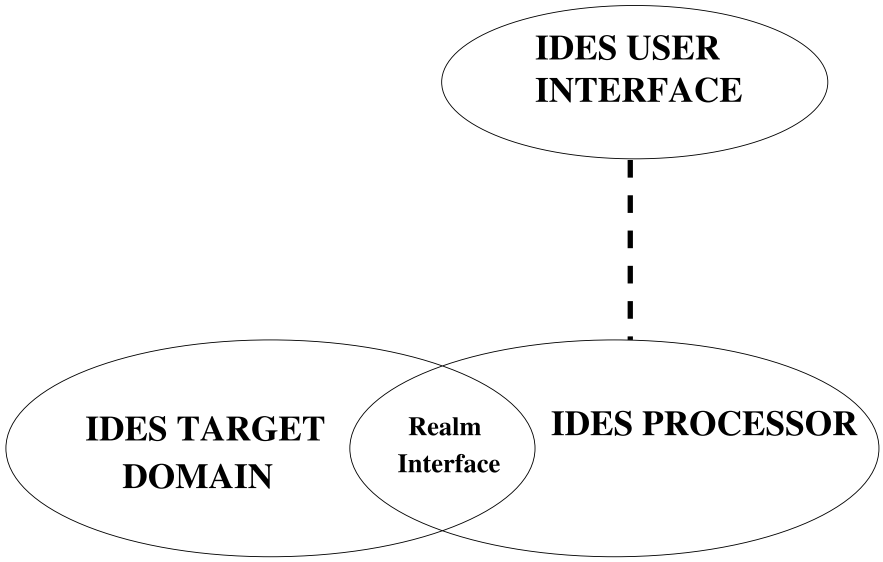

Abbildung 3.8: IDES Design Modell [LTG+92]

3.4.2.1 Zielsystem Domain

Die Zielsystem-Domain besteht aus einer Menge von realms, die auf anomale Aktivitäten hin überwacht werden. Ein *realm* ist eine Gruppe von ähnlichen Zielmaschinen, d.h. in diesem Fall, dass die Audit-Trails, die die Maschinen produzieren, das gleiche Format haben. Day bei SRI für den IDES-Prototypen benutzte Netzwerk von Sun Workstations bestand z.B. aus einem realm Typen.
In dem Zielsystem-Domain werden die Audit-Daten des überwachten Systems gesammelt. Wegen der Überwachung von mehreren Hosts kann es mehr als einen zentralisierten Audit-Daten Sammlungspunkt geben, und der Vergleich kann eine Menge von Zielsystemen und realms betreffen. Um die Zielsystem-Audit-Trails bereitzustellen, muss jede realm-Schnittstelle die Authentifikation der Audit-Daten unterstützen um die Quelle zu überprüfen aus der die Audit-Daten kommen.

3.4.2.2 Realm-Schnittstelle

Die realm-Schnittstelle verbindet die IDES-Domain und die analytische Komponente von IDES. Sie wandelt das realm-spezifische Audit-Datenformat in das IDES Audit-Datenformat um. Die Schnittstelle besteht aus zwei Teilen

1. *Realm Server :* Der Teil, der sich innerhalb des Zielsystems befindet
2. *Realm Client :* Der Teil, der lokal zu dem IDES Prozessor ist

Die beiden Komponenten sammeln die Zielsystem-Aktivitäten für die analytische Maschine von IDES und speichern sie für spätere Benutzung ab. Die realm-spezifischen Daten werden herausgefiltert und für spätere Benutzung abgespeichert. Wenn verschiedene realm-Typen vorhanden sind, haben diese ihre eigenen Audit-Trail-Formate und für jeden realm-Typ existiert eine andere realm-Schnittstelle.

Der *realm Server* konvertiert die Audit-Records in IDES Audit-Record-Formate um. Durch die Benutzung eines einfachen Datentransfer-Protokolls errichtet der Server eine Verbindung mit dem *realm Client*, um die Audit-Informationen des Ziel-Systems über das Netzwerk zu transportieren.
Der *realm Client* speichert die Audit-Daten und macht sie für die Ereignis-Subsysteme verfügbar. Es muss bemerkt werden, dass keine klare Trennung von funktionellen Verantwortlichkeiten zwischen realm Server und realm Client existiert. Die Implementierung kann, in Abhängigkeit von dem Zielsystem, variieren.

3.4.2.3 IDES Prozessor

Der IDES Prozessor ist für die Analyse der Informationen verantwortlich, die von dem IDES Domain geliefert werden. Die Audit-Records werden an verschiedene analytische Maschinen weitergereicht, die als *Ereignis Subsysteme* (event subsystem) bezeichnet werden. Dabei verarbeiten alle Ereignis-Subsysteme die Ereignisse auf der *pro-Audit-Record* basis. IDES implementiert zwei Anomalieerkennung-Ereignis-Subsysteme

1. basiert auf statistischen Methoden
2. regelbasiertes System (Expertensystem)

Die statistische Komponente vergleicht die normalisierten Audit-Daten mit den Werten einer Datenbank. Bei zu großer Abweichung wird ein Alarm ausgelöst Das regelbasierte System arbeitet ähnlich, aber anstatt statistische Abweichungen zu ermitteln, vergleicht es die Audit-Daten mit den Prämissen von Produktionsregeln. Produktionregeln legen fest, wann ein Alarm auszulösen ist. Im Alarmfall wird der SSO informiert. Eine ausführliche Beschreibung dieser beiden Subsysteme erfolgt im Abschnitt 3.6.6, weil die von EMERALD implementierten Subsysteme weitgehend denen von IDES entsprechen. Im Abschnitt 3.4.3 wird lediglich der Entwurf der statistischen Komponente beschrieben.

3.4.2.4 Benutzer-Schnittstelle

Die Benutzer-Schnittstelle erlaubt Benutzern alle erstellten und von IDES verarbeiteten Informationen einzusehen. Benutzer können Komponenten-Aktivitäten und Komponenten-Zustände beobachten. Die Benutzer-Schnittstelle unterscheidet drei Typen von Benutzern :

- System Security Officer (SSO)
- Datenanalysierer
- Systemadministrator

Abbildung 3.9 zeigt einen funktionsgemäßen Aufbau von IDES in einer vereinfachten Darstellung.

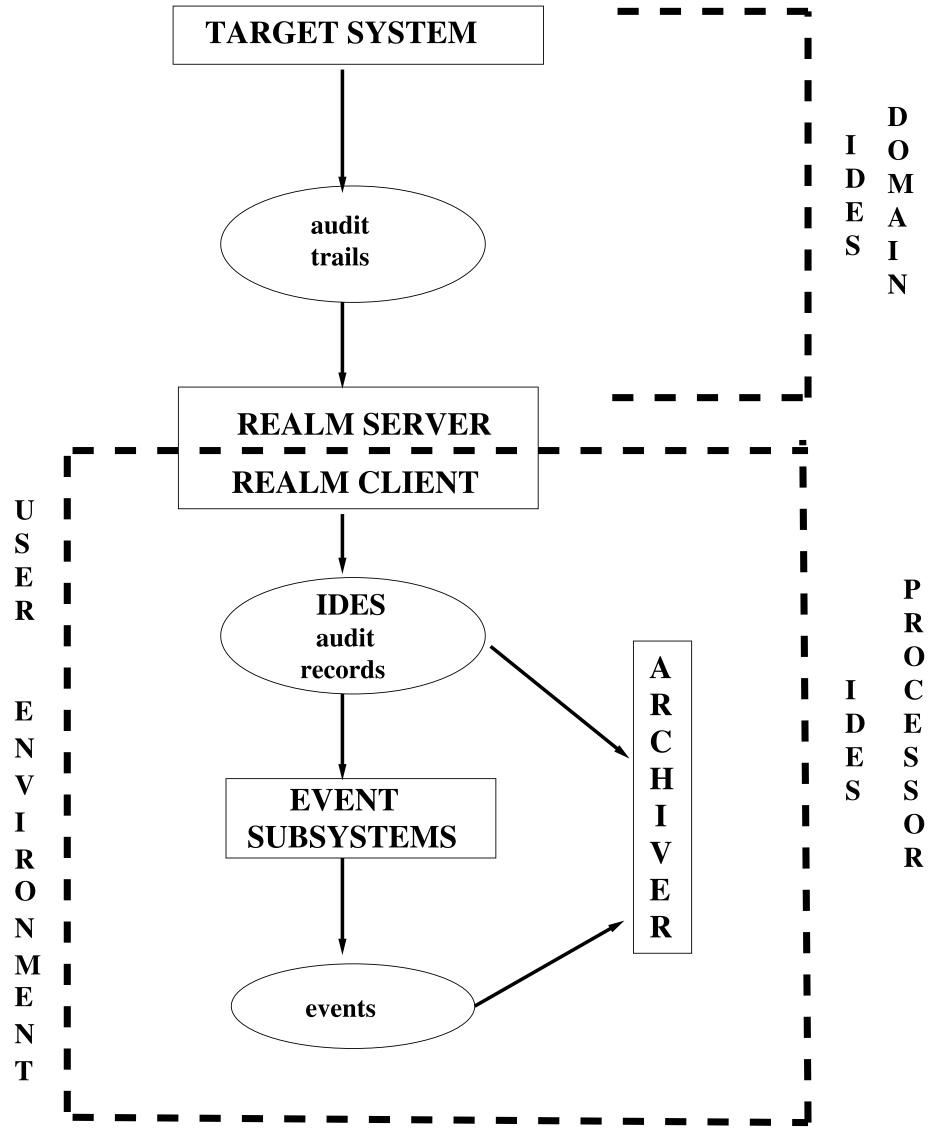

Abbildung 3.9: Funktionsgemäßes Diagramm von IDES [LTG+92]

3.4.3 Entwurf der statistischen Komponente

3.4.3.1 Die funktionelle Architektur

Die statistische Komponente von IDES wird von den ankommenden Audit-Records gelenkt. Durch Prüfung der Audit-Datensätze, die von dem Zielsystem ankommen, kann IDES, mit Hilfe verschiedener statistischer Algorithmen bestimmen, ob die beobachtete Aktivität, in Bezug auf die Profile, anomal ist. Es wird anomales Verhalten angezeigt, falls solch ein Verhalten von dem normalen Verhalten des Subjektes abweicht. Wenn eine Anomalie entdeckt wird, wird ein Anomalie-

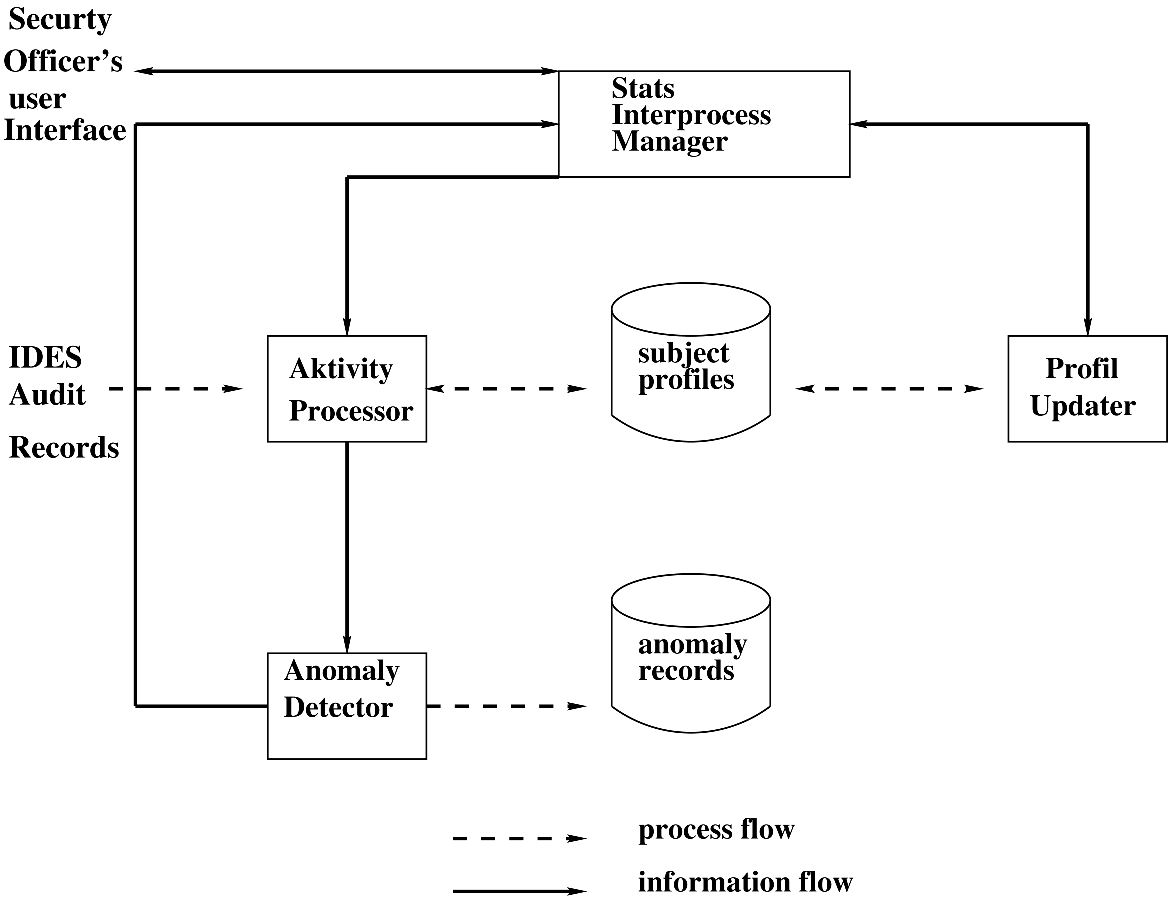

Abbildung 3.10: Statistische Anomalieerkennung-Verarbeitungseinheit [LTG+92]

Record generiert und dem SSO präsentiert. Dieser Anomalie-Record wird für weitere Analysen abgespeichert. Abb. 3.10 zeigt die Hauptbestandteile der statistischen Komponente und ihre Beziehungen untereinander. In folgenden Unterabschnitten wird die Funktionalität jedes dieser Module im Detail beschrieben.

Statistischer Interprozess Manager :
Dieses Modul ist der Treiber der gesamten statistischen Komponente. Alle Prozesse (sowohl interne, als auch externe) kommunizieren mit diesem Modul und es leitet die Prozessflüsse entsprechend weiter. Wenn die statistische Komponente von IDES gestartet wird, initialisiert der Interprozess Manager erst die zugehörigen statistischen Parameter (dadurch wird festgelegt welche Merkmale aktiviertwerden sollen, z.B. welcher Alterungsfaktor benutzt werden soll oder). Solche Informationen sind in einer Konfigurationsdatei gespeichert, die von autorisierten Benutzern, entweder vor oder während der Ausführung der statistischen Komponente, modifiziert werden kann.

Bevor der Interprozess Manager jeden Audit-Record einliest, „horcht“ er auf Meldungen, die von anderen IDES-Prozessen kommen, beispielsweise von der Expertensystem-Komponente,von dem Profil-Aktualisierer oder von einem der Benutzer-Schnittstellen-Programme. Bei Ankunft einer solchen Meldung ruft der Interprozess Manager den zugehörigen Prozess auf, der mit dieser Meldung im Zusammenhang steht, und setzt die Verarbeitung der nächsten IDES Audit-Records fort. Einige Meldungen enthalten eine Anforderung zur Auflistung des aktuellen Profil-cache‘s (profile cache) eines bestimmten Subjektes (gewöhnlich zur Aktualisierung des Profils, oder um die letzte statistische Datensammlung auszugeben), oder eine Anzeige von einem anderen Prozess (höchstwahrscheinlich der Profil-Aktualisierer), um in den Profil-Speicher des Anomalie-Detektors die neuesten Daten aufzunehmen.
Zusätzlich sendet der Interprozess Manager erforderliche Meldungen zu anderen Prozessen. Die statistische Komponente sendet laufend den Testwert T^2 (vgl. Abschnitt 3.6.6.3) für jeden Audit-Record an die Benutzer-Schnittstelle des Sicherheitsbeauftragten, sowie eine Alarm-Meldung, wenn das Ergebnis der Auswertung einen gegebenen Grenzwert für dieses Subjekt überschreitet.

Aktivitäts-Record-Prozessor :
Dieses Modul erhält IDES Audit-Datensätze von der *realm* Schnittstelle und transformiert die Daten in eine Datenstruktur, die als *Aktivitätsvektor* bezeichnet wird. Der Aktivitätsvektor besteht aus N Elementen, wobei N die Anzahl der Merkmale (aktiv und inaktiv) angibt, und beobachtete Werte für jedes Merkmal beinhalted, das von einem Audit-Datensatz berührt wurde. Es muss bemerkt werden, dass nicht alle Merkmale stets von jedem Audit-Datensatz berührt werden. Im Allgemeinen wird dies durch die Konstruktion der Auditing-Mechanismen des Zielsystems festgelegt. Wenn ein Merkmal, bzw. ein Merkmalraum, inaktiv ist, wird ein *null*-Wert in die Speicherstelle dieses Merkmals in dem Aktivitätsvektor eingefügt. Der Aktivitätsvektor wird, zwecks statistischer Berechnungen, an den Anomalie-Detektor weitergegeben.

Anomalie-Detektor :
Der Anomalie-Detektor ist das Herz der statistischen Maschine. Dieses Modul nimmt die Aktivitätsdaten, die von dem Aktivitätsprozessor produziert wurden, entgegen und ruft die Routinen auf, die die statistischen Algorithmen unterstützen, die bei der Anomalieentdeckung benutzt werden.
Der Anomalie-Detektor aktualisiert kontinuierlich die täglichen Aktivitätsanhäufungen eines Subjektes mit jedem ankommenden Audit-Record, und vergleicht diese Werte mit den historischen Profilen, um die Abweichung von erwartetem (normalem) Verhalten festzulegen. Nach der Durchführung statistischer Tests, kann ein Anomalie-Record generiert werden, falls ein beobachteter Aktivitätsvektor, in Bezug auf das historische Profil des Subjektes, ein unübliches Verhal-

ten anzeigt. Wenn dieser Fall eintrifft, wird an den Interprozess Manager eine Meldung weitergereicht, damit dieser die Meldung an die zuständigen externen Prozesse sendet, z.B. an die Benutzer-Schnittstelle des Sicherheitsbeauftragten.

Profil-Aktualisierer :
Der Profil-Aktualisierer wird im Allgemeinen einmal am Tag aufgerufen. Er nimmt die angesammelten, täglichen Aktivitätsinformationen eines einzelnen Subjektes entgegen und legt sie in das zugehörige, historische Profil ab, nachdem einige Alterungsfaktoren auf sie angewandt wurden. Das auf diese Weise aktualisierte Profil wird mit der Aktivität des nächsten Tages (für das selbe Subjekt) verglichen.
Der Profil-Aktualisierer kann in zwei Modi laufen :

- *Parallel :* Der Profil-Aktualisierer kann als ein unabhängiger Prozess laufen, der in einer vorherbestimmten Tageszeit „aufwacht" und simultan die Profile des Tages verarbeitet, während das Anomalie-Detektor-Modul weiterhin Audit-Datensätze von der realm-Schnittstelle empfängt. Jeder der eintreffenden Audit-Datensätze eines Subjektes, dessen Profil gerade aktualisiert wird, wird zeitweilig beiseite gelegt, bis der Aktualisierungsprozess für dieses Subjekt beendet ist.

- *Sequentiell :* Der Aktualisierer kann auch von dem Interprozess Manager aufgerufen werden, der die Zeitstempel auf jedem Audit-Datensatz überprüft, um festzustellen, ob ein neuer Tag angebrochen ist oder nicht. Wenn ein neuer Tag begonnen hat, wird der Anomalie-Detektor zeitweilig angehalten, während der Profil-Aktualisierungsprozess für jedes aktive Subjekt aufgerufen wird.

Wenn in irgendeines der oben dargestellten Modi gearbeitet wird,unterhält der Aktualisierungsprozess ein einfaches Kommunikationsprotokoll mit dem Interprozess Manager aufrecht, um die Verfälschung der Subjekt-Profile während der Laufphase des Anomalie-Detektors zu vermeiden. Selbst wenn der Aktualisierer im sequentiellen Modus läuft, ist es für den Anomalie-Detektor wichtig zu wissen, wann ein Subjekt-Profil modifiziert wurde, weil bestimmte Strukturen (vorwiegend Zähler) auf *null* gesetzt werden müssen, nach dem die Tagesaktivität in das historische Profil des Subjektes aufgenommen wurde. Der Anomalie-Detektor legt die Profile in einem Zwischenspeicher ab und infolgedessen muss dieser Zwischenspeicher nach einer Profil-Aktualisierung aufgefrischt werden.
Der Profil-Aktualisierer kann auch manuell aufgerufen werden, obwohl diese Methode sehr selten benutzt wird (außer in Fällen, wo eines der oben erwähnten Modi den Aktualisierer nicht korrekt startet, oder wenn der Aktualisierer für Tests, bzw. Experimente, gebraucht wird).

3.5 MIDAS - The Multics Intrusion Detection and Alerting System

MIDAS ist ein Expertensystem, das für die Multics-basierte Netzwerkumgebung des US National Computer Security Center, 1988, entwickelt wurde. Das Design folgt den Richtlinien von IDES. Bei gegebenen, statistischen, Benutzer- und Systemprofilen, werden Abweichungsformen vom normalen Verhalten entdeckt. Schlussfolgerungsangriffe wurden bei der Entwicklung nicht in Erwägung gezogen. Trojanische Pferde und Viren gehören zu einer einzigen Kategorie, während *denial of service* und *leakage* (Lecks) mit Missbräuchen zusammengefasst wurden.

3.5.1 Regeln

Die Regeln (Heuristiken) in der Regel-Basis können entsprechend dem Typ der verwendeten Regeln oder des überwachten Bereichs charakterisiert werden. Es gibt folgende Heuristiktypen :

1. *Unmittelbarer Angriff (immediate attack) :* Dieser schliesst eine kleine Menge an Daten für die Analyse mit ein und benutzt keine statistischen Informationen. Die Regeln entdecken Auditlog-Eingänge, die in sich selbst anomal sind (z.B. in der Isolation von anderen Informationen).

2. *Benutzeranomalie (user anomaly) :* Diese verwendet statistische Profile, um anomales Verhalten zu entdecken. Sie entsprichen der Intuition des SSOs (System Security Officer): „Etwas Anomales hat sich ereignet“ (z.B. loggt sich ein Benutzer in einer unüblichen login-Zeit ein, deshalb wird in solch einem Fall ein Anomalie-Record generiert).

3. *Systemzustand (system state):* Diese Regeln sind analog zu den Anomalie Regeln, definieren aber zusätzlich, was normal für das ganze System ist. sie sind unter anderem zuständig für die Entdeckung großer Anzahlen von systemweiten Login-Misserfolgen, was auf einen möglichen Systemdurchdringungsversuch hindeuten würde.

Folgende Bereiche werden bewacht :

1. *Einbruchsversuche :* Dieser Bereich konzentriert sich auf Login-Fehler. Deswegen überwachen Regeln in dieser Kategorie ausschließlich die Passwort-Misserfolge auf einem Systemaccount.

2. *Maskerade :* Die Entdeckung basiert auf der Annahme, dass die Parameter der normalen Benutzer-System Wechselwirkungen nützlich für die Abtastung der diesem Benutzer zugeschriebenen Aktivitäten sind, aber von den statistischen Normen des Benutzers abweichen. Beispiele von Faktoren,

die die Benutzer charakterisieren, sind : Verbindungsursprung, login-Zeit, Ressourcen-Benutzung.

3. *Durchdringung :* Diese schließt die Entdeckung von Verstößen gegen die Sicherheitsmechanismen ein. Es werden Anomalie- und Systemzustandsheuristiken benutzt, mit dem Ziel, Zugriffe oder Zugriffsversuche auf die sensitiven Programme/Daten zu verhindern.

4. *Missbrauch :* Anomale Ressourcen-Benutzung offenbart potentielle Maskeraden oder privilegierte Benutzer, die unerwünschte Aktivitäten ausführen. Inaktivität wird nicht als ein Angriff gemeldet. Dieser Bereich schließt die verdeckten Kanal-Angriffe mit ein : In Multics 11.0 sind alle großen, verdeckten Kanal-Quellen eliminiert worden, während mittelgroße und kleine, verdeckte Kanäle durch das Betriebssystem und ebenso durch MIDAS überwacht werden.

5. *Trojanische Pferde und Viren :* Schlüsselfaktoren sind Zugriffsverstöße gegen systemsensitive Objekte, und die Ausführungsstatistik, die die Normen verletzt, die für bestimmte Befehle aufgestellt wurden. Zugriffsverletzungen auf sensitive Objekte kennzeichnen eine mögliche Einschleusung von Viren. Durch die Überwachung von Ausführungsstatistiken wird versucht, deren Präsenz aufzudecken.

Die meisten MIDAS-Regeln sind „sensorische Regeln“, die anomale Aktivitäten entdecken und Schlussfolgerungen in die Wissens-Basis einsetzen, die das vermutete Problem repräsentiert. Eine Menge von „sekundären Regeln“ operiert nur auf dem output von sensorischen Regeln.

3.5.2 MIDAS Verfahren

MIDAS überwacht beständig Benutzeraktivitäten und den Systemzustand. Der SSO kann entscheiden, ob er direkt an den Warnmeldungen arbeiten oder die aufgezeichneten Auditdaten ausführlich, durch Benutzung des MIDAS-Interface, analysieren will.

MIDAS besteht aus folgenden Teilen :

- Ein Befehlsüberwacher, der Daten abfängt, die aus Befehlsausführungen resultieren und nicht durch Multics überprüft wurden,
- Ein Präprozessor, der die Auditlog-Einträge von dem Dockmaster in eine vorschriftsmäßige Form bringt (normalisiert),
- Ein Netzwerk-Interface-Dämon,
- Eine statistische Datenbank des Systems und Benutzerstatistiken, die aufgezeichnet wurden,

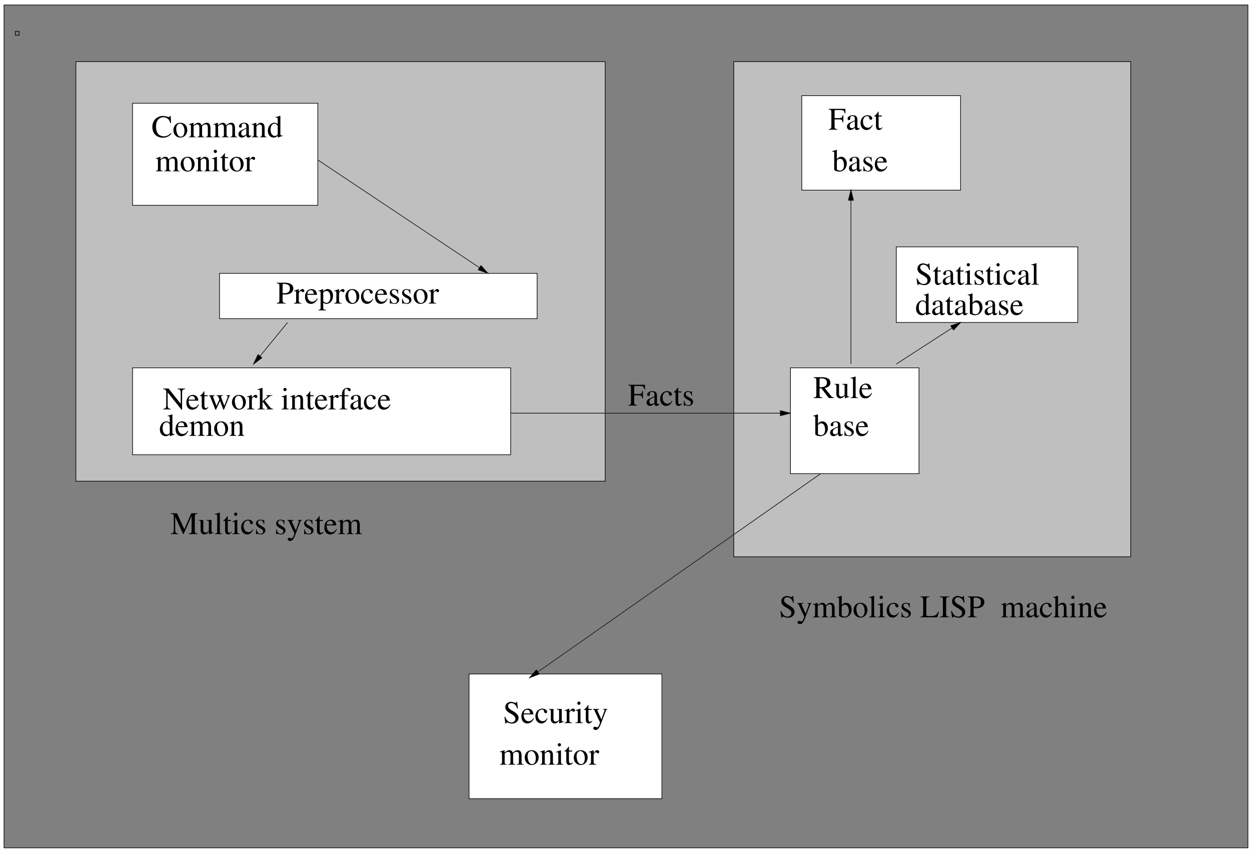

Abbildung 3.11: MIDAS Architektur [CFGP94]

- Eine Wissens-Basis, bestehend aus einer Fakten- und einer Regel-Basis,
- Ein Benutzer-Interface für den SSO.

Nach der Erzeugung eines Audit-Datensatzes, filtert der Präprozessor alle irrelevanten Daten heraus und baut daraus ein Faktum, welches über den Netzwerk-Schnittstellen-Dämon an das Expertensystem weitergereicht wird. Das Faktum wird in die Faktenbasis des Expertensystems eingefügt; dies verursacht eine Bindung zwischen dem Faktum und allen passenden Regeln der Regel-Basis. Jede dieser Regeln feuert nun. Abhängig von dem Faktum erzeugt das Feuern eine Kette von Regeln und am Ende ein Ergebnis (von Warnungen bis zum Abbruch der entsprechenden Aktion). MIDAS' Statistik zeichnet die überwachten Aktivitten auf: Der Vergleich zwischen vergangenen und laufenden Aktionen entscheidet, ob die aktuelle Aktivität in der Norm liegt oder nicht. Benutzerstatistiken werden aktualisiert, wenn sich das Verhalten ändert. Session-Statistiken werden von MIDAS für jeden Benutzer unterhalten. Diese beinhalten einige Werte, die als Schwellwerte für alle überwachten Aktivitäten eines jeden Benutzers gelten.

Ein Beispiel, das in Sebring *et al.* (1988) aufgezeigt wurde, verdeutlicht die obige Ausführung : Wenn die Statistik eines Benutzers anzeigt, dass er während seiner 350 Sitzungen im Durchschnitt 20 Fehler ausgelöst hat, mit einer Standardabweichung von 20, dann wird der Schwellenwert von 58 in das Sitzungsstatistik-Profil des Benutzers gespeichert. 58 wird die Obergrenze für seine normale Aktivität.

Wenn vorverarbeitete Audit-Daten gegeben sind, erhält man Daten der folgenden Form :

$< Subjekt > < Objekt > < Aktion > < Ausnahme > < Zeitstempel >$

Eine *Benutzerscheibe* (user pane) wird am Interface, zusammengesetzt aus Login-Zeit,Benutzer Identifikation (userid), Projekt und einer Marke (tag), für alle Prozesse in dem überwachten System unterhalten. Ein „?“ Flag zeigt den verdächtigten Benutzer an, wobei der Verdacht als Ergebnis einiger, vom Benutzer ausgelösten, Regelkombinationen oder durch unabhängige, vom Operator des Dockmaster Netzwerkes ausgeführten Beobachtungen generiert wird. Ein anderer Flag (bezeichnet durch „M“) zeigt an, dass eine Benutzersitzung streng überwacht wird. Die Audit-Daten widerspiegeln sich in der MIDAS *Warnungsscheibe* (warning pane), welche die Warnmeldunge und MIDAS Resultate, erhalten durch Regelausführungen, anzeigt. Diese Meldungen sind hierarchisch, in Klassen von ähnlichen Meldungen, gruppiert. Dem MIDAS-Operator steht eine *Kommandoscheibe* (command pane) zur Verfügung, die ihm die Modifizierung der Systemparameter, oder die Generierung verschiedener Displays erlaubt.
Anfänglich wurde MIDAS auf 105 Dockmaster-Benutzern getestet (ca. 11% der gesamten Benutzerpopulation). Bei den simulierten Angriffen, wurden nahezu alle anomalen Aktivitäten gemeldet. Zusätzlich wurde eine Anzahl von unvorhergesehenen Anomalien entdeckt und die Systemüberwachung erhöht. MIDAS konnte schnell arbeiten, weil:

- die Regeln kompiliert sind,
- die Regeln verallgemeinert (wann immer es möglich ist) sind
- viele Analysefunktionen sich im Benutzer-Interface befinden und unter der Kontrolle des Operators schneller ausgelöst werden können, als wenn sie in der Regel-Basis platziert wären und automatisch durch zusammenpassende Faktum-Muster ausgelöst werden müssten.

3.6 ID-Systeme der neueren Generation

3.6.1 NIDES

Mit den gereiften Analyse Methoden, die unter IDES entwickelt wurden, begann man beim SRI den früheren IDES-Prototypen zu erweitern, zu optimieren und umzugestalten. Dieses System wurde *Next Generation Intrusion Detection Expert System* (NIDES) genannt. Teresa Lunt und R.Jagannathan betrieben einen umfangreichen Aufwand, um die früheren IDES Forschungsergebnisse und Protoypen zu einer produktionsreifen Werkzeug-Sammlung zu reorganisieren und zu konsolidieren. Erwähnenswert ist, dass NIDES verteilte Audit-Sammlungen und Zusammenführungs-Mechanismen vereinigt, um den Bedarf von *multi-host* Intrusion Detection-Anwendungen zu berücksichtigen. Es enthält auch signifikante Erweiterungen zu den statistischen Analyse-Algorithmen und dem regelbasierten Expertensystem, und führt einen X-Window GUI für administrative Kontrolle und Überwachung ein. Im Februar 1993 gab CSL (Computer Science Laboratory, SRI International) die Alpha-Version von NIDES heraus und die Im Mai 1994 wurde die Beta-Version von NIDES vorgestellt. Im November 1994 wurde die Entwicklung von NIDES, mit einer aktualisierten Beta-Version (Beta-update release), abgeschlossen (Stand : 1997).

3.6.2 NIDES Beta-Version

NIDES Beta-Version enthält signifikante Erweiterungen gegenüber IDES und den NIDES Alpha-Versionen :

1. **Eine optimierte Profilspeicherungsstruktur :** Ursprünglich generierte NIDES (und IDES) zwei Dateien für die Subjekt-Profile, die von der statistischen Analyse benutzt wurden. Wenn die Anzahl der Subjekte sehr hoch war, stellte diese Art der Speicherung ein Problem hinsichtlich des verbrauchten Speicherplatzes dar. NIDES Beta-Version bietet eine kompaktere Repräsentation der Profile. NIDES bietet auch die Möglichkeit grosse und nicht mehr benutzte Matrizen zu entfernen. Auf diese Weise können grosse Speicherbereiche eingespart werden.

2. **Echtzeit-Konfiguration der NIDES-Analyse :** NIDES Beta-Version bietet als eine neue Möglichkeit die Konfiguration der meisten Analyse-Prozesse durch den Benutzer. Zielsysteme können z.B. von der Liste der zu überwachenden Systeme gelöscht oder in die Liste hinzugefügt werden (zu jede Zeit während NIDES läuft).

3. **Erweiterte Berichterstattung :** Die Methoden für die Meldung von Anomalien kann von Benutzern konfiguriert werden. Über das Benutzer-Interface können jeder Zeit E-mail-Empfänger hinzugefügt oder gelöscht werden. Die Konfigurierbarkeit der Alarm-Filter durch Benutzer ist ebenfalls eine Erweiterung. Für einige Subjekte kann z.B. eine Echtzeit-Alarmmeldung eingestellt werden, bei anderen, NIDES bekannten Subjekten, können die Alarme gefiltert werden.

4. **Konfiguration der statistischen- und regelbasierten Analyse-Komponente :** Viele Parameter der statistischen Analyse-Komponente sind konfigurierbar : Merkmalraum-Parameter, globale Parameter (z.B. Profil-half-live), Profil-Managementfunktionen (kopieren, ersetzen und löschen von Subjekt-Profilen), Profil-Aktualisierung und Profil-Synchronisation (unter Stapel-Betrieb können die Profil-Aktualisierungszeiten mit den Audit-Daten, die bei dem Stapel-Betrieb benutzt werden, synchronisiert werden).
   Benutzer können die regelbasierte Analyse-Komponente während der Laufzeit von NIDES modifizieren. Es können neue Regeln kompiliert, installiert und dem Echtzeit- oder Stapelanalyse-Prozess hinzugefügt werden. Regeln können auch ein- oder ausgeschaltet werden (während der Laufzeit oder im Stapel-Betrieb).

5. **Erweiterte Zustands-Berichterstattung :** Diese erfolgt in drei Stufen :

   - *System-Status :*NIDES berichtet über den Zustand (AN/AUS), und über die Durchsatzzahlen für die NIDES Analyse und die archivierten Prozesse. Die Angezeigten Zahlen umfassen auch die gezählten Audit-Records und die Alarme seit der Änderung eines Prozess-Zustandes.
   - *Zielsystem-Status :*NIDES berichtet über die Konfiguration (AN/AUS) und den Zustand jedes bekannten Zielsystems.
   - *Experiment-Status :*Für jedes aktive Stapel-Analyse-Prozess wird periodisch die Anzahl der verarbeiteten Audit-Records und der generierten Alarme angegeben (etwa alle 10 Sekunden).

6. **Daten-Management Einrichtung :** Wurde hinzugefügt, um die Archivierung und die Wiederherstellung der Audit- und Ergebnisdaten-Records zu unterstützen.

7. **Erweiterte Regel-Basis :** Die Anzahl der alarmgenerierenden Regeln wurde von 21 auf 39 erhöht. Es wurden Regeln für die Beobachtung verdächtige FTP/TFTP-Benutzungen, Fern-Ausführung und -Kürzung der Log-Datein hinzugefügt.

3.6.2.1 Erweiterungen der aktuallisierten Beta-Version

Die aktuallisierte Beta-Version hat folgende Erweiterungen :

1. **Perl Script *agen*-Dienst :** Frühere NIDES Versionen basierten auf *agen*-Prozessen, die in C geschrieben und für die Zielsystem-Architektur *Sun SPARC* kompiliert wurden. Um verschiedene Plattformen überwachen zu können, wurde ein *agen*-Prozess in Perl geschrieben (zu agen siehe 3.6.3.1).

2. **UNIX-*agen* Erweiterung :** Die SUN-Version von *agen* wurde modifiziert, um herausfinden zu können, ob das Ethernetz des Hosts in einer undefinierbaren (promiscuous) Modus ist, was auf die Präsenz eines Netzwerk-Sniffers hindeuten würde, der versucht Benutzer-Passwörter oder andre sensitive Daten abzufangen.

3. **Aktualisierte Regel-Basis und *event*-Faktenschablone :** Die regelbasierte Analyse-Komponente wurde so erweitert, dass sie die von Benutzern entwickelten, und in Perl geschriebenen, *agens* unterstützt. Die *event*-Faktenschablone wird dazu eingesetzt, NIDES-Audit-Records zu speichern, die aktuallisiert und erweitert wurden, um alle verfügbaren Datenfelder in einem Audit-Record aufnehmen zu können. Frühere Versionen unterstützten nur eine Teilmenge der Audit-Record-Felder, die bei NIDES verfügbar waren. Das waren die Felder, die unter SunOS Auditing-Systemen am häufigsten gebraucht wurden. Durch die Erweiterung haben Benutzer eine grössere Flexibilität bei der Benutzung der Analyse-Komponente und des zugrundeliegenden Audit-Record-Formats bekommen.

4. **Erweiterte Audit-Record Codes :** Durch die Benutzung der neuen, in Perl Script geschriebenen *agen* konnten die Audit-Record-Aktionscodes und die Quellcodes erweitert werden. Audit-Record-Aktionscodes zeigen den Aktionstyp an, der in dem Audit-Record repräsentiert wird. Audit-Quellcodes zeigen die Quelle der Audit-Daten an.

5. **Aktuallisierte Installationsprozeduren und Dokumentation :** Die Dokumentationen und Installationsprozeduren wurden modifiziert und den neuen gegebenheiten angepasst.

3.6.3 NIDES - Architektur

Alle NIDES-Versionen folgen einem Client/Server-Modell. Jede NIDES-Komponente hat ein modulares Design, das eine wohldefinierte Schnittstelle zu anderen Komponenten zu Verfügung stellt, während es die eigenen Algorithmen und spezifischen Verarbeitungsanforderungen vor andere Komponenten vesteckt hält.

Die NIDES Analyse läuft auf einer unbewachten Host, das ist der sogenannter *NIDES host*. *NIDES host* überwacht die Benutzung einer Reihe von Rechnern, die duch ein Ethernet-Netzwerk miteinander verbunden sind. Diese Zielsysteme (target hosts) übermitteln ihre Daten zu NIDES host. Alle Interaktionen mit NIDES werden über NIDES host ausgeführt (mittels Benutzerschnittstelle). Verschiedene Hosts können die Rolle der NIDES host zu verschiedenen Zeiten übernehmen.

In den unteren Abschnitten werden die Haupt-Komponenten von NIDES beschrieben, die durch entfernte Prozedur- (RPCs) oder Bibliotheksaufrufe miteinander kommunizieren. Einige diese Komponenten wurden in Abschnitt 3.4.2 behandelt andere werden neu beschrieben. Zu bemerken ist, dass die agen und arpool Komponenten auch Bestandteil der letzteren IDES-Prototypen waren aber erst hier erwähnt werden.

3.6.3.1 Agend, Agen und Arpool

Agend (Audit Record Generator Dämon) Der *agend*-Prozess ist ein Dämon, der konstant auf allen Zielsystemen laufen kann. Das agen-Programm wird beim Hochfahren der Zielsysteme gestartet. Agen läuft als RPC-Server-Prozess und horcht an einer bestimmten Port auf Anfragen zum starten oder beenden des Audit-Daten-Konvertierungsprogramms (agen). Die NIDES Benutzerschnittstelle ist für das Senden der Start- oder Stop-Anfragen an agend verantwortlich.

Agen (Audit Record Generator) Dieser Prozess läuft auf jedem Zielsystem, das aktuell Audit-Daten für die NIDES Analyse zu Verfügung stellt. Das *agen*-Programm konvertiert das Audit-Daten-Format in NIDES Audit-Records und übermittelt diese an das *arpool*. NIDES unterstützt drei Audit-Datentypen :

- SunOS BMS Version 1
- SunOS C2
- Standard UNIX accounting

Perl Script erlaubt aber die Entwicklung von agen-Scripten für andre Datenquellentypen.

Arpool (Audit Record Pool) Der *arpool*-Prozess sammelt Audit-Daten von verschiedenen Zielsystemen und stellt sie der Analyse-Komponente, zur Entdeckung von Anomalien, zur Verfügung.

3.6.3.2 Statistische Analyse

Das statistische Subsystem von NIDES (NIDES/Stats) benutzt einen weiten Bereich von mehrfach veränderlichen, statistischen Merkmalräumen, um das Verhaltensmuster der individuellen Benutzer oder anderer rechenbetonter Einheiten zu profilieren. Die Analyse ist profilbasiert, wobei eine statistische Auswertung jeder Sitzung zugewiesen wird, um darzustellen, inwieweit die gegenwärtig beobachtete Benutzung den errichteten Benutzungs-Mustern des entsprechenden Benutzers entspricht. NIDES/Stats produziert ein separates Benutzungs-Profil für jeden Benutzer oder andere Entitäten, und aktualisiert individuelle Profile, sobald es auf die entsprechenden Audit-Records stößt. Dazu unterhält die statistische Analyse statistische Stamm-Profile für jeden Benutzer und verursacht einen

Alarm, wenn die beobachtete Aktivität von den festgelegten Benutzungs-Mustern für einen Benutzer abweicht. Die Stamm-Daten werden regelmäßig aktualisiert und alte Daten werden bei jeder Profil-Aktualisierung herausgenommen, so dass NIDES adaptiv lernt, was es von jedem Benutzer erwartet. Diese Art der Analyse beabsichtigt, Einbrecher zu entdecken, die sich als legitime Benutzer ausgeben (masquerading). Statistische Analyse kann auch Einbrecher entdecken, die vorher unbekannte Schwachstellen ausnutzen, und die durch andere Mittel schwer zu entdecken sind. Aus der statistischen Analyse können sich auch interessante und unübliche Ereignisse ergeben, die zu sicherheitsrelevanten Entdeckungen, bei Nachforschungen durch den Sicherheitsbeauftragten, führen können.
Statistische Analyse ist anpassungsfähig. Verschiedene Parameter und Schwellwerte können, wenn ihre aktuellen Werte nicht zutreffen, verändert oder angepasst werden, und spezielle Verhaltensaspekte für die, die Statistiken erstellt werden, können ein- oder ausgeschaltet werden.

3.6.3.3 Regelbasierte Analyse

Die regelbasierte Analyse von NIDES benutzt Experten-Regeln, die bekannte Einbruchstypen charakterisieren, um einen Alarm auszulösen, falls die beobachtete Aktivität irgendeiner der verschlüsselten Regeln entspricht. Für diesen Zweck wurde die *Signatur-Analyse-Komponente* unter Benutzung von PBEST entwickelt (siehe 5.2). Diese Art der Analyse beabsichtigt die Entdeckung von Einbrüchen, die bekannte Sicherheitslücken des überwachten Systems ausnutzen, und Eindringlinge, die bestimmte Verhaltensmuster aufzeigen, die als verdächtig bekannt sind oder die bestimmte Stellen der Sicherheitspolitik verletzen. Beobachtete Aktivitäten, die irgendeinem dieser vordefinierten Verhaltensmuster entsprechen, werden „markiert“ (flagged). Auch die Regel-Basis von NIDES ist, wie am Anfang des Abschnittes erwähnt, anpassungsfähig. Es können neue Regeln in das laufende System definiert und kompiliert werden, und bereits vorhandene Regeln können ein- oder ausgeschaltet werden. Obwohl NIDES mit einer begrenzten Regel-Basis, entworfen für das Betriebssystem Sun UNIX, ausgestattet ist, können die Regel-Basen auf einzelne Umgebungen angepasst werden.

3.6.3.4 Der Resolver

NIDES führte eine *Ergebnis-Fusions-Komponente* ein, *resolver* genannt, um seine Antwort-Logik mit den Ergebnissen, die von dem statistischen Anomalie-Entdeckungs-Subsystem und dem PBEST Signatur-Analyse-Werkzeug produziert werden, in Einklang zu bringen.

Der NIDES-Resolver schirmt die Alarme, die von den statistischen Regel-Basis-Komponenten generiert werden, zunächst ab, bevor er den Sicherheitsbeauftragten davon in Kenntnis setzt. Dadurch wird die „Überflutung“ des Sicherheitsbeauftragten mit redundanten Alarmen vermieden. Warnungen können der NIDES-Konsole oder einer Liste von E-mail-Empfängern gemeldet werden. NIDES unterhält auch einige benutzerkonfigurierbare Filter. Es können beispielsweise Warnungsmeldungen für spezielle Benutzer ausgeschaltet werden, wenn bekannt ist, dass diese einige unübliche Aktionen durchführen werden. Somit können viele Fehl-Alarme vermieden werden. Auch wenn die gefilterten Warnungen nicht gemeldet werden, sind sie dennoch protokolliert.

NIDES beinhaltet noch folgende zusätzliche Einrichtungen :

- Eine Archiv-Einrichtung, die Audit-Records speichert, Ergebnisse und Warnungen analysiert, und das „Browsen“ in diesem Archiv erlaubt.
- Eine Test-Einrichtung, die dem Sicherheitsbeauftragten das Experimentieren mit neuen statistischen Parametereinstellungen oder neuen Regel-Basen erlaubt, bevor diese an das laufende NIDES übergeben werden. NIDES-Benutzer können Test-Datensätze aus dem Audit-Record-Archiv für ein spezielles Zeitfenster und eine Menge von Benutzernamen konstruieren. Die zu testenden Regel-Basen und statistischen Parameter können gegen diese Test-Datensätze, parallel zu dem laufenden NIDES getestet werden. Die Testergebnisse werden für den Vergleich archiviert.
- NIDES bietet eine Benutzerschnittstell, geschrieben durch Benutzung der MOTIF toolkit, um unter dem X-Window System arbeiten zu können. Seine Benutzerfreundlichkeit, relativ zu IDES, unterscheidet ihn im Wesentlichen auch von IDES.

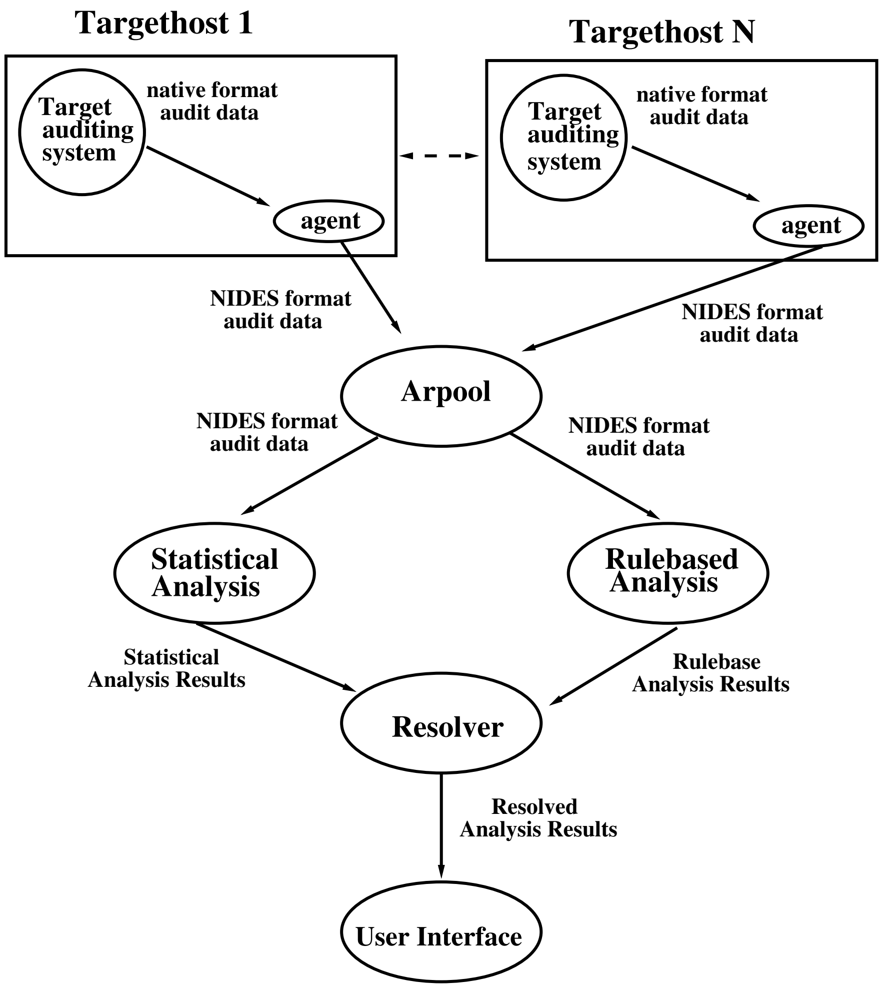

Abbildung 3.12: NIDES Prozess-Graph (Echtzeit)
[DA95]

3.6.4 IDA (Intrusion Detection and Avoidance System)

IDA ist ein Konzept für ein einbruchsentdeckendes und einbruchsvermeidendes System, welches von Prof.Dr. Simone Fischer-Hübner im Rahmen ihrer Dissertation zur Erlangung des Doktorgrades am Fachbereich Informatik der Universität Hamburg, entwickelt und exemplarisch implementiert wurde.
In IDA wird ein formaler Sicherheitskern, der das Referenz-Monitor-Konzept realisiert und dem ein formales Vertraulichkeits- und Integritätsmodell zugrunde liegt, mit einer einbruchsentdeckenden heuristischen, Expertensystem-Komponente gekoppelt.

Den meisten Systemen mit hohen Anforderungen an Sicherheit liegt ein formales Sicherheitsmodell zugrunde. Formale Sicherheitsmodelle, wie das in 1.3.1 kurz

Formaler Sicherheitsansatz	**Heuristischer Sicherheitsansatz**
Formales Sicherheitsmodell	Sicherheitsmodell mit heuristischen Regeln
Sicherheit : Formale Sicherheitseigenschaften	Unsicherheit : Statistisch "abnomales" Verhalten und verdächtige Aktionssequenzen
Beweisbarer, jedoch reduzierter Sicherheitsbegriff	Nicht beweisbarer, dafür jedoch umfassenderSicherheitsbegriff
Formale Spezifikation / Verifikation möglich	keine formale Verifikation möglich

Abbildung 3.13: Die unterschiedlichen Charakteristiken eines formalen und eines heuristischen Sicherheitsansatzes [FH92]

beschriebene Bell-LaPadula-Modell, haben den Nachteil, nur bestimmte formaler Sicherheitseigenschaften einhalten zu können. Im vorangegangenen Abschnitt wurde darauf hingewiesen, dass bei der Entdeckung unbekannter Angriffsmethoden einbruchsentdeckende Systeme an ihre Grenzen stoßen können. Formal verifizierte Systeme bieten gegen noch unbekannte Angriffsmethoden und generell unentscheidbare Sicherheitsprobleme keinen absoluten Schutz. Außerdem lässt sich mit einer Verifikation nach [FH92] lediglich die Einhaltung eines bestimmten Sicherheitsmodells beweisen. In [FH92] heißt es weiterhin :

> „ *Formale Sicherheitsmodelle reduzieren den Begriff* **Sicherheit** *auf die Gewährleistung bestimmter Sicherheitseigenschaften, die in der Regel nicht ausreichen können, um Sicherheit im semantischen Sinne zu garantieren*“

Ein heuristischer Sicherheitsansatz kann die erwähnten Gefahren reduzieren. Ein solches einbruchsentdeckendes System, dass zum größten Teil auf Dorothy Denning's *Intrusion Detection Model* basiert und einen heuristischen Sicherheitsansatz verfolgt, ist IDES, das in 3.4 beschrieben wurde.

Die oben beschriebenen Nachteile der Systeme, die ausschließlich einen formalen Sicherheitsansatz verfolgen, können als eine Motivation für die Entwicklung

von IDA gesehen werden. Prof.Dr. Simone Fischer-Hübner versuchte dabei, in IDA die Vorteile des formalen und des heuristischen Sicherheitsansatzes zu vereinen.
Der Sicherheitskern von IDA verfolgt einen formalen Sicherheitsansatz, er basiert also auf einem formalen Sicherheitsmodell. Die einbruchsentdeckende IDA-Komponente verfolgt eine sogenannten heuristischen Sicherheitsansatz, indem sie ein Sicherheitsmodell mit heuristischen Regeln verwirklicht, das Audit Informationen auswertet und in Abhängigkeit von diesen Ergebnissen direkt einschreiten kann. Somit wird ein aktives und „intelligentes" Auditing-System verwirklicht. Ein Vorteil dieser Methode ist nach [FH92], die Möglichkeit die Auswertung ohne Zeitverzögerung vorzunehmen und somit auch eine schnelle Alarmierung bzw. ein schnelles Eingreifen zu ermöglichen. In Abbildung 3.13 soll eine Gegenüberstellung dieser beiden Sicherheitsansätze gezeigt werden.

3.6.4.1 Die Architektur von IDA

Im Folgenden wird, stark anlehnend an die Ausführungen von Prof.Dr. Simone Fischer-Hübner, ein Überblick über die Architektur von IDA wiedergeben. Eine detaillierte Beschreibung von IDA würde den Rahmen dieser Diplomarbeit sprengen. Aus diesem Grund werden die Ausführungen nur die Grundkonzepte von IDA umfassen.

Die Architektur des IDA-Systemkonzepts realisiert zwei Hauptteile, die miteinander in Verbindung stehen.

1. *Der Referenzmonitor* **RM :**
   Bei jedem Zugriffswunsch eines Subjektes auf ein Objekt überprüft der RM, ob dieser Zugriffswunsch mit der zugrundeeliegenden Sicherheitsstrategie in Einklang steht. Verlangt wird weiterhin, dass der RM Modelle zur Durchsetzung von Vertraulichkeit <u>und</u> Integrität realisiert, d.h. dem RM sollten nicht nur Modelle zur Durchsetzung von Vertraulichkeit zugrunde liegen.
   Als Sicherheitsmodell setzt der RM das Bell-LaPadula- und das Biba-Modell um. Somit wird sowohl eine diskrete und eine mandatorische Vertraulichkeits-Zugriffskontrolle, als auch ein diskreter und ein mandatorischer Integritäts-Zugriffsschutz realisiert.
   Die Architektur des RM entspricht weitgehend der des Betriebssystems Multics. Die Funktionsweise des RM lässt sich, anlehnend an [FH92], folgendermaßen beschreiben :

   Die Eingaben des RM sind Kernfragen, die mit der Kernfunktion die Art des gewünschten Zugriffs festlegen, und das anfragende Subjekt, das referenzierte Objekt und Parameter spezifizieren. Als nächstes wird, mit Hilfe der zur Verfügung stehenden Zugriffskontroll-Informationen, überprüft, ob

die Gewährung des Zugriffswunsches das System wieder in einen „sicheren" Zustand überführen würde (wie es das zugrunde liegende Bell-LaPadula-Modell verlangt).
Aus der Kernfrage und den Ergebnissen dieser Überprüfung wird ein Audit-Record gebildet. Dieser Audit-Record wird vom RM, zur weiteren Sicherheitsüberprüfung der Kernanfrage, an die einbruchsentdeckende Komponente übermittelt. Die einbruchsentdeckende Komponente ihrerseits, übergibt Ergebnisse ihrer Überprüfung wieder an den RM. Dieses Ergebnis kann Einfluss auf die Entscheidung von RM über Gewährung oder Verweigerung des Zugriffswunsches und auf nachfolgende Aktionen haben.

2. *Die einbruchsentdeckende Komponente* **ID :** ID ist eine heuristische Komponente, die statische Anomalien und weitere Sicherheitseinbrüche entdeckt, und durch Kombination mehrerer solcher Indikatoren auf einen möglichen Sicherheitsverstoß schließen kann. Der von dem RM gebildete und dem ID übermittelte Audit-Record wird von dem **anomalie- und einbruchsentdeckenden Modul** der ID daraufhin überprüft, ob er eine statische Anomalie erzeugt oder ein einbruchsverdächtiges Verhaltensmuster vervollständigt. Das anomalie- und einbruchsentdeckende Modul kann aus mehreren Teilsystemen bestehen, die parallel ablaufen können und mit Hilfe unterschiedlicher Verfahren nacheinander eintreffende Audit-Records analysieren können. In dem von Prof.Dr. Simone Fischer-Hübner eingebrachten Konzeptvorschlag sind zwei Teilsysteme vorgesehen :

   (a) Statischer Anomalie-Überprüfer : Basiert auf Dorothy Denning's *Intrusion Detection Model*. Wie bei **IDES**, welches auch auf diesem Modell basiert, werden hierbei statische Profile zur Verhaltensrepräsentation von Subjekten, in Bezug auf Objekte, verwendet. Zur Erzeugung der Profile und für die Entdeckung der Anomalien werden Regeln benutzt.

   (b) a priori-Überprüfer : Grundlage für die Überprüfung sind bekannte Systemverletzlichkeiten. Der a priori-Überprüfer verwendet sogenannte *a priori-Regeln*, die Wissen über diese bekannten Systemverletzlichkeiten kodieren. Einige Regeln z.B. überprüfen Aktionssequenzen auf typische Aktionsmuster von bekannten und unbekannten Computerviren.
   Wie schon erwähnt, können auch andere Teilsysteme in Betracht gezogen werden, die nach anderen Methoden versuchen, Einbrüche zu entdecken. Als Beispiel wurde in [FH92] ein einbruchsentdeckendes System erwähnt, das durch zeitbasiertes, induktives Lernen, sequentielle Regeln, die die zeitliche Beziehung zwischen Benutzer-Aktivitäten charakterisieren, erzeugen kann (für genauere Informationen sei auf [FH92] verwiesen).

Die oben beschriebenen Teilsysteme der anomalie- und einbruchsentdeckenden Komponente, die parallel ablaufen, können weiter verfeinert betrachtet werden. Sie können, bespielsweise, wiederum jeweils aus einem vorzeitig überprüfenden und einem asynchron ablaufenden Teil bestehen, die jeweils verschiedene Typen von ankommenden Audit-Records sequentiell abarbeiten. Die Art der Rückmeldung des ID-Entscheidungsmoduls, die an RM übergeben wird, ist von den Analyseergebnissen des **vorzeitigen Überprüfers** abhängig. Diese Analyseergebnisse sind durch die Gesamtheit der vorzeitig überprüfenden Teile gegeben. Daraus ergibt sich die Notwendigkeit, dass der RM mit seiner Entscheidung auf die Kernfrage warten muss, bis die Analyse des vorzeitigen Überprüfers abgeschlossen ist und die Rückmeldung an RM erfolgt. Auf die Ergebnisse der Analyse eines Audit-Records durch den **Run-Time-Überprüfer** müssen das ID-Entscheidungsmodul und RM aber nicht warten. Der Run-Time-Überprüfer läuft asynchron zu RM. Seine Analyseergebnisse können von dem ID-Entscheidungsmodul zur Entscheidung über spätere Aktionen desselben Subjektes herangezogen werden. Wichtig ist dabei, dass die Zugriffe beider Überprüfer, auf gemeinsame Datenbasen, synchronisiert werden. Die Überprüfung wird folgendermaßen den beiden Überprüfern zugeschrieben [FH92]:

- **Vorzeitiger Überprüfer :** Überprüft vor allem besonders sicherheitsrelevante Aktionen wie Login, Zugriff auf sensitive Daten, Verändern von Zugriffsrechten oder Sicherheitslevels.
- **Run-Time-Überprüfer :** Wertet Audit-Records mit Hilfe Informationen, die erst nach der Ausführung von Kommandos zur Verfügung stehen, aus. Dazu zählt unter anderem der Betriebsmittelverbrauch. Durch die Aufdeckung von anomalem Betriebsmittelverbrauch, können Attacken wie verdeckte Kanäle, Tracker-Angriffe auf statische Datenbanken, oder Computerviren in Aktion, festgestellt werden. Der Run-Time-Überprüfer ermöglicht zwar nicht das sofortige Einschreiten auf eine verdächtige Aktion hin, kann aber solche Arten des illegalen Informationsflusses oder andere Einbruchsversuche, die zur erfolgreichen Durchführung mehrere Aktionen erfordern, noch rechtzeitig stoppen.

Die von den Teilsystemen des anomalie- und einbruchsentdeckenden Moduls entdeckten statischen Anomalien oder sonstigen verdächtige Verhaltensmustern werden, in Form von Anomalie-Records, in eine **Anomalie-Datenbasis** eingefügt, aus der zuvor veraltete Anomalie-Records gelöscht wurden.
Das **Entscheidungsmodul** erhält die aktuellen, gemeldeten Anomalien

von der Anomalie-Datenbasis. Das Entscheidungsmodul ist die Komponente, die dem RM eine Rückmeldung übermittelt. Falls das anfragende Subjekt eine bestimmte Kombination von aktuellen Anomalien verursacht hat, die denen aus dem **Anomalie-Datenbasis** entsprächen oder ähneln, so erhält RM von ID, über das Entscheidungsmodul, eine negative Rückmeldung. Somit besitzt die ID eine Art „Vetorecht". Das bedeutet, dass Kernfragen, die eine negative Rückmeldung von ID an den RM verursachen, auch von RM negativ entschieden werden. Damit eine Kernfrage als „nicht sicherheitsgefährdend" eingestuft werden kann, muss es von RM und ID als solches eingestuft werden. Negative Rückmeldungen von ID haben, außer einer negativen Entscheidung von RM, auch noch weitere Aktionen zur Folge, deren Ausführung von RM veranlasst wird. Dazu gehört z.B. das Sperren des anfragenden Subjekts, und das Entfernen des Subjekts vom System. Falls eine Kernfrage positiv entschieden wird, was eine Änderung von Subjekten oder Objekten zur Folge hat, dann muss ID über diese Änderung eine Meldung erhalten, damit es seine Datenbank (z.B. die statischen Profile) anpassen , d.h. aktualisieren, kann.
ID ist eine lernende, adaptive Komponente und im Gegensatz zu RM nicht formal verifizierbar.

RM und ID laufen im privilegierten Kernelmode, um die ID vor unautorisierten Manipulationen zu schützen. Zugriffe auf ID sind somit nur über RM möglich und werden von demselben kontrolliert. Auch der RM ist vor unerwünschten Manipulationen zu schützen, d.h. die Schnittstelle zwischen RM und ID muss exakt definiert werden.
Die Zugriffsmöglichkeiten des Auditors sind auch genau definiert. Der Auditor kann auf die Wissens-Basis von ID schreibend zugreifen. Dieser Zugriff wird vom der RM kontrolliert. Er kann auch Grenzwerte in Profile setzen und Regeln hinzufügen, so dass er Einfluss auf den aktiven Audit-Prozess nehmen kann. Das direkte Wiedereinsetzen von gesperrten Subjekten ist aber nur durch den Sicherheitsbeauftragten möglich. Um Sicherheitsalarmen nachgehen zu können, muss der Auditor auch lesend auf die Anomalie- und Audit-Daten zugreifen können. Das Entschlüsseln der Subjektfelder ist jedoch nur mit zwei Schlüsseln möglich. Einer der Schlüsseln ist im Besitz des Sicherheitsbeauftragten und der andere wird, zur Kontrolle des Sicherheitsbeauftragten, von einer zweiten Person, wie z.B. vom betrieblichen Datenbeauftragten oder vom Betriebsrat, verwaltet.

Das IDA-Konzept wurde exemplarisch auf einer VAX/VMS-Maschine implementiert. Da kein Source-Code eines realen Betriebssystems zur Verfügung stand, konnte das IDA-Konzept nicht in einem realen Betriebssystem getestet werden. Stattdessen wurde der RM in der Sprache Pascal, und die einbruchsentdeckende Komponente in der Sprache OPS5 exemplarisch implementiert. Die einbruchsentdeckende Komponente wurde dann mit dem RM gekoppelt. Es wurden ver-

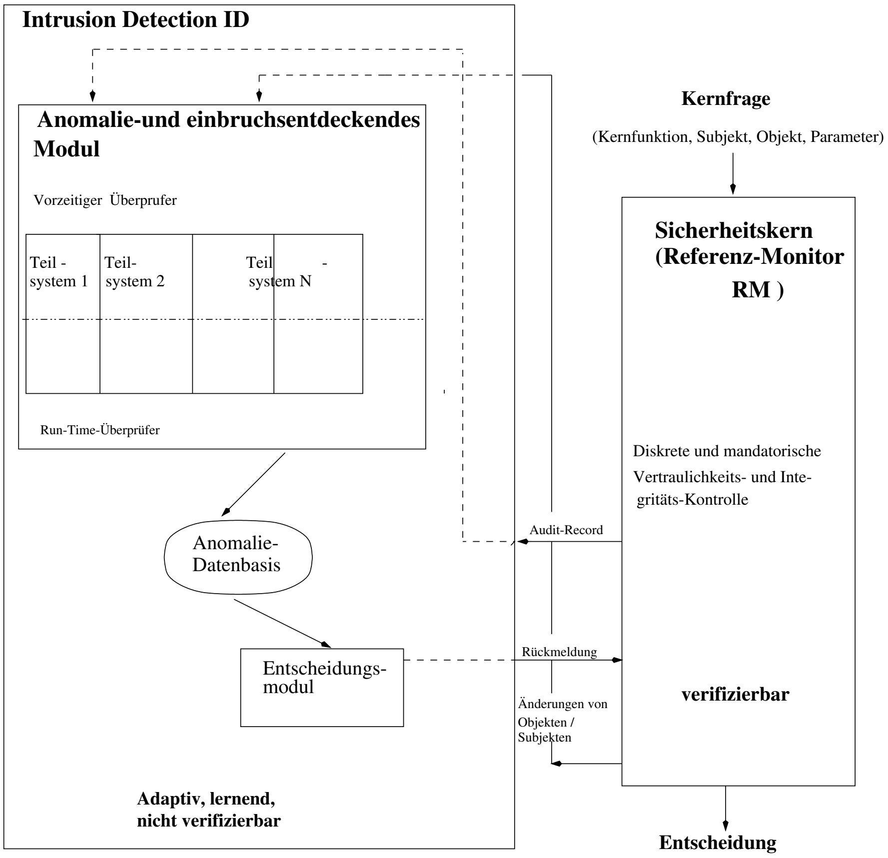

Abbildung 3.14: Architektur von IDA [FH92]

schiedene Tests mit Sequenzen von Kernfragen, die einfache Angriffe simulieren, durchgeführt, die gezeigt haben, dass IDA sie erkennen und stoppen kann. Ein intensives Testen war aber nicht möglich, weil keine geeigneten Audit-Daten vom laufenden Betrieb eines Systems über längere Zeiträume aufgezeichnet werden konnten, die in eine Kernanfrage-Sequenz für das IDA-System hätten umgewandelt werden können. Die erwähnten Probleme bei der realen Umsetzung des IDA-Konzeptes setzen die Performanz dieses System herab.

3.6.5 Intrusion Detection In Our Time (IDIOT)

IDIOT ist ein ID-System, das in der ersten Hälfte der 90iger an der Purdue Universität entwickelt wurde. Dieses System ist speziell der Missbrauchsentdeckung gewidmet. Dabei werden die Signaturen von Angriffen als eine Art gefärbter Petrinetze dargestellt. Die hier verwendete Klasse von Petrinetzen wird auch als gefärbte Petrinetzautomaten (CPA) bezeichnet. Die CPAs spiegeln die kausalen Abhängigkeiten von Ereignissen wider. Diese Ereignisse wurden wiederum durch die Handlungen, die während eines Angriffes von statten gehen, hervorgerufen und von den Auditeinheiten protokolliert. Die gefärbten Petrinetze verdeutlichen auch die verschiedenen Stadien der sequentiellen Mustererkennung. Das hier vorgestellte Modell ist somit auch eine Variante der Zustandsübergangsanalyse, die hier speziell an die Bedürfnisse der Mustererkennung angepasst wurde.

3.6.5.1 Das Modell im Überblick

Die Erkennung von Angriffszenarien lässt sich in verschiedene Stadien aufteilen. Zu jedem Stadium gehört eine Menge von Ereignissen. Diese Ereignisse sind Daten, die von den Auditeinheiten aufgezeichnet werden. Die Ereignisse bilden untereinander eine partielle Ordnung. Genauso bilden auch die Stadien eine Ordnung. Die Kausalen Abhängigkeiten der Ereignisse, die zu der Signatur eines Angriffs gehören lassen sich in sog. Auftragssysteme abbilden, wobei die Ereignisse den Aufträgen aus der Definition von Auftragssystemen gemäß [Val] entsprechen. Da die Ereignisse untereinander eine partielle Ordnung bilden, d.h. eine irreflexive, transitive und antisymmetrische Relation bilden, entspricht diese Relation der Präzedenzrelation von Auftragssystemen.
Aus den Auftragssystemen lassen sich wiederum Kausalnetze konstruieren. Kausalnetze lassen sich dann zu den noch allgemeineren gefärbten Petrinetzen erweitern, aus denen dann die Petrinetzautomaten abgeleitet werden. Die Ereignisse bekommen dabei die Rolle der Transitionen. Die zwischen den Transitionen eingefügten Stellen repräsentieren aus dem "Systemzustand" abgeleitete Gegebenheiten. Wobei hierbei berücksichtigt werden muss, das mit "Systemzustand" nur das bezeichnet sei, was aus den relevanten Auditdaten über den Zustand des Systems, wie er zu der Zeit des Aufzeichnens der Daten gewesen ist, von dem ID-System konstruiert werden konnte. Diese Gegebenheiten stellen einige der notwendigen Bedingungen dar, die bei einem Zustandsübergang des Mustervergleichs gegeben sein müssen. Es können nun noch weitere Bedingungen, sogenannte Wächter, einer jeden Transition beigefügt werden. Auch die in den "Wächtern" spezifizierten Bedingungen sind für das Feuern/Schalten der Transition notwendig. Sind die durch eine Stelle gegebenen Bedingungen erfüllt, so wird sie als markiert bezeichnet und enthält mindestens eine Marke (token). Grafisch werden solche Marken als kleine ausgefüllte schwarze Kreise dargestellt. Diese Marken haben noch weitere Eigenschaften, die weiter unten noch genauer erklärt werden.
Dieses Modell bietet die Möglichkeit der Definition von Präzedenzen von Ereignissen. Also die Angabe, welche Ereignisse stattgefunden haben müssen, damit ein anderes überhaupt auftreten kann. Das Modell stellt einen recht hohen Grad der Abstraktion von der eigentlichen Problematik, der Angriffserkennung dar. Dadurch lässt sich für das gesamte Intrusion Detection System ein geschichtetes Modell aufstellen. Die unterste Schicht wird als Informationsschicht bezeichnet. Hier sorgt eine "low level" Datenschnittstelle für die Kapselung der Auditspur vom ID-System. Die direkt folgende Schicht ist die Signaturschicht. Hier werden

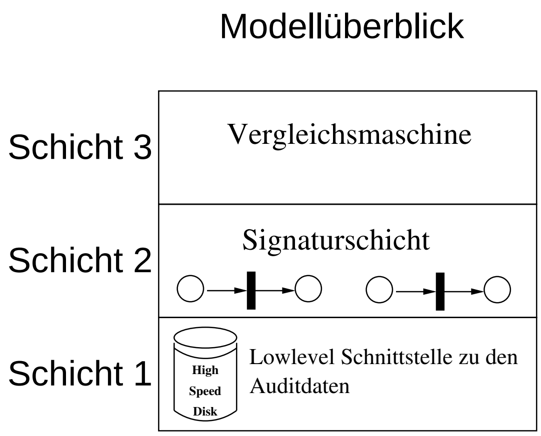

Abbildung 3.15: Schichtenmodell von IDIOT

systemunabhängig die virtuelle Maschine, die den Signaturkontext darstellt und die Signaturen der Angriffsszenarien definiert. Die 3. und letzte Schicht beheimatet schließlich die Vergleichsmaschine. D.h. hier sind die Methoden zur Mustererkennung, die ja auf den CPA aufbauen, definiert. Da das Modell allgemeiner Natur ist, kann es theoretisch auch zu anderen Zwecken, als der automatischen Einbruchsentdeckung dienen. Hierzu müssen im Prinzip nur die Ereignisse der entsprechenden Domäne in das für das Mustererkennungssystem richtige Format gebracht werden. Dies bedeutet eine Anpassung des Systems in der untersten Schicht. Danach müssen noch die zu erkennenden Muster in die entsprechenden Signaturen kodiert werden. Dies geschieht in systemunabhängiger Weise mit einer Sprache, die auf die später noch genauer eingegangen wird.

3.6.5.2 Ein Beispiel

Das in Abbildung 3.16 Angegebene Szenario besteht aus fünf Schritten. Dabei ist von der Reihenfolge her nur wichtig, dass Transition 2 nach Transition 1 und Transition 4 nach Transition 2 und Transition 3 geschieht. D.h. heißt, dass es für dieses Szenario drei Varianten gibt, nämlich $< t1t2t3t4 >$, $< t1t3t2t4 >$ und $< t3t1t2t4 >$ Alle drei Varianten werden vom gleichen CPA aus Abbildung 3.16 repräsentiert.

3.6.6 EMERALD

Die Bezeichnung EMERALD leitet sich aus "Event Monitoring Enabling Responses to Anomalous Live Disturbances" ab. EMERALD ist der Nachfolger des ebenfalls von SRI entwickelten Systems NIDES.

Die einzelnen Komponenten des Systems können unabhängig eingesetzt werden und lassen sich dynamisch aufstellen und konfigurieren. Daher ist die typische

Beispiel eines Angriffsszenarios:
1. cp /bin/sh /var/spool/mail/root #Kopiere Kommandozeileninterpreter
2. chmod 4755 /var/spool/mail #setze die Rechte auf sxr
3. touch x #Erzeute eine leere Datei
4. mail root < x #Schicke diese Datei an Root
5. /var/spool/mail/root #Kommandozeileninterpreter starten

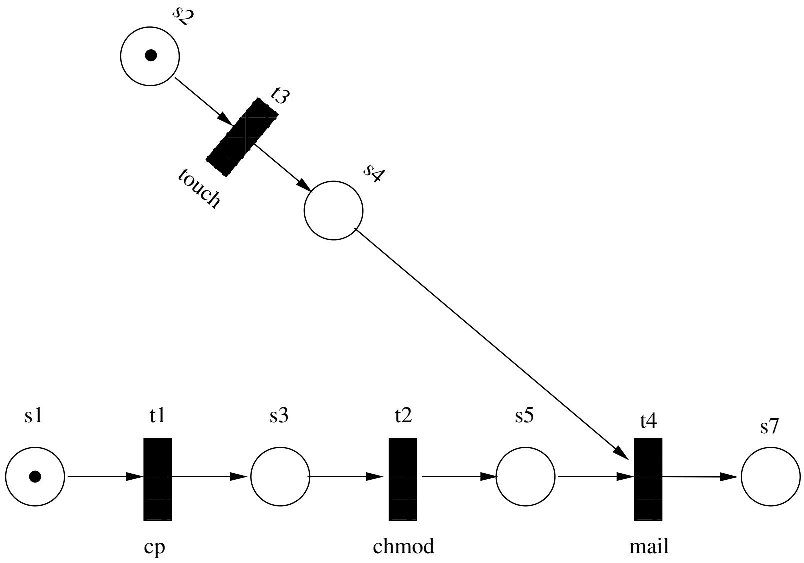

Abbildung 3.16: Ein Angriffszenario als CPA

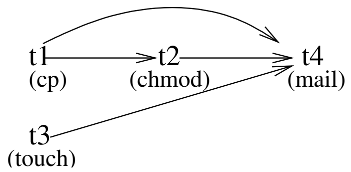

Abbildung 3.17: Die aus dem Netz in Abb. 3.16 konstruierte Präzedenzrelation

Zielumgebung für EMERALD ein großes Netzwerk, das aus vielen unabhängig von einander administrierten Subdomänen besteht. Jede solcher Domänen stellt aus der Sicht von EMERALD ein Netzwerk von lokal oder netzweit verfügbaren Diensten dar. Zwischen den Domänen bestehen Vertrauensbeziehungen (entweder von Rechner zu Rechner (peer to peer) oder hierarchisch). Benutzer können Zugänge zu einer oder mehrerer Domänen besitzen.

Wohl etablierte Konzepte der Computersicherheit, wie der Referenzmonitor, lassen sich nicht gut auf die in der Realität vorliegenden Unternehmensnetze anwenden. Einer der Gründe ist die Dynamik der Netzwerkkonfiguration eines großen Unternehmens. Eine einzige Sicherheitspolitik über ein so großes Netz zu definieren dürfte nicht praktikabel sein. Genauso wäre eine einzige Autorität, die die Einhaltung der Richtlinien aus der Sicherheitspolitik überwacht, damit überfordert oder zu schwerfällig.

Da traditionelle Ansätze schwierig anzuwenden sind, gleichzeitig aber die Notwendigkeit und der Bedarf an Sicherheit steigen, enstehen wichtige Fragen, wie, "Kann man Überwachungs- und Antwortfähigkeiten herstellen, die auf das gesamte Unternehmensnetz passen?". Wenn man dies tun will, so müssen einige Hürden überwunden werden. Ein Teil der Hürden ist durch das bisherige Paradigma der zentralisierten Architektur von ID-Systemen entstanden. Daher kann eine Verteilte Architektur diese Art der Hürden nehmen, stößt aber andererseits wieder auf neue Hindernisse, die z.B. Punkte wie die Performanz betreffen. Die Entwickler von EMERALD sahen sich daher vor allem mit folgenden Herausforderungen konfrontiert:

- *Ereigniserzeugung und Speicherung:* Die Generierung und Speicherung von tendiert dazu eine zentral angelegte und gesteuerte Aktivität zu sein. Hinzu kommt, daß die meisten nativen Auditwerkzeuge Daten in einer so nicht verarbeitbaren Form liefern. Z. B. ist das Abstraktionsniveau der Daten für eine sinnvolle Weiterverarbeitung noch zu niedrig. Eine zentrale Sammlung von Auditdaten erzeugt einen zu großen Überhang an Verwaltungsaufwand, der die Systeme über Gebühr belastet. Außerdem lassen sich zentralisierte Auditmechanismen nur schlecht auf räumlich getrennte Komponenten ausdehnen.

- *Zustandsraum-Management und Regelkomplexität:* In signaturbasierten Analysen hat die Komplexität der Signaturen (oder Regelbasis) einen direkten Einfluß auf die Leistung der Systeme. D.h. eine hoch entwickelte Regelbasis, die in der Lage ist, komplexe Ereignisordnungen mit ausgefeilten Vor-, Neben- und Nachbedingungen zu repräsentieren, erlaubt eine knappe und gut strukturierte Darstellung von Angriffszenarien. Jedoch belasten solche ausgefeilten Regelwerke die Maschinen, da große Mengen an Informationen während der Abarbeitung der Regeln verwaltet werden müssen. Dies begrenzt wiederum die Skalierbarkeit und das Volumen der Daten, die Verarbeitet werden können. Einfachere Regelbasen bieten dann möglicherweise eine größere Flexibilität in Bezug auf die Erweiterbarkeit. Hier kommt es auf den Parameter oder die Parameterkombination an, nach der das System optimiert muß. Ist die Geschwindigkeit ein dominierendes Merkmal für das System, so kann die Regelbasis nur von beschränkter Komplexität sein. Ist die Zahl der erkennbaren Angriffe wichtig, so muß man auf Geschwindigkeit verzichten.

- *Die Wissensbasis:* In Expertensystemen findet eine Trennung von Basiswissen und Analysekode und Anwortslogik statt. Die Wissensbasis ist dabei die Menge aller Inferenzregeln, die dem System bekannt sind.

- *Die Inferenz-Architektur:* Der Kern von Expertensystemen ist ein Algorithmus, der Eingaben (in diesem speziellem Falle Fakten aus den Aktivitätsprotokollen) akzeptiert, auf Regeln anwendet und so die Eingaben zu neuen Fakten verarbeitet. Dann, basierend auf den gewonnenen Informationen, wieder neue Eingaben verarbeitet usw.. Von Natur aus ist dieses Inferenzmodell zentralisiert. Dies zieht eine zentralisierte Ereignisprotokollierung nach sich, wodurch die Systeme mit hohen I/O-Raten belastet werden. Aufgrund einer solchen hohen Last läßt sich das System dann nur noch begrenzt skalieren. Andererseits würde eine vollständig verteilte signaturbasierte Analyse ihre ganz eigenen Probleme herbeiführen. Beides, die globale Korrelation und die intelligente Koordination zwischen den verteilten Analyseeinheiten bedeuten einen großen Verwaltungsüberhang. Die Suche nach dem Optimum in zwischen den beiden Extremen, dem Paradigma der zentralisierten Expertensystemanalyse und dem der verteilten Analyse, ist eine der Schlüsselherausforderungen, wenn man eine skalierbare Inferenz-Architektur entwickeln will.

3.6.6.1 Der EMERALD-Monitor

Der Hauptaspekt bei EMERALD ist der Monitorbegriff. Ein Monitor ist in diesem Zusammenhang eine komplette Analyseeinheit, in der Regel bestehend aus Komponenten, die einerseits statistische Analysen und andererseits Signatur basierte Analysen durchführen, zusammen mit einem Resolver, der die Analyseergebnisse seinerseits bewertet. Die Komponenten innerhalb eines Monitors kommunizieren über eine Programmierschnittstelle, das "Monitor **A**pplication **P**rogramming **I**nterface" (API). Dieses API wird auch für die Kommunikation einer Monitorinstanz mit anderen Monitorinstanzen oder anderen Sicherheitstools genutzt. D.h., ein Monitor kann seine Analyseergebnisse über das Monitor API wiederum an andere EMERALD-Monitore schicken oder von anderen EMERALD-Monitoren Ergebnisse bekommen.

Diese Kommunikationsfähigkeit ermöglicht, eine geschichtete Analysehierarchie aufzubauen, um auch globalere Angriffe, die z.B. Subdomänen betreffen, zu erkennen. Somit ist dieses System auch in der Lage Angriffe, die die Interkonnektivität zwischen einzelnen Subdomänen eines Unternehmens stören oder zerstören, zu erkennen, selbst bei koordinierten Angriffen, die von verschiedenen Bereichen aus geschehen.

3.6.6.2 Anomalieerkennung

Modell: Der augenblickliche Zustands eines Systems sei beschrieben durch einen n-dimensionalen Vektor $\vec{Z} \in I\!R^n$. Der (empirisch) ermittelte Mittelpunkt des Raumes von Normalzuständen des Systems sei der Vektor $\vec{P} \in I\!R$. Ferner sei

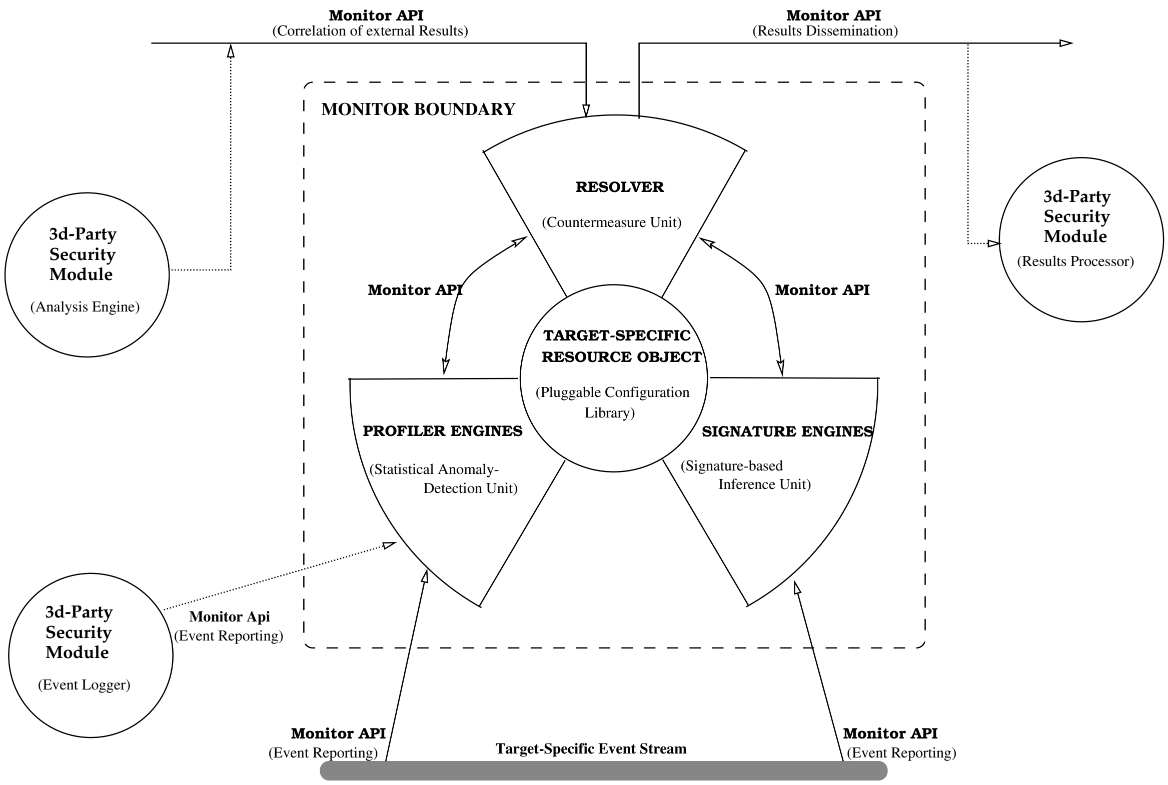

Abbildung 3.18: Architekturüberblick eines EMERALD-Monitors

$k \in I\!R$ die Entfernung von $\vec{P}$, wo der Raum der anomalen Zustände beginnt. Eine Anomalie liegt nach diesem Modell immer dann vor, wenn $|\vec{P} - \vec{Z}| > k$ gilt. Im euklidischen n-dimensionalen Raum wäre die Betrags-Funktion

$$|\vec{P} - \vec{Z}| = \sqrt{\sum_{i=1}^{n} (p_i - z_i)^2} \qquad p_i, z_i \in I\!R$$

Anstelle des einfachen Betrages verwendet man in der statistischen Analyse häufig das Quadrat des Betrages. Dadurch gewinnt man folgende Formel:

$$|\vec{P} - \vec{Z}|^2 = \sum_{i=1}^{n} (p_i - z_i)^2$$

Der Vergleich, der durchgeführt wird, um ein Verlassen des Raumes der Erlaubten Zustände zu entdecken, sieht in eine Formel gebracht wie folgt aus:

$$|\vec{P} - \vec{Z}|^2 > k^2$$

Problem: Wegen möglicher paarweiser Abhängigkeiten zwischen den einzelnen Komponenten x_i und x_j sollten auch noch andere Funktionen als Betrags-Funktionen mit einbezogen werden. Eine Möglichkeit, wie man die Betragsfunktion durch Heuristiken ergänzen kann, wird im Abschnitt 3.6.6.3 skizziert.

3.6.6.3 Anomalieerkennung mit EMERALD

Das statistische Subsystem eines EMERALD-Monitors verfolgt die Aktivitäten eines Subjektes anhand von statistischen Maßzahlen, die über verschiedene Merkmalräume gebildet werden. Die Kategorien, in die sich die Merkmalräume einteilen lassen sind die gleichen wie bei dem Systemen IDES und NIDES:

- **Unbedingte Merkmale**(*categorical measures)*: Diese Merkmalräume kommen bei nicht geordneten diskreten Mengen wie Rechnernamen, IP-Adressen oder Portnummern zum Einsatz. Auch die Bezeichnung *ungeordnete Merkmalräume* wäre zutreffend.

- **Bedingte Merkmale**(*continuous measures)*: Die Elemente dieser Mengen können Werte aus einer zusammenhängenden, geordneten Menge annehmen. Hier gibt es bei den Folgen der beobachteten Werte keine *Sprünge* oder *Lücken*. Der Begriff "continuous measures" könnte auch mit *geordnete Merkmale* gleichgesetzt werden.

- **Abgeleitete Merkmale**(*derived measures)*: Diese Merkmalräume lassen sich noch weiter unterteilen:

  - **Intensitäten**(*intensity measures)*: Hier werden die Zahl der Ereignisse innerhalb einer festgelegten Zeitspanne (zwischen einer Minute und einer Stunde) betrachtet.
  - **Ereignisverteilungen**(*event distribution measure)*: Hier wird die Auswirkung einer Aktivität auf die einzelnen Profile und Maße betrachtet. Diese Kategorie von Merkmalen kommt besonders bei Korrelation von Analyseergebnissen in Anwendung.

Grob teilen sich die Merkmale also in Werte, die schon "natürlicher Weise" numerisch sind und solche, die eigentlich Paare darstellen, bei denen die eine Komponente ein Element aus einer Beobachtungskategorie ist und die andere Komponente eine Zahl ist.

Auf der Basis der Merkmale werden Profile erstellt. Ein Profil besteht unter EMERALD aus zwei Teilen:

Kurzzeitprofil: Hier werden die Messwerte der jüngsten Vergangenheit gesammelt. Der Zeitraum, über den sich eine Sammelperiode erstreckt, ist fest und subjektspezifisch. Am Ende der Sammelperiode wird dieses Kurzzeitprofil in das Langzeitprofil gefaltet und für die nächste Periode initialisiert.

Langzeitprofil: Dies ist die Repräsentation des langfristigen normalen Verhaltens des beobachteten Merkmales.

Die Profile sind Reihen von Messwerten. Die einzelnen Messwerte der Profile unterliegen einem exponentiellen Alterungsmechanismus. Auf diese Weise werden die länger zurückliegenden Werte weniger stark gewichtet als die neueren Messwerte.

Jeder Audit-Eintrag, der durch die Aktivitäten des beobachteten Subjektes entsteht, wird in einen statistischen Testwert T^2 umgerechnet. Der Wert von T^2 ist die Zusammenfassung des Grades an Abweichung von der Norm des Subjektes. In T^2 sind die statistischen Werte von verschiedenen relevanten Merkmalen enthalten. Da die Werte der verschiedenen Merkmale sehr unterschiedlich sind, müssen die ursprünglich beobachteten Werte erst einmal umgerechnet werden. Die ursprünglich gemessenen Werte eines Merkmales i seien mit Q_i bezeichnet. Damit die einzelnen Q_i miteinander vergleichbar werden müssen einander angepasst, also skaliert werden. Der skalierte Wert eines Merkmales sei S_i. Die Art der Skalierungsfunktion variiert von Merkmal zu Merkmal. Die Skalierungsfunktion beinhaltet schon Wissen über das "*erwartete Verhalten*" und die Wahrscheinlichkeitsverteilung des Merkmales. Die Skalierungsfunktion berechnet für unwahrscheinliche Werte von Q_i große Zahlen für S_i und für die wahrscheinlichen (erwarteten) Merkmalwerte von Q_i kleine Zahlen S_i.

Die einzelnen Merkmalwerte Q_i sind nicht unabhängig von einander. Daher sind auch die S_i-Werte miteinander korreliert. Der Wert T^2 wird nun wie folgt definiert:

$$T^2 = (S_1, S_2, \ldots, S_n) \times (C^{-1} \times (S_1, S_2, \ldots, S_n)^t)$$

Dabei ist C^{-1} die zur Korrelationsmatrix C inverse Matrix ($C \times C^{-1} = E$ wobei E die Einheitsmatrix ist). Die Korrelationsmatrix C gibt an, in welchem Ausmaß ein S_i-Wert von einem anderen abhängt. D. h., C ist eine $n \times n$ Matrix. Im einfachsten Fall gilt $C = E$. Dann vereinfacht sich T^2 zu

$$T^2 = \sum_{i=1}^{n} S_i^2$$

Unglücklicherweise sind die einzelnen Elemente c_{ij} der Korrelationsmatrix nur schwer zu bestimmen. Auch ist das Verhalten dieser T^2-Funktion nicht so vorhersagbar, wie es nötig wäre. Von daher wurde die Form $T^2 = (S_1, S_2, \ldots, S_n) \times (C^{-1} \times (S_1, S_2, \ldots, S_n)^t)$ vereinfacht zur gewichteten Summe der Quadrate der S_i:

$$T^2 = \sum_{i=1}^{n} a_i S_i^2$$

Da bei dieser Form jedoch die Abhängigkeiten der S_i, S_j unberücksichtigt bleiben, wurde die Berechnung von T^2 wieder um eine heuristische Funktion $h(S_i, S_j, c_{ij})$ ergänzt:

$$T^2 = \sum_{i=1}^{n} a_i S_i^2 + \sum_{i=1}^{n} a_{ij} h(S_i, S_j, c_{ij})$$

Die heuristische Funktion $h(S_i, S_j, c_{ij})$ liefert immer dann große Werte, wenn die Werte von S_i, S_j sich nicht den Erwartungswerten gemäß verhalten. Die Wichtungsfaktoren a_{ij} sind dabei positive ganze Zahlen, die vom Sicherheitsbeauftragten spezifiziert werden können. Weiter Einzelheiten dazu, wie die Q_i-Statistiken geführt werden und wie die Abbildung $Q_i \to S_i$ funktioniert können in [LTG+92] nachgelesen werden.

Zusammenfassend sei hier einmal das Anomalieentdeckungsmodell von EMERALD dem vereinfachten Modell von Abschnitt 3.6.6.2 gegenübergestellt. In dem ersten Modell wurde die vereinfachende Annahme gemacht, der Systemzustand sei ein Vektor $\vec{Z} \in I\!R$. Bei EMERALD werden die Merkmale Q_i über eine merkmalspezifische Funktion zu den Größen S_i umgerechnet. Der Vektor $\vec{Q} = (Q_1, Q_2, \ldots, Q_n)$ entspricht dem Zustandsvektor $\vec{Z}$. Der Vektor $\vec{S} = (S_1, S_2, \ldots, S_n)$ entspricht den komponenteweisen Differenzen der Vektoren $\vec{P}$ und $\vec{Z}$. Jedoch wurde bei der Umrechnung von Q_i nach S_i schon die aus der empirischen Verteilung der bisherigen Q_i Werte abgeleitete Wahrscheinlichkeit für das Auftreten des Wertes Q_i berücksichtigt. Dies wurde wie folgt gemacht: Ist die Wahrscheinlichkeit $P(Q_i)$ klein, dann ist der Wert von S_i groß. D. h. die möglichen Werte für ein S_i wachsen umgekehrt proportional zur Wahrscheinlichkeit von Q_i. Der statistische Testwert T^2 ist das Analogon zu $|\vec{P} - \vec{Z}|^2$. Jedoch wurde bei der Berechnung von T^2 noch eine subjektspezifische Funktion $h(S_i, S_j, c_{ij})$ verwendet.

3.6.6.4 Signaturbasierte Analyse

Das Signaturanalysesubsystem eines EMERALD-Monitors benutzt eine Variante des Expertensystems P-BEST, das in Abschnitt 5.2 näher besprochen wird. P-BEST ermöglicht dem Administrator, eine Regelbasis mit bekannten Angriffszenarien für das Expertensystem zu instantiieren.
Die Resultate beider Analyseeinheiten wird durch den Resolver bewertet. Auf diese Weise wird schon innerhalb eines Monitors aus den "low-level"-Daten des Ereignisstromes durch die beiden Analysesubsysteme Daten eines höheren Abstraktionsniveaus. Der Resolver bewertet diese Ergebnisse und erzeugt daraus wiederum seine eigenen Fakten, deren Abstraktionsebene somit eine Stufe höher angesiedelt ist. Die Kombination von Anomalie- und Missbrauchsentdeckung soll dabei die jeweiligen Schwächen beider Methoden ausgleichen.

Wie sehen die Abstraktionsebenen aus? In Abbildung 3.18 ordnen sich die Analysekomponenten eines Monitors um das "Target-Specific-Resource-Object". In diesem Objekt sind alle zur Instantiierung eines Monitors nötigen Methoden und Daten enthalten. Durch das "Target-Specific-Resource-Object" wird der objektspezifische Teil der Informationen vor der entsprechenden Analyseeinheit des Monitors gekapselt. Auf diese Weise entsteht die erste Abstraktion von Information.

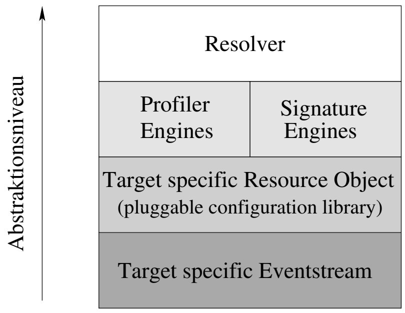

Abbildung 3.19: Schematische Darstellung der Abstraktionsniveaus innerhalb eines Monitors

Die nächste Stufe stellen die Analyseeinheiten selber dar. Deren Ergebnisse werden durch den Resolver erneut untersucht und interpretiert. Auch das ist eine weitere Abstraktion. Schickt der Monitor dann seine Ergebnisse an einen oder mehrere nachgeschaltete Monitore, so werden seine Beobachtungen in weiteren Abstraktionsstufen untersucht. So lässt sich eine Hierarchie aufbauen, die mit den Monitoren, die einzelne Netzwerkressourcen, wie z.B. WWW-, FTP oder Mail-Server überwachen, beginnt und mit Monitoren, die den Zustand des gesamten Netzwerkes beurteilen, endet.

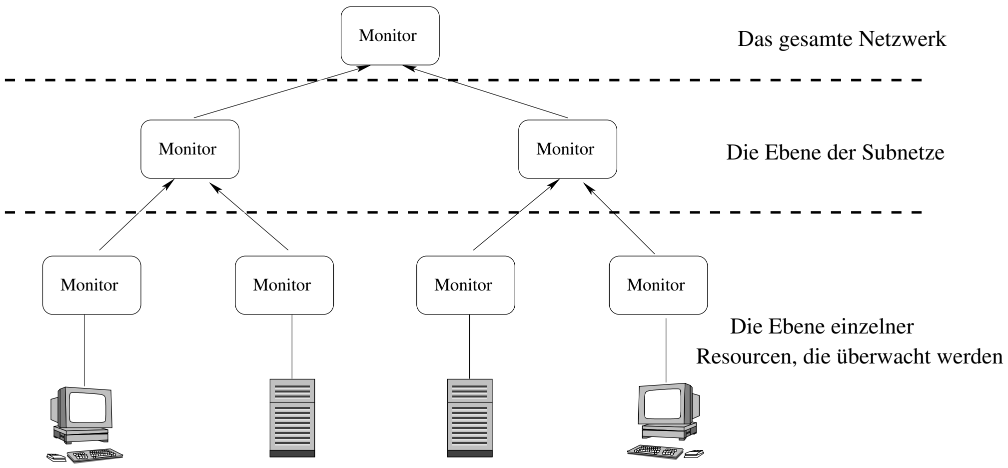

Abbildung 3.20: Schematisches Beispiel einer Hierarchie von Monitoren

3.6.6.5 Die Wissensbasis

Die Signatur-Analysekomponente wird unter EMERALD *eXpert* genannt. Da zum Zeitpunkt des Schreibens dieser Arbeit EMERALD nur in einer Version für das Betriebssystem Solaris mit dem Audittool BSM vorlag, beziehen sich die folgenden Angabe alle sämtlich auf diese Version von EMERALD, die vom Hersteller selber als EMERALD-eXpert-BSM bezeichnet wird.

Die Art der Bedrohungen ist abhängig von der Art der Netzressource. Daher ist auch das Wissen, mit dem ein Angriff auf die Sicherheit einer Ressource erkannt werden kann, von Ressource zu Ressource unterschiedlich. Es gibt also für jede Art von zu schützender Ressource eine andere Wissensbasis. Da ein EMERALD-Monitor jedoch immer gleich ist, muss die Wissensbasis in dem "Target Specific Ressource Object" untergebracht sein.

Die Wissensbasis besteht aus einer Menge von Regeln, die auch Produktionen genannt werden. Die Regelmenge wird in der Version für Solaris grob nach dem Schweregrad der vermuteten Sicherheitsverletzungen (severity metrics) unterteilt. Die Regeln werden in der Sprache des Expertensystems P-BEST verfasst, dann von dem P-BEST-NACH-C-COMPILER (pbcc) in Routinen der Programmiersprache C übersetzt. Diese C-Routinen werden dann als Programmbibliothek bei der Instantiierung des Monitors in die "pluggable configuration library" eingebunden.

Für jeden Schweregrad von Sicherheitsverletzungen gibt es eine Menge von Heu-

DEBUG_INFO	Optionale Botschaften auf der Konsole. Nur zum "debuggen" des Ereignisstromes geeignet.
INFORMATIVE	Optionale Botschaften niederer Priorität über den Status des Monitors.
WARNING	Eine ausnahmeauslösende Aktivität, die symptomatisch für mögliche Angriffe ist, wurde beobachtet. Eine Häufung solcher Alarme sollte durch den Administrator geprüft werden.
SEVERE_WARNING	Die beobachtete Aktivität passt in Angriffsschemata, kann jedoch auch nicht maliziöse Erklärungen haben.
ATTACK	Die Aktivität passt zu bekannten Angriffszenarien. Nicht maliziöse Erklärungen sind unwahrscheinlich.

Tabelle 3.1: Severity-Maße von Regeln für EMERALD-eXpert-BSM

ristiken. Heuristiken wiederum sind Teilmengen der Regelbasis. Die Heuristiken teilen sich nach der Art der Bedrohungen, die durch diese Regeln erkannt werden können.

3.6.6.6 Der Einfluss früherer Systeme auf EMERALD

Da EMERALD vom gleichen Hersteller stammt, wie IDES und NIDES, finden sich Gemeinsamkeiten bei den Systemen. Historisch steht IDES am Anfang. Schon bei diesem System wurden statistische Analysekomponenten mit signaturbasierten kombiniert. Die Signaturkomponente wurde mit dem Expertensystem

P-BEST realisiert. Mit P-BEST können Regeln sehr leicht geschrieben werden.

Die Erfahrungen, die am Computerlabor von SRI gesammelt wurden, beeinflussten die Entwicklung von EMERALD in verschiedenster Weise. So lernte man dort, dass in der Regel die verfügbaren Auditdaten in einer für die Analyse ungeeigneten Form vorliegen und vor der Analyse konvertiert werden müssen. Eine andere Erfahrung war, dass verschiedene Auditquellen vorteilhaft bei der Detektion sind. Ein höherer Grad der Abstraktion hilft vor allem bei der Analyse komplexer Netzwerke. Man erkannte auch, dass das statistische Analysesystem von NIDES für verteilte und Netzwerkumgebungen aus zwei Gründen unpassend war. Erstens, müssen die Merkmale in ihrer Gesamtheit betrachtet werden. Zweitens waren die Analyseergebnisse nicht in einer Form gegeben, die sich zur Weiterverarbeitung in höheren Ebenen eignete. Der Umgang mit NIDES zeigte den Entwicklern auch die Bedeutung der Schnittstelle zur Administration des Systems. Je flexibler dabei das System sein sollte, sowohl in Bezug auf die Entdeckungsmöglichkeiten, als auch in Bezug auf die Antworten, desto komplexer wurde die Administration.

Die Erfahrungen mit statistischen Analysekomponenten von NIDES zeigten außerdem, dass es effektiver ist, Profile über die einzelnen Funktionen eines Systems anzufertigen als Profile der Nutzer, die diese Funktionen benutzen, zu erstellen. Die Falschpositiv- und die Falschnegativraten sanken beträchtlich. Daraus schloss man, dass statistische Analysen von Systemen mit Untersystemen, wie Routern und Servern, sehr effektiv sein konnten.

3.6.6.7 Die Beobachtung von TCP/IP Strömen durch EMERALD

Das folgende ist im wesentlichen eine Zusammenfassung von [PV98].

Leistungsmerkmale Die Auswahl des Ereignisstromes hat einen entscheidenden Einfluss auf die Leistungsfähigkeit der automatischen Überwachung. Der Erfolg oder Misserfolg der Überwachung und Analyse lässt sich durch die Größen

- **Leistung:** Die Zahl der abgearbeiteten Ereignisse pro Zeiteinheit
- **Genauigkeit:** Die Zahl der falsch-positiven Alarme
- **Unvollständigkeit:** Die Zahl der falsch-negativen Vorfälle

ausdrücken. Alle drei Größen sind direkt von der Qualität des Ereignisstromes, der der Überwachung als Datenquelle dient, abhängig.

3.6.7 ASAX

Der Name ASAX leitet sich von "**A**dvanced **S**ecurity **A**udit Trail Analysis on Uni**X** ab. In diesem Abschnitt werden die Konzepte, auf denen ASAX beruht erklärt. Die Sprache RUSSEL, die dem System ASAX zugrunde liegt wird im Abschnitt 5.3 genau beschrieben. In diesem Abschnitt werden nur die Hauptmerkmale dieser Regelbeschreibungssprache dargestellt.

3.6.7.1 Die Geschichte von ASAX

ASAX wurde als ein System zur Analyse von Auditspuren (audit trails) für die Systeme BS2000 und SINIX von der Siemens-Nixdorf AG entwickelt. BS2000 und SINIX sind zwei stark unterschiedliche Betriebssysteme, daher wurde das Design und die Implementierung von ASAX parallel in zwei Versionen durchgeführt auf den jeweiligen Plattformen. BS2000 ist ein Mainframe-Betriebssystem, das für große Server-Rechner entwickelt wurde. SINIX dagegen ist ein UNIX-Derivat.

Da zu Beginn der Entwicklung von ASAX nur wenig Wissen über die semantische Information von Auditdatensätzen und das Format und Layout verfügbar war, standen am Anfang zwei Hauptziele im Vordergrund: Erstens, das Design eines hoch flexiblen und kanonischen Aufzeichnungsformates und zweitens, das Design einer mächtigen, effizienten und leicht zu erlernenden Sprache zur Spezifikation von Regeln, die dazu dienen, beliebig komplexe Angriffsmuster in den Auditaufzeichnungen zu entdecken. Das Design der Auditeinheiten wurde von Siemens-Nixdorf durchgeführt, die Konzeption und Implementation der Sprache von der Universität Namur am dortigen Institute d'Informatique geleistet.

3.6.7.2 Überblick

ASAX wurde mit dem Ziel entwickelt, ein Werkzeug zu bekommen, mit dessen Hilfe große sequentielle Datenströme schnell und effektiv automatisch analysiert werden können. Außerdem sollte dieses Werkzeug weitgehend von dem System, auf dem es eingesetzt wird, unabhängig sein. Das Problem, daß für die Erzeugung und Speicherung von Auditdaten kein Standard herrscht, wurde von ASAX dadurch gelöst, daß es sein eigenes Format einführt, das "**n**ormalized **a**udit **d**ata **f**ile format". Die Auditdaten eines Systems müssen erst durch den "format adaptor" in das NADF-Format übersetzt werden. Diese Dateien werden vom "sequential analyzer" unter Anwendung von Regeln seiner Regelbasis untersucht. Der "sequential analyzer" besitzt aktive und inaktive Regeln. Abhängig von den vorgefundenen Daten in dem Auditdatenstrom kann der "sequential analyzer", auch "RUSSEL's Engine" genannt, Aktionen ausführen oder Regeln aktivieren oder deaktivieren.

Dadurch, daß ASAX weitgehend plattformunabhängig entwickelt wurde, lässt

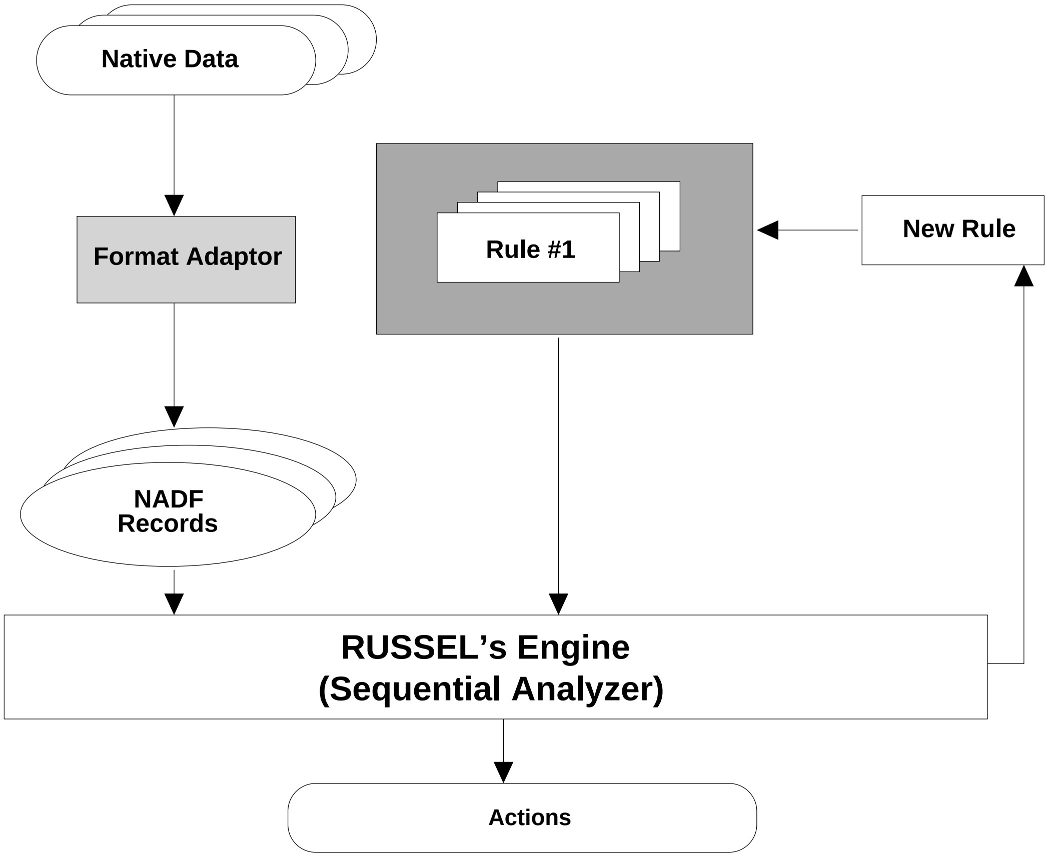

Abbildung 3.21: Konzept von Asax

es sich auch in heterogenen Rechnernetzen einsetzen. Für jedes systemspezifische Auditdatenformat wird daher ein Adaptionsprogramm, das das jeweilige Format in NADF wandelt, benötigt. ASAX unterstützt den Anwender hierbei, in dem es mit einer vielzahl von einsatzbereiten Adaptern ausgeliefert wird. Tritt dennoch der Fall ein, dass man einen Adapter braucht, der nicht mitgeliefert wird, so soll es sehr leicht sein, den fehlenden Adapter selbst zu schreiben. Auch bei dieser Aufgabe wird man von den unter ASAX mitgelieferten Programmen unterstützt.

Das Herz von ASAX bildet der "sequential analyzer". Hier findet die eigentliche "Intrusion Detection" statt. Die Regeln, die der "sequential analyzer" benutzt, werden in der Sprache RUSSEL eingegeben. Effizienz ist bei Intrusion Detection Systemen ein besonders wichtiger Punkt, da man schnelle Auskünfte aus einer sehr großen Zahl von Daten braucht. Dieses versucht man mit ASAX durch zwei Prinzipien zu erreichen:

1. NADF-Datensätze werden in einem Durchgang *single pass* analysiert.

2. Optimierung bei sich wiederholenden Schritten.

3.6.7.3 ASAX und UNIX

In diesem Abschnitt soll erklärt werden, wie das System ASAX in einer UNIX-Umgebung arbeitet. Dabei wird zuerst das normalisierte Auditdatenformat (NADF) vorgestellt. Dann wird die Schnittstelle von ASAX zur Programmiersprache C dargestellt.

Der Aufbau von NADF-Dateien. Die Auditdaten, die von ASAX analysiert werden sollen, müssen in dem ASAX-spezifischen Format, das **n**ormalisiertes **A**udit**d**aten**f**ormat (NADF) genannt wird, vorliegen. Eine NADF-Datei ist eine Folge von Auditdatensätzen. Jeder Datensatz ist seinerseits eine Folge von Datenfeldern. Die Semantik der einzelnen Feldinhalte ist nicht festgelegt, genauso ist die Reihenfolge der Feldtypen nicht vorgeschrieben. Die Arten der verschiedenen Feldtypen sind auch nicht festgelegt oder vorbestimmt. Festgelegt ist lediglich, dass ein Datensatz stets mit der Angabe der Länge des Auditdatensatzes in Bytes beginnt.

Ein Datenfeld wird als ein Paar $(Feldname, Wert) \in I\!N \times WERTE$ aufgefasst, wobei $WERTE$ als stellvertretend für die Menge von möglichen Werten steht. Da die möglichen Werte alle in Folgen von Bytes ausgedrückt werden, ist die Länge eines Datenfeldes nicht bekannt. Daher ist es nötig, entweder ein bestimmtes Feldbegrenzungszeichen in Form einer festgelegten Folge von Bytes zu definieren, oder eine zusätzliche Angabe über die Länge des Feldes in jedes Datenfeld mit aufzunehmen.

Die Repräsentation der Auditdaten. Die Daten eines Auditdatenfeldes werden als ein Tripel $(identifier, length, value) \in UINT \times UINT \times BYTESEQ$ repräsentiert. Dabei ist $UINT = \{x | 0 \geq x \leq 2^{16} - 1\}$ die Menge aller zwei Byte langen vorzeichenlosen ganzen Zahlen. $BYTESEQ$ ist die Menge aller möglichen Bytefolgen, die als Repräsentanten von Werten in Frage kommen. Bytesequenzen mit einer ungeraden Zahl von Bytes werden um ein Füllbyte ergänzt, so dass die Länge einer Bytesequenz immer eine gerade Zahl ist. Dabei wird das Füllbyte jedoch nicht bei der Angabe der Länge der Sequenz berücksichtigt.

Auditdatensätze müssen immer eine durch 4 teilbare Länge haben. Hat ein Datensatz eine kürzere Länge, so werden die fehlenden Bytes als Füllbytes ergänzt. Da schon die einzelnen Felder eines Satzes immer auf Längen gerader Zahlen aufgefüllt werden, kann es nur den Fall geben, dass einem Datensatz zwei Füllbytes hinzugefügt werden. Das NAD-Format enthält die Daten in Binärform, das heißt, die Daten lassen sich nicht mit einem Texteditor direkt anschauen, sondern müssen erst in eine für Menschen lesbare Form gebracht werden, will man diese Daten einmal "von Hand" anschauen. Das Binärformat schränkt auch den Datenaustausch zwischen den unterschiedlichen Plattformen ein. Generell kann

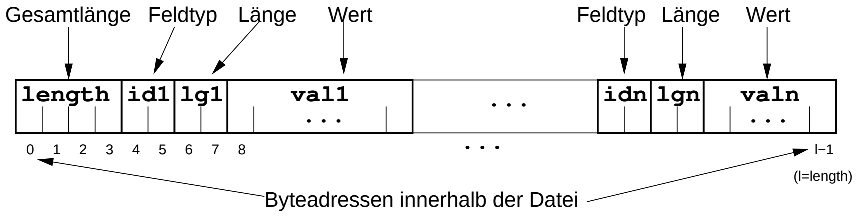

Abbildung 3.22: Allgemeines Layout des NAD-Formates

man auf dieser Ebene zwei Arten von Hardwareplattformen unterscheiden:

1. Little Endians
2. Big Endians

Diese beiden Arten von Plattformen unterscheiden sich hinsichtlich ihrer Repräsentation von Zahlen. Bei beiden Arten werden ganze Zahlen (Integerzahlen) als Bytefolgen fester Länge dargestellt. Auf 32-Bit-Architekturen sind Integerzahlen meist vier Elemente lange Bytefolgen. Unterschiedlich ist jedoch die Berechnung des Zahlenwertes aus den einzelnen Bytes. Bei Little Endians sieht die Berechnung wie folgt aus:

$$I = \sum_{i=0}^{3} byte_i * 2^{8(3-i)}$$

Bei Big Endians geschieht die Berechnung nach folgender Formel:

$$I = \sum_{i=0}^{3} byte_i * 2^{8i}$$

Auditdatenbeschreibungsdateien. Bisher wurde noch nichts über die Bedeutung von einzelnen Auditdatenfeldern gesagt. Dies liegt daran, dass das NAD-Format hierüber keinerlei Annahmen macht. In den Expertensystemregeln, die die Auditdaten untersuchen, wird auf die einzelnen Felder eines Auditdatensatzes mit Hilfe von Feldnamen zugegriffen. Die Feldnamen geben dabei auch Hinweise über die Art des Feldinhaltes, also die Semantik des jeweiligen Feldes. Die Adressierung der Felder über Feldnamen kommt dem Menschen sehr entgegen, ist aber für einen Computer umständlich, daher ist es aus Gründen der Effizienz von Vorteil, wenn die Adressierung der Felder während der Bearbeitung durch einen Rechner über Zahlen geschehen kann.

Eine Auditdatenbeschreibungsdatei (audit data description file) ist eine Textdatei (ASCII), in der die Abbildung zwischen Feldnamen und Feldidentifikationsnummern aufgelistet wird. Jedes Auditdatenfeld wird dabei durch eine Folge von fünf Zeilen beschrieben. Jede Zeile hat dabei eine feste Bedeutung. Die

Feldbeschreibungszeilen werden von eins bis fünf durchnummeriert. Jede Zeile beginnt mit einer Zeilennummer gefolgt von einem Leerzeichen. Dabei gibt die Zeilennummer eins an, dass sich die folgende Angabe auf eine Feldidentifikationsnummer bezieht. Die Zeilennummer zwei bedeutet, dass in dieser Zeile der Datentyp des nativen Auditdatenfeldes angegeben wird. Die Zeile mit der Nummer drei gibt den Datentyp in dem NADF-Datensatz an. Die vierte Zeile gibt den Namen des Auditdatenfeldes an und die fünfte Zeile enthält einen Kommentar über die Bedeutung des Feldinhaltes. Abbildung 3.23 illustriert den Aufbau

```
A Audit Data Description File
B Demo-Daten-Beschreibungsdatei
C 2001, January
D Hamburg

1 7
2 int
3 int
4 uid
5 user id
```

Abbildung 3.23: Ausschnitt einer NADF-Beschreibungsdatei

einer NADF-Datenbeschreibungsdatei. Optional kann eine solche Datenbeschreibungsdatei einen Kopfteil aus Kommentaren enthalten. Die erste Kommentarzeile beginnt mit dem Buchstaben A gefolgt von einem Leerzeichen. Ihr folgen null oder mehr Zeilen die mit einem A beginnen. Dann können null oder mehr Zeilen folgen, die mit einem B beginnen, gefolgt von Zeilen die mit einem C beginnen und so weiter. Zur Verfügung stehen dabei die Buchstaben A bis F.

Ein bisschen C. Im Folgenden wird etwas näher auf die Programmiersprache C eingegangen. Dies ist zum besseren Verständnis der Programmierschnittstelle von ASAX notwendig. Dabei wird jedoch nur die Definition von Variablen und Feldern (Arrays) besprochen.

Zwei C-Datentypen. Im Folgenden wird unter der Bezeichnung `char` die Menge aller ganzen Zahlen in dem Intervall $[-2^7, 2^7 - 1]$ gemeint sein. Dies ist die Menge aller vorzeichenbehafteten Zahlen, die sich durch 8 Bit darstellen lassen. Die Typangabe `int` bezeichnet die Menge der Zahlen aus dem Intervall $[-2^{31}, 2^{31} - 1]$. Dies sind alle vorzeichenbehafteten Zahlen, die sich durch 32 Bit darstellen lassen.

Zeiger in C. Zeiger (pointer) spielen in der Programmiersprache C eine wichtige Rolle. Zeiger sind Datentypen, die Speicheradressen repräsentieren. Zu jedem Datentyp in C gibt es einen zugehörigen Zeigertyp. Zeiger eines bestimmten Typs werden in C durch den Typnamen und den unären Sternoperator `*` angegeben. Der Sternoperator macht bei der Variablendefinition die jeweilige Variable, vor der er steht, zu einer Zeigervariablen. So definiert z. B. die Angabe `int *Laenge` eine Variable vom Typ Zeiger auf `int` mit dem Namen `Laenge`. Um an den Inhalt der durch einen Zeiger referenzierten Speicherstelle zu kommen, kann ebenfalls der Sternoperator eingesetzt werden. Taucht nach den Variablendefinitionen im Programmtext z. B. die Anweisung `*Laenge=40;` auf, so wird der Inhalt der Speicherstellen, auf die der Zeiger `Laenge` weist, auf den Wert rechts des Gleichheitszeichens (in diesem Fall also 40) gesetzt. In diesem Zusammenhang wird der unäre Sternoperator als **Inhalts-** oder **Dereferenzierungsoperator** bezeichnet, da er aus einem Zeiger kurzzeitig eine "normale" Variable macht.

Mit Zeigern kann man in C auch rechnen. Zu einem Zeiger kann man etwas addieren oder subtrahieren, dann ändert sich jedesmal in entsprechender Weise die Speicheradresse, auf die der Zeiger verweist. Dies wird als **Zeigerarithmetik** bezeichnet.

Der Sternoperator lässt sich auch mehrfach hintereinander anwenden. Bei der Definition von Variablen werden dann Zeiger von Zeigern gebildet. Beispiel: `char **Text`. Dieses Beispiel erzeugt einen Zeiger auf einen Zeiger vom Typ `char`. Die Programmiersprache C kennt keinen Feld- oder Arraydatentyp. In C werden Zeiger stellvertretend für Felder (arrays) genutzt.

Felder (Arrays). Obwohl es in C keine Arrays als Datentypen gibt, kann man dennoch Arrays deklarieren. Dieser Widerspruch liegt daran, dass in C ein Feld als ein zusammenhängender Speicherbereich, in dem verschiedene Instanzen desselben Typs abgelegt sind, interpretiert wird. Die Anfangsadresse dieses Speicherbereiches steht in einer Zeigervariablen. D. h., eine Variable, die mit `char *name` definiert wird, kann auf den Beginn eines zusammenhängenden Speicherbereiches mit `char`-Werten zeigen, also auf ein Feld vom Typ `char`. Die Deklaration eines Feldes kann in C z. B. wie folgt aussehen: `char pathname[256]`. Damit wird der C-Compiler angewiesen, eine Zeigervariable namens `pathname` anzulegen und sie mit der Anfangsadresse eines reservierten, zusammenhängenden Speicherbereichs von der Größe 256 Bytes (Werte des Typs `char`) zu initialisieren. Auf die Elemente des Feldes kann jetzt auf zwei syntaktisch verschiedene, aber semantisch äquivalente Weisen zugegriffen werden:

1. Mit `pathname[i]`, wobei `i` einen Wert zwischen 0 und 255 annehmen darf und vom Typ `int` oder `unsigned int` sein muss.

2. Mit `*(pathname+i)`, wobei hier für `i` das gleiche gilt wie unter 1.

Eine Überprüfung der Feldgrenzen gibt es in C nicht. Dies liegt völlig in der Verantwortung des Programmierers.

Die virtuellen Ein- und Ausgabefunktionen (virtual interface) Um ASAXs Inferenzmaschine (RUSSELS's engine) frei von den Details der Ein- und Ausgabe von NADF-Dateien zu halten, wurde eine Schnittstelle mit virtuellen Routinen zur Ein- und Ausgabe eingeführt.

Der Begriff der virtuellen Routinen oder Methoden spielt in der objektorientierten Programmierung eine wichtige Rolle. In der objektorientierten Programmierung liegt der Schwerpunkt auf dem Entwurf von Datentypen. Aus bestehenden Datentypen können über den Vererbung genannten Mechanismus neue, abgeleitete Datentypen gewonnen werden. Um Datentypen so allgemein wie möglich zu halten, sollten alle system- oder kontextabhängigen Methoden eines Typs überschreibbar sein. Dann können bei der Instantiierung eines Objektes in einem System die originalen Methoden durch die systemspezifischen Versionen dieser Methoden überschrieben werden. Virtuelle Methoden sind häufig auch sogenannte abstrakte Methoden, bei denen nur die Arten und Anzahl der Aufrufparameter und die Art des Ergebnisses angegeben werden. Der konkrete Algorithmus zur Berechnung des Ergebnisses jedoch fehlt, da dieser system- oder kontextabhängig ist.

Die Ein- und Ausgabeschnittstelle von ASAX ist eine virtuelle Schnittstelle, d. h. sie definiert nur die Namen und Eigenschaften der Methoden zur Ein- und Ausgabe. Die Ein- und Ausgabe ist immer systemabhängig, daher diese virtuelle Schnittstelle. Die Schnittstelle definiert die Methoden `int vopen(int argc, char **argv)`, `int vread(char **ptr)` und `int vclose()` zum Öffnen , Lesen und Schließen eines Auditdatenstromes.

Die Methode `vopen()` erwartet dabei als Aufrufparameter eine Integerzahl `argc`, die die Größe des Feldes von Zeichenketten angibt, auf das der zweite Parameter `char **argv` zeigt. Der Parameter `argv` ist vom Typ eines Zeigers, der auf ein Feld von Zeigern des C-Datentyps `char` verweist. Ein Zeiger, der auf ein Feld von `char`-Werten zeigt, ist das C-Analogon zu Zeichenketten (Strings). Dieses Feld von Strings ist eine Liste mit Dateinamen. Aus diesen Dateien wird dann die NADF-Datei zur Analyse erzeugt. Der Rückgabewert der Methode ist vom Typ Integer und gibt den Erfolg bzw. Misserfolg des Dateiöffnens an. Bei erfolgreicher Ausführung wird der Wert null zurückgegeben, sonst ein Wert ungleich null.

Die Methode `vread()` hat den Zweck, die Adresse des nächsten zu lesenden Datensatzes zu liefern. Wenn kein Datensatz mehr vorhanden ist, wird als Ergebnis der Wert null zurückgegeben. Bei Erfolg steht nach der Ausführung von `vread()`

die Adresse des nächsten zu lesenden Datensatzes in `ptr`. Diese Methode benutzt die C-Version der Werteübergabe "call by reference". D. h. diese Methode hat als Seiteneffekt die Veränderung des Inhaltes des übergebenen Parameters `ptr`.

Die letzte Methode `vclose()` ist für das ordnungsgemäße Schließen des Datenstromes (der Datei) zuständig. Bei Erfolg wird als Ergebnis die Integerzahl null zurückgegeben. Im Falle eines Fehlers wird -1 zurückgegeben.

Wenn eine lauffähige Version von ASAX auf einem gegebenen System erstellt (instantiiert) werden soll, müssen die Algorithmen für die Methoden zur Ein- und Ausgabe bereitgestellt werden. Die Instantiierung einer lauffähigen ASAX-Version besorgt das Programm `masax`. Diesem Programm kann über Kommandozeilenparameter mitgeteilt werden, welche Datei die Algorithmen für die Methoden "`vopen()`", "`vread()`" und "`vclose()`" enthält. Es erzeugt dann aus dem Modul mit den Implementierungen der virtuellen Ein-/Ausgabefunktionen und der Bibliothek mit dem ASAX-Kernel das ausführbare Programm.

Die Programmierschnittstelle zu C Ein wichtiges Merkmal der Sprache RUSSEL ist die Erweiterbarkeit der Sprache mit Routinen, die in C geschrieben sind.

Um neue Routinen der Sprache RUSSEL hinzuzufügen müssen zwei Schritte unternommen werden:

1. Die Ausführung des Programmes "`libman`". Dieses Programm verarbeitet die Informationen von Bibliotheksbeschreibungsdateien in Informationen für RUSSEL. Die Bibliotheksbeschreibungsdateien enthalten "high-level"-Beschreibungen der C-Module, mit denen RUSSEL zusammenarbeiten soll.

2. Der Aufruf des Programmes "`masax`", welches die Objektdateien und die Kernelbibliothek von ASAX zu einem ausführbaren Programm zusammenfügt.

Diese Schritte erfordern keine Detailenntnisse über die Implementierung von ASAX.

Die Bibliotheksbeschreibungsdateien sind einfache Textdateien (ASCII) die für jede Routine der C-Bibliothek eine einfache "high-level"-Beschreibung enthalten. Die Beschreibung bezieht sich dabei auf die Angabe des Typs des Rückgabewertes der Routine, des Namens, der Typen der Aufrufparameter und der Objektdatei, die den Objektcode der Routine enthält. Die Abbildung 3.24 enthält einen Ausschnitt aus der Bibliotheksbeschreibungsdatei "`sys_ldf`", die schon standardmäßig vordefinierte Erweiterungen der Sprache RUSSEL enthält.

```
integer         accept()                                 : lib_reg;
string          append(string:val, string:val)           : lib_reg;
integer         bytesToInt(string:val)                   : lib_reg;
integer         match(string:val, string:val)            : lib_reg;
integer         strToInt(string:val)                     : lib_reg;
```

Abbildung 3.24: Ausschnitt aus der Bibliotheksbeschreibung “`sys_ldf`”

Damit der Datentransfer bei der Argument- und Ergebnisübergabe zwischen C-Routinen und der Inferenzmaschine (RUSSEL’s engine) funktioniert, muss der Autor der C-Routinen sich an bestimmte, ASAX-spezifische Gepflogenheiten halten. Als Beispiel wird an dieser Stelle einmal die Definition der Routine “`append`” aus Abbildung 3.24 genauer untersucht.

In der Bibliotheksbeschreibungsdatei wird “`append`” als eine Funktion zweier Zeichenketten definiert, deren Funktionswert auch wieder eine Zeichenkette ist. Die Repräsentation von Zeichenketten in RUSSEL ist ein Paar bestehend aus einer natürlichen Zahl l, die die Länge der Zeichenkette angibt und einer Folge von l Bytes, die die einzelnen Zeichen enthalten. Im Arbeitsspeicher eines Rechners sieht die Repräsentation der Zeichenkette wie in Abbildung 3.25 aus. Dabei wurde die natürliche Zahl l in eine 32 Bit lange Integerzahl abgebildet. Der Datentyp `integer` in ASAX entspricht dem vier Byte langen Zahlentyp `int`, `short` oder `long` in C, je nach Plattform.

Das Format der Argumentübergabe sieht so aus, dass jede C-Routine die Form

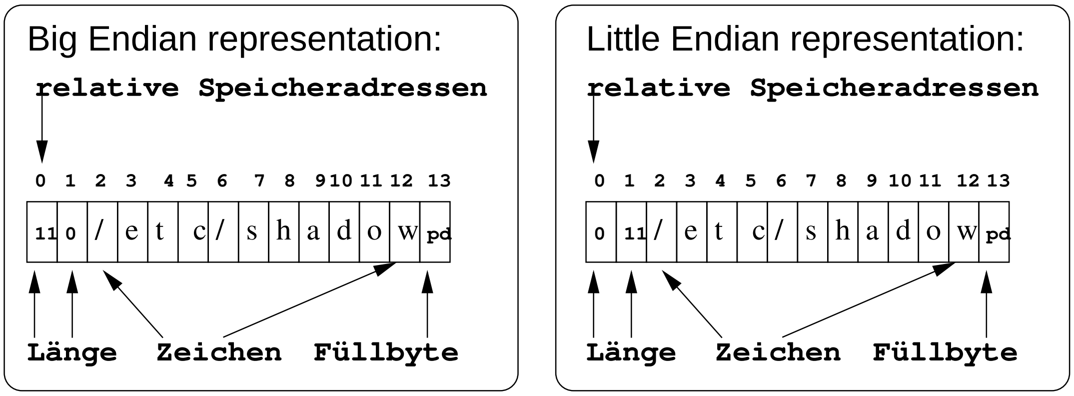

Abbildung 3.25: Repräsentationen einer Zeichenkette in ASAX

von Abbildung 3.26 hat. Nach dem Muster in Abbildung 3.26 scheint es so, als könne man an ASAX nur Funktionen definieren, deren Aufrufparameter Zeichenketten sind (der Datentyp “`char *`” steht in C stellvertretend für den Da-

```
Typ_des_Rueckgabewertes p(char *s){
/*
 * Anstelle von Typ des Rueckgabewertes kann
 * entweder ``int'' oder ``char *'' stehen.
 * Hierauf folgen die Anweisungen und
 * Funktionsaufrufe!
 */
...
}
```

Abbildung 3.26: Muster einer benutzerdefinierten C-Routine für ASAX

tentyp "`string`", wie er in einigen anderen Programmiersprachen bekannt ist.). Tatsächlich steht die Deklaration `char *s` jedoch lediglich für einen Speicherbereich, in dem die Parameter abgelegt werden. Anhand der Bibliotheksbeschreibungsdatei weiß ASAX, wie der Speicherbereich, auf den der Parameter `s` zeigt, zu interpretieren ist.

Wenden wir die gerade gewonnenen Erkenntnisse nun auf das Beispiel der Funktion `append` an. In der Datei aus Abbildung 3.24 besitzt `append` den Typ `string` und zwei Aufrufparameter des gleichen Typs. Der Autor darf in C aber nur die Funktion wie in Abbildung 3.27 deklarieren. Die in dieser Abbildung vorgeschlagene Implementierung der Funktion `append` hat nicht den Anspruch besonders effizient oder geschickt zu sein. Sie ist lediglich ein Lösungsvorschlag. Der Speicherbereich, auf den die Variable `s` zeigt muss dabei zweimal eine Referenz (Zeiger) auf einen String gemäß Abbildung 3.25 enthalten.

Nun besteht die Aufgabe der Parametergewinnung aus dem Speicher, auf den die Variable `s` zeigt, darin, die Adressen der Anfänge der Zeichenketten zu gewinnen und die Längen der Zeichenketten zu ermitteln. In Abbildung 3.27 ist eine Beispielimplementierung, in der diese Aufgaben gelöst werden, zu sehen.

3.6.7.4 Die Instantiierung von ASAX

Zum Zeitpunkt des Schreibens dieser Arbeit ist im World-Wide-Web eine Version aus dem Jahre 1997 zum freien Herunterladen verfügbar. Der URL lautet: `ftp://ftp.info.fundp.ac.be/pub/projects/asax/`. Dort findet man eine mit GNU-Zip komprimierte Tape-Archiv-Datei (tar-file) mit den Quellen von ASAX und eine Postscript-Version der Benutzeranleitung.

Die Compilierung von ASAX. Nach dem Entpacken der Archivdatei müssen in der Datei namens "`Makefile`" einige plattformspezifische Änderungen vorge-

Feb 07, 01 13:15 **append.c** Seite 1/1

```
#include <string.h>

const char fuellbyte=' ';
char *append(char *s){
  void *p1, *p2, *appended, **param;
  char *s1, *s2, *sa;
  unsigned short l1, l2, lneu, *plneu;
  param=(void **)s;
  p1=param[0]; /* Die Adresse des ersten Parameters */
  p2=param[1]; /* Die Adresse des zweiten Parameters */
  l1=*(unsigned short *)p1; /* Zuweisungen der Zeichenketten- */
  l2=*(unsigned short *)p2; /* Laengen an l1 und l2.          */
  s1=(char *)p1+sizeof(l1); /* Zuweisungen der Anfangsadressen*/
  s2=(char *)p2+sizeof(l2); /* der Zeichenketten.             */
  lneu=l1+l2; /* Berechnung der Laenge der neuen Zeichenkette.*/
  appended=malloc(sizeof(lneu)+lneu+lneu%2);
  /* Speicher fuer den neuen String inklusive eines
   * eventuellen Fuellbytes (pad) besorgen.
   */
  plneu=(unsigned short *)appended;
  /* Den Zeiger, der auf die Laenge des
   * neuen Strings verweisen soll, die
   * Adresse der Speicherstellen zuweisen,
   * die die Laenge aufnehmen sollen.
   */
  *plneu=lneu; /* Wertzuweisung der Laenge */
  sa=(char *)appended+sizeof(lneu);
  /* Die Folge der Zeichen beginnt ab sa. */
  memcpy((void *)sa, (void *)s1, (size_t)l1);
  /* Der erste Teilstring wird kopiert. memcpy ist eine
   * Bibliotheksroutine aus der Standardbibliothek von C.
   */
  memcpy(&sa[l1], (void *)s2, (size_t)l2);
  /* Der zweite Teilstring wird kopiert.
   */
  if( (lneu%2) ) sa[lneu]=fuellbyte;
  /* Ungerade Laenge? Dann mit fuellbyte auffuellen. */
  return (char *)appended; /* Fertig! */
}
```

Mittwoch Februar 07, 2001 1/1

Abbildung 3.27: Eine benutzerdefinierte Version von `append`

nommen werden. Danach lässt sich durch die Kommandos "`make libman`", "`make masax`" und "`make asaxKernel.o`" die Instantiierung einer lauffähigen Version von ASAX vorbereiten. Dabei werden die beiden Programme `masax`, `libman` und das Kernsystem von ASAX, welches danach als eine statische Bibliothek vorliegt, erstellt. Nun hat man die Möglichkeit, schon vorbereitete Versionen für die Plattformen Solaris und SunOS zu instantiieren, also ablauffähige Analyseprogramme für diese Plattformen zu erzeugen.

Will man ASAX für eine andere Plattform, wie FreeBSD oder Linux, auf 80X86 CPUs erzeugen lassen, so muss man noch ein wenig Programmierarbeit investieren, da für diese Plattformen keine funktionierende ASAX-IO-Schnittstelle existiert. Die von Haus aus mitgelieferte IO-Schnittstelle kann lediglich Auditdaten im NAD-Format lesen und schreiben. Man kann nun ein eigenständiges Formatadaptionsprogramm schreiben oder die Umwandlungsroutinen in das ASAX-System mit einbinden lassen, so dass die Formatumwandlung im System geschieht. Letzteres hätte den Vorteil, dass die Analyse auch online durchgeführt werden könnte, wenn man eine entsprechende Schnittstelle erstellt.

3.6.7.5 Die Benutzung von ASAX

Das lauffähige Analyseprogramm sei hier mit "`asax.exec`" bezeichnet. Der einfachste Weg "`asax.exec`" zu starten ist der Aufruf des Programmes in einem Terminal wie folgt:

```
myhost:~ >asax.exec my_audit.ddf main my_audit.nadf
```

Dies veranlasst das Starten von ASAX. ASAX lädt dann die Auditdatenbeschreibungsdatei, wertet sie aus und initialisiert Teile seines Speichers entsprechend. Dann lädt es die Datei "`main.asa`", das Postfix "`.asa`" ergänzt das Programm automatisch, und compiliert den darin enthaltenen Programmtext in einen Bytecode. Ist der Compiliervorgang erfolgreich, wird die NADF-Datei "`my_audit.nadf`" geöffnet, und die Analyse beginnt. Dabei wird die Datei mit den Auditdaten Datensatz für Datensatz nacheinander abgearbeitet, bis zum Dateiende. Abhängig von den RUSSEL-Anweisungen in dem Modul "main", welches durch die Datei "`main.asa`" repräsentiert wird, und den Modulen, die von dem Hauptmodul oder den folgenden Modulen angesprochen werden, und den Auditdaten, wird ASAX während der Arbeit Meldungen auf dem Bildschirm ausgeben und eventuell andere Befehle starten. Die eben beschriebene Arbeitsweise von ASAX bedeutet scheinbar eine "offline"-Analyse. Das muss jedoch nicht so sein. Es gibt verschiedene Wege, wie man die Ausgaben von beliebigen Programmen in den Eingabestrom anderer Programme umleiten kann. Die bekannteste Möglichkeit stellen die sogenannten Pipes dar. Dies sind Unix-Spezialdateien, die einen unidirektionalen Kommunikationsweg darstellen. Die Daten, die an so einer Pipe anliegen, werden nach dem FIFO-Prinzip einer nach dem anderen abgearbeitet. Ein Beispiel: Angenommen, es existiere ein Auditdatenumwandlungsprogramm, welches Auditdaten im NAD-Format für ASAX online, also in Echtzeit, erzeugt. Welcher Datenquellen sich das Programm bedient spielt bei dieser Betrachtung keine Rolle, aber es könnte ja ein Programm sein, das die Ausgabe des Programmes "`tcpdump`" in das NAD-Format wandelt. Dieses Programm dient ansonsten der Überwachung von Netzwerkschnittstellen, wie Ethernetkarten. Das Auditprogramm heiße "`tcpd2nadf`" und benutze zur Eingabe den Standardeingabekanal, zur Ausgabe den Standardausgabekanal.

"`tcpdump`" schreibt seine Ausgabe in den Standardausgabekanal. ASAX kann die Auditdaten jedoch nur aus einer Datei lesen, die mit Namen angegeben wird. Dann erzeugt man eine benannte pipe z. B. durch Aufruf des Befehls "`mknod .nadf-pipe p`". Und jetzt startet man folgende Befehlszeile:

```
tcpdump || tcpd2nadf > .nadf-pipe &
```

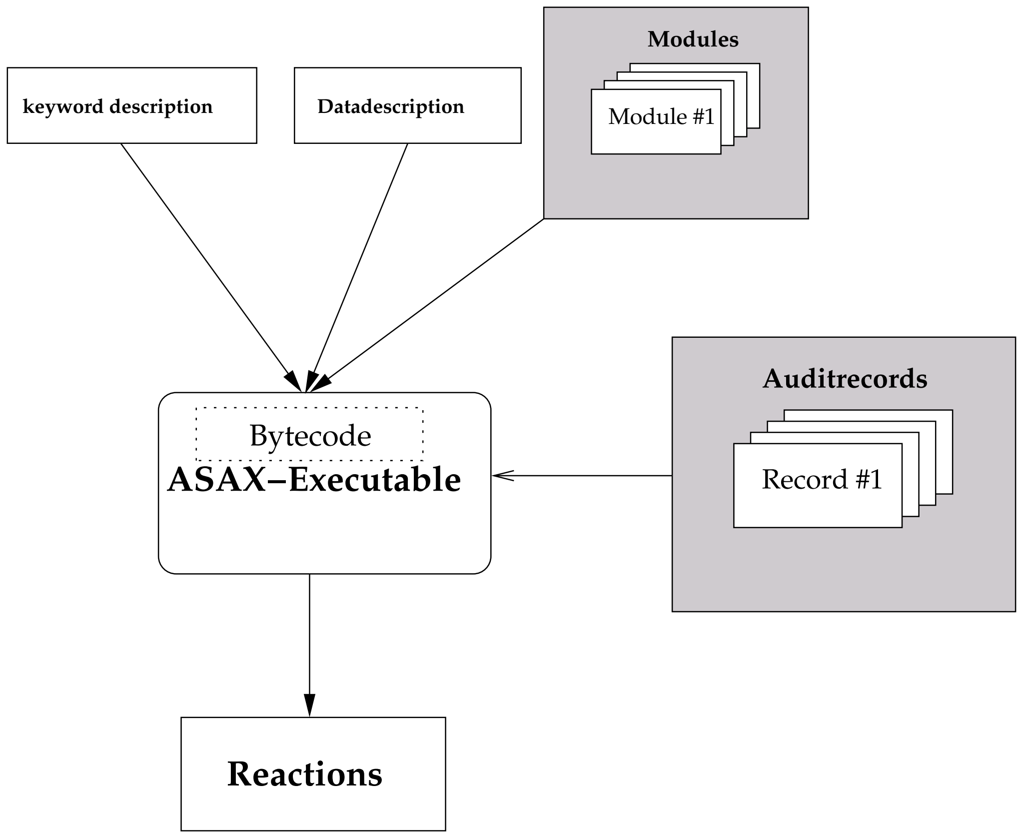

Abbildung 3.28: ASAX und verschiedene Komponenten, die von ASAX benötigt werden

Und dann noch:

```
asax.exec datadescription main .nadf-pip
```

Jetzt läuft die Analyse der Ausgabe von "`tcpdump`".

Bisher unerwähnt geblieben ist die Schlüsselwortbeschreibungsdatei (keyword description file). Diese Datei ist eine Art Wörterbuch, denn sie realisiert eine Abbildung von Wörtern zu Werten. Auf diese Weise lassen sich die Regeln in RUSSEL lesbarer schreiben, so kann man z. B. in RUSSEL einen Ausdruck in der Form `if UID=root` angeben. Nun muss aber der Wert für das Wort `root` festgelegt sein. Dies kann entweder innerhalb eines Moduls, oder aber in einer Schlüsselwortbeschreibungsdatei geschehen. Der genaue Aufbau der Schlüsselwortbeschreibungsdatei wird in [Mou] genau erklärt.

Kapitel 4

ANFORDERUNGEN AN EIN ID-SYSTEM

Im ersten Abschnitt dieses Kapitels werden zunächst einige allgemeine Anforderungen an ein ID-System aufgestellt.
Im Abschnitt 4.2 werden dann Untersuchungsergebnisse einiger ID-Systeme vorgestellt, die ausführlich in [vHK98] dokumentiert sind. Dabei wird auch angegeben auf welche Kriterien hin die Systeme untersucht worden sind. Die Kriterien stellen ein Klassifikationsschema dar, weil die ID-Systeme nach Erfüllung oder nicht-Erfüllung dieser Kriterien bewertet werden können. Als Beispielsysteme werden nur IDES, NIDES und EMERALD angegeben, weil diese Systeme in den vorangegangenen Abschnitten ausführlich beschrieben worden sind, und die Darlegung der Ergebnisse aller untersuchten ID-Systeme den Rahmen dieser Diplomarbeit sprengen würde.
In Abschnitt 4.3 wird abschliessend der Prototyp für eine ID-Funktionalitätsklasse vorgestellt, dass in [vHK98] aufgestellt worden ist.

4.1 Allgemeine Anforderungen

Wie schon mehrfach erwähnt, gründen die meisten ID-Systeme ihre Operationen auf die Analyse von Audit-Daten. Diese Daten können als eine „Fußspur“ der Systembenutzung über die Zeit angesehen werden. Durch die Analyse der Audit-Daten ist ein ID-System in der Lage zu entscheiden, ob ein Einbruch gegenwärtig vorliegt.
Ein „qualitatives“ ID-System sollte allgemein folgende Eigenschaften besitzen, unabhängig davon auf welchen Mechanismen es basiert :

1. Es muss ohne menschliche Aufsicht <u>kontinuierlich laufen</u>. Das ID-System muss zuverlässig genug sein, damit man ihm das Laufen im Hintergrund des zu überwachenden Systems erlauben kann. Es sollte nicht eine Art „black

box“ darstellen, d.h. die interne Arbeitsweise muss von außen überprüfbar sein.

2. Es muss fehlertolerant sein, d.h. es muss Systemcrashs überleben und seine Wissens-Basis nicht beim *restart* wieder neu aufbauen müssen.
3. Es sollte auf dem System nur einen minimalen overhead verursachen. Ein ID-System, welches einen Computer zu einem „Kriechtempo“ verlangsamt, ist praktisch nicht einsetzbar.
4. Es muss Abweichungen vom normalen Verhalten bemerken.
5. Es muss leicht an das, in Frage kommende System, angepasst werden können, d.h. es muss systemunabhängig sein. Jedes System hat ein anderes Benutzungs-Muster, daher sollten die Verteidigungsmechanismen leicht an diese Muster angepasst werden können.
6. Es muss mit verändertem Systemverhalten über die Zeit, wie das Hinzufügen von neuen Applikationen, zurechtkommen können, d.h. das ID-System muss anpassungsfähig sein.
7. Es muss schwer zu überlisten sein.

Der letzte Punkt wirft ein Problem mit gewissen Fehlern-Typen auf, die üblicherweise im System vorkommen. Diese können als, *falsch-positiv-Fehler, falsch-negativ-Fehler* und *Subversions-Fehler*, kategorisiert werden.

Falsch-positiv-Fehler : Ein *falsch-positiv-Fehler* kommt vor, wenn das System eine Aktion als anomal klassifiziert (ein möglicher Einbruch), obwohl es eine legitime Aktion ist. *Falsch-positiv-Fehler* können die Benutzer des ID-Systems dazu verleiten, dessen outputs zu ignorieren sowie legitime Aktionen als Einbrüche zu klassifizieren. Das Auftreten dieses Fehlertyps sollte daher minimiert werden (es wird nicht möglich sein ihn vollständig zu eliminieren), um den Operatoren brauchbare Informationen zur Verfügung stellen zu können. Wenn zu viele *falsch-positiv-Fehler* generiert werden, würden die Operatoren mit der Zeit dazu übergehen, die Ausgaben des Systems zu ignorieren und somit auch einen aktuellen Einbruch, obwohl dieser vom System entdeckt wurde.

Falsch-negativ-Fehler : Ein *falsch-negativ-Fehler* tritt auf, wenn eine aktuelle „aufdringliche“ Aktion vorliegt, aber das System sie als ein „nicht-aufdringliches“ Verhalten gelten lässt. *Falsch-negativ-Fehler* sind schwerwiegender als *falsch-positiv-Fehler*, weil sie ein irreführendes Sicherheitsgefühl hervorrufen. Allen Aktionen das Fortfahren zu erlauben führt dazu, dass eine verdächtige Aktion nicht die Aufmerksamkeit des Operators auf sich zieht. Das ID-System ist somit eher eine Belastung, denn der Sicherheitszustand des Systems ist nun niedriger als vor der Installation des Einbruch-Detektors.

Subversions-Fehler : Ein *Subversions-Fehler* liegt dann vor, wenn ein Einbrecher die Operationen des Einbruch-Detektors modifiziert, um das Auftreten eines *falsch-negativ-Fehlers* zu erzwingen. *Subversions-Fehler* sind komplexer und stehen in Beziehung zu *falsch-negativ-Fehlern*. Ein Einbrecher kann das Wissen über das Innere des ID-Systems dazu benutzen, um dessen Funktionen zu ändern (z.B. um anomalen Aktionen das Weiterlaufen zu erlauben). Der Einbrecher kann dann die operationalen Sicherheitsbarrieren des Systems durchbrechen. Dies kann unter Umständen von einem Operator, der die Logs des Einbruch-Detektors überprüft, entdeckt werden, aber es würde trotzdem den Anschein erwecken, als arbeite das ID-System noch korrekt. Eine andere Form von *Subversions-Fehler* ist das Überlisten des Systems im Verlauf der Zeit (auch als „zum Narren halten" bezeichnet). Während das ID-System das Systemverhalten über die Zeit hinweg beobachtet, kann es möglich sein, Operationen durchzuführen, von denen jede für sich genommen keine Bedrohung darstellt, aber die in ihrer Gesamtheit die Integrität des Systems bedrohen können. Dies ist aus folgenden Gründen möglich : Zunächst einmal überprüft das ID-System kontinuierlich, ob seine Annahme über die normale Benutzung des Systems zutrifft. Es wird damit gerechnet, dass im Verlauf der Zeit in den Systembenutzungs-Mustern Veränderungungen auftreten, und das ID-System muss mit diesen Veränderungen zurechtkommen. Wenn aber ein Einbrecher Aktionen in verschiedenen Zeitabschnitten durchführen kann, die nur geringfügig außerhalb der normalen Systembenutzungs-Muster liegen, dann ist es möglich, dass sie als legitime Aktionen akzeptiert werden, obwohl sie, jede einzeln, in Wirklichkeit einen Teil eines Einbruchsversuches bilden.

4.2 Ein Klassifikationsschema für vorhandene ID-Systeme

Um speziellere Anforderungen an ID-Systeme stellen zu können, wäre es angebracht, zunächst die vorhandenen ID-Systeme zu klassifizieren und auf einige Aspekte hin zu untersuchen. Die daraus resultierenden Ergebnisse können Aufschluss darüber geben, welche Anforderungen, sowohl technisch als auch organisatorisch, erfüllbar oder Notwendig sind.
In [vHK98] wurde der Versuch unternommen, für die wichtigsten ID-Systeme, die auf dem Markt vorhanden sind, die in der Forschung aktuell erprobt werden oder die von besonderer Bedeutung sind, ein Klassifikationsschema zu entwickeln, aus dem die wesentlichen Merkmale der ID-Systeme hervorgehen. Zu diesem Zweck

wurden die ID-Systeme auf folgende Aspekte hin untersucht [vHK98] :

- Praktische Einsetzbarkeit, insbesondere ihre Kooperation mit Firewalls
- Skalierbarkeit in Bezug auf größere Netze und neue Angriffsszenarien
- Wie wurden die Forschungsergebnisse aus dem ID-Bereich umgesetzt
- Sicherheitsrelevante Aspekte, wie z.B. die Modifikation der IDS Konfiguration
- Grenzen des ID-Systems

Die Klassifikation wurde dabei in Sektionen unterteilt [vHK98] :

- **Architekturaspekte :** Die Architektur des ID-Systems übt großen Einfluß auf die Performance und Erweiterbarkeit der ID-Systeme aus. Auch die Skalierbarkeit wird dadurch beeinflusst, denn ein IDS mit verteilter Datensammlung ist skalierbarer als eines, das seine Daten von zentraler Stelle aus sammelt.
- **Systemspezifische Aspekte :** Hierbei kommen Fragen nach Verfügbarkeit, Kostenfaktor, Einsetzumgebung (Betriebssysteme, Rechner) und der zusätzlich notwendigen Hardware der ID-Systeme zum Tragen.
- **Angriffe :** Art und Typ der erkannten Angriffe spielen bei der Qualitätsbewertung eines ID-Systems eine entscheidende Rolle. Wichtig ist auch, wie effektiv die Erkennung von Angriffen ist, die von einer Firewall nicht erkannt wurden und wie gut sich das IDS selbst schützen kann.

Diese drei Sektionen stellen einige grobe Klassifikationskriterien für ID-Systeme dar und können weiter verfeinert werden, um genauere Klassifikationskriterien daraus abzuleiten. Mit Hilfe der in [vHK98] durchgeführten Verfeinerung werden im Folgenden einige Eigenschaften der in dieser Diplomarbeit erwähnten ID-Systeme spezifiziert.

4.2.0.6 Untersuchungskriterien

1. **Angriffsszenarien :** Nach [vHK98] können die aktuell verfügbaren ID-Systeme nur eine begrenzte Teilmenge der möglichen Angriffe erkennen. Dabei sind die Stärken und Schwächen im Erkennen von Angriffen von System zu System verschieden. Diese Unterschiede sind im Wesentlichen auf die Art der implementierten Datensammlungskomponenten zurückzuführen. Die Angriffe, auf die in diesem Kapitel Bezug genommen wird, und die eine Teilmenge der in Kapitel 2 vorgestellten Angriffe darstellen, sind folgende :

- Abtastende Suche nach TCP/UDP/RPC-Diensten (mit Half Open Scans und Stealth Scans)
- Spoofing (IP, DNS, WEB etc.)
- Denial-of-Service-Angriffe (SYN Flooding, PING etc.)
- Routing Angriffe (SYN Flooding PING etc.)
- Versuchte Einbrüche (Erraten des Passworts, Erraten des NIS file handles)
- Angriffe auf bekannte Sicherheitslöcher (sendmail, NIS, NFS, bind, etc.)
- Software mit Schadensfunktion (Viren, Würmer etc.)
- Angriffe über WWW-Dienste (ActiveX, Java, MIME etc.)

Die Erkennung/Nicht-Erkennung dieser Angriffe durch einige ID-Systeme, wird in späteren Abschnitten tabellarisch dokumentiert.

2. **Audit-Daten :** Die Stärke eines IDS steht im direkten Zusammenhang mit dem Typ der verwendeten Audit-Daten. Wenn bespielsweise nur Protokolldaten des Betriebssystems verwendet, also analysiert werden, können bestimmte Angriffe nicht erkannt werden (z.B. IP-Spoofing).
Die Bearbeitung von Audit-Daten erfolgt in einigen ID-Systemen gemäß einer Abstraktionshierarchie. Die Daten, die auf Netzebene von einem Sniffer gesammelt wurden, werden zur Repräsentation auf eine höhere Abstraktionsebene überführt. Ziel dieses Verfahrens ist die Reduzierung der zu analysierenden Datenmenge. Bei ID-Systemen, die solch ein Abstraktionverfahren einsetzen, ist nicht nur die Quelle, aus der die Daten stammen (z.B. Sniffer) wichtig, sondern auch welches Ziel sie haben (z.B. Expertensysteme). Die Granularität der gesammelten Daten ist ein weiterer wichtiger Aspekt. Zur Erkennung von Angriffen auf unteren Protokollebenen sind genügend detaillierte Daten erforderlich, d.h. hier muss die Granularität der Daten sehr fein sein.
Die Fähigkeit eines IDS, Daten aus verschiedenen Quellen verarbeiten zu können erhöht die Praxistauglichkeit dieses Systems. Als Quellen können z.B. die Audit-Daten-Sammlungs-Komponenten des beobachteten Ziel- oder eines andern ID-Systems in Frage kommen. Auch Firewalls stellen Datenquellen dar, wenn ID-Systeme als eine ergänzende Schutzmaßnahme neben Firewalls eingesetzt werden. Wichtig ist dabei, dass sowohl die Sammlungs-Komponente, als auch die Analyse-Komponente ein gemeinsames Datenformat verarbeiten können.

3. **Datenanalyse :** Der tatsächliche Erkennungsprozess der Angriffe findet in der Analysekomponente des IDS statt. In Abschnitt 3.2 wurde erwähnt,

dass es dabei zwei unterschiedliche Erkennungstechniken gibt : *Missbrauchserkennung* (auch als *Signaturanalyse* bezeichnet), die das Ziel verfolgt bekannte Angriffe zu identifizieren, und *Anomalieerkennung* mit dem Ziel, die unmittelbaren Folgen eines Angriffs zu erkennen. Zur Datenanalyse können, nach [vHK98], folgende Ergänzungen und Klassifikationskriterien, die im Wesentlichen die Vor- bzw. Nachteile der Missbrauchs- und Anomalieerkennung darlegen, angegeben werden :

- Durch die Datenanalyse sollen Verletzungen der Sicherheitsvorgaben, nach Möglichkeit vor Entstehung von Schäden durch den Angriff, entdeckt werden. Die beiden ID-Methoden bieten dafür keine endgültige Lösung weil,

  (a) der Erfolg bei **Missbrauchserkennung** sich auf das Wissen über bekannte und erforschte Angriffe gründet. Die Wartung der Signaturdatenbank muss regelmäßig durchgeführt werden. Wenn ein Anwender die Erkennungs-Muster selbst definieren will, so muss er laufend die Informationsquellen über neue Angriffsarten verfolgen. Auch Variationen bekannter Angriffe könnten von der Missbrauchs-Komponente nicht erkannt werden. Die Missbrauchsanalyse erfordert, dass die Audit-Daten vollständig und in richtiger Reihenfolge vorhanden sind. Bei nicht richtig zusammengebauten TCP/IP-Paketen kann es z.B. passieren, dass Angriffe nicht erkannt werden.

  (b) unbekannte Angriffe mittels **Anomalieerkennung** nur dann erkannt werden, wenn sie ein normabweichendes Verhalten erzeugen. *Anomalieerkennung* bietet, zumindest vom Ansatz her, die Möglichkeit das System vor noch unbekannten Angriffen zu schützen. Aber auch hier müssen die Anomalieerkennungs-Komponenten gewartet werden, um Falsch-Alarme vorzubeugen und Angriffe korrekt zu erkennen. Angreifer können, wenn sie nicht von dem als „normal“ definierten Verhalten abweichen, ohne von der Anomalieerkennung enttarnt zu werden, ihre Aktionen ausführen. Das Anomalieerkennungs-System muss in der Lage sein, sich dynamisch an neues Benutzerverhalten anzupassen. Während diese Phase sollte das ID-System das „Angriffsverhalten“ nicht als normal lernen. In [vHK98] wird darauf hingewiesen, dass

  > *„... Anomalieerkennung alle Vor- und Nachteile statistischer Inferenz und induktiver Techniken ererbt, wie z.B. falsche Vorhersagen zu treffen. Dies könnte dazu führen, die Zweckmäßigkeit des ID-Systems in Frage zu stellen, weil es unzuverlässig arbeitet.“*

  Aus diesem Grund wäre bei Anomalieerkennung eine Angabe über die Höhe der Fehlerwahrscheinlichkeit der ermittelten Ergebnisse von großem Nutzen. Mathematische Modelle mit deren Hilfe dieser Art Aussagen gemacht werden können lassen sich sehr schwer auf Anomalieerkennung übertragen,

> „..*weil es keine brauchbare mathematische Definition eines Angriffes gibt* . “ [vHK98]

4. **Architektur :** Man kann die Arbeitsweise von ID-Systemen grob in drei Arbeitsschritte unterteilen: *Datensammlung, Datenanalyse, Ergebnisdarstellung.* Ein ID-System besitzt demnach auch drei Hauptkomponenten : *Datensammlungs-Komponente, Analyse-Komponente und Ergebnisdarstellungs-Komponente*
   In Bezug auf die ersten beiden Komponenten wurde in [vHK98] eine weitere Unterteilung vorgenommen :

   (a) Das gesamte Rechnernetz wird von <u>einer</u> zentralen Datensammlungs-Station überwacht, die ihre Beobachtungen an <u>eine</u> zentrale Analysekomponente weiterreicht.
      - Datensammlung : **zentral**
      - Datenanalyse : **zentral**

   (b) <u>Mehrere</u> Monitore (Sammelstationen) geben ihre Daten an <u>eine</u> zentrale Analysestation weiter.
      - Datensammlung : **verteilt**
      - Datenanalyse : **zentral**

   (c) <u>Mehrere</u> Monitore geben ihre Daten an <u>verschiedene</u> Analysestationen weiter.
      - Datensammlung : **verteilt**
      - Datenanalyse : **verteilt**

   (d) Analysedaten werden lokal aufbereitet und dann an eine zentrale Analysestation weitergereicht. Man spricht in diesem Fall von einer **hybriden** Datenanalyse.
      - Datensammlung : **zentral oder verteilt**
      - Datenanalyse : **verteilt und zentral**

5. **Benachrichtigung :** Bei Entdeckung eines möglichen Angriffs kann der SSO auf verschiedene Arten vom System benachrichtigt werden ([vHK98]) :

   - Der Sicherheitsbeauftragte wird über größere Entfernungen mit Hilfe von Pager-Diensten und Mobil Telefonen benachrichtigt.

- Benachrichtigung erfolgt über das interne Netz.
  Voraussetzung : Die Verbindungen zwischen den beteiligten Stationen ist kryptographisch gesichert und beteiligte Systeme sind vor *Denial-of-Service-Angriffen* geschützt.
- Alarmmeldungen an ein entferntes System erfolgt über ein gesichertes Netzsegment (z.B. einen dedizierten, seriellen Anschluss).
- Alarm werden lokal ausgelöst.
  Voraussetzung : Ständige Beobachtung der IDS-Konsole durch das Sicherheitspersonal

Andere wichtige Aspekte in diesem Zusammenhang sind :

- Der **Bericht** und die **Angriffsprotokollierung :** Sämtliche Informationen und Aktivitäten, die zu einem erfolgreichen Angriff geführt haben, sollten aufgezeichnet werden. Dies ist für das Auffinden von Sicherheitsschwächen in einem Rechner oder Rechnernetz notwendig.

In später folgenden Abschnitten findet sich eine Auflistung der erfüllten, oder nicht erfüllten, Klassifikationskriterien, bezogen auf die ID-Systeme **IDES, NIDES** und **EMERALD**. Der Eintrag **Angriffsprotokollierung** wird dabei mit **„JA“** gekennzeichnet, falls das IDS in der Lage ist, dem Angriff die richtigen Daten zuzuordnen.

6. **Reaktion :** Die meisten, verfügbaren, ID-Systeme beinhalten bereits *Intrusion Response-Komponenten* (IR-Systeme). Diese können im Angriffsfall selbstständige Gegenmaßnahmen durchführen, indem sie z.B. die Ports blockieren und/oder die Router rekonfigurieren.

7. **Benutzerschnittstelle :** Eine wichtige Voraussetzung für die effiziente Bedienung eines ID-Systems, ist eine graphische Benutzeroberfläche (Graphical User Interface, GUI). Die Handhabung über eine zeichenorientierte Konsole durch den SSO wäre wegen der vielen Parameter, die ein ID-System besitzt, nicht möglich. Eine GUI ist auch für die schnelle Einarbeitung des zuständigen Personals in die Bedienung des Systems von Bedeutung.

8. **Thumbprinting :** Thumbprinting erlaubt die Identifizierung von Benutzern innerhalb eines Netzes, obwohl sich ihre User-ID (UID) geändert hat. Beim Thumbprinting wird dem Benutzer eine Netz-ID (NID) zugewiesen. Dies erschwert die Durchführung eines Angriffs.

9. **Zielumgebung :** Dies ist der Einsatzbereich, bzw. die Einsatzumgebung des ID-Systems. Gefragt wurde der Untersuchung, für welche Einsatzumgebung sich das ID-System eignet (z.B. einzelner Arbeitsplatz oder grosse Netze).

10. **Kooperation mit Firewalls :** In Bezug auf die Firewalls wurde folgenden Fragen nachgegangen [vHK98] :

    - In welchem Umfang ist das IDS in der Lage, die von einer Firewall produzierten Audit-Daten zu verarbeiten ?
    - In welchem Umfang kann das IDS neben einer Firewall eingesetzt werden, ohne dass sich beide Systeme gegenseitig verhindern ?
    - Welche Firewall-Produkte werden unterstützt ?
    - Inwieweit können Firewall und IDS kooperieren, so dass redundante Analysen des Datenstroms (d.h. Datendurchsatzes) vermieden werden können ?
    - Inwieweit ist das IDS in der Lage, als Gegenmaßnahme zu einem Angriff die Firewall zu rekonfigurieren ?

11. **Erweiterbarkeit/Programmierschnittstelle :** Es geht hierbei um die Beschreibung von Möglichkeiten, die zur Erweiterung des IDS mit neuen Angriffssignaturen, eingesetzt werden können. In diesem Zusammenhang ergeben sich nach [vHK98] folgende Fragen :

    - Kann der Anwender eigene Signaturen selbst entwickeln und diese anschließend einbinden ?
    - Wie häufig wird die Signaturdatenbank durch den Hersteller aktualisiert, und welche Kosten entstehen bei einem solchen Update ?
    - Wie gut kann das IDS an die Zielumgebung angepasst werden ?
    - Ist es möglich, systemspezifische Informationen, benutzerspezifische Informationen und/oder lokale Sicherheitsrichtlinien (z.B. : Benutzer sollen ActiveX nicht verwenden) zu hinterlegen ?

12. **Performance :** Es wurden in [vHK98] zwei Performanzaspekte berücksichtigt :

    (a) **Analyse :** Wie schnell kann ein Angriff entdeckt werden (ist eine *Echtzeit-Analyse möglich*?
    Ist das System dazu in der Lage Angriffe vorherzusagen ?

    (b) **Gesamtsystem :** Wie hoch ist die Belastung des Gesamtnetzes, bzw. der Rechner, durch das ID-System ?

13. **Adaptivität :** Unterschieden wird zwischen zwei Arten der Adaptivität.

(a) **Anpassung an das Benutzerverhalten :** Dies betrifft in erster Linie die Anomalieerkennung. Es geht hierbei um die Frage, wie schnell das System dazu in der Lage ist zu lernen, einen Angriff von neuen, erlaubten Benutzerverhaltensmustern zu unterscheiden, die z.B. aufgrund veränderter Aufgabenbereiche entstehen können. Der administrative Aufwand, der für die Umstellung des ID-Systems betrieben werden muss, um es an die neue Ausgangslage anzupassen, ist auch ein wichtiges Bewertungskriterium.

(b) **Automatisches Wechseln der Abstraktionsebene :** Gefragt wird in diesem Zusammenhang nach der Fähigkeit des ID-System die Granularität der Audit-Daten, im Falle eines Angriffes, automatisch zu verfeinern, um die Informationsausbeute für den entprechenden Angriff zu erhöhen.

14. **Datenschutzrechtliche Aspekte :** Bei der Protokollierung fällt eine erhebliche Menge an personenbezogenen Daten an. Ein wichtiger Aspekt ist daher der Umgang mit diesen sensitiven Daten. In [vHK98] wurde in diesem Zusammenhang auf folgende Fragen Wert gelegt :

    - Was geschieht mit den Benutzerdaten ?
    - Haben sich die Entwickler des IDS Gedanken zum Datenschutz gemacht ?
    - Inwieweit ist ein ehrlicher Benutzer vor dem Sammlungseifer des IDS geschützt ?
    - Werden die Daten anonymisiert und wann geschieht das ?

15. **Systembeschreibung :** Dies betrifft die genaue Beschreibung des IDS. In dem, in [vHK98] beschriebenen Projekt, wurden keine eigenen Tests mit der Software durchgeführt. Die Informationen zu den Systemen wurden aus Herstellerunterlagen, Vergleichstests, technischen Berichten etc. abgeleitet. Auf die Systembeschreibung wird in diesem Kapitel nicht eingegangen, weil die Systembeschreibung von IDES, NIDES und EMERALD bereits in Kapitel 3 vorgenommen worde.

16. **Format/Semantik der Ein-/Ausgabedaten :** Hier wird der Frage nachgegangen welche Audit-Daten analysiert werden, welches Format sie haben und welche Bedeutung ihre Einträge besitzen.

17. **Verfügbarkeit :** Ein wichtiger Aspekt war die Verfügbarkeit der untersuchten Systeme, d.h. ob es sich bei dem vorliegenden IDS um ein kommerzielles Produkt oder um einen universitären Prototyp handelte.

18. **Systemanforderungen :** Die Frage, die in diesem Zusammenhang gestellt wurde, betrifft die technischen Voraussetzungen (Hardware, Software), die notwendig sind, um das IDS in Betrieb nehmen zu können. Dies impliziert auch die Frage nach organisatorischen und finanziellen Voraussetzungen, die für den Einsatz des ID-Systems erfüllt werden müssen.

19. **Sicherheit :** Hauptaugenmerk liegt hier auf der Sicherheit des IDS selbst. D.h. nach [vHK98] :

    - Wie gut schützt sich das IDS selbst gegen Angriffe ?
    - Können ID-Komponenten von einem Angreifer durch andere Komponenten ersetzt werden ?
    - Wie sicher läuft die Kommunikation zwischen einer Datensammlungsstation und der Analysestation ab ?

20. **Dokumentation :** Gibt es ausreichend gute Dokumente, mit denen die Funktionsweise des ID-Systems nachvollzogen werden kann ? Dies ist für die Bedienbarkeit des ID-Systems von grosser Bedeutung.

Die Folgenden Tabellen zeigen inwieweit die oben aufgelisteten Kriterien auf die ID-Systeme IDES, NIDES und EMERALD zutreffen, und ob die gestellten Anforderungen von diesen Systemen erfüllt werden. In den Tabellen wird folgende Notation benutzt [vHK98] :

JA : Merkmal ist bei dem entsprechenden IDS verfügbar.

NEIN : Merkmal ist bei dem entsprechenden IDS nicht verfügbar.

N.A. : Not Addressed.

NEIN* : Das IDS verfügt im Lieferzustand (out-of-the-box) nicht über dieses Merkmal, kann jedoch entsprechend konfiguriert werden.

Zur Ermittlung der Systemprofile wurden in [vHK98] folgende Quellen benutzt :

- Konferenzbeiträge
- White Papers und Produkt-Information des Herstellers
- Technische Berichte
- Begutachtete Zeitschriftenartikel
- Fragebogen an die Hersteller

4.2.1 EMERALD

Angriffsszenarien :

Abtastende Suche nach TCP/UDP/RPC-Diensten (mit Half Open Scans und Stealth Scans)	**NEIN***
Spoofing (IP, DNS, WEB etc.)	**JA**
Denial-of-Service-Angriffe (SYN Flooding PING etc.)	**JA**
Routing Angriffe (Source Routing,ICMP Redirect)	**NEIN***
Versuchte Einbrüche (Erraten des Passwort, Erraten des NIS File Handles)	**N.A.**
Angriffe auf bekannte Sicherheitslöcher (sendmail, NIS, NFS, bind, etc.)	**N.A.**
Software mit Schadensfunktion (Viren, Würmer etc.)	**N.A.**
Angriffe über WWW-Dienste (ActiveX, Java, MIME etc.)	**NEIN***
Erkennungsstärken des IDS	Die EMERALD-Architektur erlaubt es, heterogene Angriffe sowie Normabweichungen des Benutzerverhaltens in Bezug auf Zuverlässigkeit, Sicherheit etc. zu erkennen. Tatsächliche Signaturen und Profile wurden in diesem Zusammenhang jedoch nicht erstellt.

Tabelle 4.1: Angriffstypen [vHK98]

Auditdaten :	
Quelle :	Benutzerlogdaten, Applikationslogdaten, Datagramme, SNMP Verkehr, externe Monitore (vgl. Systembeschreibungen), Daten anderer ID-Systeme
Ziel :	Monitore (vgl. 3.6.6)
Granularität :	fein - grob
Datenanalyse :	
Rechnerebene :	JA
Netzebene :	JA
Signaturanalyse :	JA, Signature Engine (vgl. Systembeschreibung)
Anomalieerkennung :	JA, Profiler Engine
statistisch :	JA
logisch :	JA
weitergehende Analyse :	NEIN
Architektur :	
Datensammlung :	verteilt
Datenanalyse :	hybrid
Kommunikation :	asynchron
Reaktion :	
Benachrichtigung :	Der „Resolver" versendet hierarchisch geordnete Benachrichtigungen
Bericht :	JA, verschiedene Formen
Maßnahmen :	automatische
Angriffsprotokollierung :	JA
Benutzerschnittstelle :	JA, graphisch
Thumbing :	JA
Zielumgebung :	Große Firmennetze, Zusatzwerkzeug zu anderen zentralisierten ID-Komponenten
Zusammenarbeit mit Firewalls :	SunScreen, andere möglich

Erweiterbarkeit/ Programmierschnittstelle :	API verfügbar, Protokoll-spezifikation „fast fertig“
Performanz :	
Analyse :	Echtzeit
Gesamtsystem :	N.A.
Adaptivität :	
Anpassen an das Benutzerverhalten :	JA (keine weitere Informationen)
Automatischer Wechsel der Abstraktionsebene :	JA (keine weitere Information)
Datenschutzrechtliche Aspekte :	N.A.
Format/Semantik der Ein-/Ausgabedaten :	In [vHK98] wurde bekannt gegeben, dass die Dokumentation des Common Intrusion Detection Format (CIDF) in Kürze veröffentlicht werden sollte, was inzwischen der Fall sein sollte
Verfügbarkeit :	EMERALD ist zum Zeitpunkt dieser Untersuchung (1998) noch in der Entwicklung. Das Projekt sollte bis 1999 gefördert werden.
Systemanforderungen :	SUN OS oder Solaris
Sicherheit :	Kryptographische Authentisierung

4.2.2 IDES

Angriffsszenarien :

Abtastende Suche nach TCP/UDP/RPC-Diensten (mit Half Open Scans und Stealth Scans)	**N.A.**
Spoofing (IP, DNS, WEB etc.)	**N.A.**
Denial-of-Service-Angriffe (SYN Flooding PING etc.)	**N.A.**
Routing Angriffe (Source Routing,ICMP Redirect)	**N.A.**
Versuchte Einbrüche (Erraten des Passwort, Erraten des NIS file Handles)	**N.A.**
Angriffe auf bekannte Sicherheitslöcher (sendmail, NIS, NFS, bind, etc.)	**N.A.**
Software mit Schadensfunktion (Viren, Würmer etc.)	**N.A.**
Angriffe über WWW-Dienste (ActiveX, Java, MIME etc.)	**N.A.**
Erkennungsstärken des IDS	**N.A.**

Tabelle 4.2: Angriffstypen [vHK98]

Auditdaten :

Quelle :	SUN OS Auditdaten, SUN C2 Auditdaten, UNIX Accounting
Ziel :	Realm Schnittstelle (diese konvertiert die reinen Auditdaten in das IDES Auditdaten-Format)
Granularität :	mittel

Architektur :

Datensammlung :	verteilt
Datenanalyse :	zentral
Kommunikation :	RPC über TCP/IP-Kanal

Datenanalyse :	
Rechnerebene :	JA
Netzebene :	NEIN
Signaturanalyse :	JA
Anomalieerkennung :	JA
statistisch :	JA
Methode :	N.A.
Meßdaten :	CPU-Last,I/O-Last,Durchsatz Mailverkehr, Durchsatz Netzaktivität.
logisch :	JA(XPS-gestützte Anomalie-erkennung)
weitergehende Analyse :	N.A.
Reaktion :	
Benachrichtigung :	JA
Bericht :	JA
Maßnahmen :	N.A.
Angriffsprotokollierung :	N.A.
Benutzerschnittstelle :	X-Windows, benutzerdefinierbare Kategorien (View)
Thumbprinting :	N.A.
Zielumgebung :	N.A.
Zusammenarbeit mit Firewalls :	N.A.
Erweiterbarkeit / Programmierschnittstellen :	Schnittstellenbibliothek Signaturdatenbank
Performanz :	
Analyse :	Echtzeit
Gesamtsystem :	auf dediziertem Rechner
Adaptivität :	
Anpassen an das Benutzerverhalten :	N.A.
Automatischer Wechsel der Abstraktionsebene :	N.A.
Datenschutzrechtliche Aspekte :	N.A.

Format/Semantik der Ein-/Ausgabe :	Die Eingabedaten enthalten Informationen über die CPU-Last, Dateizugriffe, Netzdurchsatz etc.
Verfügbarkeit :	Verschiedene Betaversionen sind verfügbar (1992),werden jedoch nicht mehr unterstützt.
Systemanforderungen :	SUN OS 4.0.x
Einschränkungen :	N.A.
Sicherheit :	N.A.
Bemerkungen :	Das Werkzeug wird nicht mehr unterstützt. Zu IDES und NIDES sind viele Forschungsberichte verfügbar.

4.2.3 NIDES

Abtastende Suche nach TCP/UDP/RPC-Diensten (mit Half Open Scans und Stealth Scans)	**NEIN***
Spoofing (IP, DNS, WEB etc.)	**NEIN***
Denial-of-Service-Angriffe (SYN Flooding PING etc.)	**NEIN***
Routing Angriffe (Source Routing,ICMP Redirect)	**NEIN***
Versuchte Einbrüche (Erraten des Passwort, Erraten des NIS file Handles)	**NEIN***
Angriffe auf bekannte Sicherheitslöcher (sendmail, NIS, NFS, bind, etc.)	**NEIN***
Software mit Schadensfunktion (Viren, Würmer etc.)	**NEIN***
Angriffe über WWW-Dienste (ActiveX, Java, MIME etc.)	**NEIN***
Erkennungsstärken des IDS	Eine Menge von 39 Regeln steht für SUN UNIX zu Verfügung.

Tabelle 4.3: Angriffstypen [vHK98]

Auditdaten :

Quelle :	propritäres generisches Auditformat
Ziel :	N.A.
Granularität :	N.A

Architektur :

Datensammlung :	verteilt
Datenanalyse :	zentral
Kommunikation :	N.A.

Datenanalyse :	
Rechnerebene :	JA
Netzebene :	NEIN
Signaturanalyse :	JA (falls entsprechende Regeln vorhanden)
Anomalieerkennung :	JA
statistisch :	JA
Methode :	X^2 ähnlich
logisch :	JA (die Regelmenge)
weitergehende Analyse :	N.A.
Reaktion :	
Benachrichtigung :	JA, Systemadministrator wird per E-Mail benachrichtigt
Bericht :	N.A.
Maßnahmen :	N.A.
Angriffsprotokollierung :	N.A.
Benutzerschnittstelle :	X-Windows, kontextsensitive Hilfe
Thumbprinting :	N.A.
Zielumgebung :	N.A
Zusammenarbeit mit Firewalls :	N.A.
Erweiterbarkeit / Programmierschnittstellen :	NIDES kann mittels des generischen Auditdatenformats an viele Systeme angepasst werden.
Performanz :	
Analyse :	Echtzeit oder Batch-Betrieb
Gesamtsystem :	Echtzeit oder Batchbetrieb
Adaptivität :	
Anpassen an das Benutzerverhalten :	JA
Automatischer Wechsel der Abstraktionsebene :	N.A.
Datenschutzrechtliche Aspekte :	N.A.

Format/Semantik der Ein-/Ausgabe :	N.A.
Verfügbarkeit :	Prototyp im Beta Test (1994), nicht weiter unterstützt, nicht verfügbar
Systemanforderungen :	SUN OS 4.1.x oder Solaris 1.1 SUN BSM oder C2 auditing
Einschränkungen :	N.A.
Sicherheit :	N.A.
Bemerkungen :	Nach Informationen der Entwickler ist NIDES kaum verfügbar. EMERALD verfügt jedoch über den vollen Funktionalitätsumfang von NIDES. Auch der Prototyp von NIDES wird nicht weiter unterstützt.

4.3 Ein Prototyp für eine ID-Funktionalitätsklasse

In diesem Abschnitt werden die Hauptanforderungen an ein ID-System beschreiben, aus denen sich eine Liste von funktionalen Merkmalen ableiten lässt. Mit Hilfe dieser Liste und den Anforderungen zur Vertrauenswürdigkeit eines ID-Systems, wird in [vHK98] der Versuch unternommen, eine Funktionalitätsklasse gemäß ITSEC[1] zu erstellen.
Die Erkennung von Angriffen auf zu schützende Ressourcen stellt eine Grundanforderung dar. Diese Anforderung dient den übergeordneten Zielen ([vHK98]),

- die Verfügbarkeit der vom IDS zu schützenden IT-Dienste sicherzustellen,
- die Integrität sowie,
- die Verfügbarkeit der im zu schützenden System gespeicherten Informationen zu garantieren.

In [vHK98] wird ein Prototyp für die Funktionalitätsklasse vorgeschlagen, der sechs Kategorien umfaßt. Die Kategorien werden in dieser Diplomarbeit mit *ID-F1* bis *ID-F6* kenntlich gemacht.

- **ID-F1 :** Sicherheitsaudit
- **ID-F2 :** Wartungsunterstützung
- **ID-F3 :** Identifizierung und Authentifizierung
- **ID-F4 :** Schutz der Sicherheitsfunktionen
- **ID-F5 :** Vertraulichkeit personengebundener Daten
- **ID-F6 :** Verfügbarkeit der Dienste

Jede Klasse enthält Funktionalitätsbeschreibungen, die zum Erreichen des erfolgreichen Einsatzes eines ID-Systems notwendig sind. Im Folgenden wird die Funktionalitätsklasse und deren Kategorien näher beschreiben.

[1] *„Obwohl sich der Anforderungskatalog zur Definition einer Funktionalitätsklasse gemäß ITSEC eignet und sich an den CC orientiert,stellt er aber noch keine Funktionalitätsklasse dar.“*[vHK98] .

4.3.1 ID-F1 Sicherheitsaudit

In [vHK98] werden für das Sicherheitsaudit folgende Anforderungen aufgestellt :

1. Das IDS muss sicherstellen, dass sämtliche von ihm gesammelten Daten nur aus vertrauenswürdigen Quellen stammen (**Datengenerierung, Datensammlung**)
2. Die Analyse darf die zu schützenden Systeme nicht zu stark belasten (**Datenanalyse**)
3. Das IDS muss den SSO bei der Überprüfung der, von ihm gesammelten und ausgewerteten Daten so weit unterstützen, dass verschiedene Sichtweisen auf die Audit-Daten und ihre Analyse möglich sind (**Präsentation der Daten**)
4. Das IDS muss Funktionalität gewährleisten.
5. Das IDS muss in der Lage sein, die gesammelten Daten abzuspeichern, und ihre Echtheit und Unverfälschtheit für spätere juristische Zwecke sicherzustellen.
6. Das IDS muss, nach hinreichend genauer Analyse der Audit-Daten, im Angriffsfall einen Alarm auslösen und das IRS mit allen Daten versorgen, die das IRS zur Einleitung entsprechende Gegenmaßnahmen benötigt (**security audit automated response**).

Die oben aufgeführten Anforderungen führen zur Festlegung weiterer, spezifischerer Anforderungen, die im einzelnen folgende Punkte betreffen, und auf die hier nicht im Detail eingegangen wird :

- **Generierung und Sammlung der der Audit-Daten :** Gefordert wird, dass das ID-System möglichst viele Quellen zu Datensammlung berücksichtigt und in der Lage ist sowohl auf Betriebssystemebene, als auch auf Netzebene Daten zu überwachen. Ferner soll das IDS beide Erkennungsmethoden implementieren (Anomalie- und Missbrauchserkennung).
- **Analyse der Auditdaten :** Die Echtzeit-Analyse wird als eine Hauptanforderung betrachtet. Verlangt wird von dem ID-System auch die Angriffsvorhersage. Es wird Vorausgesetzt, dass die Anomalie- und Missbrauchserkennungs-Komponenten fehlerfrei arbeiten. Die Erstellung von Benutzerprofilen soll zumindest halbautomatisch erfolgen und die Konsole des ID-Systems muss eine graphische Benutzeroberfläche unterstützen.

- **Präsentation der Auditdaten :** Das IDS muss Berichte erstellen können, die Zusatzinformationen zu den gesammelten Audit-Daten enthalten. Diese Forderung schliesst auch die Unterstützung von genügenden Aktivitätsberichten mit ein. Die Daten müssen so aufbereit werden, dass unter Umständen auch eine manuelle Weiterverarbeitung möglich ist. Es muss ferner möglich sein, den Datensatz, der den Alarm ausgelöst hat darzustellen.

- **Speicherung von Audit-Daten :** Das IDS muss die Authentizität und die Integrität der Audit-Daten sicherstellen.

- **Ereignisauswahl für Auditdaten :** Es wird verlangt, dass das ID-System eine dynamische Einstellung der Audit-Daten-Granularität ermöglicht und dem SSO die Änderungen der Systemeinstellungen erlaubt.

Zu bemerken ist, dass das Sicherheitsaudit hohe Lasten für Speicher und CPU bedeutet. Daher wird das Sicherheitsaudit oft nicht durchgeführt.

4.3.2 ID-F2 Wartungsunterstützung

Wichtige Forderung in diesem Zusammenhang ist die **Erweiterbarkeit** und die **Skalierbarkeit** des IDS im Hinblick auf zukünftige Angriffe und Angriffstaktiken. Diese Forderung bringt die Erweiterbarkeit der Signaturdatenbank mit sich. Bei den Benutzerprofilen müssen zusätzliche Merkmale (neben CPU-Zeit, Seitenwechselrate etc.) hinzugefügt werden können. Weiterhin muss die IDS-Analysekomponente (IDS-Kernel) skalierbar sein. Die Skalierbarkeit ist für die Anpassung an schnellere Netze, Vergrösserte Knotenmenge und neue Protokolle notwendig. Wenn der IDS-Kernel diesen Veränderungen nicht angepasst werden kann, könnte das ID-System zu einem Engpass für das gesamte System werden.

4.3.3 ID-F3 Identifizierung und Authentifizierung

Der Zugriff auf Audit-Daten, Systemkonfigurationsdateien und andere sicherheitsrelevante Daten des IDS, sollte nur dem SSO erlaubt sein. Das IDS sollte daher Mechanismen oder Maßnahmen vorsehen, um dies zu gewährleisten. Der Zugang zu sicherheitsrelevanten Funktionen der IDS-Analyse-Komponente sollte nur über die IDS-Konsole erlaubt sein. Entfernte Zugriffe (Remote Login) und Daten-/Datei-Transfer(FTP bzw. TFTP) zu anderen Rechnern sollte über diese Konsele nicht erlaubt sein . Die Verbindung zwischen Konsole und IDS sollte über einen vertrauenswürdigen Pfad stattfinden (z.B. über eine verschlüsselte Verbindung oder ein eigenständiges Netzwerk).

4.3.4 ID-F4 Schutz der Sicherheitsfunktionen

Bestimmte Sicherheitsfunktionen eines jeden IDS müssen geschützt werden, damit die Angreifer das ID-System nicht überlisten oder modifizieren können. Einige Möglichkeiten einen ungeschützten IDS in einen unbrauchbaren Zustand zu versetzen sind nach [vHK98] folgende :

- Bei Verteiltensystemen : Angriff findet auf eine bestimmten Komponente statt (z.B. Überwachungs-Komponente). Auf diese Weise können Teile des ID-Systems abgeschnitten werden.
- Ausnutzen einer Systemschwachstelle, mit dem Ziel das IDS funktionsunfähig zu machen (z.B. durch einen Pufferüberlauf).
- Angriff auf die Lern-Komponente der Anomalieerkennung.
  Ziel : Außerkraftsetzung der IDS-Analyse.
  Angriffsmethode : Dem Angreifer gelingt es das System so zu überlisten, dass die Lern-Komponente das Angriffsverhalten als normal lernt.
  Folge : Bei einem Angriff, der nach diesem Verhaltensmuster durchgeführt wird, gibt es keine Angriffserkennung.

Um solche Angriffsmöglichkeiten zu vermeiden müssen gewisse Anforderungen erfüllt werden :

- Schutz interne ID-Mechanismen vor Missbrauch.
- Vertrauenswürdige Wiederherstellung (Trusted Recovery) der IDS-Funktionen nach einem unerwartetem Ausfall.
- Alarmauslösung bei Komponentenausfällen und die Vermeidung von Datenverlussten beim Wiederherstellungsvorgang einer unterbrochenen Verbindung zwischen den IDS-Komponenten (Reference Mediation)

4.3.5 ID-F5 Vertraulichkeit (Privacy)

Hierunter fallen Datenschutzrechtliche Anforderungen. Die Art und Weise, mit der die Audit-Daten aufgezeichnet werden, dürfen dem (deutschen) Datenrechtschutz nicht widersprechen. Die Aufzeichnung personenbezogener Daten sollte, soweit es geht, vermieden werden. Eine Lösung bietet hierbei die Anonymisierung der Nutz-Daten.

4.3.6 ID-F6 Verfügbarkeit der Dienste

Die Verfügbarkeit des ID-Systems ist von grosser Bedeutung. Diese Kategorie kann nicht mit speziellen Kriterien beschrieben werden, weil die Verfügbarkeit der Dienste stark von der Durchsetzung, bzw. Erfüllung der anderen Anforderungen abhängt, insbesondere von *ID-F2* und *ID-F4*. In *ID-F4* wird z.B. gefordet, dass bei Komponentenausfällen das ID-System genau vordefinierte Aktivitäten ausführen muss, um die Sicherheitsvorgaben umzusetzen. Eine andere Forderung war, dass bei Komponentenausfällen sofort ein Alarm ausgelöst werden muss, um das System vor Denial-of-Service Angriffen zu schützen. Die Umsetzung diese Anforderungen trägt zur Verfügbarkeit des Systems bei. Aus diesem Grund konnten keine differenzierten Anforderungen für die Verfügbarkeit des ID-Systems aufgestellt werden. Es muss bemerkt werden, dass diese Abhängigkeiten auch zwischen anderen Kategorien existieren. Der Abhängigkeitsgrad dieser Kategorien lässt aber, im Gegensatz zu Verfügbarkeit, ihre getrennte Betrachtung zu.

Für eine ausführliche Beschreibung der Funktionalitätskategorien soll an dieser Stelle auf die Quelle [vHK98] verwiesen werden.

4.3.7 Vertrauenswürdigkeit (Assurance)

Oben wurden rein funktionale Aspekte eines IDS beschrieben. Zur Stärkung des Vertrauens in die Implementierung müssen weitere Aspekte beachtet werden, wie z.B. die Vertrauenswürdigkeit. Die ausführliche Dokumentation und eine effiziente Benutzerführung wurde in [vHK98] als eine Hauptanforderung für das Vertrauen in ein Sicherheitsprodukt angegeben. Zur einer guten Dokumentation gehört dabei ein Handbuch in ausgedruckter Form, sowie eine Online-Hilfe, die den SSO bei der Interpretation der Analyseergebnisse unterstützen, und als Entscheidungshilfe bei der Einleitung von Sofortmaßnahmen dienen soll. Insbesondere sollte die IDS-Dokumentation die Installation und den Betrieb des IDS verständlich und umfassend darstellen. Der SSO sollte anhand der Dokumentation verstehen können

- welcher Angriff gegen das System durchgeführt wird,
- welche Ausmaße dieser Angriff hat und welche manuellen Gegenmaßnahmen getroffen werden müssen,
- welche Recovery-Maßnahmen eingeleitet werden müssen.

4.3.8 Bemerkungen

Die *Funktionalitätskritärien* und die unter *Vertrauenswürdigkeit* dargestellten Aspekte, erfassen nicht die organisatorischen Maßnahmen, die von den ITSEC-Kriterien erfordert werden. Unter organisatorischer Maßnahmen fällt auch die

Einhaltung datenschutzrechtlicher Bestimmungen, wie z.B. die Aufklärungspflicht über die Aufzeichnung personenbezogener Daten oder die Löschung personenbezogener Daten spätestens nach Beendigung der Nutzung.

Kapitel 5

ENTWICKLUNGSSPRACHEN FÜR IDS REGELN

5.1 Das Beobachten und Erkennen von Missbrauch durch Expertensysteme

In den Folgenden Abschnitten wird in das Thema der Expertensysteme eingeführt. Von da aus wird erklärt, was Regelsprachen sind und welche Fähigkeiten sie haben müssen. Danach werden zwei Regelspachen, P-BEST und RUSSEL, die zu den Intrusiondetection-Systemen EMERALD und ASAX gehören vorgestellt. Daran schließt sich ein Vergleich der beiden Sprachen anhand von Gegenüberstellungen von Regeln zur Angriffsentdeckung.

5.1.1 Einführung in Expertensysteme

Der Mensch ist oft überfordert, wenn er eine (relativ) große Menge von Daten mit einer (relativ) großen Menge von Regeln sinnvoll verknüpfen soll, um dann die richtigen Schlüsse für seine weiteren Handlungen zu ziehen. Dies gilt erst recht bei der Analyse von Auditdaten.

Einen Ausweg bieten Expertensysteme, denn sie bieten automatische Verfahren, um Fakten ihres speziellen Fachgebietes (die Domäne des Expertensystems) zu verarbeiten. Der Vorgang des Schlüsseziehens, **die Inferenz**, geschieht auf der Grundlage von Inferenzregeln.

Das Wissen von Expertensystemen, d.h., die Menge der Inferenzregeln, die sie beherrschen, wird **Regelbasis (*rule base*)** genannt.

Der Regelbasis steht eine Menge an Fakten gegenüber, die das Expertensystem verarbeiten soll. Diese Faktenmenge heißt **Faktenbasis (*factbase*)**.

Expertensysteme im Bereich Intrusion Detection haben die Aufgabe, aus den Fakten, die sie von den Auditeinheiten präsentiert bekommen (der Faktenbasis), zu schließen, ob sich gerade ein sicherheitsrelevanter Vorfall ereignet oder nicht. Oder ob sich ein Vorfall in der Vergangenheit ereignet hat. Die Abbildung 5.1

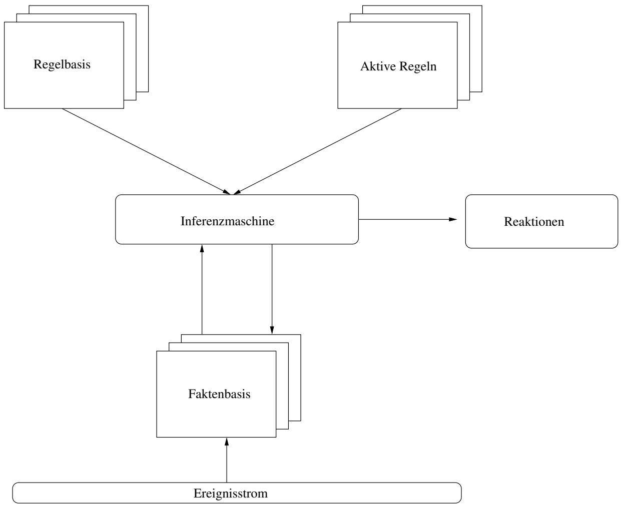

Abbildung 5.1: Schematische Übersicht der einzelnen Teile eines Expertensystems

zeigt die Komponenten eines Expertensystems, dass unter Intrusion-Detection eingesetzt wird. Die Inferenzmaschine versucht dabei jedesmal, wenn ein Fakt in die Faktenbasis eingefügt wird, Regeln aus der aktiven Regelbasis auf die Fakten der Faktenbasis anzuwenden. Dabei werden neue, abgeleitete Fakten geschaffen, die nun ihrerseits in die Faktenbasis eingebracht werden. Dies löst sofort wieder die Suche nach passenden Regeln und deren Anwendung aus.

Die **Lösungsstrategien**, die Expertensysteme anwenden, sind von der jeweiligen **Domäne**, dem jeweiligen Wissensgebiet des Systems, **unabhängig**. Anders sieht es bei den Bedingungen aus, unter denen ein Expertensystem zum Einsatz kommen soll. Diese Bedingungen können die am besten einzusetzende Lösungsstrategie bestimmen. Ein Beispiel für solche Bedingungen ist die Frage, ob das System die Einbruchsentdeckung in Echtzeit oder im "offline"-Betrieb erledigen soll.

5.1.2 Mathematische Grundlagen der Inferenz

5.1.2.1 Eine kleine Einführung in die Logik

Die Tabelle 5.1 erklärt die Bedeutung der in der Aussagenlogik üblicherweise verwendeten Verknüpfungssymbole. Wird das Zeichen $\rightarrow$ im Zusammenhang mit der Definition von Abbildungen verwendet, so gibt es die Richtung der Abbildung

Zeichen	Bedeutung
$\rightarrow$	Die Implikation (Wenn ... dann)
$\leftrightarrow$	Die Äquivalenz
$\wedge$	Das logische Und
$\vee$	Das logische Oder
$\neg$	Die Negation

Tabelle 5.1: Die verwendeten booleschen Operatoren

an, von der Ursprungsmenge zur Zielmenge[1]. Sind zwei Formeln A, B semantisch äquivalent, so schreibt man auch $A \equiv B$ um von Aussagen der Form $A \leftrightarrow B$ zu unterscheiden. Diese Unterscheidung wird deutlich, wenn man eine gegebene Aussage durch Anwendung von Rechenoperationen in eine inhaltlich gleiche aber syntaktisch andere umwandelt. Beispiel:

$$\boxed{A \wedge (A \vee B) \equiv (A \wedge A) \vee (A \wedge B) \equiv A}$$

Die Zahlen 0 und 1 werden in diesem Abschnitt jeweils als Synonyme für *falsch* und *wahr* verwendet. Dies ist eine bijektive, d.h. umkehrbare Abbildung:

$$\boxed{W : \{0, 1\} \rightarrow \{wahr, falsch\}}$$

Wobei für ein beliebiges Element $x \in \{0, 1\}$ gilt:

$$\boxed{W(x) = \begin{cases} falsch & : \quad x = 0 \\ wahr & : \quad x = 1 \end{cases}}$$

5.1.2.2 Modus-Ponens

Die Lösungsstrategie, die in Expertensystemen bei der Online-Analyse angewandt wird, heißt **Modus-Ponens**. Ihre Funktionsweise wird im Folgenden hergeleitet.

Es seien eine Tautologie, also eine immer wahre Aussage, der Form $p \rightarrow q$ und die Teilaussage p gegeben, dann ist Wahrheitswert der Teilaussage q gleich dem der Aussage p. Das folgende Beispiel soll diesen Zusammenhang verdeutlichen:

Beispiel 2 Die Tautologie $p \rightarrow q$ sei hier durch die Aussage "Wenn Schnee liegt, ist es kalt!" gegeben. Diese *Regel* besteht aus den zwei unteilbaren Aussagen "Es liegt

[1] Auch dies lässt sich auf die logische Verknüpfung der Implikation zurückführen, denn eine Abbildung $f : O \rightarrow F$ besagt ja, ist ein Element $x \in O$ gegeben, so gibt es dazu ein Element $f(x) = y \in F$.

Schnee." die hier mit p bezeichnet wird, und die Behauptung "Es ist kalt.", die hier mit q bezeichnet wird. Wenn man feststellt, dass Schnee liegt, schließt man, dass es kalt ist. D.h., aus der Regel $p \to q$ und der Tatsache $p = 1$ folgert man, $q = 1$.

$$((p \to q) \wedge p) \quad \equiv \tag{5.1}$$
$$((\neg p \vee q) \wedge p) \quad \equiv \tag{5.2}$$
$$(p \wedge q) \tag{5.3}$$

Wie man aus den Formeln und Umformungen 5.1 bis 5.3 sieht, gilt:

$$(p \to q) \wedge p \equiv (p \wedge q)$$

Aus $(p \to q) \wedge p$ wird, wenn die Regel $(p \to q)$ gültig ist:

$$(1 \wedge p) \equiv (p \wedge q) \equiv p$$

Damit muss, wenn $p = 1$ gilt, $q = 1$ gelten.

Es können neben der Regel $(p \to q)$ auch noch weitere Regeln der Form $(o_i \to q)$ existieren. D.h es kann auch noch weitere Fakten geben, die ebenfalls das Vorhandensein von q beweisen. Um bei dem Bild des obigen Beispiels zu bleiben, es muss kein Schnee liegen wenn es kalt ist.

Die zu einer Regel R passende Aussage p hat dabei die Form

$$p = \bigwedge_{i \in A_R} \phi_i \tag{5.4}$$

wobei A_R die Indexmenge derjenigen Literale ist, die aus den Fakten die für die Regel R von Interesse sind, konstruiert wurden. D. h., für jedes Literal ϕ_i gilt $\phi_i \in \{f_1, \ldots, f_k, \neg f_1, \ldots, \neg f_k\}$. Wobei die Menge der Fakten durch $\{f_1, \ldots, f_k\}$ gegeben ist.

Bekommt das Expertensystem ein Faktum in Form einer wahren Aussage p, so sucht es in seiner Wissensdatenbank nach einer Aussage der Form $p \to q$. q stellt dann die Schlussfolgerung oder Konsequenz von p dar, während p eine hinreichende Voraussetzung von q darstellt. Die Konsequenz q kann als neues Faktum in die Faktenbasis eingefügt werden und somit als Voraussetzung für weiter Schlüsse dienen. Auf diese Weise können Ketten von Schlussfolgerungen nach dem Muster in Formel 5.5 abgearbeitet werden. Dabei hat jede Regel x eine eigene Indexmenge I_x, deren Elemente die für diese Regel nötigen Fakten indizieren, mit Ausnahme der Fakten q_x, die durch Anwendung der Regel $x - 1$ abgeleitet worden sind.

$$\left(\bigwedge_{j \in I_i} \phi_j\right) \wedge q_i \to q_{i+1}, \left(\bigwedge_{j \in I_{i+1}} \phi_j\right) \wedge q_{i+1} \to q_{i+2}, \ldots, \left(\bigwedge_{j \in I_{n-1}} \phi_j\right) \wedge q_{n-1} \to q_n \tag{5.5}$$

Dies lässt sich mit dem Bild einer Kette von Dominosteinen vergleichen, bei der der erste angestoßen wird und sein Sturz zur Konsequenz hat, dass viele andere mit umfallen.

5.1.2.3 Hornformeln und Hornklauseln

Eine Hornformel ist eine boolsche Formel in konjunktiver Normalform, bei der jedes Disjunktionsglied höchstens ein positives Literal enthält. Die einzelnen Disjunktionsglieder werden dann Hornklauseln genannt.

Eine Menge von Hornklauseln ist eine Menge von Implikationen der Form:

$$(a \to b) \equiv (\neg a \vee b) \tag{5.6}$$

Es gilt meist noch, dass a in der Form von Formel 5.4 vorliegt. Dann wird aus Gleichung 5.6

$$(a \to b) = \left(\left(\bigwedge_{i \in A_R} \phi_i \right) \to b \right) \equiv \left(\left(\bigvee_{i \in A_R} \neg\phi_i \right) \vee b \right) \tag{5.7}$$

Die ϕ_i sind wieder Literale, die aus einer Faktenmenge $\{f_1, \ldots, f_k\}$ gebildet wurden. Da für ein Literal $\phi_i = \neg f_j$ gelten kann, liegen die Regeln der Regelbasis nicht notwendigerweise alle als Hornklauseln vor. Es ist jedoch möglich, die Regelbasis durch einführung zusätzlicher komplementärer Fakten so umzuformen, das alle Regeln der Form der Hornklauseln entsprechen. Dazu wird jedesmal, wenn in einer Regel die Teilformel $\neg f_i$ vorkommt, ein Fakt $fcompl_i = \neg f_i$ eingeführt und die Regel, durch eine neue Regel ersetzt, in der jedes vorkommen von $\neg f_i$ durch $fcompl_i$ ersetzt wird. Die Regeln der Regelbasis sind nun Hornklausesln, wenn der abgeleitete Fakt eine atomare Aussage ist. Dies lässt sich dadurch erreichen, dass aus einer Regel, die aus gegebenen Fakten eine aus n Teilen zusammengesetzte Aussage $b = \bigwedge_{i=1}^{n} \beta_i$ ableitet, in n Regeln aufgeteilt werden kann.

$$\left(a \to \bigwedge_{i=1}^{n} \beta_i \right) \equiv \bigwedge_{i=1}^{n} (a \to \beta_i) \tag{5.8}$$

Die Regeln der Regelbasis liegen dann in der Form 5.7 vor.

5.1.3 Evaluation

Die Regelbasis eines Expertensystemes, das zur Missbrauchsentdeckung eingesetzt wird, kann hinsichtlich verschiedener Punkte bewertet werden.

Erstens, kann die Gültigkeit oder Konsistenz der Regeln betrachtet werden. Die Konsistenz ist aus der Sicht der mathematischen Logik eine notwendige für die

Voraussetzung Richtigkeit der Schlussfolgerungen des Expertensystems. Denn unter der Voraussetzung der Gültigkeit aller Regeln der Regelbasis sind alle Schlussfolgerungen des Expertensystems richtig.

Zweitens kann die Regelbasis in Bezug auf ihre Fähigkeit, Missbrauch zu entdecken, bewertet werden. Hierbei gibt es zwei Größen, die zur Bewertung herangezogen werden können, die Falschpositivrate und die Falschnegativrate. Die Falschpositivrate bedeutet ihrerseits, dass das Expertensystem falsche Schlüsse zieht, was wiederum bedeutet, das die Regelbasis inkonsistent ist.

Eine andere wichtige Größe ist die Leistung. Sie hat keinen Einfluss auf die Richtigkeit der Regelbasis, bestimmt aber die Einsetzbarkeit des Systems.

5.1.3.1 Die Leistung

Bei der Bewertung von Intrusion Detection Systemen spielt die Leistung eine ganz wichtige Rolle. Die Leistung ist die Zahl der abgearbeiteten Ereignisse aus dem Strom der Auditdaten pro Zeiteinheit. Die Leistung eines gegebenen Systems hängt von verschiedenen Faktoren ab. Einer der Faktoren ist die Zahl der Fakten in der Faktenbasis. Zwei andere Faktoren sind die Mächtigkeit der Regelbasis und die Komplexität der Regeln. Die Komplexität der Regeln ist zum einen von der Zahl der Fakten, die bei der Regelanwendung berücksichtigt werden müssen, zum anderen von der Zahl der Regeln, die durch die aktuelle Regel beeinflusst werden, abhängig.

Die Leistung ist Ausdruck der Effektivität, mit der der Inferenzalgorithmus arbeitet, der Komplexität von Fakten- und Regelbasis und der Leistung des Rechensystems, auf dem das ID-System läuft.

Für die Echtzeit-Überwachung ist die Leistung das wichtigste Kriterium, das über den Sinn des Einsatzes des Überwachungssystems entscheidet.

5.1.3.2 Die Konsistenz der Regelbasis

Die Regeln stellen für das Expertensystem Axiome dar. D. h., sie sind für das Expertensystem unwiderlegbare Gesetze, nach denen es zu handeln hat. Damit ist die Aussagefähigkeit der Schlussfolgerungen des Systems von der Widerspruchsfreiheit der Regeln seiner Regelbasis abhängig. Enthält die Regelbasis widersprüchliche Regeln, so können Widersprüche, die der Form $a \to \neg a$ entsprechen bei der sukzessiven Anwendung der Inferenzmethode, z. B. Modus Ponens auftreten. Zur Illustration des Problems nun folgendes Beispiel:

Beispiel 3 Die Regelbasis bestehe aus folgender Regelmenge:

$$R = \{R_1 = (a \rightarrow b), R_2 = (c \rightarrow d), R_3 = ((d \wedge e) \rightarrow \neg a)\} \tag{5.9}$$

Wenn die Faktenbasis mit $(a = 1, b = 0, c = 1, d = 0, e = 1)$ belegt ist, führt das durch die Anwendung von Modus-Ponens zu zwei möglichen Ergebnissen, abhängig von der Reihenfolge der Auswertung der Regeln.

Reihenfolge 1: R_1, R_2, R_3. Dies führt nacheinander zu folgenden Belegungen der Faktenbasis:

$$(a = 1, b = 1, c = 1, d = 0, e = 1)$$
$$(a = 1, b = 1, c = 1, d = 1, e = 1)$$
$$(a = 0, b = 1, c = 1, d = 1, e = 1)$$

Reihenfolge 2: R_2, R_3. Die aus dieser Reihenfolge der Regelanwendungen resultierende Folge von Belegungen ist:

$$(a = 1, b = 0, c = 1, d = 1, e = 1)$$
$$(a = 0, b = 0, c = 1, d = 1, e = 1)$$

Regel R_1 braucht nicht mehr angewendet werden, da sie in diesem Fall die Belegung nicht mehr ändern würde.

Diese Abhängigkeit des Ergebnisses von der Reihenfolge der Regelanwendungen bedeutet die Inkonsistenz der Regelbasis.

Definition 4 (Konsistenz der Regelbasis) *Eine Regelbasis heißt* **konsistent**, *wenn es keine Belegung der Faktenbasis gibt, bei der die Folgebelegung von der Reihenfolge der Regelanwendung der Regeln abhängt.*

Aus der Konsistenz der Regelbasis folgt nicht deren Widerspruchsfreiheit. Aus der Widerspruchsfreiheit folgt jedoch die Konsistenz.

Das Erfüllbarkeitsproblem Die Überprüfung der Konsistenz der Regelbasis lässt sich auf das Erfüllbarkeitsproblem reduzieren. Als Beispiel wird wieder die Regelmenge von Ziffer 5.9 in Beispiel 3 betrachtet. Die Regeln R_1, R_2, R_3 sind aus der Sicht des Expertensystems Tautologien. Daher gilt für die Inferenzmaschine des Systems

$$R_1 \wedge R_2 \wedge R_3 \equiv 1 \tag{5.10}$$

für alle Belegungen der Faktenbasis. Um zu überprüfen, ob das wirklich so ist, muss die Erfüllbarkeit der Negation der Formel 5.10 geprüft werden.

Beispiel 4 (Fortsetzung von Beispiel 3) Die Negation von Formel 5.10 ergibt:

$$\begin{aligned}\neg R_1 \vee \neg R_2 \vee \neg R_3 =\\ \neg(\neg a \vee b) \vee \neg(\neg c \vee d) \vee \neg(\neg d \vee \neg e \vee \neg a) \equiv\\ (a \wedge \neg b) \vee (c \wedge \neg d) \vee (d \wedge e \wedge a) \not\equiv 0\end{aligned} \tag{5.11}$$

Die resultierende Formel 5.11 ist erfüllbar, somit enthält die Regelbasis widersprüchliche Regeln.

Das Erfüllbarkeitsproblem ist allgemein nur mit exponentiell wachsendem Aufwand in Ahängigkeit von der Zahl an atomaren Aussagen, die in der Formel enthalten sind, zu lösen. Das liegt daran, dass im schlimmsten Fall alle möglichen Belegungen der atomaren Aussagen durchgegangen werden müssen, bis klar ist, ob die Formel erfüllbar ist oder nicht. Dies sind $2^{\text{Zahl der atomaren Aussagen}}$ Möglichkeiten. Das Erfüllbarkeitsproblem wird als NP-vollständig bezeichnet, denn es kann nur unter Benutzung eines nicht deterministischen Algorithmus' mit weniger als exponentiellem Aufwand gelöst werden, und es lassen sich alle anderen Probleme, die mit polynomiellem Aufwand nicht deterministisch gelöst werden können, auf dieses Problem in polynomieller Zeit reduzieren.

Es gibt jedoch Formelmengen, deren Erfüllbarkeit in polynomieller Zeit geklärt werden kann. Dies sind z. B. die in Abschnitt 5.1.2.3 behandelten Hornformeln. Dieser Algorithmus lässt sich nicht zur Überprüfung der Konsistenz der Regelbasis eines Expertensystemes anwenden, denn die Regeln sind schon Hornklauseln. Angenommen, die Regelbasis bestehe aus den Regeln $R_1, \ldots, R_n$. Wenn alle Regeln gültig sind, dann gilt

$$\bigwedge_{i=1}^{n} R_i \equiv 1 \tag{5.12}$$

Zur Überprüfung der Gültigkeit wird die Formel negiert. Die resultierende Formel wird dann auf Erfüllbarkeit getestet. Die Negation dieser Formel ergibt

$$\bigvee_{i=1}^{n} \neg R_i \equiv 0 \tag{5.13}$$

Diese Formel erfüllt nicht mehr die Voraussetzungen, eine Hornformel zu sein. Die Fakten eines Expertensystems, das zur Missbrauchsentdeckung eingesetzt wird, können in zwei Klassen eingeteilt werden:

1. Die Menge der Fakten, die direkt aus den Auditdaten stammen.

2. Die Menge der Fakten, die durch die Inferenz entstehen.

Die Fakten der Klasse 1 stellen die Menge der möglichen Ausgangsfakten dar. Bei dieser Faktenmenge kann es sein, dass bestimmte Kombinationen von Fakten in der Praxis nie vorkommen können. Damit gibt es Faktenbelegungen, die zwar theoretisch möglich sind, aber in der Praxis nicht vorkommen. Dies entspricht Regeln der Form $p \to \neg q$, die nicht in der Regelbasis des Expertensystems vorkommen. Zur Illustration wird wieder auf das Beispiel 3 zurückgegriffen:

Beispiel 5 (Fortsetzung von Beispiel 3) Die Menge der Fakten, die direkt aus den Auditdaten stammen, sei hier:

$$\{a, c, e\}$$

Ferner seien a und e so miteinander korreliert, dass implizit immer $a \to \neg e$ gilt, also werden Belegungen mit $a = e = 1$ nie auftreten. Damit würde die Problematik der Inkonsistenz in diesem Falle nicht auftreten.

Es wird für diejenigen, die mit der Erstellung der Regelbasis beauftragt sind, nicht immer möglich sein, alle diese implizit innerhalb der Fakten vorhandenen Regeln zu kennen. Damit werden Inkonsistenzen in den Regelbasen nie ausgeschlossen werden können.

5.1.3.3 Die Falschpositiv- und die Falschnegativrate

Der Sinn der Regelbasis bei der Missbrauchsentdeckung ist die Befähigung des Expertensystems, Verletzungen der Sicherheitskriterien durch die im System ablaufenden Handlungen zu erkennen. Die Handlungen werden dabei nicht direkt beobachtet. Es werden die Spuren, die die im System ablaufenden Aktionen in den Auditdaten hinterlassen, stellvertretend für die Aktionen beobachtet und analysiert. Die Auditdaten werden von speziellen Programmen oder Einrichtungen, die direkt im Kern des Betriebssystems verankert sind, als Folge der Aktionen erzeugt.

Die Falschnegativ- und die Falschpositivraten sind von der Qualität der Regeln und von der Qualität der Auditdaten abhängig.

Diese Raten werden sich näherungsweise durch Experimente bestimmen lassen.

5.1.4 Backwardchaining

Alternativ zu der oben beschriebenen Methode kann auch andersherum vorgegangen werden. Es wird zuerst die Hypothese q aufgestellt, z.B. $q =$„Es ist kalt.“. Nun wird nach Fakten wie z.B. $p =$„Es liegt Schnee.“gesucht, die die Hypothese q stützen. Es also wird nach Regeln $p \to q$ gesucht, und dann wird die Teilaussage p auf ihren Wahrheitsgehalt hin geprüft. Steht die Aussage q am Ende einer Kette von Implikationen, so wird diese Kette von hinten bis nach vorne zu den Anfangsfakten durchgegangen. Dabei erstellt das System eine Liste aller Fakten, die in der Faktenbasis vorhanden sein müssen, damit die Aussage q wahr ist. Dieses Vorgehen wird mit *backward-chaining* bezeichnet.

Backwardchaining kann man sich auch als Suche in einem Baum vorstellen. Die Suche beginnt in der Wurzel des Baumes. Von hier aus werden alle Möglichkeiten verfolgt, wobei durch die vorgefundenen Fakten die Zahl der Möglichkeiten immer weiter eingeschränkt wird, bis im Idealfall nur noch eine Möglichkeit vorhanden ist. Die Wurzel des Baumes stellt dabei die Aussage mit dem höchsten Abstraktionsniveau dar.

Backwardchaining entspricht dem **Top Down**-Prinzip.

5.1.5 Regeln als Petrinetze

Beispiel 6 (Fortsetzung Beispiel 3) Die Regelanwendungen bestehen in diesem Beispiel darin, nacheinander folgende Zuweisungen durchzuführen:

$$b := a, d := c, a := \neg d \vee \neg e$$

Die Regelanwendung besteht aus lediglich zwei Arten von Operationen auf Variablen: der logischen Verknüpfung $\wedge$ und der Zuweisung. Im Folgenden wird gezeigt, wie sich diese Operationen grafisch veranschaulichen lassen. Dabei werden Petrinetze zur Darstellung der Regelanwendung und ihrer Wirkung auf das Expertensystem benutzt. Petrinetze sind gut geeignet um zeitliche oder kausale Abhängigkeiten und nebenläufige Vorgänge zu verdeutlichen.

5.1.5.1 Formale Definition von Netzen

Nun werden erst einmal einige wichtige Definitionen folgen, die sich an die entsprechenden Definitionen aus [Val] anlehnen. Diese Definitionen werden im Anschluss daran angewendet (Abschnitt 5.1.5.2).

Definition 5 (Netz) *Ein Petrinetz, kurz Netz, besteht aus zwei disjunkten endlichen Mengen, Stellen (S) und Transitionen (T) genannt. Die Menge der Transitionen darf nicht leer sein. Hinzu kommt eine Flussrelation:*

$$F \subset (S \times T) \cup (T \times S)$$

Ein Netz wird als Tripel $N = (S, T, F)$ notiert.

Definition 6 (Markierung) *Es sei $N = (S, T, F)$ ein Netz mit endlicher Stellenmenge $S = \{s_1, \ldots, s_n\}$. Eine* **Markierung** *(marking) m für N ist eine Abbildung*

$$m : S \to \mathbb{N}_0$$

wobei $m(s_i)$ die Zahl der **Marken***(tokens) in der Stelle s_i angibt. Mit M_S oder M wird die Menge aller möglichen Markierungen über der Stellenmenge S bezeichnet.*

Diese Definition wird nun erweitert, so dass sich Zustandsänderungen modellieren lassen.

Definition 7 (Stellen/Transitionsnetz (S/T-Netz)) *Ein Stellen/Transitionsnetz ist ein 6-Tupel $N = (S, T, F, K, W, m_0)$. Die ersten drei Komponenten des Tupels stellen ein gewöhnliches Netz dar. Die folgenden Komponenten sind:*

Kapazitätsfunktion: $K : S \to I\!N \cup \{\omega\}$ *Diese Funktion ordnet jeder Stelle $s \in S$ eine Kapazität zu. Die Kapazität gibt die maximale Zahl an Marken an, die sie aufnehmen kann. Das Element ω steht dabei für eine nicht endliche Kapazität.*

Kantenbewertungsfunktion: $W : (S \times T \cup T \times S) \to I\!N \cup \{\omega\}$ *Diese Funktion gibt die Zahl der Marken an, die beim Schalten der entsprechenden Transition bewegt werden. Dabei gilt $W(x, y) = 0$ genau dann, wenn $(x, y) \notin F$ gilt.*

Anfangsmarkierung: $m_0 : S \to I\!N \cup \{\omega\}$ *mit $m_0(s) \leq K(s)$ Diese Funktion gibt die Zahl der Marken in jeder Stelle an, bevor eine Transition geschaltet hat.*

Diese allgemeine Definition wird nun eingeschränkt, denn im Folgenden werden nur Netze betrachtet, deren Stellen eine Kapazität von 1 haben. Jetzt muss noch das Schalten einer Transition definiert werden. Erst danach ist es möglich, die Arbeitsweise der Inferenz unter der Modus-Ponens-Strategie als einen Prozess zu modellieren.

Definition 8 (aktiviert und schalten) *Es seien $N = (S, T, F, K, W, m_0)$ ein S/T-Netz, $t \in T$ eine Transition und $m_1, m_2 \in M_S$ Markierungen.*

aktiviert: *Eine Transition t heißt* **aktiviert** *in m_1, wenn $m_1(s) \geq W(s, t) \quad \forall s \in S$ und $(m_1(s) - W(s, t)) + W(t, s) \leq K(s) \quad \forall s \in S$ gilt.*

schalten: *Eine Transition t* **schaltet** *die Markierung m_1 zur Markierung m_2, wenn t in m_1 aktiviert ist und $m_2(s) = (m_1(s) - W(s, t)) + W(t, s) \quad \forall s \in S$ gilt.*

Wirkung: *Die Wirkung einer Transition t auf eine Stelle s sei mit $wk(t, s) = W(t, s) - W(s, t)$ bezeichnet.*

Als letzte vorbereitende Definition folgt nun die Definition von Bedingungs/Ereignis-Netzen. Auf diesen Netzen baut die Konstruktion von Netzen aus einer Regelbasis auf.

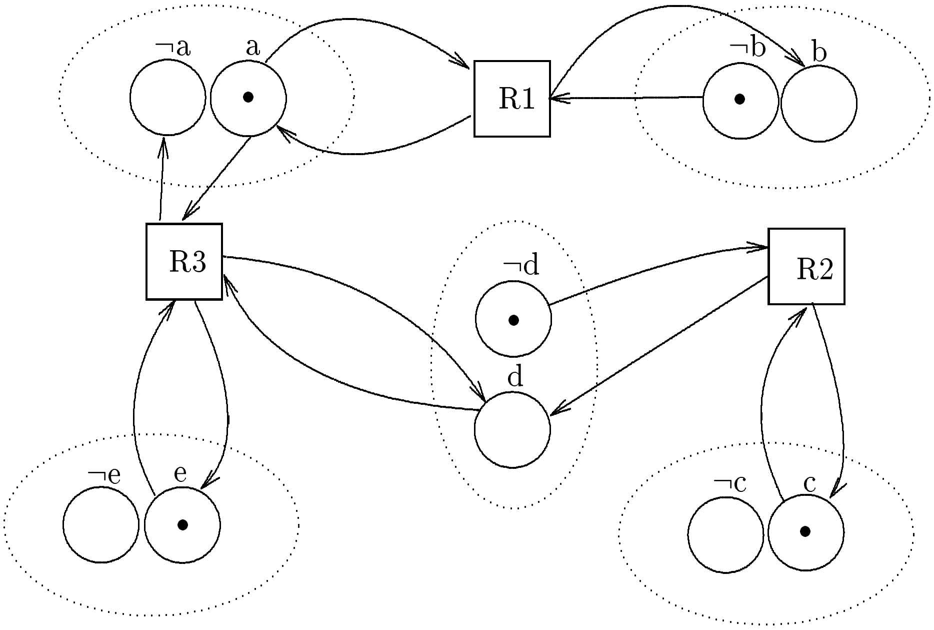

Abbildung 5.2: Die Regelbasis und Faktenbasis aus Beispiel 3 als B/E-Netz

Definition 9 (Bedingungs/Ereignis-Netz) *Ein* **Bedingungs/Ereignis-Netz***, auch B/E-Netz, ist ein S/T-Netz mit folgenden Einschränkungen:*

1. *Die Kapazität aller Stellen ist 1.* $K(s) = 1 \qquad \forall s \in S$

2. *Die Kantenbewertungsfunktion ist für alle Kanten 1.* $W(x, y) = 1 \qquad \forall (x, y) \in F$

Da K und W festgelegt sind, können sie bei der Notation von N weggelassen werden: $N = (S, T, F, m_0)$

Bei B/E-Netzen kann die Wirkung einer Transition t auf eine Stelle s $wk(t, s)$ kann einer der Werte aus $\{-1, 0, 1\}$ sein.

5.1.5.2 Die Konstruktion eines Regelnetzes

Es sei ein Expertensystem mit Regelbasis $R = \{p_1 \rightarrow q_1, \dots, p_n \rightarrow q_n\}$ und Faktenmenge $facts = \{f_1, \dots, f_k\}$ gegeben. Zu dieser Menge $facts$ gibt es die Menge der möglichen Literale

$$lfacts := facts \cup \{l_i | l_i = \neg f_i, 1 \leq i \leq k\}$$

Die Teilformeln $p_1, \dots, p_n$ der n Regeln stellen die Bedingungsteile der Regeln dar und sind dabei wie in dem Modell 3.3.1 auf Seite 151 aufgebaut. Die Teilformeln $q_1, \dots, q_n$ sind Literale aus der Menge $lfacts$.

Aus jeder Regel $R_x = p_x \rightarrow q_x$ wird eine Transition t_{R_x} erstellt, mit $1 \leq x \leq n$. Zu jeder Transition t_{R_x} werden zunächst die Stellen erstellt, die die Bedingungen für die Regel R_x repräsentieren. Damit wird für jedes Literal l das in p_x vorkommt eine Stelle s_l in die (am Anfang leere) Menge S eingefügt:

$$S := S \cup \{s_l | l \text{ kommt in } p_x \text{ vor.}\}$$

Sollte die Stelle s_l schon in der Menge S existieren, so würde sich durch die erneute Einfügung der Stelle in die Menge S die Menge nicht ändern. Die Operation bliebe in diesem Fall ohne Effekt. Dass die Stelle s_l schon in der Menge S ist, kann zwei Gründe haben:

1. Eine Transition, die schon in das Netz eingefügt worden war, benutzt in ihrem Bedingungsteil das gleiche Literal.

2. Eine Transition, die schon in das Netz eingefügt worden war, benutzt in ihrem Folgerungsteil das gleiche Literal.

Dann werden die Verbindungen der Stellen zur Transition t_{R_x} hergestellt. Für jedes Literal l, das in p_x vorkommt, werden zwei Kanten (t_{R_x}, s_l) und (s_l, t_{R_x}) in die Flussrelation F eingefügt.

$$F := F \cup \{(t_{R_x}, s_l), (s_l, t_{R_x})\}$$

Auf diese Weise werden alle zu den Literalen aus p_x gehörenden Stellen zu Nebenbedingungen, d. h., die Markierung der Stellen ändert sich durch das Schalten der Transition t_{R_x} nicht.

Der Folgerungsteil q_x der Regel R_x, der entweder $f_x \in lfacts$ oder $(\neg f_x) \in lfacts$ sein kann, besteht aus zwei Stellen s_{f_x} und $s_{\neg f_x}$.

$$S := S \cup \{s_{f_x}, s_{\neg f_x}\}$$

Ist $q_x = f_x$, so werden die zwei Kanten $(s_{\neg f_x}, t_{R_x})$ und (t_{R_x}, s_{f_x}) in F eingefügt. Ist die Folgerung $q_x = \neg f_x$, so werden die Kanten (s_{f_x}, t_{R_x}) und $(t_{R_x}, s_{\neg f_x})$ in F eingefügt.

$$F := F \cup \{(s_{f_x}, t_{R_x}), (t_{R_x}, s_{\neg f_x})\}$$

Wenn dies für alle Regeln gemacht wurde ist ein Netz wie das in Abbildung 5.2 entstanden. Die Startmarkierung wird so gewählt, dass für jeden gegeben Fakt $f_i \in facts$ die Stelle s_{f_i} markiert wird. Für jeden nicht gegeben Fakt f_i wird, wenn die Stelle $s_{\neg f_i}$ existiert, diese markiert. Auf diese Weise gilt zu Beginn $f_i = 1 \equiv m(s_{f_i}) = 1$.

Das Beispiel 7 illustriert die eben erläuterten Schritte zur Konstruktion eines B/E-Netzes aus einer Regelbasis.

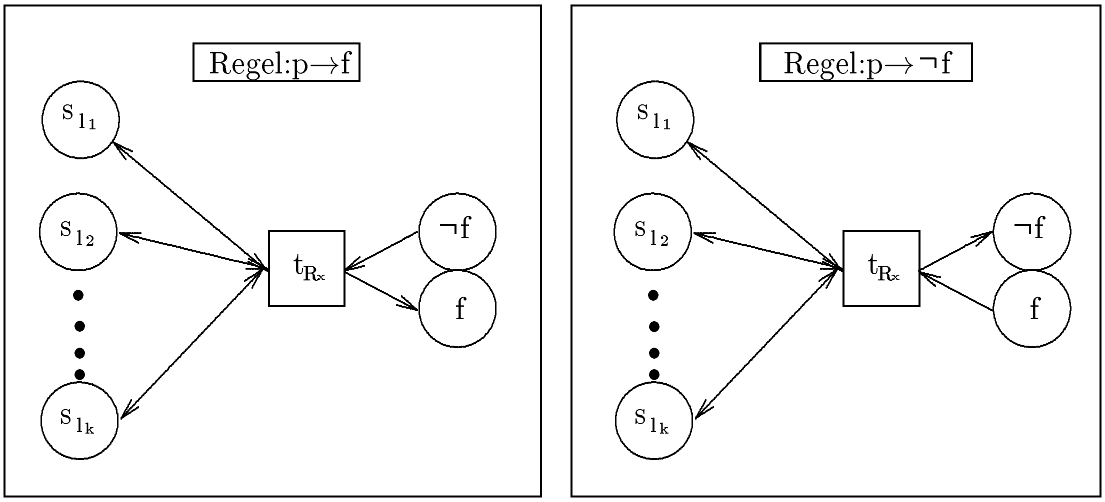

Abbildung 5.3: Eine Regel als Transition

Beispiel 7 (Fortsetzung Beispiel3) Das Expertensystem sei wieder gegeben durch $R = \{R_1 = (a \to b), R_2 = (c \to d), R_3 = (d \land e \to \neg a)\}$. Es sei $facts = \{a, b, c, d, e\}$ und $lfacts = facts \cup \{\neg a, \neg b, \neg c, \neg d, \neg e\}$. Die Transitionsmenge sei $T = \{t_{R_1}, t_{R_2}, t_{R_3}\}$. Die Mengen S, F seien leer.

- Die Abbildung der Regel R_1:
  - Der Bedingungsteil: $S := S \cup \{s_a\}$ und $F := F \cup \{(s_a, t_{R_1}), (t_{R_1}, s_a)\}$
  - Der Folgerungsteil: $S := S \cup \{s_b, s_{\neg_b}\}$ und $F := F \cup \{(s_{\neg b}, t_{R_1}), (t_{R_1}, s_b)\}$
- Die Abbildung der Regel R_2:
  - Der Bedingungsteil: $S := S \cup \{s_c\}$ und $F := F \cup \{(s_c, t_{R_2}), (t_{R_2}, s_c)\}$
  - Der Folgerungsteil: $S := S \cup \{s_d, s_{\neg d}\}$ und $F := F \cup \{(s_{\neg d}, t_{R_2}), (t_{R_2}, s_d)\}$
- Die Abbildung der Regel R_3:
  - Der Bedingungsteil: $S := S \cup \{s_d, s_e\}$ und $F := F \cup \{(s_d, t_{R_3}), (t_{R_3}, s_d), (s_e, t_{R_3}), (t_{R_3}, s_e)\}$
  - Der Folgerungsteil: $S := S \cup \{s_a, s_{\neg a}\}$ und $F := F \cup \{(s_a, t_{R_3}), (t_{R_3}, s_{\neg a})\}$

Die Startmarkierung sei $m_0 = \langle s_a^1, s_{\neg a}^0, s_b^0, s_{\neg b}^1, s_c^1, s_d^0, s_{\neg d}^1, s_e^1 \rangle$. Dann sind die Transitionen t_{R_1}, t_{R_2} aktiviert. Für Transition t_{R_1} gilt $m_0(s_d) = 0 < W(s_d, t_{R_1}) = 1$, daher ist diese Transition in dieser Markierung nicht aktiviert.

Anschaulich ist nun gezeigt worden, dass es möglich ist, Expertensysteme, die mit der Inferenz-Methode "Modus Ponens" arbeiten, in B/E-Netze abzubilden.

Die hier gezeigte Art der Abbildung hat jedoch große Nachteile: Die resultierenden Netze können sehr unübersichtlich werden und bieten Information auf einem recht niedrigen Abstraktionsniveau. Durch geeignet vorgenommene Vergröberungen kann der Abstraktionsgrad eines solchen Netzes erhöht werden. Jedoch wird das resultierende Netz dadurch die Eigenschaften von B/E-Netzen verlieren. Werden verschiedene Stellen zu einer Stelle vergröbert, so werden aus den verschiedenen Fakten, für die Stellen gestanden haben, ein einziger. Außerdem erhöht sich die Kapazität der resultierenden Stelle von 1 auf die Zahl der in dieser Stelle zusammengefassten Original-Stellen. Die Marken, die jetzt in der vergröberten Stelle enthalten sein können, stehen für unterschiedliche Aussagen, die wahr sind. Da Marken dieser Netze aber ununterscheidbar sind, lässt sich nicht feststellen, für welche Aussage eine individuelle Marke steht.

5.1.5.3 Gefärbte Petrinetze

Gefärbte Petrinetze sind für die Darstellung der Verarbeitung von komplexen Datentypen besser geeignet als die S/T-Netze. In gefärbten Petrinetzen können die Marken einer Stelle als Instanzen einer Datenklasse aufgefasst werden. Die Klasse kann man sich durch die Stelle gegeben vorstellen. Damit sind die Stellen Behälter, die Instanzen eines bestimmten Datentypentyps aufnehmen. Die Kantenbewertungsfunktion K wird durch eine Abbildung ersetzt, die jede Kante mit einer Menge von Variablen assoziiert. Die Variablen sind von demselben Datentyp, wie die Stelle, die mit der Kante verbunden ist. Der Schaltvorgang einer Transition kann nun in drei Phasen eingeteilt werden.

1. Aktivierung: Die für das Schalten notwendigen Bedingungen sind erfüllt. Dies heißt, dass Marken aus den Stellen, die die Vorbedingungen darstellen, an die Variablen gebunden werden können, so dass keine Variable frei bleibt. Außerdem muss in allen Stellen, in die die Transition Marken einfügen wird, genügend Platz zur Aufnahme der neuen Marken sein.

2. Die Variablenbindung: In dieser Phase werden die durch die Transition angeforderten Marken an die Variablen der Kanten gebunden. Diese Bindung entspricht einer Wertzuweisung an die Variablen.

3. Das Schalten: In dieser Phase werden Marken in die Ausgangsstellen der Transition eingefügt. Bei den Netzen die im Folgenden zur Modellierung eines Expertensystems mit Fakten- und Regelbasis kann die Einfügung von Marken in Stellen auch eine Instantiierung neuer Marken des entsprechenden Stellentyps beinhalten.

Zusätzlich können auch noch Stellen mit anonymen Marken, wie in den S/T-Netzen, vorkommen. Diese Stellen können dann Kontrollzwecken, der Kapazitätssteuerung oder der Darstellung von Präzedenzen dienen.

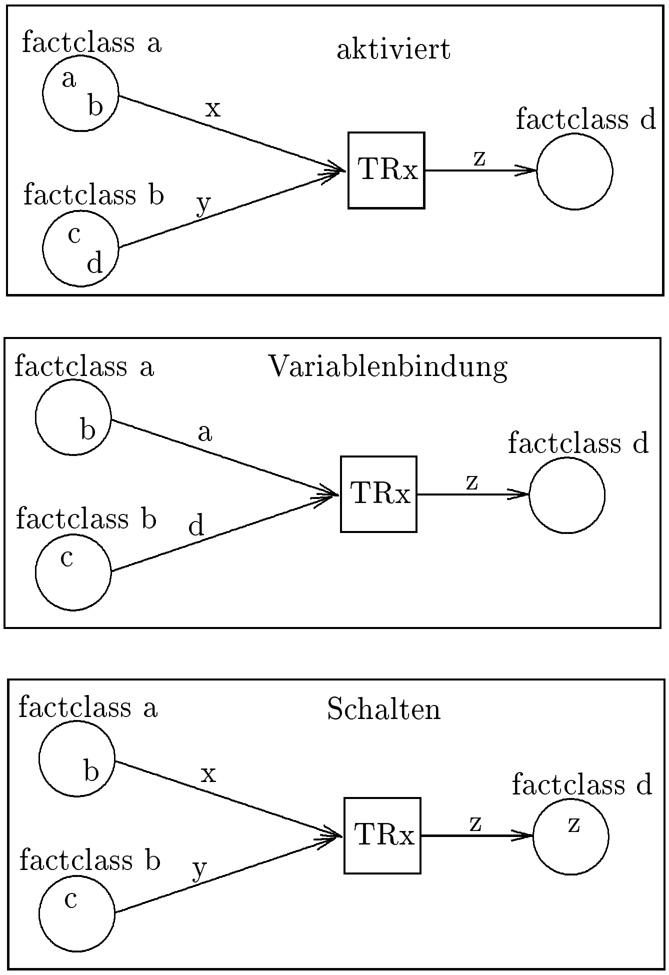

Abbildung 5.4: Eine Transition in einem gefärbten Petrinetz in drei Phasen.

Die Fakten, die aus den Auditdaten konstruiert werden können wie in **??** auf Seite **??** aufgebaut sein. Damit sind die Fakten nicht mehr einfache boolsche Ausdrücke die nur die Werte *wahr* oder *falsch*, bzw. 0 oder 1, annehmen können. Fakten sind unter dieser Betrachtung komplexe Datenstrukturen mit einem sehr großen Wertebereich. Daher sind die Bedingungen, die bei der Prüfung, ob eine Transition aktiviert ist oder nicht, getestet werden müssen auch komplexer. Meist reicht die Bedingung, dass eine Instanz einer bestimmten Art von Fakten vorhanden ist, nicht aus. Es müssen dann noch zusätzliche Informationen abgefragt werden, wie z. B. “`username='root'`”.

5.2 Production-Based Expert System Toolset (P-BEST)

Seit mehr als 10 Jahren wird P-BEST zur Einbruchsentdeckung und zur Entwicklung von IDS eingesetzt. Die ursprüngliche Version von P-BEST stammt von Alan Whitehurst. Es war ein wichtiger Bestandteil des *Multics Intrusion Detection and Alerting System* (siehe 3.5) (MIDAS), welches die Einbruchs- und Missbrauchsentdeckung auf dem mit dem Internet verbundenen Mainframecomputer des National Computer Security Center durchführte. P-BEST wurde später am SRI von Whitehurst verbessert. Es wurde danach von Fred Gilham in IDES (siehe 3.4) und NIDES (siehe 3.6.1) eingebunden.

5.2.1 Die Bestandtteile von P-BEST im Überblick

Das P-BEST Toolset besteht aus einem Regelübersetzer, einer Programmbibliothek mit Laufzeitroutinen und einer Menge von *garbage collector*-Routinen. Die Regeln und Fakten werden in der P-BEST Regel- und Produktionsspezifizierungssprache angegeben. Der Übersetzer *pbcc* übersetzt die spezifizierten Regeln (Produktionen) in ein C-Programm, daß dann zu einem lauffähigem Expertensystem übersetzt werden kann. Dabei gibt es zwei Möglichkeiten:

1. Man erzeugt aus den Routinen ein lauffähiges Programm, das ganz alleine für sich stehen kann,

2. oder man erzeugt eine Programmbibliothek, in der der Kern der P-BEST Inferenzmaschine enthalten ist, die später in ein größeres Programm mit eingebunden wird.

Ein Vorteil dieser Verfahrensweise ist, dass das resultierende System recht schnell ist und sich daher für die Echtzeitanalyse von Auditdaten einsetzen lässt. Ein nicht zu unterschätzender Nachteil dieses Vorgehens ist, dass im Falle einer Änderung der Regeldatenbank das gesamte ID-System neu kompiliert werden muss. Damit ist es nicht so leicht möglich auf neue Angriffstechniken zu reagieren.

5.2.2 Die P-BEST Sprache

Die Wissensbasis des Systems ist die Menge der Regeln, die es kennt (siehe auch 5.1.1. Sie wird auch mit *production memory* bezeichnet. Die Menge Fakten bildet die Faktenbasis (siehe auf Abschnitt 5.1.1). Das System wendet nun sein Wissen auf die Fakten seiner augenblicklichen Faktenbasis an. Die Faktenbasis und die Menge der aktiven Regeln können hierdurch eine Änderung erfahren, denn der verarbeitete Fakt wird in Abhängigkeit der Regeln aus der Wissensbasis durch einen (oder mehrere) neue ersetzt oder aus der Faktenbasis gelöscht. Die P-BEST

Sprache wurde nun entwickelt um Regeln und Fakten für das System zu spezifizieren.

5.2.2.1 Der Mustertyp (ptype)

Ein Fakt wird hier auch als Instanz Mustertyp (*pattern type* oder*ptype*) bezeichnet. Diese Instanzen von Mustertypen werden aus den Auditdaten konstruiert und dann in die Faktenbasis eingebracht. Die Konstruktion der Fakten aus den Auditdaten wird von C-Routinen, die der Benutzer bei Instantiierung des Expertensystems angeben muss, ausgeführt. Dadurch ist das Expertensystem nahezu universell einsetzbar, lastet dem Benutzer aber die Bürde der Programmierung von eigenen C-Routinen zur Faktenkonstruktion auf.

In Abbildung 5.5 soll beispielhaft ein *ptype* mit der P-BEST Sprache definiert werden, der den Namen *event* trägt und aus vier Feldern besteht. Das Wort `ptype` ist ein Schlüsselwort der P-BEST Sprache. Es gibt an, dass der Ausdruck in den eckigen Klammern einen neuen Mustertyp festlegt. Die eckigen Klammern dienen dabei als Begrenzer, die zeigen, wo die Festlegung des Typs beginnt und wo sie endet. Die Definition eines `ptype` in P-BEST entspricht mathematisch der Beschreibung einer Menge. In dem Beispiel von Abbildung 5.5 wird eine Menge `event` beschrieben. Die Elemente der Menge `event` sind in diesem Fall Tupel aus vier Komponenten, wobei die erste Komponente aus der Menge `event_type`, die zweite aus der Menge `return_code`, die dritte aus der Menge `username` und die letzte Komponente aus der Menge `hostname` stammt. Die Menge `event` kann nun so beschrieben werden:

$$\texttt{event} \subset \texttt{event\_type} \times \texttt{return\_code} \times \texttt{username} \times \texttt{hostname}$$

Jetzt fehlt nur noch die Beschreibung der jeweiligen Mengen. Diese Mengenbeschreibung folgt auf den Doppelpunkt nach dem Namen der Komponenten des `ptype`. Dies ist auch eine Beschreibung der internen Repräsentation der Daten. Mit `event_type:int` wird angegeben, dass die Komponente `event_type` ein Datum des Typs `int` beinhalten soll. Das Wort `int` ist ein Schlüsselwort der Programmiersprache C und bezeichnet einen vorzeichenbehafteten einfachen Ganzzahligen Datentyp. In welchem Wertebereich die Zahlen dieses Typs liegen ist von der Architektur des Systems und dem Betriebssystem bzw. dem Compiler abhängig.

```
ptype[event event_type:int,
    return_code:int,
    username:string,
    hostname:string]
```

Abbildung 5.5: Definition des *ptype* `event`

Auf vielen X-Bit-Architekturen ist der Wertebereich von Zahlen dieses Typs das Intervall $[-2^{X-1}, 2^{X-1} - 1]$, wobei X Typischerweise ein Wert aus der Menge $\{16, 32, 64\}$ ist.

Die Komponente `return_code` soll ebenfalls durch eine Zahl vom C-Typ `int` repräsentiert werden. Die nächsten beiden Komponenten `username` und `hostname` sollen Zeichenketten (Strings) beinhalten. In der Programmiersprache C gibt es jedoch keinen Datentyp `string`. In C werden Zeichenketten in der Regel einfach als Felder (Arrays) von Zeichen (character) implementiert. Eines der wesentlich Mankos in C ist, dass die Sprache keine Elemente besitzt, die eine Bereichsüberschreitung von Feldindizes abfangen. Daher ist es in C sehr leicht möglich, fehlerhafte Programme zu schreiben, die zu einer Speicherbereichsverletzung führen.

Diese Definition dient der Konstruktion von Fakten aus den Auditdaten. Fakten können auch, als Folgerung, von der Inferenzmaschine selbst konstruiert werden. In Abbildung 5.6 wird, als ein weiteres Beispiel für eine Faktendefinition, ein

```
ptype[bad_login username: string,
                hostname: string]
```

Abbildung 5.6: Beispieldefinition eines Faktes namens `bad_login`

Fakt namens `bad_login` definiert, der aus zwei Zeichenketten besteht. Begreift man diese Definition wieder als eine Beschreibung einer Menge, so gilt für die neue Art von Fakt:

$$\texttt{bad\_login} \subset \texttt{string} \times \texttt{string}$$

Das Beispiel in Abbildung 5.7 zeigt, wie ein Fakt dieser Art aus einem anderen konstruiert werden kann. Die erzeugten Fakteninstanzen werden in die Faktenbasis eingefügt.

5.2.2.2 Regeln oder Produktionen

Regeln bestehen aus zwei großen Teilen:

Voraussetzung: Hier werden die Bedingungen angegeben, die erfüllt sein müssen, damit die Regel zur Anwendung kommen kann.

Folgerung: Hier werden die Handlungen angegeben, die ausgeführt werden, wenn die Bedingungen der Voraussetzung erfüllt sind. Die wichtigsten Handlungen sind die der Konstruktion, bzw. Instantiierung neuer Fakten, deren Einfügen in die Faktenbasis und die Löschung von nicht mehr benötigten Fakten aus der Faktenbasis. Darüber hinaus gibt es noch Aktionen, die die

Faktenbasis nicht verändern, z. B. Aufrufe von Systemfunktionen und Aktionen, die zwar den Bestand an Fakten nicht ändern, aber Änderungen an Systemvariablen durchführen.

Der Inferenzmechanismus. Die Faktenbasis wird von der Inferenzmaschine daraufhin untersucht, ob es zu den vorhandenen Fakten passende Regeln (Produktionen) gibt. Dies ist äquivalent zu der Frage, ob in der Faktenbasis ein Fakt f existiert, so dass es eine Regel aus der Menge der Regeln gibt, in der f an der ersten Stelle vorkommt. Seien F die Faktenbasis und R die Regelbasis so kann die Frage als Formel notiert werden:

$$\exists f \in F : (f \rightarrow q) \in R$$

Dieser Mechanismus setzt die im Abschnitt 5.1.2.2 beschriebene Inferenzmethode "Modus Ponens" um.

```
rule[Bad_Login(#10;*):
   [+e:event | event_type == login,
               return_code == BAD_PASSWORD]
==>
   [+bad_login | username = e.username,
                 hostname = e.hostname]
   [- | e]
   [! | printf("Bad login for user %s from host %s\n",
                e.username, e.hostname)]
]
```

Abbildung 5.7: Definition einer Regel in der P-BEST Sprache

Regeldefinition. Eine Regeldefinition beginnt mit dem Schlüsselwort `rule` gefolgt von einer öffnenden eckigen Klammer. Darauf folgt der Name der Regel. Zu jeder Regel gehört eine Liste von Optionen. Diese wird nach dem Regelnamen innerhalb von runden Klammern angegeben. Die einzelnen Optionen werden dabei durch ein Semikolon getrennt. Nachdem der Regelname und die Optionen angegeben wurden, folgt ein Doppelpunkt. Nun folgt mindestens ein in eckigen Klammern stehender Ausdruck. Das Zeichen `==>` zeigt an, dass nun der erste Teil der Regel, in dem die Voraussetzung definiert wird, zuende ist. Es folgt jetzt die Folgerung der Voraussetzung. Die Folgerung besteht, wie die Voraussetzung wieder aus einer Reihe von (mindestens einem) mit eckigen Klammern umfassten Ausdrücken. Ganz am Ende der Regeldefinition steht eine schließende eckige Klammer als passendes Gegenstück zu der öffnenden Klammer nach dem Wort `rule` in der ersten Zeile der Regeldefinition. Eine Regeldefinition hat nun die folgende Struktur:

$$\texttt{rule[}\langle Name\rangle\texttt{(}\langle Optionen\rangle\texttt{):}\langle Voraussetzung\rangle\texttt{==>}\langle Folgerung\rangle\texttt{]}$$

Regelnamen. Der Name einer Regel kann frei gewählt werden. Es darf in einer Regelbasis ein Name nur mit einer Regel assoziiert sein. Regelnamen müssen von den Schlüsselwörtern verschieden sein.

Optionen. Die einzigen Optionen, die bei P-BEST verwendet werden, wenn das Expertensystem in ein Intrusion-Detection-System eingebunden wird, sind die Priorität, die Wiederholungsoption und die Option, die besagt, dass die Regel zu Beginn nicht aktiv ist. Die Priorität hat folgendes Aussehen:

$$\texttt{\#}\langle Nummer\rangle$$

Eine Nummer ist dabei eine ganze Zahl. Regeln, deren Priorität hoch ist (eine große Zahl), werden bei der Anwendung zuerst berücksichtigt. Die Wiederholungsoption besteht aus dem Zeichen `*`. Soll eine Regel nicht von Anfang an zu den aktiven Regeln der Regelbasis gehören, so gibt man das durch das Zeichen `-` an.

Die Voraussetzung. Die Voraussetzung einer Regel besteht aus einer Liste von Bedingungen. Ein Beispiel für eine Bedingung ist in Abbildung 5.8 gegeben.

```
[+e:event | event_type == login, return_code == BAD_PASSWORD]
```

Der Ausdruck liest sich wie folgt:

> „Es existiert ein Fakt vom Typ `event`, das wir an dieser Stelle mit `e` Bezeichnen. Für diesen Fakt gilt, das seine Komponente `event_type` gleich `login` ist und das die Komponente `return_code` gleich `BAD_PASSWORD` ist.“

Abbildung 5.8: Beispiel für einen Ausdruck, der nach einem bestimmten ptype fragt.

Das Zeichen `+` dient hier als Existenzquantor $\exists$. Der obige Text lässt sich als Formel darstellen:

$$\exists \texttt{e} \in \texttt{event} | \texttt{event\_type} = \texttt{login} \wedge \texttt{return\_code} = \texttt{BAD\_PASSWORD}$$

Das `e` dient als Aliasname für ein Fakt vom Typ `event`. D. h., für den Zeitraum der Regelanwendung wird die gefundene Instanz des Faktes an diesen Namen gebunden. Wird anstelle des `+` ein `-` verwendet, so steht das Minuszeichen für $\nexists$, die Negation des Existenzquantors.

Der Ausdruck in Abbildung 5.8 ergibt den Wert *wahr*, wenn ein `event` in der Faktenbasis existiert, dass den Kriterien `event_type` gleich `login` und `return_code` gleich `BAD_PASSWORD` genügt.

Das Fragezeichen dient als Frageoperator in der Form: `[?|...]`. Werte von Feldern eines Faktes, von Variablen und Ergebnisse von Funktionsaufrufen können mit diesem Operator abgefragt werden.

Von der Definition des Mustertyps `event` aus Abbildung 5.5 ist bekannt, dass `event_type` und `return_code` durch zwei Zahle des C-Typs `int` repräsentiert werden. Somit müssen auch `login` und `BAD_PASSWORD` Zahlen des gleichen Typs sein. In diesem Beispiel sind es die symbolischen Namen von bestimmten schon vorher irgendwo im System festgelegten Zahlen, also Zahlenkonstanten. Damit ist die Voraussetzung der Regel aus Abbildung 5.7 zuende besprochen.

Die Folgerung. In diesem Teil der Regeldefinition werden jetzt die Aktionen angegeben, die die Anwendung dieser Regel beinhaltet. Die Aktionen teilen sich in zwei Kategorien:

- Aktionen, die auf Ziele innerhalb des Expertensystems gerichtet sind.
- Aktionen, die auf Ziele außerhalb des Expertensystems gerichtet sind.

Zur ersten Kategorie zählen Aktionen, die Fakten aus der Faktenbasis löschen, einfügen, Feldwerte eines Faktes ändern, Regeln aktivieren oder Regel deaktivieren. Die zweite Kategorie beinhaltet den Aufruf von C-Funktionen, über die letztlich das gesamte Computersystem gesteuert werden kann. Alle Aktionen verändern den Zustand des Expertensystems und des Computersystems! Bei Aktionen der ersten Kategorie ist klar, dass jede Änderung des Zustandes des Expertensystems auch eine Zustandsänderung des Computerssystems bedeutet. Bei den Aktionen der zweiten Kategorie wird deren Wirkung auf das Computersystem über die Auditdaten auch eine Zustandsänderung des Expertensystems nach sich ziehen.

Der erste Ausdruck in der Folgerung in Abbildung 5.7 bewirkt die Instantiierung eines Fakts vom Typ `bad_login` wie er in Abbildung 5.6 definiert wurde. Die neue Instanz vom Typ `bad_login` wird in die Faktenbasis eingefügt. Der nächste Ausdruck in der Folgerung der Regel aus Abbildung 5.7 bewirkt die Löschung des

Fakts `e` aus der Faktenbasis. D.h. das Minuszeichen stellt hier einen Löschungsoperator (*delete operator*) dar.

Die Aktivierung und Deaktivierung von Regeln. Die Regelbasis besteht im laufenden Programm aus zwei Klassen von Regeln:

1. Aktiven Regeln
2. Inaktiven Regeln

Die Deaktivierung einer Regel wird mit dem Befehl `[-#`⟨*Regelname*⟩`]` erreicht. Entsprechend bewirkt der Befehl `[+#`⟨*Regelname*⟩`]` die Aktivierung einer Regel. Diese Möglichkeit wurde vor allem wegen der Möglichkeit der Leistungsoptimierung in P-BEST eingeführt. Denn die Überprüfung der Regeln auf ihre Anwendbarkeit auf die Fakteninstanzen in der Faktenbasis kostet Zeit.

Die Markierung von Fakten durch Regeln. Regeln können Fakten, auf die sie angewendet wurden, mit Markierungen versehen. Die Markierungen können auch wieder durch Regeln entfernt werden. Der Befehl `[$|e:`⟨`Markierung`⟩`]` markiert ein Fakt, das hier mit `e` referenziert wurde. Der Befehl `[^|e:`⟨`Markierung`⟩`]` entfernt die Markierung von einem Fakt.

Diese Markierungen können auch getestet werden. Der Ausdruck `[+e|:event$ TESTED]` überprüft, ob es in der Faktenbasis eine Instanz eines Faktes des Typs `event` gibt, die mit der Markierung `TESTED` versehen ist. Der Befehl `[+e|:event^ TESTED]` prüft ob eine Instanz des Typs `event` gibt, die noch nicht mit `TESTED` markiert wurden.

Der Aufruf von Standard-C-Funktionen. Der letzte Ausdruck in der Regel aus Abbildung 5.7 bewirkt einen Funktionsaufruf aus der Standard- Ein- Ausgabebibliothek von C. Durch Ausdrücke der Art `[!|...]` können aus den Regeln des Expertensystems heraus Aufrufe von Betriebssystemfunktionen gemacht werden. Diese Aufrufe ans Betriebsystem werden durch die Standart-C-Bibliothek vermittelt.

Produktionen. Die Abarbeitung der Regel entspricht in Bezug auf die Faktenbasis einem Zustandsübergang. Sei Z der Zustand vor Ausführung der Regel r und Z' der Zustand nach Ausführung der Regel r, so lässt sich der Zustandsübergang wie folgt veranschaulichen:

$$Z \overrightarrow{r} Z'$$

Dabei müssen Z, Z' wohldefinierte Zustände der Faktenbasis sein. D. h., Z' darf sich nur durch die von der Regel r verursachten Unterschiede von Z unterscheiden.

Unter dieser Modellvorstellung muss die Abarbeitung einer Regel der Regelbasis in Bezug auf die Faktenbasis als unteilbare Handlung geschehen.

Die Änderung des Zustandes der Faktenbasis durch die Anwendung einer Regel ist der Produktionsregel einer Grammatik ähnlich. Die Herleitung von Sätzen einer Sprache mittels einer Grammatik entspricht jedoch einer Top-Down-Strategie, während das Expertensystem P-BEST mit der Bottom-Up-Strategie arbeitet. Erstellt man jedoch die Regelbasis, so wird man sich zuerst darüber im Klaren sein, was das Expertensystem erkennen soll. Dies wird man in Fakten abbilden. Danach werden schrittweise Überlegungen zeigen, wie diese Fakten auf andere zurückzuführen sind, bis hinunter zu den Daten, die in den Auditspuren zu finden sind.

5.2.2.3 Benutzerdefinierte Funktionen und Variablen.

Es ist auch möglich, benutzerdefinierte Funktionen über diese Schnittstelle aufzurufen. Benutzerdefinierte Funktionen sind in diesem Zusammenhang alle C-Funktionen, die dem P-BEST-System nicht a priori bekannt sind. Um eine solche selbstdefinierte C-Funktion aufrufen zu können, muss diese als ein externer Typ `xtype` dem System bekannt gemacht werden. Die Definition in Abbildung 5.9 zeigt eine `xtype`-Definition, die eine benutzerdefinierte C-Funktion, deren Rückgabewert eine Zahl vom C-Typ `int` ist, definiert. Die möglichen Aufrufparameter

```
xtype[test_stream: intfunc]
xtype[file_status: int]
```

Abbildung 5.9: P-BEST-Definition zweier `xtype`

erkennt der Compiler *pbcc* an der Art, wie die Funktion aufgerufen wird. D. h., wird in der Folgerung einer Regel der Aufruf

```
[!|'file_status=test_stream(23)]
```

getätigt, so erkennt der *pbcc*, dass es sich bei der Funktion `test_stream` um eine Integerfunktion mit einem Aufrufparameter vom Typ Integer handelt.

Das direkte Einbinden von C-Quellcode in P-BEST-Dateien. P-BEST gestattet es, C-Quellcode direkt in die P-BEST-Programme einzugeben. Durch eine öffnende geschweifte Klammer zeigt man dem Compiler an, dass an dieser Stelle ein Abschnitt mit C-Code beginnt. Die schließende geschweifte Klammer zeigt an, dass dieser Bereich zu Ende ist.

5.3 Die Sprache von ASAX: RUSSEL

Die Sprache, in der für das System ASAX (Kapitel 3.6.7) die Regeln für die Regelbasis spezifiziert werden, wird in diesem Abschnitt genauer erläutert.

5.3.1 Die wichtigsten Merkmale der Sprache RUSSEL

Der Name leitet sich von "**Ru**le ba**S**ed **S**equence **E**valuation **L**anguage" ab. In dem Namen steckt also die Information, dass RUSSEL speziell zur Analyse von sequentiellen Auditspuren erschaffen wurde.

Regelbasiert und bottom-up RUSSEL ist regelbasiert, denn die wichtigste Gruppierung von Anweisungen werden Regeln genannt. Eine Regel besteht dabei aus einem Namen und verschiedenen Aufrufparametern. Dann folgen eine Reihe von Anweisungen. RUSSEL kennt geschützte Anweisungen (guarded commands), wie sie in vielen imperativen Sprachen bekannt sind.

RUSSEL ist *bottom-up*, denn am Anfang steht die Sammlung von Fakten aus den Auditdaten. Diese Fakten dienen dann als Eingabe der aktiven Regeln. Die Regelanwendung kann dazu führen, dass die Ausgangsfakten durch neue abgeleitete Fakten ersetzt werden. Aus den *primitiven* Auditdaten werden so schrittweise immer komplexere Fakten aufgebaut.

Imperativer Stil Regeln werden in dieser Sprache in einem imperativen Programmierstil angegeben. Eine Regeldeklaration besteht dabei aus einem Namen, einer Menge von Parametern, einer Menge von Variablen und einem Aktionsteil. Eine Regel erscheint damit einer Prozedur mit Aufrufparametern sehr ähnlich. Daneben gibt es in RUSSEL die klassischen Kontrollstrukturen imperativer Programmiersprachen.

Modularität. Mehrere zusammenhängende Regeln können in Modulen zusammengefasst werden. Dabei können innerhalb eines Moduls globale, d.h. Variablen, die auch außerhalb des Moduls sichtbar sind, und lokale Variablen, d.h. Variablen, die nur innerhalb eines Moduls sichtbar sind deklariert werden. Genauso lassen sich auch externe und interne Regeln deklarieren. Für jedes Modul existiert eine Datei mit Regeln. RUSSEL benutzt damit die Modulabstraktion [Lou94].

5.3.2 Module

Eine Regelbasis wird unter RUSSEL auch als ein Programm bezeichnet. So ein Programm besteht aus verschiedenen Modulen. Jedes Modul steht in einer eigenen

Datei. Ein Modul teilt sich in Bereiche:

1. Die Liste der benutzten Module
2. einer globalen Variablendeklaration
3. einer Liste mit Regeldeklarationen
4. einem Initialisierungsteil

Die Liste der benutzten Module wird mit dem Schlüsselwort `uses` eingeleitet. Darauf folgt dann eine Liste mit Modulnamen, die durch Kommata getrennt sind. Nach dem letzten Modulnamen folgt ein Semikolon.

Die globalen Variablendeklarationen beginnen mit dem Wort `global` gefolgt von einem Klassenbezeichner. In RUSSEL gibt es zwei Klassen von Variablen, externe und interne. Dementsprechend kann der Klassenbezeichner das Wort `internal` oder `external` sein. Hierauf folgt dann die Liste der durch Kommata getrennten Variablennamen. Nach dem letzten Namen folgt ein Doppelpunkt. Daran schließt sich die Typangabe an.

Der Regeldeklarationsteil, der sich nun anschließt, wird in Abschnitt 5.3.4.2 genauer betrachtet.

Zum Schluss kommt der Initialisierungsteil. Er wird durch die Angabe des Schlüsselwortes `init_action;` dem System angezeigt. Er besteht aus zwei Teilen, in dem ersten werden die Variablen deklariert, in dem zweiten die Aktionen angegeben.

5.3.3 Datentypen

In RUSSEL werden nur zwei Datentypen, namentlich Zeichenketten (**string**) und Integerzahlen (**integer**), verwendet. Zeichenketten werden als Folgen von Bytes (8 Bit Werte) repräsentiert. Integerzahlen sind 32 Bit lang und werden als Folge von vier Bytes repräsentiert. Der Wertebereich ist das Intervall $[-2^{31}[2^{31}-1]$. Die Operationsprimitiven für Zeichenketten sind $<, <=, >, >=, =$ und $!=$. Sie stehen für kleiner, kleiner gleich, größer, größer gleich, gleich und ungleich. Der Integerdatentyp kennt zusätzlich zu den Vergleichsoperationen $<, <=, >, >=, =$ und $!=$ noch die Operationen $+, *, \mathbf{mod}, \mathbf{div}$, die für die Addition, die Multiplikation, die modulo Division und die ganzzahlige Division ohne Rest stehen. Hinzu kommt noch ein spezieller Wert **undef**, der für nicht definierte Zahlenwerte steht.

Die Beschränkung auf nur diese beiden Datentypen und die relativ wenigen vordefinierten Operationen erlaubt der Sprache so einfach wie möglich und dennoch anwendbar auf reale Problemstellungen zu sein [Mou97]. Der Datentyp kann als uninterpretierte Kette von Bytes aufgefasst werden. Dies ermöglicht es, das

Zusammengesetzte Datentypen aus der Programmiersprache C in RUSSEL als **string** zu repräsentieren.

5.3.4 Deklarationen

In diesem Abschnitt wird die syntaktische Struktur von RUSSEL-Modulen erklärt. Dabei wird jedoch nur die Syntax der Deklaration von globalen Variablen, Regeln und Modulen erklärt. Dies ist nötig für die sich anschließenden Erläuterungen der Semantik von RUSSEL-Befehlen.

5.3.4.1 Die Deklaration globaler Variablen

Der Deklarationsteil für globale Variablen beginnt mit dem Schlüsselwort **external** oder **internal**. Dann folgt das Schlüsselwort **global**.

5.3.4.2 Regeldeklaration

Eine Regeldeklaration besteht aus drei Teilen:

1. Dem Regelkopf.
2. Dem Variablendeklarationsteil.
3. Dem Aktionsteil.

Der Kopfteil einer Regel. Die Deklaration einer Regel wird durch das Schlüsselwort `rule` eingeleitet. Darauf folgt die Angabe eines Namens. Dem Namen folgt optional eine von runden Klammern eingeschlossene Liste von Parametern. Vor dem Schlüsselwort `rule` kann noch angegeben werden, ob die Regel lokal, d. h. nur innerhalb dieses Moduls, oder global, d. h. auch außerhalb des Moduls angesprochen werden kann. Dies wird durch die Angabe von entweder `external` oder `internal` gemacht.

Die Parameterliste besteht aus einer Liste von durch Semikolon getrennten Parametergruppen. Eine Parametergruppe besteht aus einer Liste von Parameternamen, die mit einer Typangabe endet.

```
internal rule chownmod(flag: integer);
```

Abbildung 5.10: Deklaration einer internen Regel

Der Variablendeklarationsteil einer Regel. Die Variablendeklaration wird mit dem Schlüsselwort `var` eingeleitet. Darauf folgt eine Liste von durch Semikolons getrennten Variablendeklarationen. Eine Variablendeklaration besteht ihrerseits aus einer Liste von durch Kommata getrennten Bezeichnern und einer Typangabe. Die Typangabe besteht aus einem Doppelpunkt gefolgt von einem der Schlüsselworte `integer` oder `string`.

Der Aktionsteil. Hier endlich beginnt der Körper der Regel, in dem die Handlungen definiert werden. Die einfachste Handlung ist die, nichts zu tun. Hierfür gibt es das Schlüsselwort `skip`. Ansonsten können Handlungen aus Wertzuweisungen von Variablen bestehen, aus geschützten Anweisungen, Schleifen, zusammengesetzten Handlungen, dem Anstoßen von Regelanwendungen oder dem Aufruf von vordefinierten (oder durch den Benutzer definierten) Prozeduren. Abbildung

```
rule nonsens;
skip.
```

Abbildung 5.11: Beispiel eines vollständigen RUSSEL-Moduls

5.11 zeigt das kürzeste mögliche Modul in RUSSEL. Hat man mehr als eine Anweisung in einer Regel, so müssen diese zwischen den Schlüsselwörtern `begin` und `end` stehen. Ein Beispiel wird in Abbildung 5.12 gegeben. Das Beispiel in Abbildung 5.12 enthält zwei durch `begin` und `end` geklammerte zusammengesetzte Handlungen. Die erste Klammerung umrahmt den gesamten Regelkörper. Die zweite Klammerung fasst mehrere Handlungen zu einer geschützten Anweisung in einer Schleife zusammen. Die durch `do` und `od` eingefassten Anweisungen werden so lange ausgeführt, wie die Bedingung `d` > `1` gegeben ist. Die Bedingung ist der "Wächter" der dann folgenden zusammengesetzten Anweisung.

Der wichtigste Befehl ist `trigger off`, da dieser die Regelanwendung auf die Auditdaten anstößt. Dieser Befehl kennt drei verschiedene Arbeitsmodi (triggering modes):

`for_current`: Dieser Modus bezieht sich auf die Auditdaten, die der gerade aktuelle Datensatz beinhaltet.

`for_next`: Dieser Modus sorgt dafür, dass ein neuer Datensatz, sofern vorhanden, eingelesen und dann ausgewertet wird.

`at_completion`: Hierdurch wird die angegebene Regel erst auf die Daten angewendet, wenn das Ende des Auditdatenstromes erreicht wurde.

```
global fibon:integer;
/** Es wird die globale Variable fibon deklariert.
  * Sie dient als Auffangbehaelter fuer das Ergebnis.
  */
external rule fibonacci(n:integer);
var a,b,c,d :integer;
begin                  /* Begin der ‘compound action’ */
  a:=0;
  b:=1;
  d:=n;
  do                   /* Beginn einer Schleife */
    d>1 --> begin      /* Beginn der zweiten ‘compound action’ */
              c:=b;
              b:=a+b;
              a:=c;
              d:=d-1
            end        /* Ende der zweiten ‘compound action’ */
  od;                  /* Ende einer Schleife */
  fibon:=b;
  println('Result:',b)
end;                   /* Ende der ersten ‘compound action’ */

init_action;
trigger off for_current fibonacci(9).
/* Der Befehl ‘‘trigger off’’ stoesst die Regelanwendung an! */
```

Abbildung 5.12: Ein Modul, das Fibonacci-Zahlen berechnet.

Bedingungen. Bedingungen steuern die Regelanwendung. Wenn eine gegebene Bedingung wahr ist, dann werden die folgenden Aktionen ausgeführt. Daher spielen Bedingungen (conditions) eine wichtige Rolle bei der Programmierung von Regeln. Unter RUSSEL sind Bedingungen Disjunktionen oder Konjunktionen von Disjunktionen oder Konjunktionen. Im einfachsten Fall ist eine Konjunktion (oder Disjunktion) einer der Werte aus der Menge $\{\texttt{true}, \texttt{false}\}$. Ansonsten besteht eine Konjunktion aus Bedingungen, die mittels des Operators `and` miteinander verbunden werden. Eine Disjunktion ist, ähnliche einer Konjuntion, eine Verbindung von Bedingungen mittels des Operators `or`. Die Operatoren `and` und `or` verhalten sich wie die logischen Verknüpfungen $\wedge$ und $\vee$.

Einfache Bedingungen haben die Gestalt wie $\texttt{d} > \texttt{1}$ aus Abbildung 5.12. Diese Bedingung wird als relationaler Ausdruck (relational expression) bezeichnet. Dieser Ausdruck fragt, ob die Relation $>$ zwischen den beiden Ausdrücken links und rechts des Zeichens gegeben, also wahr ist. Für relationale Ausdrücke stehen die relationalen Operatoren $<, >, <=, >=, =, ! =$ zur Verfügung. Die Operatoren stehen jeweils für "kleiner", "größer", "kleiner gleich", "größer gleich", "gleich" und "nicht gleich". Diese Operatoren können auch auf Zeichenketten angewendet werden. Bei Zeichenketten stehen sie für die Aussage, ob die erste Zeichenkette lexikalisch vor der zweiten oder nach der zweiten kommt. Bei Zeichenketten gibt es auch noch einen besonderen Vergleichsoperator $\% =$, der bei dem Vergleich von Zeichenketten die Leerzeichen außer Acht lässt. So ist der Ausdruck `'Name '%='Name'` wahr. Dieser Operator wird in dem Modul aus Abbildung 5.15 verwendet.

Die Existenz eines bestimmten Datenfeldes in einem Auditdatensatz kann durch das Prädikat `present` gefolgt von einem Feldnamen abgefragt werden. So fragt z. B. der Ausdruck `present user_id`, ob ein Feld namens `user_id` in den Auditdaten existiert. Dabei ist eine Feldangabe wie `user_id` die symbolische Angabe einer numerischen Feldidentifikationsnummer. Die Umsetzungstabelle von symbolischen zu numerischen Feldidentifikationen steht in einer Auditdatenbeschreibungsdatei (siehe auch 3.6.7). Das Expertensystem "weiß" aus der Auditdatenbeschreibungsdatei, was diese Feldangabe bedeutet.

Ein RUSSELPROGRAMM. Im Folgenden wird ein vollständiges RUSSEL-Programm vorgestellt. Die Module sind alle bei den Quellen von ASAX als Demonstrationsbeispiele enthalten. Das Hauptmodul des Einbruchsentdeckungspro-

```
uses fscorrupt, nbbadlogin;.
```

Abbildung 5.13: Das Hauptmodul der Regelbasis

grammes für ASAX enthält lediglich die Anweisung `uses` gefolgt von einer Liste der benutzten Modulnamen.

Das erste angegebene Modul dient der Erkennung von Veränderungen von Systemdateien. Im Kopfteil des Moduls wird eine für das Modul globale Integervariable deklariert. Sie wird von den Regeln als Zählervariable eingesetzt. Dieses Modul besteht aus zwei Regeln, `echo_total` und `fscorrupt`. Die Regel `fscorrupt` besitzt eine geschützte Anweisung und den Befehl, sich selbst auf den nächsten anstehenden Datensatz anzuwenden. Der Wächter der geschützten Anweisung (der Bedingungsteil) ist eine Konjunktion zweier Disjunktionen. In den Disjunktionen wird die benutzerdefinierte Funktionsprozedur `match` aufgerufen. Diese Funktion vergleicht zwei Zeichenketten miteinander und liefert bei Gleichheit als Wert die Zahl eins zurück.

Das nächste Modul, welches durch das Hauptmodul `main` eingebunden wird, ist

Feb 18, 01 18:54 **fscorrupt.asa** Seite 1/1

```
global n: integer;

rule fscorrupt;
begin
   if
      (     match('create', event) = 1
        or  match('access', event) = 1
        or  match('write' , event) = 1
      )
        and
      (    match('/sbin/',    au_file_name)=1
        or match('/sbin/',    au_name_dir)=1
        or match('/usr/bin/', au_file_name)=1
        or match('/usr/bin/', au_name_dir)=1
        or match('/bin/'    , au_file_name)=1
        or match('/bin/',     au_name_dir)=1
      )
      -->begin
           println('');
           println('___________________________________________________________');

           print('attempt to corrupt system file', X'09', ': ');
           if
              present au_file_name --> println(au_file_name);
              present au_name_dir  --> println(au_name_dir)
           fi;
           println('by user',     ': ', strToInt(auid));
           println('at the time', ': ', gettime(au_time));
           println('on the host', ': ',  hostname);
           n := n + 1
           end
    fi;
    trigger off for_next fscorrupt
end;

rule echo_total;
begin
        println('');
        println('====================================');
        println(n, ' Attempts to corrupt system files');
        println('====================================')
end;

init_action;
begin
        n := 0;
        trigger off for_next fscorrupt;
        trigger off at_completion echo_total
end.
```

Abbildung 5.14: Dieses Modul dient der Entdeckung von Änderungen an Systemdateien

das Modul `nbbadlogin`. Die Regeln dieses Moduls überwachen die fehlgeschlagenen Loginversuche. Dieses Modul besitzt drei Regeldeklarationen. Die erste Regel, namens `failed_login`, überprüft den aktuellen Auditdatensatz auf erfolglose Loginversuche. Ist einer gefunden, so wird die Regel `count_rule` aufgerufen. Diese Regel überwacht die folgenden Datensätze im Hinblick auf weitere Loginfehler, die sich auf das gleiche Benutzerkonto beziehen. Wird ein Loginfehler erkannt, und ist außerdem die Zeitspanne zu dem vorhergehenden Loginfehler klein genug, so wird ein Zähler heruntergezählt. Hat dieser Zähler den Wert 0 erreicht, so wird eine Warnung auf den Bildschirm des Administrators gedruckt. Ansonsten wird wieder die Regel `failed_login` für den nächsten Datensatz angestoßen.

nbbadlogin.asa Seite 1/1

```
global v: integer;

rule failed_login(max_times, duration: integer);
begin

   if    record_type %= 'TEXT'
     and event       %= 'login_logout'
     and au_text_2   %= 'login'
     and au_text_4   %= 'incorrect password'
     --> begin
           trigger off for_next
            count_rule(au_text_3,
                       strToInt(au_time)+duration,max_times-1)
         end;
     true
      --> trigger off for_next failed_login(max_times, duration)
    fi
end;

rule count_rule(target_uid: string; expiration, count_down: integer);
begin
   if    record_type  %= 'TEXT'
     and event        %= 'login_logout'
     and au_text_2    %= 'login'
     and au_text_4    %= 'incorrect password'
     and au_text_3     = target_uid
     and strToInt(au_time) < expiration
     -->
       if count_down > 1
          --> begin
                trigger off for_next count_rule(target_uid,
                                  expiration, count_down-1)
             end;
          count_down = 1
          --> begin
                v := v + 1;
                println('');
               print('3 FAILED LOGINS ON ',
                     target_uid, '(',  strToInt(auid),')');
               print(', AT ', gettime(au_time), ' from ', au_text_6);
               trigger off for_next failed_login(3,120);
               println('');
               println('')
             end
       fi;

      strToInt(au_time) > expiration
        -->
           trigger off for_next
               failed_login(3,120);
      true
        --> trigger off for_next
               count_rule(target_uid,expiration, count_down)

   fi
end;

rule echo;
begin
       println('    ');
       println('==================================');
       print(v);
       println(' sequence(s) of failed login found');
       println('==================================')
end;

init_action;
begin
     v := 0;
     trigger off for_next failed_login(3,120);
     trigger off at_completion echo
end.
```

Abbildung 5.15: Modul zum Zählen der fehlgeschlagenen Loginversuche

Die letzte Regel `echo` wird aufgerufen, wenn das Ende des Auditdatenstromes erreicht wurde. Sie gibt nur noch einmal aus, wieviele fehlerhafte Loginereignisse in den Auditdaten gefunden wurden.

5.3.4.3 Resumee

Die Sprache RUSSEL zeigt in der Syntax und in dem modularen Aufbau ihrer Programme eine äußerliche Ähnlichkeit mit der Programmiersprache Modula. Auch die Struktur der Regeln, die äußerlich wie Prozeduren einer imperativen Programmiersprache erscheinen trägt zu diesem Eindruck bei.

Die Regelanwendung wird von den Regeln selbst angestoßen. Bei der Regelanwendung werden die Datenfelder eines Auditdatensatzes wie Variablen angesprochen. Jedoch können ihnen keine Werte zugewiesen werden. Jede Regel bekommt bei der Regelauswertung eine eigene Kopie der Daten. Auf diese Weise werden die Daten eines Auditdatensatzes, wie bei einem Prozeduraufruf, per Wert an die Regeln übergeben.

Es ist den Regeln nicht direkt möglich, alle Fakten der Faktenbasis zu sehen. Dies kann nur erreicht werden, indem die Werte der Fakten an globale Variablen übergeben werden.

Lässt man die Beschränkung des Wertebereiches der Integerzahlen von RUSSEL außer Acht, dann ist die Sprache turing-vollständig. Dies ist eine Voraussetzung dafür, alle erdenklichen sequentiellen Muster in den Auditdaten erkennen zu können.

5.4 Vergleich von P-BEST und RUSSEL

Nachdem in den vorangegangenen Abschnitten die Arbeitsweise von Expertensystemen und die beiden Regelsprachen P-BEST und RUSSEL vorgestellt wurden, wird in diesem Abschnitt ein Vergleich zwischen den beiden Sprachen durchgeführt.

5.4.1 Die Unterschiede

5.4.1.1 Einleitung

Der Sprachenvergleich in diesem Kapitel wird anhand von Gegenüberstellungen von vergleichbaren Fakten, Anweisungen, Abstraktionsmechanismen und Konzepten durchgeführt. Die Mächtigkeit beider Sprachen wird intuitiv bestimmt.

5.4.1.2 Fakten

RUSSEL. In RUSSEL sind die Fakten Variablen von einfachen Datentypen. Damit können Fakten unter RUSSEL entweder vom Typ "`string`" oder "`integer`" sein. Eine Regel erhält immer nur Kopien der Fakten. Damit Besitzt jede Regel ihre eigene Faktenbasis. Diese Faktenbasis enthält nur die Fakten des Auditdatensatzes, auf den die Regel gerade angesetzt wurde. Sollen Fakten für eine spätere Bearbeitung gespeichert werden, so müssen sie in globalen Variablen gespeichert werden. In RUSSEL gibt es daher keine Befehle zum Einfügen oder Löschen von Fakten. Damit kann eine Regel in RUSSEL die Faktenbasis nur ändern, indem sie Werte an globale Variablen zuweist.

P-BEST. In P-BEST stehen alle Fakten allen Regeln zur Verfügung. Wenn eine Regel auf einen Fakt zugreift, so bekommt sie ihn per Referenz zugewiesen. Alle Änderungen, die eine Regel an einem Fakt vornimmt, sind für alle Regeln sichtbar. Jeder Fakt, auf den eine Regel zugreift, wird für die Dauer der Regelanwendung für alle anderen Regeln gesperrt. Haben zwei oder mehr Regeln das Interesse an einem Fakt, so entscheiden die Prioritäten der Regeln darüber, welche Regel zuerst auf diesen Fakt zugreifen darf. Haben mehrere Regeln die gleiche Priorität, so entscheidet der Zufall, wer auf diesen Fakt zugreifen darf. Dies kann zu Inkonsistenzen führen. Daher muss bei der Erstellung der Regelbasis darauf geachtet werden, dass diese "race conditions" vermieden werden. Dies kann durch die Einführung von Präzedenzen in Form von Prioritäten geschehen, indem darauf geachtet wird, dass keine zwei Regeln, die auf die gleichen Fakten zugreifen, die gleiche Priorität haben. Regeln können Fakten aus der Faktenbasis löschen oder Fakten in die Faktenbasis einfügen. Fakten sind in P-BEST komplexe zusammengesetzte Datenstrukturen.

5.4.1.3 Die Anweisungen

RUSSEL. In RUSSEL gibt es Anweisungen zur Manipulation von Variablen, bedingte Anweisungen, zusammengesetzte Anweisungen und Schleifenanweisungen. Dies sind Anweisungen, wie sie auch von "normalen" imperativen Programmiersprachen her bekannt sind. Außerdem vefügt RUSSEL noch über Anweisungen, die die Regelanwendung auslösen. Von diesen "rule-triggering"-Anweisungen gibt es drei Varianten:

1. `trigger off for_current`: Dies löst die Regelanwendung auf den gleichen Audit-Datensatz aus, den die aktuelle Regel auch gerade bearbeitet. Die durch diesen Befehl angestoßene Regel erhält dabei eine eigene Kopie der Daten.

2. `trigger off for_next`: Dies löst die Regelanwendung auf den nächsten zu lesenden Audit-Datensatz aus.

3. `trigger off at_completion`: Die Regelanwendung findet erst statt, wenn der Audit-Datenstrom zu Ende ist.

Diese Anweisungen führen zu einer Regelhierarchie, die der Prozesshierarchie von UNIX ähnelt. Am Anfang steht dabei die Regel, die die Anwendung aller anderen auslöst. Die Auslösung der Regelanwendung ist deterministisch.

P-BEST. In P-BEST hat der Programmierer der Regelbasis keinen direkten Einfluss darauf, wann eine Regel angewendet wird. Die Regelanwendung steuert allein die Inferenzmaschine. Sie untersucht bei jeder Änderung der Faktenbasis oder Regelbasis, welche Regeln bei dem augenblicklichen Bestand an Fakten zur

Anwendung kommen können. Dabei werden die Regeln nach ihren Prioritäten geordnet. Haben mehrere Regeln die gleiche Priorität, so ist über die Reihenfolge der Regelanwendungen nichts bekannt.

Regeln können auch die Regelbasis, durch die Befehle zur Aktivierung und Deaktivierung von Regeln, verändern. Damit steuern die Regelanwendungen auch direkt den Zustand des Expertensystemes.

5.4.1.4 Regeln

RUSSEL. In RUSSEL sind Regeln Prozeduren, die auch parametrisiert sein können. Sie haben keinen fest geordneten Aufbau.

P-BEST. Eine Regel besteht in P-BEST aus zwei Teilen:

1. Der Voraussetzung. Hier wird geprüft, ob die Schlussfolgerung gezogen werden darf.
2. Der Schlussfolgerung. Hier werden die Änderungen der Fakten- und der Regelbasis durchgeführt.

Diese beiden Teile werden streng voneinander getrennt, durch das Zeichen `==>`. Dadurch ähneln die Regeln den Produktionen einer Grammatik.

5.4.1.5 Konzepte

RUSSEL. Die Sprache RUSSEL wird interpretiert. Daher muss ein RUSSEL-Programm nicht erst übersetzt werden, wenn es ausgeführt werden soll. Dies macht die Regelbasis sehr flexibel, da der Austausch einzelner Module nicht zur Folge hat, dass das gesamte System neu compiliert werden muss.

P-BEST. P-BEST-Programme müssen erst einmal von dem "P-BEST-TO-C-COMPILER" (pbcc) in die Programmiersprache C übersetzt werden. Danach muss dann durch den Aufruf eines C-Compilers aus den C-Routinen ein lauffähiges Programm erstellt werden. Der Vorteil dieser Methode ist, dass das resultierende System recht schnell ist. Der Nachteil, der durch die Notwendigkeit des in der Regel recht komplexen Kompiliervorganges entsteht, wird teilweise dadurch wieder gutgemacht, dass durch den Aufruf des Programmes pbcc nicht nur C-Routinen erstellt werden, sondern es wird auch ein "Makefile" erstellt. Diese Datei dient dann mittels des Programmes "make" der Steuerung des Übersetzungsvorganges.

5.4.2 Die Gemeinsamkeiten

5.4.2.1 Die Mächtigkeit beider Sprachen

Was müssen Regelsprachen eigentlich können, oder wie groß ist die Mächtigkeit dieser Sprachen? Die beiden Sprachen P-BEST und RUSSEL wurden als allgemeine Sprachen entworfen. Damit sind sie in ihrer Anwendung nicht auf die Einbruchsentdeckung beschränkt. Tatsächlich sind beide Sprachen genauso mächtig, wie eine Turingmaschine. Eine Turingmaschine ist ein mathematisches Konstrukt, das äquivalent zu dem intuitiven Begriff des Algorithmus' ist. D. h., alle Algorithmen lassen sich von Turingmaschinen nachvollziehen. Damit können sie theoretisch zur Lösung beliebiger berechenbarer Probleme benutzt werden.

Bevor nun begründet wird, warum die beiden Sprachen turingmächtig oder turingvollständig sind, werden ohne Beweis einige Sätze aus [Sch97] zitiert.

Satz 1 *Jedes GOTO-Programm ist durch ein WHILE-Programm mit nur einer Schleife simulierbar. Jedes WHILE-Programm ist durch ein GOTO-Programm simulierbar.*

Damit sind Programmiersprachen, die das Konstrukt einer WHILE-Schleife besitzen, genauso mächtig wie Programmiersprachen, die statt dieses Konstruktes nur die GOTO-Anweisung und die IF-THEN-Anweisung kennen.

Satz 2 *Jedes WHILE-Programm kann durch eine Turingmaschine simuliert werden.*

Damit kann auch jedes GOTO-Programm durch eine Turingmaschine simuliert werden.

Satz 3 *Jede Turingmaschine kann durch ein GOTO-Programm simuliert werden.*

RUSSEL. Dass RUSSEL die gleiche Mächtigkeit wie eine Turingmaschine besitzt, sieht man sofort ein, da die RUSSEL-Anweisung `do...od` gleichberechtigt einer WHILE-Schleife ist. Russel erlaubt außerdem die Definition beliebig vieler Variablen vom Typ Integer. Wenn man die Wertebereichsgrenzen dieses Datentyps außer Acht lässt, dann ist die Sprache RUSSEL tatsächlich turingmächtig.

P-BEST. In P-BEST gibt es keine Schleifenanweisungen und keine unbedingte Sprunganweisung wie GOTO. WHILE-Schleifen lassen sich in P-BEST jedoch durch sich wiederholende Regelanwendungen simulieren. Die Inferenzmaschine von P-BEST ist so angelegt, dass sie eine Regel so lange anwendet, bis keine Fakten mehr da sind, die auf diese Regel passen. Eine Regel, die in ihrer Voraussetzung keinen Fakt referenziert, kann immer angewendet werden. Wenn nun in der Voraussetzung der Regel eine Klausel wie `[?|'x!=0]` und außerdem die Variable `x` existieren, wird die Regel solange angewendet, bis die Variable `x` den Wert 0 angenommen hat. Dies entspricht einer WHILE-Schleife.

5.4.2.2 Weitere Gemeinsamkeiten

- Beide Sprachen verfügen über die Möglichkeit, Variablen einfacher Datentypen zu deklarieren.
- Beide Sprachen verfügen über Anweisungen zur Manipulation von Variablen.
- Beide Sprachen können externe Prozeduren und Funtionen aufrufen.
- Beide Sprachen besitzen die Möglichkeit, das Betriebssystem, auf dem die Expertensysteme laufen, zu beeinflussen.

Damit sind die wichtigsten Gemeinsamkeiten der beiden Sprachen aufgezählt.

Kapitel 6

UNIX-NETZWERKSICHER-HEITSARCHITEKTUR

6.1 Überblick

In den vorangegangenen Abschnitten wurde die Funktionsweise von verschiedenen ID-Systemen beschrieben. In diesem Abschnitt wird zunächst ein intuitives Modell einer UNIX-Netzwerksicherheitsarchitektur, das von Robert B. Reinhardt in [Rei93] vorgeschlagen wurde, vorgestellt. Das Modell basiert auf dem Ansatz, die Anbindungssicherheit an das Internet oder an einem WAN mit Hilfe einer Firewall zu implementieren. Das Modell definiert verschiedene Schichten eines Firewalls, welche gleichzeitig die Verwundbarkeits-Schichten eines Netzwerkes (LAN) angibt. Dieses Modell wird in späteren Abschnitten um IDS-Komponenten erweitert.

6.1.1 Ein Sicherheitsschichtenmodell

Layer		Name	Beschreibung
LAYER	7	POLICY	Definition der Sicherheitsmaßnahmen
LAYER	6	PERSONAL	Die Menschen, die mit Ausrüstung und Daten umgehen
LAYER	5	LAN	Computerausrüstung und Daten
LAYER	4	Abgrenzung nach innen	Konzentratoren und interne Vernetzung
LAYER	3	GATEWAY	Funktionen nach OSI-Schichten 7,6,5,4
LAYER	2	Paket Filter	Funktionen nach OSI-Schichten 3,2,1
LAYER	1	Abgrenzung nach außen	Verbindung nach außen und öffentlicher Zugang

Tabelle 6.1: Sicherheitsarchitektur unter UNIX [Rei93]

Das Sicherheitsmodell aus [Rei93] hat, in Anlehnung an das OSI-Referenzmodell, sieben Schichten. Den einzenlnen Schichten sind auf diese Weise Abstratkionsniveaus zugeordnet, die entsprechend den Nummern der Schichten geordnet sind. Die einzelnen Schichten können noch feiner in Unterschichten unterteilt werden. Obwohl eine Firewall bei der Netzwerküberwachung nur auf den Schichten zwei bis fünf angesiedelt ist, stellen die beiden obersten Schichten, sechs und sieben, notwendige Voraussetzungen dar, um die Sicherheit auf den unteren Ebenen erfolgreich durchsetzen zu können. Die Sicherheitspolitik lässt sich zwar nicht für alle Teile des Unternehmens genau festlegen, weil ein Unternehmen einem ständigen Wandel unterworfen ist, was sich auch in der Konfiguration der Rechnernetzwerke des Unternehmens niederschlägt. Man kann aber grobe Richtlinien festlegen, deren Durchsetzung in allen Ebenen gewährleistet werden sollte.
Die Schichten fünf bis zwei des Modells beschreiben oder definieren die Werte, deren Schutz in der Sicherheitspolitik (Schicht 7) festgelegt wurde.
Der Zweck dieses Modelles ist es, die Beziehung zwischen den höheren und niedrigeren Schichten, aus denen ein komplettes Netzsicherheitssystem besteht, zu illustrieren. Der Schichtweise Aufbau des Modells dient folgenden Zwecken :
a) mit Hilfe einer Firewall Methode eine Zugriffskontrolle zu implementieren und
b) übergreifende Wirkungen der verschiedenen Schichten auf ihre jeweiligen Nachbarschichten darzustellen. Das Folgende ist daher eine allgemeine Beschreibung der Schichten und ihrer Beziehungen untereinander.

Im Folgenden werden zunächst die Schichten, so wie sie in [Rei93] beschrieben worden sind, vorgestellt. Anschliessend werden die Schichten fünf bis zwei gesondert noch einmal aufgegriffen, weil sie für das Einbringen von ID-Systemen in dieses Modell von besonderer Bedeutung sind.

Schicht 7-POLICY: Diese Schicht definiert die gesamte Menge der Maßnahmen, die getroffen werden müssen, um das System sicher zu machen. Hier wird die Sicherheitspolitik (*policy*) festgelegt, inklusive der Risiko- Nutzen-Abschätzung und es werden diejenigen Maßnahmen getroffen, um die Sicherheitspolitik in allen Ebenen durchzusetzen. Ohne eine vollständige Sicherheitspolitik ist das Sicherheitsprogramm nicht komplett. Die Sicherheitspolitik ist anwenderspezifisch und ist daher abhängig von der Intention des Einsatzes eines Systems.

Schicht 6-PERSONAL: Diese Schicht bezieht sich auf diejenigen Personen, die das System installieren, betreiben und in Stand halten, oder auf eine andere Weise Zugang zu dem System haben. Dies bezieht auch Menschen mit ein, die nicht zur Organisation gehören (z.B. Beschäftigte von Unternehmen, die in diesem Unternehmen eine Dienstleistung vollbringen). Diese Personen lassen sich nicht auf die gleiche Weise kontrollieren, wie interne Mitarbeiter und stellen daher eine weitere, schwer abzuschätzende Gefahrenquelle dar.

Schicht 5-LAN: Diese Schicht definiert die Endgeräte und Daten, die durch das Sicherheitsprogramm geschützt werden sollen. Dies beinhaltet auch monitoring und Kontrollprozeduren, die meist dazu genutzt werden, um Teile der Sicherheitspolitik durchzusetzen. Von dieser Schicht an wird die Sicherheitspolitik automatisiert im LAN selbst durchgesetzt.

Schicht 4-INTERNE ABGRENZUNG: Diese Schicht befasst sich mit den Geräten, die die physikalische Verbindung des LAN zum WAN über die Firewall, die als Pufferzone dient, herstellen. Der minimale Anspruch sollte die Möglichkeit beinhalten, dass bei Bedarf einzelne Bereiche des LAN vom WAN getrennt werden können.

Schicht 3-GATEWAY: Hier ist die Plattform, die die Netzwerkschnittstelle zwischen der internen Abgrenzung in Schicht 4 und dem Paketfilter/Router, der die Anbindung an das WAN herstellt, beheimatet. Hier werden Firewall-Dienste zur Verfügung gestellt. Welche Dienste das im Einzelnen sind, wird durch die policy in Schicht 1 definiert. Der Gateway-Rechner, der dieser Schicht zugeordnet wird, darf ausschließlich so genutzt werden, wie die policy es vorschreibt. Zu den Firewall-Diensten werden auch Monitorfunktionen gezählt.

Schicht 2-FILTER: Hier ist der Paketfilter angesiedelt. Er regelt die Kommunikation zwischen LAN und WAN.

Schicht 1-ABGRENZUNG NACH AUSSEN: Auf dieser Ebene werden alle Verbindungen betrachtet, auf die man keinen direkten Einfluss hat oder die nicht unserer Kontrolle unterliegen, wie z.B. das Telefonnetz oder andere mediale Verbindungen.

6.1.2 Ergänzung des Modells um IDS-Komponenten

In dem oben vorgestellten Modell spielt eine Firewall die zentrale Rolle bei der Durchsetzung der Sicherheit. Jedoch sind Firewalls nur passive Komponenten, die anhand von Filterungsregeln versuchen, eine IP-basierte Zugriffskontrolle zu etablieren. Das Aufstellen von optimalen Filterungsregeln, d. h., Regeln die nur den gutartigen IP-Verkehr zulassen und jeden maliziösen fernhalten, ist nicht möglich. Das Modell soll im Folgenden um aktive Komponenten, nämlich Intrusion Detection Systeme, erweitert werden, denn diese Systeme sind in der Lage, mit den Ergebnissen der Filterregeln weiter zu arbeiten und Schlüsse in Hinblick auf Sicherheit oder Funktionsfähigkeit des Systems zu ziehen. Der Aufbau des Modells eignet sich besonders gut für die Einbindung von ID-Systemen, weil ein Intrusion Detection-System, insbesondere EMERALD aus Abschnitt 3.6.6, ebenfalls einen schichtweisen Aufbau einer Netzwerküberwachungsinfrastruktur ermöglicht, die von der Schicht zwei dieses Sicherheitsmodells bis zur Schicht fünf reicht. Zuvor müssen jedoch die Paketströme in Audit-Trails abgebildet werden. Diese Audit-Trails werden auch Ereignisströme genannt. Im Folgenden einige Aspekte zur Erstellung von Ereignisströmen aus Paketströmen:

- **Discarded traffic:** Pakete, die aufgrund der Filterungsregeln abgelehnt wurden.
- **Pass-through traffic:** Pakete, die durchgelassen wurden.

Dies lässt sich noch feiner unterteilen:

- **Protocol specific traffic:** Pakete, deren Kopfteil ein ganz bestimmtes Protokoll angibt.
- **Unassigned port traffic:** Pakete, die an einen Port gerichtet sind, für den kein Dienst durch die Administratoren eingerichtet wurde.
- **Transport management traffic:** Pakete, die der Verwaltung von TCP-Verbindungen dienen (SYN,ACK, [window_resize]).
- **Source-address monitoring:** Die Überwachung von Paketen, die von ganz bestimmten Adressen stammen.
- **Destination-address monitoring:** Pakete, die an bestimmte Zieladressen gerichtet sind
- **Application-layer monitoring:** Die Überwachung von Paketen, die an bestimmte Dienste gerichtet sind.

Diese Ereignisströme lassen sich nun mittels verschiedener Methoden (z.B. Anomaly und Misuse Detection), auf Angriffe oder Fehler hin, analysieren. Damit das Modell um IDS-Komponenten erweitert werden kann, muss vorher festgestellt werden wo sie platziert werden können. Die Schichten fünf bis zwei müssen für diesen Zweck nochmal aufgegriffen und näher spezifiziert werden.

6.1.2.1 IDS in Schicht 5

Schicht fünf des Modells beschreibt die Umgebung, die dem größten potentiellen Risiko ausgesetzt ist. Die ersten Schichten erörtern Wege, um den Zugriff auf diese Schicht des Netzwerkes zu schützen. In Schicht fünf werden als zu schützende Werte die einzelnen Bestandteile eines LANs betrachtet (Workstations, File Server, Datenbanken und andere Netzwerkressourcen), einschließlich der Prozeduren, die diesen Schutz gewährleisten sollen. Es darf nicht davon ausgegangen werden, dass der Schutz dieser Ressourcen schon in Schicht eins bis vier sichergestellt wurde. Die Benutzung der einzelnen Computer und andere Ressourcen des lokalen Netzwerkes muss auch in dieser Schicht überwacht werden, z.B. durch einige Routine-Kontrollen, um die Verletzung der Firewall-Funktionen vorzubeugen.

Als eine zusätzliche Schutzmaßnahme können die Kontrollprozeduren eines ID-Systems in dieser Schicht untergebracht werden. Das ID-System kann zum Schutz der Programme und Daten auf den einzelnen Rechner bzw. Rechnergruppen zum Einsatz kommen. Das ID-Systeme kann auch hinter dem Firewall platziert werden und aktiv auf das Geschehen einwirken.

Zu überwachende Subjekte bzw. Objekte : Benutzer oder Benutzergruppen.

Methoden : Mit Schwerpunkt Anomaly Detection; Misuse Detection.

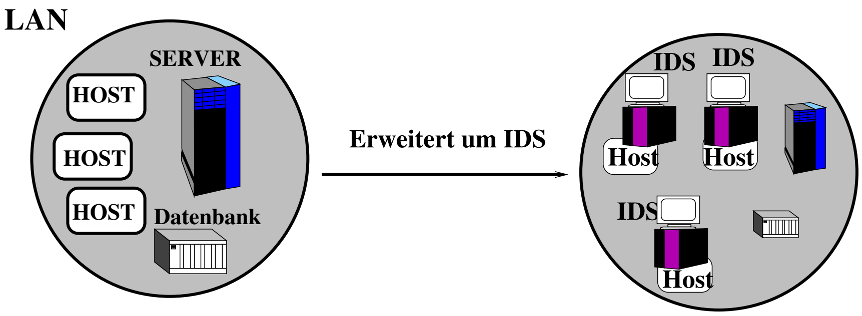

Abbildung 6.1: ID-System in Schicht 5

6.1.2.2 IDS in Schicht 4

In Schicht vier werden die Geräte eines Netzwerkes betrachtet, über die eine Anbindung der internen Rechner oder Rechnergruppen an das Internet, bzw. an ein WAN stattfindet (z.B. die Verbindung zwischen Rechner/Teilnetz des LANs und einem LAN-Gateway oder die Anbindung über die Firewall). In dieser Schicht implementiert der Router die Sicherheitspolitik für die drei unteren drei OSI-Schichten und die Aufgabe des Gateways besteht darin die Sicherheitspolitik für die höheren Schichten zu implementieren. ID-Systeme können vor dem LAN-Gatewayrechner platziert werden. Das IDS soll auf dieser Betrachtungsebene die Kommunikation der Rechner innerhalb des LANs und die Kommunikation zwischen Rechner und Gateway überwachen. Das ID-System überwacht auch die von aussen nach innen gerichteten (und umgekehrt) Netzwerkverkehr (hier befindet IDS allerdings innerhalb des LANs).

Zu überwachende Subjekte bzw. Objekte : Geräte eines LANs die für die Internet-Anbindung zuständig sind; Host, Host-Gruppen oder sonstige Rechner, die eine Verbindung zu dem Gatewayrechner besitzen.

Methoden : Anomaly Detection; Misuse Detection.

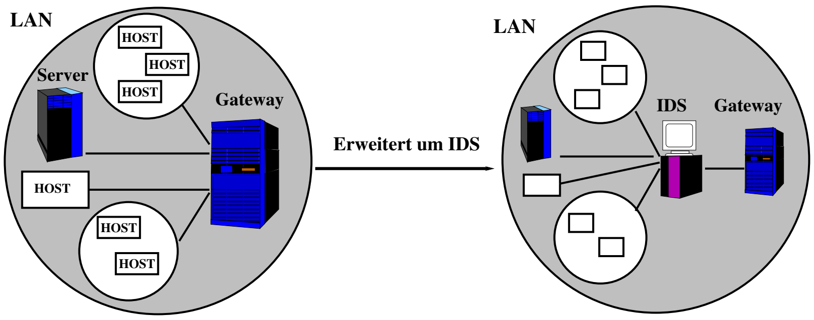

Abbildung 6.2: ID-System in Schicht 4

6.1.2.3 IDS in Schicht 3

Diese Schicht beschreibt die Netzwerkschnittstelle zwischen dem Gateway und dem Router. Zu schützen ist in dieser Schicht der Gatewayrechner, der die Verfügbarkeit des internen und externen Netzes sicherstellt, indem er zwischen verschiedenen Hostsystemen und dem WAN vermittelt. Die Firewall sitzt auf dem Gateway und Kontrolliert die Zugriffe auf diesen. Ein ID-System kann die Firewall sinvoll ergänzen, wenn es auf das Firewall aufsetzt und somit eine dritte (nach Router und Firewall) Kontroll-Instanz bei der Überwachung des Gateways darstellt.

Zu überwachende Subjekte bzw. Objekte : Gatewayrechner (oder Gatewayrechner, die z.B. als eine Firewall implementiert sind); Netzwerkschnittstelle zwischen Gatewayrechner und Router.

Methoden : Anomaly Detection; Misuse Detection.

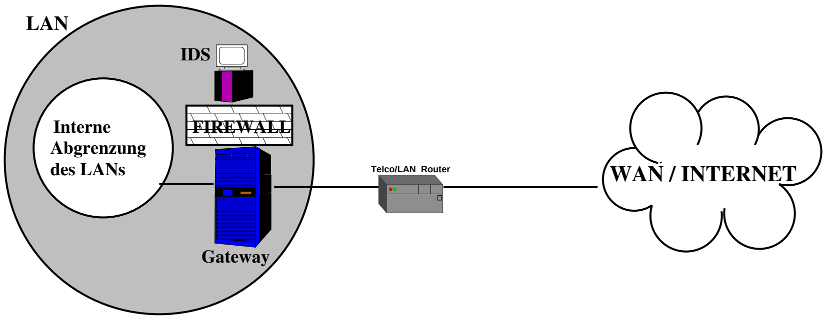

Abbildung 6.3: ID-System in Schicht 3

6.1.2.4 IDS in Schicht 2

Hier wird die physikalische Anbindung eines LANs an ein WAN betrachtet, auch die Betrachtung des Datenstroms erfolgt in dieser Schicht. Der Router dekodiert die elektrischen Signale der physikalischen Verbindung und wandelt sie in

Pakete von zusammengefassten Daten für die verschiedenen Netzwerkprotokolle um. Diese Pakete enthalten die Quelladresse, Zieladresse, Protokoll-ID und das Datagram selbst. Viele Router verfügen über die Fähigkeit Zugriffskontrolllisten (ACLs) zu verwalten, um die Kontrolle der herausgehenden oder hereinkommenden Daten zu ermöglichen. Dadurch können Paketinhalte herausgefiltert werden (z.B. basierend auf Quelladresse, Zieladresse, Protokoll, oder spezifische Port-Nummer). Auf diese Weise kann eine Netzwerkzugriffskontroll-Politik etworfen werden die am Router durchgesetzt werden kann, bevor Zugriff auf die internen Netzwerkressourcen gewährt wird. Der Router würde auf diese Weise die Firewall-Funktionen unterstützen. Der Einsatz eines ID-System in dieser Schicht kann als eine effiziente Ergänzung zum Firewall gesehen werden. Das IDS kann in diesem Zusammenhang zwei Funktionen ausführen:

1 Das ID-System kann überprüfen, ob die Filterungsregel des Firewall ausreichend sind.

2 Das ID-System kann die abgelehnten und die durchgelassenen Pakete überwachen oder analysieren.

Zu überwachende Subjekte bzw. Objekte : Physikalische Verbindung zwischen LAN und WAN; Router ; Firewall.

Methoden : Anomaly Detection; Misuse Detection.

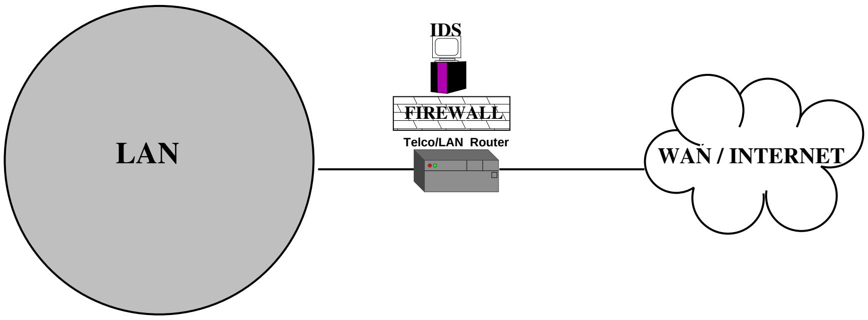

Abbildung 6.4: ID-System in Schicht 2

Anhang A

Zuordnung der Autoren zu den einzelnen Abschnitten

KAPITEL 1 .... Sibel Mutlu
Abschnitt 2.1- 2.4.11 .... Emine Yüksel
Abschnitt 2.4.11.1 .... Axel Schnell
Abschnitt 2.4.11.2- 2.4.11.5 .... Emine Yüksel
Abschnitt 3.1- 3.3.3.1 .... Sibel Mutlu
Abschnitt 3.3.3.2 .... Sibel Mutlu, Axel Schnell
Abschnitt 3.3.4- 3.6.4 .... Sibel Mutlu
Abschnitt 3.6.5- 3.6.8 .... Axel Schnell
KAPITEL 4 .... Sibel Mutlu
KAPITEL 5 .... Axel Schnell
KAPITEL 6 .... Sibel Mutlu, Emine Yüksel, Axel Schnell

Index

*-Property, 28
0-Flag, 83

AT&T, 58
DNS (Domain-Name-Service), 111
Sicherheitsanforderungen, 12, 24
Sicherheitskern, 29

ACK-Flag, 93
ACLs, 16, 18, 70
Aktive Angriffe, 98
Aktiver Port Scans, 100
Angriffsarten, 98
Angriffsplanung, 100
Angriffsschaden, 97
Applet, 123
ARP, 104
ARP-Spoofing, 104
ARPANET, 75
ASAX, 211
ASAX-Instantiierung, 220
AT&T, 58, 75
Auditdatenbeschreibungsdateien, 214
Authentication Server, 35
Authentifikation, 21, 23
Authentifikationsarchitektur, 33
Autorisierung, 22
Autorisierungsarchitektur, 33

B1-Systeme, 41
B2-Systeme, 42
B3-Systeme, 43
Backwardchaining, 258
Bell-LaPadula-Modell, 16, 26
Beweissicherung, 22
Boot-Block, 64
BSD, 58

C, 215
C1-Systeme, 40
C2-Systeme, 40
Capabilities, 16–18, 70, 74
Capability-Liste, 74
CC, 39
CGI, 124
Checksum, 92
Common Criteria (CC), 54
Cookis, 123
Cracker, 95

DAC, 15, 28
Dateiattribute, 65
Datenblocke, 65
DF-Flag, 83
Diskrete Politik, 24
Diskrete Zugriffskontrolle, 16
DNS Spoofing, 109, 112
DNS-Angriffe, 111
Dorothy Denning, 33
ds-Property, 28

Effektive Benutzernummer, 69
EMERALD, 200
EMERALD-Monitor, 203
Ereignisstrome, 290
Evaluation, 254
EVG, 54
Expertensysteme, 250

Faktenbasis, 250
FAU-Sicherheitsprotokollierung, 55
Filterungsregel, 293
Final-Flag (FIN), 92

Finger, 103
Finger Flooding, 117
Firewall, 289, 290, 292
Firewall-Dienste, 289
Flooding, 110
FORK, 67
Fragmentierung, 85
FTP, 113
Funktionalitatsklasse, 48

GATEWAY, 287
Gateway, 289, 291, 292
Gatewayrechner, 292
gefarbte Petrinetze, 264
getty, 69
GID, 67, 68
Green Book, 47
Gruppen-Identifikation, 71

Hacker, 95
Half Open Scan, 101
Hornformeln, 254
Hornklauseln, 254
HOST-ID, 84
Host-to-Host-Protokoll, 75
HTTP, 114

I&A, 30
ICMP, 106
ICMP Redirect, 106
ICMP Tunnel, 107
ICMP-Protokoll, 87
Identd, 103
Identifikation, 21, 23
IDIOT, 198
IHL (Internet Header Length), 81
Inferenz, 251
Inferenzmechanismus, 268
init, 69
Inode-Liste, 64
Insider, 96
Integritat, 12, 13, 23, 26
Intrusion Detection Model, 33
IP Spoofing, 107
IP-Adressen, 84
IP-Header, 80
IP-Internet Protocol, 80
IP-Paketformat, 81
IPv4, 81
IPv6, 81
ISO/OSI-Referenzmodell, 76
IT-Evaluationshandbuch, 47
IT-Sicherheitskriterien, 47
ITSEC, 52

Java, 121
Java Sandkasten, 123
Javaplattform, 122

Kerberos, 33
Kerberos Authentication Server, 34
Kerberos Datenbank, 34, 36
Kerberos Datenbank System, 37
Kerberos Datenbank-Server, 36
Kerberos Ticket Grantig Server (TGS), 35
Kerberos Ticket Granting Server (TGS), 34
Kerberos Ticket Granting Server TGS, 34
Kerberos-Architektur, 35
Kernel, 61
Klasse-A-Systeme, 44
Klasse-B-Systeme, 40
Klasse-C-Systeme, 40
Klasse-D-Systeme, 39
Konsistenz der Regelbasis, 255
Kriterienkatalog, 26

least privilege, 27
Leistung, 255
Levelfunktion, 27

MAC, 15, 28
MAC-Adresse, 104
Machtigkeit der Sprachen, 285
Mail Spoofing, 117
Mandatorische Politik, 24

Mandatorische Zugriffskontrolle, 16
MF-Flag, 83
MIME, 124
Modularitat, 274
Modus Ponens, 252
MULTICS, 57
Mustertyp, 267

NADF, 213
need to know, 27
Netstat, 103
Network Control Program (NCP), 75
Netz-ID, 84
Netzwerk-Header, 80
NFS-Network File System, 115
NIS, 113

Orange Book, 26, 39, 45
Outsider, 96

P-BEST, 266
P-BEST Sprache, 266
Passive Angriffe, 98
PDP-7, 58
Petrinetze, 259
PID, 67
ping of death, 106
Protokollierung, 23, 33
Pusch-Funktion (PSH), 92

Qualitatsstufen, 49

Reale Benutzernummer, 69
Realm, 34
Red Book, 39
Referenzmonitor, 29
Regelbasis, 250
Regeldefinition in RUSSEL, 276
Regeldefinitionen in P-BEST, 268
Regelnetze, 259
Reset (RST), 92
RIP Spoofing, 109
Router, 289
Routing, 86
RPC Stealth Scan, 102
Rstatd, 103
rusers, 103
RUSSEL, 274
RUSSELPROGRAMM, 279
Rwho, 103

Sendmail, 116
Session Hijacking, 109
setuid, 68
Shell, 69
Sicherheitsanforderungen, 14
Sicherheitsarchitekturen, 28
Sicherheitskern, 29
Sicherheitskriterien-Kataloge, 39
Sicherheitslevel, 27
Sicherheitsmechanismus, 14
Sicherheitsmodell, 24–26
Sicherheitspolitik, 24, 288
Simple Security Property, 27, 28
SMTP-Servern, 117
Source Routing, 109
Spezialdatei, 66
Spoofing, 107
Stealth Scanning, 101
Subjekt-Identifikation, 71
Super-Block, 64
SYN-Flooding, 105
Synchronization-Flag (SYN), 92
sysstat, 103
System V, 58
Systemschwachen, 96

TCB, 29–31, 42, 43, 46
TCP (Transmission Control Protocol, 90
TCP Port Scans, 100
TCP-Header, 78, 91
TCP/IP, 75, 76
TCP/IP-Analyse, 210
TCSEC, 39, 46
three-way-handshake, 100
Time Server, 37

TNI - Red Book, 45
Transmission Control Protocol/Internet Protocol, 76
Trojanische Pferde, 119
Trusted Computing Base, 29
Type Of Service (TOS), 82

Ubertragungs-Header, 80
UDP Port Scans, 100, 102
UDP-Protokoll, 81
UDP-User Datagram Protocol, 93
UID, 67, 68
UNIX, 57, 58, 61
UNIX Login, 69
UNIX Version 6, 58
UNIX-Dateisystem, 62, 64
UNIX-Netzwerksicherheitsarchitektur, 287
UNIX-Prozess, 66
UNIX-System, 60
UNIX-Verzeichnis, 63
Urgent-Flag (URG), 92
Urgent-Pointer, 92

Verfugbarkeit, 12, 14, 23
Vergleich von P-BEST und RUSSEL, 282
Vertraulichkeit, 12, 13, 26
Viren, 119
virtual interface, 217

whois, 103
Wurmer, 120
WWW, 120
WWW Spoofing, 125

X11, 114

Zugriffskontrolle, 15, 23, 69
Zugriffskontrollliste, 71, 73
Zugriffskontrolllisten, 16
Zugriffsmatrix, 16, 30

Literaturverzeichnis

[AB01] Anatol Badach, Erwin Hoffmann: *Technik der IP-Netze.* Carl Hanser Verlag, 2001.

[Bru] Brunnstein, Prof. Dr. Klaus: *Einführung in die IT-Sicherheit Teil 3.* Skript zur gleichnamigen Vorlesung.

[bsi89] *IT-Sicherheitskriterien*, Januar 1989. http://www.bsi.bund.de/sinet1.htm.

[bun91] *IT - Evaluationskriterien.* Bundesanzeiger, 1991.

[cc99a] *Common Criteria, Gemeinsame Kriterien für die Prüfung und Bewertung der Sicherheit von Informationstechnik*, 1999. CC Teil 1: Einführung und allgemeines Modell, Entwurfsstand 26.03.1999.

[cc99b] *Common Criteria, Gemeinsame Kriterien für die Prüfung und Bewertung der Sicherheit von Informationstechnik*, 1999. CC Teil 2, Anhang vom 15.11.1999, 2-te Überarbeitung.

[CFGP94] Castano, S., M.G. Fugini, G.Martella und P.Samarati: *Database Security*, Addison-Wesley Publishing Company, 1994.

[DA95] Debra Anderson, Thane Frivold, Alfonso Valdes: *Next-generation Intrusion Detection Expert System (NIDES).* Technischer Bericht, SRI International, Computer Science Laboratory, May 1995.

[dD82] Dudenredaktion, Wissenschaftlicher Rat der: *Das Fremwörterbuch.* Dudenverlag Mannheim/Wien/Zürich, 1982.

[Den87] Denning, Dorothy E.: *An Intrusion-Detection Model.* In: *IEEE TRANSACTIONS ON SOFTWARE ENGINEERING*, Band SE-13, No 2, Seiten 222–232, Februar 1987.

[DH00] Doraswamy, Naganand und Dan Harkins: *IP-Sec, der neue Sicherheitsstandard für das Internet, Intranets und virtuelle private Netze.* Addison-Wesley, Oktober 2000.

[Die00] DIERKS, JÖRN: *GRIDS: General Real-Time Intrusion Detection System.* Diplomarbeit, Universität Hamburg, 2000.

[Esc98] ESCAMILLA, TERRY: *Intrusion Detection: Network Security Beyond the Firewall.* John Wiley & Sons, Oktober 1998.

[FH92] FISCHER-HÜBNER, SIMONE: *IDA (Intrusion Detection and Avoidance System): Ein einbruchsentdeckendes und einbruchsvermeidendes System.* Doktorarbeit, Universität Hamburg, 1992.

[Fuh00] FUHRBERG, DR. KAI: *Internet-Sicherheit.* Carl Hanser Verlag, 2000.

[Gel99] GELLERT, OLAF: *Überblick Sicherheitsprobleme.* Technischer Bericht, Vortrag aus Seminar "Sicherheit in vernetzten Systemen", Fachbereich Informatik, Universität Hamburg, April 1999.

[Hin] HINDERER, K.: *Grundlagen der Stochastik für Informatiker und Ingeneure.*

[its91] *Information Technology Security Evaluation Criteria: Provisional Harmonised Criteria.* Catalogue number: CD-71-91-502-EN-C, 1991. Brüssel-Luxemburg.

[J.G95] J.GULBINS, K.OBERMAYR: *UNIX System V.4. Begriffe, Konzepte, Kommandos, Schnittstellen.* Springer Verlag, 1995.

[Kü96] KÜHNHAUSER, DR.WINFRIED: *Vorlesung: Sicherheit in Verteilten Systemen.* http://www.tu-chemnitz.de/informatik/osg/ lehre/vorl_SI.html, 1996.

[Kya98] KYAS, OTHMAR: *Sicherheit im Internet.* International Thomson Publishing Company, Zweite Auflage, 1998. meta-wissenschaftlich.

[LE99] LESSIG, ANDREAS G. und ANDREAS ENGEL: *Internetgestützte Angriffe und ausgewählte Gegenmaßnahmen.* Diplomarbeit, Universität Hamburg, Mai 1999.

[Lie00] LIENEMANN, GERHARD: *TCP/IP-Grundlagen.* Heinz-Heise Verlag, 2000.

[Lor88] LORENZ, FALKO: *Lineare Algebra.* BI Wissenschaftsverlag Mannheim/Wien/Zürich, 1988.

[Lou94] LOUDEN, KENNETH C.: *Programmiersprachen.* MITP-Verlag, 1994.

[LTG+92] LUNT, TERESA F., ANN TAMARU, FRED GILHAM, R. JAGANNATHAN, CAVEH JALALI, PERTER G. NEUMANN, HAROLD S. JAVITZ, ALFONSO VALDES und THOMAS D. GARVEY: *A Real-Time Intrusion-Detection Expert System (IDES).* Technischer Bericht, SRI International, February 1992.

[Lun93] LUNT, TERESA F.: *Detecting Intruders in Computer Systems.* In: *Conference on Auditing and Computer Technology*, Computer Science Laboratory, SRI International, Menlo Park, California 94025, 1993.

[Mou] MOUNJI, ABDELAZIZ: *ASAX (Advanced Sequential File Analysis Product).* Verfügbar unter ftp://ftp.info.fundp.ac.be/pub/projects/asax.

[Mou97] MOUNJI, ABDULAZIZ: *Languages and Tools for Rule-Based Ditributed Intrusion Detection.* Doktorarbeit, Facultés Universitaires Notre-Dame de la Paix Namur (Belgium), 1997.

[Mü00] MÜCK, HANS-JOACHIM: *Leitfaden zur Absicherung von Rechnersystemen in Netzen.* ftp://ftp.cert.dfn.de/pub/docs/leitfaden, 2000.

[ncs87] *Trusted Network Interpretation Criteria of the Trusted Computer System Evaluation Criteria*, Juli 1987.

[PH90] POHL, DR. HARTMUT und LUDGER HÜTTE: *Der Stand der Dinge bei der Zertifizierung in den USA.* Computerwoche, 12, März 1990. Bewertungstrend geht zur Klasse B1.meta-wissenschaftlich.

[PN] PORRAS, PHILLIP A. und PETER G. NEUMANN: *EMERALD: Event Monitoring Enabling Responses to Anomalous Live Disturbances.*

[PV98] PORRAS, PHILLIP A. und ALFONSO VALDES: *Live Traffic Analysis of TCP/IP Gateways.* Internet Society's Networks and Distributed Systems Security Symposium, March 1998.

[Rei93] REINHARDT, ROBERT B.: *An Architectural Overview of UNIX Network Security.* http://www.alw.nih.gov/Security/Docs/network-security.html, February 1993.

[Rö99] RÖHRIG, DR. BERNHARD: *Linux im Netz.* Computer & Literatur, 1999. **meta-wissenschaftlich**.

[Sch97] SCHÖNING, UWE: *Theoretische Informatik kurzgefaßt.* Spektrum Akademischer Verlag GmbH, 1997.

[Sch98] SCHILLER, JEFFREY I.: *Sicherheit im Daten-Nahverkehr.* In: *DOSSIER: Die Welt im Internet*, Kapitel Internet und Sicherheit, Seiten 52 – 57. Spektrum der Wissenschaft, Januar 1998. meta-wissenschaftlich.

[Sch00] Schöning, Uwe: *Logik für Informatiker*. Spektrum Akademischer Verlag GmbH, 2000.

[SF00] Stefan Fischer, Ulrich Walther: *Linux Netzwerke*. SuSE Press, 2000.

[Ste00] Stevens, W. Richard: *Programmieren von UNIX-Netzwerken*. Carl Hanser Verlag, Februar 2000.

[Tan95] Tanenbaum, Andrew S.: *Moderne Betriebssysteme*. Hanser Fachbuchverlag, 1995.

[TT91] T.D., Garvey und Lunt T.F.: *Model-based intrusion detection*. In: *In proc. 14th National Computer Secutity Conf.*, 1991.

[Val] Valk, Rüdiger: *Modelle für Rechensysteme*. Vorlesungsskript für das Wintersemester 1997/98.

[vHK98] Helden, Dr.Josef von und Dr.Stefan Karsch: *Grundlagen, Forderungen und Marktübersicht für Intrusion Detection Systeme (IDS) und Intrusion Response Systeme (IRS)*. http://www.bsi.bund.de/sinet1.htm, 1998. Version 1.4.

[YL99] Yves Lepage, Paul Larrera: *Die UNIX Sysad Bibel*. MITP-Verlag, 1999.