

Modulares System zur Erstellung von flexiblen
Intrusion–Detection und
–Countermeasure–Umgebungen

An der Fachhochschule Dortmund
im Fachbereich Informatik
erstellte Diplomarbeit
im Studiengang *Allgemeine Informatik [Studienrichtung Multimedia]*
zur Erlangung des Grades *Diplom–Informatiker (FH)* von

Thomas Biege, geboren am 19.05.1977

Erster Betreuer: Prof. Dr. Swik
Zweiter Betreuer: Prof. Dr. Eren

Dortmund, 17. Dezember 2002

Motivation

Der Antrieb für diese Diplomarbeit bestand zum Einen in der Herausforderung ein Intrusion-Detection-System (IDS) zu entwickeln, zum Anderen sollte die schlechte Lage der hostbasierten Intrusion-Detection-Systeme im Open-Source-Bereich verbessert werden.

Aber nicht nur im Open-Source-Bereich, sondern auch in der Forschung besteht Handlungsbedarf. Dort werden beispielsweise Analyseverfahren zur Erkennung von Sicherheitsverstößen entwickelt, die dann aber aufgrund von zu hoher Entwicklungsarbeit nie ihre Laborumgebung verlassen, um in praxistauglichen System eingesetzt zu werden.

Zusammenfassung

Im Laufe der Diplomarbeit wurden eine Reihe von Programmen entwickelt, von denen jedes eine spezielle Aufgabe erledigt, um das *Common Intrusion-Detection Framework* nachzubilden. Zu den Aufgaben gehört das Protokollieren von Systemaufrufen, das Weiterleiten von Daten, Datenarchivierung, Analyse der Daten, Management von Alarmen und zu guter Letzt die Reaktion auf erkannte Angriffe.

Die Programme kommunizieren hauptsächlich über ein TCP/IP-Netzwerk, oder über lokale Interprozesskommunikation.

Wichtige Funktionsteile der Programme wurden in Modulen ausgelagert, die ,ähnlich wie *Shared Libraries* oder *Dynamic Linked Libraries*, als binäre Einheiten zur Laufzeit geladen werden können.

Das beste Beispiel hierfür ist der sog. *BufferDaemon*. Seine Aufgabe besteht lediglich darin, Daten vom Netzwerk zu lesen, zwischenspeichern und nach einer gewissen Zeit zu verarbeiten. Diese drei Arbeitsschritte sind in nahezu jedem Server-Prozess zu finden. Die Abweichungen bestehen lediglich im Datenformat, sowie in der Datenverarbeitung. Die kontextbezogenen Aufgaben wurden in Modulen ausgelagert. Dies ermöglicht dem *BufferDaemon*, indem er die entsprechenden Module lädt, jedes beliebige Datenformat (Syslog, SNMP, IDMEF/IAP, usw.) vom Netz zu lesen und zu verarbeiten (Speicherung in einer Datenbank, Analyse, Verwaltung vom Alarmen, usw.).

Durch die systematische Aufteilung der verschiedenen Aufgaben eines Intrusion Detection Systems in eigene Programme und Module, die untereinander kommunizieren können, erhält man die Möglichkeit, wie in einem Baukastensystem, ein Intrusion-Detection-System so zusammenzusetzen, dass es unterschiedlichsten Anforderungen gerecht werden kann. Man kann alle Komponenten auf einem Rechner laufen lassen oder jedes Programm so verteilen, dass größere Netzwerke überwacht werden können. Es wäre zudem auch denkbar, dass mehrere Analyseverfahren in einer Hierarchie zusammengeschaltet werden oder die Aussagen von verschiedenen Analyseverfahren mit Hilfe einer Korrelationseinheit zu einem endgültigen Ergebnis ausgewertet werden. Die Modularisierung macht es Entwicklern und Forschern einfach, ihre neu entwickelten Verfahren in ein bestehendes System einzufügen.

Inhaltsverzeichnis

1	Einführung	1
1.1	Anforderungen an ein Intrusion-Detection-System	2
1.2	Intrusion-Detection-Systemkomponenten	3
1.3	Intrusion-Detection-Technologien	6
1.3.1	Audit-Datenquelle und Architekturen	6
	Hostbasierte Systeme	7
	Netzbasierte Systeme	7
	Alternativen	10
1.3.2	Analyseverfahren	12
	Mustererkennung	13
	Anomalieerkennung	16
2	Konzept	19
2.1	Audit-Datenquelle	19
2.2	Datenverarbeitung auf dem Client	21
2.3	Analyse	24
2.4	Datenbanken	24
2.5	Management-Host	24
2.6	Heartbeat	25
2.7	Graphical User Interface	25
2.8	Verschlüsselte Kommunikationsprotokolle	25
3	Implementierung	28
3.1	Intrusion Detection Message Exchange Format	28
3.1.1	IDMEF-Message-Aufbau	28
3.2	Design	31
3.2.1	Die Bausteine	34
	SCSLog — Das Kernel-Modul	34

	DataForwarder — Datenweiterleitung	41
	BufferDaemon — multifunktionaler Netzwerk-Server	45
	ReactionProcess — Reaktionen nach Maß	50
	ReactionDaemon — Reaktionen ausführen	51
3.2.2	Die Module	55
	mice_mod_filter	56
	mice_mod_logformat	57
	mice_mod_pseudonymizer	58
	mice_mod_mysql	58
	mice_mod_aa_regex	60
	mice_mod_enc_logformat_twofish	63
	mice_mod_enc_idmef_twofish	63
	mice_mod_act_generic	64
	mice_mod_rct_system	65
3.2.3	Die kryptografischen Protokolle	66
3.3	Beispielkonfiguration	70
3.3.1	Client	70
	SCSLog	71
	DataForwarder	72
	ReactionDaemon	74
3.3.2	Datenbank für rohe Log-Daten	75
	Tabellen erstellen	75
	BufferDaemon	76
	Tabellen auslesen	76
3.3.3	Analyseeinheit	79
	Analysemodul	80
3.3.4	Management-Host	84
	Act-Generic Modul	86
	Reaktionsprozesse	87
	Datenbank für Alarme	88
	Tabellen erstellen	88
	Tabellen auslesen	88
4	Testergebnisse	90
5	Schlussbetrachtung	94
A	Terminologie und Abkürzungen	96

B	Abbildungsverzeichnis	98
C	Literaturverzeichnis	99
D	Erklärung gemäß §26 (1) DPO	103

Kapitel 1

Einführung

Es gibt kaum noch Firmen, die nicht ans Internet angeschlossen sind; inzwischen sind sogar in Deutschland rund 31,8 Millionen Privatpersonen über das Internet zu erreichen. Neben klassischen, kabelbezogenen Verbindungen finden auch immer mehr Internetverbindungen über Satelliten oder Funknetzwerke in der Praxis Anwendung.

Neben der Anzahl der Internetanschlüsse ist auch die Zahl der Angriffe drastisch gestiegen. Wenn im Jahr 1998 dem COMPUTER EMERGENCY RESPONSE TEAM (CERT) [12] noch ca. 3.700 Anzeichen für Sicherheitsverstöße (Incidentsreports) gemeldet wurden, waren es im darauffolgenden Jahr mit 9.800 schon mehr als doppelt so viele. Die Häufigkeit der Angriffe steigt weiter an, bereits in den ersten beiden Quartalen des Jahres 2002 sind über 43.000 Incidentsreports beim CERT eingegangen. Diese Zahlen machen einen explosionsartigen Anstieg der Bedrohung im Internet sehr deutlich.

Zur Zeit gibt es vier generelle Ansätze, um der steigenden Anzahl von Angriffen Herr zu werden. Zum einen *Firewalls*, die als sehr einfaches Mittel für die Zugangskontrolle auf Netzwerk- (Packetfilter) und Applikationsebene (Proxy) eingesetzt werden. Beim Versandt von Daten über nicht vertrauenswürdige Netzwerke wird die Vertraulichkeit, Echtheit und Integrität mit Hilfe von sog. *Virtual Private Networks* (VPN) geschützt. Für die sichere Authentifikation von Benutzern wird bspw. *SecurID* von RSA SECURITY [38] eingesetzt. Dabei generiert ein sog. *SecurID* Token in bestimmten Zeitintervallen ein Einmal-Passwort, welches zur Anmeldung über ein unsicheres Medium benutzt werden kann. Wenn alle Abwehrmaßnahmen fehlgeschlagen sind, dann können Einbrüche immer noch durch *Intrusion-Detection-Systeme* (IDS) entdeckt werden.

Intrusion-Detection ist noch ein junges Gebiet der Sicherheitsindustrie; jedoch wurden die ersten Intrusion-Detection-Systeme bereits in den sechziger Jahren in

amerikanischen Telefonnetzen und in den Achtzigern vom US Militär zur Überwachung von Datenbanken eingesetzt. Man sieht also, dass die Idee schon lange existierte, als noch niemand an das Internet, in dem Ausmaß wie wir es heute kennen, gedacht hatte. Ende der neunziger Jahre erlebten Intrusion-Detection-Systeme ihre Renaissance und wurden für den kommerziellen Bereich wiederentdeckt.

Da die Erkennung von Angriffen in Computersystemen eine sehr komplexe Angelegenheit ist, die schon viele Richtungen und Ideen hervorgebracht hat, soll zunächst ein Überblick über bestehende Technologien und Ansätze gegeben werden. Dieser Überblick soll auch helfen, dass hier entwickelte System, M-ICE, besser einzuordnen und zu bewerten zu können.

1.1 Anforderungen an ein Intrusion-Detection-System

Im Jahre 1998 hat das BUNDESAMT FÜR DIE SICHERHEIT IN DER INFORMATIONSTECHNIK (BSI) von der DEBIS IT-SECURITY SERVICES GMBH eine Studie über Intrusion-Detection-Systeme [8] anfertigen lassen. Neben ID-Grundlagen und Produktübersichten wurden in Kapitel 4 und 5 Anforderungen an Intrusion-Detection und Intrusion-Response-Systeme definiert. Es soll hier nur ein Auszug der Forderungen wiedergegeben werden:

- Der Grund für einen Alarm muss vom IDS angezeigt werden können.
- Große Datenmengen müssen so darstellbar sein, dass eine manuelle Verarbeitung möglich ist.
- Angriffssignaturen sollten von dem Hersteller schnell bereitgestellt werden.
- Eigene Angriffsmuster müssen vom Benutzer manuell eingegeben werden können.
- Das System, auf dem die IDS-Software läuft, muss gut abgesichert sein.
- Das IDS selbst muss gegen Angriffe gehärtet worden sein.
- Das Produkt muss leicht skalierbar sein.
- Um effizienter zu arbeiten, sollte das IDS externe, sicherheitsrelevante Informationen aufnehmen können. Beispiel: Urlaubs-, Feier- und Krankheitstage
- Das Intrusion-Detection-System sollte hybrid sein, d.h., es sollte Misuse- und Anomaly Detection unterstützen.

- Daten aus mehreren Quellen (Netzwerk, Logfiles, etc.) sollten verarbeitet werden können.
- Die Daten sollten in „Echtzeit“ analysiert werden.
- Die Authentizität der gesammelten Audit-Daten muss gewährleistet werden können.
- Aus rechtlichen Gründen müssen Benutzerdaten pseudonymisiert werden.
- Die unter den einzelnen IDS-Komponenten ausgetauschten Daten müssen verschlüsselt werden.
- Die Integrität der gesammelten Audit-Daten muss erhalten bleiben. Dies kann durch Speicherung auf einmal beschreibbare Medien oder durch Kryptographie erzielt werden.
- Es muss möglich sein, die Granularität der Audit-Daten zu verändern. Beispielsweise kann während eines Angriffes die Granularität erhöht werden.
- Mehrere Angriffe müssen vom IDS erkannt und priorisiert werden.
- Beim Ausfall von Komponenten des Intrusion-Detection-Systems muss sofort Alarm gegeben werden.

1.2 Intrusion-Detection-Systemkomponenten

Für unsere Betrachtung soll die IDS-Spezifikation des *Common Intrusion-Detection Frameworks* (CIDF) [13] verwendet werden. Nach dem CIDF besteht ein IDS aus vier Komponenten, sog. Boxen.

- Event (E-) Box
- Analysis (A-) Box
- Countermeasure (C-) Box
- Storage/Data (D-) Box

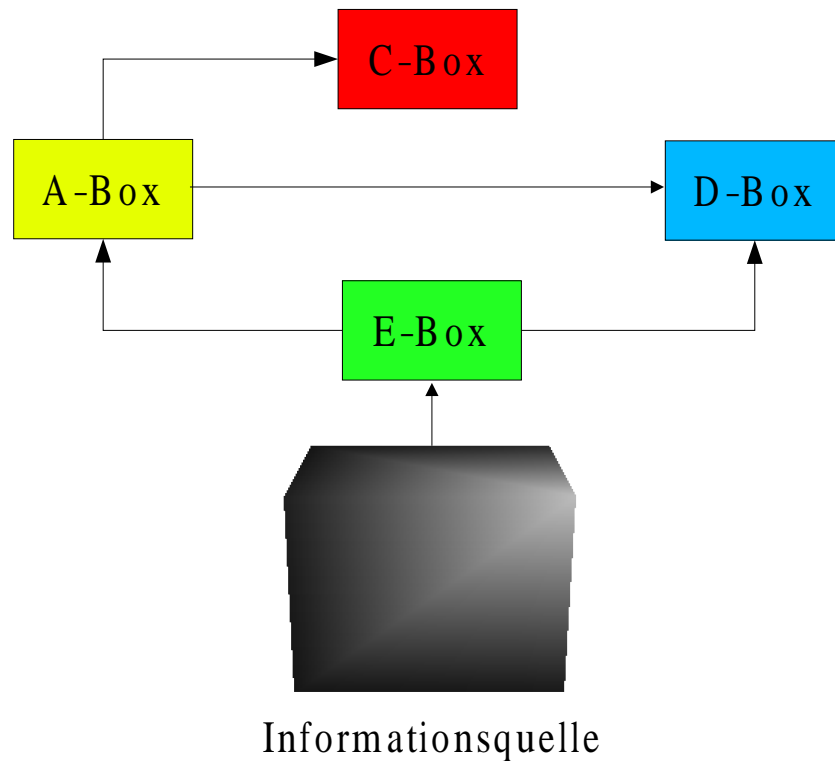


Abbildung 1.1: CIDF Modell mit E-, D-, A- und C-Box

Die Event-Box ist dafür verantwortlich, dass Audit-Daten, in Form von rohen Netzwerkpaketen und/oder Betriebssystem-Log-Daten aus der Kernel- und Applikationsebene nahtlos gesammelt werden. Anschließend werden die Daten in ein einheitliches Format (vgl. [1] [14] [15]) gebracht, an die Analysis-Box versandt und je nach Konfiguration auch an die Data-Box übertragen. Zuvor sollten jedoch redundante und unwichtige Informationen entfernt werden, um Speicherplatz zu sparen und die Informationsverarbeitung zu beschleunigen. In der Analysis-Box findet die Hauptarbeit des IDS statt. Hier werden die Daten von der Event-Box auf Angriffsmuster, Anomalien und nicht eingehaltene Sicherheitsvorgaben überprüft. Existiert für das gewonnene Analyseresultat eine vom Benutzer definierte Aktion, wird diese durch die Countermeasure-Box ausgeführt. Desweiteren wird das Analyseergebnis an die Data-Box weitergeleitet. Folgende Analyseverfahren werden i.d.R. benutzt:

- Misuse Detection:

- Experten System
- Zustandsautomaten (State Transition)
- Beschreibungssprachen
- Anomaly Detection:
 - quantitative Analyse
 - statistische Messungen
 - regelbasierte Systeme
 - neuronale Netzwerke

Die von der Countermeasure-Box durchgeführten Reaktionen können zwar in den verschiedensten Formen auftreten; unterliegen aber grundsätzlich zwei Kategorien:

- Passiv:
 - Email verschicken
 - Nachricht an einen Pager oder Handy senden
 - Audiodaten abspielen
 - Dialogfenster öffnen
 - etc.
- Aktiv:
 - Menge der aufzuzeichnenden Audit-Daten erhöhen
 - kritische oder angegriffene Systeme und Dienste abschalten
 - Packetfilterregeln von Firewalls anpassen
 - Informationen über das System sammeln, von dem die Angriffe ausgehen
 - das attackierende Computersystem mit Denial-of-Service (DoS) Attacken ausser Gefecht setzen
 - etc.

Die aktiven Reaktionen werden i.d.R. gar nicht oder sehr selten angewandt, da der Angegriffene, das Opfer, selbst zum Täter von Straftaten werden kann. Erschwerend kommt noch hinzu, dass sich der Angreifer gar nicht auf dem System befinden

muss, von dem die Attacken ausgehen. Der Angreifer kann mit Hilfe von *IP-Address-Spoofing*¹ seinen wahren Standort verschleiern. Oder das System wird vom Angreifer nur als Ausgangspunkt seines Angriffs missbraucht. Desweiteren kann der Angreifer seine Attacken so aussehen lassen, als würden sie von den IP-Adressen eines Geschäftspartners kommen. Die C-Box würde daraufhin die Firewallkonfiguration verändern, um die Pakete aus dem Partnernetz zu blocken, das hätte zur Folge, dass eine Zusammenarbeit nicht mehr möglich wäre.

Leider können die passiven Handlungen der C-Box ebenfalls von dem Angreifer gegen das zu schützende Netz gewandt werden. Zum Beispiel können durch zuviele vom IDS verschickte Alarmmeldungen an Email oder Pager die Mail- oder Modemserver überlasten (Denial-of-Service Attacke) oder der Security Officer wird dazu gebracht, die wirklich wichtigen Meldungen zu übersehen oder einen Pager gleich ganz abzuschalten bzw. die Emails ungesehen zu löschen.

Um diesen Gefahren entgegenzuwirken kann das IDS bspw. so konfiguriert werden, dass bestimmte IP-Adressen nicht gesperrt werden, sondern nur der SO alarmiert oder mit ihm Rücksprache gehalten wird. Alarmmeldungen desselben Types in einer bestimmten Zeitspanne sollten nur einmal mit Angabe der Menge an den SO gemeldet werden.

1.3 Intrusion-Detection-Technologien

1.3.1 Audit-Datenquelle und Architekturen

So vielfältig, wie die Einsatzgebiete der Intrusion-Detection-Systeme sind, so vielfältig sind auch deren Architekturen. Hier sollen nur die verbreiteten Varianten angesprochen werden. Als erstes müssen einige Unklarheiten bei den Begriffen „hostbasiertes“ und „netzbasiertes“ IDS aus dem Weg geräumt werden. Der Begriff „hostbasiert“ mag suggerieren, dass ein IDS zur Überwachung der Sicherheit von Computersystemen verwendet wird; das ist aber nicht ganz korrekt. Als hostbasierte Intrusion-Detection-Systeme werden Produkte bezeichnet, die ihre Audit-Daten, also Kernel- und Applikations-Log-Daten, von einem Host bekommen. Hostbasierte Intrusion-Detection-Systeme sind in der Lage 1 bis n Rechner zu überwachen, sie sind also nicht auf einen Rechner beschränkt.

Netzbasierte Intrusion-Detection-Systeme analysieren Netzwerkdaten, es ist egal wie sie diese erhalten und wo sie sie sammeln. Dabei überwachen sie die Sicherheit von Computersystem in einem Netzwerk. Häufig wird der Fehler gemacht, nur solche

¹hierbei wird die Absenderadresse eines IP-Paketes gefälscht

Intrusion-Detection-Systeme als netzbasierend zu bezeichnen, die ihre Daten von einem Netzwerkinterface erhalten, welches im sog. *Promiscuous-Mode* arbeitet, um alle vorbeikommenden Netzpakete mitzulesen. Solche Systeme sind die klassischen, sensorbasierten Intrusion-Detection-Systeme, neuere Ansätze verwenden Agenten (s. AID [17]), die auf den zu überwachenden Computersystem installiert sind, um die Netzpakete aus den unterschiedlichen Schichten des TCP/IP-Stacks abzufangen und an eine zentrale Analyseeinheit zu schicken.

Hostbasierte Systeme

Die hostbasierten Intrusion-Detection-Systeme existieren am Längsten. Sie wurden vom amerikanischen Militär eingesetzt, um Datenbank-Hosts o.ä. zu überwachen. Im Normalfall werden User-Level Log-Daten (syslog bei Unix Systemen) und/oder Kernel-Level Log-Daten (z.B. BSM Auditing Subsystem von Solaris, AIX Audit Subsystem, NT Event Log, Linux Trace Toolkit, Syscalltracker, SCSLog o.ä.) auf Einbruchsmuster untersucht. Andere H-Intrusion-Detection-Systeme, wie z.B. *Port-Sentry* [27], lauschen auch an TCP-/UDP-Ports und alarmieren den SO, wenn eine Verbindung zu den entsprechenden Ports aufgebaut wird.

H-Intrusion-Detection-Systeme haben den Vorteil, dass sie weniger *False Positives* auslösen als netzbasierte Intrusion-Detection-Systeme, da sie genau „wissen“, wie das Computersystem reagiert. Zudem können sie auch Angriffe sehen, für die ein N-IDS blind ist, z.B. Angriffe über die Konsole, Modems, verschlüsselte Verbindungen oder bösartige Programme (wie *Trojanische Pferde* o.ä.).

Nachteilig macht sich bemerkbar, dass ein H-IDS auf jedem zu überwachenden Rechner installiert und gewartet werden muss. Denial-of-Service Angriffe, die Fehler im TCP/IP-Stack eines Systems ausnutzen, können von einem H-IDS nicht ausreichend oder sogar gar nicht erkannt werden, weil das System vorher abstürzt und ein hostbasiertes Intrusion-Detection-System i.d.R. auf einer höheren Ebene operiert als ein netzbasiertes IDS.

Netzbasierte Systeme

Sensorbasierte N-Intrusion-Detection-Systeme erfreuen sich seit einigen Jahren großer Beliebtheit, da diese Systeme sich gut zentral verwalten und an zentralen Punkten einsetzen lassen. Somit kann ein IDS ganze Computernetzwerke einfach mit Hilfe eines Rechners überwachen und es muss nicht an jeden einzelnen Host Hand angelegt werden.

Diese Eigenschaft macht sensorbasierte N-Intrusion-Detection-Systeme sehr attraktiv, da sie weniger administrativen und finanziellen Aufwand erfordern. Die

Event-Box eines sensorbasierten N-IDS liest alle Daten, die über einen Netzwerksegment transportiert werden, und reicht sie anschließend an die Analysebox weiter.

Anhand der Informationen in den *Packet Headers* (Flags, Attribute, etc.) können DoS-Attacken oder *Scans* (Abtasten des Netzwerks auf Rechner oder Abtasten von Rechnern auf angebotene Dienste) erkannt werden. Die Payload enthält die Daten, die von dem TCP/IP-Stack an die Applikationsebene weitergereicht werden. Diese Daten werden von der A-Box auf Angriffsmuster geprüft. Bei der Verwendung von N-Intrusion-Detection-Systemen treten massive Probleme auf, die teilweise in der Zukunft noch größere Auswirkungen haben werden.

Das offensichtlichste Problem, dem sensorbasierte N-Intrusion-Detection-Systeme gegenüberstehen, ist die Verschlüsselung der Daten. Nicht nur VPN-Lösungen wie *IPsec* o.ä., sondern auch chiffrierte Login-Verbindungen, wie bei *Secure Shell* (SSH), machen ein klassisches N-IDS blind. Die Daten können nicht dechiffriert und somit auch nicht analysiert werden. In solchen Fällen ist ein hostbasiertes System klar im Vorteil. Der Versuch, diesem Problem mit einer *Public Key Infrastructure* (PKI) oder ähnlichen Keymanagementverfahren begegnen zu wollen, ist nicht sinnvoll, da dadurch die Komplexität stark erhöht würde, und somit die Robustheit des IDS sinkt. Zudem würde sich die Analyse extrem verlangsamen, da kryptografische Algorithmen, vor allem asymmetrische, rechenintensiv sind. Weiterhin kommt erschwerend hinzu, dass nicht alle Protokolle und Dienste mit einer PKI zusammenarbeiten können.

Die wachsende Netzwerkbandbreite macht es einem sensorbasierten N-IDS schwer, alle Daten nahtlos zu lesen, ohne dass interne (Ring-) Puffer überlaufen und Daten verloren gehen. Der *Network Flight Recorder* bspw. schafft es, ein saturiertes 100MBit Netzwerk zu überwachen. Jedoch existieren inzwischen Technologien im Gigabitbereich, die auch für den NFR ein Problem darstellen könnten.

Ein weitere Hürde, die es zu bewältigen gilt, stellt die gewitchte Netzwerkinfrastruktur dar. Im herkömmlichen *Ethernet* teilen sich alle Rechner das Übertragungsmedium (physikalisches Segment). Das bedeutet, wenn zwei Rechner gleichzeitig Pakete übermitteln wollen, dann kollidieren die Pakete u. U. und müssen erneut versandt werden. Die Rechner befinden sich also in einer gemeinsamen *Collision Domain*. Ein Switch vermeidet Kollisionen, indem er jedem Rechner einen eigenen physikalischen Netzwerkstrang über Ports zuweist und die Pakete genau zu dem Rechner schickt, an den sie adressiert sind. Ein Switch erzeugt also kleinere physikalische Segmente, in denen sich nur zwei Rechner befinden, ähnlich einer Telefonvermittlung. Wenn aber die Netzpakete nur an den dafür vorgesehenen Rechner gelangen, dann kann die E-Box des N-IDS die Pakete nicht mehr mitlesen; das IDS ist somit wieder einmal blind.

Viele Switches haben einen sog. *Trunk Port*, der es dem N-IDS erlaubt, die Pakete zu lesen. Für ein 100MBit Netzwerk hat jeder Port, inkl. *Trunk Port*, des Switches eine Bandbreite von 100MBit. Wenn der Switch zehn Ports, ohne *Trunk Port*, hat, müsste der *Trunk Port* 10 x 100MBit Bandbreite haben, um alle Daten von allen Ports mit voller Geschwindigkeit transportieren zu können; er hat aber nur $\frac{1}{10}$ dieser Bandbreite. Bei schlechter Implementierung der Software auf dem Switch kann bei einem voll ausgelasteten Netzwerk ein N-IDS u.U. nicht alle Daten lesen. Der *Trunk Port* ist also eher eine Notlösung für den Intrusion-Detection-Bereich.

Um die Daten analysieren zu können, muss das sensorbasierte N-IDS alle gängigen Netzprotokolle wie IPv4, IPv6, T/TCP, IPX, Appletalk etc. verstehen. Wenn in dem zu schützenden Netz ein sehr exotischen Protokoll gesprochen wird, welches der IDS Hersteller nicht implementiert hat, dann ist das IDS somit wertlos.

Die Spezifikationen für den TCP/IP-Stack sind zwar in den *Request for Comments* (RFC-791 und RFC-793 [37]) Dokumenten vorgegeben, werden aber von jedem Betriebssystemhersteller anders oder unvollständig implementiert. Das hat zur Folge, dass das sensorbasierte N-IDS nie genau weiss, in welchem Zustand sich der TCP/IP-Stack eines Hosts befindet. Die Diskrepanz in den verschiedenen Stack-Implementierungen kann sich ein Angreifer zunutze machen, um seine Pakete gegenüber dem IDS zu „verstecken“, oder er kann Pakete im Datenstrom des IDS einfügen, die das Zielsystem nicht anerkennt und ignoriert [29]. Dieser Sachverhalt soll an einem einfachen Beispiel erklärt werden: Ein Angreifer schickt ein TCP-Paket, welches die Angriffsdaten enthält, an einen unsicheren Netzdienst des Zielsystems. Die Prüfsumme im TCP-Header ist absichtlich falsch berechnet. Das IDS sieht das Paket, ignoriert es aber, weil die Prüfsumme falsch ist. Das Zielsystem empfängt das Paket, überprüft die Prüfsumme aber nicht, weil es nicht implementiert wurde und reicht die Nutzdaten an die Serverapplikation weiter. Resultat: Das IDS hat das Paket verworfen, aber das Zielsystem akzeptierte es und der Angreifer konnte das Sicherheitsloch unbemerkt ausnutzen.

Diese gravierenden Probleme machen den Einsatz von rein sensorbasierten N-Intrusion-Detection-Systemen bereits heute fraglich.

Knotenbasierte (nodebased) N-Intrusion-Detection-Systeme vermeiden viele Probleme der klassischen N-Intrusion-Detection-Systeme (s. AID [17]), sind aber aufgrund ihres dezentralen Charakters nicht so einfach einzusetzen wie ihre Vorgänger. Auf den einzelnen Computersystemen werden kleine „Monitore“ eingesetzt, die in den TCP/IP-Stack eingebunden werden, um die Netzwerkpakete auf einer höheren Ebene zu lesen und sie dann zentral (oder auch dezentral) zu analysieren. Inkonsistenzen bei der Interpretation von Netzverbindungen, überfüllte Puffer und Verschlüsselungen auf niedrigerer Ebene (*IPsec* etc.) sind somit kein Problem mehr.

Verschlüsselung auf höherem Level z. B. durch SSH, PGP, SSL usw. können aber immer noch nicht im Klartext betrachtet werden, ausser natürlich mit komplizierten Keymanagementverfahren.

Alternativen

Neben den host- und netzbasierten Ansätzen gibt es noch andere Entwicklungen in der Architektur von Intrusion-Detection-Systemen, die zum großen Teil aber keinen größeren Einsatz in der Praxis fanden (z. B. mobile autonomous Agents). Die bekannteste Form ist wohl der *Honeypot*, der dem Angreifer ein verwundbares System vorgaukelt.

Honeypot Systeme Honeypots sind Systeme, die vorgeben, Sicherheitslücken zu haben. Wenn ein Angreifer dieses System entdeckt, dann wird er es attackieren und keine Probleme haben, die Sicherheitsmechanismen zu überwinden. Der Angreifer denkt, er befindet sich auf einem normalen Computersystem, wurde in Wahrheit aber in eine abgeschottete Umgebung umgeleitet. Der SO wird darüber benachrichtigt und hat nun die Möglichkeit, das Verhalten des Angreifers zu beobachten und zu analysieren. Somit ist es vielleicht möglich, das Bestreben des Angreifers zu erkennen oder sogar neue Angriffsformen zu entdecken. *Das Deception Toolkit* (DTK) und *ManTrap* sind gute Beispiele für ein hostbasiertes Honeypot System. Theoretisch sind auch netzbasierte Honeypots (Honeynet) möglich, die dem Angreifer ein ganzes Netz vorgaukeln, indem sie auf einem Rechner mehrere *Virtual Machines* (VM) mit virtuellen Netzverbindungen implementieren.

Honeypots sind sehr zuverlässig beim Erkennen von Attacken, können dem SO aber unnötig Arbeit machen, da jeder noch so unbegabte Angreifer in das System einbrechen kann. Diese Angreifer wollen häufig keine speziellen Informationen stehlen, sondern nur ihre Langeweile vertreiben oder *IRC Bots* aufsetzen. Solche Angriffe sind zeitraubend und liefern dem SO keine neuen Erkenntnisse. Da ein Honeypot nur eine Falle darstellt kann es natürlich passieren, dass der Angreifer nicht in sie hineintappt und er somit unentdeckt bleibt. Honeypots eignen sich besonders in einer DMZ oder vor einer Firewall, da der Angreifer dort nur wenig Angriffsfläche findet und so die Wahrscheinlichkeit höher ist, dass der Honeypot Ziel der Attacken wird.

Agentenbasierte Intrusion Detection Systeme (A-IDS) A-Intrusion-Detection-Systeme sind ein gutes Beispiel für *Distributed Computing*. Auf dem zu observierenden System laufen eigenständige, kleine Softwareprogramme (Agenten), die einfach nur Kommandos überwachen oder sogar nach Angriffen Ausschau halten. Die In-

formationen, die von den Agenten gesammelt wurden, werden an andere Einheiten des Intrusion-Detection-Systems weitergeleitet, um dort verarbeitet zu werden. Zwei sehr bekannte Beispiele für A-Intrusion-Detection-Systeme sind *Autonomous Agents for Intrusion-Detection* (AAFID) [23] und EMERALD [24]. Anhand von AAFID soll dieser IDS-Typ näher erläutert werden.

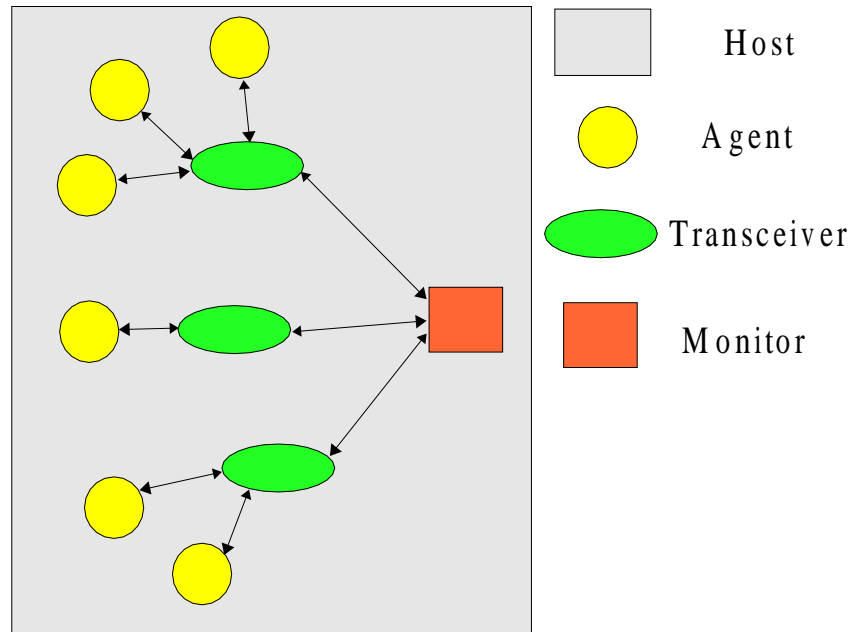


Abbildung 1.2: AAFID-Modell

AAFID ist hierarchisch strukturiert. An unterster Stelle stehen die Agenten. Es können mehrere Agenten, die jeweils unterschiedliche Aufgaben haben, pro Host eingesetzt werden. Alle Agenten senden ihre Informationen an einen Transceiver, der sich auf demselben Host befindet. Der Transceiver überwacht und steuert die Agenten, er kann sie starten, anhalten und rekonfigurieren. Zusätzlich ist der Transceiver noch für die Datenreduktion verantwortlich und übergibt seine Ergebnisse an einen oder mehrere Monitore, die die nächst höhere Instanz der Hierarchie darstellen. Die Monitore selbst können wieder hierarchisch angeordnet werden. Sie kontrollieren die Daten von den Transceivern und fügen sie zusammen. Die Fähigkeit der Monitore, die Daten von verschiedenen Hosts zu vereinigen, macht es möglich, dass Attacken,

die von mehreren Hosts ausgehen, erkannt werden können. Der Aufbau von AAFID erlaubt eine leichte Skalierung. Wenn dem Netz ein neuer Server hinzugefügt wird, müssen einfach nur die AAFID-Komponenten installiert werden, die dann ihre Daten an einen übergeordneten Monitor schicken.

1.3.2 Analyseverfahren

Bei den Analyseverfahren haben sich in den letzten Jahren drei Richtungen herauskristallisiert. Die älteste Methode, um Einbrüche festzustellen, ist *Misuse Detection*. Bei diesem Verfahren wird ein Mustervergleich (Pattern Matching) vorgenommen. Die Daten der Event-Box werden mit Angriffsmustern (Attack Signatures) aus einer Datenbank verglichen. Ist dieser Vergleich positiv, dann wurde eine Verletzung der Security Policy erkannt und es wird entsprechend reagiert. Im kommerziellen sowie im nichtkommerziellen Bereich ist *Misuse Detection* immer noch das am häufigsten benutzte Verfahren. Es ist einfach zu realisieren, anzuwenden und nicht sehr anfällig für falsche Alarmer (*False Positives*). Der große Nachteil dieses Verfahrens ist, dass es nur bekannte Angriffe erkennt, was zur Folge hat, dass neue Angriffe, die sich noch nicht in der Signature-Datenbank befinden, keinen Alarm auslösen (*False Negatives*) und somit unbemerkt bleiben.

Um dieses Defizit auszugleichen wurde ein neuer Weg eingeschlagen und die *Anomaly Detection* entwickelt. *Anomaly Detection* geht davon aus, dass alles was nicht zur Menge des „normalen“ Verhaltens gehört, also demnach „anomal“ ist, ein Angriff sein muss. Diese Methode hat gegenüber *Misuse Detection* den Vorteil, dass sie es ermöglicht, neue Angriffe zu erkennen, da sie ein abnormales Verhalten darstellen. Zudem muss keine Datenbank mit Angriffsmustern aktualisiert und gepflegt werden. Aber auch hier tauchen wieder Schwierigkeiten auf, die die Verbreitung von *Anomaly Detection* im kommerziellen Bereich stark behindern. Verfahren zur Anomalieerkennung müssen zuerst das „normale“ Verhalten eines Netzes oder Computersystems erlernen, indem sie Profile für die Benutzer und das System anlegen. Diese Phase an sich stellt schon eine Hürde dar und könnte auch von einem Angreifer ausgenutzt werden, um dem IDS Angriffe als normales Verhalten beizubringen. Das Intrusion-Detection-System könnte somit zukünftige Angriffe dieser Art nicht mehr erkennen. Ein weiteres Manko besteht in der hohen Rate an *False Positives*, die durch Störungen in der normalen Systemaktivität ausgelöst werden, aber keine Angriffe darstellen. Zusätzlich ist die Implementierung von *Anomaly Detection* gegenüber der *Misuse Detection* schwieriger, da die angewandten Verfahren komplexer sind.

Ein noch sehr junges Verfahren, das als *Burglar Alarm*, *Passive Traps*, oder *Strict Anomaly Detection* bezeichnet wird, benutzt einen relativ simplen, jedoch sehr effek-

tiven Ansatz. Es besagt, dass alles was nicht „richtig“ ist, „falsch“ sein muss. Diese Aussage erinnert an Anomaly Detection; es wird aber Mustererkennung wie bei *Misuse Detection* benutzt, d.h., das bekannte, normale Verhalten des Systems wird als Signature in eine Datenbank abgelegt und jede Systemaktivität, die nicht einem der Muster aus der Datenbank entspricht, ist anomales Verhalten und signalisiert einen Angriff. Es müssen nur relativ wenige Muster in der Datenbank abgespeichert werden. Diese Muster müssen nicht wie bei *Misuse Detection* für jeden neuen Angriff aktualisiert werden, sondern nur bei einer Veränderung des IT-Systems. Dadurch ergibt sich ein geringer Verwaltungsaufwand im laufenden Betrieb als bei normalen *Misuse Detection* Systemen.

Damit das ganze etwas klarer wird, ein kleines Beispiel: Der eMail-Proxy in einer DMZ darf nur TCP-Pakete auf Port 25 empfangen und absenden. Desweiteren dürfen diese Pakete nur in das Internet oder in das zu schützende Netz geschickt werden. Das ist das „richtige“ Verhalten, welches als ein Muster in einer Datenbank abgelegt wird. Das IDS überprüft nun den Paketfluss des eMail-Proxies mit dem gespeicherten Muster. Sollte der eMail-Proxy diesem Verhalten einmal nicht folgen, so besteht die Wahrscheinlichkeit eines Angriffs. Die Muster müssen sich nicht auf abstrakte Netzverbindungen beziehen, sondern können auch tiefere Ebenen wie Header-Informationen der einzelnen Netzwerkschichten oder Systemaufrufe eines Betriebssystems widerspiegeln.

Damit die Fehlerquote bei der Erkennung von Angriffen vergleichbar gering bleibt, müssen Anomalien, wie sie immer mal wieder in Netzwerken auftreten, als Ausnahme explizit angegeben werden oder in einen Toleranzbereich fallen. Somit können *False Positives* zwar auftreten, sind aber weitaus geringer als bei *Anomaly Detection*, *False Negatives* sind nahezu ausgeschlossen. Wenn Angriffe unter der Toleranzgrenze bleiben, dann werden sie natürlich nicht detektiert.

Nachfolgend sollen die verschiedenen Analyseverfahren für Misuse und *Anomaly Detection* kurz erläutert werden. Auf *Strict Anomaly Detection* und andere alternativen Analyseverfahren (Immunsystem, Genetische Algorithmen, Datamining, etc.) soll hier nicht näher eingegangen werden, da sie noch zu neu sind und bisher noch keine Verwendung in der Praxis finden.

Mustererkennung

Experten Systeme Für die Analyse der Daten wurden schon früh Fähigkeiten von Experten Systemen entdeckt. Folgende Intrusion-Detection-Systeme benutzten Experten Systeme:

- MIDAS

- IDES
- NIDES [16]
- DIDS
- AID (benutzt RTworks) [17]
- Emerald (benutzt eine Abwandlung von P-BEST) [24]
- CMDS (benutzt das freie System CLIPS [26])

Experten Systeme können mit *if-then-else*-Ausdrücken von dem Anwender leicht programmiert werden. Sie haben aber das Problem, dass sie große Mengen dieser Regeln nicht schnell genug abarbeiten können, da sie nur Interpreter, und damit bekanntlich langsam sind. Hinzu kommt noch, dass ein Experten System natürlich immer nur so gut sein kann, wie die Person, die es programmiert hat. Eine Veränderung des Regelwerks gestaltet sich schwierig, da es Auswirkungen auf die restlichen Regeln hat. Durch die reinen *if-then-else*-Bedingungen sind Experten Systeme nicht in der Lage Unsicherheiten bzgl. der Angriffsanalyse zu handhaben, sondern können nur „Ja“– oder „Nein“–Aussagen machen.

Zustandsautomaten (State Transition) Für *Misuse Detection* bieten optimierte Zustandsautomaten zur Mustererkennung eine flexible und leistungsfähige Möglichkeit Eindringlinge zu erkennen.

Das erste Intrusion–Detection–System, das diese Technik benutzte, war STAT [34] und später USTAT, NetSTAT und WinSTAT, die alle an der Universität von Californien, Santa Barbara, entwickelt wurden. Auf dem ACM Workshop im Jahr 2000 wurde eine flexible und kraftvolle Beschreibungssprache namens STATL für zustandsautomatenbasierte Intrusion–Detection–Systeme, ebenfalls von der Universität von Californien, Santa Barbara, vorgestellt.

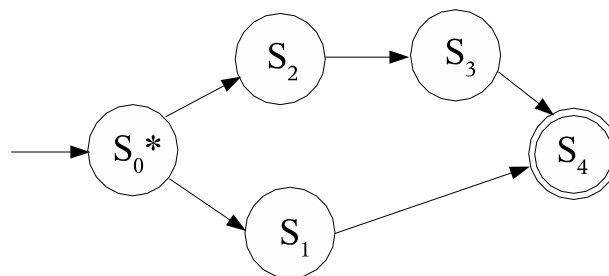


Abbildung 1.3: Zustandsautomat

Das obige Bild zeigt den Weg von einem Startzustand (Kreis S_0) über mehrere *Transitionen* und *States* zu einem Endzustand (Kreis S_4), der einen Einbruchversuch signalisiert. Ein Eindringen in das System besteht i.d.R. aus mehreren Aktionen. Jede Aktion wird durch eine *Transition* wiedergegeben. Die *States* spiegeln den Systemzustand wieder, der bspw. aus User-Privilegien, vorhandenen Dateien oder dem derzeitigen Zustand eines Netz- oder Systemdienstes bestehen kann. Das Intrusion-Detection-System verwaltet für jedes Angriffsszenario ein State-Transition-Diagramm. Wenn eine Systemaktivität stattfindet, überprüft die Analyseeinheit diese mit jedem State-Transition-Diagramm und überführt jeden Zustandsautomaten in den neuen State, wenn die Aktivität mit der Transition übereinstimmt. Dieses Verfahren macht es möglich, die Wahrscheinlichkeit und den Verlauf von Angriffen zu ermitteln und die Schritte des Angreifers vorherzusagen, indem die verschiedenen Zustandsautomaten bzgl. ihres aktuellen States untersucht werden. Wurde der Endzustand erreicht, d.h. ein Eindringen liegt vor, so wird ein entsprechendes Ereignis von der Intrusion-Response-Einheit (C-Box) ausgeführt. Die Zustandsdiagramme bieten ein abstraktes Abbild der Angriffsmuster, sie sind somit leicht verständlich und unabhängig vom Format der Audit-Daten und technischen Details. Zustandsautomaten stellen das Minimum an Aktivität für einen Angriff dar. Aus diesem Grund können Varianten einer Attacke schon mit wenigen, kleinen Zustandsautomaten abgedeckt werden. Auf State-Transition basierte Intrusion-Detection-Systeme ermöglichen es sogar koordinierte und stark verlangsamte Attacken zu erkennen. Eine verbesserte Variante der State-Transition-Analyse wurde mit dem sog. Colored Petri (CP)-Netz im IDIOT IDS implementiert.

Beschreibungs-/Interpretersprachen Damit sich die Benutzung von Analyseeinheiten in Intrusion-Detection-Systemen noch etwas einfacher gestaltet, wurden Sprachen zur Beschreibung von Angriffsszenarien entwickelt. Bekannte Beispiele sind:

- RUSSEL
- STALKER
- N-Code
- Snort's Beschreibungssprache
- STATL

Anomalieerkennung

Quantitative Analyse Es gibt verschiedene Arten der quantitativen Analyse, die geläufigste ist die Schwellwerterkennung (*Threshold Detection* oder auch *Threshold and Triggers* genannt). Bei der Schwellwerterkennung werden System- und Benutzeraktivitäten mit Hilfe von Zählern dargestellt. Der Wert der Zähler ist bis zu einem gewissen Schwellwert als normales Verhalten definiert, bei der Überschreitung dieses Schwellenwertes, wird ein möglicher Angriff gemeldet. Als Beispiel kann man sich vorstellen, dass ein Benutzer in den letzten zwei Stunden drei Mal erfolglos versucht hat sich an ein System anzumelden. Der Wert 3 ist in diesem Fall der Schwellwert und das IDS reagiert, indem es z.B. den Benutzeraccount sperrt. Als Erweiterung der Schwellwertanalyse wurde die *heuristische Schwellwert Analyse* entwickelt. Bei der *heuristischen Schwellwert Analyse* sind die Grenzwerte nicht starr sondern werden dynamisch angepasst. In unserem Beispielfall würde der Grenzwert nicht statisch auf 3 gesetzt werden, sondern wäre anhand des Einlogverhaltens des Users in der Vergangenheit berechnet worden. Der Grenzwert könnte bspw. aus der alten Anzahl von fehlerhaften Logins, zuzüglich einer Standardabweichung, errechnet werden. Als weiteres quantitatives Analyseverfahren kann man noch die Integritätsprüfung (*Target-Based Integrity Check*) nennen. Bei dieser Methode wird die Veränderung von Systemobjekten (Dateien, Programme, Hardware etc.), die sich eigentlich nicht verändern dürfen, überwacht. Das bekannteste Beispiel ist *Tripwire*. *Tripwire* überwacht Dateien und Programme, indem es kryptographische Hashwerte (MD5 Algorithmus) von ihnen generiert, sie in einer Datenbank ablegt und diese MD5-Hashwerte in regelmäßigen Zeitabständen mit den Hashwerten der z.Zt. im System befindlichen Dateien bzw. Programme vergleicht. Stimmen die Hashwerte nicht überein, so wurde die entsprechende Datei oder das Programm verändert, z.B. indem sie von dem Angreifer durch ein trojanisches Pferd ersetzt oder der Inhalt der Passwort-Datenbank verfälscht wurde. Die quantitativen Methoden werden aber nicht nur zur Erkennung von Angriffen benutzt, sondern auch zur Datenreduzierung. Die Eigenschaft, Systemattribute und Benutzerverhalten in Zahlen auszudrücken, macht sich das NADIR IDS zunutze, um die redundanten und nutzlosen Informationen in den Audit-Daten zu reduzieren. Die NADIR-Entwickler stellten Benutzeraktivitäten mit Hilfe von Vektoren dar. Diese *Profile-Vektoren* können für die Experten Systeme und für die statistische Analyse benutzt werden.

Statistische Messungen IDES und NIDES benutzen statistische Verfahren und waren die ersten funktionierenden *Anomaly Detection* Systeme (eigentlich sind es hybride Intrusion-Detection-Systeme, da sie sowohl *Misuse Detection* als auch *Anoma-*

ly *Detection* benutzen). Die Systeme verwalten eine Menge von statistischen Profilen, die das Normalverhalten der Benutzer und der Systemkomponenten beinhalten. Die Profile werden in periodischen Zeitabständen aktualisiert. Ein Sicherheitsverstoß wird gemeldet, sobald eine starke Abweichung vom Normal-Profil auftritt.

Ein weiteres IDS, das auf statistische Messungen basiert, ist Haystack. Haystack benutzt zwei Mengen von statistischen Profilen. Eine Menge dient zum Vergleich des Benutzerverhaltens gegenüber Angriffsverhalten, die zweite Menge wird benutzt um eine Abweichung vom Normalverhalten festzustellen. Mit statistischen Verfahren können Angreifer erkannt werden, die sich als andere Benutzer maskieren, indem sie deren Account benutzen. Da sich der Angreifer anders verhält, als der Benutzer, dessen Account missbraucht wird, wird das IDS darauf aufmerksam und alarmiert den SO. Zudem können auch völlig neue Angriffstaktiken und -methoden erkannt werden, die ein System mit quantitativer Analyse nicht erkennen würde. Der große Nachteil vieler statistischer Verfahren ist die Unfähigkeit im Echtzeitbetrieb zu arbeiten, was eine sofortige Reaktion auf einen Angriff unmöglich macht. Zur Zeit der Entwicklung dieser Analysemethoden war der *Batchmode-Betrieb* völlig ausreichend, da sie für die Überwachung von *Mainframes* konzipiert waren und es noch keine so ausgeprägte Vernetzung gab. Der Versuch, statistische Analysen für Realtime-Intrusion-Detection-Systeme zu verwenden, scheiterte an der hohen Performanz, die dafür nötig war. Hinzu kommt, dass statistische Messungen nur eine Aktivität und nicht eine Abfolge von Aktivitäten analysieren können. Diese Eigenschaft schränkt die Verwendung solcher Systeme im Hinblick auf die Menge der zu detektierenden Attacken stark ein, da die meisten Angriffe aus einer Abfolge von Handlungen bestehen.

Die o.g. Verfahren basieren alle auf Annahmen bzgl. des normalen und nicht-normalen Verhaltens eines Systems. Der Grenzwert, den das IDS benutzt, darf nicht zu niedrig sein, um falsche Alarmierungen (*False Positives*) zu vermeiden, was nur dazu führt, dass das IDS nicht mehr beachtet wird. Andererseits darf der Grenzwert nicht zu hoch liegen, so dass Angriffe nicht erkannt werden (*False Negatives*).

Die folgenden Ansätze benötigen keine Parameter für den Grenzwert und versuchen, die Nachteile der parameterbehafteten Analyseverfahren auszugleichen.

Regelbasierte Systeme Diese Systeme funktionieren im Grunde genau wie statistische Systeme, mit dem einzigen Unterschied, dass Regeln anstatt Statistiken benutzt werden. *Wisdom and Sense* (W&S) ist bspw. ein regelbasiertes IDS. Die Regeln können bei W&S entweder durch den Benutzer eingegeben werden (um die *Security Policy* darzustellen) oder es kann die Regel aus alten Audit-Daten generieren. Das *Time-Based Inductive Machine* (TIM) IDS benutzt auch Regeln, geht aber einen ganz anderen Weg als andere *Anomaly Detection* Systeme. Es fahndet nicht

nach individuellen Ereignissen, sondern überwacht deren zeitliche Abfolge. Eine Anomalie liegt vor, wenn die zeitliche Abfolge von Systemereignissen nicht eingehalten wurde.

Neuronale Netzwerke Neuronale Netzwerke können anomales Verhalten mit Hilfe von adaptiven Lerntechniken erkennen und benötigen deshalb keine Parameter, die durch den Benutzer vorgegeben werden müssen. Das neuronale Netzwerk muss erst mit sauberen, d.h. nicht mit Angriffsaktivitäten verunreinigten, Daten trainiert werden. Aufgrund dieser Lerneigenschaft stellen neuronale Netzwerke einen großen Wert für die Anomalieerkennung dar. Leider lassen sich durch Benutzung von Neuronalen Netzwerken nicht die Ursachen einer Anomalie herausfinden. Dem SO kann nur das Vorhandensein eines Sicherheitsverstoßes nicht aber dessen Grund gemeldet werden. Um dieses Problem zu umgehen, werden neuronale Netzwerke entwickelt, die sich nur auf einen Angriffstyp beschränken. Das bedeutet, wenn das neuronale Netzwerk für *SYN-Flooding* feuert, dann weiss man, um welchen Angriff es sich handelt. Diese Lösung ähnelt dann aber wieder der Mustererkennung.

Kapitel 2

Konzept

Um das Ziel zu erreichen unterschiedlichsten Umgebungen und Ansprüchen gerecht werden zu können, muss ein äußerst modulares System geschaffen werden. Dabei sollen die einzelnen Komponenten, wie sie im *Common Intrusion-Detection-Framework* beschrieben werden, als eigenständige Programme implementiert werden. Daten zwischen den Komponenten sollen über ein TCP/IP Netzwerk oder lokal über FIFOs ausgetauscht werden.

Eine solche Architektur reicht aber noch nicht aus, um die gewünschte Modularität zu erfüllen. Aus diesem Grunde sollen wichtige Funktionen aus den einzelnen IDS-Komponenten entnommen und in dynamisch ladbare Module ausgelagert werden. Die Module sind einfach auszutauschen, und man umgeht das lästige und fehleranfällige Anpassen des Quellcodes. Somit ist es nicht nur möglich, sich einer gegebenen IT-Infrastruktur besser anpassen zu können, bspw. durch ein Datenbank-Modul für eine bereits vorhandene ORACLE Datenbank, sondern es ist auch möglich, eigene Verfahren in ein bestehendes System zu integrieren, um es zu verbessern oder einfach nur, um seine eigenen Entwicklungen zu testen.

Das IDS soll unter SuSE Linux entwickelt und implementiert werden. Bei der Entwicklung soll aber auf Portabilität geachtet werden, um spätere Portierungen zu vereinfachen.

2.1 Audit-Datenquelle

Jedes IDS lebt von den Daten, die es für seine Analyse benötigt. Hostbasierte Intrusion-Detection-Systeme analysieren i.d.R. Log-Daten von Applikationen und dem Kernel.

In Applikations-Logs lassen sich anhand von Fehlermeldungen mögliche Angriffe erkennen. Aber nicht nur direkte Attacken können erkannt werden, sondern auch

simple Verstöße gegen die Sicherheitspolitik. Beispielsweise könnte das erfolgreiche Anmelden eines Administrators mitten in der Nacht, außerhalb der normalen Arbeitszeiten, ein Hinweis auf den Missbrauch des Administrator-Accounts sein.

Leider sind Applikations-Logs oft nicht aussagekräftig genug, deshalb lassen sich Angriffe auf Kernel-Ebene besser erkennen. Mit Hilfe von Informationen über Systemaufrufe, wie der Rückgabewert, die Identität des Aufrufers, der Aufrufsequenz, die Parameter oder einfach aufgrund des Vorhandenseins eines bestimmten Systemaufrufes lassen sich konkrete Aussagen über das Verhalten von Benutzern und Applikationen machen.

Da Linux von Haus aus keine Möglichkeit bietet Systemaufrufe¹ im *User Space* zu verfolgen, soll ein eigenes *Loadable Kernel Module* (LKM) entwickelt werden, das diese Aufgabe erfüllt.

Um die Log-Daten aus dem Kernel vor Manipulation zu schützen, soll nicht der Umweg über *Syslog* oder andere Systemapplikationen gegangen werden. Eine eigene Gerätedatei (Schnittstelle im Dateisystem zwischen *User Space* und *Kernel Space*) soll im Linux-Dateisystem angelegt werden, die es Benutzerprogrammen erlaubt, die Daten mit `read(2)` direkt aus dem *Kernel Space* zu lesen. Diese Gerätedatei soll nicht nur dazu dienen Daten aus dem LKM zu lesen, sondern sie soll auch als Schnittstelle zur Steuerung des LKMs mit Hilfe von `ioctl(2)` genutzt werden. Es soll möglich sein, das Logging für bestimmte Systemaufrufe an- oder abzuschalten und den Status diverser, modulinterner Parameter (Syscall-Status, Debug-Level, Ringpuffergröße) in Erfahrung zu bringen.

Unser IDS soll also Textdateien in denen Log-Daten enthalten sind, überwachen können und zudem eine Gerätedatei auslesen, um Informationen aus dem Kernel zu bekommen.

Da u.U. sehr viele Daten von dem LKM generiert werden könnten, muss man sich folgende Frage stellen: „Wohin mit all den Audit-Daten, wenn niemand sie aus dem *Kernel Space* holt?“ Es gibt für ein solches Problem nur zwei Lösungen: Entweder man vergrößert dynamisch den Puffer, in dem man die Daten zwischenspeichert, oder man hat einen festen Speicherbereich und überschreibt alte Einträge.

Der erste Ansatz hat den Nachteil, dass der ganze Kernel-Speicher aufgebraucht werden könnte, was zur Folge hätte, dass das System stehen bliebe. Der zweite Ansatz löscht Daten, die vielleicht wichtig für die Analyse sind.

Für unsere Implementierung soll der letzte Ansatz gewählt werden. Die Daten werden in einem Ringpuffer gespeichert, so dass immer die ältesten mit den neusten Einträgen überschrieben werden.

¹inzwischen gibt es ab SuSE-Kernel 2.4.19 das *Linux Trace Toolkit*, um das System auf Kernel-Ebene zu überwachen

Desweiteren muss das LKM selber vor Manipulation geschützt werden. Es soll die Option geschaffen werden, das LKM vor den Augen des Angreifers zu verstecken und es fest im System zu verankern, so dass es nicht mehr entfernt werden kann. Um die Verarbeitung der Informationen zu erleichtern, soll das LKM ein einfach zu analysierendes Log-Format mit festen *Tokens* und *Delimitern* benutzen.

2.2 Datenverarbeitung auf dem Client

Die Log-Zeilen können Daten enthalten, die vom Benutzer beeinflusst wurden, aus diesem Grund müssen beim Design der Client-Komponente von M-ICE entsprechende Sicherheitsvorkehrungen getroffen werden. Die Client-Komponente soll nur da, wo es wirklich nötig ist, mit Super-User-Rechten arbeiten, zudem sollen, wenn immer möglich, logisch zusammengehörige Codeteile in eigene Prozesse aufgespalten werden.

Die auf dem Client generierten Daten müssen auf sicherem Wege an andere IDS-Komponenten für die Weiterverarbeitung verteilt werden. Dazu sind folgende Aufgaben nötig:

- Sammlung
- Filterung
- Pseudonymisierung
- Formatierung
- Verschlüsselung
- Weiterleitung

Die Filterung, Pseudonymisierung und Formatierung der Log-Daten soll jeweils in ein eigenes Modul ausgelagert werden, da diese wichtigen Aufgaben durch verschiedenste Methoden realisiert werden können.

Sammlung Das Sammeln von Audit-Daten soll darin bestehen, immer die aktuellste Textzeile aus einer Log-Datei zu lesen. Dabei kann die Datei eine normale Datei, eine Gerätedatei oder ein FIFO sein. Somit können also *Syslog*-Dateien, Applikations-Log-Dateien und bspw. die Gerätedatei des LKMs als Datenquelle dienen.

Filterung Die Filterung soll als erstes in der langen Kette der Datenverarbeitung greifen, um die Menge der zu analysierenden und zu speichernden Daten zu reduzieren.

Eine effiziente Datenreduktion, wie bspw. durch Abstraktion der Audit-Daten in Vektoren (s. NADIR IDS), hat den Nachteil, dass die originalen Audit-Daten nicht weiter in der Analyse, und, noch viel wichtiger, nicht mehr im Management-Teil des Intrusion-Detection-Systems auftauchen. Für M-ICE sollen reguläre Ausdrücke vom Anwender in das System eingegeben werden können, um Audit-Daten zeilenweise zu filtern. Diese Methode ist einfach zu implementieren, ausreichend effektiv und verfälscht zudem die Daten nicht. Als kleiner Nachteil hierfür sei zu bedenken, dass der Administrator sein System kennen muss, um entscheiden zu können, welche Log-Daten unwichtig sind und welche nicht.

Pseudonymisierung Die Pseudonymisierung von benutzerbezogenen Daten steckt sozusagen noch in den Kinderschuhen. Pseudonymisierung wird aus Datenschutzgründen eingesetzt und ist nicht zu verwechseln mit Anonymisierung. Bei der Anonymisierung wird bspw. die Identität einer Person so verändert, dass es nicht mehr möglich ist festzustellen, welche reale Person sich hinter einer anonymisierten Identität verbirgt. Diese Technik macht den Einsatz im Intrusion-Detection-Bereich nutzlos, da man natürlich wissen möchte, wer hinter einem Sicherheitsverstoß steht. Die Pseudonymisierung erlaubt es hingegen, die Privatsphäre von Computerbenutzern zu schützen und gleichzeitig diese Privatsphäre wieder aufzuheben, sobald ein Angriff entdeckt wurde.

Zwei in Deutschland entwickelte Verfahren sind recht bekannt und wurden größtenteils auch schon implementiert.

An der Universität Cottbus wurde im Laufe der Entwicklung des Intrusion-Detection Systems AID [17] von M. Sobirey et al. ein Verfahren zur Pseudonymisierung mit Hilfe von symmetrischer Kryptographie entwickelt. Dabei wird auf dem zu beobachteten Rechner mit einem geheimen Schlüssel die Benutzeridentitäten chiffriert. Wird ein Angriff erkannt, dann wird die verschlüsselte Benutzeridentität mit dem geheimen Schlüssel wieder in Klartext übersetzt.

Eine andere Methode wurde von U. Flegel an der Universität Dortmund entwickelt. Hier wurde ein anderer kryptographischer Ansatz benutzt, das sog. *Secret Sharing* (s. [11], S. 84 ff.). Beim *Secret Sharing* wird ein geheimer Schlüssel in n Teile aufgebrochen. Diese n Schlüsselteile benötigt man, um Ciphertext wieder in Klartext umwandeln zu können. Man definiert also einen Schwellwert und nur wenn diese Schwelle überschritten wird kann die Identität eines Benutzers wieder hergestellt werden. Zum besseren Verständnis ein kleines Beispiel: Die Sicherheitspolitik

erlaubt drei fehlgeschlagene Login-Versuche, bevor der Administrator benachrichtigt wird. Der Schwellwert liegt also bei drei und somit müssen auch drei *Shared Secrets* generiert werden. Wurden vom System drei Fehlermeldungen erzeugt, so ist es möglich, die Identität des Benutzers wieder herzustellen.

Für M-ICE soll kein Pseudonymisierungsverfahren implementiert werden, um den Entwicklungsaufwand gering zu halten. Es soll jedoch durch ein Modul in der Client-Anwendung von M-ICE die Möglichkeit geschaffen werden, ein solches Verfahren später problemlos einzuführen.

Formatierung Die nativen Unix-Log-Daten sollen mit weiteren Informationen angereichert und in ein einheitliches Format gebracht werden. Folgende zusätzliche Informationen können nützlich sein:

- Rechnername
- Domainname
- IP-Adresse
- Betriebssystem
- Betriebssystem-Release
- Betriebssystem-Version
- Datum
- Uhrzeit

Verschlüsselung Die Verschlüsselung auf dem Client soll mit Hilfe eines symmetrischen Verschlüsselungsverfahrens geschehen. Bei symmetrischer Kryptographie in verteilten Systemen besteht das Problem der Schlüsselverteilung. Im Fall von M-ICE reicht es aus, wenn der Administrator den geheimen Schlüssel in die entsprechende Konfigurationsdatei einträgt.

Weiterleitung Die verschlüsselten Daten müssen in zwei Richtungen weitergeleitet werden: zur Analyseeinheit und an eine Datenbank zur Speicherung. Da die Verdopplung des Datenstroms u.U. dazu führen kann, dass das Netzwerk überlastet wird, sollen entsprechende Optionen bereitgestellt werden, diesem bei Bedarf entgegen wirken zu können.

2.3 Analyse

Es ist nicht Ziel der Diplomarbeit, ein besonders ausgefeiltes Analyseverfahren zu implementieren oder sogar zu entwickeln. Dies würde den Umfang einer eigenen Diplomarbeit in Anspruch nehmen. Um das System aber testen zu können, soll eine einfache Analyse mit Hilfe von regulären Ausdrücken implementiert werden.

Der Quellcodeteil, der die Analyse durchführt, wird in ein Modul ausgelagert. Dieses Modul soll umfassend konfigurierbar sein, damit ausreichend Flexibilität geboten wird, die sogar eine Zusammenschaltung der Analyseverfahren erlaubt.

Analyseergebnisse, die an den Management-Host oder an eine andere Analyseeinheit weitergereicht werden, sollen im *Intrusion Detection Message Exchange Format* eingepackt und verschlüsselt werden.

2.4 Datenbanken

Das System soll über zwei Datenbanken verfügen. Eine Datenbank dient zur Speicherung von rohen *Syslog*-Daten und analysierten Kernel-Daten, um eine genauere, manuelle Analyse bei verdächtigen Handlungen durchführen zu können. Die zweite Datenbank speichert lediglich die gemeldeten Alarme und die entsprechende IDMEF-Nachricht.

Zur Bewahrung der Datenintegrität sollen die Log-Daten an der Quelle kryptographisch signiert werden. So gespeicherte Daten lassen Manipulationen leicht erkennen und genießen zudem einen höheren Echtheitswert, um bspw. auch in juristischen Auseinandersetzungen genutzt werden zu können.

2.5 Management-Host

Der Management-Host soll Alarme von verschiedenen Analyseeinheit in unterschiedlichen Formaten entgegennehmen können und benutzerdefinierte Reaktion auslösen.

Damit der Management-Host Ereignisse von verschiedenen Sensoren verwalten kann, muss eine Tabelle für die Zuordnung zwischen Klassifikation und Klassifikationsbeschreibung und eine Tabelle zur Abbildung der Alarme auf vorhandene Reaktionen existieren.

Kleine und leicht austauschbare Programme sollen die Reaktionen des Systems übernehmen. Dabei soll die Möglichkeit bestehen, Reaktionen ohne Probleme hinzuzufügen oder zu entfernen.

Zusätzlich sollen Schnittstellen geschaffen werden, um andere IDS Sensoren, bspw. Snort [19], mit dem Management-Host verwalten zu können. Diese Eigenschaft würde den Aufbau eines hybriden Intrusion-Detection-Systems erlauben.

2.6 Heartbeat

Die Komponenten des Intrusion-Detection-Systems sollen an dem Management-Host ihren Prozessstatus melden, damit festgestellt werden kann, ob ein Prozess eingefroren oder sogar beendet wurde. Dieses kann absichtlich durch einen Angreifer geschehen sein oder auch unabsichtlich durch Fehler oder Überlastung des Client-Systems. Hierfür soll die IDMEF-Heartbeat-Klasse benutzt werden.

Der Management-Host muss eine Liste mit allen Komponenten verwalten, die je eine Heartbeat Nachricht gesendet haben oder noch senden müssen. Fällt eine Einheit aus, so bleibt auch die Heartbeat-Nachricht aus und der Management-Host kann den Security Officer entsprechend darüber informieren.

2.7 Graphical User Interface

Damit der Benutzer die erkannten Angriffe besser überschauen und weiterverarbeiten kann, soll die Benutzung eines *Graphical User Interfaces* (GUI) berücksichtigt werden; es soll jedoch keine GUI implementiert werden, da auch diese Aufgabe den Umfang dieser Diplomarbeit überschreiten würde. Einer möglichen GUI müssen also genügend Schnittstellen und Informationsquellen zur Verfügung gestellt werden, damit der Benutzer mit allen wichtigen Daten versorgt werden kann. Als Informationsquelle könnte bspw. eine SQL-Datenbank oder im XML-Format gespeicherte Daten dienen.

2.8 Verschlüsselte Kommunikationsprotokolle

Die Kommunikation über das Netzwerk soll verschlüsselt erfolgen, um die Daten und die Komponenten zu schützen. Oberster Leitsatz hierbei ist jedoch, alles einfach zu halten, um eventuelle logische Fehler im Design der Protokolle zu vermeiden.

Damit die Prozesse des Intrusion-Detection-Systems nicht unnötig ausgebremst oder aufgebläht werden, soll für die Verschlüsselung ein symmetrischer und kein asymmetrischer Algorithmus benutzt werden. Eine Schlüsselverteilung à la *Kerberos* [30] oder *Needham-Schroeder* [31] soll nicht eingesetzt werden. Die manuelle

Schlüsselverteilung ist völlig ausreichend. Glücklicherweise verlieren die übermittelten Daten ihren Wert innerhalb einiger Sekunden (Dauer der Verarbeitung durch die IDS-Komponenten).

Kommunikationsprotokolle werden bei jeder Transaktion von Daten zwischen allen Prozessen, egal ob lokal oder entfernt, benutzt. Verschlüsselung ist hingegen aber nur nötig, wenn die Daten über ein Netzwerk an ein entfernten Prozess geschickt werden. Sollten die Daten bspw. über ein *Loopback Device* gehen, muss dem Anwender die Möglichkeit bleiben, die Verschlüsselung abzuschalten.

Bei der Verschlüsselung der Daten wurden drei Ziele verfolgt:

- Schutz vor Manipulation
- Schutz vor unbefugtem Lesen
- Schutz gegen *Replay Attacks*

Manipulation Die Manipulation unserer Audit-Daten würde unweigerlich dazu führen, dass wir nach dem Entschlüsseln unbrauchbare Daten erhielten. Das Resultat wäre zum einen fehlerhafte Einträge in unseren Datenbanken und zum andern, was wesentlich schlimmer ist, wird die Analyse dadurch gestört. Um dem zu entgegen muss eine Prüfsumme von unserem Klartext berechnet werden. Diese Prüfsumme wird mit dem Klartext verschlüsselt und muss vom Kommunikationspartner validiert werden.

Lesen Das Lesen unserer Daten muss mit Hilfe von Kryptographie verhindert werden, um einem Angreifer die Möglichkeit zu nehmen, in Erfahrung zu bringen, was das IDS sieht und erkennt. Zudem ist die Verschlüsselung zwingend notwendig, um die o.g. Manipulation zu verhindern.

Replay Attacks Ein potentieller Angreifer kann *Replay Attacks* (s. [11], S. 70) dazu benutzen, die Analyseeinheit unseres Intrusion-Detection-Systems durcheinander zu bringen, Einträge in den Datenbanken vorzunehmen oder sogar, um Reaktionen ausführen zu lassen.

Die einfache Verschlüsselung der Nachricht hilft hier nicht gegen einen Angriff auf das Protokoll. Da immer der selbe Schlüssel benutzt wird, kann eine aufgezeichnete Nachricht später wieder eingespielt werden und würde vom IDS akzeptiert und verarbeitet.

Es gibt mehrere Lösungsansätze, um *Replay Attacks* zu verhindern. Für unser IDS wird einfach ein Zeitstempel benutzt. Voraussetzung hierfür ist aber, dass auf

allen beteiligten Rechnern die Zeit synchronisiert wird bspw. über NTP. Da das Paket von Applikationsebene zu Applikationsebene jedoch etwas Zeit benötigt muss der Empfänger ein Zeitfenster erlauben, in dem ein Paket ankommen darf. Dieses Fenster ist aber nur für das erste Paket nötig, nachfolgende Pakete müssen immer einen höheren Zeitstempel haben als das vorherige. Dieses Fenster erlaubt immer noch einen *Replay Angriff* mit dem ersten Paket; die Angriffsfläche wird jedoch verkleinert. Da diese Lösung nicht absolut wasserdicht ist, muss eine Verbesserung vorgenommen werden.

Mit einigen Veränderungen ließe sich die Gefahr einer *Replay Attacke* noch weiter verringern. Beispielsweise könnte das erste Paket ausschließlich zur Zeitsynchronisation benutzt werden und keine weiteren Daten enthalten. Damit könnte ein Angreifer aber immer noch die ganze Verbindung inkl. Zeitsynchronisationspaket duplizieren, um sein Ziel zu erreichen. So ein Angriff ist aber nur möglich, wenn der Server multi-threaded, also parallel, läuft. Ein parallel laufender Server sollte also noch Absenderadresse und Absenderport in einer globalen Tabelle für aktive Client-Sessions verwalten und bei neuen Verbindungen diese Werte prüfen.

Für unser IDS wurde jedoch ein viel einfacheres und dennoch effektives Mittel gegen *Replay Attacken* gefunden. Zwischen den parallelen Clientverbindungen soll einfach eine bestimmte Zeit (Zeitfenster + n) gewartet werden, so dass der Zeitstempel wiedereingespielter Pakete auf jedenfall zu alt ist.

Kapitel 3

Implementierung

3.1 Intrusion Detection Message Exchange Format

Das *Intrusion Detection Message Exchange Format* (IDMEF [1]) ist ein von der *Intrusion Detection Working Group* (IDWG [28]) entwickeltes objektorientiertes Datenformat. IDMEF basiert auf *Unified Modeling Language* (UML) und *Extensible Markup Language* (XML), um ein erweiterbares, aber dennoch einheitliches Format definieren zu können.

IDMEF wurde entwickelt, um Informationen zwischen einem IDS-Sensor und dem IDS-Manager auszutauschen. Aber man kann es auch benutzen, um bspw. Alar-me von verschiedenen Intrusion-Detection-Systemen in einer einzigen Datenbank zu sammeln, zu korrelieren oder mit Hilfe eines *Graphical User Interface* darzustellen.

3.1.1 IDMEF-Message-Aufbau

UML bietet die Möglichkeit, Beziehungen zwischen Klassen aufzuzeigen und stammt eigentlich aus der Softwareentwicklung. Eine Klasse besteht aus einem Namen und aus Attributen, dabei müssen Attribute nicht zwingend angegeben werden. Klassen werden wie folgt dargestellt:

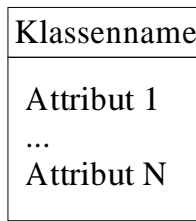


Abbildung 3.1: UML Klasse

Im UML Schema werden folgende Mengenangaben definiert:

- **n**: genau n (ohne Angabe: 1)
- **0..***: Null oder mehr
- **1..***: Eins oder mehr
- **0..1**: Null oder Eins (oder auch „optional“)
- **n..m**: von n bis m (einschließlich)

Das IDMEF-Modell benutzt lediglich die Beziehungen Vererbung und Gruppierung. Bei der Vererbung gibt es eine Oberklasse und eine beliebige Anzahl von Unterklassen. Eine Unterklasse erbt alle Attribute, Operationen und Beziehungen der Oberklasse. Sie ist aber in der Lage weitere Operationen oder Attribute zu definieren.

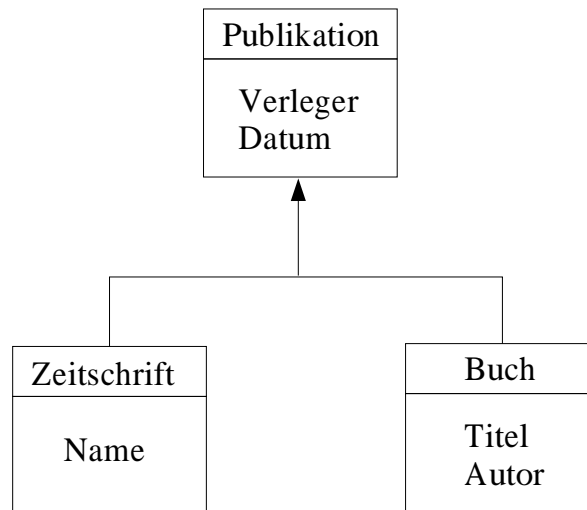


Abbildung 3.2: UML Vererbung

Eine Gruppierung ist eine Vereinigung von Klassen. Dabei sind die einzelnen Klassen und deren Attribute Teil der Gruppierungsklasse.

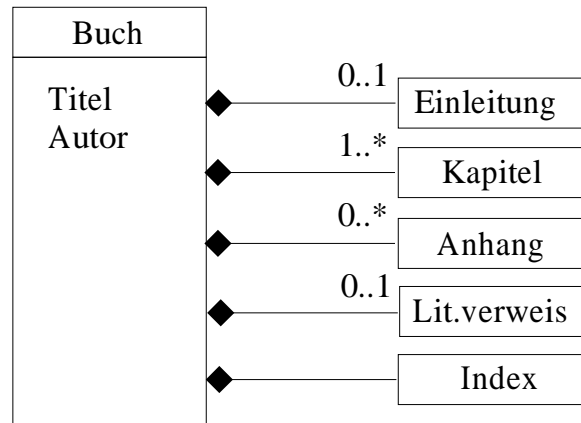


Abbildung 3.3: UML Gruppierung

Eine komplette IDMEF-Nachricht ist wie folgt aufgebaut:

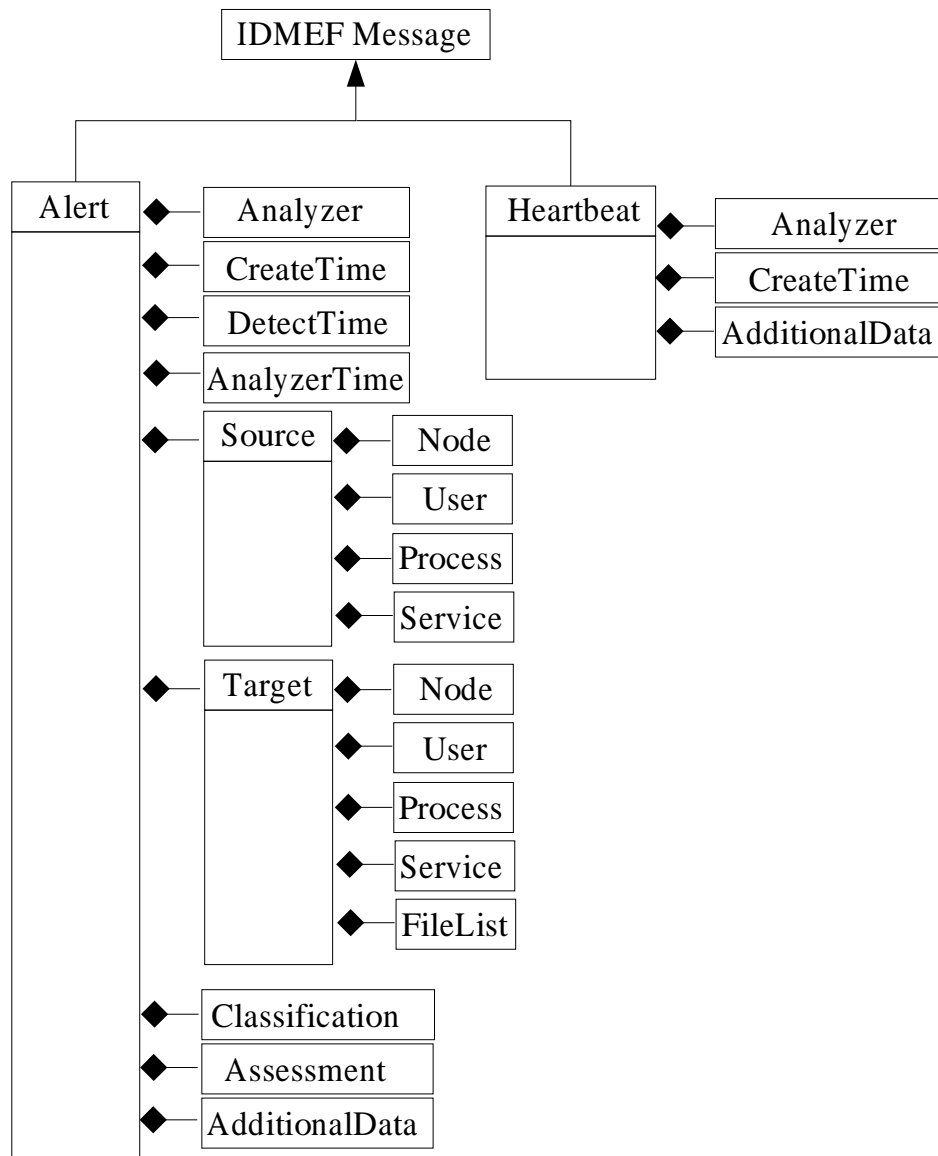


Abbildung 3.4: IDMEF–Aufbau

3.2 Design

Um die geforderte Modularität in Bezug auf dynamisch ladbare Module zu erfüllen, kann unter Linux die Funktion `dlopen(3)` benutzt werden. Diese Funktion dient primär zum Laden von *Shared Libraries*, kann aber hervorragend für unsere Zwecke

benutzt werden. Leider verwenden die unterschiedlichen Unix-Derivate ebenso unterschiedliche Verfahren, um *Shared Libraries* einzubinden. Aus diesem Grund wird die Bibliothek *libltdl* des *libtool Projektes* von GNU [32] benutzt, um die Portabilität zu wahren.

Zur Unterstützung des IDMEF-Models wurde die *libidmef* Bibliothek von SILICONDEFENSE [33] benutzt. Diese Bibliothek stellt eine C-API zur Verfügung, um die IDMEF-Dokumente quasi objektorientiert erzeugen und handhaben zu können. Leider fehlt hier die Implementierung eines geeigneten Transportprotokolls (bspw. IAP und IDXP) ¹, so dass ein eigenes Protokoll entwickelt werden musste.

Der detaillierte Aufbau von M-ICE soll mit Hilfe des folgenden Diagramms verdeutlicht werden.

¹*libiap* [35] und *libidxp* [39] wurden später entdeckt und nicht mehr verwendet

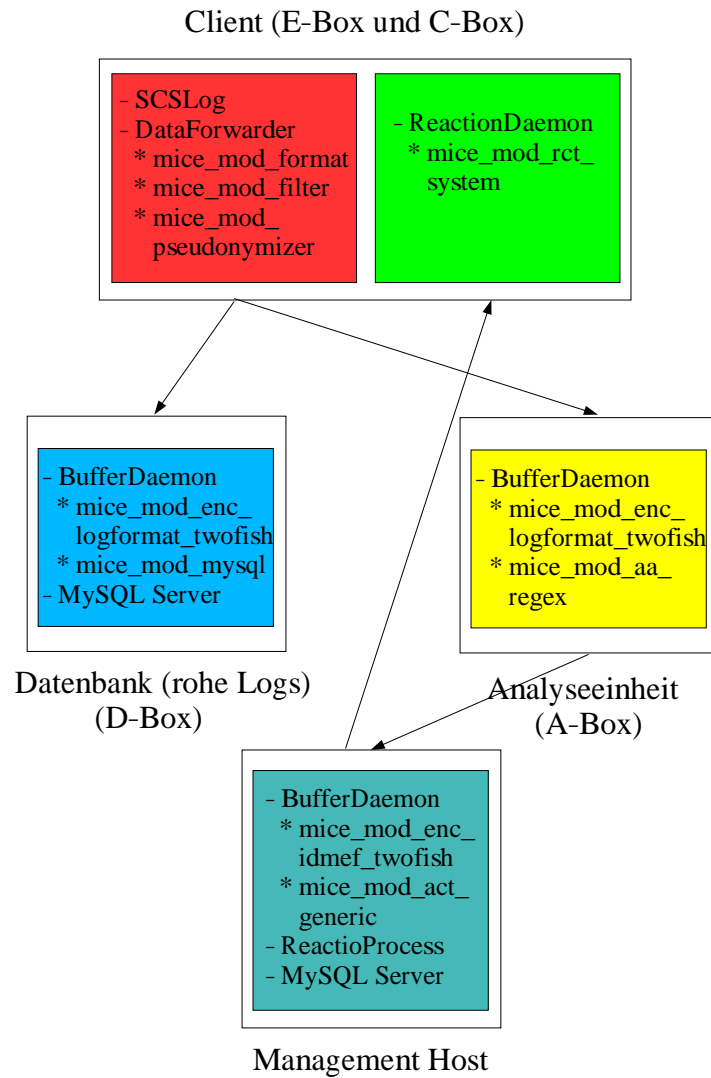


Abbildung 3.5: Detaillierter Aufbau von M-ICE

Stark abstrahiert stellt das ganze also einen Kreislauf dar. Der Kreis beginnt mit einer Aktion auf dem Client, diese Aktion wird im Verlauf des Kreises analysiert und die Reaktion auf dem Client schließt den Kreis wieder.

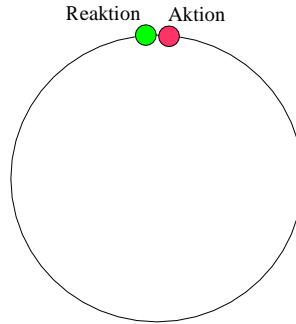


Abbildung 3.6: Abstrakter Aufbau von M-ICE

3.2.1 Die Bausteine

Nachfolgend soll die Implementierung genauer betrachtet werden.

SCSLog — Das Kernel-Modul

Da Linux von Haus aus kein Syscall-Logging unterstützt, wurde ein *Loadable Kernel Module* (LKM) entwickelt, welches dieses ermöglicht. Der Aufbau sieht wie folgt aus:

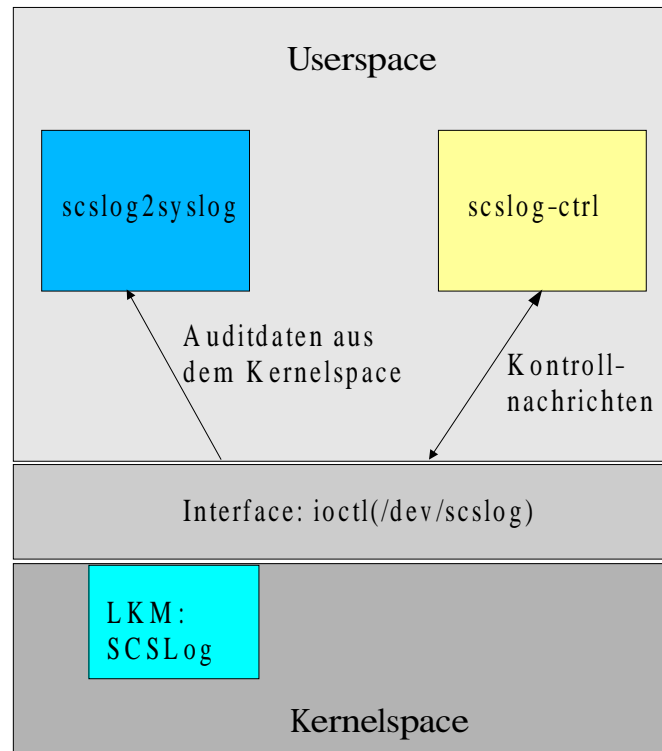


Abbildung 3.7: SCSLog Architektur

SCSLog besteht aus fünf Teilen, drei LKMs und zwei Benutzerprogrammen:

- scslog.o: Logging LKM
- scshide.o: versteckt ein LKM
- scsunrmv.o: macht ein LKM „unentfernbar“
- scslog-ctrl: Steuerung des SCSLog-Moduls
- scslog2syslog: Weiterleitung von SCSLog nach Syslog

SCSLog Das LKM *SCSLog* muss die originalen, vom Kernel angebotenen, Systemaufrufe durch seine Log-Funktionen erweitern. Die Systemaufrufe dürfen nicht komplett ersetzt werden, da sonst ein Arbeiten mit dem System nicht mehr möglich wäre. Um dies zu erreichen, wird die *Syscall Table* des Kernels verändert, indem der

Zeiger auf die original Kernel-Funktion gesichert und anschließend auf die Adresse der Log-Funktion gesetzt wird.

```
/* remember old value (pointer to original function) */
sci_table[syscall_num].bakup_ptr = sys_call_table[syscall_num];
```

```
[...]
```

```
/* replace with our funtion (intercept all calls) */
sys_call_table[syscall_num] = (void *) log_ptr;
```

Folgende Funktion aus scslog.o übernimmt diese Aufgabe:

```
/*
** System Call-Table Redirect Routine
*/
static int redirect_syscall_table(u_int action,
                                int syscall_num,
                                char *syscall_name,
                                int (*log_ptr)())
{
    DBG1(KERN_INFO "SCSLog - Debug: redirect_syscall_table...\n")

    switch(action)
    {
        case ACTION_WRAP:
            if(sci_table[syscall_num].logged == TRUE)
                return(-1);

            sci_table[syscall_num].logged = TRUE;

            printk(KERN_INFO "SCSLog - start monitoring syscall: %s()\n",
                    syscall_name);

            /* remember old value (pointer to original function) */
            sci_table[syscall_num].bakup_ptr = sys_call_table[syscall_num];
```

```

    /* replace with our funtion (intercept all calls) */
    sys_call_table[syscall_num] = (void *) log_ptr;
    break;
case ACTION_UNWRAP:
    if(sci_table[syscall_num].logged != TRUE)
        return(-1);

    sci_table[syscall_num].logged = FALSE;

    printk(KERN_INFO "SCSLog - stop monitoring syscall: %s()\n"
            ,sci_table[syscall_num].name);

    /* restore old pointer */
    sys_call_table[syscall_num] = sci_table[syscall_num].bakup_ptr;
    sci_table[syscall_num].bakup_ptr = NULL;
    break;
default:
    return(1);
}

return(0);
}

```

Um die Analyse der Log-Daten von Benutzerapplikationen wie gefordert zu vereinfachen, wurde ein einfaches Format gewählt. Hier ein Beispiel (Zeile wurde umgebrochen):

```

Syscall: socketcall | Program: reactiondaemon | PID: 2083 |
UID: 0 | EUID: 0 | Call: 5 = socketcall(5, bffff414) |
Comment: accept from IP: 10.0.0.10 Port: 33211 |

```

Die Log-Zeile besteht aus sieben Elementen, die genug Informationen für eine Analyse liefern sollten.

Feld	Token	Beschreibung
1	Syscall:	Name des Syscalls
2	Program:	Programm, welches den Syscall ausführt
3	PID:	Prozess ID des Programms
4	UID:	reale User ID des Programms
5	EUID:	effektive User ID des Programms
6	Call:	kompletter Syscall mit Rückgabewert und Parametern
7	Comment	u.U. Kommentar zur weiteren Erklärung

Der Linux-Kernel 2.4.18 bietet ca. 220 Syscalls an. Zum aktuellen Zeitpunkt werden aber nur die wichtigsten Systemaufrufe unterstützt.

- `execve()`
- `chmod()`
- `open()`
- `symlink()`
- `setuid()`
- `setgid()`
- `setreuid()`
- `setregid()`
- `socketcall()`

Im Konzept wurde festgelegt, einen Ringpuffer für die Zwischenspeicherung zu benutzen. Nachfolgend werden die globalen Variablen und Funktionen, die zur Handhabung des Ringpuffers notwendig sind, gezeigt.

```
/*  
** Ring Buffer for buffering Log Data  
*/
```

```

u_int rb_maxline = MAXBYTESPERLINE;
u_int rb_maxring = MAXLINESPERRING;

u_int rb_pos_w;
u_int rb_pos_r;
u_int rb_overflow_w;
u_int rb_overflow_r;
u_int rb_records;

char ringbuffer[500][4096];

[...]

int rb_write(char *log_data, int overwrite);
int rb_read(char *log_data, int release);
[...]
```

Zur Zeit ist dieser Ringpuffer noch statisch, es sind aber alle nötigen Ansätze im Quellcode von *SCSLog* vorhanden, um später die Größe aus der Benutzerebene via `ioctl(2)` anpassen zu können.

SCSHide Um *SCSLog* „unsichtbar“ werden zu lassen, wurde *SCSHide* entwickelt, welches sich eines einfachen Tricks bedient. Der Linux-Kernel benutzt eine verkettete Liste, um Informationen über geladene LKMs zu verwalten. *SCSHide* geht nun diese Liste durch und sucht nach dem Modulnamen, den es als Argument auf der Befehlszeile erhalten hat. Ist das Modul gefunden, wird der Zeiger aus dem vorigen Listenelement, der auf das zu versteckende Listenelement zeigt, auf die Adresse für das nachfolgende Element gesetzt. Somit ist der Eintrag für unser Modul nicht mehr Teil der Liste und somit unzugänglich für den Kernel.

Folgender Code ist dafür verantwortlich:

```

/* check the names of all modules loaded before scshide */
for(prevmod = thismod, currentmod = thismod->next;
    currentmod != NULL;
    prevmod = currentmod, currentmod = currentmod->next)
{
    if(strcmp(currentmod->name, module))
        continue;
```

```

    if(messages)
        printk(KERN_INFO "SCSHide - found module '%s'\n", currentmod->name);

    break;
}

if (currentmod == NULL)
{
    printk(KERN_INFO "SCSHide - module '%s' not found!\n", module);
    return(0);
}

/* remove the module structure from the modules list */
if(messages)
    printk(KERN_INFO "SCSHide - remove listentry of module '%s'\n",
                                                    currentmod->name);
prevmod->next = currentmod->next;

```

SCSUnrmv Das LKM *SCSUnrmv* ist das Letzte in der Reihe der Kernel-Module. Genau wie *SCSHide* verändert auch *SCSUnrmv* die verkettete Liste im Linux-Kernel. Dabei wird diesmal aber nicht die Reihenfolge der Liste verändert, sondern lediglich der *Use Counter* inkrementiert. Linux entfernt ein Modul nicht, solange der *Use Counter* größer als Null ist.

SCSLog-Ctrl Zur Steuerung des LKMs *SCSLog* über die Gerätedatei */dev/scslog* dient das Programm *scslog-ctrl*. So kann man bspw. wie folgt das Logging für den Syscall *open(2)* und *setuid(2)* ein- und wieder ausschalten:

```

root@HotSpot# scslog-ctrl -s 5,23
root@HotSpot# scslog-ctrl -u 5,23

```

Als kleines Beispiel soll gezeigt werden, wie über *ioctl(2)* das Logging für einen beliebigen Systemaufruf gesetzt wird, nachdem sicher gestellt wurde, dass dieser Systemaufruf nicht bereits überwacht wird.

```

if((retval = ioctl(sdev, SCSLOG_IOCQLOGGING, &sc_table[idx].num)) < 0)
{
    err_mesg(WARN, "%s: Can't query status for syscall number %d (%s)"
            , pname, sc_table[idx].num, sc_table[idx].name);
}

```

```

    return(-2);
}

if(retval == TRUE)
    continue; // syscall already set, skip to the next

if(ioctl(sdev, SCSLOG_IOCLOGGING, &sc_table[idx].num) < 0)
{
    err_mesg(WARN, "%s: Can't set monitoring for syscall number %d (%s)"
            , pname, sc_table[idx].num, sc_table[idx].name);
    return(-3);
}

```

Scslog2Syslog Das Hilfsprogramm `scslog2syslog` hat lediglich die Aufgabe, in einer Endlosschleife die Log-Zeilen von der Gerätedatei `/dev/scslog` zu lesen und an *Syslog* weiterzureichen.

DataForwarder — Datenweiterleitung

Im Konzept wurde für den *DataForwarder* gefordert, dass die Privilegien des Super-Users nur wenig genutzt werden und die logischen Teile, soweit wie möglich, in eigene Prozesse aufgeteilt werden. Die untenstehende Abbildung zeigt das Design des *DataForwarders*.

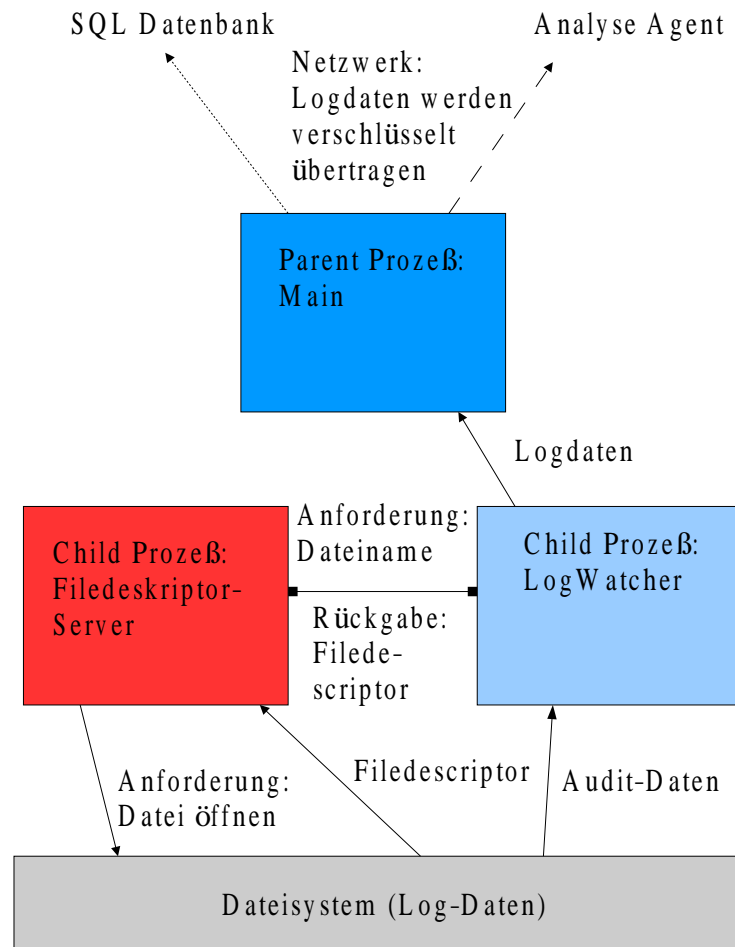


Abbildung 3.8: DataForwarder Architektur

In diesem Modell läuft nur der **Filedescriptor Server** (rot gekennzeichnet) mit Super-User-Rechten, da er die zu überwachenden Dateien öffnet; die anderen beiden Prozesse laufen mit normalen Benutzerrechten. Der **LogWatcher-Prozess** fordert den **Filedescriptor Server** auf, eine bestimmte Datei zu öffnen und erhält dann den zugehörigen Dateideskriptor über einen *Unix Domain Socket* zurück. Entdeckt der **LogWatcher-Prozess** neue Daten, dann werden diese in einem *Shared Memory Segment* geschrieben und dort für die weitere Verarbeitung vom **Main-Prozess** gelesen. Zur Synchronisation zwischen den beiden Prozessen für Lese- und Schreiboperationen von und in das *Shared Memory Segment* dient ein *Semaphore*.

Desweiteren enthielt die Konzeptbeschreibung für den *DataForwarder* sechs Hauptaufgaben:

- Sammlung
- Filterung
- Pseudonymisierung
- Formatierung
- Verschlüsselung
- Weiterleitung

Sammlung Aus der Konfigurationsdatei des *DataForwarders* wird eine Liste von Dateinamen gelesen, die durch den **Filedescriptor Server** geöffnet und vom **LogWatcher** überwacht werden. Der **LogWatcher** achtet dabei auf folgende Veränderung bei den Dateien:

- Entfernung aus dem Dateisystem
- neuer Inode
- gestutzte Dateilänge
- Modifikationszeit

Zusätzlich ist die Reihenfolge, in der die Veränderungen geprüft werden, wichtig, da bspw. eine Änderung in der Modifikationszeit einer Datei nicht zwangsweise auf das Hinzufügen von Daten schließen lässt. Die Reihenfolge, wie sie oben in der Liste angegeben ist, ist auch die Reihenfolge, die im Code des **LogWatchers** benutzt wird.

Bei der Bearbeitung der Dateien wird zwischen regulären Dateien und Gerädateien unterschieden. Bei Gerädateien wird einfach geprüft, ob Daten zum Lesen vorhanden sind, wenn dem so ist, dann werden alle vorhandenen Daten gelesen und in das *Shared Memory Segment* geschrieben.

Filterung Die zur Filterung geforderten regulären Ausdrücke wurden mit Hilfe der „POSIX regex functions“ der GNU C-Bibliothek implementiert. Das Filtermodul `mice_mod_filter` wird im Kapitel »Die Module« genauer betrachtet.

Pseudonymisierung Das Pseudonymisierungsmodul `mice_mod_pseudonymizer` enthält keinen funktionalen Code.

Formatierung Die Formatierung wurde im Modul `mice_mod_logformat` implementiert und wird ebenfalls im Kapitel »Die Module« genauer beschrieben.

Verschlüsselung Für den kryptografischen Teil von M-ICE wurde die Bibliothek MCrypt [36] benutzt.

Weiterleitung Die Konfigurationsdatei des *DataForwards* muss die IP-Adressen und Port-Nummern vom Datenbank-Server und der Analyseeinheit enthalten, damit die Daten weitergeleitet werden können. Sind IP-Adresse und Port-Nummer auf 0 gesetzt, wird nicht versucht, Daten an die entsprechende Komponente zu schicken. Wenn eine Verbindung aufgrund eines Fehlers nicht zustande kommen sollte, dann wird erneut versucht die Verbindung zu öffnen. Der untenstehende Code-Abschnitt soll dies für den SQL-Server verdeutlichen.

```
/*
** Send Data to MySQL Server.
*/
if(SqlSock >= 0)
{
    if(writen(SqlSock, (char *) &Message, sizeof(Message)) < 0)
    {
        log_mesg(WARN_SYS, "%s: Error while sending Data to SQL Server.
                        Try to reopen Connection...\n", cPrograme);

        close(SqlSock);

        if((SqlSock = tcp_open(CfgSQLIP[iSectSQL], NULL, CfgSQLPort[iSectSQL])) < 0)
            log_mesg(WARN_SYS, "%s: Error while opening Socket to SQL Server.\n"
                        , cPrograme);

        if(writen(SqlSock, (char *) &Message, sizeof(Message)) < 0)
        {
            log_mesg(WARN, "%s: Error while sending Data to SQL Server. Abort!\n"
                        , cPrograme);
            close(SqlSock);
        }
    }
}
```

```

    exit(-1);
  }
}
}

```

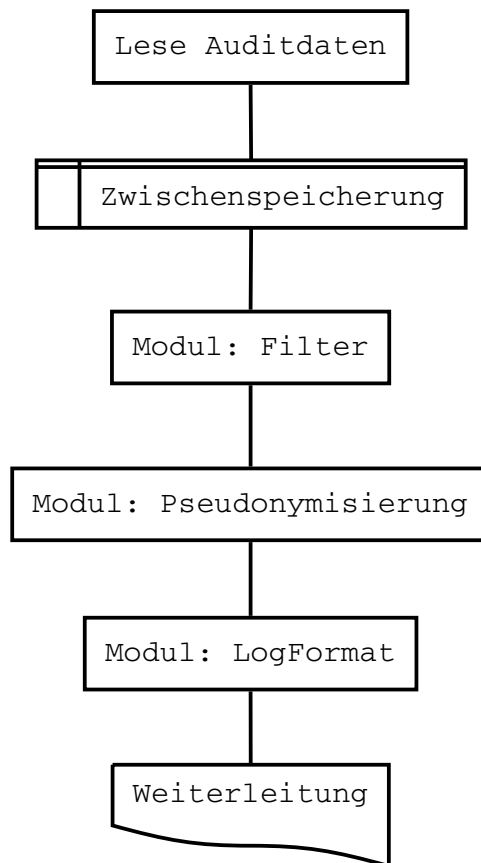


Abbildung 3.9: Datenflussdiagramm des DataForwarders

BufferDaemon — multifunktionaler Netzwerk-Server

Der *BufferDaemon* ist ein multifunktionaler Netzwerk-Server, der mit Hilfe von Modulen verschiedene Datenformate (*Encoding Modul*) vom Netz entgegennehmen und

in bestimmten Zeit-abständen verarbeiten (*Output Modul*) kann. Zur Zwischenspeicherung dient ein Ringpuffer, der entweder im RAM oder über *Memory Mapping* auf der Festplatte gehalten wird. Jede Zuordnung zwischen *Encoding Modul* und *Output Modul* bekommt ihren eigenen Ringpuffer, ihren eigenen Socket und einen eigenen Interval-Timer. Um diese Informationen zu verwalten, wird ein Array der Struktur **ProcQ** benutzt.

```

struct ProcessQueue
{
    struct ProcessQueue *prev;

    struct enc_info
    {
        int                iSock;
        saddr_in           SAddrIn;
        int                iCliSock;
        saddr_in           CliAddrIn;
        size_t             MsgSize;
        pthread_t          TID;
        char               *cModName;

        lt_dlhandle        dlHandle;
        const lt_dlinfo     *dlInfo;
        size_t             (*InitPtr)(char *);
        char *             (*FuncPtr)(char *, size_t);
        int                (*ClosePtr)(void);
    } Encoding;

    struct out_info
    {
        time_t             TimeInv;
        size_t             MsgSize;
        pthread_t          TID;
        char               *cModName;

        lt_dlhandle        dlHandle;
        const lt_dlinfo     *dlInfo;
        size_t             (*InitPtr)(char *);
        char *             (*FuncPtr)(char *, size_t);
        int                (*ClosePtr)(void);
    } Output;

    enum CacheMethod       CMethod;
    struct rb_info         Ringbuffer;
    struct mm_info         MMap;

```

```

    struct ProcessQueue *next;

} *ProcQ;

```

Der *BufferDaemon* lauscht an einem benutzerdefinierten Port, um Daten zu empfangen. Die empfangenen Daten werden an das *Encoding Modul* übergeben, dort dekodiert und in Klartextform als eindimensionales Char-Array an den Aufrufer zurückgegeben. Dieses Array wird im Ringpuffer gespeichert, nach einem bestimmten Zeitintervall vom *Output Modul* wieder ausgelesen und verarbeitet. Beide Module laufen in verschiedenen Threads.

```

/*
** Call Encode-Modules Decode-Function to get Plaintext Message
*/
if
(
    (
        cDecodedMsg =
            (*ProcQ[ProcQEnt].Encoding.FuncPtr)(cEncodedMsg,
                                                ProcQ[ProcQEnt].Encoding.MsgSize)
    ) == NULL
)
{
    log_mesg(WARN, "%s: Error while decoding received Data!\n", cProgrname);
    continue;
}

/*
** Now write the Plaintext Message to the Ringbuffer
*/
pthread_mutex_lock(&ProcQ[ProcQEnt].Ringbuffer.rb_mutex);
if
(
    intrBWrite(&ProcQ[ProcQEnt].Ringbuffer,
              cDecodedMsg, ProcQ[ProcQEnt].Output.MsgSize,
              TRUE) < 0
)
{

```

```

    log_mesg(WARN, "%s: Error: intrBWrite(cDecodedMsg,
        ProcQ[ProcQEnt].Output.MsgSize, TRUE)!\n", cProgrname);
}

pthread_mutex_unlock(&ProcQ[ProcQEnt].Ringbuffer.rb_mutex);

[...]

/*
** Read Data from Ringbuffer
*/
pthread_mutex_lock(&ProcQ[ProcQEntry].Ringbuffer.rb_mutex);
while
(
    intrBRead(&ProcQ[ProcQEntry].Ringbuffer,
        cMsg,
        ProcQ[ProcQEntry].Output.MsgSize,
        TRUE) != -1
)
{
    /*
    ** Call Module Function
    */
    if
    (
        (*ProcQ[ProcQEntry].Output.FuncPtr)(cMsg, ProcQ[ProcQEntry].Output.MsgSize)
        < 0
    )
    {
        log_mesg(WARN, "%s: Debug: Error while calling Modules FUNC Function\n"
            , cProgrname);
    }

    memset(cMsg, 0, ProcQ[ProcQEntry].Output.MsgSize);
}
pthread_mutex_unlock(&ProcQ[ProcQEntry].Ringbuffer.rb_mutex);

```

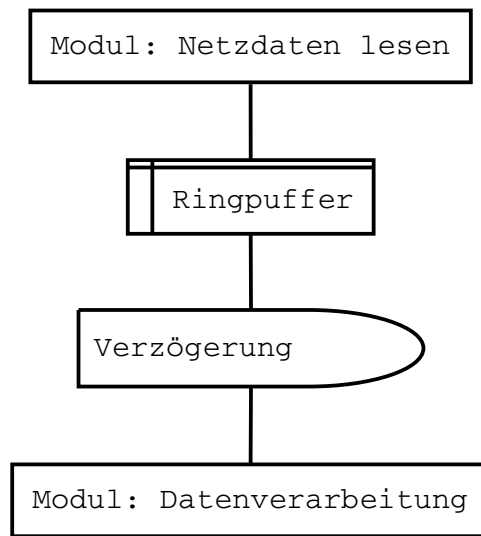



Abbildung 3.10: Datenflussdiagramm des BufferDaemons

ReactionProcess — Reaktionen nach Maß

Die *ReactionProcesses* lesen die *Reaction Message*, die vom Management-Host geschickt wurde, von einem FIFO und führen Ihre Aktion aus. M-ICE ist bei der Menge und Art seiner Reaktionen sehr flexibel, d.h. es kann jede denkbare Reaktion implementiert werden.

Aufbau der *Reaction Message*:

```

typedef struct
{
    char    cIdmefMsg[MAX_IDMEFMSGSIZE+1]        __attribute__((packed));
    char    cAlertID[RIDMSG_MAX_ALERTID+1]        __attribute__((packed));
    char    cAlertIDDesc[RIDMSG_MAX_ALERTDESC+1]  __attribute__((packed));
    int     iRID;
} RIDMsgFormat;
  
```

Folgende Reaktionsmöglichkeiten wurden implementiert.

Name	Beschreibung
rid_1_write_to_syslog	Gibt Alarminformationen an Syslog weiter
rid_2_send_to_alert_db	Schickt Alarminformationen zur Speicherung an eine Datenbank
rid_3_save_to_file	Schreibt die rohe IDMEF–Message in eine Datei
rid_4_countermeasure	Informiert den ReactionDaemon über Gegenmaßnahmen

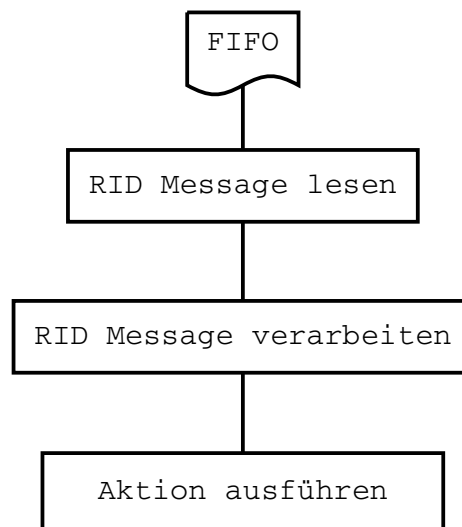


Abbildung 3.11: Datenflussdiagramm des ReactionProcess'

ReactionDaemon — Reaktionen ausführen

Für den *ReactionDaemon* wird eine Art *Remote Procedure Call* benötigt. Die angebotenen Funktionen werden hier ebenfalls als Module eingebunden. Unser Protokoll erlaubt jeweils drei Modi für Client– und für Server–Nachrichten.

Client–Nachrichten:

Modus	Beschreibung
EXEC	Führe Funktion aus
SHOW	Zeige alle angebotenen Funktionen an
CHECK	Überprüfe ob Funktion unterstützt

Server-Nachrichten:

Modus	Beschreibung
RETVAL	Rückgabewert der ausgeführten Funktion
ALL	Rückgabe aller angebotenen Funktionen
SUPPORTED	Funktion wird unterstützt

Die Strukturen für den *Remote Procedure Call* werden nachfolgend gezeigt, dabei ist für die einzelnen Modi jeweils eine Struktur vorhanden, diese Struktur wird dann in die eigentliche *Reaction Message*-Struktur eingefügt.

```
typedef struct
{
    char        alert_id[RIDMSG_MAX_ALERTID+1];
    u_int       reaction_id;
    uid_t       uid_for_exec;
    gid_t       gid_for_exec;
    u_int       function_id;
    u_int       num_of_args;
    char        arg_fmt_string[MAX_ARGSTRG_SIZE+1] __attribute__((packed));
    char        arg_fmt_param[MAX_FMTSTRG_SIZE+1] __attribute__((packed));
} stExecMsg;
```

```
typedef struct
{
    u_int       show;
} stShowMsg;
```

```
typedef struct
{
    u_int       function_id;
} stCheckMsg;
```

```
typedef struct
{
    int         ret_val;
} stRetValMsg;
```

```

typedef struct
{
    struct
    {
        u_int    uiID;
        char      *cModName[MAX_MODNAME+1] __attribute__((packed));
    } Function[MAX_FUNCID] __attribute__((packed));
} stAllMsg;

```

```

typedef struct
{
    u_int    supported;
} stSupportedMsg;

```

```

typedef struct
{
    short     sChkSum;
    time_t    Timestamp;

    u_int     Mode;

    union
    {
        stExecMsg      Exec;
        stShowMsg       Show;
        stCheckMsg      Check;
        stRetValMsg     Retval;
        stAllMsg        All;
        stSupportedMsg  Supported;
    } ModeData;
} stReactionMsg;

```

Die auszuführende Funktion im EXEC-Modus wird über eine **Function ID** Nummer bestimmt. Die **Function ID** ist direkt einem Modul zugeordnet. Die Konfigurationsdatei des *ReactionDaemons* verdeutlicht dies gut:

```
[FUNCTION_ID]  
FID = 0x000001
```

```
[REACTION_MODULES]  
RCT_MOD = mice_mod_rct_system
```

```
[REACTION_MODULES_CONFIG_FILE]  
RCT_FILE = /etc/M-ICE/mice_mod_rct_system.conf
```

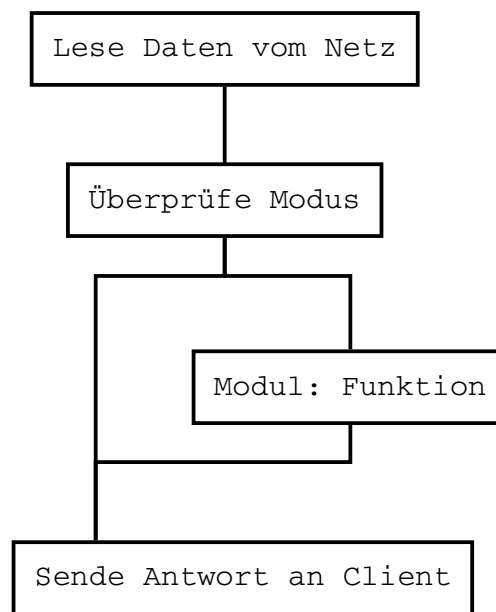


Abbildung 3.12: Datenflussdiagramm des ReactionDaemons

3.2.2 Die Module

Wie zuvor schon erwähnt, wurden wichtige Funktionen in dynamisch ladbare Module ausgelagert, um die Flexibilität und Erweiterbarkeit weiter zu erhöhen. Im Laufe der Entwicklung von M-ICE wurden einige Module implementiert, die hier nur kurz erklärt werden sollen.

Die meisten Module bestehen lediglich aus einer Konfigurationsdatei und dem Modul selbst:

- /etc/M-ICE/mice_mod_<name>.conf
- /lib/M-ICE/mice_mod_<name>.*

Jedes Modul muss drei Funktionen zur Verfügung stellen:

- mice_mod_<name>_LXT_init: Initialisierung des Moduls
- mice_mod_<name>_LXT_func: eigentliche Funktion
- mice_mod_<name>_LXT_close: Modul schließen

Die Funktionen haben folgende Argumente:

Funktion	Argumente
init	char *cConfigFile
func	funktionsabhängig
close	void

mice_mod_filter

Zur Datenreduktion wird direkt auf dem Client mit Hilfe von regulären Ausdrücken gefiltert. Die Filterregeln werden einfach in die Konfigurationsdatei `mice_mod_filter.conf` eingetragen und durch folgenden Code verarbeitet:

```
for
(
    TmpList = _mmf_CfgRules[_mmf_iSectFilterRules];
    TmpList != NULL;
    TmpList = TmpList->next
)
{
    if(re_compile_pattern(TmpList->str, strlen(TmpList->str), _mmf_RegExBuf) != 0 )
    {
        log_mesg(WARN, "mice_mod_filter: Error while compiling Regular Expression
```

```

        '%s'. Skipped...\n", TmpList->str);
    continue;
}

_mmf_RegExBuf->regs_allocated = REGS_FIXED;

if(re_match(_mmf_RegExBuf, cLogData, LogDataLen, 0, NULL) >= 0)
    break;
}

if(TmpList != NULL)
{
    /*
    ** We find a matching Pattern.
    ** Let's skip this Logentry, because the User wants to filter it out.
    */
    return(1);
}

return(0);

```

Das Filtermodul gibt 1 an den Aufrufer zurück, wenn die Regel zutrifft, und 0 wenn nicht. Dem Aufrufer ist nun überlassen, was er mit dieser Information anfängt. In unserem Fall ignorieren wir bei einem Rückgabewert von 1 den Log-Eintrag.

mice_mod_logformat

Um die nativen Log-Daten des Clients mit weiteren wichtigen Informationen anzureichern, benutzt der *DataForwarder* das Modul **mice_mod_logformat**. Die Struktur der aufgewerteten Audit-Daten sieht wie folgt aus:

```

typedef struct
{
    char    cHost      [MAX_HOST]      __attribute__((packed));
    char    cDomain    [MAX_DOMAIN]    __attribute__((packed));
    char    cIP        [MAX_IP]        __attribute__((packed));
    char    cOSSystem  [MAX_OS]        __attribute__((packed));
    char    cRelease   [MAX_RELEASE]   __attribute__((packed));

```



```

char    cVersion  [MAX_VERSION]    __attribute__((packed));
char    cDate     [MAX_DATE]       __attribute__((packed));
char    cTime     [MAX_TIME]       __attribute__((packed));
char    cLogLine  [MAX_DATA]       __attribute__((packed));
u_short sChkSum                               __attribute__((packed));
} LogFormat;

```

mice_mod_pseudonymizer

Ein Pseudonymisierungsverfahren wurde nicht implementiert.

mice_mod_mysql

Damit der *BufferDaemon* die Audit-Daten in eine MySQL-Datenbank speichern kann, lädt er das Modul `mice_mod_mysql` als *Output Modul*. Dieses Modul verwaltet drei Tabellen `rawlog_line`, `scslog_line` und `firewall_line`. Die *SCSLog*- und die *SuSEfirewall2*-Einträge werden vom Modul analysiert, dabei werden die Daten in die entsprechenden Tokens und deren Werte aufgespalten. Die Werte werden in die dazugehörige Spalte der jeweiligen Tabelle eingetragen. Diese Aufteilung ermöglicht es dem Anwender zielgenau nach Systemaufrufen, Benutzeridentitäten, IP-Adressen, Port-Nummern und dergleichen zu suchen.

Tabellenaufbau für *Syslog*:

```

CREATE TABLE rawlog_line ( hostname    TEXT,
                             domain     TEXT,
                             ip         TEXT,
                             osystem    TEXT,
                             release    TEXT,
                             version    TEXT,
                             date       DATE,
                             time       TIME,
                             logline    TEXT,
                             signature  TEXT );

```

Tabellenaufbau für *SCSLog*:

```

CREATE TABLE scslog_line ( hostname TEXT,
                             domain TEXT,
                             ip TEXT,
                             osystem TEXT,
                             release TEXT,
                             version TEXT,
                             date DATE,
                             time TIME,
                             syscall TEXT,
                             program TEXT,
                             pid INT UNSIGNED NOT NULL,
                             uid INT UNSIGNED NOT NULL,
                             euid INT UNSIGNED NOT NULL,
                             call TEXT,
                             comment TEXT );

```

Tabellenaufbau für *SuSEfirewall2*:

```
CREATE TABLE firewall_line ( action      TEXT,
                             if_in       TEXT,
                             if_out      TEXT,
                             mac         TEXT,
                             source      TEXT,
                             destination TEXT,
                             ip_length   TEXT,
                             tos         TEXT,
                             prec        TEXT,
                             ttl         TEXT,
                             id          TEXT,
                             protocol    TEXT,
                             src_port    TEXT,
                             dst_port    TEXT,
                             pac_length  TEXT,
                             date        DATE,
                             time        TIME );
```

mice_mod_aa_regex

Das Analysemodul verarbeitet die Log-Daten aus dem Ringpuffer des *BufferDaemons* mit Hilfe von regulären Ausdrücken. Dabei werden nach erfolgreicher Erkennung intern Informationen über die passende Regel gespeichert. Hier ein Beispiel für die Authentication Messages:

```
/******
**
** A U T H    R U L E S
**
*****/

// Auth Success
for(TmpList = _mice_mod_aa_regex_CfgAuth.aCfgAuthSucc
      [_mice_mod_aa_regex_CfgAuth.iSectionNr],
    iRuleCtr = 0;
    TmpList != NULL;
```

```

        TmpList = TmpList->next, iRuleCtr++)
{
    if(_mice_mod_aa_regex_DoesMatch(cLogLine, LogLineLen, TmpList->str) == TRUE)
    {
        /*
        ** Fill in Match Info Data
        */
        mInfo->iMatched      = TRUE;
        mInfo->iSectNr       = _mice_mod_aa_regex_CfgAuth.iSectionNr;
        mInfo->iSectType     = ST_AUTH;
        mInfo->iRuleNr       = iRuleCtr;
        mInfo->cRuleType     = RT_AUTH_S;
        mInfo->cMatchedRule  = TmpList->str;
        mInfo->cSendTo       = _mice_mod_aa_regex_CfgAuth.cCfgSendTo
                               [_mice_mod_aa_regex_CfgAuth.iSectionNr];

        return(TRUE);
    }
}

```

Die Informationen werden später benötigt, um unsere IDMEF-Nachricht zu bauen. Dafür werden folgende Funktionen von `mice_mod_aa_regex` benutzt:

- `xmlNodePtr _mice_mod_aa_regex_FormatIDMEF(LogFormat LogFmt, MatchInfo mInfo)`
- `xmlNodePtr _mice_mod_aa_regex_BuildMsg(LogFormat LogFmt, MatchInfo mInfo)`
- `xmlNodePtr _mice_mod_aa_regex_BuildMsgTree(LogFormat LogFmt, MatchInfo mInfo)`
- `xmlNodePtr _mice_mod_aa_regex_BuildAnalyzer(LogFormat LogFmt)`
- `xmlNodePtr _mice_mod_aa_regex_BuildSource(LogFormat LogFmt)`
- `xmlNodePtr _mice_mod_aa_regex_BuildTarget(LogFormat LogFmt)`

In der Funktion `_mice_mod_aa_regex_BuildMsgTree` werde alle IDMEF-Klassen zusammengeführt:

```

MessageClass = newIDMEF_Message
(
    newAttribute("version",IDMEF_MESSAGE_VERSION),
    newAlert
    (
        newSimpleElement("ident",
            intToString(_mice_mod_aa_regex_StaticInfo.ulAlertID)),
        AnalyzerClass,
        newCreateTime(NULL),
        newDetectTime(NULL),
        newAnalyzerTime(NULL),
        SourceClass,
        TargetClass,
        newClassification
        (
            newAttribute("origin", "vendor-specific"),
            newSimpleElement("name", mInfo.cRuleType),
            newSimpleElement("url", "http://unkonwn.de"),
            NULL
        ),
        newAdditionalData
        (
            newAttribute("meaning","Logline"),
            newAttribute("type","string"),
            newSimpleElement("value", LogFmt.cLogLine),
            NULL
        ),
        NULL
    ),
    NULL
);

```

Nachdem die IDMEF-Nachricht gebaut wurde, wird sie mit dem Twofish-Algorithmus verschlüsselt und an ihr Ziel geschickt. Als Ziel kann nichts, der Management-Host, die Analyseeinheit oder beide angegeben werden. Dieses Verhalten kann für jede Kategorie in der Konfigurationsdatei des Analysemoduls separat gesetzt werden. Es ist denkbar, bereits analysierte Daten mit Hilfe eines anderen Verfahrens von einem anderen Analyseagenten genauer untersuchen zu lassen oder Statistiken zu führen.

mice_mod_enc_logformat_twofish

Zum Lesen der Daten vom *DataForwarder* muss der *BufferDaemon* das *Encoding Modul* `mice_mod_enc_logformat_twofish` laden.

```
memcpy(cDecodedMsg, CMsg->cCipherText, CMsg->CipherTextLen);

for(iCnt = 0; iCnt < CMsg->CipherTextLen; iCnt++)
    mdecrypt_generic(_mice_mod_enc_logformat_twofish_CryptoInfo.CryptModule,
                    &cDecodedMsg[iCnt], 1);

LFmt = (LogFormat *) cDecodedMsg;

[...]

/*
** Verify Checksum (CRC)
*/
sChkSum_Orig    = LFmt->sChkSum;
LFmt->sChkSum    = 0;
sChkSum_New     = in_chksum((u_short *) LFmt, sizeof(LogFormat));

if(sChkSum_Orig != sChkSum_New)
{
    log_mesg(WARN, "%s: Debug: Checksum does not match... skipping Message\n",
            _mice_mod_enc_logformat_twofish_cPrograme);
    free(cDecodedMsg);
    return(NULL);
}

LFmt->sChkSum = sChkSum_Orig;

return(cDecodedMsg);
```

mice_mod_enc_idmef_twofish

Der *BufferDaemon*, der auf dem Management-Host läuft, liest das IDMEF-Format der Analyseeinheit mit Hilfe des *Encoding Moduls* `mice_mod_enc_idmef_twofish`. Dieses

Modul unterscheidet sich strukturell nicht besonders vom Modul `mice_mod_enc_logformat_twofish`. Es wird nach der Validierung der Prüfsumme lediglich ein Zeiger auf eine IDMEF-Nachricht zurückgegeben und nicht auf eine `LogFormat`-Nachricht.

`mice_mod_act_generic`

Das Modul `mice_mod_act_generic` dient als *Output Modul* für den *BufferDaemon*, um auf dem Management-Host die Forderungen nach ausbaubaren und systemübergreifenden Reaktionen zu erfüllen. Der Schlüssel für diese Eigenschaft ist das sog. *Match File*. Die Analyzer ID, die in der IDMEF-Nachricht kodiert ist, wird dazu benutzt, um das richtige *Match File* ausfindig zu machen. Die richtige Zuordnung wird durch folgende Schleifenverschachtelung eruiert:

```
for(iCnt_Matchfile = 0; iCnt_Matchfile <= MFInfoEntries; iCnt_Matchfile++)
{
    for(iCnt_Alert = 0; iCnt_Alert < IDMEFmsg->nalerts; iCnt_Alert++)
    {
        for
        (
            iCnt_Classification = 0;
            iCnt_Classification < IDMEFmsg->alerts[iCnt_Alert]->nclassifications;
            iCnt_Classification++
        )
        {
            // Check for RID 1
            for(iCnt_AID = 0;
                iCnt_AID <= stMFInfo[iCnt_Matchfile].AID_1Entries;
                iCnt_AID++)
            {
                if(stMFInfo[iCnt_Matchfile].iAID_1[iCnt_AID] == -1) // no entry
                    continue;

                // Should we react on this AlertID?
                if
                (
                    !strcasecmp(IDMEFmsg->alerts[iCnt_Alert]->
                                classifications[iCnt_Classification]->
                                name,
```


3.2.3 Die kryptografischen Protokolle

Bei der Datenverschlüsselung wurden drei Ziele verfolgt:

- Schutz vor Manipulation
- Schutz vor unbefugtem Lesen
- Schutz gegen *Replay Attacks*

Schutz vor unbefugtem Lesen und Manipulation Die geforderte Prüfsumme von unserem Klartext wird vom Sender berechnet und vom Empfänger nach der Entschlüsselung überprüft. Als erklärendes Beispiel soll hier die *LogFormat*-Nachricht zwischen *DataForwarder* und *BufferDaemon* der Analyseeinheit dienen:

```
typedef struct
{
    char    cHost      [MAX_HOST]      __attribute__((packed));
    char    cDomain    [MAX_DOMAIN]    __attribute__((packed));
    char    cIP         [MAX_IP]       __attribute__((packed));
    char    cOSSystem   [MAX_OS]       __attribute__((packed));
    char    cRelease    [MAX_RELEASE]   __attribute__((packed));
    char    cVersion    [MAX_VERSION]   __attribute__((packed));
    char    cDate       [MAX_DATE]     __attribute__((packed));
    char    cTime       [MAX_TIME]     __attribute__((packed));
    char    cLogLine    [MAX_DATA]     __attribute__((packed));
    u_short sChkSum     __attribute__((packed));
} LogFormat;

typedef struct
{
    u_int    IVLen                __attribute__((packed));
    char     IV[16]               __attribute__((packed));
    u_int    CipherTextLen        __attribute__((packed));
    char     cCipherText[1*sizeof(LogFormat)] __attribute__((packed));
} CipherMsg;
```

In `LogFormat.sChkSum` wird die Prüfsumme unserer *LogFormat*-Klartextnachricht gespeichert. Anschließend wird die *LogFormat*-Struktur mit Hilfe des Twofish-Algorithmus

verschlüsselt und nach `cCipherText[]` kopiert. Um Entwicklungs- und Ausführungszeit zu sparen wurde zur Berechnung der Prüfsumme ein einfacher *Cyclic Redundant Check*-Algorithmus und kein kryptografisch sicherer Hashalgorithmus, wie bspw. SHA, benutzt. Das Erstellen der Prüfsumme geschieht im Modul `mice_mod_logformat`:

```
/*
** Keep Data Integrity by calculating a CRC sum
*/
LogEntry->sChkSum = 0;
LogEntry->sChkSum = in_chksum((u_short *) LogEntry, sizeof(LogFormat));
```

Die *Chipher Message*, die unsere verschlüsselte `LogFormat`-Struktur als Nutzlast enthält, wird nun von dem *DataForwarder* über ein Netzwerk an einen Server geschickt.

```
for(iCnt = 0; iCnt < sizeof(LogFormat); iCnt++)
    mcrypt_generic(CryptModule, &Message.cCipherText[iCnt], 1);

[...]

if(writen(SqlSock, (char *) &Message, sizeof(Message)) < 0)
{
    log_mesg(WARN_SYS, "%s: Error while sending Data to SQL Server.
                Try to reopen Connection...\n", cPrograme);

    [...]
}
```

Der *BufferDaemon* muss dann nur noch die empfangene Nachricht entschlüsseln, die Prüfsumme berechnen und mit der vom Client mitgeschickten Prüfsumme vergleichen. Dieser Prozess findet im *Encoding Modul* statt, bspw. in `mice_mod_enc_logformat_twofish`.

```
/*
** Message is encryted. Let's decrypt it!
*/
[...]
memcpy(cDecodedMsg, CMsg->cCipherText, CMsg->CipherTextLen);
```

```

for(iCnt = 0; iCnt < CMsg->CipherTextLen; iCnt++)
    mdecrypt_generic(_mice_mod_enc_logformat_twofish_CryptoInfo.CryptModule,
                    &cDecodedMsg[iCnt], 1);

LFmt = (LogFormat *) cDecodedMsg;

/*
** Verify Checksum (CRC)
*/
sChkSum_Orig    = LFmt->sChkSum;
LFmt->sChkSum    = 0;
sChkSum_New     = in_chksum((u_short *) LFmt, sizeof(LogFormat));

if(sChkSum_Orig != sChkSum_New)
{
    log_mesg(WARN, "%s: Debug: Checksum does not match... skipping Message\n"
              , _mice_mod_enc_logformat_twofish_cProgname);
    free(cDecodedMsg);
    return(NULL);
}

LFmt->sChkSum = sChkSum_Orig;

```

Replay Attacken Der Schutz vor *Replay Attacken*, wie sie im Kapitel »Konzepte« beschrieben wurden, wurde wie folgt umgesetzt:

```

/*
** Handle Client Request - Thread
*/
void *voidHandleClientRequest(void *vArg)
{
    [...]

    time_t CurrentTimestamp = time(NULL) - REPLAYATTACK_DELAYWINDOW;

    [...]

```

```

/*
** Check Timestamp to avoid replay attacks
*/
if((time_t) ntohl(RctMsgPtr->Timestamp) > CurrentTimestamp)
    CurrentTimestamp = (time_t) ntohl(RctMsgPtr->Timestamp);
else
{
    log_mesg(WARN, "%s: Timestamp (%d) is lower then current Timestamp (%d).
                Close Connection to Client!\n", cPrograme,
    (time_t) ntohl(RctMsgPtr->Timestamp), CurrentTimestamp);
    goto THREAD_EXIT;
}

[...]

/*****
* We are done with Initialisation Phase, now let's start accepting *
* Client Requests *
*****/
CliInfo.CliAddrLen = sizeof(struct sockaddr_in);
while(TRUE)
{
    /*
    ** Wait for Client Request
    */
    log_mesg(WARN, "%s: Waiting for Client Requests...", cPrograme);

    if
    (
        (
            CliInfo.iCliSock =
            accept(iSock, (saddr *) &CliInfo.CliAddrIn, &CliInfo.CliAddrLen)
        ) == -1
    )
    {
        log_mesg(WARN_SYS, "%s: Warning: accept() fails! | Syserror", cPrograme);
        continue;
    }
}

```

```

/*
** Create new Thread, which handles Client Data.
*/
if(pthread_create(&CliInfo.TID, NULL,
                 voidHandleClientRequest,
                 (void *)&CliInfo))
    log_mesg(WARN, "%s: Error: Unable creat HandleClientRequest Thread"
             , cPrograme);

sleep(REPLAYATTACK_DELAYWINDOW+1);
}

```

3.3 Beispielkonfiguration

Nun, da die einzelnen Programme und Protokolle erläutert wurden, soll deren Zusammenspiel anhand einer Beispielkonfiguration verdeutlicht werden.

3.3.1 Client

Auf dem Client-Rechner, den es zu überwachen gilt, muss wenigstens der *DataForwarder* laufen, um die restlichen M-ICE-Einheiten mit Daten versorgen zu können. Zusätzlich kann das LKM *SCSlog* und der *ReactionDaemon* auf dem Client installiert werden.

Folgende Dateien und Verzeichnisse werden angelegt:

- /var/run/M-ICE
- /lib/M-ICE
- /etc/M-ICE
- /usr/local/sbin/dataforwarder
- /etc/M-ICE/dataforwarder.conf
- /sbin/rcMICE-dataforwarder
- /lib/M-ICE/mice_mod_filter.*
- /lib/M-ICE/mice_mod_pseudonymizer.*

- /lib/M-ICE/mice_mod_logformat.*
- /dev/scslog
- /lib/modules/2.4.18/scslog.o
- /lib/modules/2.4.18/scshide.o
- /lib/modules/2.4.18/scsunrmv.o
- /lib/modules/2.4.18/scsmkrmv.o
- /usr/sbin/scslog-ctrl
- /usr/sbin/scslog2syslog
- /sbin/rcMICE-syscallmonitor
- /etc/M-ICE/scslog-ctrl.list
- /usr/include/scslog/scslog.h

SCSLog

Der Benutzer wird beim Starten und Anhalten von *SCSLog* mit einem Shell-Skript unterstützt. Hier wird gezeigt, wie *SCSLog* gestartet, überprüft und gestoppt wird.

```
root@client # rcMICE-syscallmonitor start
M-ICE IDS: Starting SCSLog                               done
root@client # rcMICE-syscallmonitor status
scslog                2060576    2
root@client # rcMICE-syscallmonitor stop
M-ICE IDS: Stopping SCSLog                               done
```

Das Starten und Stoppen über *rcMICE-syscallmonitor* beinhaltet das Setzen und Entfernen der Logging-Funktionen via *scslog-ctrl*.

Beim Starten erscheint folg. Meldung in den *Syslog*-Dateien.

SCSLog - loaded (Thomas Biege <thomas@suse.de>)

SCSLog erzeugt Log-Zeilen (umgebrochen) in folgender Weise:

```
Sep 18 12:41:40 otaku.wlan-vpn.ashpool.org scslog2syslog[1442]:  
SCSLog - Syscall: socketcall |  
Program: dataforwarder | PID: 3676 | UID: 503 | EUID: 503 |  
Call: 0 = socketcall(3, bffffbd14) |  
Comment: connect to IP: 10.0.1.1 Port: 53 |
```

DataForwarder

Auch der *DataForwarder* kann einfach über ein rc-Skript gestartet und angehalten werden, zuvor jedoch müssen die dynamisch ladbaren Module an die richtige Stelle kopiert und die Konfigurationsdatei des *DataForwarders* angepasst werden.

Nachdem die Module in den Verzeichnissen *2.1_Filter-Module*, *2.2_LogFormat-Module* und *2.3_Pseudonymizer-Module* mit *make* erstellt wurden, müssen sie mit Hilfe von *libtool* kopiert werden. Hier ein Beispiel für *mice_mod_filter*:

```
root@client:~/M-ICE/2.1_Filter-Module/mice_mod_filter> \  
libtool cp mice_mod_filter.la /lib/M-ICE/  
cp .libs/mice_mod_filter.so /lib/M-ICE/mice_mod_filter.so
```

Entsprechend verfährt man für die anderen Module.

Anschließend muss noch der Pfad für die Module in die Konfigurationsdatei des *DataForwarders* (/etc/M-ICE/dataforwarder.conf) eingetragen werden.

```
[MODULES_SEARCH_PATH]
```

```
MODPATH    = /lib/M-ICE/
```

```
[MODULES]
```

```
MOD_FILTER    = mice_mod_filter
```

```
MOD_LOGFORMAT = mice_mod_logformat
```

```
MOD_PSEUDONYM = mice_mod_pseudonymizer
```

```
[MODULES_CONFIG_FILE]
```

```
FILTERCONF    = /etc/M-ICE/mice_mod_filter.conf
```

```
LOGFORMATCONF = /etc/M-ICE/mice_mod_logformat.conf
```

```
PSEUDONYMCONF = /etc/M-ICE/mice_mod_pseudonymizer.conf
```

Die drei untenstehenden Abschnitte der Konfigurationsdatei des *DataForwarders* geben an, wo die Daten zu finden sind und wohin sie geschickt werden sollen.

```
[SQL_SERVER]
```

```
SQLIP        = wintermute.ashpool.org
```

```
SQLPORT      = 3366
```

```
SQLPROTO     = TCP
```

```
[ANALYSIS_SERVER]
```

```
ASIP         = wintermute.ashpool.org
```

```
ASPORT       = 3367
```

```
ASPROTO      = TCP
```

```
[...]
```

```
[LOGFILE_LIST]
```

```
FILE         = /var/log/messages
```

```
FILE         = /var/log/xferlog
```

```
FILE         = /var/log/secure_server.log
```

```
FILE         = /var/log/ntp
```

```
FILE         = /var/log/faillog
```

```
FILE         = /var/log/http.access_log
```

```
FILE         = /var/log/http.error_log
```


ReactionDaemon

Als letztes wird der *ReactionDaemon* auf unserem Client installiert und konfiguriert, damit der Management-Host Gegenmaßnahmen einleiten kann.

Hierfür ist nicht viel nötig. Zunächst müssen alle Module, die später einer **Function ID** zugeordnet werden, auf dem Client installiert werden. Dies geschieht, ebenso wie bei den Modulen für den *DataForwarder*, mit **libtool**.

Folgende Dateien und Verzeichnisse werden angelegt:

- /var/run/M-ICE
- /var/log/M-ICE
- /lib/M-ICE
- /etc/M-ICE
- /usr/local/sbin/reactiondaemon
- /etc/M-ICE/reactiondaemon.conf
- /sbin/rcMICE-reactiondaemon
- /lib/M-ICE/mice_mod_rct_system.*

Am Ende muss die **Function ID** und das korrespondierende Modul noch in die Konfigurationsdatei eingetragen werden:

```
[FUNCTION_ID]  
FID = 0x000001
```

```
[REACTION_MODULES]  
RCT_MOD = mice_mod_rct_system
```

```
[REACTION_MODULES_CONFIG_FILE]  
RCT_FILE = /etc/M-ICE/mice_mod_rct_system.conf
```

3.3.2 Datenbank für rohe Log-Daten

Für den Aufbau dieser Datenbank wird der *BufferDaemon* und die Module `mice_mod_enc_logformat_twofish` und `mice_mod_mysql` verwendet.

Folgende Dateien und Verzeichnisse werden angelegt:

- `/var/run/M-ICE`
- `/var/log/M-ICE`
- `/lib/M-ICE`
- `/etc/M-ICE`
- `/usr/local/sbin/bufferdaemon`
- `/etc/M-ICE/bufferdaemon-rawlogdb.conf`
- `/sbin/rcMICE-bufferdaemon-rawlogdb`
- `/lib/M-ICE/mice_mod_mysql.*`
- `/lib/M-ICE/mice_mod_enc_logformat_twofish.*`

Tabellen erstellen

Unsere Datenbank enthält drei Tabellen. Eine Tabelle für *Syslog*, eine für *SCSLog* und eine für *SuSEfirewall2*. Die Tabellen werden wie folgt erstellt:

- Erstellen der Datenbank `mice_rawlog_tab`
- Tabellen generieren mit `mysql mice_rawlog_tab < create_mysql_tab_for_raw_logs.txt`
- Einen User `mice` einrichten
- INSERT- sowie SELECT-Operation für den Benutzer auf Datenbank `mice_rawlog_tab` erlauben

BufferDaemon

Zuerst wird die IP-Adresse und die Port-Nummer, an die der *BufferDaemon* lauschen soll, in die Konfigurationsdatei eingetragen.

```
[IP_ADDRESS]
IP  = 0.0.0.0
```

```
[PORT_NUMBERS]
PORT = 3366
```

Nachdem die Module installiert wurden, werden deren Namen in die Konfigurationsdatei geschrieben.

```
[MODULES_SEARCH_PATH]
MODPATH = /lib/M-ICE/
```

```
[ENCODING_MODULES]
ENC_MOD = mice_mod_enc_logformat_twofish
```

```
[ENCODING_MODULES_CONFIG_FILE]
ENC_FILE = /etc/M-ICE/mice_mod_enc_logformat_twofish.conf
```

```
[OUTPUT_MODULES]
OUT_MOD = mice_mod_mysql
```

```
[OUTPUT_MODULES_CONFIG_FILE]
OUT_FILE = /etc/M-ICE/mice_mod_mysql.conf
```

Danach wird der *BufferDaemon* mit Hilfe von *rcMICE-bufferdaemon* gestartet.

```
root@rawlog_db # rcMICE-bufferdaemon start
M-ICE IDS: Starting BufferDaemon                               done
```

Tabellen auslesen

Die SQL-Tabellen lassen sich mit einem SQL-Client abfragen. Dabei kann man nach Rechnernamen, Datum, IP-Adressen, Programmname, User ID, Syscall und dergleichen suchen lassen.

MySQL Navigator 1.3.8 - mice@127.0.0.1

File Edit Style MySQL Tools Help

mice_raw

select * from firewall_line where date>="2002-11-22";

Query - select * from rawlog_line where date>="2002-11-22";

File Tools Help

hostname	domain	ip	osystem	release	version	date	time	logline
<input type="checkbox"/>	ldoru	ashpool.org	172.16.0.40	Linux	2.4.19-4GB #1 Fri Sep 13 13:14	2002-11-22	15:36:58	Nov 22 15:36:58 ldoru scslog2syslog[264]: SCSLog - Syscall: socketcall Program: wget PID: 2272 UI
<input type="checkbox"/>	ldoru	ashpool.org	172.16.0.40	Linux	2.4.19-4GB #1 Fri Sep 13 13:14	2002-11-22	15:37:02	Nov 22 15:37:02 ldoru scslog2syslog[264]: SCSLog - Syscall: socketcall Program: wget PID: 2272 UI
<input type="checkbox"/>	gate	ashpool.org	172.16.0.1	Linux	2.4.19-4GB #1 Wed Oct 9 13:27	2002-11-22	15:37:47	Nov 22 15:37:47 gate kernel: SuSE-FW-DROP-DEFAULT IN=ippp0 OUT= MAC= SRC=145.254.95.186 DS
<input type="checkbox"/>	gate	ashpool.org	172.16.0.1	Linux	2.4.19-4GB #1 Wed Oct 9 13:27	2002-11-22	15:54:47	Nov 22 15:54:46 gate kernel: SuSE-FW-DROP-DEFAULT IN=ippp0 OUT= MAC= SRC=202.32.214.13 DS
<input type="checkbox"/>	gate	ashpool.org	172.16.0.1	Linux	2.4.19-4GB #1 Wed Oct 9 13:27	2002-11-22	16:09:59	Nov 22 16:09:57 gate kernel: SuSE-FW-DROP-DEFAULT IN=ippp0 OUT= MAC= SRC=148.245.58.69 DS
<input type="checkbox"/>	gate	ashpool.org	172.16.0.1	Linux	2.4.19-4GB #1 Wed Oct 9 13:27	2002-11-22	16:24:32	Nov 22 16:24:30 gate kernel: SuSE-FW-DROP-DEFAULT IN=ippp0 OUT= MAC= SRC=81.48.195.233 DS
<input type="checkbox"/>	gate	ashpool.org	172.16.0.1	Linux	2.4.19-4GB #1 Wed Oct 9 13:27	2002-11-22	16:31:31	Nov 22 16:31:30 gate kernel: SuSE-FW-DROP-DEFAULT IN=ippp0 OUT= MAC= SRC=145.254.95.186 DS
<input type="checkbox"/>	gate	ashpool.org	172.16.0.1	Linux	2.4.19-4GB #1 Wed Oct 9 13:27	2002-11-22	16:37:55	Nov 22 16:37:55 gate kernel: SuSE-FW-DROP-DEFAULT IN=ippp0 OUT= MAC= SRC=64.123.89.88 DS
<input type="checkbox"/>	gate	ashpool.org	172.16.0.1	Linux	2.4.19-4GB #1 Wed Oct 9 13:27	2002-11-22	16:38:01	Nov 22 16:38:01 gate kernel: SuSE-FW-DROP-DEFAULT IN=ippp0 OUT= MAC= SRC=145.254.97.80 DS

Fields: 10, Records: 1071

Query - select * from scslog_line where date>="2002-11-22";

File Tools Help

hostname	domain	ip	osystem	release	version	date	time	syscall	program	pid	uid	euid	call	comment
<input type="checkbox"/>	ldoru	ashpool.org	172.16.0.40	Linux	2.4.19-4GB #1 Fri Sep 13 13:14	2002-11-22	13:46:32	socketcall	wget	1609	0	0	0 = socketcall(3, bffff0f0)	connect to IP: 131.159.72.
<input type="checkbox"/>	ldoru	ashpool.org	172.16.0.40	Linux	2.4.19-4GB #1 Fri Sep 13 13:14	2002-11-22	13:46:36	socketcall	wget	1612	0	0	0 = socketcall(3, bffff0e0)	connect to IP: 131.159.72.
<input type="checkbox"/>	ldoru	ashpool.org	172.16.0.40	Linux	2.4.19-4GB #1 Fri Sep 13 13:14	2002-11-22	13:46:40	socketcall	wget	1612	0	0	0 = socketcall(3, bffff110)	connect to IP: 131.159.72.
<input type="checkbox"/>	ldoru	ashpool.org	172.16.0.40	Linux	2.4.19-4GB #1 Fri Sep 13 13:14	2002-11-22	13:46:44	socketcall	wget	1615	0	0	0 = socketcall(3, bffff0e0)	connect to IP: 131.159.72.
<input type="checkbox"/>	ldoru	ashpool.org	172.16.0.40	Linux	2.4.19-4GB #1 Fri Sep 13 13:14	2002-11-22	13:46:50	socketcall	wget	1615	0	0	0 = socketcall(3, bffff110)	connect to IP: 131.159.72.
<input type="checkbox"/>	ldoru	ashpool.org	172.16.0.40	Linux	2.4.19-4GB #1 Fri Sep 13 13:14	2002-11-22	13:46:52	socketcall	wget	1618	0	0	0 = socketcall(3, bffff0e0)	connect to IP: 131.159.72.
<input type="checkbox"/>	ldoru	ashpool.org	172.16.0.40	Linux	2.4.19-4GB #1 Fri Sep 13 13:14	2002-11-22	15:36:58	socketcall	wget	2272	0	0	0 = socketcall(3, bffff0b0)	connect to IP: 131.159.72.
<input type="checkbox"/>	ldoru	ashpool.org	172.16.0.40	Linux	2.4.19-4GB #1 Fri Sep 13 13:14	2002-11-22	15:37:02	socketcall	wget	2272	0	0	0 = socketcall(3, bffff0e0)	connect to IP: 131.159.72.

Fields: 15, Records: 193

Query - select * from firewall_line where date>="2002-11-22";

File Tools Help

action	if_in	if_out	mac	source	destination	ip_length	tos	prec	ttl	id	protocol	src_port	dst_port	pac_length	date	time	
<input type="checkbox"/>	DROP-DEFAULT	ippp0	NO VALUE	NO VALUE	64.123.89.88	145.254.96.202	78	0	0	108	9320	UDP	1029	137	78	2002-11-22	16:37:55
<input type="checkbox"/>	DROP-DEFAULT	ippp0	NO VALUE	NO VALUE	145.254.97.80	145.254.96.202	78	0	0	126	1300	UDP	1029	137	78	2002-11-22	16:38:01
<input type="checkbox"/>	DROP-DEFAULT	ippp0	NO VALUE	NO VALUE	145.254.95.186	145.254.96.202	78	0	0	127	59189	UDP	1027	137	78	2002-11-22	16:39:25
<input type="checkbox"/>	DROP-DEFAULT	ippp0	NO VALUE	NO VALUE	216.230.154.212	145.254.96.202	78	0	0	106	56652	UDP	1549	137	78	2002-11-22	16:39:53

Fields: 17, Records: 4

Abbildung 3.13: Alle drei Tabellen in der Übersicht

MySQL Navigator 1.3.6 - mice@127.0.0.1

File Edit Style MySQL Tools Help

mice_rav select * from rawlog_line;

Query - select * from rawlog_line;

File Tools Help

hostname	domain	ip	osystem	release	version	date	time	logfile
gate	ashpool.org	172.16.0.1	Linux	2.4.19-4GB #1 Wed Oct 9 13:27:	2002-12-04 19:05:02	Dec 4 19:05:00	gate /usr/sbin/cron[5602]: (root) CMD (/	
ldoru	ashpool.org	172.16.0.40	Linux	2.4.19-4GB #1 Fri Sep 13 13:14	2002-12-04 19:10:00	Dec 4 19:10:00	ldoru /usr/sbin/cron[2842]: (root) CMD (/	
gate	ashpool.org	172.16.0.1	Linux	2.4.19-4GB #1 Wed Oct 9 13:27:	2002-12-04 19:10:01	Dec 4 19:10:00	gate /usr/sbin/cron[5671]: (root) CMD (/	
gate	ashpool.org	172.16.0.1	Linux	2.4.19-4GB #1 Wed Oct 9 13:27:	2002-12-04 19:15:00	Dec 4 19:15:00	gate /usr/sbin/cron[5682]: (root) CMD (/	
ldoru	ashpool.org	172.16.0.40	Linux	2.4.19-4GB #1 Fri Sep 13 13:14	2002-12-04 19:15:01	Dec 4 19:15:00	ldoru /usr/sbin/cron[2853]: (root) CMD (/	
ldoru	ashpool.org	172.16.0.40	Linux	2.4.19-4GB #1 Fri Sep 13 13:14	2002-12-04 19:20:01	Dec 4 19:20:00	ldoru /usr/sbin/cron[2882]: (root) CMD (/	
gate	ashpool.org	172.16.0.1	Linux	2.4.19-4GB #1 Wed Oct 9 13:27:	2002-12-04 19:20:02	Dec 4 19:20:00	gate /usr/sbin/cron[5711]: (root) CMD (/	
gate	ashpool.org	172.16.0.1	Linux	2.4.19-4GB #1 Wed Oct 9 13:27:	2002-12-04 19:25:01	Dec 4 19:25:00	gate /usr/sbin/cron[5720]: (root) CMD (/	
ldoru	ashpool.org	172.16.0.40	Linux	2.4.19-4GB #1 Fri Sep 13 13:14	2002-12-04 19:25:02	Dec 4 19:25:00	ldoru /usr/sbin/cron[2891]: (root) CMD (/	
gate	ashpool.org	172.16.0.1	Linux	2.4.19-4GB #1 Wed Oct 9 13:27:	2002-12-04 19:30:01	Dec 4 19:30:00	gate /usr/sbin/cron[5732]: (root) CMD (/	
ldoru	ashpool.org	172.16.0.40	Linux	2.4.19-4GB #1 Fri Sep 13 13:14	2002-12-04 19:30:02	Dec 4 19:30:00	ldoru /usr/sbin/cron[2902]: (root) CMD (/	
ldoru	ashpool.org	172.16.0.40	Linux	2.4.19-4GB #1 Fri Sep 13 13:14	2002-12-04 19:35:01	Dec 4 19:35:00	ldoru /usr/sbin/cron[2931]: (root) CMD (/	
ldoru	ashpool.org	172.16.0.40	Linux	2.4.19-4GB #1 Fri Sep 13 13:14	2002-12-04 19:40:01	Dec 4 19:40:00	ldoru /usr/sbin/cron[2940]: (root) CMD (/	
gate	ashpool.org	172.16.0.1	Linux	2.4.19-4GB #1 Wed Oct 9 13:27:	2002-12-04 19:40:02	Dec 4 19:40:00	gate /usr/sbin/cron[5770]: (root) CMD (/	
ldoru	ashpool.org	172.16.0.40	Linux	2.4.19-4GB #1 Fri Sep 13 13:14	2002-12-04 19:45:00	Dec 4 19:45:00	ldoru /usr/sbin/cron[2951]: (root) CMD (/	
gate	ashpool.org	172.16.0.1	Linux	2.4.19-4GB #1 Wed Oct 9 13:27:	2002-12-04 19:45:01	Dec 4 19:45:00	gate /usr/sbin/cron[5781]: (root) CMD (/	
gate	ashpool.org	172.16.0.1	Linux	2.4.19-4GB #1 Wed Oct 9 13:27:	2002-12-04 19:50:01	Dec 4 19:50:00	gate /usr/sbin/cron[5810]: (root) CMD (/	
ldoru	ashpool.org	172.16.0.40	Linux	2.4.19-4GB #1 Fri Sep 13 13:14	2002-12-04 19:50:01	Dec 4 19:50:00	ldoru /usr/sbin/cron[2980]: (root) CMD (/	
ldoru	ashpool.org	172.16.0.40	Linux	2.4.19-4GB #1 Fri Sep 13 13:14	2002-12-04 19:55:01	Dec 4 19:55:00	ldoru /usr/sbin/cron[3007]: (root) CMD (/	
gate	ashpool.org	172.16.0.1	Linux	2.4.19-4GB #1 Wed Oct 9 13:27:	2002-12-04 19:55:02	Dec 4 19:55:00	gate /usr/sbin/cron[5819]: (root) CMD (/	
gate	ashpool.org	172.16.0.1	Linux	2.4.19-4GB #1 Wed Oct 9 13:27:	2002-12-04 19:57:49	Dec 4 19:57:48	gate kernel: SuSE-FW-DROP-DEFAULT IN=ip	
ldoru	ashpool.org	172.16.0.40	Linux	2.4.19-4GB #1 Fri Sep 13 13:14	2002-12-04 19:59:00	Dec 4 19:59:00	ldoru /usr/sbin/cron[3032]: (root) CMD (/	
gate	ashpool.org	172.16.0.1	Linux	2.4.19-4GB #1 Wed Oct 9 13:27:	2002-12-04 19:59:01	Dec 4 19:59:00	gate /usr/sbin/cron[5828]: (root) CMD (/	
ldoru	ashpool.org	172.16.0.40	Linux	2.4.19-4GB #1 Fri Sep 13 13:14	2002-12-04 20:00:00	Dec 4 20:00:00	ldoru /usr/sbin/cron[3038]: (root) CMD (/	
gate	ashpool.org	172.16.0.1	Linux	2.4.19-4GB #1 Wed Oct 9 13:27:	2002-12-04 20:00:02	Dec 4 20:00:00	gate /usr/sbin/cron[5832]: (root) CMD (/	
ldoru	ashpool.org	172.16.0.40	Linux	2.4.19-4GB #1 Fri Sep 13 13:14	2002-12-04 20:05:00	Dec 4 20:05:00	ldoru /usr/sbin/cron[3087]: (root) CMD (/	
gate	ashpool.org	172.16.0.1	Linux	2.4.19-4GB #1 Wed Oct 9 13:27:	2002-12-04 20:05:01	Dec 4 20:05:00	gate /usr/sbin/cron[5861]: (root) CMD (/	

Fields: 10, Records: 36333

Abbildung 3.14: Komplettes Auslesen der rawlog_line-Tabelle

MySQL Navigator 1.3.8 - mice@127.0.0.1

File Edit Style MySQL Tools Help

mice_rav select * from scslog_line where syscall="open";

Query - select * from scslog_line where syscall="open";

File Tools Help

hostname	domain	ip	osystem	release	version	date	time	syscall	program	pid	uid	euid	call
<input type="checkbox"/> otaku	wlan-vpn.ashpool.org	10.0.1.10	Linux	2.4.18-4GB	#7 Wed Jul 10 12:39	2002-08-17	17:15:34	open	sendmail	957	0	0	6 = open(/proc/loadavg, 0, 266)
<input type="checkbox"/> otaku	wlan-vpn.ashpool.org	10.0.1.10	Linux	2.4.18-4GB	#7 Wed Jul 10 12:39	2002-08-17	17:15:36	open	dataforwarder	336	503	503	7 = open(/etc/hosts, 0, 266)
<input type="checkbox"/> otaku	wlan-vpn.ashpool.org	10.0.1.10	Linux	2.4.18-4GB	#7 Wed Jul 10 12:39	2002-08-17	17:15:38	open	sendmail	957	0	0	6 = open(/proc/loadavg, 0, 266)
<input type="checkbox"/> otaku	wlan-vpn.ashpool.org	10.0.1.10	Linux	2.4.18-4GB	#7 Wed Jul 10 12:39	2002-08-17	17:15:40	open	dataforwarder	336	503	503	7 = open(/etc/hosts, 0, 266)
<input type="checkbox"/> otaku	wlan-vpn.ashpool.org	10.0.1.10	Linux	2.4.18-4GB	#7 Wed Jul 10 12:39	2002-08-17	17:15:44	open	sendmail	957	0	0	6 = open(/proc/loadavg, 0, 266)
<input type="checkbox"/> otaku	wlan-vpn.ashpool.org	10.0.1.10	Linux	2.4.18-4GB	#7 Wed Jul 10 12:39	2002-08-17	17:15:46	open	dataforwarder	336	503	503	7 = open(/etc/hosts, 0, 266)
<input type="checkbox"/> otaku	wlan-vpn.ashpool.org	10.0.1.10	Linux	2.4.18-4GB	#7 Wed Jul 10 12:39	2002-08-17	17:15:48	open	sendmail	957	0	0	6 = open(/proc/loadavg, 0, 266)
<input type="checkbox"/> otaku	wlan-vpn.ashpool.org	10.0.1.10	Linux	2.4.18-4GB	#7 Wed Jul 10 12:39	2002-08-17	17:15:50	open	kdeinit	1244	500	500	10 = open(/opt/kde3/share/icons/h
<input type="checkbox"/> otaku	wlan-vpn.ashpool.org	10.0.1.10	Linux	2.4.18-4GB	#7 Wed Jul 10 12:39	2002-08-17	17:15:54	open	scslog-ctrl	1486	0	0	3 = open(/dev/scslog, 2, 0)

Fields: 15, Records: 9

Abbildung 3.15: Auslesen der `scslog_line`-Tabelle für Syscall `open(2)`

3.3.3 Analyseeinheit

Das zweite Ziel der Audit-Daten ist die Analyseeinheit. Die Analyseeinheit ist ähnlich aufgebaut wie die Datenbank für die rohen Logs, lediglich das *Output Modul* für die Analyse ist ein anderes.

Folgende Dateien und Verzeichnisse werden angelegt:

- /var/run/M-ICE
- /var/log/M-ICE
- /lib/M-ICE
- /etc/M-ICE
- /usr/local/sbin/bufferdaemon
- /etc/M-ICE/bufferdaemon-aa.conf
- /sbin/rcMICE-bufferdaemon-aa
- /lib/M-ICE/mice_mod_aa_regex.*
- /lib/M-ICE/mice_mod_enc_logformat_twofish.*

Analysemodul

Das *Output Modul* `mice_mod_aa_regex` benutzt reguläre Ausdrücke, um die rohen Log-Daten, die vom Client generiert wurden, nach Angriffsmustern zu durchsuchen. Die Datei `match_file_aa_regex.txt` ist Teil des Modul `mice_mod_aa_regex` und muss später auf dem Management-Host in das Verzeichnis `/etc/M-ICE/matchfiles/` kopiert werden. Sie dient dazu, einer `AlertID` eine Beschreibung zuzuordnen.

```
[ALERT_ID]
AID = 0x1010    # RT_AUTH_S
AID = 0x1020    # RT_AUTH_F
AID = 0x2010    # RT_ROOT_W
AID = 0x2020    # RT_ROOT_O
AID = 0x2030    # RT_ROOT_R
AID = 0x2040    # RT_ROOT_S
AID = 0x2050    # RT_ROOT_E
AID = 0x3010    # RT_READ_S
AID = 0x3020    # RT_READ_F
AID = 0x4010    # RT_WRITE_S
AID = 0x4020    # RT_WRITE_F
AID = 0x5010    # RT_MONI_U
AID = 0x5020    # RT_MONI_G
```

```
AID = 0x6010    # RT_APPS_N
AID = 0x7010    # RT_EXPL
AID = 0x8010    # RT_FW_D
AID = 0x8020    # RT_FW_R
AID = 0x8030    # RT_FW_A
AID = 0x8040    # RT_FW_I
```

```
[ALERT_ID_DESCRIPTION]
AID_DESC = "Authentication Success"
AID_DESC = "Authentication Failure"
AID_DESC = "Superuser writes to File"
AID_DESC = "Superuser opens File"
AID_DESC = "Superuser reads from File"
AID_DESC = "Superuser does Operation on Socket"
AID_DESC = "Superuser executes Code"
AID_DESC = "Read Success"
AID_DESC = "Read Failure"
AID_DESC = "Write Success"
AID_DESC = "Write Failure"
AID_DESC = "Monitoring UserID"
AID_DESC = "Monitoring GroupID"
AID_DESC = "Monitoring Application"
AID_DESC = "Exploit Signature detected"
AID_DESC = "Firewall Rule: Drop"
AID_DESC = "Firewall Rule: Reject"
AID_DESC = "Firewall Rule: Accept"
AID_DESC = "Firewall Rule: Illegal"
```

Die AlertID wird in der IDMEF-Nachricht in der Klasse `Classification` des Elements `name` kodiert und wird vom Management-Host ausgewertet. Desweiteren muss angegeben werden, welche Reaktion bei welcher AlertID ausgeführt wird.

```
# Write to Syslog
[RID_1]
AID_1 = 0x1010
AID_1 = 0x1020
```



```

AID_1 = 0x2010

# Send to Alert DB
[RID_2]
AID_2 = 0x5010
AID_2 = 0x5020
AID_2 = 0x6010
AID_2 = 0x7010
AID_2 = 0x8010
AID_2 = 0x8020
AID_2 = 0x8030
AID_2 = 0x8040

# Save to File
[RID_3]
AID_3 = 0x1010
AID_3 = 0x1020
AID_3 = 0x4020
AID_3 = 0x5010

# Countermeasure
[RID_4]
AID_4 = 0x6010
AID_4 = 0x7010

# EMPTY
[RID_5]
[RID_6]
[RID_7]
[RID_8]
[RID_9]
[RID_10]

```

Die Reaction ID bezieht sich direkt auf den *Reaction Process* `rid_<nummer>_<name>`. In dieser Zuordnung gibt man für eine Reaktion alle **AlertIDs** an, auf die reagiert werden soll. Beispielsweise wird die **Reaction ID 4** (Countermeasure) bei **AlertID 0x6010** (Monitoring Applications) und 0x7010 (Exploit Patterns) ausgeführt.

Die Regeln für die Analyse befinden sich in der Datei mice_mod_aa_regex.conf im Verzeichnis /etc/M-ICE/. Sie sind in Kategorien, wie Authentication Messages, Root Actions, Syscall: read, Syscall: write, Monitoring UID, GID, Monitoring Application und Exploit Patterns, unterteilt.

```
# Authentication Messages
[AUTH]
AuSendTo  = "M"
AuS_Rule  = ".*session started for user.*"
AuF_Rule  = ".*FAILED SU.*"
AuF_Rule  = ".*Failed password.*"
AuF_Rule  = ".*Failed rsa.*"
AuF_Rule  = ".*ROOT LOGIN REFUSED FROM.*"

# root Actions
[ROOT]
RoSendTo  = "N"
RoW_Rule  = ""
RoR_Rule  = ""
RoO_Rule  = ""
RoS_Rule  = ""
RoE_Rule  = ""

# read(2)
[READ]
ReSendTo  = "N"
ReS_Rule  = ""
ReF_Rule  = ""

# write(2)
[WRITE]
WrSendTo  = "N"
WrS_Rule  = ""
WrF_Rule  = ""

# Monitor special UID/GID
[MONITORING]
MoSendTo  = "N"
Mo_UID    = ""
```

```

Mo_GID      = ""

# Special Applications
[APPS]
ApSendTo    = "M"
Ap_Name     = ".*sshd.*"

# Exploit Patterns
[EXPLOIT]
ExSendTo    = "M"
Ex_Rule     = ".*crc32 compensation attack: network attack detected.*"
Ex_Rule     = ".*GET /scripts/root.exe?/c+dir.*"
Ex_Rule     = ".*RECV: 163129 bytes of data, 0 total lines.*"
Ex_Rule     = ".*REMOTE TSI ?%.*"

# Firewall Patterns
[FIREWALL]
FwSendTo    = "M"
FwD_Rule    = ".*SuSE-FW-DROP.*"
FwR_Rule    = ""
FwA_Rule    = ".*SuSE-FW-ACCEPT.*"
FwI_Rule    = ".*SuSE-FW-ILLEGAL.*"

```

3.3.4 Management-Host

Der Management-Host wird ebenfalls mit Hilfe des *BufferDaemons* implementiert. Hier dient nun aber das Modul `mice_mod_enc_idmef_twofish` als *Encoding Modul* und `mice_mod_act_generic` als *Output Modul*.

Folgende Dateien und Verzeichnisse werden angelegt:

- /var/run/M-ICE
- /var/log/M-ICE
- /lib/M-ICE
- /etc/M-ICE
- /usr/local/sbin/bufferdaemon
- /etc/M-ICE/bufferdaemon-mngmthost.conf

- /sbin/rcMICE-bufferdaemon-mngmthost
- /lib/M-ICE/mice_mod_aa_regex.*
- /lib/M-ICE/mice_mod_enc_idmef_twofish.*
- /etc/M-ICE/rid_1_write_to_syslog.conf
- /usr/local/sbin/rid_1_write_to_syslog
- /sbin/rcMICE-rid_1_write_to_syslog
- /etc/M-ICE/rid_2_send_to_alert_db.conf
- /usr/local/sbin/rid_2_send_to_alert_db
- /sbin/rcMICE-rid_2_send_to_alert_db
- /etc/M-ICE/rid_3_save_to_file.conf
- /usr/local/sbin/rid_3_save_to_file
- /sbin/rcMICE-rid_3_save_to_file
- /etc/M-ICE/rid_4_countermeasure.conf
- /usr/local/sbin/rid_4_countermeasure
- /sbin/rcMICE-rid_4_countermeasure

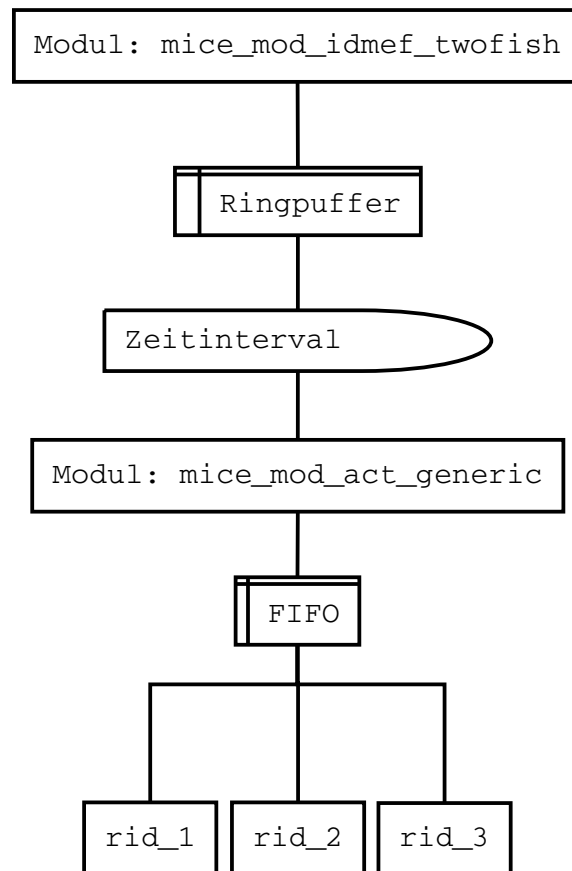


Abbildung 3.16: Datenflussdiagramm des Management-Hosts

Act-Generic Modul

Bei der Entwicklung dieses Moduls wurde darauf geachtet, dass es nicht nur auf IDMEF-Alarme des RegEx-Modul reagieren kann, sondern auch auf Analyseeinheiten anderer Systeme. Diese Funktionalität wird durch die sog. *Match Files* erreicht. Die *Analyzer ID*, die in der IDMEF-Nachricht kodiert ist, wird dazu benutzt, das richtige *Match File* ausfindig zu machen. Die Zuordnung findet in der Konfigurationsdatei von `mice_mod_act_generic.conf` statt.

```
# Every Analyzer must send an ID...
[ANALYZER_ID]
```

```
ANALYZER_MATCHFILE]
MF = match_file_aa_regex.txt
```

ANAID = "MICE-RegEx"

```
# ... to select the right file for matching Alert ID -> Reaction ID
```

In der selben Datei wird auch der Zusammenhang von Reaction ID und FIFO hergestellt. An jedem FIFO lauscht ein *Reaction Process*. Zur besseren Übersicht trägt der FIFO den selben Namen wie der *Reaction Process*.

```
# List of Reaction IDs
```

```
[REACTION_ID]
```

```
RID = 1
```

```
RID = 2
```

```
RID = 3
```

```
RID = 4
```

```
# and associated named pipes
```

```
[RID_PIPENAME]
```

```
PIPE = rid_1_write_to_syslog
```

```
PIPE = rid_2_send_to_alert_db
```

```
PIPE = rid_3_save_to_file
```

```
PIPE = rid_4_countermeasure
```

Reaktionsprozesse

Durch die flexible Architektur des Management-Hosts lässt sich jeder denkbare *Reaction Process* implementieren. Starten über rc-Skript:

```
root@management host# rcMICE-rid_1_write_to_syslog start
M-ICE IDS: Starting Reaction Service 'rid_1_write_to_syslog'           done
root@management host# rcMICE-rid_2_send_to_alert_db start
M-ICE IDS: Starting Reaction Service 'rid_2_send_to_alert_db'         done
root@management host# rcMICE-rid_3_save_to_file start
M-ICE IDS: Starting Reaction Service 'rid_3_save_to_file'             done
root@management host# rcMICE-rid_4_countermeasure start
M-ICE IDS: Starting Reaction Service 'rid_4_countermeasure'           done
```

Datenbank für Alarme

Die MySQL-Datenbank für Alarme läuft in unserem Fall direkt auf dem Management-Host und wird vom *Reaction Process* `rid_2_send_to_alert_db` versorgt.

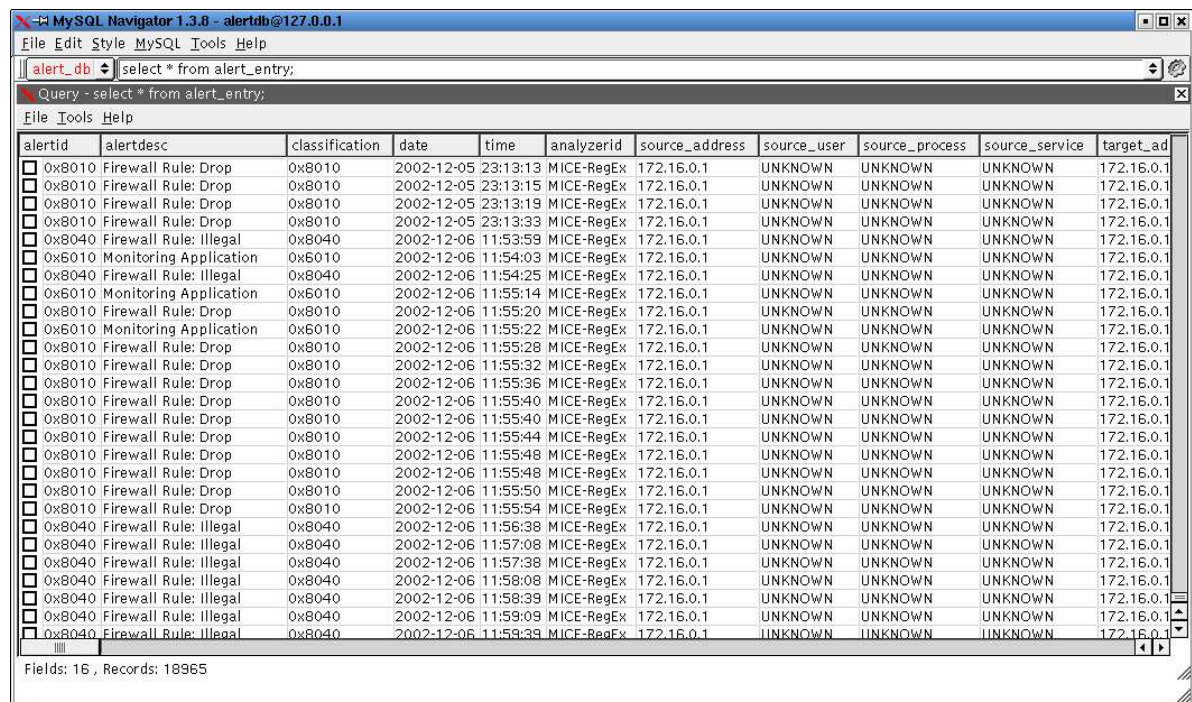
Tabellen erstellen

Diese Tabelle enthält alle nötigen Informationen aus der IDMEF-Nachricht. Um die Tabelle zu erstellen sind folgende Schritte nötig:

- Erstellen der Datenbank `alert_db`
- Tabellen generieren mit `mysql mice_rawlog_tab < create_mysql_tab_for_alert_logs.txt`
- User `alertdb` einrichten
- INSERT- sowie SELECT-Operation für den Benutzer auf Datenbank `alert_db` erlauben

Tabellen auslesen

Die Alarm-Datenbank kann nun mit einem MySQL Client ausgelesen werden.



alertid	alertdesc	classification	date	time	analyzerid	source_address	source_user	source_process	source_service	target_ad
0x8010	Firewall Rule: Drop	0x8010	2002-12-05	23:13:13	MICE-RegEx	172.16.0.1	UNKNOWN	UNKNOWN	UNKNOWN	172.16.0.1
0x8010	Firewall Rule: Drop	0x8010	2002-12-05	23:13:15	MICE-RegEx	172.16.0.1	UNKNOWN	UNKNOWN	UNKNOWN	172.16.0.1
0x8010	Firewall Rule: Drop	0x8010	2002-12-05	23:13:19	MICE-RegEx	172.16.0.1	UNKNOWN	UNKNOWN	UNKNOWN	172.16.0.1
0x8010	Firewall Rule: Drop	0x8010	2002-12-05	23:13:33	MICE-RegEx	172.16.0.1	UNKNOWN	UNKNOWN	UNKNOWN	172.16.0.1
0x8040	Firewall Rule: Illegal	0x8040	2002-12-06	11:53:59	MICE-RegEx	172.16.0.1	UNKNOWN	UNKNOWN	UNKNOWN	172.16.0.1
0x6010	Monitoring Application	0x6010	2002-12-06	11:54:03	MICE-RegEx	172.16.0.1	UNKNOWN	UNKNOWN	UNKNOWN	172.16.0.1
0x8040	Firewall Rule: Illegal	0x8040	2002-12-06	11:54:25	MICE-RegEx	172.16.0.1	UNKNOWN	UNKNOWN	UNKNOWN	172.16.0.1
0x6010	Monitoring Application	0x6010	2002-12-06	11:55:14	MICE-RegEx	172.16.0.1	UNKNOWN	UNKNOWN	UNKNOWN	172.16.0.1
0x8010	Firewall Rule: Drop	0x8010	2002-12-06	11:55:20	MICE-RegEx	172.16.0.1	UNKNOWN	UNKNOWN	UNKNOWN	172.16.0.1
0x6010	Monitoring Application	0x6010	2002-12-06	11:55:22	MICE-RegEx	172.16.0.1	UNKNOWN	UNKNOWN	UNKNOWN	172.16.0.1
0x8010	Firewall Rule: Drop	0x8010	2002-12-06	11:55:28	MICE-RegEx	172.16.0.1	UNKNOWN	UNKNOWN	UNKNOWN	172.16.0.1
0x8010	Firewall Rule: Drop	0x8010	2002-12-06	11:55:32	MICE-RegEx	172.16.0.1	UNKNOWN	UNKNOWN	UNKNOWN	172.16.0.1
0x8010	Firewall Rule: Drop	0x8010	2002-12-06	11:55:36	MICE-RegEx	172.16.0.1	UNKNOWN	UNKNOWN	UNKNOWN	172.16.0.1
0x8010	Firewall Rule: Drop	0x8010	2002-12-06	11:55:40	MICE-RegEx	172.16.0.1	UNKNOWN	UNKNOWN	UNKNOWN	172.16.0.1
0x8010	Firewall Rule: Drop	0x8010	2002-12-06	11:55:40	MICE-RegEx	172.16.0.1	UNKNOWN	UNKNOWN	UNKNOWN	172.16.0.1
0x8010	Firewall Rule: Drop	0x8010	2002-12-06	11:55:44	MICE-RegEx	172.16.0.1	UNKNOWN	UNKNOWN	UNKNOWN	172.16.0.1
0x8010	Firewall Rule: Drop	0x8010	2002-12-06	11:55:48	MICE-RegEx	172.16.0.1	UNKNOWN	UNKNOWN	UNKNOWN	172.16.0.1
0x8010	Firewall Rule: Drop	0x8010	2002-12-06	11:55:48	MICE-RegEx	172.16.0.1	UNKNOWN	UNKNOWN	UNKNOWN	172.16.0.1
0x8010	Firewall Rule: Drop	0x8010	2002-12-06	11:55:50	MICE-RegEx	172.16.0.1	UNKNOWN	UNKNOWN	UNKNOWN	172.16.0.1
0x8010	Firewall Rule: Drop	0x8010	2002-12-06	11:55:54	MICE-RegEx	172.16.0.1	UNKNOWN	UNKNOWN	UNKNOWN	172.16.0.1
0x8040	Firewall Rule: Illegal	0x8040	2002-12-06	11:56:38	MICE-RegEx	172.16.0.1	UNKNOWN	UNKNOWN	UNKNOWN	172.16.0.1
0x8040	Firewall Rule: Illegal	0x8040	2002-12-06	11:57:08	MICE-RegEx	172.16.0.1	UNKNOWN	UNKNOWN	UNKNOWN	172.16.0.1
0x8040	Firewall Rule: Illegal	0x8040	2002-12-06	11:57:38	MICE-RegEx	172.16.0.1	UNKNOWN	UNKNOWN	UNKNOWN	172.16.0.1
0x8040	Firewall Rule: Illegal	0x8040	2002-12-06	11:58:08	MICE-RegEx	172.16.0.1	UNKNOWN	UNKNOWN	UNKNOWN	172.16.0.1
0x8040	Firewall Rule: Illegal	0x8040	2002-12-06	11:58:39	MICE-RegEx	172.16.0.1	UNKNOWN	UNKNOWN	UNKNOWN	172.16.0.1
0x8040	Firewall Rule: Illegal	0x8040	2002-12-06	11:59:09	MICE-RegEx	172.16.0.1	UNKNOWN	UNKNOWN	UNKNOWN	172.16.0.1
0x8040	Firewall Rule: Illegal	0x8040	2002-12-06	11:59:39	MICE-RegEx	172.16.0.1	UNKNOWN	UNKNOWN	UNKNOWN	172.16.0.1

Fields: 16 , Records: 18965

Abbildung 3.17: Komplettes Auslesen der alert_entry Tabelle

MySQL Navigator 1.3.8 - alertdb@127.0.0.1

File Edit Style MySQL Tools Help

alert_db | select * from alert_entry where alertdesc="Authentication Failure";

Query - select * from alert_entry where alertdesc="Authentication Failure";

File Tools Help

alertid	alertdesc	classification	date	time	analyzerid	source_address	source_user	source_process	source_service	target_address
0x1020	Authentication Failure	0x1020	2002-07-25	20:38:06	MICE-RegEx	10.0.1.10	UNKNOWN	UNKNOWN	UNKNOWN	10.0.1.10
0x1020	Authentication Failure	0x1020	2002-07-25	20:38:20	MICE-RegEx	10.0.1.10	UNKNOWN	UNKNOWN	UNKNOWN	10.0.1.10
0x1020	Authentication Failure	0x1020	2002-07-25	20:38:32	MICE-RegEx	10.0.1.10	UNKNOWN	UNKNOWN	UNKNOWN	10.0.1.10
0x1020	Authentication Failure	0x1020	2002-07-25	20:38:44	MICE-RegEx	10.0.1.10	UNKNOWN	UNKNOWN	UNKNOWN	10.0.1.10
0x1020	Authentication Failure	0x1020	2002-07-25	20:38:57	MICE-RegEx	10.0.1.10	UNKNOWN	UNKNOWN	UNKNOWN	10.0.1.10
0x1020	Authentication Failure	0x1020	2002-07-25	20:39:09	MICE-RegEx	10.0.1.10	UNKNOWN	UNKNOWN	UNKNOWN	10.0.1.10
0x1020	Authentication Failure	0x1020	2002-07-25	20:39:23	MICE-RegEx	10.0.1.10	UNKNOWN	UNKNOWN	UNKNOWN	10.0.1.10
0x1020	Authentication Failure	0x1020	2002-07-25	20:39:35	MICE-RegEx	10.0.1.10	UNKNOWN	UNKNOWN	UNKNOWN	10.0.1.10
0x1020	Authentication Failure	0x1020	2002-07-26	20:51:20	MICE-RegEx	10.0.1.10	UNKNOWN	UNKNOWN	UNKNOWN	10.0.1.10
0x1020	Authentication Failure	0x1020	2002-07-26	20:52:00	MICE-RegEx	10.0.1.10	UNKNOWN	UNKNOWN	UNKNOWN	10.0.1.10
0x1020	Authentication Failure	0x1020	2002-07-26	20:53:00	MICE-RegEx	10.0.1.10	UNKNOWN	UNKNOWN	UNKNOWN	10.0.1.10
0x1020	Authentication Failure	0x1020	2002-07-26	20:53:58	MICE-RegEx	10.0.1.10	UNKNOWN	UNKNOWN	UNKNOWN	10.0.1.10
0x1020	Authentication Failure	0x1020	2002-08-15	18:44:57	MICE-RegEx	10.0.1.10	UNKNOWN	UNKNOWN	UNKNOWN	10.0.1.10
0x1020	Authentication Failure	0x1020	2002-08-15	18:51:47	MICE-RegEx	10.0.1.10	UNKNOWN	UNKNOWN	UNKNOWN	10.0.1.10
0x1020	Authentication Failure	0x1020	2002-08-15	18:53:25	MICE-RegEx	10.0.1.10	UNKNOWN	UNKNOWN	UNKNOWN	10.0.1.10
0x1020	Authentication Failure	0x1020	2002-08-16	12:32:02	MICE-RegEx	10.0.1.10	UNKNOWN	UNKNOWN	UNKNOWN	10.0.1.10
0x1020	Authentication Failure	0x1020	2002-08-17	11:50:16	MICE-RegEx	217.160.107.174	UNKNOWN	UNKNOWN	UNKNOWN	217.160.107.1
0x1020	Authentication Failure	0x1020	2002-08-17	17:14:02	MICE-RegEx	10.0.1.10	UNKNOWN	UNKNOWN	UNKNOWN	10.0.1.10
0x1020	Authentication Failure	0x1020	2002-08-18	13:45:12	MICE-RegEx	10.0.0.40	UNKNOWN	UNKNOWN	UNKNOWN	10.0.0.40
0x1020	Authentication Failure	0x1020	2002-08-20	17:40:13	MICE-RegEx	217.160.107.174	UNKNOWN	UNKNOWN	UNKNOWN	217.160.107.1
0x1020	Authentication Failure	0x1020	2002-08-20	18:05:21	MICE-RegEx	217.160.107.174	UNKNOWN	UNKNOWN	UNKNOWN	217.160.107.1
0x1020	Authentication Failure	0x1020	2002-08-20	18:05:27	MICE-RegEx	217.160.107.174	UNKNOWN	UNKNOWN	UNKNOWN	217.160.107.1
0x1020	Authentication Failure	0x1020	2002-08-27	15:43:24	MICE-RegEx	217.160.107.174	UNKNOWN	UNKNOWN	UNKNOWN	217.160.107.1
0x1020	Authentication Failure	0x1020	2002-11-17	22:52:17	MICE-RegEx	172.16.0.40	UNKNOWN	UNKNOWN	UNKNOWN	172.16.0.40
0x1020	Authentication Failure	0x1020	2002-11-18	15:14:12	MICE-RegEx	172.16.0.40	UNKNOWN	UNKNOWN	UNKNOWN	172.16.0.40
0x1020	Authentication Failure	0x1020	2002-11-18	18:15:40	MICE-RegEx	172.16.0.1	UNKNOWN	UNKNOWN	UNKNOWN	172.16.0.1
0x1020	Authentication Failure	0x1020	2002-11-27	16:02:13	MICE-RegEx	172.16.0.1	UNKNOWN	UNKNOWN	UNKNOWN	172.16.0.1

Fields: 16, Records: 11728

Abbildung 3.18: Auslesen der alert_entry Tabelle für alle fehlgeschlagenen Authentifizierungen

Kapitel 4

Testergebnisse

M-ICE wurde einigen Tests unterzogen, bei denen mehr auf Fehlerbehebung als auf Performanz Rücksicht genommen wurde. Der Grund dafür, ist die Tatsache, dass der Flaschenhals eines solchen Systems beim Datentransport und bei der Datenanalyse liegt. Der Datentransport ließe sich neben breitbandigen Netzen und schnelleren Computersystemen, vor allem durch bessere Aufteilung der IDS-Komponenten beschleunigen. Zudem ist die Datenanalyse, ausser zu Testzwecken, nicht Bestandteil der Entwicklung gewesen; ein Test würde also keinen Sinn machen. Trotzdem sind natürlich einige Ergebnisse zu Tage getreten.

Als Testumgebung diente folgender Aufbau.

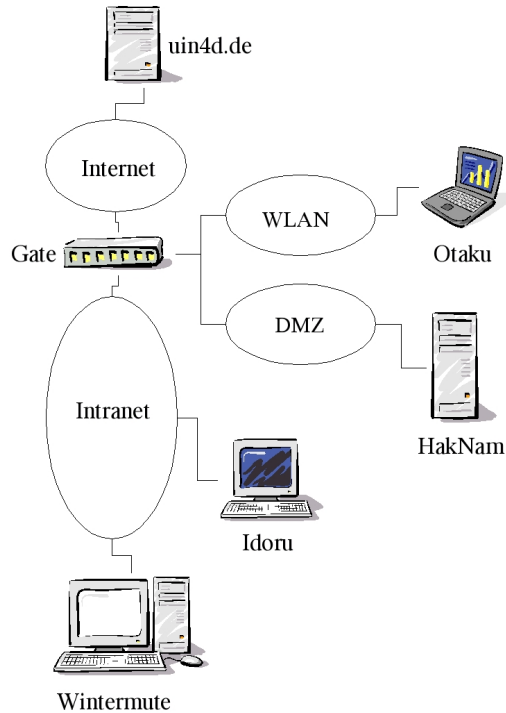


Abbildung 4.1: Aufbau des Testnetzwerkes

Bei dieser Testumgebung lief die Analyseeinheit, die Datenbanken und der Management-Host auf dem Rechner WINTERMUTE. Als Client dienten fünf Rechner (IDORU, OTAKU, HAKNAM, GATE, WWW.UIN4D.DE) in verschiedenen Netzwerken. Folgende Tabelle zeigt die Hardware- und Softwareausstattung der einzelnen Rechner:

Name	CPU	RAM	Netzanbindung	Betriebssystem
WINTERMUTE	2 x 700 MHz	1 GB	100 Mbit (Intranet)	SuSE Linux 8.0
IDORU	733 MHz	128 MB	100 Mbit (Intranet)	SuSE Linux 8.0
OTAKU	700 MHz	256 MB	<= 10 Mbit (WLAN)	SuSE Linux 8.0
HAKNAM	266 MHz	128 MB	10 Mbit (DMZ)	SuSE Linux 7.0
GATE (Router)	266 MHz	128 MB	10/100 Mbit / 64 Kbit	SuSE Linux 8.1
WWW.UIN4D.DE	1,2 GHz	256 MB	64 Kbit (Internet)	SuSE Linux 7.2

Die meisten Daten werden vom *DataForwarder* generiert, da er den Datenstrom dupliziert und u.U. noch ins gleiche Netzsegment sendet. Um hier einen möglichen Engpass zu vermeiden, könnte man den *DataForwarder* so einrichten, dass er die Audit-Daten nur an die Datenbank sendet, aber nicht an die Analyseeinheit. Das *Encoding Modul* des Analysesystems müsste sich nun in gewissen Zeitintervallen die Audit-Daten von der Datenbank holen. Somit gehen die Daten zwar immernoch zweimal über das Netzwerk, jedoch nicht zum selben Zeitpunkt. Desweiteren könnte man die Datenbank und die Analyseeinheit auf dem selben Rechner installieren und so noch mehr Bandbreite einsparen. Ein zusätzlicher Vorteil für die Analyse würde sich aus der Tatsache ergeben, dass die rohen Log-Daten bereits in logische Teile zerlegt wurden, was die Analyse vereinfacht. Nachteil ist die Abhängigkeit von der Datenbank und die Aufteilung der Hardware-Ressourcen.

Glücklicherweise tauchten diese theoretischen Probleme nicht in der Praxis auf. Selbst der Rechner `WWW.UIN4D.DE`, welcher nur über eine ISDN-Leitung angebunden war, und auf dem ein öffentlicher Web- und Gameserver läuft und durchschnittlich drei Benutzer gleichzeitig arbeiten, hat die Bandbreite nicht saturieren können.

Die meisten Audit-Daten wurden vom Paketfilter des Routers GATE erzeugt. Durch die ständigen *Scans* nach offenen SMB Ports entstanden eine Menge uninteressanter Daten. Innerhalb von 7,5 Stunden wurden vom Paketfilter 392 Pakete abgewiesen, 332 davon waren UDP Pakete mit dem Zielport 137 (SMB). Unser IDS ist in der Lage diese ungefährlichen Angriffsversuche bereits auf dem Client auszufiltern. Dank des Filter-Moduls im *DataForwarder* und der entsprechenden Filterregel (`.*PROTO=UDP.*DPT=137.*`) konnte das Datenvolumen stark reduziert werden. In diesem Fall sind es sogar ca. 80 % weniger Daten, die gespeichert und analysiert werden müssen.

Die Rechner GATE und IDORU generieren innerhalb von 15 Stunden etwa 840 Einträge in der `rawlog_line`-Tabelle. Nach der Analyse befinden sich ca. 475 Einträge in der `alert_entry`-Tabelle. Davon sind 466 Einträge vom Typ `Firewall Rule: Drop` und die restlichen von Typ `Firewall Rule: Illegal`, `Authentication Success` und `Monitoring Applikation`. Hier sieht man, wie selbst die einfache Analyse durch reguläre Ausdrücke, das Datenaufkommen um knapp 50 % reduzieren kann.

Die maximale Größe eines Client-Paketes beträgt 1710 Byte. GATE sendet in 15 Stunden je 760 Pakete an die Datenbank für rohe Log-Daten und an die Analyseeinheit, das sind weniger als 170 KByte pro Stunde. Datenmengen dieser Dimension fallen bei heutigen Vernetzungstechnologien sogar gar nicht ins Gewicht. In einem 100 MBit Netzwerk, mit einer angenommenen realen Transportgeschwindigkeit von 8 MByte/s, würde man ungefähr 170.000 Rechner, mit einem Audit-Datenvolumen, vergleichsweise mit dem von GATE, benötigen, um das Netzwerk voll auszulasten.

Diese Zahlen machen sehr deutlich, dass der befürchtete Engpass nicht existiert.

Der Anstieg der Rechnerauslastung war ebenfalls kaum merkbar, selbst nicht auf WINTERMUTE, der immerhin fast alle IDS-Komponenten beherbergte.

Der *DataForwarder* belegte ca. 0,6 % des Speichers auf dem Client-System GATE. Dieser Wert wird hauptsächlich durch die Größe des *Shared Memory Segments* bestimmt. Zum Zeitpunkt der Datenweiterleitung wurde eine Prozessorauslastung von 0,3 % gemessen.

Im Fall der Analyseeinheit benötigt der *BufferDaemon* im Leerlauf 0,2 % des Speichers und 0,0 % der Prozessorleistung. Während der Datenanalyse stieg lediglich die Prozessorauslastung auf 0,1 %.

Ein weiterer wichtiger Faktor ist die Reaktionsgeschwindigkeit. Als Reaktion wird auf dem Client (GATE) eine Datei im Verzeichnis `/tmp` angelegt. Um die Geschwindigkeit der Verarbeitung bis zur Reaktion messen zu können, wurde ein kleines Programm entwickelt, das zwei Threads startet. Ein Thread löst eine Aktion aus und protokolliert die Startzeit. Die Aktion veranlasst den Management-Host (WINTERMUTE) dazu, eine Reaktion auf dem Client-System auszuführen. Der zweite Thread prüft wann die Reaktion auftritt und misst ebenfalls die Zeit. Der Haupt-Thread berechnet dann die Differenz der beiden Zeitwerte und gibt diese als Ergebnis aus. Es wurden mehrere Messungen durchgeführt. Die gemessene Reaktionszeit lag zwischen 51 und 71 Sekunden. Anschließend wurde nacheinander die Priorität (*Nice Value*: -10) der folgenden Prozesse erhöht:

- ReactionDaemon
- DataForwarder
- Analysis-Agent
- Management-Host
- Syslog-Daemon (Client-System)

Erstaunlicherweise brachte die Prioritätenerhöhung keine Beschleunigung der Reaktion. In angesicht der Tatsache, dass das Netzwerk keinen Engpass darstellt, läßt das Ergebnis nur den Schluss zu, dass die Operationen von M-ICE sich durch Prioritätenerhöhung nicht beschleunigen lassen, oder dass die Verarbeitungszeit der M-ICE-Komponenten nicht ausschlaggebend ist.

Kapitel 5

Schlussbetrachtung

Abschließend kann man sagen, dass alle nötigen Eigenschaften eines hostbasier-ten Intrusion-Detection Systems implementiert werden konnten. M-ICE bietet die Möglichkeit, Audit-Daten zu generieren, zu klassifizieren, zu speichern und entsprechende Gegenreaktionen auszulösen. Leider konnte aufgrund von Zeitmangel die Heartbeat-Komponente und die kryptografische Signatur für die Datenbankeinträge nicht realisiert werden.

Die vorhandene, modulare Architektur erlaubt es, mit wenig Aufwand, ein sehr leistungsfähiges und professionelles IDS zu bauen. Die hier, nur zu Testzwecken, implementierte Analyseeinheit auf Basis von regulären Ausdrücken, ließe sich leicht durch die auf Zustandsautomaten aufbauende, freie Analyse-Engine *STAT* der Universität von Californien, Santa Barbara [34] ersetzen. Zudem bietet die UCSD bereits vielfältige Muster für die Angriffserkennung an. Diese Angriffsmuster nutzen bspw. das *Common Logging Format* des Apache Webserver oder den Log-Output von Solaris' *Basic Security Module* (BSM). Für letzteres Format gibt es eine, leider nicht mehr weiterentwickelte, Implementierung für Linux, so dass man die BSM-Angriffsmuster für *SCSLog* umschreiben müsste.

M-ICE soll in Zukunft weiterentwickelt werden, da es in der Testphase vielversprechende Ergebnisse geliefert hat und die Entwicklung sehr lehrreich war.

Anhang A

Terminologie und Abkürzungen

BSM Basic Security Module

CERT Computer Emergency Response Team

DMZ Demilitarisierte Zone, bspw. sicheres Netz zw. Internet und internem Netz

DoS Denial-of-Service

FIFO First In, First Out; Named Pipe; lokale Interprozesskommunikation

GNU GNU's Not Unix!, Projekt der *Free Software Foundation* zur Entwicklung von freien Unix-Tools

GUI Graphical User Interface

IAP Intrusion Alert Protocol

IDXP Intrusion Detection Exchange Format

IDMEF Intrusion Detection Message Exchange Format

IDS Intrusion Detection System

IP Internet Protocol, s. RFC-791 [37]

LKM Loadable Kernel Modul

M-ICE Modular Intrusion Detection and Countermeasure Environment

NTP Network Time Protocol, s. RFC-1119 [37]

PGP Pretty Good Privacy, eMail Verschlüsselungsprogramm

PKI Public Key Infrastructure

RegEx Regular Expressions, reguläre Ausdrücke

RPC Remote Procedure Call

SMB Server Message Block

SNMP Simple Network Management Protocol

SO Security Officer

SQL Structured Query Language

SSL Secure Socket Layer, Verschlüsselung auf Präsentationsebene

Syslog natives Unix Logsystem

TCP Transmission Control Protocol, s. RFC-793 [37]

UDP User Datagram Protocol, s. RFC-768 [37]

UML Unified Modeling Language

VPN Virtual Private Network

VM Virtual Machine

XML Extensible Markup Language

Anhang B

Abbildungsverzeichnis

1.1	CIDF Modell mit E-, D-, A- und C-Box	4
1.2	AAFID-Modell	11
1.3	Zustandsautomat	14
3.1	UML Klasse	29
3.2	UML Vererbung	29
3.3	UML Gruppierung	30
3.4	IDMEF-Aufbau	31
3.5	Detaillierter Aufbau von M-ICE	33
3.6	Abstrakter Aufbau von M-ICE	34
3.7	SCSLog Architektur	35
3.8	DataForwarder Architektur	42
3.9	Datenflussdiagramm des DataForwarders	45
3.10	Datenflussdiagramm des BufferDaemons	50
3.11	Datenflussdiagramm des ReactionProcess'	51
3.12	Datenflussdiagramm des ReactionDaemons	54
3.13	Alle drei Tabellen	77
3.14	Komplettes Auslesen der rawlog_line-Tabelle	78
3.15	Auslesen der scslog_line-Tabelle für Syscall open(2)	78
3.16	Datenflussdiagramm des Management-Hosts	86
3.17	Komplettes Auslesen der alert_entry Tabelle	88
3.18	Auslesen der alert_entry Tabelle für alle fehlgeschlagenen Authentifizierungen	89
4.1	Aufbau des Testnetzwerkes	91

Anhang C

Literaturverzeichnis

- [1] D. Curry, H. Debar, Intrusion Detection Message Exchange Format — Data Model and Extensible Markup Language (XML) Document Type Definition, IDWG, February 2002
- [2] Gupta, Buchheim, Feinstein, Harvey Mudd College, Matthews, Pollock, IAP: Intrusion Alert Protocol, IDWG, March 2001
- [3] Feinstein, Matthews, White, Harvey Mudd College, MITRE Corporation, The Intrusion Detection Exchange Protocol (IDXP), IDWG, May 2001
- [4] Hofmeyr, Forrest, Somayaji, Intrusion Detetction using Sequences of System Calls, August 1998
- [5] Biskup, Flegel, On Pseudonymization of Audit Data for Intrusion–Detection, July 2000
- [6] Sobirey, Fischer–Hübner, Rannenber, Pseudonymous Audit for Privacy Enhanced Intrusion–Detection, 1997
- [7] Flegel, Pseudonymizing Unix Log Files
- [8] von Helden, Grundlagen, Forderungen und Marktübersicht für Intrusion Detection Systeme (IDS) und Intrusion Response Systeme (IRS), debis IT Security Service, Studie für das *Bundesamt für Sicherheit in der Informationstechnik* (BSI), 1998
- [9] Rebecca Gurley Bace, Intrusion–Detection, MTP, 2000

- [10] Paul E. Proctor, The Practical Intrusion Detection Handbook, Prentice Hall, 2001
- [11] Bruce Schneier, Angewandte Kryptographie, Addison–Wesley, 1996
- [12] CERT Statistiken: http://www.cert.org/stats/cert_stats.html
- [13] Common Intrusion Detection Framework (CIDF), <http://www.gidos.org>
- [14] Matt Bishop, Standard Audit Trail Format, 1995 National Information Systems Security Conference, pages 136–145
- [15] Common Intrusion Specification Language (CISL), <http://www.gidos.org/drafts/language.txt>
- [16] Next Generation IDS (NIDS), <http://www.sdl.sri.com/nides/index.html>
- [17] AID, <http://www-rnks.informatik.tu-cottbus.de/de/security/aid.html>
- [18] Specter, <http://www.specter.ch/>
- [19] Snort, <http://www.snort.org>
- [20] Packet Monster, <http://www.inas.mag.keio.ac.jp/ids/pakemon/>
- [21] Prelude, <http://prelude-ids.org/>
- [22] Deception ToolKit, <http://www.all.net/dtk/>
- [23] Autonomous Agents for Intrusion Detection (AAFID), <http://www.cerias.purdue.edu/coast/projects/aafid.html>
- [24] Emerald, <http://www.sdl.sri.com/emerald/>
- [25] The Honeynets Project, <http://project.honeynet.org>
- [26] CLIPS, <http://www.unet.univie.ac.at/~a9715573/res/clips.html>
- [27] Psionic Software, <http://www.psionic.com/>
- [28] IDWG, <http://www.silicondefense.com/idwg/>
- [29] Thomas H. Ptacek und Timothy N. Newsham, Inseration, Evasion and Denial of Service: Eluding Newtork Intrusion–Detection, <http://www.robertgraham.com/mirror/Ptacek-Newsham-Evasion-98.html>

- [30] MIT Kerberos, <http://web.mit.edu/kerberos/www/>
- [31] RM Needham, MD Schrieder, “Using Encryption for Authentication in Large Networks of Computers“, in *Communications of the ACM*, v21 no 12 (Dec 1978), pp 993–999
- [32] Libtool, <http://www.gnu.org/software/libtool/>
- [33] LibIDMEF, <http://www.silicondefense.com/idwg/libidmef/index.htm>
- [34] STAT, <http://www.cs.ucsb.edu/~rsg/STAT/>
- [35] LibIAP, <http://snortnet.scorpions.net/>
- [36] LibMCrypt, <http://mccrypt.hellug.gr>
- [37] RFC Datenbank, <http://www.rfc-editor.org/>
- [38] RSA Security, <http://www.rsasecurity.com/>
- [39] LibIDXP, <http://idxp.codefactory.se/>

Anhang D

Erklärung gemäß §26 (1) DPO

Hiermit erkläre ich, dass die Diplomarbeit von mir selbstständig verfasst und angefertigt wurde, nur die angegebenen Quellen und Hilfsmittel benutzt wurden und Zitate kenntlich gemacht wurden.

Unterschrift (Thomas Biege)