Universität Hamburg
Fachbereich Informatik
Arbeitsbereich AGN

in Zusammenarbeit mit der Firma
Netlife Internet Consulting und Software GmbH

Diplomarbeit:

Sicherheitsmanagement für UNIX-Server am Modell eines verteilten Unternehmensnetzwerks

Autor:
Christoph Haas
(info@christoph-haas.de)

Vorgelegt zur Begutachtung bei:
Prof. Klaus Brunnstein
Dr. Hans-Joachim Mück
Ralph-Thomas Außem (Netlife GmbH)

Zusammenfassung

Die vorliegende Diplomarbeit befasst sich mit Sicherheitsbetrachtungen von UNIX-Servern in verteilten Netzwerken (weltweit mehrere Niederlassungen) im unternehmerischen Bereich. Im ersten Teil werden exemplarisch die Möglichkeiten und die Durchführung eines systematischen Angriffs auf das Unternehmsnetzwerk aufgezeigt. Im zweiten Teil wird eine zentralisierte Instanz zur Sicherheitsüberwachung entworfen und implementiert.

Abstract

This diploma thesis discusses security issues in UNIX networks in distributed company networks (several subsidiaries spread worldwide). The first part shows the possibilities and the process of attacking these networks systematically. The second part describes the design and implementation of a centralized security information system.

Erklärung

Die Nennung von Produkt- und Markennamen erfolgt aus informellen Gründen und beabsichtigt keine Verletzung von Rechten der Inhaber.

Dank der Unterstützung der Abteilung Service & Support der Firma Netlife Internet Consulting und Software GmbH konnten viele Aspekte eines heterogenen verteilten Netzwerks und der damit verbundenen Sicherheitsprobleme sehr praxisnah betrachtet werden. Die Details des in dieser Arbeit vorgestellten Modellunternehmens stehen allerdings nicht direkt im Zusammenhang mit der Systemtopologie der Firma Netlife, sondern stellen ein abstraktes Modell dar.

Der Quelltext der im Verlauf dieser Arbeit erarbeiteten Implementation einer Sicherheitsüberwachung wird in Auszügen in dieser Arbeit dargestellt. Der gesamte Quelltext wurde den Betreuern zur Begutachtung vorgelegt, stellt jedoch keinen Teil dieser Arbeit dar. Die gesamte Ausarbeitung und Dokumentation ist urherberrechtlich geschützt. Alle Rechte vorbehalten.

Diese Arbeit wurde nach den Regeln der 1995 in Kraft getretenen neuen deutschen Rechtschreibung verfasst. Ich habe diese Arbeit selbstständig erstellt und ausschließlich die angegebenen Hilfsmittel und Quellen und die während während meiner beruflichen Tätigkeiten und meines Studiums erworbenen Kenntnisse und Erfahrungen genutzt.

Hamburg, den 29. August 2001

Inhaltsverzeichnis

1. Einleitung

Motivation

Das Internet entstand aus dem Anfang der 70er Jahre als Forschungsprojekt begonnenen Projekt ARPANET. Die Entwicklung zum heute größten Datennetz der Welt war damals nicht vorhersehbar. Da das ARPANET als geschlossenes Netzwerk geplant war, wurde dem Faktor Sicherheit bei der Konzeption ein vergleichsweise geringer Stellenwert eingeräumt. Ein wichtiges Entwurfskriterium war eine hohe Verfügbarkeit. Aus Angst vor feindlichen EMP-Angriffen (*EMP=electromagnetic pulse*) sollten wichtige Datenbestände im Netzwerk mehrfach gehalten werden. Würden einige Systeme ausfallen, wären die Informationen immer noch auf einem anderen System verfügbar. Doch gerade heute, wo das Internet für Finanztransaktionen oder zur Übertragung geheimer Informationen verwendet wird, ist eine gesicherte Kommunikation mindestens so wichtig wie eine hohe Verfügbarkeit. Deshalb werden große Anstrengungen unternommen, das Internet nachträglich sicherer zu machen. Eine absolute Sicherheit kann auf diese Weise allerdings nicht erreicht werden.

Diese Arbeit gliedert sich in zwei Teile. Im ersten Teil wird ein systematischer Einbruch in ein heterogenes verteiltes Unternehmensnetzwerk beschrieben, um verbreitete Schwachpunkte aufzuzeigen. Gleichzeitig werden Möglichkeiten aufgezeigt, die damit verbundenen Risiken durch adäquate Systemkonfigurationen zu minimieren und damit die passive Sicherheit zu erhöhen. Die Verbesserung der aktiven Sicherheit ist das zentrale Thema des zweiten Teils, in dem ein zentrales Sicherheitsüberwachungssystem entworfen und implementiert wird, um aktuelle sicherheitsrelevante Ereignisse in einem großen heterogenen Netzwerk aufzuspüren und zu melden.

Mit dieser Arbeit soll verdeutlicht werden, welche essenzielle Bedeutung das Thema Netzwerksicherheit für Unternehmen hat, die eine direkte Verbindung zum Internet betreiben. Häufig stehen für Anwender nur die Verlässlichkeit und Verfügbarkeit im Vordergrund, nicht aber weitere Aspekte der Sicherheit. Diese Arbeit betrachtet die Möglichkeiten und Gefahren von Angriffen auf Netzwerke und zeigt ein Konzept auf, mittels eines implementierten Sicherheits-Informationssystems ein verteiltes Netzwerk zu überwachen.

Aufbau dieser Arbeit

1. In diesem ersten Kapitel wird die Motivation zur Erstellung dieser Arbeit dargestellt.
2. Das zweite Kapitel legt die Sicherheitsanforderungen fest, die im allgemeinen an IT-Systeme gestellt werden. Außerdem werden Grundbegriffe der Netzwerktechnik eingeführt, auf die in späteren Kapiteln aufgebaut wird. Zuletzt wird die Struktur des betrachteten Modellunternehmens erläutert.
3. Das dritte Kapitel widmet sich internetbasierten Angriffen. Dabei wird beispielhaft dargestellt, wie ein Angreifer das Modellunternehmen angreifen könnte. In jedem Schritt wird dabei erläutert, inwiefern diese Angriffsart erkannt werden kann und wie man Angriffe durch eine adäquate Systemkonfiguration erschweren bzw. a priori verhindern kann.
4. Im vierten Kapitel wird der Grundstein für die spätere Implementation eines Sicherheitsüberwachungssystems gelegt. Hier werden die Anforderungen festgelegt und daraus eine Konzeption abgeleitet.
5. Die Implementation des Systems ist das Thema des fünften Kapitels. Dort werden Details der Implementation erläutert und an Beispielen die Bedienung der Benutzerschnittstellen gezeigt.
6. Im sechsten Kapitel werden die erarbeiteten Ergebnisse in Bezug auf die Anforderungen aus dem vierten Kapitel bewertet und ein Ausblick gegeben, welche Punkte der Implementation noch erweitert werden könnten.

1.1. Danksagung

Ich möchte an dieser Stelle einigen Personen besonders danken, die diese Arbeit erleichtert bzw. erst möglich gemacht haben:

Prof. Klaus Brunnstein und Dr. Hans-Joachim Mück für die Betreuung der Arbeit,

Ralph-Thomas Aussem für die Themenidee und seine Hilfe bei der Strukturierung, wo jegliche Struktur abhanden zu kommen schien,

dem DFN-CERT für die exzellente Vorlesung „Sicherheit in vernetzten Systemen“, die auch großen Einfluss auf den ersten Teil dieser Arbeit hatte,

der Firma Netlife für die Möglichkeit, die Implementation in einem großen Netzwerk zu testen,

Dr. Michael Sievers für seine Tipps und Fehlerkorrekturen,

Leslie Lamport für das TEX-Makropaket *LATEX*,

Markus Kohm and Frank Neukam für ihren LATEX-Stil *scrbook*, in dem diese Arbeit gesetzt wurde,

Larry Wall für die Skriptsprache *PERL*,

Bram Moolenaar für den Editor, *VIM* mit dem diese Arbeit ediert wurde und nicht zuletzt

meiner Freundin Pamela die mir beim Korrekturlesen half und viele Abende im Kampf mit LaTeX geduldig ertragen hat.

2. Grundlagen

Inhaltsangabe

Im weiteren Verlauf dieser Arbeit werden viele Fachbegriffe verwendet. Dieses Kapitel soll diese Begriffe als technische Grundlagen einführen und auch als Glossar dienen.

2.1. Sicherheitsanforderungen

Die Sicherheitsanforderungen an ein IT-System werden durch die Sicherheits-Policy vorgegeben. Nach der Definition des Orange-Book (den „Trusted Computer Systems Evaluation Criteria“— TCSEC) ist es „eine Sammlung von Gesetzen, Regeln und Praktiken, welche regeln, wie eine Organisation sensitive Informationen verwaltet, schützt und verteilt“(Übersetzung von [scsmfm]). Das klassische regelbasierte Modell ist das Bell-LaPadula-Sicherheitsmodell. Es basiert auf den Sicherheitsanforderungen der Task-Force des amerikanischen Verteidigungsministeriums von 1967 und wurde von den Autoren David Elliot Bell und Leonhard J. LaPadula in mehreren Versionen in den Jahren 1973 bis 1976 veröffentlicht. Allerdings betrachtet es ausschließlich den Aspekt der Vertraulichkeit. Weitere Kriterien wie die Integrität oder die Verfügbarkeit, die heutzutage für den kommerziellen Bereich grundlegend sind, waren noch nicht Teil des Modells. Beim Bell-LaPadula-Modell werden alle Elemente in Sicherheitsstufen klassifiziert. Es wird dann verhindert, dass Informationen von einer höheren zu einer niedrigeren Sicherheitsstufe fließen (*No-Read-Up/No-Write-Down*).

Modernere Sicherheitsmodelle fordern weitere Aspekte wie die Verlässlichkeit, die Zurechenbarkeit, die Verbindlichkeit, die Nichtabstreitbarkeit, die informationelle Selbstbestimmung, die Anonymität und die Pseudonymität. Zur Vereinheitlichung der national unterschiedlichen Sicherheits-Bemessungsmaßstäbe wurden aus mehreren Modellen die *Common Criteria* (vgl. [comcrit]) entwickelt. Diese Kriterien bilden einen Maßstab zur Evaluation der Sicherheit von IT-Systemen.

Diese Arbeit betrachtet nur ausgewählte Aspekte dieser Sicherheitmodelle. Speziell sind dies:

Vertraulichkeit: Die Forderung, Daten nur entsprechend autorisierten Personen zugänglich zu machen, wird mit dem Faktor *Vertraulichkeit* bezeichnet. Dieser Schutz bezieht sich sowohl auf fest gespeicherte Daten (wie auf einer Festplatte) als auch auf Daten während einer Übertragung (z.B. Kommunikationsprotokolle im Internet).

Integrität: Der Schutz von Informationen vor nicht autorisierter Veränderung wird durch die *Integrität* gegeben.

Verfügbarkeit: Die ungestörte Funktion von Betriebsmitteln und Diensten zur Erfüllung einer definierten Aufgabe fällt in den Bereich der *Verfügbarkeit*.

2.2. Begriffsbestimmung

2.2.1. Internet

Die Wurzeln des Internet reichen zurück bis ins Jahr 1957, als das US-amerikanische Verteidigungsministerium die ARPA (*Advanced Research Projects Agency)* gründete, um im Kalten Krieg durch intensivere Forschung einen technologischen Vorsprung gegenüber der UdSSR zu erreichen. Es existierten zu dieser Zeit bereits Netzwerke, die allerdings sehr empfindlich waren, da bereits beim Ausfall nur eines Netzknotens die Kommunikation nicht mehr möglich war. Um auch nach einem feindlichen Nuklearschlag noch kommunizieren zu können, erhielt die ARPA den Auftrag, ein Netzwerk zu entwickeln, das weniger störanfällig sein sollte. Mit dem ARPANET sollte das erste paketvermittelnde Netzwerk entstehen, das 1969 in Betrieb genommen wurde. 1971 entstanden dann die ersten Protokolle Telnet und FTP, mit denen es ermöglicht wurde, auf entfernte Systeme zuzugreifen. Die internationale Verbreitung begann, nachdem im Jahr 1973 das *Transmission Control Protocol* (TCP) als gemeinsames Protokoll eines internationalen Internet akzeptiert wurde. Seitdem expandiert das Internet weltweit und hat heute eine geschätzte Zahl von 500 Mio. Benutzern und 100 Mio. verbundenen Rechensystemen.

2.2.2. Internet Protocol (IP)

Entsprechend der Netzwerkschicht im OSI-Modell[1] bildet die *Internetschicht* die Grundlage der Kommunikation auf Übertragungswegen im Internet. Die wichtigsten

[1] (siehe [kerner])

Protokolle dieser Schicht sind das *Internet Protocol (IP)*, das *Address Resolution Protocol (ARP)* und das *Internet Control Message Protocol (ICMP)*.

Das *Internet Protocol (IP)* ist ein verbindungsloses Übertragungsprotokoll, das lediglich für die ungesicherte Zustellung von Datenpaketen zu einem bestimmten System zuständig ist. Die Sicherung dieses Protokolls liegt in der Verantwortung der höheren Transportprotokolle.

Netzwerkkomponenten und Systeme werden über 32 Bit lange IP-Adressen adressiert, die zur besseren Lesbarkeit in je 4 Bytes aufgeteilt werden (z.B. `134.28.240.5`).

ICMP

Das *Internet Control Message Protocol (ICMP)* dient der Mitteilung von Fehlern in der Zustellung von Datenpaketen und der Überprüfung und Beeinflussung (*redirect*) von Netzwerk-Routen. Die UNIX-Befehle `ping` und `traceroute` basieren auf den Nachrichten *ICMP Echo-Request* und *ICMP Echo-Reply*, um das Routing zu einem bestimmten Zielsystem zu überprüfen und en detail zu verfolgen.

Die Aufgaben der OSI-Transport- und Sitzungsschicht übernehmen das verbindungsorientierte *Transmission Control Protocol (TCP)* und das verbindungslose *User Datagram Protocol (UDP)*.

TCP

Das *Transmission Control Protocol (TCP)* ist die Basis für die meisten Anwendungsprotokolle im Internet. Es arbeitet verbindungsorientiert und bietet Aspekte wie eine automatische Fehlerkorrektur oder eine Paketpufferung. Zu den verbreitetsten Anwendungen des TCP zählen Dienste wie das *Hypertext Transfer Protocol (HTTP)* zum Anfordern von Hypertextdokumenten, das *File Transfer Protocol (FTP)* zum Dateitransfer und das *Simple Mail Transfer Protocol (SMTP)* zur Übertragung von E-Mails.

Jedem TCP-basierten Dienst ist ein TCP-Port zugewiesen, über den entsprechende Prozesse die Protokollanfragen beantworten. Portnummern zwischen 1 und 1023 sind entsprechend der *Internet Assigned Numbers Authority (IANA)*[2] festgelegt und können vereinbarungsgemäß nur von Diensten belegt werden, die Administrator-Rechte auf einem System haben. Ports mit Nummern ab 1024 unterliegen diesen Beschränkungen nicht und können beliebig von Benutzerprozessen auch mit geringeren Zugriffsrechten verwendet werden.

Der Aufbau einer TCP-Verbindung erfolgt mit einem *Three-Way Handshaking* (s. Abb. 2.1). Im ersten Schritt sendet der Absender ein Datenpaket mit der initialen Sequenz-Nummer und der Fenstergröße der folgenden Datenpakete an das Zielsystem (SYN-Paket). Das Zielsystem bestätigt dieses Datenpaket mit seiner eigenen Sequenznummer an das Absendersystem (SYN-ACK-Paket). Schließlich bestätigt das Absendersystem noch einmal die Sequenznummer des Zielsystems (ACK-Paket).

[2] vgl. [iana]

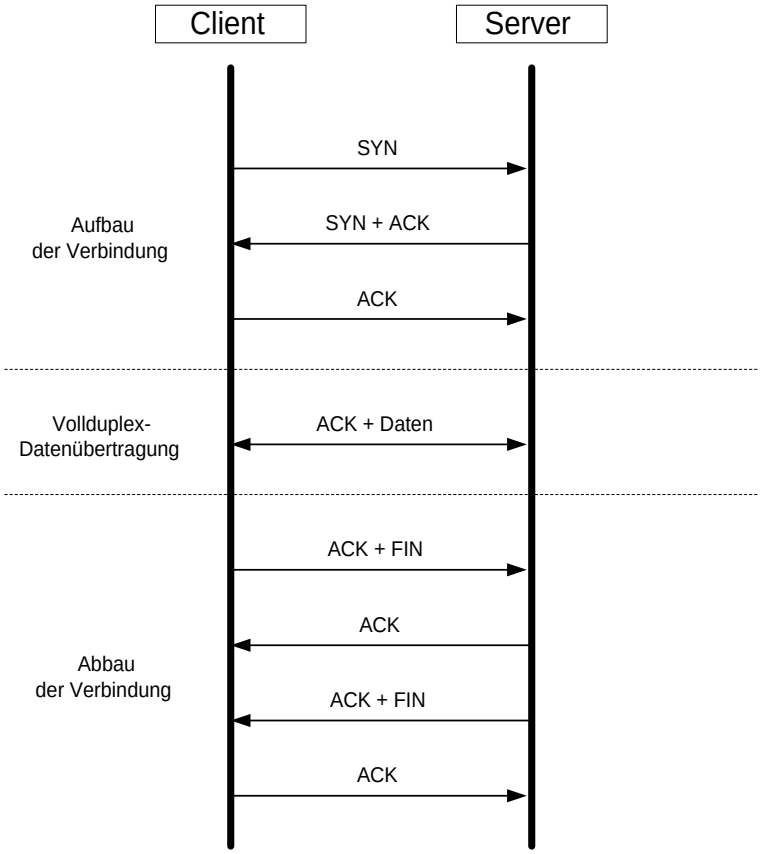

Abbildung 2.1.: Aufbau, Nutzung und Abbau einer TCP-Verbindung

UDP

Das *User Datagram Protocol (UDP)* wird für schnelle Datenübertragungen benötigt, da es verbindungslos und ungesichert arbeitet. Weder der Empfang noch die korrekte Paketreihenfolge werden garantiert. Namensdienste wie das *Network File System* (NFS), der *Domain Name Service* (DNS) oder das zur Systemüberwachung verwendete *Simple Network Management Protocol* (SNMP) werden mittels UDP realisiert. Auch UDP-basierte Protokolle arbeiten mit eindeutigen Portnummern, die von der IANA (s.o.) festgelegt werden. Das UDP wird hauptsächlich in lokalen Netzwerken (LANs) verwendet, da dort die Wahrscheinlichkeit für Fehler geringer ist als bei Datenübertragungen in Weitverkehrsnetzen (WANs).

2.2.3. IP-basierte Anwendungsprotokolle

SNMP (Simple Network Management Protocol)

Das *Simple Network Management Protocol (SNMP)* ist ein UDP-basiertes Protokoll zum Netzwerkmanagement. Viele Netzwerk-Komponenten bieten die Abfrage und Verände-

rung von Systemparametern über SNMP an. Außerdem können Komponenten auch selbstständig über SNMP-Traps Fehler und Zustandsänderungen an vorgegebene Überwachungssysteme melden.

DNS (Domain Name Service)

Ein weiterer UDP-basierter Dienst ist der *Domain Name Service* (DNS). Seine Hauptaufgabe ist die Umsetzung von Rechnernamen in IP-Adressen. Darüberhinaus speichert ein *Domain Name Server* auch weitere namensbezogene Informationen wie z.B. die Zielsysteme für die Zustellung von E-Mails mittels SMTP.

ARP (Address Resolution Protocol)

Das *Address Resolution Protocol (ARP)* wird verwendet, um innerhalb eines lokalen Netzwerks IP-Datenpakete einem Netzwerkendgerät (z.B. einer Netzwerkkarte) zustellen zu können. Netzwerkkarten werden eindeutig über eine MAC-Adresse (*Media Access Control*) adressiert. Analog zum *Domain Name Service*, der Rechnernamen in IP-Adressen umsetzt, findet das ARP die gesuchte Netzwerkadresse (*MAC-Adresse*) zu einer benötigten IP-Adresse. Anders als beim DNS unterhält jedes System eine ARP-Tabelle. Es gibt also keinen dedizierten Server für den ARP-Dienst. Wird eine Hardwareadresse benötigt, fragt das suchende Systeme im gesamten Netzwerksegment mittels Broadcast nach dieser Information.

HTTP (Hypertext Transfer Protocol)

Eine der häufigsten Internet-Anwendungen im kommerziellen Bereich ist das World-Wide-Web. Elementares Protokoll dieses Dienstes ist das HTTP (*Hypertext Transfer Protocol*). Es dient der Übertragung von Hypermedia-Inhalten auf TCP-Basis und wurde als RFC2068 beschrieben

TFTP (Trvial File Transfer Protocol)

TFTP ist ein einfaches UDP-Protokoll zur Dateiübertragung. Im Gegensatz zu anderen Transferprotokollen (wie FTP) verfügt es über keine Authentisierungsmechanismen. Es wird vorwiegend beim Bootvorgang von plattenlosen Systemen (sog. *diskless workstations*) oder zur Übertragung von Konfigurationsdateien zwischen Netzwerkkomponenten verwendet. TFTP wurde als RFC1350 beschrieben.

2.2.4. Virtual Private Network (VPN)

Ein virtuell-privates Netzwerk (*virtual private network*) beschreibt einen logischen Verbund aus mehreren Rechnern oder lokalen Netzwerken, der ein öffentliches unsicheres Datennetz zum Aufbau sicherer Verbindungen zwischen den Systemen bzw. Netzwerken benutzt. Meistens wird das Internet als öffentliches Netz genutzt, denn man hat von den meisten Orten einen kostengünstigen Zugang. Die angenommene Topologie für das Modellunternehmen basiert auf VPNs und wird in Abbildung 2.5 auf Seite

21 illustriert. Zur Absicherung des Datenverkehrs werden zusätzliche Protokolle in der Sicherungsschicht wie z.B. IPsec genutzt. Auf IP-Protokollebene bleibt die VPN-Kommunikation damit für die Anwendungen transparent.

2.2.5. Firewall

Eine Firewall ist eine Instanz, die als Übergangspunkt zwischen zwei Netzwerken eine Zugriffskontrolle der übertragenen Datenpakete vornimmt. Obwohl es auch wichtig ist, jedes einzelne System möglichst gut abzusichern, sollte man die Bemühungen zur Erhöhung der Sicherheit auf die Firewall konzentrieren. Es gibt mehrere Firewall-Architekturen, die im folgenden kurz skizziert werden sollen. Alle Firewall-Konzepte haben aber eine gemeinsame Schwäche, denn sie betrachten das interne Netzwerk als sicher. Insider-Angriffe können so nicht verhindert werden. Kommerzielle Firewallsysteme wie Checkpoint (s. [checkp]) unterstützen mehrere der folgenden Konzepte, um mittels *Application Level Firewalls* eine höchstmögliche Sicherheit am Netzübergang zu bieten.

Packet Screen

Packet Screens (Paketfilter) sind die einfachste Realisierung regelbasierter Filter. Als Informationen für die Entscheidung, ob ein Paket geroutet oder verworfen werden soll, dienen lediglich die IP-Quelladresse, die IP-Zieladresse und die Quell- und Zielports der angeforderten Verbindung. Zusätzlich werden noch Flags wie das sogenannte *ACK-Bit* überprüft, mit dem festgestellt werden kann, ob die Verbindung bereits besteht und deshalb ein Antwortpaket zurückgeschickt wird. Mehr Informationen stehen dem Paketfilter nicht zur Verfügung, so dass hier nur eine kontextfreie Filterung durchgeführt werden kann. So kann lediglich eine relativ geringe Sicherheit erreicht werden, da schädliche Inhalte über zugelassene Ports unbemerkt getunnelt werden können — eine Überprüfung der verwendeten Protokolle auf einem Port kann nicht vorgenommen werden. Selbst preisgünstige IP/ISDN-Router unterstützen aber mittlerweile diese einfachen Filterfunktionen.

Proxy-Server

Häufig sind nicht nur Serverdienste angreifbar sondern auch Clientdienste. Bei allgemein als ungefährlich betrachteten Diensten wie dem HTTP können durch maliziöse Inhalte der angezeigten Webseiten Schwächen in der Browserimplementation zur Kompromittierung des Clientsystems ausgenutzt werden. Ein Paketfilter kann hier keinen Schutz bieten. Als Lösung bieten sich Proxy-Server an. Clients im LAN greifen nicht mehr direkt auf Dienste im Internet zu, sondern verwenden einen speziellen Server, der die Anfragen auf Protokollbasis verstehen kann und diese an das gewünschte Ziel weiterleitet (vgl. Abb. 2.2 als Beispiel eines HTTP-Proxys). Der Zielserver kann so keine Unsicherheiten in der Client-Version ausnutzen, da die Anfrage scheinbar vom Proxy-Server stammt. So lassen sich keine Protokolle über unübliche Ports leiten, da der Proxy-Server nur bestimmte Protokolle interpretieren kann. Nachteilig an dieser Architektur ist, dass die Client-Systeme proxyfähig sein müssen und dass für jedes erforderliche Protokoll ein Proxy-Server benötigt wird.

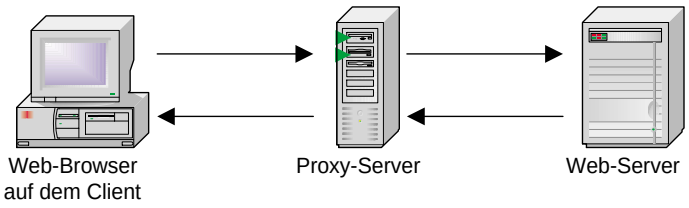

Abbildung 2.2.: Kommunikation über einen HTTP-Proxy

Screened Subnets (DMZ)

Eine weitere Möglichkeit der Absicherung ist der Einsatz von *Screened Subnets* — auch *demilitarisierte Zonen* genannt. Durch den Einsatz von zwei Paketfiltern wie in Abbildung 2.3 kann der Datenverkehr besser gesteuert werden, wenn Dienste für das Internet angeboten werden sollen. Diese Server werden in der demilitarisierten Zone (DMZ) platziert. Jeder angebotene Dienst ist eine potenzielle Gefahr. Würden die Server in dieser Topologie kompromittiert, so können diese nicht als Basis für weitere Angriffe in das interne Netz oder in das Internet genutzt werden. Die Paketfilter werden so konfiguriert, dass nur Verbindungen vom Internet in die DMZ und vom LAN in die DMZ aufgebaut werden können.

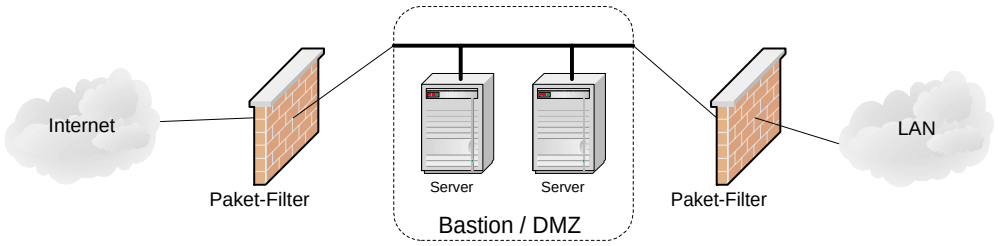

Abbildung 2.3.: Screened Subnet / DMZ

2.2.6. Zugangsschritte

Personen erhalten in der Regel erst dann Zugang zu einem Rechensystem, wenn sichergestellt wurde, dass sie in einem System wirklich erwünscht sind. Die Berechtigtungen des Benutzers werden innerhalb von drei Schritten festgelegt:

Identifizierung

Im ersten Schritt wird die Identität einer Person festgestellt. Bei der Anmeldung an einem System geschieht dies durch die Eingabe des Benutzernamens oder einer anderen vereinbarten Zugangskennung.

Authentifizierung

Während der Authentifizierung beweist die identifizierte Person ihre Identität. Dieser Beweis kann durch verschiedene Methoden erbracht werden. Gängig ist hier die Eingabe eines Kennwortes, das nur der Person bekannt ist und eindeutig vom System erkannt werden kann (z.B. durch Einweg-Passwort-Verschlüsselung). Aber auch andere Authentifizierungsmethoden sind denkbar, wie die Verwendung von Einmalpasswörten (die bei jedem Login geändert werden), der Einsatz von Token oder SmartCards oder auch biometrische Methoden.

Autorisierung

Nachdem die Identität festgestellt und bewiesen wurde, werden die Zugangsrechte für diese Person festgelegt. UNIX-Systeme z.B. benutzen ein Gruppensystem für die Autorisierung des Zugangs zu bestimmten Diensten oder Applikationen.

2.2.7. Portmapper / RPC

Viele UNIX-Dienstprogramme kommunizieren untereinander über *RPC (Remote Procedure Call)*. Aus Gründen der Verfügbarkeit und Lastverteilung wurde ein Systemprogramm namens `rpcbind` (früher auch `portmapper`) eingeführt, das die Anfragen der verschiedenen Programme behandelt und eine Verbindung zum Zielsystem herstellt. Dabei ist der letztendlich verwendete Zielport (TCP oder UDP) variabel und wird vom Portmapper festgelegt. Ein Dienstprogramm eines Systems meldet sich beim Portmapper des gewünschten Zielsystems mit seiner Programmnummer und einer Versionsnummer an und erhält den zugewiesenen Port. Der Portmapper selbst empfängt Anfragen auf einem festgelegten Port. Mittels des Programms `rpcinfo` lassen sich die zugewiesenen Ports der Dienstprogramme abfragen.

2.2.8. Intrusion Detection System

Ein Intrusion Detection System (*IDS*) betrachtet eingehende und ausgehende Netzwerkaktivität und identifiziert verdächtige Muster, die auf einen Angriff auf ein System oder das Netzwerk hindeuten. Im Gegensatz zu reinen Firewall-Systemen analysieren IDS den gesamten Netzwerkverkehr und können so auch Angriffe erkennen, die aus dem internen Netz gestartet werden. Außerdem können semantische Erkennungen durchgeführt werden, die beim Einsatz reiner Firewall-Architekturen nicht möglich sind. Es gibt mehrere Kategorien dieser Systeme.

Zum einen unterscheidet man, ob nach Missbräuchen oder Anomalien gesucht werden soll. Bei der Erkennung von Missbräuchen vergleicht das System die analysierten Daten mit einer Datenbank von Angriffssignaturen. Wie bei einem Antivirus-System hängt die Effektivität dieser Suche unmittelbar von der Qualität der Datenbank ab. Anders arbeiten Systeme, die nach Anomalien suchen. Hier werden Parameter wie Auslastungen oder Protokollarten festgelegt, die als normal gelten. Das IDS analysiert dann den Datenstrom auf dem Netzwerk auf Abweichen von diesen Parametern.

Außerdem unterscheidet man passive und reaktive Systeme. Ein passives IDS erkennt lediglich Angriffe und alarmiert einen Systemadministrator. Reaktive IDS können auf

ein Ereignis reagieren und verändern z.B. den Regelsatz einer Firewall, um weitere Angriffe durch blockieren weiteren Netzverkehrs aus dem Netzbereich des Angreifers zu verhindern.

2.3. Modellunternehmen

Die Betrachtungen dieser Arbeit beziehen sich auf ein fiktives Modellunternehmen, das viele Aspekte real existierender mittelständischer Softwareunternehmen im informationstechnologischen Sektor beinhalten soll. Das Unternehmen beschäftigt 100 Mitarbeiter und entwickelt Anwendungen für andere Unternehmen. Die Werbung und der Vertrieb der Software erfolgen direkt über das Internet mittels eigener Webserver.

Im Unternehmen werden auch besonders geschützte Daten gehalten wie z.B. die Kunden- und Lieferantendaten und die Quellcodes der entwickelten und vertriebenen Produkte. Da das Internet den Hauptweg des Vertriebs darstellt, hat seine Verfügbarkeit eine sehr hohe Bedeutung. Außerdem sind die 10 Niederlassungen mit dem Internet verbunden und über ein virtuelles privates Netzwerk (VPN) miteinander gekoppelt.

Diese Arbeit behandelt generelle Aspekte der Netzwerksicherheit. Da man häufig in der Praxis Rechensysteme auf UNIX-Basis im Serverbereich findet und dieser Bereich für Angreifer besonders interessant ist, beschränken sich konkrete Betrachtungen von Betriebssystemen auf UNIX als Beispiel.

2.3.1. Rollen

Globaler IT-Manager: Der globale IT-Manager ist verantwortlich für die Planung und Überwachung des gesamten Unternehmensnetzwerks. Er verfasst die Policy (das globale Sicherheitskonzept) und entscheidet über den Einsatz von Sicherheitssoftware (z.B. IDS), die in allen Niederlassungen eingesetzt wird. Im Gegenzug erhält er Informationen von den lokalen Systemmanagern der Niederlassungen über Konfiguration und Betriebszustand der einzelnen Systeme.

Lokaler Systemmanager: Der lokale Systemmanger soll die Vorgaben des lokalen Niederlassungsleiters praktisch umsetzen. Ist für den Betrieb und die Überwachung seiner Systeme selbst verantwortlich.

Lokales Incident Response Team (IRT): Das IRT erhält automatische Alarmierungen von den sicherheitsüberwachenden Komponenten und muss Störungen des Betriebs beheben.

2.3.2. Informationsfluss

Der globale IT-Manager in der Zentral-Niederlassung stellt Sicherheitsrichtlinien auf, die mit der von der Geschäftsleitung propagierten Unternehmensphilosophie übereinstimmen, um ein optimales Maß an Sicherheit für das Unternehmen zu gewährleisten. Diese Richtlinien werden verbindlich an die Leiter der Niederlassungen übergeben und

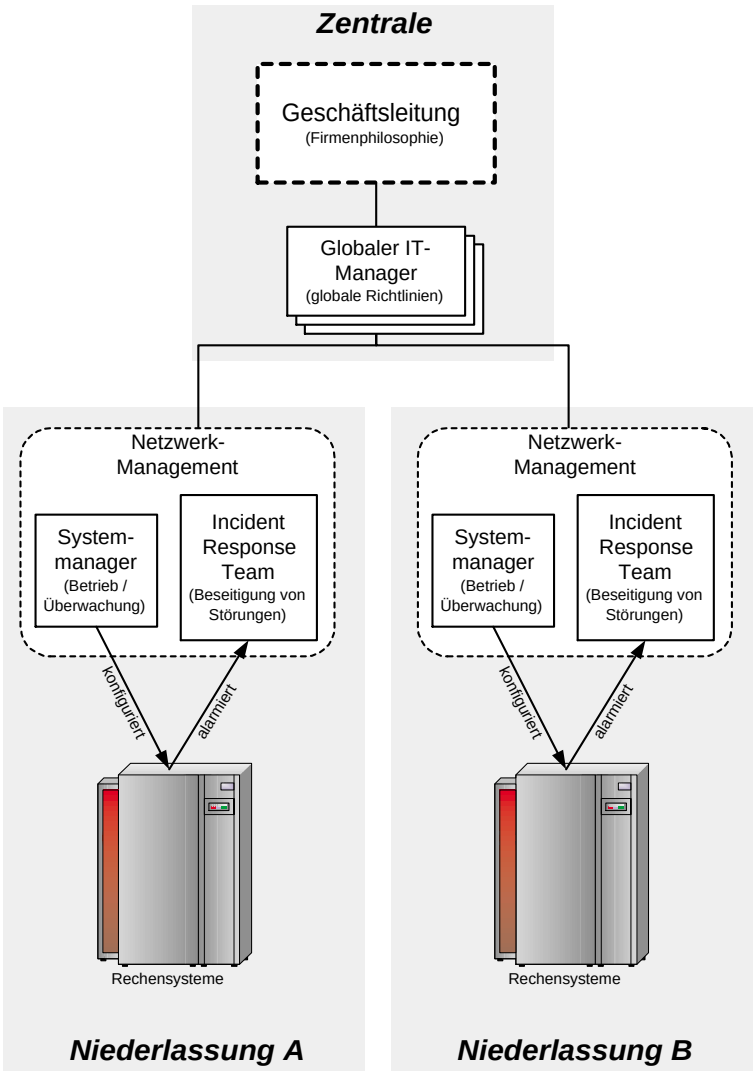

Abbildung 2.4.: Rollen im Modellunternehmen

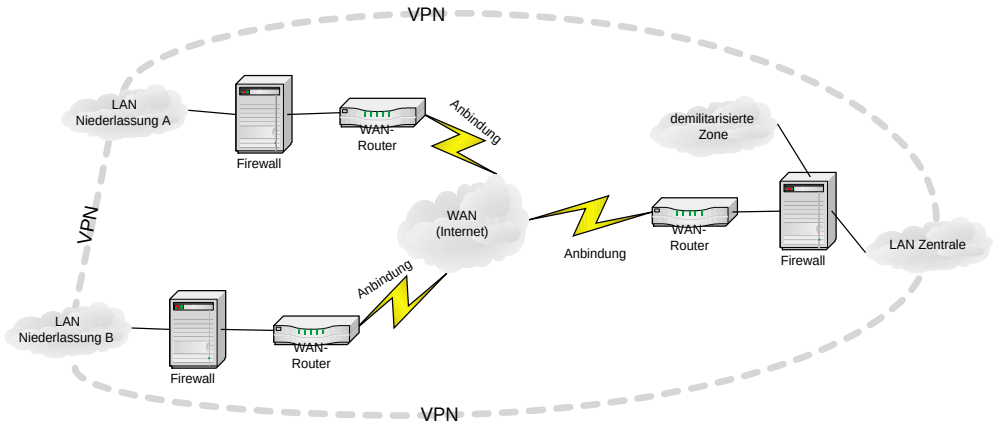

Abbildung 2.5.: Topologie des Modellunternehmens

werden in deren Verantwortung umgesetzt. Die Systemmanager betreiben und überwachen dabei System- und Netzwerktechnik nach den lokalen Gegebenheiten (spezielle Ausrichtung einer Niederlassung).

Im Falle einer technischen Störung wird das Incident Response Team (IRT) alarmiert, das diese Störung umgehend behebt. Bei gezielten Angriffen auf das Gesamtnetz können auch die Niederlassungsleiter aller anderen Niederlassungen informiert werden, um einen Angriff dieser Art abzuwehren und globale Abwehrmaßnahmen zu koordinieren.

In Abbildung 2.4 wird die Hierarchie der Rollen aufgezeigt. Dabei ist zu bedenken, dass die Zentral-Niederlassung selbst auch eigene Rechensysteme und Netztechnik betreibt und somit auch zuständige Systemmanager und ein Incident Response Team unterhält.

2.3.3. Topologie

Die Topologie des Modellunternehmens ist in Abbildung 2.5 anhand der Zentrale und zwei Niederlassungen veranschaulicht. Alle Niederlassungen sind über das Internet miteinander verbunden. Aus Kostengründen ist diese Konnektivität mehreren Standleitungen vorzuziehen, da jeweils nur eine Verbindung zum lokalen Internet Service Provider (ISP) aufgebaut werden muss — im günstigsten Fall ein Ortsgespräch über eine Wählleitung. Aus Sicherheitsgründen sind die lokalen Netze (LANs) der Niederlassungen durch eine Firewall von WAN-Router getrennt, der die direkte Verbindung zum Internet herstellt. Da die Kommunikation im Internet auf dem ungesicherten (im Sinne von: nicht kryptografisch gesicherten) TCP/IP-Protokoll basiert, sind alle Niederlassungen über Virtual Private Networks miteinander verbunden, um automatisch eine gesicherte Verbindung bei jeder Verbindung zwischen zwei Niederlassungen zu gewährleisten.

3. Internetbasierte Angriffe

Inhaltsangabe

3.1. Einführung

Es gibt viele Möglichkeiten, die IT-Infrastruktur eines Unternehmens anzugreifen. In dieser Arbeit beschränkt sich die Betrachtung allerdings auf Angriffe aus dem Internet. Eine komplette Sicherheits-Policy umfasst natürlich noch weitere Aspekte wie z.B. die physikalische Sicherheit (räumliche Absicherung sensibler Bereiche wie eines Rechenzentrums), kommunikative Sicherheit (Vertrauen in das Funktionieren von Kommunikationsstrecken externer Anbieter) oder die soziale Sicherheit. Gerade der letzte Punkt beschreibt ein kritisches Sicherheitsproblem, das als *Social Engineering* bezeichnet wird. Dabei werden Angehörige des Unternehmens überlistet, sicherheitsrelevante Informationen an nicht-autorisierte Personen weiterzugeben. Da bei der sozialen Interaktion häufig keine Sicherheitsprotokolle zum Schutz der Vertraulichkeit eingesetzt werden wie sie bei Rechensystemen üblich sind, können so Informationen leicht erschlichen werden. Die Gefahr durch Social Engineering ist deshalb vermutlich größer als durch internetbasierte Angriffe. Die thematische Einschränkung auf Angriffe aus dem Internet stellt also keine Auswahl nach Wichtigkeit dar.

3.1.1. Aufbau dieses Kapitels

Um geeignete Abwehrmaßnahmen gegen Angriffe aus dem Internet planen zu können, muss man den Ablauf eines Angriffs verstehen. Dieses Kapitel beschreibt den Ablauf

eines möglichen gezielten Angriffs auf ein Unternehmensnetz. Im jeweiligen Kontext werden sowohl die Schritte des Angreifers als auch die empfohlenen Gegenmaßnahmen und deren Durchsetzung erläutert. Die hier beschriebenen Angriffe sind keine Spielereien von „Skript-Kiddies“, sondern gezielte ernstzunehmende Angriffe mit der Absicht, Zugriff auf das Unternehmensnetz zu bekommen.

Im Verlauf dieser Arbeit werden primär netzbasierte Angriffe beschrieben. Wo im Detail auf Betriebssysteme eingegangen wird, beschränken sich die Betrachtungen auf UNIX-Derivaten wie z.B. Solaris, BSD und Linux. Diese Einschränkung soll lediglich den Umfang der Beschreibung der verschiedenen Angriffsarten beschränken. Natürlich werden in der Praxis verschiedenste Betriebssysteme (mit spezifischen Sicherheitsproblemen) eingesetzt.

3.1.2. Motivation der Angreifer

So unterschiedlich wie die Angreifer sind auch deren Motivationen. In den Anfangszeiten der Vernetzung erschien es erstrebenswert, in Systeme einzudringen, um Rechenzeit nutzen zu können. Heutzutage interessieren sich die professionellen Angreifer eher für die wirtschaftlichen Aspekte des Eindringens in Systeme anderer Unternehmen. Zum einen versuchen diese Angreifer, sicherheitsrelevante Informationen zu erschleichen, die dem beauftragenden Unternehmen dienlich sein können. Ebenso interessant kann es sein, das angegriffene Unternehmen zu sabotieren, in dem wichtige Systeme abgeschaltet werden oder öffentliche Informationen (wie Webseiten) verändert werden, um das Unternehmen in Misskredit zu bringen. Desweiteren können die kompromittierten Systeme genutzt werden, um weitere Ziele im Internet anzugreifen. So wird der eigentliche Ursprung des Angriffs verschleiert.

3.1.3. Charakterisierung der Angreifer

Die Motivationen potenzieller Angreifer sind ebenso verschieden wie ihre Methoden. An dieser Stelle sollen deshalb die bekanntesten Kategorien unterschieden werden.

Hacker

Die Begriffe *Hacker* und *Cracker* werden leider sehr inkonsequent benutzt. Die Unterscheidung an dieser Stelle soll lediglich erklären, welche Bedeutung diesen Begriffen in dieser Arbeit beigemessen wird.

Als Hacker werden Personen verstanden, die sich mit Technologien beschäftigen und sich durch die Herausforderung motivieren, Sicherheitssysteme zu überwinden und Schwachstellen darin aufzudecken, ohne sich selbst einen materiellen Vorteil zu verschaffen. Der Begriff *Hacker* entstand aus dem englischen „to hack“, was die Tätigkeit des „auf der Tastatur herumhackens“ beschreiben sollte. Ein Beispiel ist der im November 1984 von Herwart Holland und Steffen Wernéry (Mitgliedern des *Chaos Computer Clubs Hamburg*) durchgeführte Angriff auf das Bildschirmtextsystem der Deutschen Bundespost, bei dem die Hamburger Sparkasse mit Gebühren von über 135000 DM belastet wurde. Dieser Angriff wurde den Betreibern des Bildschirmtextsystems bereits drei Tage vorher angekündigt. Der gestohlene Betrag wurde der Hamburger Sparkasse

umgehend zurücküberwiesen und die Details des Angriffs den Betreibern bekannt gemacht. Zweck dieser Aktion war lediglich, auf die vermuteten und von den Betreibern verschwiegenen Sicherheitsmängel öffentlich aufmerksam zu machen. ([hackbib] S. 35 ff.)

Cracker

Eine kriminelle oder sogar zerstörerische Energie kann Crackern vorgeworfen werden. Cracker interessieren sich nicht für die Aufdeckung von Sicherheitslöchern, sondern werden die erlangten Kenntnisse geheimhalten, um sie immer wieder für sich nutzen zu können. Cracker werden nur versuchen, in Systeme einzudringen, um für sich selbst einen Nutzen daraus zu ziehen.

Cyberterroristen

Mindestens ebenso gefährlich wie Cracker sind die Cyberterroristen. Sie handeln nicht nur im eigenen Interesse, sondern sind politisch und finanziell motiviert, eine politische Denkweise durch Manipulationen in Netzwerken (z.B. Hinterlegen von Propaganda-Dokumenten auf öffentlichen Webservern) zu verbreiten. Ihr Interesse kann aber auch der Verkauf sicherheitsklassifizierter Dokumente aus Industriekonzernen oder des Militärs sein, um andere Aktivitäten finanziell zu unterstützen.

Tiger Teams

Manche Unternehmer haben viel Energie in die Absicherung der IT-Systeme ihres Unternehmens investiert und möchten nun prüfen, ob ein gut vorbereiteter Hacker in der Lage wäre, dennoch in das System einzudringen. Dazu engagiert die Geschäftsführung sogenannte *Tiger Teams*, die vom Standpunkt eines Angreifers aus einen sorgfältig geplanten Angriff auf das Netzwerk versuchen. Um eine realistische Situation zu schaffen, werden die Systemadministratoren in der Regel nicht über diese Aktionen informiert.

Skript-Kiddies

Eine mehr oder weniger ungefährliche Variation der Cracker sind sogenannte *Skript-Kiddies*. Sie verfügen über geringe Kenntnisse von Sicherheitskomponenten. Ihre Gefährlichkeit hängt von den Werkzeugen ab, mit denen sie Systeme angreifen: den Scripts. Scripts sind einfach zu bedienende Softwarepakete, deren Aufgabe es ist, unter Verwendung definierter Regelsätze und Angriffsmechanismen in ein System einzudringen. Der Angreifer versteht die Funktion des Programms meist nicht und könnte daher auch mit dem möglicherweise erfolgreichen Einbruch wenig anfangen. Die Gefährdung ist also abhängig von der Qualität der Scripts und der Kompetenz der Skript-Kiddies, die erlangten Informationen weiter zu nutzen.

3.2. Beispiel eines internetbasierten Angriffs

Es gibt eine Vielzahl von Möglichkeiten, fremde Netzwerke vom Internet aus anzugreifen. Einige sind u.a. in [iaag] (S. 91 ff) beschrieben. In dieser Arbeit soll ein geplanter Angriff ausführlich beschrieben werden. Diese Beschreibung soll exemplarisch für den gesamten Ablauf eines Angriffs von der Profilbildung bis zur Systemnutzung (s. Abb. 3.1) sein. Andere Angriffsformen sind ebenso denkbar. Beispielsweise wird an dieser Stelle von einem externen Angriff in das eigene Unternehmensnetzwerk über das Internet ausgegangen. Es wäre ebenso denkbar, dass ein Angehöriger des Unternehmens das Netzwerk von einem anderen Punkt innerhalb des Netzwerkes oder über das Telefonnetz angreift. In den einzelnen Phasen des Angriffs werden nicht nur die Gefahren aufgezeigt, sondern auch entsprechende Lösungsvorschläge zur Systemkonfiguration gegeben. Auch die Auffälligkeiten verschiedener Angriffstechniken werden beschrieben, um sie später automatisiert im Rahmen der Implementation des Sicherheitsüberwachungssystems erkennen zu können.

3.2.1. Profilbildung

Der erste Schritt eines erfolgversprechenden Angriffs auf ein fremdes Netzwerk ist die Profilbildung (engl. *footprinting*). In diesem ersten Schritt werden unauffällig Informationen über das Unternehmen gesammelt. Die damit verbundenen Aktionen beschränken sich auf öffentlich zugängliche Informationen und sollen nicht als Angriffe erkannt werden. Aber bereits in diesem Schritt können relativ einfach wichtige Informationen erlangt werden wie z.B.:

- Anschrift
- Telefonnummern
- Domain-Namen
- IP-Adressbereiche
- Spezielle Server (SMTP-Mailserver, DNS)
- E-Mail-Adressen wichtiger Systemadministratoren
- bisher aufgedeckte Sicherheitslöcher

Angriff: Eine Reihe von öffentlichen Datenbanken speichert administrative Informationen über jedes Unternehmen mit einer Anbindung ans Internet. Dazu gehört zunächst die RIPE-Datenbank[1] (*Reseaux Internet Protocol Europeens*), in denen u.a. Auskünfte über IP-Netzbereiche, Domainnamen und Routing zu finden sind.

Über die Domain „uni-hamburg.de“ sind z.B. auszugsweise folgende Informationen in der RIPE-Datenbank gespeichert:

```
domain:      uni-hamburg.de
descr:       Universitaet Hamburg; Regionales Rechenzentrum
```

[1] http://www.ripe.net

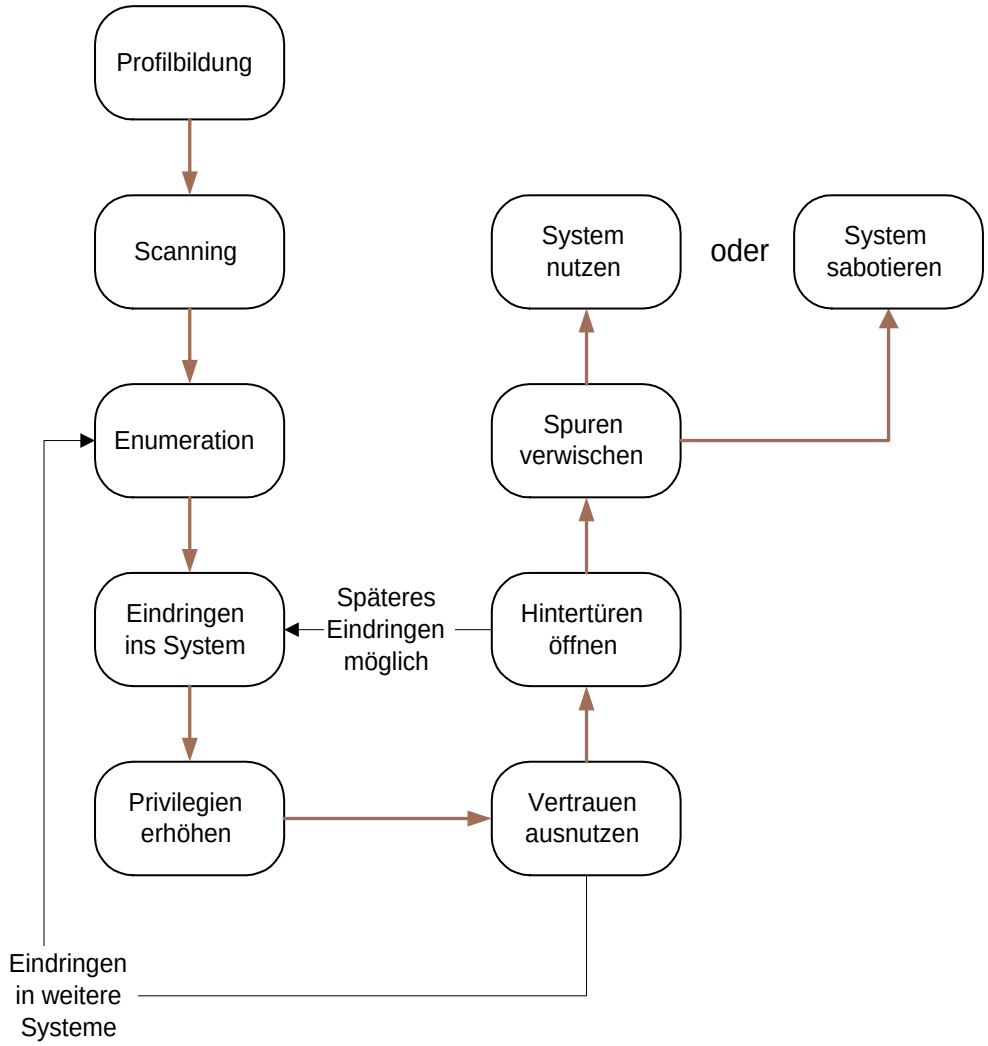

Abbildung 3.1.: Beispielhafter Ablauf einer internetbasierten Kompromittierung

```
descr:      Regionales Rechenzentrum
descr:      Schlueterstrasse 70
descr:      D-20146 Hamburg
descr:      Germany
nserver:    rzaix240.rrz.uni-hamburg.de 134.100.38.190
nserver:    rzdspc1.informatik.uni-hamburg.de 134.100.9.61
nserver:    ws-mue1.win-ip.dfn.de

person:     Gerrit Henken
address:    Universitaet Hamburg
address:    Regionales Rechenzentrum
address:    Schlueterstrasse 70
address:    20146 Hamburg
e-mail:     henken@rrz.uni-hamburg.de

person:     Heino Peters
address:    Universitaet Hamburg
address:    Regionales Rechenzentrum
address:    Schlueterstr. 70
address:    20146 Hamburg
phone:      +49 40 42838 6969
fax-no:     +49 40 42838 3096
e-mail:     heino.peters@rrz.uni-hamburg.de
```

Ein Angreifer kennt jetzt bereits die Anschrift der Organisation (Schlüterstraße 70), den Nameserver (134.100.38.190), zwei administrative Ansprechpartner mit E-Mail-Adresse und eine Telefon- und Faxnummer. Es wäre denkbar, dass sich ein Angreifer als Heino Peters ausgibt (dazu bedarf es lediglich einer gefälschten FROM-Zeile in einer E-Mail) und beim Internet-Service-Provider wichtige RIPE-Daten wie das Routing ändern lässt (s. *Social Engineering*).

Neben der RIPE-Datenbank enthält auch die Datenbank des Domain-Nameservers interessante Informationen. Die MX-Einträge geben z.B. die MX-Server (*mail exchange server*) an, die als SMTP-Server für eine gegebene Domain fungieren. Möglicherweise lassen sich später Schwachstellen in der Mailer-Software (wie z.B. Sendmail) ausnutzen. Dies sind die MX-Einträge für die Domain „uni-hamburg.de“.

```
uni-hamburg.de mail is handled (pri=100) by host1.dfn.de
uni-hamburg.de mail is handled (pri=20) by host2.uni-hamburg.de
uni-hamburg.de mail is handled (pri=10) by host3.ecrc.de
```

Wichtige Informationen lassen sich aber auch über freiwillig herausgegebene Medien des Unternehmens erlangen, wie z.B. durch die eigenen Web-Seiten. Dort findet der Angreifer Adresse, Ansprechpartner und häufig auch versteckt im HTML-Quellcode weitere nützliche Informationen, die auf eingesetzte Anwendungen, Betriebssysteme oder Serverdienste hinweisen können.

Desweiteren kann auch die Netzwerktopologie der Internetanbindung interessant sein. Mit dem UNIX-Programm `traceroute` lässt sich die Route zu einem

Zielsystem feststellen. Die Route bis zum System „`www.uni-hamburg.de`“ sieht wie folgt aus:

```
1  ir-frankfurt2.g-win.dfn.de (194.31.232.222)  22.124 ms
2  cr-frankfurt1.g-win.dfn.de (188.1.80.37)  21.147 ms
3  cr-muenchen1.g-win.dfn.de (188.1.18.26)  29.779 ms
4  cr-hamburg1.g-win.dfn.de (188.1.18.30)  45.870 ms
5  ar-hamburg1.g-win.dfn.de (188.1.92.2)  45.076 ms
6  134.100.38.100 (134.100.38.100)  45.411 ms
7  www.uni-hamburg.de (134.100.33.230)  45.985 ms
```

Auffälligkeiten: Der Angreifer ist darauf bedacht, bei der Profilbildung keine auffälligen Spuren zu hinterlassen. Es ist also nicht möglich, aus diesen Aktionen bereits einen Angriff zu vermuten.

Gegenmaßnahmen: Gegen diese Art des Ausspionierens gibt es kaum Abwehrmöglichkeiten. Die RIPE- und DNS-Datenbanken sind für den ordnungsgemäßen Betrieb eines Netzwerks unverzichtbar. Man kann lediglich versuchen, die Anzahl der Angaben in diesen Datenbanken auf ein technisch notwendiges Minimum zu reduzieren.

Lediglich traceroute-Anfragen können intelligente IDS (*Intrusion Detection System*) erkennen. An dieser Stelle lassen sich auch Programme einsetzen, die traceroute-Antworten erzeugen und so einen Angreifer in die Irre führen (z.B. RotoRouter). Es kann aber auch ausreichen, bestimmte ICMP- und UDP-Anfragen abzulehnen oder zu verwerfen.

3.2.2. Scanning

Nach der Profilbildung kennt ein Angreifer Netzbereiche, die er genauer untersuchen kann. Hauptsächlich wird er über freigegebene Dienste in Systeme eindringen. Auf IP-Basis kann das Netz auf freigegebene Dienste der Protokolle ICMP, UDP und TCP untersucht werden. Das Scanning ist noch kein direkter Angriff, kann aber vom Zielsystem mit entsprechenden IDS (*Intrusion Detection Systems*) erkannt werden.

Angriff: Unter Verwendung des ICMP-Protokolls können verfügbare Systeme mittels eines Ping-Scans eruiert werden. Der Angreifer sendet dabei an jede einzelne IP-Adresse aus dem Zielnetz ein Typ 8-ICMP-Paket (*ECHO REQUEST*) und erwartet dann vom Zielsystem ein Typ 0-ICMP-Paket (*ECHO REPLY*). Ein einzelnes System lässt sich auch mit dem verbreiteten `ping`-Befehl abfragen. Für die Abfrage ganzer Netzbereiche eignen sich Programme wie `fping`[2] besser, die nicht ein System nach dem anderen Abfragen, sondern parallel eine große Anzahl an Abfragen an das Zielsystem stellen und somit schneller sind. Immerhin dauert das Scannen eines Klasse-A-Netzes (mit über 16 Mio. IP-Adressen) sonst bereits etwa ein halbes Jahr, wenn man ein System pro Sekunde überprüft. In jedem Fall ist dieser kontrollierte Massen-Ping einem Broadcast-Ping vorzuziehen, bei dem alle Systeme eines Netzes angewiesen werden, gleichzeitig zu antworten.

[2] http://www.fping.com

Dies kann schnell zu einer Überflutung des Netzes führen und damit zu einem Denial-of-Service-Zustand.

Neben den ECHO-REQUEST-Paketen gibt es auch noch zwei weitere nützliche Paket-Typen: ADDRESS MASK REQUEST (Typ 18) und TIMESTAMP (Typ 13). Eine ADDRESS MASK REQUEST-Anfrage beantwortet ein Server mit der IP-Netzmaske und gibt somit den lokalen Netzbereich an. Als Antwort auf eine TIMESTAMP-Anfrage erhält der Client die aktuelle Systemzeit des Servers. Diese Informationen lassen sich z.B. mit dem Programm `icmpquery`[3] abfragen.

Häufig werden aber ICMP-Pakete bereits von Netzwerkkomponenten wie Routern herausgefiltert. In diesem Fall bietet sich statt eines ICMP-Ping-Scans ein TCP-Ping-Scan an. Dazu sind Programme wie `nmap`[4] oder `Hping`[5] gut geeignet. Hier gibt es mehrere mehr oder weniger auffällige Varianten:

TCP-Verbindungs-Scan: Bei dieser Scan-Art wird versucht, eine TCP-Verbindung vollständig mit einem Drei-Wege-Handshake aufzubauen (vgl. Abb. 2.1). Der Client sendet ein SYN-Paket, der Server antwortet mit einem SYN-ACK-Paket und der Client bestätigt wiederum die Verbindung mit einem ACK-Paket.

TCP-SYN-Scan: Hier wird eine TCP-Verbindung „halb“ geöffnet. Der Client sendet auch hier ein SYN-Paket an den Server. Ist der TCP-Port offen (im Status `LISTENING`), so antwortet der Server mit einem SYN-ACK-Paket. Anderenfalls sendet der Server ein RST-ACK-Paket, um damit anzuzeigen, dass kein Dienst auf dem Port auf Anfragen wartet. Diese Art des Scans ist unauffälliger, weil nicht vollständig geöffnete TCP-Verbindungen häufig nicht protokolliert werden. Deshalb nennt man diesen Scan auch TCP-Stealth-Scan.

UDP-Scan: Beim verbindungslosen UDP gibt es keine komplizierte Aufbauphase, wie beim Drei-Wege-Handshake des TCP. Dennoch gibt es auch hier Möglichkeiten, nicht verfügbare UDP-Ports zu erkennen. In diesem Fall sendet der Server eine Nachricht mittels ICMP zurück (`ICMP port unreachable`). Ansonsten kann man davon ausgehen, dass der Port verfügbar ist. Da UDP ein ungesichertes Protokoll ist, sind die Ergebnisse eines UDP-Scans möglicherweise weniger brauchbar als die von TCP-Scans.

Entsprechend den Vorschlägen des RFC 793 gibt es noch weitere mögliche Scans wie den TCP-FIN-Scan, den TCP-Xmas-Tree-Scan und den TCP-Null-Scan (vgl. [hackexp] S. 40).

Nachdem mit den aufgezeigten Scan-Methoden aktive Systeme geortet werden konnten, können jetzt auf diesen Systemen die angebotenen Dienste abgefragt werden, um sie als mögliche Angriffswege zu nutzen. Auch hier ist die Software `nmap` geeignet, denn sie bietet einerseits grundlegendes ICMP-, TCP- und UDP-Scanning an. Zusätzlich können die IP-Datenpakete fragmentiert werden, denn ältere IDS-Systeme oder Paket-Filter untersuchen Fragmente nur einzeln. Modernere IDS-Systeme defragmentieren zunächst alle zusammengehörigen Datenpakete, bevor diese auf ihren Inhalt überprüft werden. Desweiteren können

[3] http://www.securiteam.com/tools/ICMPQuery_remote_host-type_detection.html
[4] http://www.insecure.org/nmap
[5] http://www.kyuzz.org/antirez/hping.html

neben den Scan-Datenpaketen noch weitere überflüssige Daten gesendet werden (*decoy scanning*), um IDS-Systeme zu verwirren. Auch die Fähigkeit von `nmap` zum *Ident-Scanning* ist interessant, mit dem der Ident-Dienst (s. RFC 1413) auf UNIX-Servern angewiesen wird, die Benutzeridentifikation auszugeben, unter dessen Rechten dieser Dienst läuft. Hat man einen Dienst identifiziert, der „root"-Rechte auf einem System hat, kann man durch Kompromittierung alleine dieses Dienstes bereits volle Zugriffsrechte auf ein System erhalten.

`nmap` ist interessanterweise sogar fähig, das Betriebssystem eines Zielsystems zu erraten. Es zieht dazu u.a. Informationen heran, welche TCP- und UDP-Ports bei welchen Betriebssystemen standardmäßig geöffnet sind. Bei Microsoft Windows-Systemen sind z.B. häufig die Ports 135 bis 139 ansprechbar (Netbios-Dienste). Aber auch die Reaktion auf IP-Anfragen gibt Anhaltspunkte auf die Implementation des IP-Stacks und erlaubt somit Rückschlüsse auf das Betriebssystem. `nmap` wird mit mit mehreren hundert Signaturen ausgeliefert, die diese Erkennung erlauben.

Auffälligkeiten: Wo die Zielorganisation bei der Profilbildung gegen das Sammeln von Informationen noch relativ machtlos war, tritt der Angreifer beim Scanning deutlich in Erscheinung. Das Muster eines ICMP- oder TCP-Scans ist relativ eindeutig und kann schnell von IDS-Systemen erkannt werden, denn wenige erwünschte Anwendungen benötigt sukzessiven Zugriff auf alle Systeme in einem Netz in der Reihenfolge der IP-Adressen. Als IDS-System lässt sich hier z.B. Snort[6] einsetzen.

Die meisten meisten Unternehmen setzen zwar bereits Firewalls (oder zumindest grundlegende Paket-Filter) ein, analysieren dann aber leider häufig nicht die Protokolldateien. Dabei ist es besonders wichtig, Anomalien im Netzwerkverkehr zu erkennen und ihnen entgegenzuwirken. Selbst ein ausgeklügeltes aber statisches Regelsystem auf einem modernen Firewallsystem kann einen Angreifer nicht über längere Zeit aufhalten. Es ist also wichtig, bereits Scan-Versuche ernst zu nehmen und den Angreifer aus dem System z.B. durch restriktivere Regeln des Paket-Filters auszuschließen. In jedem Fall ist eine erhöhte Wachsamkeit empfohlen, denn weitere Unregelmäßigkeiten könnten Anzeichen für einen gezielten Angriff sein.

Gegenmaßnahmen: Diese Scans lassen sich verhindern, wenn die vom Paket-Filter durchgelassenen ICMP-Pakete eingeschränkt werden. Grundsätzlich sind nur drei der insgesamt 18 ICMP-Pakete für den Betrieb wirklich nötig — dies sind: ECHO-REPLY, HOST UNREACHABLE und TIME EXCEEDED.

Zur Erhöhung der passiven Sicherheit sollten alle nicht benötigten Dienste auf einem System abgeschaltet werden. Jeder Dienst ist ein potenzieller Angriffsweg und je mehr Dienste offen gehalten werden, um so leichter ist ein Einbruch ins System.

3.2.3. Enumeration

Die bisherigen Schritte Profilbildung und Scanning waren nicht-intrusiv. Die Profilbildung war für das Zielunternehmen überhaupt nicht feststellbar — das Scanning

[6] http://www.snort.org

nur durch den Einsatz von IDS-Systemen. Zu diesem Zeitpunkt kennt der Angreifer die Dienste des Zielunternehmens, die er als Ansatzpunkt für gezielte Angriffe nutzen kann. Es hängt natürlich von der Art des Dienstes ab, auf welche Weise ein Angriff möglich ist. An dieser Stelle seien lediglich Beispiele für Anfälligkeiten gängiger UNIX-Dienste erwähnt.

Angriff auf NFS-exportierte Verzeichnisse

Angriff: Das *Network File Systems (NFS)* ist ein Netzwerkdateisystem, mit dessen Hilfe ganze Verzeichnisbäume von einem Server auf mehreren Clients verwendet werden können. Hat das Scanning ergeben, dass der Port 2049 auf einem System auf Anfragen reagiert, so ist dort vermutlich ein NFS-Server zu finden. Der Befehl `showmount` zeigt dann die exportierten Verzeichnisse an. Ist der NFS-Server in seinen Zugriffsrechten auf diese Verzeichnisse nicht beschränkt, so könnten beliebige Systeme Zugriff darauf erhalten. Möglicherweise wird sogar der gesamte Dateibaum exportiert und ist beschreibbar. In diesem Fall könnte ein Angreifer wichtige Systemdaemonen überschreiben und an deren Stelle eigene Programme mit root-Rechten ablaufen lassen.

Auffälligkeiten: Das Anzeigen der exportierten Verzeichnisse mittels `showmount` ist die korrekte Arbeitsweise des NFS-Daemonen. Es ist hier nicht feststellbar, ob auf ein exportiertes Verzeichnis in böswilliger Absicht zugegriffen wird.

Gegenmaßnahmen: Nach Möglichkeit sollten keine Systemverzeichnisse wie / oder `/etc` exportiert werden. Ebenso sollten Verzeichnisse nur dann als beschreibbar exportiert werden, wenn dies unbedingt nötig ist. Zudem lassen sich die meisten NFS-Server konfigurieren, welchen Systemen (auf Basis der IP-Adresse) überhaupt ein Zugriff erlaubt wird. Da der NFS-Dienst auch über den Portmapper (RPC-Dienst) angesprochen wird, bietet sich auch der Einsatz eines Secure Portmappers an. Ansonsten sollte zumindest der Port 2049 durch einen Paket-Filter blockiert werden, damit NFS-exportierte Verzeichnisse nicht vom Internet aus verwendet werden können.

Angriff auf NIS-Verzeichnisdienste

Angriff: Der Verzeichnisdienst *NIS (Network Information Service)* — früher *YP (Yellow Pages*) genannt — bietet eine Datenbank von verschiedenen Netzwerk-Informationen an. Dort sind unter anderem Benutzerdaten einschließlich der dazugehörigen verschlüsselten Passwörter gespeichert. Die NIS-Datenbank kann relativ unbeschränkt abgefragt werden, sobald der NIS-Domänenname bekannt ist.

Auffälligkeiten: Da der NIS nicht über spezielle Sicherheitsmechanismen wie eine Authentisierung verfügt, die die Datenbank besonders sichern könnten, lässt sich ein unautorisierter Zugriff nicht erkennen.

Gegenmaßnahmen: Auf jeden Fall sollte der NIS-Domänenname vom DNS-Domänenname verschieden und auch sonst schwer zu erraten sein, um Angriffsversuche zu erschweren. Auch die Einschränkung des Dienstes auf einen Bereich von IP-Adressen ist sinnvoll. Falls möglich, bietet sich auch der Einsatz von NIS+ an,

das eine Verschlüsselung und Authentifizierung über Secure-RPC erlaubt. Nur relativ wenige Systeme unterstützen allerdings bisher NIS+ (z.B. Solaris).

Angriff über Finger/rusers/rwho

Angriff: Der Finger-Dienst (TCP-Port 79) gibt einem Angreifer Auskunft darüber, welche Benutzer sich gerade an einem System angemeldet haben und wann der letzte Befehl ausgeführt wurde. So kann sich ein Angreifer versichern, nicht beobachtet zu werden, wenn er ein System angreifen möchte. Mancher der erhaltenen Informationen könnten auch für einen Social Engineering-Angriff genutzt werden. Auch die Dienste `rusers` und `rwho` können ähnliche Informationen ausgeben.

Auffälligkeiten: Den Finger-Befehl auszuführen stellt noch keine erkennbare aggressive Handlung dar.

Gegenmaßnahmen: Nach Möglichkeit sollten die Dienste `fingerd`, `rwhod` und `rusersd` abgeschaltet werden, da sie für den Betrieb eines Systems nicht unbedingt benötigt werden.

Angriff über Sendmail

Angriff: Der `sendmail`-Dienst[7] zur Zustellung von E-Mails kann ebenfalls dazu benutzt werden, die Existenz von Benutzernamen zu verifizieren. Dazu bieten sich die Befehle `EXPN` oder `VRFY` an.

Auffälligkeiten: Soweit bekannt verfügt `sendmail` über keine Alarmierungs-Mechanismen bei der Ausführung bestimmter Befehle während einer SMTP-Verbindung.

Gegenmaßnahmen: Durch entsprechende Konfiguration des Sendmail-Dienstes lassen sich bestimmte Befehle wie auch `EXPN` und `VRFY` abschalten.

Angriff über TFTP

Angriff: Es ist besonders nützlich für den Angreifer, Zugriff auf die Datei `/etc/passwd` zu bekommen, um dort weitere Benutzerdaten und möglicherweise sogar deren (verschlüsselte) Passwörter auszulesen (sofern kein Shadowing-Verfahren zur Speicherung der Passwörter benutzt wird). Ein häufig benötigter Dienst, der dazu missbraucht werden kann, ist das `TFTP` *(Trivial File Transfer Protocol)*. Dieses Protokoll zur Dateiübertragung verfügt über keinerlei Authentisierungsmechanismen und wird primär zur Erstkonfiguration von Netzwerkkomponenten eingesetzt. Bei einer korrekten Konfiguration erhält ein Benutzer nur Zugriff auf ein angegebenes Unterverzeichnis (z.B. „`/tftpboot`“). Wäre hier fälschlicherweise das Hauptverzeichnis „/“ angegeben, könnte ein Angreifer beliebige Dateien (und so auch `/etc/passwd`) auslesen.

[7] http://www.sendmail.org

Auffälligkeiten: Bei entsprechender Konfiguration des TFTP-Daemonen können die übertragenen Dateien protokolliert werden. Es ist also möglich, den Zugriff auf Dateien wie `/etc/passwd` festzustellen. Allerdings ist es dann eher sinnvoll, den Zugriff von vornherein einzuschränken bzw. den Dienst vollständig abzuschalten.

Gegenmaßnahmen: Der TFTP-Dienst sollte nur aktiviert werden, solange er unbedingt benötigt wird. Und auch dann sollte nur ein spezielles Unterverzeichnis zur Benutzung freigegeben werden.

3.2.4. Zugriff erlangen

Das Ziel dieses Schrittes ist es, überhaupt Zugang zum Zielsystem zu erreichen. Viele Dienste werden sicherheitshalber nicht unter vollen Rechten laufen gelassen, sondern unter Pseudo-Benutzernamen wie „nobody“. Aber auch, wenn ein Angreifer Kontrolle über einen Benutzernamen mit geringen Rechten erlangt hat, kann er möglicherweise diese Privilegien erhöhen. Immerhin ist er zunächst in der Lage, das System zu betreten und nach weiteren Schwachstellen zu suchen.

Angriff durch fehlende Eingabeüberprüfung

Angriff: Häufig werden Benutzereingaben nicht auf Korrektheit überprüft. So können besonders im Bereich der CGI-Skripte[8] Programme durch geschickte Eingaben vom Angreifer gewünschte Programme gestartet werden. Beispielsweise konnte ein Angreifer ein altes PHF-Skript (das standardmäßig mit gängigen Webservern wie dem Apache oder dem NCSA-HTTPd ausgeliefert wurde) wie folgt aufrufen

```
/cgi-bin/phf?Qalias=x\%0a/bin/cat+/etc/passwd
```

und erhielt als Ausgabe den Inhalt der Datei `/etc/passwd`, da das Zeichen „%0a“ als Zeilenwechsel und damit als Beginn einer neuen Befehlssequenz verstanden wurde. Es war sogar möglich, mittels

```
/cgi-bin/phf?Qalias=x\%0a/bin/cat\%20xterm\%20-display...
```

ein Terminalfenster unter den Rechten des Webservers zu öffnen.

In einem anderem Beispiel gibt ein Webserver den Inhalt eines abgesendeten Formularfeldes noch einmal aus. Wurde der Inhalt des Feldes nicht überprüft und einfach wieder ausgegeben, konnte sich ein Angreifer der Technik der Server-Side-Includes bedienen. Damit können Programme auf dem Webserver ausgeführt und das Ergebnis des Programmlaufs in die Webseite eingebettet werden. Die Eingabe von

```
<!--#exec cmd="/bin/mail angreifer@domain.com < cat /etc/passwd"-->
```

führte z.B. dazu, dass der Inhalt der Datei `/etc/passwd` per E-Mail an den Angreifer gesendet wurde.

[8] *Common Gateway Interface*: Ausführung serverseitiger Skripte auf Webservern

Auffälligkeiten: Es gibt keine generellen Indizien eines Angriffs dieser Art.

Gegenmaßnahmen: Moderne Webserver verfügen über verschiedene Techniken wie CGIs, PHP etc. Es sollten nur die Features aktiviert werden, die für den Betrieb unbedingt benötigt werden. Bei der Programmierung von CGI-Skripten sollten die Eingaben von Benutzern immer überprüft werden.

Angriff durch Manipulationen mittels NFS

Angriff: Wie bereits auf Seite 3.2.3 diskutiert, stellen NFS-exportierte Verzeichnisse ein Risiko dar. Erhält ein Angreifer sogar Schreibrechte auf Systemverzeichnisse, kann er z.B. System-Daemonen wie den `in.ftpd` durch ein beliebiges Programm oder Skript austauschen. Er braucht dann lediglich noch diesen Dienst über TCP/IP anzusprechen und ruft damit sein dort installiertes Programm auf. Läuft eines dieser Dienstprogramme unter der Benutzerkennung `root`, kann sich der Angreifer einen Zugang mit vollen Zugriffsrechten sichern.

Auffälligkeiten: Manipulationen im Dateisystem können mit speziellen Programmen entdeckt werden, die Prüfsummen über die vorhandenen Dateien bilden und Veränderungen aufspüren. Beispiele für diese Programme sind *AIDE*[9] und *Tripwire*[10].

Gegenmaßnahmen: Nach Möglichkeit sollte der Export von NFS-Verzeichnissen sehr restriktiv konfiguriert sein, um Missbrauch zu vermeiden.

3.2.5. Privilegien erhöhen

Im ersten Schritt hat ein Angreifer Zugang zum Zielsystem gefunden. Um das System wirklich nutzen zu können, fehlt ihm bis zu diesem Punkt allerdings die Berechtigung. Möglicherweise hat er einen Systemdienst kompromittiert, der aber nicht unter vollen Zugriffsrechten abläuft. Es wird jetzt sein Ziel sein, höhere Rechte zu erreichen — im günstigsten Fall den Zugang als Systemadministrator (*root*).

Angriff mittels Brute-Force-Passwort-Dekodierung

Angriff: Genau genommen lassen sich Passwörter der Benutzer von UNIX-Systemen nicht dekodieren, da sie über eine Einweg-Verschlüsselungsfunktion kodiert und gespeichert werden.[11] Ein Angreifer kann aber alle möglichen Passwörter über die Funktion neu verschlüsseln und prüfen, ob das ausprobierte Passwort korrekt war. Intelligente Programme wie `crack` von Alec Muffett nutzen dabei menschliche Vorlieben bei der Auswahl von Passwörtern aus.

Zeitintensiver ist das direkte Ausprobieren von Benutzernamen und Passwörtern an einem System über Dienste wie `telnet` oder `FTP`. Bei vielen Betriebssystemen wird der Zugang nach drei Fehlversuchen für einige Sekunden blockiert. Die

[9] `http://www.cs.tut.fi/~rammer/aide.html`
[10] `http://www.tripwire.com`
[11] Anmerkung: es gibt keine echten Einweg-Funktionen. Die Entschlüsselung ist zwar theoretisch möglich, der Aufwand zur Entschlüsselung ist aber extrem hoch — höher als exponentiell.

Dauer der Sperre erhöht sich jedesmal. Damit ist es zeitlich sehr aufwendig, eine große Anzahl von Probe-Logins ablaufen zu lassen.

Auffälligkeiten: Die Dekodierung von Passwörtern findet meist auf dem System des Angreifers statt und ist somit vom angegriffenen Unternehmen aus nicht feststellbar.

Gegenmaßnahmen: Es sollte bereits im Voraus verhindert werden, dass selbst die verschlüsselten Passwörter aus dem System ausgelesen werden können (siehe voriges Kapitel). Müssen unsichere Komponenten wie NIS eingesetzt werden, sollten die Passwörter so sicher wie möglich gewählt werden. Ansonsten bietet sich auch der Einsatz von Shadowing-Verfahren an, bei dem Passwörter nicht mehr direkt in der Datei `/etc/passwd` stehen, sondern in einer gesonderten Datei, die nur für den Benutzer `root` zugreifbar ist — eine Technologie, die leider bei NIS nicht einsetzbar ist. Ebenfalls wichtig ist die regelmäßige Änderung des Passwortes. Falls möglich, können auch Einmalpasswörter verwendet werden, die sich bei jedem Login ändern. Allerdings muss man damit rechnen, dass sich Benutzer dieses Passwort dann aufschreiben. Moderne Authentisierungssysteme wie SecurID erlauben auch den Zugriff mit einer Kombination aus einem gewählten Passwort und einer von einem Token angezeigten Kennzahl (dem sogenannten *Passcode*). Falls Benutzer eigene Passwörter wählen dürfen, sollten diese bereits bei der Auswahl auf ihre Sicherheit geprüft werden.

Angriff durch Ausnutzung von Schwachstellen in Software

Angriff: Eine weitere Möglichkeit zum Missbrauch von Systemprogrammen sind Pufferüberläufe (*buffer overflows*). Viele Programme haben einen Puffer festgelegter Größe, in dem Eingabewerte gehalten werden. Wird diese Maximalgröße überschritten, kann ein Programmcode an diese Eingabe angehängt werden und wird auf dem Server möglicherweise ausgeführt. Es ist verhältnismäßig schwierig, diese Programmcodes zu entwerfen, da sie je nach Version des Daemonen und der Hardwareplattform des Zielsystems verschieden sind. Bei erfolgreichem Einsatz verfügt der Angreifer allerdings über die Kontrolle dieses Prozesses. Ist bei einem Programm zudem noch das SetUID-Flag gesetzt, so erhält der Angreifer volle Zugriffsrechte über das System.

Auffälligkeiten: Es gibt keine generellen Indizien eines Angriffs dieser Art.

Gegenmaßnahmen: Daemonen im System sollten unter möglichst geringen Rechten laufen, damit ein Angreifer mit einem kompromittierten Dienstprogramm so wenig wie möglich anfangen kann. Besonders sollten Daemonen nicht SUID ablaufen. Auf manchen UNIX-Systemen ist es auch möglich, die Ausführung von Programmcode auf dem Stack zu verbieten, wodurch Angriffe nicht mehr möglich sind, die Pufferüberläufe ausnutzen.

3.2.6. Vertrauen ausnutzen

UNIX-Systeme sind mittels sogenannter r-Dienste so konfigurierbar, dass anderen Systemen gegenüber ein Vertrauensverhältnis konfiguriert wird. Sie erlauben einem Anwender mit einem bestimmten Namen im Netzwerk Zugriff auf andere Systeme erlau-

ben, ohne eine erneute Authentisierung durchzuführen. Für einen Angreifer reicht es dann aus, lediglich ein System zu infiltrieren, womit ihm durch dieses ungerechtfertigte Vertrauen automatisch Zugriff auf alle Systeme im Netzwerk gegeben wird.

Angriff durch Nutzung von r-Diensten

Angriff: Ein früher häufig verwendeter Dienst war `rlogin` (*remote login*). Damit ist es möglich gewesen, die Authentisierung eines Benutzers zu umgehen, indem ein Vertrauensverhältnis anderen Systemen gegenüber ausgesprochen wurde. Konkret muss ein Benutzer dazu in seinem Home-Verzeichnis eine Datei „`.rhosts`“ anlegen, in der er namentlich den Namen eines Rechensystems und einen Benutzernamen angibt. Der entsprechende Benutzer kann dann ohne neue Eingabe des Passwortes direkt unter den Rechten dieses Benutzers arbeiten. Im schlimmsten Fall wird dem Benutzer root so eine Datei untergeschoben.

Auffälligkeiten: Dieser Angriff kann leicht erkannt werden, in dem regelmäßig neu erstellte `.rhosts`-Dateien gesucht werden.

Gegenmaßnahmen: Der `rlogin`-Dienst ist veraltet und wird nur noch sehr selten eingesetzt. Es empfiehlt sich eher die Verwendung der *Secure Shell (SSH)*, die neben einer verbesserten Authentisierung die benutzten Terminalverbindungen zusätzlich kryptografisch sichert. Sofern möglich, sollte der `rlogin`-Dienst deaktiviert werden.

3.2.7. Hintertüren öffnen und Spuren verwischen

Installation von Root-Kits

Nachdem ein Angreifer volle Zugriffsrechte über ein System erlangt hat, kann er dieses System als Ausgangspunkt für weitere Angriffe nutzen. Da sein Angriff möglicherweise entdeckt wird, ist es wichtig, dass er sich weitere Möglichkeiten öffnet, die ihm einen späteren Zugang erlauben.

Es existiert eine Anzahl von Software-Paketen, die nach einem erfolgreichen Einbruch wichtige Systemfunktionen verändern. Ein Angreifer hat dann weiterhin Zugriff zum System. Der Systemadministrator kann mit seinen Systemprogrammen nicht feststellen, ob unerwünschte Programme laufen oder neue Zugänge geöffnet wurden, da die manipulierten Programme die Zugänge gezielt verbergen.

Angriff: Root-Kits lassen sich häufig sehr einfach installieren. Hat ein Angreifer einmal einen Root-Zugang erreicht, braucht er lediglich das Archiv des Root-Kits auf das Zielsystem zu übertragen, es auszupacken und einen Befehl aufzurufen. Danach stehen ihm verschiedene Hintertüren automatisch zur Verfügung, um das System später wieder benutzen zu können. Außerdem beseitigt das Installationsprogramm noch Datenspuren in Protokolldateien, die auf einen Angriff schließen lassen. Sehr gefährlich sind ebenfalls Sniffer-Programme, die ständig den Datenverkehr auf dem Netzwerk abhören bzw. Logins mitschneiden und so Passwörter sammeln.

Auffälligkeiten: Wie bereits auf Seite 35 beschrieben, lassen sich Veränderungen im Dateisystem mit prüfsummenbildenden Programmen wie `AIDE` oder `Tripwire` aufspüren. Die Prüfsummen werden nach erwünschten Änderungen im System vom Administrator neu geschrieben und dann an einem sicheren Ort verwahrt (z.B. auf einer schreibgeschützten Diskette).

Wird die Installation eines Root-Kits vermutet, dann sollte das gesamte System gelöscht werden. Das Einspielen von Sicherheitskopien ist nicht mehr zu empfehlen, da auch diese kompromittiert sein könnten. Es empfiehlt sich dann eine komplette Neuinstallation von den Original-Medien.

Gegenmaßnahmen: Angriffe dieser Art lassen sich schwer abwehren. Sobald ein Angreifer root-Rechte in einem System erlangt hat, kann man seine weiteren Schritte nicht mehr einschränken.

Protokolldateien manipulieren

Für einen Angreifer ist es grundsätzlich wichtig, nach einem erfolgreichen Einbruch nicht aufzufallen. Häufig kontrollieren automatische Sicherheitsprogramme in einem System die Protokolldateien auf Sicherheitsverstöße. Der Angreifer wird also versuchen, diese Protokollierung zu verhindern.

Angriff: Auf UNIX-Systemen werden die letzten angemeldeten Benutzer in einer Datei namens `wtmp` aufgezeichnet. Mit Hilfe von Manipulationsprogrammen wie `wzap` kann ein Angreifer sehr einfach seine eigenen Sitzungen aus der Protokolldatei entfernen.

Auffälligkeiten: Häufig verfügen IDS-Systeme über Funktionen, die Veränderungen wichtiger Protokolldateien aufspüren. Es ist verdächtig und sehr leicht aufzuspüren, wenn eine Protokolldatei plötzlich kürzer wird.

Gegenmaßnahmen: Grundsätzlich gibt es zwei Möglichkeiten, die Manipulation von Protokolldateien zu verhindern. Einerseits lassen sich Dateien erzeugen, an die Daten angehängt werden können, die aber ansonsten nicht verändert werden können. Zum anderen lassen sich mit dem Syslog-Mechanismus Protokolleinträge auf andere Systeme weiterleiten. Ein Angreifer müsste dann also zusätzlich ein weiteres System kompromittieren, um seine Spuren zu verwischen.

3.3. Bedarf für Systemmanagement

Im diesem Kapitel wurde gezeigt, welche Informationen und Methoden ein Angreifer verwendet und welche Spuren er beim Angriff hinterlässt. Viele potenzielle Angriffsmöglichkeiten lassen sich durch eine sorgfältige Konfiguration von Diensten eliminieren. Häufig werden aber auch Risiken bewusst in Kauf genommen, wenn ein Unternehmen auf bestimmte Dienste nicht verzichten kann. Soweit es möglich ist, wurden die Auffälligkeiten verschiedener Angriffstechniken erklärt, um diese später im Rahmen einer automatisierten Sicherheitsüberwachungsinstanz erkennen und alarmieren zu können. Die Vorüberlegungen zum Entwurf dieser Instanz sind Thema des folgenden Kapitels.

4. Anforderungen an ein Sicherheitsüberwachungssystem

Inhaltsangabe

In [intman, S. 103] wird Netz- und Systemmanagement wie folgt definiert:

> „Stehen bei einem Verbund von Rechensystemen Managementaspekte im Vordergrund, die Ressourcen betreffen, die primär der Kommunikationsmöglichkeit zwischen den Rechensystemen dienen, sprechen wir von *Netzmanagement*. Stehen jedoch Managementaspekte im Vordergrund, die den Betrieb eines Verbundes von Rechensystemen als verteiltes System mit entsprechender verteilter Diensterbringung betreffen, dann sprechen wir von *Systemmanagement*."

Diese Arbeit befasst sich schwerpunktmäßig mit dem Systemmanagement. Natürlich spielt aber auch das Netzmanagement eine Rolle, denn die Verfügbarkeit von Diensten ist schließlich auch von funktionierenden Netzwerkstrecken abhängig.

Das Systemmanagement befasst sich seinerseits laut [intman, S. 101] u.a. mit den folgenden Aufgaben:

1. Verfügbarkeit der Datenbestände auf File-Servern
2. Backup und Archivierung von Datenbeständen
3. I/O auf Datenträgern (optische und magnetische Medien)
4. Spooling (Druckerqueues)
5. E-Mail-Kommunikation
6. Benutzerverwaltung
7. Installation und Konfiguration der Hardware
8. Installation und Konfiguration der Software
9. Überwachung und Gewährleistung der Sicherheit des Systems
10. Fehlermanagement
11. Performance-Tuning

Im Rahmen dieser Arbeit werden nicht alle diese Aspekte des Systemmanagements betrachtet. Aus der vorigen Aufstellung der Aufgaben werden die Punkte 7 (z.B. Konfiguration von Netzwerkkomponenten), 8 (z.B. Konfiguration von Firewalls oder Serverdiensten), 9 (z.B. Verhinderung des Eindringens ins Netzwerk), 10 (z.B. Reparatur von Komponenten) und 11 (z.B. Erhöhung der Bandbreite bei überlasteten Netzwerkstrecken) ausgewählt. Die Punkte 8 und 9 gehören zum Aspekt des *Sicherheitsmanagements*.

Eine *absolute* Sicherheit ist aber auch mit einem sorgfältig durchgeführten Sicherheitsmanagement nicht erreichbar. Die Methoden zur Abwehr neuer Angriffe basieren auf den Erfahrungen aus bisher erfolgten Angriffen. Es ist nur schwer möglich, zukünftige Angriffsmöglichkeiten in einem ständig wechselnden Systemumfeld vorherzusehen und präventiv dagegen vorzugehen. Kein Systemadministrator wird die Zeit finden, die Quellcodes aller seiner eingesetzten Betriebssysteme und Sicherheitsprogramme zu analysieren — sofern diese überhaupt für ihn verfügbar sind. Er wird also auf grundlegende Funktionen und deren Sicherheit vertrauen müssen.

Häufig gibt es geduldete Sicherheitsmängel. Benutzer eines Systems sind durch ihre häufig unsicheren Passwörter oder mangelndes Sicherheitsbewusstsein für die Sicherheit gefährlich. Trotzdem wäre es sinnlos, ihnen deshalb den Zugang zum System zu verweigern, da damit ein normaler Betrieb im Unternehmen nicht mehr möglich wäre. Das hier verfolgte Ziel ist also nicht die *absolute Sicherheit* sondern eine den Umständen entsprechend *optimale Sicherheit*.

Sicherheit wird langfristig geplant und beinhaltet als ihre zentrale Maßnahme die Policy. Sie beschreibt beispielsweise die Konfiguration des Firewall-Regelsatzes, der wiederum von der Firewall durchgesetzt wird. Der Schwerpunkt dieser Arbeit liegt dabei

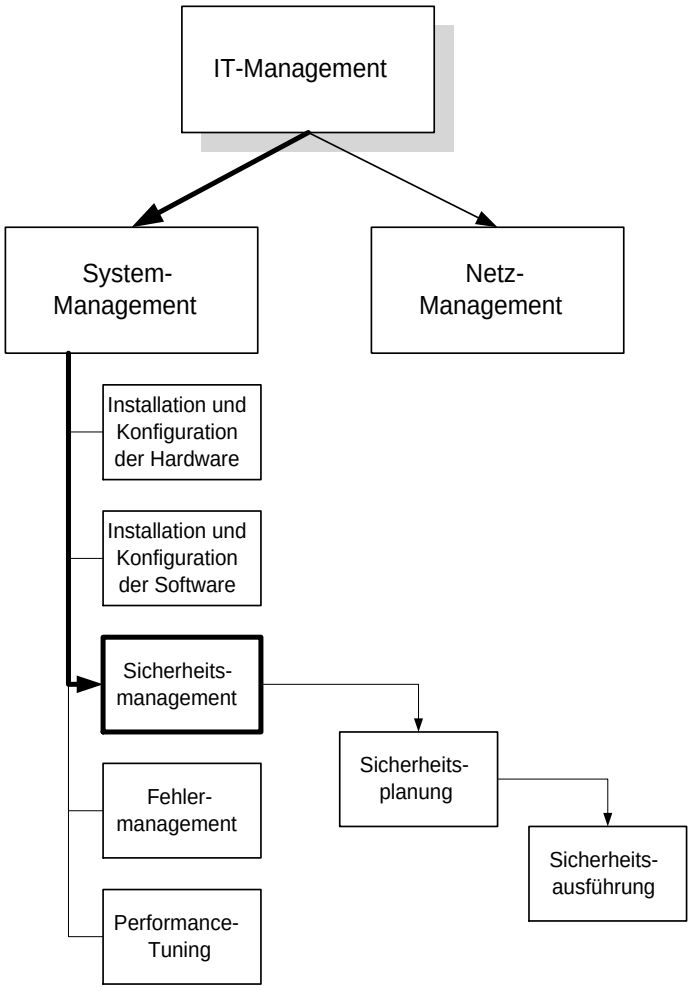

Abbildung 4.1.: Sicherheitsmanagement als Teil des IT-Managements

auf der Sicherheit eines verteilten Netzwerks. Das Modellunternehmen betreibt mehrere räumlich getrennte Niederlassungsnetzwerke. Da diese Netzwerke aber miteinander verbunden sind, stellt sich die Frage nach einem *globalen Sicherheitsmanagement*. Immerhin können Angreifer leichter andere Niederlassungsnetze angreifen, denen der Zugang zu einem Netzwerk gelungen ist. Die integrale Bestandteile des Sicherheitsmanagements sind die *Sicherheitsplanung* und die *Sicherheitsausführung*. Diese Abhängigkeit wird in Punkt 4.1 dargestellt. Die ersten Kapitel dieser Arbeit haben sich mit den Punkten 7 (Konfiguration der Hardware) und 8 (Konfiguration der Software) befasst und beschrieben, welche präventiven Maßnahmen zu einer Erhöhung der passiven Sicherheit führen. Passive Sicherheit ist keine Garantie für eine ausreichende Sicherheit. Selbst eine gut konfigurierte Firewall kann nur die Angriffe abwehren, auf die sie programmiert ist. An dieser Stelle muss die aktive Sicherheit erhöht werden. Eine Firewall ist relativ nutzlos, wenn sie unbemerkt angegriffen werden kann. Es muss hier eine Instanz geben, die Sicherheitsverstöße an der Firewall aufzeichnet und auswertet. Die folgenden Kapitel befassen sich deshalb mit der Implementation einer Sicherheitsüberwachung, die für eine verbesserte aktive Sicherheit sorgen soll.

Zur Vorbereitung der Implementation werden im folgenden die Anforderungen definiert, die an die Sicherheitsüberwachung gestellt werden.

4.1. Anforderungen

Im vorigen Kapitel wurden die Gefahren und möglichen Abwehrmaßnahmen eines internetbasierten Angriffs vorgestellt. Aber selbst, wenn die Risiken einer Kompromittierung so gering wie möglich gehalten werden, bleiben Restrisiken vorhanden. Das Ziel soll aber nicht das Erreichen einer *absoluten* Sicherheit sondern einer *optimalen* Sicherheit sein. Die Grundlage für die Entwicklung eines Sicherheitsüberwachungssystems soll diese optimale Sicherheit sein — also ein möglichst gut abgesichertes Netzwerk, in dem Angriffe zwar möglich sind, aber zumindest erkannt und gemeldet werden sollen.

In diesem Abschnitt sollen nun die Anforderungen an das gewünschte Sicherheitsüberwachungssystem festgelegt werden. Der folgende Abschnitt befasst sich mit der Konzeption, formalisiert diese Anforderungen und sucht nach geeigneten Komponenten für die Implementation, die im dritten Abschnitt im Detail beschrieben wird.

4.1.1. Monitoring

Die automatische Situationsanalyse benötigt zur Entscheidungsfindung ein möglichst genaues Abbild des aktuellen Zustandes des gesamten Unternehmensnetzwerks. Es ist die Aufgabe des Monitoring, diesen Gesamtzustand festzustellen. Dazu gehört die Messung von *permanenten* Systemparametern wie der Verfügbarkeit von Diensten oder der Auslastung von Ressourcen. Aber auch die Überwachung auf *spontane* Ereignisse wie erkannte Komprimittierungen, Portscans oder der Ausfall der Stromversorgung gehört in dessen Bereich. Dabei ist weder die Speicherung noch die Auswertung dieser Informationen Aufgabe des Monitoring. Die einzelnen Instanzen laufen auf jedem überwachten System selbstständig ab und informieren über das Reporting das zentrale Überwachungssystem. Die erlangten Informationen sollen dann an zentraler Stelle gespeichert werden.

Dieser zentralisierte Ansatz stellt vom Sicherheitsstandpunkt aus nicht die beste Lösung dar, weil ein Ausfall des Überwachungssystems den Totalausfall der Sicherheitsüberprüfung zur Folge hätte. Um die Implementation aber nicht zu komplex zu gestalten, wurde dieser vereinfachte Ansatz verfolgt. Die zur Überwachung benötigten Prozesse sollten zumindest von einem anderen System auf ihre Verfügbarkeit geprüft werden, damit die Überwachung nicht unbemerkt ausfallen kann.

Im vorigen Kapitel wurde gezeigt, dass Angreifer viele verschiedene Möglichkeiten nutzen, um in Netzwerke einzudringen. Da sich das Sicherheitsumfeld ständig durch neue Protokolle, Betriebssysteme, Serverprozesse (Daemons) und Anwendungsprogramme ändert, soll die Monitoring-Komponente flexibel bei veränderten Anforderungen anpassbar sein. Es ist kaum vorhersehbar, welche Angriffe in einigen Monaten oder Jahren aktuell sein werden. Auch diese Fälle müssen aber implementiert werden können.

Das Monitoring soll z.B. folgende Überwachungen ermöglichen:

Einbruchserkennung: Systemeinbrüche sollen natürlich soweit möglich durch passive Sicherheit vermieden werden. Sie können aber in der Praxis nicht mit absoluter Sicherheit verhindert werden. Das Sicherheitssystem soll einen Einbruchsversuch zumindest so früh wie möglich erkennen.

Beispiel: Erkennen von Port-Scans, Aufspüren veränderter Systemprogramme oder Protokolldateien, Erkennen veränderter Benutzerkennungen.

Verfügbarkeit von Diensten: Durch gezieltes Abfragen von Diensten soll die Verfügbarkeit angebotener Dienste (z.B. IP-Dienste) geprüft werden. Die Abfrage soll möglichst auf Protokollebene stattfinden.

Beispiel: Kommunikation mit einem Web-Server auf TCP-Port 80 mit Emulation grundlegender Teile des HTTP-Protokolls.

Leistungsüberwachung: Die Leistungsüberwachung prüft variable skalare Parameter wie Auslastungen.

Beispiel: Alarmierung, falls die CPU-Auslastung eines Fileservers über 95% während eines Zeitraums von 5 Minuten liegt.

4.1.2. Reporting

Nachdem jede Einzelkomponente des Monitoring Informationen gesammelt hat, werden diese über das Reporting an das überwachende System weitergegeben. Da diese Übertragung über Niederlassungsgrenzen hinweg stattfindet, muss eine geeignete Absicherung gefunden werden. Die gesammelten Daten werden auf dem Überwachungssystem gespeichert.

4.1.3. Speicherung der Konfigurations- und Prozessdaten

Zur Steuerung des Analyseprozesses müssen Konfigurationsdaten zur Verfügung gestellt werden, die den Ablauf der Situationsanalyse steuern. Z.B. müssen die verschiedenen Überprüfungen vorgegeben werden, anhand derer die Analyse Entscheidungen treffen kann. Es ist sinnvoll, diese Daten nicht als Teil des Programms aufzunehmen,

damit auch während der Laufzeit noch Parameter geändert werden können, ohne den Prozess unterbrechen und neu starten zu müssen.

Außerdem werden im Verlauf der Situationsanalyse verschiedene Daten gesammelt — z.B. über den Status verschiedener Dienste. An dieser Stelle muss eine Möglichkeit gefunden werden, diese Daten zu speichern und auch später wieder auf sie geeignet zugreifen zu können.

Bei der Auswahl eines Systems zur Datenspeicherung darf nicht vergessen werden, dass komplexe Zusammenhänge aus den gespeicherten Daten hergestellt werden müssen, um die Gesamtsituation überhaupt beurteilen zu können. Die Daten sollten also auch möglichst so gespeichert werden, dass sie in beliebiger Form und in verschiedenen Zusammenhängen wieder abgerufen werden können.

Es muss auch berücksichtigt werden, dass die Implementation verschiedene Prozesse verwendet, um seine Aufgabe zu erfüllen. So könnten auch mehrere Instanzen gleichzeitig auf diese Datenbestände zugreifen wollen. Es müssen also Mechanismen zur Serialisierung dieser Zugriffe gefunden oder entwickelt werden, um Inkonsistenzen im Datenbestand zu verhindern.

Ein weiterer wichtiger Aspekt ist auch die hohe Geschwindigkeit der Zugriffe. Die Implementation wird aus mehreren Komponenten bestehen, die ständig Daten abrufen. Alleine der Prozess der Situationsanalyse wird voraussichtlich bei jeder Analyse mehrere tausend Lesezugriffe auf die Daten ausführen.

Folgende Informationen sollen dauerhaft gespeichert werden:

1. Informationen über die zu prüfenden Dienste und Parameter, die dem Monitoring übergeben werden,
2. Ergebnisse des Monitoring (Rohdaten),
3. Ergebnisse der Auswertungslogik zur Status-Anzeige,
4. Verantwortlichkeiten einzelner Dienste mittels administrativer Domänen (damit die entsprechend verantwortlichen Systemadministratoren des IRTs alarmiert werden können) und
5. Alarmierungen (Trouble-Tickets)

4.1.4. Situationsanalyse

Es soll die Aufgabe einer weiteren Instanz sein, die vom Monitoring gesammelten und in der Datenbank gespeicherten Rohdaten über die Gesamtsituation des Netzwerks auszuwerten. Die Situation soll periodisch anhand vorgegebener Entscheidungskriterien analysiert werden. Um eine große Flexibilität bei der Entscheidungsfindung zu ermöglichen, sollen verschiedene Algorithmen (sogenannte Alarmmodule — vgl. Kapitel 4.2.3) zur Verfügung stehen, die häufig benötigte Überwachungen anbieten. An anderer Stelle werden die zu überprüfenden Dienste festgelegt. Zu jeder verlangten Prüfung wird das passende Alarmmodul aufgerufen, um über seinen Teilbereich zu entscheiden. Es soll jederzeit leicht möglich sein, eigene Alarmmodule in das bestehende System zu integrieren.

Das System soll zwei Arten von Alarmen auslösen können: ereignisbasierte und zustandsbasierte Alarme. Ein ereignisbasierter Alarm basiert auf einem einzelnen Ereignis beliebiger Art. Dies kann z.B. ein von einem IDS erkannter Angriff sein. Im Gegensatz dazu beruhen zustandsbasierte Alarme auf einem reversiblen Zustand wie z.B. dem Ausfall einer Komponente. Ist diese Komponente wieder verfügbar geworden, so wäre der Alarmzustand damit beendet und der Alarm könnte widerrufen werden. An dieser Stelle sollen einige Beispiele die Unterscheidung dieser beiden Alarmtypen veranschaulichen, wobei zu beachten ist, dass die Informationen, auf denen die Alarme basieren, vom Reporting in das System eingebracht werden und somit nicht Teil der Analyse sind. Die Analyse wertet lediglich die Gesamtsituation aus, die sich aus der Summe der Informationen ergibt, die verschiedene Reporting-Instanzen (wie z.B. IDS) gesammelt haben.

- Erkennen anormaler Netzwerkereignisse (Ausgehende Pakete aus der demilitarisierten Zone ins LAN, Scanning der Firewall)
  $\longrightarrow$ ereignisbasierter Alarm
- Veränderung der Konfiguration sicherheitskritischer Systeme (Veränderung der Firewall-Regeln, Benutzeraccounts oder der Systemkonfiguration)
  $\longrightarrow$ ereignisbasierter Alarm
- Überwachung physikalischer Parameter (Abfrage der CPU-Temperatur mittels SNMP, um die Verfügbarkeit zu gewährleisten)
  $\longrightarrow$ zustandsbasierter Alarm
- Störungen im Betrieb komplexer Serverdienste (Lotus Notes-Groupware-Server mit mehreren Kommunikationsdiensten)
  $\longrightarrow$ zustandsbasierter Alarm

4.1.5. Alarmierung

Entscheidet die Situationsanalyse, dass die Situation eine Alarmierung erfordert, dann müssen die verantwortlichen System-Administratoren darüber informiert werden. Der Meldeweg muss sorgfältig ausgewählt werden, damit die Alarmierung auch beim Ausfall wichtiger Komponenten noch erfolgen kann. Es ist beispielsweise sinnlos, den Administrator mittels E-Mail darüber zu informieren, dass das E-Mail-System gestört oder sogar ausgefallen ist — er würde diese Nachricht nicht erhalten. Der Meldeweg muss so kurz wie möglich gehalten werden und soll nicht über überwachte Systeme verlaufen.

Denkbar ist eine Alarmierung über den Kurznachrichtendienst (*Short Message Service* bzw. *SMS)*) der GSM-Funknetze. Eine mögliche Erweiterung dieser Alarmierung wäre die mobile interaktive Kommunikation des Administrators mit dem System z.B. über das *Wireless Application Protocol* (WAP).

4.1.6. Trouble-Ticketing

Die Situationsanalyse soll für jeden Alarmfall ein Trouble-Ticket erzeugen. Ein Administrator kann dieses Ticket annehmen und zeigt sich damit verantwortlich für die

Beseitigung des alarmierten Problems. Alle anderen Administratoren derselben administrativen Domäne (die ebenfalls über dieses Trouble-Ticket alarmiert wurden) erhalten in diesem Fall die Information, dass das Problem bereits bearbeitet wird. Erkennt die Auswertungslogik, dass das Problem nicht länger existiert, storniert es das Trouble-Ticket selbstständig.

Es soll auch der Fall ereignisbasierter (einmaliger) Alarmierungen berücksichtigt werden. Ein ereignisbasierter Alarm wird von einem Ereignis wie z.B. einem Portscan-Angriff ausgelöst. Jeder ereignisbasierte Alarm kann vom Systemadministrator bestätigt werden. Im Gegensatz dazu können zustandsbasierte Alarme automatisch storniert werden, sofern die Situation, die diesen Alarm ausgelöst hat, sich wieder normalisiert hat. Ein Beispiel wäre der Ausfall eines Netzwerkdienstes. Sobald dieser Dienst wieder zur Verfügung steht, sollte das System erkennen, dass der Alarm nicht mehr aktuell ist und das entsprechende Trouble-Ticket stornieren.

4.1.7. Bedienschnittstelle

Das Überwachungssystem soll nicht nur einseitig die zuständigen Systemadministratoren des IRT über auftretende Probleme informieren, sondern auch eine Kommunikation der Administratoren mit dem System erlauben. Es soll jedem Administrator möglich sein, sich eine Übersicht über den Gesamtstatus des Systems einzuholen und mit dem System zu interagieren. Für einen derartigen Zugang zum System muss außerdem ein geeigneter Authentisierungsmechamismus gefunden werden.

Diese Funktionen sollen den Administratoren angeboten werden:

Anzeige des quantitativen Gesamtstatus: Wieviele Alarme sind im System aufgetreten? Wann wurde die Situation zuletzt überprüft? Wieviele Trouble-Tickets sind noch offen?

Details der Alarme: Welche Alarme stehen zur Zeit an? Wodurch wurden diese Alarme verursacht? Wer wurde über diese Alarme informiert? Wie ist der aktuelle Stand der Bearbeitung?

Übersicht der überwachten Systeme: Auf welchen Systemen stehen momentan Alarme an? Wieviele Alarme wurden ausgelöst?

Anzeige der Trouble-Tickets: Welche Trouble-Tickets sind offen? Aufgrund welcher Überprüfungung wurden sie erzeugt? Wie ist der Stand der Bearbeitung? Welchem Systemadministrator sind sie verantwortlich zugewiesen?

Logbuch: Welche Ereignisse wurden zuletzt im System registriert? Z.B. Übernahme eines Trouble-Tickets oder Auslösen eines Alarms.

Verwaltung administrativer Daten: Auswahl der Benachrichtigungsoptionen für jeden einzelnen Systemadministrator. Möglichkeit der Wahl zwischen E-Mail und SMS. Änderung des Zugangspasswortes.

Wartungsfunktionen: Ausgewählte Überprüfungen sollen temporär von der Überwachung ausgeschlossen werden können, damit geplante Wartungsarbeiten nicht zu Alarmen führen.

4.2. Netzwerk-Sicherheits-Policy

4.2.1. Bedeutung der Policy

Die Sicherheits-Policy bildet als Regelwerk die Grundlage für die Konfiguration und Überwachung von Systemen im Netzwerk. Sie soll die Integrität des Netzwerks sicherstellen und soll helfen, Risiken und Verluste durch Bedrohungen und den Schaden durch Angriffe auf die Funktionalität des Netzwerkes zu mindern. Dabei basiert sie auf klassischen Sicherheitsmodellen (s. S. 11) und auf Erfahrungen und Empfehlungen anderer Netzwerkadministratoren, die bereits ihre eigene Policy verfasst haben. Allerdings muss sie ständig erweitert und revidiert werden, um aktuelle Risiken und Gefahren zu erfassen und um ihnen entgegenzuwirken. Als Ergebnis beinhaltet die Policy hauptsächlich diese Aspekte:

- eine Vorschrift der Konfiguration aller Netzwerkkomponenten, um eine optimale passive Sicherheit zu erreichen,
- die Verantwortlichkeiten für alle Bereiche, die die IT-Sicherheit betreffen,
- Hilfen zum Verhalten im Falle eines erfolgreichen Einbruchs und
- allgemeine Verhaltensmaßregeln zur geeigneten Benutzung für alle Benutzer, um einen absichtlichen oder unabsichtlichen Missbrauch der IT-Ressourcen zu vermeiden.

4.2.2. Beschreibungsformen

Meistens findet man Policys als verbal-textuelle Regelwerke. Allerdings gibt es auch Ansätze wie in [intman] beschrieben, wo der Versuch unternommen wird, eine Policy zu formalisieren. Die dort beschriebene *Network Access Policy (NAP)* definiert, von welchen Ausgangspunkten welche Dienste zu welchen Endpunkten erlaubt sind. Dies wird durch eine Relation formalisiert:

$$\text{NAP} \subseteq \underbrace{\mathcal{U} \times \mathcal{C}}_{\text{Ausgangspunkt}} \times \mathcal{S} \times \underbrace{\mathcal{U} \times \mathcal{C}}_{\text{Endpunkt}}$$

Die Relation ist ein Kreuzprodukt der Mengen der Benutzer $\mathcal{U}$ (users), der Komponenten $\mathcal{C}$ (components) und der Dienstes $\mathcal{S}$ (services). Könnte man die gesamte Policy auf diese Relation abbilden, so hätte man die Möglichkeit, die Einhaltung der Policy automatisch zu überwachen.

4.2.3. Abbildung der Policy auf die konzipierten Datenstrukturen

Analog zum Ansatz von [intman] ist man bei der Implementation darauf angewiesen, ein maschinenlesbares Format bei der Beschreibung der erwünschten und unerwünschten Aktionen einzuhalten. Die Situationsanalyse benötigt diese Informationen, um algorithmisch beurteilen zu können, ob ein Sicherheitsverstoß vorliegt. Bei der Festlegung eines geeigneten Formates stellte sich schnell heraus, dass zwar die

Anfangs- und Endpunkte einer Aktion leicht definierbar sind (z.B. durch den Hostnamen), aber die Art der Überprüfung lässt sich formal schwer erfassen. Deshalb wurde das Konzept der *Alarmklassen* und *Alarmmodule* eingeführt.

Eine Alarmklasse gibt die Art der Information an, die ein Monitoring-Modul liefert. Ein Alarmmodul ist ein Programmmodul, das Informationen einer oder mehrerer Alarmklassen analysiert. Zum Verständnis ein Beispiel:

Die Policy gibt vor, dass auf dem Fileserver `olymp` die Füllung der Partition für die Homeverzeichnisse der Benutzer nie über 90% liegen darf. Das Monitoring sammelt ständig Informationen über Plattenfüllungen und stellt sie der Situationsanalyse in der Alarmklasse `DISK` zur Verfügung. Diese Information könnte so aussehen:

Prüfsystem	System	Alarmklasse	Parameter	Status
olymp	olymp	DISK	/home	87

Natürlichsprachlich bedeutet diese Information: „Das System `olymp` hat festgestellt, dass die Füllung der Partition `/home` auf dem System `olymp` 87% beträgt.“ Dabei wird neben dem geprüften System auch noch das prüfende System angegeben. In anderen Situationen kann es sinnvoll sein, denselben Dienst von mehreren Systemen auch zu überprüfen (z.B. im Fall der Verfügbarkeitsüberwachung von Netzwerkstrecken). In diesem Fall aber überprüft sich der Fileserver selbst und sendet diese Information an die Situationsanalyse.

An einer anderen Stelle wäre die Policy für die Analyse formal definiert. Ein Tupel der Policy-Relation könnte so aussehen:

System	Alarmmodul	Parameter
olymp	DISK	/home,90

Diese Regel würde bedeuten: „Die Füllung der Partition `/home` auf dem System `olymp` darf nie über 90% liegen.“ In diesem einfachen Beispiel ist die Analogie zwischen Monitoring-Information und Policy-Regel klar zu erkennen. Das Alarmmodul `DISK` muss nicht mehr tun, als die entsprechende Monitoring-Information zu der gewünschten Überprüfung abzufragen und den Wert 87 mit dem Wert 90 zu vergleichen. Es sind aber auch sehr komplexe Alarmmodule denkbar, die z.B. die Verfügbarkeit und Antwortzeit eines Webservers von verschiedenen Prüfsystemen aus überwachen. In so einem Fall würde das Alarmmodul mehrere Informationen vom Monitoring auswerten.

4.3. Konzeption und Vorentscheidung

Nachdem bereits im Abschnitt 4.1 die Anforderungen an eine Sicherheitsüberwachung gestellt wurden, sollen jetzt geeignete Komponenten zur Implementation gefunden werden.

4.3.1. Monitoring

Die Anforderungen an das Monitoring verlangen ein flexibles Konzept, um auf vielen UNIX-Systemen selbstständig Monitoring-Informationen zu sammeln. An dieser Stel-

le bietet sich z.B. das Softwarepaket PIKT (*gesprochen: Picket*) an, das bereits einen großen Teil dieser Arbeit leistet. PIKT (vgl. [pikt]) ist ein freies Softwarepaket, das an der Universität Chicago von Robert Osterlund entwickelt wurde, um die Systemadministration in großen heterogenen UNIX-Netzwerken zu unterstützen. Es bietet eine eingebaute Skriptsprache mit einem Präprozessor und beinhaltet bereits eine Sammlung von Modulen, die häufig wiederkehrende Wartungs- und Überwachungsaufgaben erledigen. So können Dateirechte automatisch überwacht und korrigiert, Mount-Verbindungen von Netzwerkdateisystemen (NFS) wieder aufgebaut, fehlerhafte Mail-Weiterleitungen aufgespürt, Ausführungen des SU-Befehls erkannt, modifizierte oder überlange Protokolldateien gefunden, überlastete Systeme festgestellt, unautorisierte System-Neustarts oder überfüllte Festplattepartionen erkannt werden.

Um der Heterogenität der Systeme zu begegnen, beinhalten die einzelnen PIKT-Module verschiedene Programmteile, die je nach UNIX-Derivat ausgeführt werden. In den Basismodulen unterstützt PIKT Linux, Solaris, FreeBSD, OpenBSD, HP-UX, Irix und AIX. Die Unterscheidung der Systemplattformen erfolgt durch Defines und entsprechende Präprozessoranweisungen wie „`#ifdef openbsd`“. Mit den angebotenen Basismodulen werden bereits einige Parameter überwacht, die für die Sicherheitsüberwachung hilfreich sind. Weitere Module lassen sich zwar selbst mit vertretbarem Aufwand in der PIKT-Skriptsprache entwickeln, dennoch ist der Leistungsumfang der Sprache sehr auf die Ausführung von Systemprogrammen wie `ps`, `df` oder `netstat` und die Überwachung des Dateisystems beschränkt. Für komplexere Anwendungsfälle lassen sich aber Perl-Skripte in PIKT integrieren. Da Perl-Skripte sehr portabel sind, lassen sie sich unverändert auf nahezu jedem UNIX-System einsetzen.

Eines der Hauptmerkmale von PIKT ist die automatische Verteilung der Module über das Netzwerk. Auf dem Zielsystem muss lediglich das PIKT-Grundsystem installiert sein und ein entsprechender Dienst vorhanden sein, um Befehle vom Hauptsystem (dem *PIKT-Master*) entgegennehmen zu können. Die einzelnen Systeme sind in der Lage, an beliebiger Stelle Informationen über E-Mail oder Syslog an andere Systeme zu übermitteln.

Bisher wurde immer nur von UNIX-Derivaten gesprochen. Natürlich werden in der Praxis weitere Plattformen als nur UNIX-Systeme eingesetzt. Im Client-Bereich trifft man vorwiegend auf Windows-Systeme. Diese Arbeit betrachtet aber lediglich die Sicherheitsaspekte im Server-Bereich des betrachteten Modellunternehmens und damit die spezifischen Probleme von UNIX-Servern.

Bei der Implementation können auch andere Monitoring-Komponenten eingesetzt werden. Einige Firewalls erlauben die Ausgabe von Protokoll-Informationen in vorgegebenen Formaten oder zumindest die Ausführung von Kommandozeilenbefehlen. Erkennt eine Firewall z.B. einen Portscan, so kann diese Information ebenfalls an das Überwachungssystem weitergegeben werden. Natürlich lassen sich an dieser Stelle auch vorhandene andere Systeme — insbesondere Intrusions-Erkennungs-Systeme (IDS) — wie Snort, Tripwire oder AIDE integrieren. In dieser Arbeit werden aber lediglich bestehende externe Reporting-Instanzen konfiguriert und eingebunden.

4.3.2. Reporting

Es wurde bereits gefordert, dass das Monitoring Informationen sammelt, die über das Reporting an das zentrale Überwachungssystem übertragen und in der zentralen Da-

tenbank gespeichert werden.

Wie bereits erwähnt, erlaubt PIKT die Ausgabe von Informationen über die erledigten Tätigkeiten auch über den Syslog-Mechanismus. Syslog ist nicht nur in der Lage, Protokolldaten auf dem lokalen System nach vorgegebenen Kriterien in verschiedenen Dateien zu speichern, sondern kann diese auch auf andere Systeme übertragen. Syslog überträgt Informationen in diesem Fall allerdings unverschlüsselt. Bei einer Kommunikation zwischen Niederlassungen über das Internet, wäre das sicherheitstechnisch nicht vertretbar. Allerdings wird festgelegt, dass beim Modellunternehmen alle Niederlassungen über virtuelle private Netzwerke (VPN) untereinander kryptografisch gesichert übertragen können.

Auf der Seite des zentralen Überwachungssystems muss es nun eine Instanz geben, die diese Syslog-Informationen entgegennimmt und speichert. Da PIKT den größten Teil des Monitoring ausmachen wird, orientiert sich die Syntax am vorgegebenen PIKT-Syslog-Format. Eine Protokollzeile könnte so aussehen:

```
Sep 1 01:18:16 src@dathon pikt[123]: EMERG: 968458696 mcp magrathea
PORT (smtp,25) 'ok' 'TCP port 25 is working properly on SMTP'
```

Bei diesem Format werden diese Angaben in der Datenbank gespeichert:

`968458696` Anzahl Sekunden seit dem 1. Januar 1970 (UTC) — dem Beginn der UNIX-Zeitzählung.

`mcp` Name des Systems, welches die Monitoring-Information angefordert hat.

`magrathea` Name des Systems, auf welches sich diese Information bezieht.

`PORT` Name der Alarmklasse, um die sematische Bedeutung der Information festzulegen. Informationen derselben Alarmklasse haben immer dieselbe semantische Bedeutung. Hier beziehen sich die Informationen auf die Verfügbarkeitsüberwachung eines TCP-Ports.

`(smtp,25)` Parameter der Alarmklasse. Hier geben sie an, dass der TCP-Port 25 auf die Verfügbarkeit eines SMTP-fähigen Serverdienstes geprüft wurde.

`'ok'` Status-Beurteilung der Information. An dieser Stelle ist kein Problem aufgetreten. In einer anderen Alarmklasse zur Überwachung der Festplattenfüllung könnte an dieser Stelle z.B. `good` oder `full` stehen.

`'TCP port 25 is working properly on SMTP'` Eine Freitextmeldung des Monitoring. Die Situationsanalyse kann diesen Text später verwenden, um weitere Informationen anzuzeigen. In der Regel reicht aber die vorige Angabe des Status aus.

4.3.3. Speicherung der Konfigurations- und Prozessdaten

Die Anforderungen an die Datenhaltung könnten durch mehrere Möglichkeiten erfüllt werden. Die Daten könnten einfach in Textdateien gespeichert werden. Diese Methode

ist einfach zu implementieren, verhindert aber nicht, dass mehrere Instanzen gleichzeitig auf die Dateien zugreifen möchten. Außerdem ist zu erwarten, dass die Geschwindigkeit der Zugriffe eher gering ist und dass leichter Fehler passieren, da jeder Prozess die Dateien parsen muss.

Alternativ ist auch möglich, ein bestehendes Datenbankmanagementsystem (DBMS) zu verwenden. In diesem sind Mechanismen zur Serialisierung der Zugriffe und Schutzmechanismen wie die Authentisierung bereits vorhanden. Um das Rad nicht neu zu erfinden, bietet sich bei der Datenhaltung ein fertiges DBMS an. Insbesondere reicht hier eine relationale Datenbank (RDBMS) aus. Ein genereller Nachteil relationaler Datenbanksysteme ist die vergleichsweise geringe Abfragegeschwindigkeit bei komplexen Abfragen, die die Informationen aus verschiedenen Tabellen verknüpfen. Der Analyseprozess wird aber nur Informationen aus einzelnen Tabellen abfragen, womit diese Einschränkung nicht weiter ins Gewicht fällt.

Ein RDBMS lässt sich durch die standisierte Abfragesprache SQL einfach abfragen. Die Situationsanalyse müsste selbst relativ wenige Anfragen an die Datenbank stellen; alleine durch die adäquate Formulierung von SQL-Abfragen könnte man bereits komplizierte Beziehungen zwischen den gespeicherten Daten direkt vom RDBMS analysieren lassen. Die Vorteile, die ein bestehendes RDBMS bietet, führen zu der Entscheidung, dieses System zur Datenhaltung einzusetzen.

Wichtig für das Design der Datenbank ist auch das gespeicherte Datenvolumen. In ersten Tests aufgrund bestehender Sicherheitssysteme konnten diese Zahlen geschätzt werden:

- Es werden etwa 200 unterschiedliche Überprüfungen von Systemparametern vorgenommen.
- Diese Parameter werden in etwa 100 algorithmischen Überprüfungen analysiert, um Anomalien im Netzwerk feststellen zu können.
- Die bestehenden Systeme erzeugen in einem Jahr etwa 400 Alarme.
- Das IRT besteht aus etwa 10 Mitarbeitern.

Diese Quantitäten müssen mit der Größe eines einzelnen Datensatzes multipliziert werden, um die Größe der Gesamtdatenbank zu erhalten. Deshalb sollen jetzt die benötigten Datenbanktabellen definiert werden. Im weiteren Verlauf dieses Textes werden Datenbanktabellen in einem *Kasten* gesetzt. Für alle Tabellen gilt, dass die Spalte `id` immer ein primäres Indexfeld ist, um einen schnellen Zugriff auf Einträge anhand der Identifikationsnummer zu erhalten.

Administrative Domänen: admdom

Die Verantwortlichkeiten der Systemadministatoren für bestimmte Dienste und Sicherheitsbeschränkungen, die durch die *checks* definiert werden, werden in der Tabelle *admdom* (*administrative Domänen*) festgelegt. Diese Tabelle ist folgendermaßen aufgebaut:

Feldbezeichner	Feldart	Bedeutung

id	Zahl	Fortlaufende Nummerierung der Einträge. (Index-Feld, automatische Inkrementierung)
domain	Text	Name der administrativen Domäne.
admins	Text	Durch Kommas getrennte Liste der Systemadministratoren, die für diese Domäne verantwortlich sind.
info	Text	Textuelle Beschreibung dieser Domäne.

Systemadministratoren: admins

In der Tabelle *admins* werden Informationen über die einzelnen Systemadministratoren sowie Informationen über die Alarmierungswege gespeichert.

Feldbezeichner	Feldart	Bedeutung
id	Zahl	Fortlaufende Nummerierung der Einträge. (Index-Feld, automatische Inkrementierung)
name	Text	Username des Administrators.
realname	Text	Vor- und Nachname des Administrators.
password	Text	Verschlüsseltes Zugangspasswort (UNIX-Crypt-Verschlüsselung).
gsmno	Text	Rufnummernkennung des GSM-Mobiltelefons dieses Administrators.
gsmjammed	Enum(y,n)	Kennzeichen, ob dieser Administrator zur Zeit keine weiteren Nachrichten an sein Mobiltelefon bekommen soll (s.u.).
email	Text	E-Mail-Adresse des Administrators.
amethod	Set(sms,email)	Alarmierungsweg an den Administrator. Wahlweise E-Mail oder SMS.

Überprüfungen: checks

Die Situationsanalyse erledigt ihre Arbeit anhand einer Liste von Überprüfungen. Diese Informationen stehen in der Tabelle *checks* und bestehen aus dem aufgerufenen Alarmmodul, verschiedenen Parametern und Angaben über die Zuständigkeiten (administrative Domäne), erfolgte Alarmierungen (Trouble-Tickets) und die Steuerung von Alarmzeitpunkten.

Feldbezeichner	Feldart	Bedeutung
id	Zahl	Fortlaufende Nummerierung der Einträge. (Index-Feld, automatische Inkrementierung)
src	Text	Name des Rechensystems, das die Überprüfung vornehmen soll.
dst	Text	Name des überprüften Rechensystems.

amodule	Text	Name des Alarmmoduls, das anhand der gesammelten Situationsdaten in der Tabelle *events* eine Entscheidung über eine nötige Alarmierung treffen soll.
apar1,apar2,apar3	Text	Parameter, die dem Alarmmodul mit übergeben werden, um die Überprüfungen zu spezifizieren.
aparams	Text	Textuelle Beschreibung der übergebenen Parameter.
status	Enum(unknown, ok, warn, alert)	Letztes Ergebnis dieser Überprüfung. Dieses Feld wird nach der Rückkehr aus dem Alarmmodul gesetzt und gibt den aktuellen Status an.
info	Text	Textuelle Beschreibung des aktuellen Status.
admdomain	Text	Administrative Domäne. Bei einer Alarm-Entscheidung des Alarmmoduls werden die entsprechenden Systemadministratoren dieser Domäne benachrichtigt.
ticketid	Zahl	Falls bereits ein Alarm ausgelöst wurde, so wird die Identifikationsnummer des Trouble-Tickets hier eingetragen. Steht hier eine Zahl größer als Null, so steht momentan ein Alarm an.
alertdelay	Zahl	Entscheidet das Alarmmodul über einen Alarm, so wird die Alarmierung verzögert, so dass kurze Störungen im Betrieb zwar aufgezeichnet, aber nicht alarmiert werden. (Angabe in Minuten)
maintenance	Enum(y,n)	Wird dieses Feld auf y gesetzt, so werden alle Entscheidungen der Alarmmoduls ignoriert. Ein Systemadministrator kann eine Überprüfung auf Wartung setzen, um Alarmierungen während geplanter Wartungsarbeiten zu verhindern.
alerttimes-monday	Set(0-24)	Falls Dienste zu festen Zeiten nicht zur Verfügung stehen (z.B. Anhalten eines Datenbanksystems zur Datensicherung), so können hier Zeiten definiert werden, zu denen eine Alarmierung nur möglich ist.
⋮		
alerttimes-sunday	Set(0-24)	

Monitoring-Informationen: events

Verschiedene Monitoring-Instanzen sammeln Informationen über die aktuelle Situation im Netzwerk und speichern diese letztendlich in der Tabelle *events*. Die Situationsanalyse zieht diese Information heran, um über eine Überprüfung aus den *checks* zu entscheiden.

Feldbezeichner	Feldart	Bedeutung
id	Zahl	Fortlaufende Nummerierung der Einträge. (Index-Feld, automatische Inkrementierung)
src	Text	Ursprung dieser Monitoring-Information. Das hier angegebene Rechensystems hat diese Information an die Datenbank weitergegeben.
dst	Text	Ziel dieser Monitoring-Information. Die Daten betreffen das hier angegebene Rechensystem.
time	Timestamp	Zeitpunkt der Informationsübermittlung.
comment	Text	Freitextmeldung des Monitoring.
aclass	Text	Diese Klasse gibt die Art der Überprüfung an. Die Situationsanalyse kann gezielt Informationen einer Alarmklasse zur Entscheidungsfindung heranziehen.
apar1,apar2,apar3	Text	Weitere Parameter, die klassenspezifisch angeben, worauf sich die gewonnenen Informationen genau beziehen.
status	Text	Ergebnis des Monitoring. Dieser Wert bezieht sich direkt auf die Klasse.
processed	Enum(y,n)	Dieses Feld wird lediglich für ereignisbasierte Überprüfungen verwendet. Hat die Situationsanalyse ein Ticket für dieses Ereignis erzeugt, markiert es dies über dieses Flag.

Protokolldatei: logs

Während der Testphase des Prototypen zeigte sich, dass die Abläufe in der Situationsanalyse wenig transparent sind. Es existieren zwar Methoden zur gezielten Ausgabe von Informationen zur Fehlerbehebung (Debugging), die aber nicht ständig aufgezeichnet werden. Um den Systemadministratoren diese Betriebsinformationen zugänglich zu machen, werden relevante Vorgänge in der Tabelle *logs* dokumentiert.

Feldbezeichner	Feldart	Bedeutung
id	Zahl	Fortlaufende Nummerierung der Einträge. (Index-Feld, automatische Inkrementierung)
time	Timestamp	Zeitpunkt des Ereginisses
text	Text	Freitextmeldung

SMS-Versandwarteschleife: smsqueue

Das Kommunikationsmodul für den Versand von SMS-Nachrichten benutzt die Tabelle *smsqueue* zur Speicherung der ausgehenden Nachrichten. Diese Einträge werden nacheinander an die angegebenen Empfänger versendet und aus dieser Tabelle gelöscht. Die Perl-Hilfsklasse `MrDB` verfügt über eine Methode `spool_sms`, um Kurznachrichten in diese Versandwarteschleife einzureihen.

Feldbezeichner	Feldart	Bedeutung
id	Zahl	Fortlaufende Nummerierung der Einträge. (Index-Feld, automatische Inkrementierung)
recipient	Text	Name des Administrators (laut der Tabelle *admins*, der diese Nachrichten erhalten soll.
message	Text	Freitextmeldung

Trouble-Ticket: tickets

Trifft die Situationsanalyse auf einen Alarmfall, so wird ein Trouble-Ticket erzeugt und in der Tabelle *tickets* gespeichert. Die Identifikationsnummer des Tickets merkt sich das System zusätzlich in der Tabelle *checks* im Feld `ticketid`.

Feldbezeichner	Feldart	Bedeutung
id	Zahl	Fortlaufende Nummerierung der Einträge. (Index-Feld, automatische Inkrementierung)
time	Timestamp	Zeitstempel, wann das Ticket erzeugt wurde
admin	Text	Name des Systemadministrators, der dieses Trouble-Ticket übernommen hat. Zunächst ist dieses Feld leer.
status	Enum(alerted, taken, deferred, cancelled, completed)	Zustand des Trouble-Tickets (s.u.)
info	Text	Freitext, der den Status des Trouble-Tickets beschreibt. Der zuständige Systemadministrator kann diese Meldung selbst ändern, um den Stand seiner Arbeit zu beschreiben.
cause	Text	Vom Alarmmodul erzeugte Textmeldung, die die Ursache des Problems beschreiben soll.
lastnotified	Text	Durch Kommas getrennte Liste derjenigen Administratoren, die zuletzt über eine Statusänderung bezüglich dieses Trouble-Tickets informiert wurden.

Das Status-Feld kann fünf verschiedene Zustände beinhalten, die dem Trouble-Ticket die folgende Bedeutung geben:

alerted: Die Situationsanalyse hat dieses Trouble-Ticket automatisch erzeugt. Bislang wurde die Lösung des Problems noch keinem Systemadministrator zugewiesen. Das Feld `admin` ist noch leer. Es wurden alle Administratoren der zugehörigen administrativen Domäne alarmiert.

taken: Ein Systemadministrator hat sich dieses Trouble-Ticket selbst zugewiesen und arbeitet an einer Lösung des Problems.

deferred: Da eine sofortige Lösung des Problems nicht möglich scheint, hat der zugewiesene Systemadministrator die Lösung verzögert. Der Alarm steht weiterhin an. Das Feld `info` sollte eine aussagekräftige Beschreibung der Problematik enthalten.

cancelled: Das verursachende Problem ist nicht mehr existent. Die Situationsanalyse hat deshalb dieses Trouble-Ticket widerrufen.

completed: Dieser Zustand ist nur bei ereignisbasierten Prüfungen möglich. Ein Systemadministrator hat das Ereignis zur Kenntnis genommen.

4.3.4. Situationsanalyse

Der zentrale Prozess im gesamten Überwachungssystem ist die Situationsanalyse. Ihre Aufgabe ist es, die gewünschten Überprüfungen laut der Einträge in den *checks* vorzunehmen. Zu jeder Überprüfung wird das entsprechende Alarmmodul aufgerufen. Das Alarmmodul wertet die Gesamtsituation im Netzwerk aus, über die es Informationen aus der Tabelle *events* beziehen kann. Dieser Durchlauf aller Überprüfungen soll periodisch ablaufen. Eine Synchronisierung mit anderen Komponenten ist nicht sinnvoll, da durch die verteilten Monitoring-Instanzen kein definiertes Ende einer Monitoring-Phase erkennbar ist. Es ist natürlich zu empfehlen, wichtige Dienste vom Monitoring entsprechend häufiger überprüfen zu lassen, um die Reaktionszeiten zu reduzieren.

Bei anderen Komponenten wie dem Monitoring oder der Datenbank wäre es nicht sinnvoll, eigene Komponenten zu entwickeln. SQL-Datenbanken wie MySQL erfüllen alle Anforderungen, die in diesem Falle an ein Datenbanksystem gestellt werden. Im Falle der Situationsanalyse wäre es aber nicht zweckmäßig, eine vorgefertigte Software zu verwenden. Immerhin wurde in den Anforderungen an die Implementation eine hohe Flexibilität gefordert. Es erscheint zweckmäßiger, eine Programmiersprache einzusetzen. Es wurde bereits erwähnt, dass die Skriptsprache Perl (*Practical Extraction and Reporting Language*) zur Implementation der Situationsanalyse verwendet werden soll, um einen guten Kompromiss aus Flexibilität, Übersichtlichkeit und Geschwindigkeit zu erzielen.

> Perl is a language optimized for scanning arbitrary text files, extracting information from those text files, and printing reports based on that information. It's also a good language for many system management tasks. The language is intended to be practical (easy to use, efficient, complete) rather than beautiful (tiny, elegant, minimal).

(Perl Manpage)

Perl ist eine sehr flexible und portable Interpretersprache[1], die häufig bei der Bewältigung von Systemmanagementaufgaben eingesetzt wird. Sie ist frei verfügbar und bietet sich dank ihrer Portabilität besonders zum Einsatz in heterogenen Netzen an. Ihr Design lehnt sich stark an UNIX-Systemprogramme wie `sed` und `awk` und an Standard-Shells an. Perl ist nicht typisiert und verfügt über eine dynamische Speicherzuteilung, so dass sich der Programmierer nicht mit typischen Verwaltungsaufgaben wie manueller Speicherzuteilung befassen muss. Es bietet aber eine große Anzahl optionaler Sicherheitsmechanismen, um Programmierfehler zu vermeiden. Dennoch erzieht es den Programmierer nicht, übersichtlichen Programmcode zu schreiben. Deshalb wurde bei der Implementation großer Wert auf ein sauberes Design gelegt.

Seit der Version 5 erlaubt Perl durch die Einführung der objektorientierten Programmierung eine gute Kapselung von Programm- und Datenstrukturen. Ein Systemadministrator wird zukünftig leichter Überwachungsmodule erstellen können, wenn er sich nicht mit den Details der Implementation auseinandersetzen muss. Das einzelne Alarm-Modul soll möglichst wenig Verwaltungsaufgaben übernehmen, sondern vielmehr aus den erhaltenen Daten über Störungen, Gefahren, Risiken und Ausfälle entscheiden. Wie bereits im vorigen Abschnitt erwähnt, bietet das verwendete RDBMS MySQL auch ein API[2] zu Perl an, mit dem Perl-Skripte einfach auf MySQL-Datenbank zugreifen können.

Die Situationsanalyse soll zur Laufzeit periodisch eine Folge von Überprüfungen anstoßen. Diese Überprüfungen werden über den Namen des auswertenden Moduls und weitere Parameter aufgerufen. So kann der Entscheidungsprozess in *Alarmmodule* gekapselt werden. Für wiederkehrende Überprüfungen können dann diese Module in mehreren Prüfungen (mit verschiedenen Aufrufparametern) eingesetzt werden.

4.3.5. Alarmierung

Die Anforderungen verlangen einen möglichst direkten Kommunikationsweg zwischen dem Überwachungssystem und den verantwortlichen Systemadministratoren. Außerdem soll eine bidirektionale Kommunikation möglich sein.

In der Implementation wird zur Alarmierung der Weg über den SMS (*Short Message Service*) der GSM-Funknetze benutzt. Dieser Kurznachrichtendienst ist ein weltweit akzeptierter Standard zur ungesicherten Übermittlung von alphanumerischen Nachrichten bis zu 160 Zeichen Länge zwischen mobilen Endgeräten und Hostsystemen wie E-Mail-Gateways oder Alarmierungssystemen. Diese Systeme fasst man unter der Abkürzung SME (*Short Messaging Entity*) zusammen. Eine SME versendet die kodierte Nachricht an ein entsprechendes SMSC (*Short Message Service Center*), welches die endgültige Zustellung an das Zielsystem übernimmt. Es ist möglich, eine Empfangsbestätigung beim Zielsystem anzufordern, indem der Nachricht die Kennung „`*N#`“ vorangestellt wird.

Jeder Administrator kann über sein Mobiltelefon direkt mit dem System kommunizieren. Die Sicherheit von GSM-Funknetzen in der Datenübertragung ist allerdings umstritten, da häufig Verbindungen über analoge unverschlüsselte Richtfunkstrecken

[1] Während der Erstellung dieser Arbeit wurde ein vollwertiger Compiler für Perl-Skripte angekündigt.
[2] Application Programming Interface

geleitet werden. Außerdem schwankt die Verfügbarkeit mit dem GSM-Betreiber und dem Aufenthaltsort. Aufgrund der überwiegenden Vorteile wurden diese Risiken aber als vernachlässigbar eingestuft, zumal die ausgetauschten Informationen kaum ein Angriffspotential bieten. Sicherheitssensitive Änderungen wie das Abschalten von Überwachungsprozessen oder das Blockieren von Alarmierungen sind über SMS nicht möglich.

Die GSM-Netze bieten automatisch eine Authentifizierung des Absenders über seine weltweit eindeutige GSM-Rufnummer (vgl. Kapitel 5.5.3). Ein Administrator kann also mit seinem Mobiltelefon direkt mit den Tabellen der Datenbank kommunizieren und den aktuellen Status abfragen, ohne dass hierbei unberechtigte Zugriffe von anderen Mobiltelefonen befürchtet werden müssen. Allerdings wurde auch hier bereits bewiesen, dass SIM-Karten älterer Generationen kopiert werden konnten. Als Erweiterung der Implementation sind zusätzliche Authentisierungssysteme durch Transaktionsnummern (TANs) oder Passwörter denkbar.

Für die Implementation bieten sich mehrere Systeme zur SMS-Kommunikation an.

YAPS: Das Programmpaket `YAPS` (*Yet Another Pager Software*) bietet die Möglichkeit, von der UNIX-Kommandozeile aus Kurznachrichten an beliebige Empfänger zu senden. YAPS bedient sich der UCP- und TAP-Protokolle, mit denen man über eine Einwahl über Modem oder ISDN direkt beim GSM-Anbieter Nachrichten in das GSM-Netz senden kann. Erste Tests einer einfachen Alarmierung haben aber bereits gezeigt, dass diese Einwahl eine zu geringe Verfügbarkeit aufweist. Außerdem würde hier der Ausfall des Telefonsystems eine Alarmierung verhindern. Eine weitere Einschränkung ist die unidirektionale Kommunikation: YAPS kann zwar Nachrichten versenden, allerdings hat das System dann keine eindeutige Absenderkennung, um Nachrichten zu empfangen. Nicht zuletzt würde sich die Alarmmeldung um 30–60 Sekunden verzögern bedingt durch den langsamen Verbindungsaufbau.

SMSC: Eine zweite Möglichkeit wäre die Einrichtung eines eigenen SMSC (*Short Message Service Center*). Bei dieser Lösung wird zum GSM-Anbieter eine paketorientierte Datenverbindung über das X.25-Protokoll aufgebaut. Dies würde den direktesten Weg in das GSM-Netz anbieten — eine Lösung, die auch von kommerziellen Dienstanbietern genutzt wird. Das Kommunikationsmodul wäre allerdings deutlich komplizierter zu realisieren als bei den anderen aufgezeigten Lösungen. Außerdem wären die entstehenden Kosten für die eher geringe Anzahl an versendeten Nachrichten vergleichsweise hoch.

GSM-Modem: Die dritte Lösung wäre der Einsatz eines GSM-Modems. Ein solches Modem ist vom Funktionsprinzip her ein vollwertiges Mobiltelefon ohne Menüführung und Hörerfunktion, das primär zur Datenkommunikation in mobilen Systemen eingesetzt wird. Wie ein herkömmliches GSM-Mobiltelefon wird eine SIM-Karte eingelegt, über die die entstehenden Gebühren abgerechnet werden. Eine angeschlossene externe Antenne sorgt für die nötige Funkverbindung zum GSM-Netz. Diese GSM-Modems lassen sich mit Befehlen ähnlich dem AT-Hayes-Befehlssatz über eine serielle Konsole ansteuern. Die SIM-Karte identifiziert dieses Endgerät außerdem eindeutig und kann somit auch Kurznachrichten verarbeiten, die von Administratoren an diese Rufnummer gesendet werden. Die Implementation der SMS-Kommunikation basiert auf dem Einsatz eines GSM-Modems.

Eine Erweiterungsmöglichkeit ist der Einsatz des moderneren WAP-Protokolls, die prinzipiell jederzeit möglich wäre. Zusätzlich zur SMS-Kommunikation ist die Situationsanalyse auch in der Lage E-Mails zu versenden, falls ein Systemadministrator nicht über ein GSM-Mobiltelefon verfügt. Die Alarmierung über SMS ist aber wegen der geringeren Reaktionszeiten und direkteren Kommunikation vorzuziehen.

Wie auch die Situationsanalyse wird das SMS-Kommunikationsmodul in Perl programmiert. Unter Verwendung der POSIX-Schnittstelle zu allen Funktionen der IEEE 1003.1-Definition lässt sich das GSM-Modem über eine serielle RS232-Kommunikationsschnittstelle am Überwachungsrechner ansteuern. Der Aufbau des Kommunikationsmoduls und die verwendeten Kommunikationsbefehle zur Ansteuerung des GSM-Modems werden später in der Implementation behandelt.

4.3.6. Trouble-Ticketing

Das zentrale Objekt des gesamten Überwachungssystems ist das Trouble-Ticket. Entscheidet ein Alarmmodul, dass aufgrund einer gegebenen Situation ein Alarm ausgelöst werden soll, so wird im direkten Zusammenhang mit der Alarmierung ein neues Trouble-Ticket erzeugt, das symbolisch für die Verantwortlichkeit zur Lösung des aufgetretenen Problems steht.

Ein Trouble-Ticket enthält (entsprechend den Einträgen in der Tabelle *tickets*) diese Informationen:

- Eine Identifikationsnummer, die auch nach der Erledigung des Trouble-Tickets einmalig bleibt.
- Den Zeitpunkt, zu dem diese Störung erkannt wurde.
- Den momentanen Status des Tickets (alarmiert, offen, übernommen, geschlossen oder storniert).
- Den Inhalt des Tickets, der die Entscheidung des Alarmmoduls textuell beschreibt.
- Den verantwortliche Systemadministrator, der sich dieser Störung angenommen hat.
- Eine textuelle Beschreibung des verantwortlichen Systemadministrators über den momentanen Stand der Problemlösung.
- Die Historie der Änderungen an diesem Ticket, damit die Bearbeitungszustände protokolliert werden.

4.3.7. Bedienschnittstelle

Nach den Anforderungen an die Bedienschnittstelle soll eine bidirektionale Kommunikation zwischen den Systemadministratoren des IRT und der Datenbank ermöglicht werden. Um den Implementationsaufwand gering zu halten, bietet sich zum einen die Nutzung des SMS-Diestes an. Da bereits ein SMS-Alarmierungsmodul gefordert ist, kann man so die Alarmierung und die Kommunikation gemeinsam lösen.

Abbildung 4.2.: Abfrage des momentanen Status

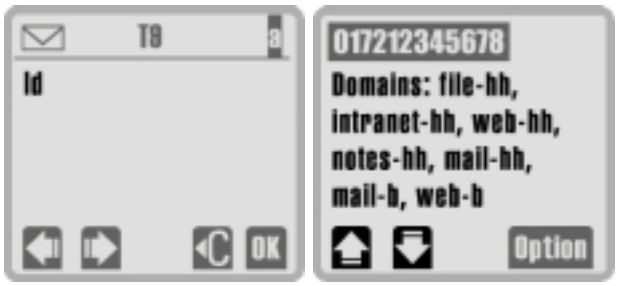

Abbildung 4.3.: Abfrage aller administrativen Domänen

Da dem Benutzer aber nur verhältnismäßig geringe Datenmengen über Kurznachrichten gesendet werden können (160 Zeichen je Nachricht), sollte noch eine weitere Möglichkeit der Bedienung vorgesehen werden. Da als Implementations-Komponenten bereits Perl und MySQL verwendet werden, bietet sich hier die Verwendung einer webbasierten Schnittstelle unter Verwendung des Apache-Webservers an.

Die beiden Kommunikationsschnittstellen haben beide eine umfangreiche Funktionalität und werden deshalb im folgenden getrennt behandelt.

SMS-Modul

Der Forderung nach einem möglichst kurzen Meldeweg kommt die Kommunikation mittels SMS nach. Der Administrator kann durch das Versenden von Befehlen in Kurznachrichten mit dem überwachenden System kommunizieren. Im folgenden sollen die Möglichkeiten anhand von Beispielsitzungen gezeigt werden. Tabelle 4.11 zeigt eine Übersicht über die zu implementierenden Befehle. Empfängt das SMS-Modul eine Kurznachricht, so überprüft es die Absenderkennung (GSM-Rufnummer) anhand der gespeicherten Daten in der Datenbank. So kann der Absenderkennung direkt der Name des Administrators zugeordnet werden. Dadurch wird auch ermöglicht, dass sich der Absender ein Trouble-Ticket selbst zuweisen kann, da das System weiß, von welcher Person diese Anfrage gestellt wurde.

Befehl vom Administrator	Bedeutung
?	Gesamtstatus der Netzwerküberwachung (vgl. Abb. 4.3.7)
ld	(List Domains) Anzeige aller administrativen Domänen mit Markierung derjenigen Domänen, in denen dieser Administrator eingetragen ist. (vgl. Abb. 4.3.7)
lw	(List Warnings) Anzeige aller momentanen Warnungen der Alarmmodule. (vgl. Abb. 4.3.7)
la	(List Alerts) Anzeige aller momentanen Alarme der Alarmmodule. (vgl. Abb. 4.3.7)
ut	(Unassigned Tickets) Anzeige der Trouble-Tickets, die noch von keinem anderen Administrator übernommen wurden. (vgl. Abb. 4.3.7)
tt #	(Take Ticket) Weist ein Trouble-Ticket dem Absender des Befehls zu. (vgl. Abb. 4.3.7)
et # abc	(Edit Ticket) Verändert den Informationstext eines Trouble-Tickets. (vgl. Abb. 4.3.7)
mt	(My Tickets) Anzeige der eigenen offenen Tickets (mit dem Status deferred oder taken). (vgl. Abb. 4.3.7)
ping	Schickt eine Antwortnachricht, in der der Absender namentlich identifiziert wird. Hauptsächlich zu Testzwecken und zur Überprüfung der Verfügbarkeit des SMS-Moduls. (vgl. Abb. 4.3.7)

Tabelle 4.11.: Befehle an das SMS-Modul

Abbildung 4.4.: Abfrage der aktuellen Warnungen

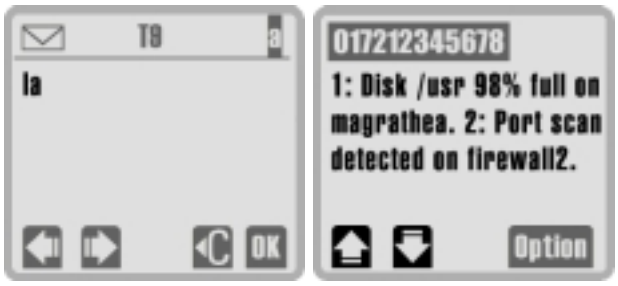

Abbildung 4.5.: Abfrage der aktuellen Alarme

Abbildung 4.6.: Abfrage der offenen Trouble-Tickets

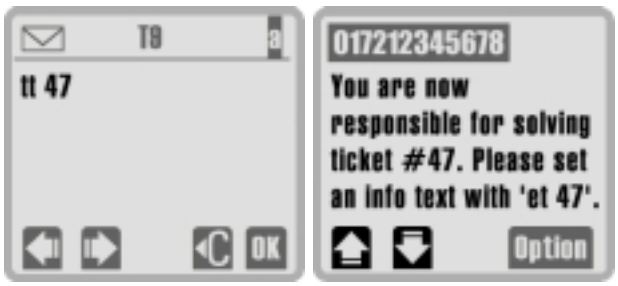

Abbildung 4.7.: Übernehmen eines offenen Trouble-Tickets

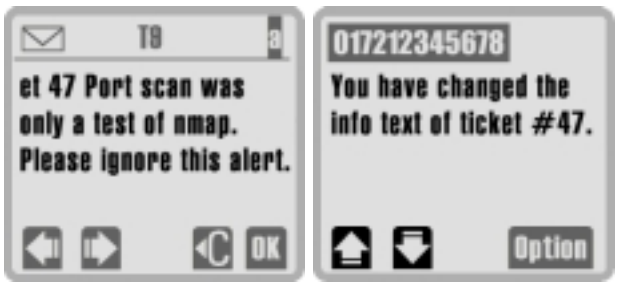

Abbildung 4.8.: Editieren eines Trouble-Tickets

Abbildung 4.9.: Abfrage der eigenen Trouble-Tickets

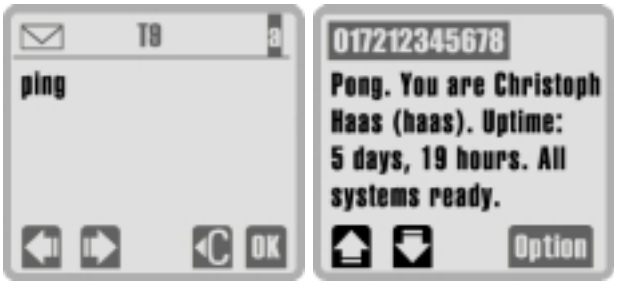

Abbildung 4.10.: Testanfrage

Web-Frontend

Wurde ein Systemadministrator mittels E-Mail oder SMS über ein aufgetretenes Problem alarmiert, so bietet ihm das SMS-Kommunikationsmodul nur eine rudimentäre Sicht auf die Gesamtsituation. Eine komfortablere Interaktion mit dem Überwachungssystem soll eine webbasierte Bedienschnittstelle bieten.

4.4. Interaktion der Management-Komponenten

In den vorigen Abschnitten wurden die Funktionen der einzelnen Komponenten des Überwachungssystems behandelt. Jetzt soll das Zusammenspiel dieser Komponenten im Gesamtsystem (wie in Abbildung 4.11 gezeigt) betrachtet werden.

Die Basis für alle Sicherheitsüberwachungen ist die *Sicherheits-Policy*, die vom globalen Netzwerkadministrator definiert wird. Dort sind der Einsatz von Monitoring-Komponenten, die Konfiguration der Rechensysteme und die erlaubten bzw. unerwünschten Handlungen im Netzwerk festgelegt. Es liegt nun in der Verantwortung der lokalen Systemadministratoren der einzelnen Niederlassungen, diese Richtlinien durchzusetzen und mit Hilfe des hier vorgestellten Sicherheitssystems zu überwachen. Die Sicherheitsüberwachung wird über drei Mechanismen konfiguriert:

Monitoring-Komponenten: Um überhaupt Informationen über Betriebszustände zu erhalten, müssen Monitoring-Instanzen (wie z.B. PIKT) auf den zu überwachen-

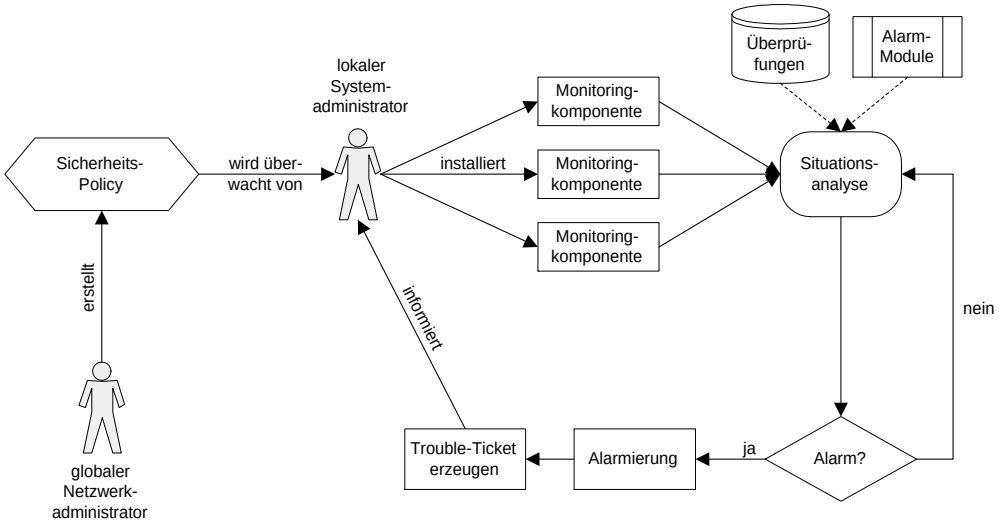

Abbildung 4.11.: Grundlegender Ablauf der Sicherheitsüberwachung

den Rechensystemen installiert werden, damit diese Informationen in die zentrale Datenbank (in die Tabelle *events*) übertragen werden können.

Überprüfungen: Die Tabelle *checks* enthält die Definitionen der Überprüfungen, die das Sicherheitssystem vornehmen soll. Hier werden das betreffende Zielsystem, die Art der Überprüfung (namentlich das Alarmmodul) und die Prüfparameter eingestellt. Diese Einstellungen sind die Basis für die durchgeführten Überprüfungen der Situationsanalyse. Eine Überprüfung könnte z.B. die folgende Formulierung in einer maschinenlesbareren Form beinhalten: *„Überprüfe, ob die Füllung der Partition* `'/var'` *des Systems* `'vigor3'` *einen geringeren Wert als 90% hat.“* Eine Sicherheitsüberwachung soll möglichst jeden Aspekt der Sicherheits-Policy in diesen Überprüfungen abbilden (vgl. Kapitel 4.2.3).

Alarmmodule: Das Alarmmodul ist der Algorithmus, der anhand der gegebenen Situation (die in der Tabelle *events* festgehalten wird) entscheiden kann, ob auf einem Rechensystem ein definierten Parameter einen kritischen Wert erreicht hat oder ob ein kritisches Ereignis eingetroffen ist. Im vorigen Beispiel entspricht ein Modul dem Teil *„Auslastung der Partition“* der Überprüfung. Dieses Alarmmodul könnte den Namen `DISK` haben und aus den Daten der Tabelle *events* entnehmen, ob die Auslastung einer durch Parameter angegebenen Partition noch innerhalb der Toleranz liegt. Das Alarmmodul entscheidet schließlich, ob das IRT alarmiert werden muss, um diese Störung zu beheben.

Asynchron zum Ablauf der Monitoring-Komponenten wird die Situationsanalyse periodisch alle gewünschten Überprüfungen ausführen. Dazu durchläuft es alle Datensätze der Tabelle *checks* und erhält daraus alle Parameter für eine Überprüfung. Die wichtigsten Daten sind `dst` (der Name des Rechensystems, auf dem ein Parameter geprüft wird), `amodule` (der Name des Alarmmoduls, das algorithmisch über die verfügbaren Informationen entscheidet) und `apar1`/`apar2`/`apar3` (zusätzliche Parameter für die

Überprüfung). Das Alarmmodul wird dann mit dem kompletten Datensatz der Überprüfung aufgerufen, um eine Entscheidung zu treffen.

4.4.1. Prüfparameter des Alarmmoduls 'DISK'

An dieser Stelle soll ein praktisches Beispiel die Funktionsweise eines Alarmmoduls verdeutlichen. Der Zweck der folgenden Überprüfung sei wie oben angegeben, ob die Auslastung der Partition `/var` auf dem System `vigor3` unter 90% liegt. Der Datensatz in der Tabelle *checks* könnte wie folgt aussehen:

Feldbezeichner	Inhalt
id	39
src	vigor3
dst	vigor3
amodule	DISK
apar1	/var
apar2	90
apar3	
aparams	Utilisation of partition /var
status	ok
info	89% full
admdomain	operator-hamburg
ticketid	
alertdelay	2
maintenance	n
alerttimes-monday	9,10,11,12,13,14,15,16,17
⋮	
alerttimes-friday	9,10,11,12,13,14,15,16,17
alerttimes-saturday	
alerttimes-sunday	

Die Überprüfung hat die laufende Nummer 39 und soll vom System `vigor3` ausgehend das System `vigor3` überprüfen. Die Angaben über das Prüfsystem (`src`) erlauben, denselben Parameter auf demselben System von verschiedenen Punkten aus zu testen. So lassen sich z.B. Verfügbarkeiten über verschiedenen Netzwerkstrecken überprüfen. Das gewünschte Alarmmodul ist `DISK` — an dieses Modul werden diese gesamten Parameter im nächsten Schritt übergeben. Der Alarmparameter `apar1` enthält die Angabe `/var`, also die zu überprüfende Partition. Der zweite Parameter (`apar2`) beinhaltet die Angabe 90, die hier für den Grenzwert der Auslastung in Prozent steht. Die Angabe `aparams` enthält eine textuelle Angabe dieser Überprüfung, die für die Ausgabe von Alarmtexten verwendet wird. Der `status` der letzten Überprüfung war `ok` — diese Information dient der Anzeige über die Bedienschnittstelle. Der Eintrag `info` gibt dazu eine textuelle Beschreibung des Ergebnisses der letzten Überprüfung an. Im Falle einer Alarmierung werden die Administratoren informiert, die zum IRT `operator-hamburg` gehören. Alle IRT-Zugehörigkeiten sind an anderer Stelle in der Tabelle *admdom* gespeichert. Das Feld `ticketid` ist leer, da zur Zeit kein Alarm anliegt und so auch kein zugehöriges Trouble-Ticket existiert. Würde das Alarmmodul entscheiden, dass der ak-

tuelle Zustand alarmierend ist, würde die Alarmierung noch 2 Minuten (`alertdelay`) verzögert werden. So können kurzzeitige Ausfälle ignoriert werden. Die Überprüfung ist momentan aktiv, da der Parameter `maintenance` den Wert `'n'` (no/nein) hat. Außerdem geben die Alarmierungszeiten (`alerttimes...`) an, dass nur werktags zwischen 9 Uhr und 18 Uhr alarmiert werden soll.

4.4.2. Aufruf des Alarmmoduls

Die Parameter werden an das Modul `DISK` übergeben. Der Entscheidungsalgorithmus dieses Moduls ist relativ einfach. Die Tabelle *events* beinhaltet alle gesammelten Informationen der Monitoring-Instanzen — so auch Informationen über die Auslastungen der verschiedenen Partitionen. Zunächst würde das Modul die Information aus der Tabelle *events* abrufen, die die Auslastung der Partition `/var` auf dem System `vigor3` angeben. Eine entsprechende SQL-Abfrage sähe so aus:

```
SELECT * FROM events WHERE dst=vigor3 AND aclass=DISK AND apar1='/var'
```

Wären Monitoring-Informationen vorhanden, so könnte der erhaltene Datensatz wie folgt aussehen:

Feldbezeichner	**Inhalt**
id	283
src	vigor3
dst	vigor3
time	20001204131453
comment	Partition /var is 91% full
aclass	DISK
apar1	/var
apar2	
apar3	
status	91
processed	n

Zunächst überprüft das Alarmmodul das Alter dieser Information. Dazu zieht es den Zeiteintrag `time` heran (13:14:53 am 4. Dezember 2000) und vergleicht ihn mit der aktuellen Zeit. So lässt sich feststellen, ob längere Zeit von den Monitoring-Instanzen keine Informationen geliefert wurden. In diesem Fall könnte ebenfalls ein Alarm ausgelöst. Das Modul kann dem Feld `status` entnehmen, dass die aktuelle Auslastung in Prozent '91' beträgt. Da in der Überprüfung ein Grenzwert von 90% definiert wurde, ist dieser Zustand alarmierend. Das Alarmmodul gibt eine entsprechende Datenstruktur zurück an die Situationsanalyse.

4.4.3. Alarmierung

Die Situationsanalyse erhält nun die Informationen über die Entscheidung des Alarmmoduls zurück. Wurde entschieden, dass die Situation einen Alarm erfordert, so legt die Situationsanalyse ein neues Trouble-Ticket an (sofern nicht bereits im letzten

Durchlauf eins erzeugt wurde) und trägt die laufende Nummer des Trouble-Tickets in der Tabelle *checks* ein. Damit die verantwortlichen Personen des IRTs informiert werden können, werden dessen Namen aus der Tabelle *admdom* ausgelesen:

Feldbezeichner	Inhalt
id	14
domain	operator-hamburg
admins	mueller,meier,schulze
info	Operator-Dienste des RZ-Hamburg

Laut Datensatz der *checks*-Tabelle ist die Gruppe 'operator-hamburg' für die Beseitigung dieses Problems verantwortlich. Mittels der Abfrage

```
SELECT admins FROM admdom WHERE domain='operator-hamburg'
```

erhält man die Namen der Mitglieder des IRTs — in diesem Beispiel die Administratoren 'mueller', 'meier' und 'schulze'. Jede dieser Personen wird einzeln alarmiert. Dazu gibt es wiederum in der Tabelle *admins* je einen Datensatz pro Person, der u.a. angibt, wie diese Person alarmiert werden soll. Als Beispiel soll hier der Datensatz für 'mueller' angegeben werden:

Feldbezeichner	Inhalt
id	6
name	mueller
realname	Klaus Müller
password	39aak309ajq
gsmno	+491791827462
gsmjammed	n
email	klaus.mueller@unternehmen.de
amethod	sms,email

Aus dieser Tabelle entnimmt die Alarmierungs-Subroutine die gewünschten Alarmierungs-Methoden (`amethod`). Ist die Alarmierung mittels SMS gewünscht, so wird eine Nachricht an die Mobilfunknummer gesendet, die im Feld `gsmno` angegeben ist. Alternativ ist auch eine Alarmierung durch eine Email möglich. Im Falle der SMS-Alarmierung wird die Nachricht in die Tabelle *smsqueue* geschrieben und weiter verarbeitet vom...

4.4.4. SMS-Modul

Dieser Prozess überprüft permanent eingehende und ausgehende Kurznachrichten. Bei einer eingehenden Nachricht wird diese geparset und eine entsprechende Behandlungsroutine aufgerufen, die mit der Datenbank kommuniziert und eine Antwort an den Absender richtet. Außerdem überprüft das Modul regelmäßig den Inhalt der Tabelle *smsqueue* und versendet alle hier enthaltenen Kurznachrichten. Der Versand wird lediglich dann nicht ausgeführt, wenn das Flag `gsmjammed` in der Tabelle *admins*

gesetzt ist. Dieses Flag wird vom SMS-Modul automatisch gesetzt, wenn mehr als 6 Kurznachrichten für einen Systemadministrator anstehen. So kann eine Überflutung des Empfängers mit Nachrichten verhindert werden, falls durch eine Fehlkonfiguration mehrere Alarme gleichzeitig ausgelöst werden. Da die SIM-Karte eines Mobiltelefons (des Systemadministrators) in der Regel nur 10 Kurznachrichten speichern kann, wäre ein Versand von mehr als 10 Nachrichten sinnlos. Das SMSC würde zwar weitere Nachrichten später zustellen, aber da diese Verzögerungen in der Praxis mehrere Stunden sind, ist die Alarmierung meist nicht mehr aktuell. Der Systemadministrator wird lediglich darüber informiert, dass der automatische Versand von Kurznachrichten blockiert wurde. Er kann aber trotzdem weiterhin das SMS-Modul abfragen, um sich einen Überblick über die Situation zu verschaffen. Den Versand kann er allerdings nur über das Web-Frontend wieder freigeben, wo er sich auch eine Liste der wartenden Kurznachrichten anzeigen lassen kann.

Der alarmierte Systemadministrator hat also die Alarmierung erhalten und kann jetzt das automatisch erzeugte Ticket mit seinem Mobiltelefon übernehmen. Angenommen das Alarmierungsmodul hätte eine Kurznachricht mit dem Inhalt

```
New ticket #325 created: No SMTP response from mailserver1
```

an alle verantwortlichen Systemadministratoren der administrativen Domäne verschickt, so könnte einer der Empfänger die Nachricht

```
tt 325
```

zurücksenden. `tt` steht für *„take ticket“* und bedeutet die Übernahme des Tickets. Das SMS-Modul würde ihm diese Transaktion mit

```
You are now responsible for solving ticket #47. Please set an
info text with 'et 325 text'.
```

bestätigen. Alle anderen Administratoren der Domäne erhalten die Meldung

```
Ticket #325 was just taken by Christoph Haas.
```

vom SMS-Modul. Zu diesem Zweck wird in der Tabelle *tickets* das Feld `lastnotified` verwaltet, in dem alle zuletzt informierten Personen festgehalten werden. Sobald die Ursache des Problems behoben ist, wird die Situationsanalyse die veränderte Situation wahrnehmen und das Trouble-Ticket stornieren. Dann erhält die zuständige Person die Nachricht

```
Ticket #325 was just revoked because: mailserver1 replied to
our SMTP request.
```

Alle anderen Systemadministratoren und auch der globale Netzwerkadministrator können den Zustand des Trouble-Tickets jederzeit über das Web-Frontend abfragen.

5. Implementation: Mr.Network

Inhaltsangabe

Die Implementation kann nahezu beliebige Sicherheitsüberwachungen durchführen, da sie auf den Monitoring-Informationen externer Programme basiert und intern die Möglichkeit der Einbindung von Alorithmen zur Auswertung der Situation bietet. Die Monitoring-Instanzen sind allerdings nicht Teil dieser Arbeit. Bei der Entwicklung der Implementation wird an dieser Stelle exemplarisch auf die Verfügbarkeitsüberwachung eingegangen. Beliebige andere Sicherheitsüberprüfungen (wie sie im Kapitel 3 beschrieben wurden) können aber ebenfalls mit dieser Implementation realisiert werden.

Im vorigen Kapitel wurden die Anforderungen an die Implementation genannt. An dieser Stelle werden nun konkrete Software-Komponenten ausgewählt, die diese Anforderungen erfüllen.

Das gesamte System wurde `Mr.Network`[1] getauft und besteht aus den Teilkomponenten `Mr.Analyse` (Auswertungslogik), `Mr.Web` (Web-Frontend), `Mr.Log2SQL` (Importiert Monitoring-Informationen) und `Mr.SMS` (SMS-Kommunikationsmodul). Die genaue Funktion dieser Komponenten wird in den folgenden Kapiteln detailliert behandelt.

[1] In Anlehnung an den Kaffeeautomaten *Mr.Coffee* und den Radarempfänger *Mr.Radar* aus der Science-Fiction-Parodie *Spaceballs* von Mel Brooks (1987).

5.1. Monitoring und Reporting

Bei der Auswahl der Monitoring-Komponente wurde eine hohe Flexibilität gefordert, um aktuellen und zukünftigen Anforderungen an eine Sicherheitsüberwachung gerecht zu werden. Die eigentliche Aufgabe des Monitoring ist es, möglichst umfangreiche Informationen über den Gesamtzustand des Netzwerks periodisch an die zentrale Datenbank zu übermitteln.

Zum Zeitpunkt der Testphase der Implementation basierte das Monitoring hauptsächlich auf dem bereits auf Seite 49 beschriebenen Programmpaket `PIKT`. Um den Rahmen dieser Arbeit nicht zu sprengen, werden die einzelnen Monitoring-Subroutinen in der PIKT-Sprache an dieser Stelle nicht erklärt. Die PIKT-Module umfassen u.a. folgende Informationsarten:

- Verfügbarkeit von TCP-Diensten
- Festplattenauslastungen
- CPU-Auslastungen

Die erhaltenen Informationen werden von den PIKT-Clients über den Syslog-Protokollmechanismus an das Überwachungssystem weitergegeben und dort in die Datenbank eingetragen. Da die Niederlassungen wie gefordert über ein VPN verbunden sind und der Datenverkehr zwischen den Niederlassungen damit verschlüsselt wird, werden auch diese Informationen gesichert übertragen.

Neben PIKT wurde auch AIDE als externes IDS in den Überwachungsprozess mit eingebunden, das Systemeinbrüche oder Einbruchsversuche erkennen soll.

5.2. Datenbank: MySQL

Nach der Evaluation mehrerer relationaler Datenbanksysteme fiel die Auswahl auf MySQL der schwedischen Firma TcX DataKonsult AB (siehe [mysql]). MySQL folgt dem ANSI SQL92-Befehlssatz und erlaubt somit standardisierte Anfragen an die Datenbank. Im Benchmark-Vergleich mit anderen Datenbanken wie DB2, Informix und Sybase wurden Lese- und Einfügeoperationen bis zu 500-mal schneller ausgeführt. MySQL wurde im Juni 2000 mit der Version 3.2319 unter der GNU General Public License freigegeben.

MySQL bietet einen großen Vorteil, denn es existiert mit der `DBI`-Klasse eine vollständige Schnittstelle (API) zur Anbindung von Perl-Modulen an das DBMS. Zur Kapselung der Verwaltungsfunktionen des Überwachungssystems vom Datenbankdesign wird die Implementation eine Perl-Klasse beinhalten, die gängige Datenabfragen vereinfachen, aber auch das Risiko falscher Zugriffe durch fehlerhafte direkte Datenbankabfragen minimiert (*information hiding*). Diese Verwaltungsklasse bietet u.a. diese Funktionen:

- Hinzufügen von Einträgen zur Protokolldatei
- Ausführen von SQL-Abfragen
- Erzeugen, Abfragen, Zuweisen und Stornieren von Trouble-Tickets

- Alarmieren zuständiger Administratoren des IRT
- Versenden von Kurznachrichten (SMS) über die Datenbank-Queue
- Setzen von Status-Einträgen nach Entscheidungen der Situationsanalyse
- Verschiedene Konvertier- und Ausgabefunktionen z.B. für Zeit- und Datumsinformationen

5.3. Situationsanalyse: Mr.Analyse

5.3.1. Ablaufplan der Analyse

Der genaue Ablauf der Situationsanalyse soll anhand des Ablaufplans in Abbildung 5.1 erläutert werden. Zunächst wird die gesamte Datenbank einmalig einer Konsistenzprüfung unterzogen. Dadurch soll vermieden werden, dass die Situationsanalyse aufgrund von Inkonsistenzen der Datenbankinhalte auf Fehlersituationen trifft, die sie nicht selbstständig beheben kann. Da diese Prüfung verhältnismäßig viel Rechenzeit benötigt, wird sie nur einmal ausgeführt. Nach umfangreichen Änderungen in den Datenbanktabellen, die zur Steuerung des Analyseprozesses verwendet werden, ist es zweckmäßig, die Situationsanalyse neu zu starten, um erneut die Konsistenz der Datenbank überprüfen zu lassen. Während der Laufzeit des Analyseprozesses wird an vielen Stellen die Konsistenz der Datenbank geprüft, um den Prozess kontrolliert abbrechen zu können, falls eine Inkonsistenz auftritt, die nicht automatisch behoben werden kann.

Die Analyse durchläuft dann periodisch eine Folge von Überprüfungen. Für jede Überprüfung wird das entsprechende Alarmmodul aufgerufen, um zu entscheiden, ob die Gesamtsituation im Netzwerk problematisch ist. Im Rahmen jeder Überprüfung laufen folgende Schritte ab:

1. Das aufgerufene Alarmmodul entscheidet algorithmisch mittels der Informationen der Tabelle *events*, ob die aktuelle Situation eine Alarmierung erfordert.
2. Der Analyseprozess aktualisiert die erhaltenen Informationen in der Tabelle *checks* z.B. für eine spätere Anzeige über die Bedienschnittstelle.
3. Ist die Überprüfung im Zustand `maintenance` (Wartung), so bricht die Analyse an dieser Stelle ab und springt zur nächsten Überprüfung.
4. Hat das Alarmmodul entschieden, dass das IRT alarmiert werden soll, so wird ein neues Ticket angelegt (falls nicht schon ein offenes Ticket zu diesem Fall existiert). Es wird dann die angegebene Alarmverzögerung (`alertdelay`) abgewartet und überprüft, ob die Alarmierung zum momentanen Zeitpunkt überhaupt erwünscht ist (`alerttimes`). Erst dann werden die zuständigen Systemadministratoren des IRT über den gewünschten Alarmierungsweg informiert. Der Parameter `alertdelay` wurde eingeführt, um kurze vorübergehende Probleme ignorieren zu können. Ansonsten wäre es möglich, dass ein ausgefallener Dienst bereits wieder verfügbar ist, bevor ein Mitglied des IRT zeitlich in der Lage war, sich dieses Problems anzunehmen. Um Probleme mit einer geringeren Gefahrenklassifizierung nur zu bestimmten Tageszeiten bearbeiten zu müssen (z.B. während

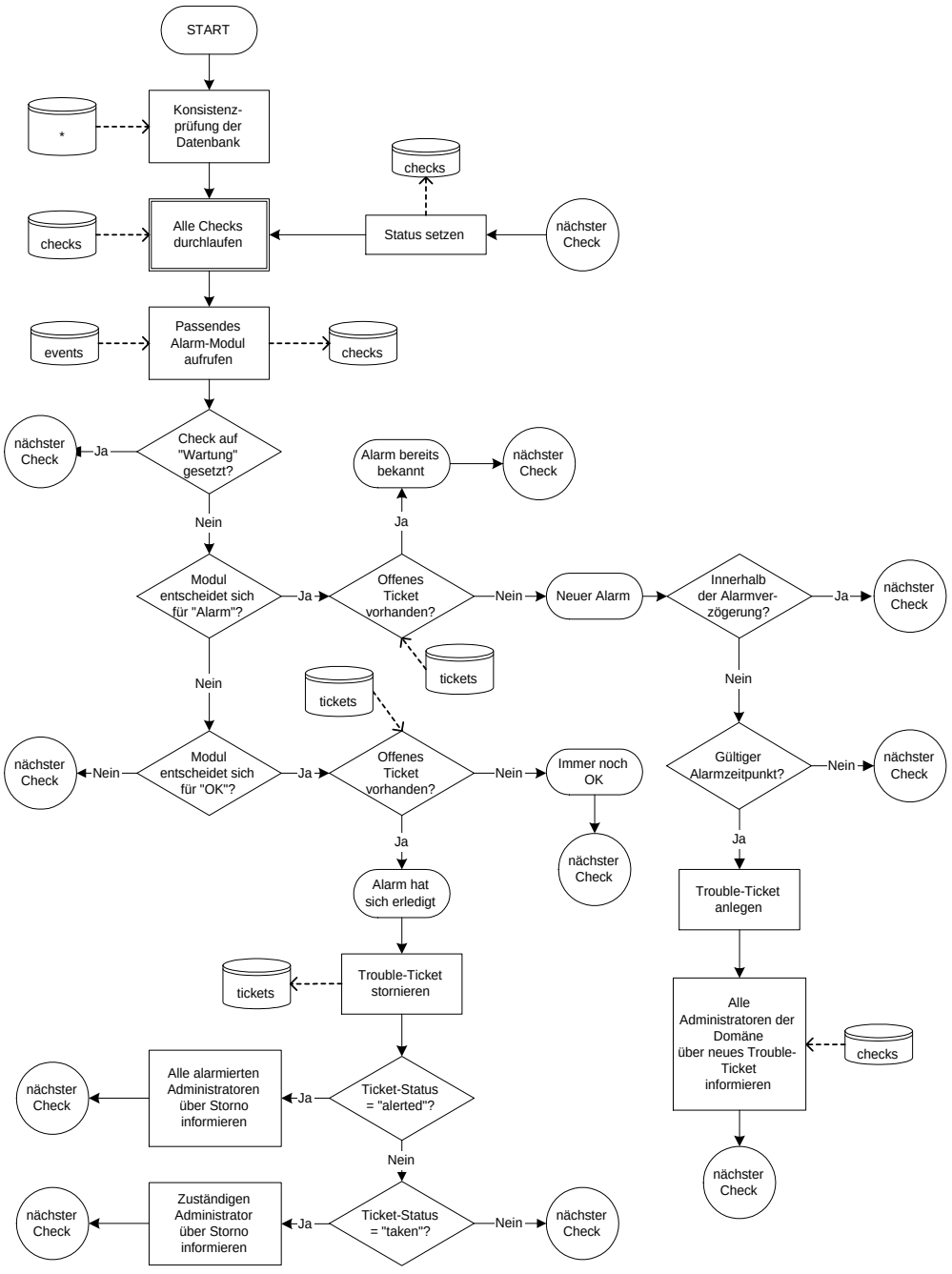

Abbildung 5.1.: Ablaufplan der Situationsanalyse

der Bürozeiten), kann die Alarmierung mit dem Parameter `alerttimes` zeitlich genau konfiguriert werden.

5. Lautete die Entscheidung, dass keine Alarmierung nötig ist (z.B. weil die Situation innerhalb normaler Parameter ist), so wird lediglich ein möglicherweise noch existierendes Trouble-Ticket storniert und ansonsten zur nächsten Überprüfung gesprungen. Bei einem Storno eines Trouble-Tickets wird das IRT ebenfalls informiert.

5.3.2. Funktionsweise der Alarmmodule

Jedes Alarmmodul führt entsprechend der übergebenen Parameter aus der Tabelle *checks* Datenbankabfragen ab, um aus den gewonnenen Informationen algorithmisch zu einer Entscheidung über mögliche Problemzustände zu kommen. Die Entscheidung des Alarmmoduls wird an den Analyseprozess zurückgegeben. Die Funktionalität der Alarmmodule wurde bewusst auf die algorithmische Verarbeitung der Situation beschränkt. Alle wiederkehrenden Tätigkeiten wie die Alarmierung und die Verwaltung der Trouble-Tickets werden vom Analyseprozess übernommen und sind somit nicht Bestandteil der einzelnen Alarmmodule.

5.3.3. Prototyp eines Alarmmoduls

Da die Datenbankfunktionen durch die eigens entwickelte `MrDB`-Objektklasse gekapselt sind, gestaltet sich der Prototyp eines Alarmmoduls relativ einfach. Ein Systemadministrator kann aus diesem Prototyp mit Hilfe eigener Datenbankabfragen eigene Entscheidungsalgorithmen entwickeln.

```
sub GENERIC_AMODULE
{
   my $self = shift;    # $self ist die Objektreferenz

   my $MrDB = new MrDB; # dieses Modul instanziert die Hilfsklasse
                        # "MrDB", um dessen Hilfsfunktionen nutzen
                        # zu können

   # $pars ist eine Referenz auf ein assoziatives Feld (hash array),
   # in dem die folgenden Felder der Datenbank übergeben werden:
   # - id
   # - amodule
   # - src
   # - dst
   # - apar1
   # - apar2
   # - apar3
   # - status
   my $pars = shift;

   # ENTSCHEIDUNGSPROZESS
```

```
# Datenbankabfrage aus der EVENTS-Tabelle
my $query = "select * from events where...";
my $qhandle = $MrDB->sql_query
(
   $query,
   $pars->{src},
   $pars->{dst},
   $pars->{amodule},
   $pars->{apar1},
   $pars->{apar2},
   $pars->{apar3},
) or &fatal_error(...);

# Parameter-String zur Anzeige der überprüften Eigenschaften
$parameters = "Parameter: ".$pars->{apar1};

# Ergebnis der Datenbankabfrage holen und auswerten
$event = $MrDB->sql_result($qhandle)
   or return
   # (noch) keine Ereignisse in EVENTS gespeichert
   {
      info => "Keine Information verfügbar.",
      decision => "warn",
      parameters => $parameter;
   };

# SQL-Abfrage beenden
$MrDB->sql_querydone($qhandle);

# Ist die Information älter als 30 Minuten?
my $eventage = $MrDB->timestamp_age($event->{time});
if ($eventage > (30*60) )
{
   $decision = "warn";
   $info = "Information ist veraltet."
}

# Entscheidung über Situation
elsif ($event->{status} <= 12345)
{
   $decision = "ok";
   $info = "Situation normal."
}
else
{
   $decision = "alert";
   $info = "Situation alarmierend."
}
```

```
   # Rückgabe aller Informationen an die Situationsanalyse
   return
   {
      info => $info,
      decision => $decision,
      parameters => $parameters,
      time => $event->{time}
   }
}
```

5.3.4. Implementierte Methoden

Zur Kapselung der Verwaltungsebene des Überwachungssystems wurden verschiedene Methoden (im objektorientierten Sinne) implementiert, die auch im obigen Prototypen des Alarmmoduls verwendet werden:

newest_event(src,dst,aclass,apar1,apar2,apar3) Diese Methode gibt entsprechend der Suchkriterien das zeitlich aktuellste Event-Objekt zurück. So kann für ein bestimmtes Kriterium das neueste Ereignis aus der EVENTS-Tabelle abgefragt werden. Rückgabewert ist das Event-Objekt jüngsten Datums.

sql_query(sql-query-string) Um auch direkt auf die Datenbank zugreifen zu können, bietet diese Methode SQL-Abfragen an. Rückgabewert ist ein Datenbank-Handle für diese Abfrage. Mit dem Datenbank-Handle kann das Ergebnis der Abfrage ausgelesen werden.

sql_result(dbhandle) Die Ergebnisse der mit sql-query gestartete Datenbankabfrage können mit dieser Methode ausgelesen werden. Wurden mehrere passende Datensätze gefunden, so gibt diese Methode solange assoziative Arrays (hash arrays) zurück, bis kein weiteres Ergebnis mehr gefunden wird (dann UNDEF).

create_ticket(admin,status,info,cause,lastnotified) Zum Anlegen eines Tickets wird diese Methode verwendet. Sie übergibt an die Datenbank alle notwendige Informationen, um ein vollständiges Ticket anzulegen und das zuständige IRT zu alarmieren.

cancel_ticket(id,info) Ist ein Ticket abgebrochen worden, so kann mit dieser Methode der Status des Tickets auf CANCELLED gesetzt werden. Die Methode unternimmt alle weitere Schritte (Mitteilung an das zuständig ERT über die Alarmierung).

assign_ticket(id,info,admin) Sobald ein Administrator des IRT sich für die Beseitigung der Störung bereit gefunden hat, weist er sich dieses Trouble-Ticket selbst zu. Diese Methode informiert alle anderen alarmierten Administratoren und verändert das Ticket entsprechend.

defer_ticket(id,info) Kann eine Störung nicht sofort behoben werden, wird die Lösung verzögert (*deferred*). Diese Methode zeigt die Verzögerung im Ticket an.

get_domain_admins(admdom) Mit Hilfe dieser Methode erhält man ein Array auf die Namen aller Systemadministratoren einer angegebenen administrativen Domäne. Diese Methode wird häufig bei der Alarmierung eingesetzt.

spool_sms(to,msg) Diese Methode legt eine neue SMS-Nachricht zur Alarmierung in der SMS-Versandwarteschleife ab.

5.4. Web-Interface: Mr.Web

Das Web-Frontend besteht aus mehreren Teilen. Die Webseite ist in zwei Frames aufgeteilt. Der linke Frame besteht aus einer HTML-Seite mit mehreren PHP3-Teilen. PHP3[2] ist eine in HTML eingebettete Skriptsprache, die bevorzugt zur Abfrage von Msql- und MySQL-Datenbanken verwendet wird und dessen Syntax an gängige Programmiersprachen wie C und Java angelehnt ist. Die Skripte werden serverseitig ausgeführt und deren Ausgabe in die HTML-Seite eingebettet. PHP3 wird hierbei verwendet, um die Anzahl der Einträge in Klammern zu zählen, die in der Ansicht angezeigt würden, falls dieser Menüpunkt ausgewählt wird. Im gezeigten Beispiel in Abbildung 5.4 stehen drei Alarme an.

Im rechten Frame wird ein CGI-Skript (*Common Gateway Interface*) ausgeführt, das wie alle anderen Module in Perl programmiert ist. *CGI* ist ein Kommunikationsverfahren, bei dem ein serverseitiges Skript die Benutzereingaben verarbeitet und eine HTML-Ausgabeseite erzeugt. In Abbildung 5.2 wird die Titelseite des Web-Frontends gezeigt.

Im Gegensatz zur zwangsläufigen Authentisierung des Absenders bei der SMS-Kommunikation ist ein webbasierter Zugriff zunächst anonym. Der Apache-Webserver (der bei dieser Implementation eingesetzt wird) stellt aber Schnittstellen zur Verfügung, um eigene Mechanismen zur Authentisierung und Autorisierung einzusetzen. Auch hier wird wieder die Verwendung von Perl-Skripten ermöglicht[3]. Wie bereits beschrieben, werden die Passwörter der zugangsberechtigten Administratoren in der Tabelle *admins* im UNIX-Crypt-Verfahren verschlüsselt gespeichert. Perl bietet ebenso die Crypt-Funktion an. So lässt sich relativ leicht ein Authentisierungsmodul in Perl als Zugangsbeschränkung zum Web-Frontend realisieren. Der Quellcode dieses Moduls befindet sich im Anhang A.2. Dieses Verfahren bietet einen weiteren Vorteil: Ändert ein Benutzer sein Passwort über das Web-Frontend, so wird das neue Passwort unmittelbar aktiv und der Benutzer muss sich sofort mit seinem neuen Kennwort anmelden.

Der Quelltext des Web-Frontends wird in dieser Arbeit nicht beschrieben, da es sich um eine typische CGI-Formularverwaltung in Perl handelt. An dieser Stelle sollen aber zumindest die verwendeten SQL-Abfragen aufgezeigt werden, mit denen die Datenbank abgefragt wird:

Anzahl der ordnungsgemäßen Prüfungen

```
select count(*) from checks where status='ok';
```

Auswahl der ordnungsgemäßen Prüfungen

```
select * from checks where status='ok';
```

[2] siehe [php]
[3] vgl. [modperl]

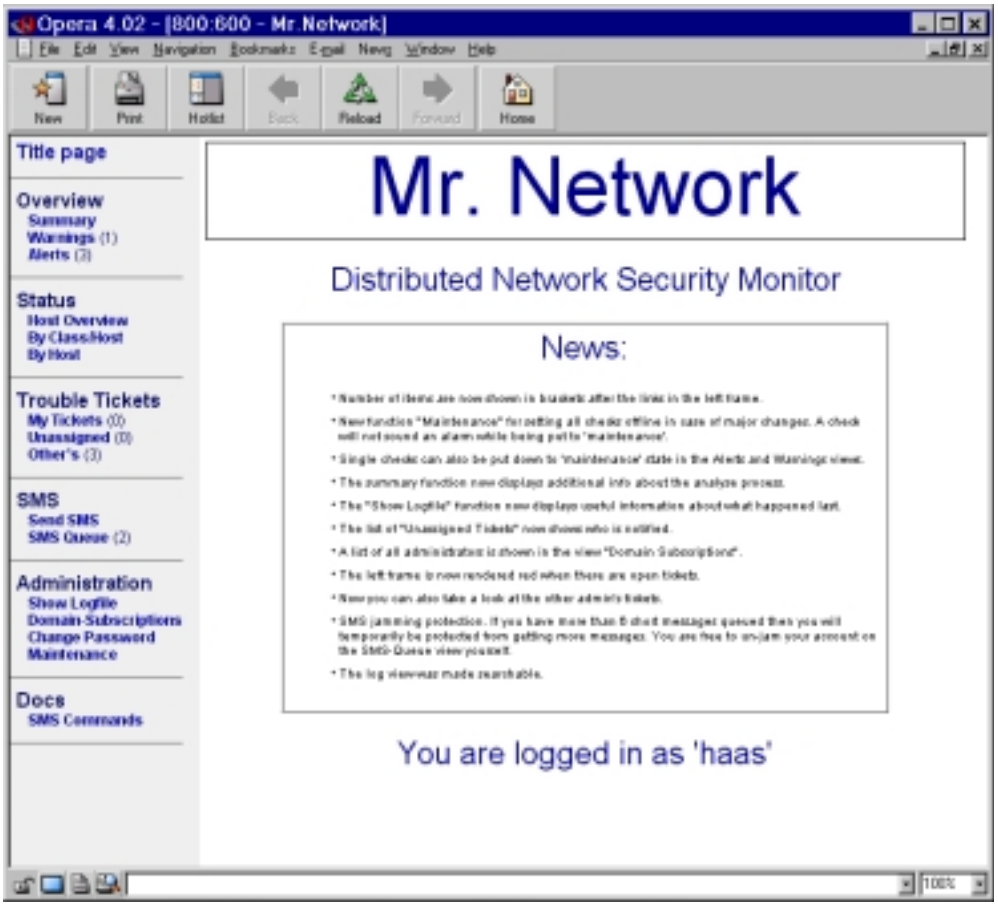

Abbildung 5.2.: Web-Frontend: Titelseite

Anzahl der Prüfungen mit Warnungen

```
select count(*) from checks where status='warn';
```

Auswahl der Prüfungen mit Warnungen

```
select * from checks where status='warn';
```

Anzahl der Prüfungen mit Alarmen

```
select count(*) from checks where status='alert';
```

Auswahl der Prüfungen mit Alarmen

```
select * from checks where status='alert';
```

Anzahl der Prüfungen in Wartung

```
select count(*) from checks where maintenance='y';
```

Anzahl der offenen Trouble-Tickets

```
select count(*) from tickets where status='alerted';
```

Anzahl der Trouble-Tickets in Bearbeitung

```
select count(*) from tickets where status in ('taken','deferred');
```

Anzahl der Nachrichten in der SMS-Versandwarteschleife

```
select count(*) from smsqueue;
```

Anzeige der ersten Überprüfungen[4] sortiert nach Status

```
select * from checks order by status,dst limit 0,20;
```

Auswahl aller Systemadministratoren zum Versand von Kurznachrichten

```
select name,realname from admins where gsmno!='';
```

Abfrage der GSM-Mobiltelefonnummer eines Administrators

```
select gsmno from admins where name='xyz';
```

Abfrage der eigenen Trouble-Tickets

```
select * from tickets where admin='$loginuser'
and status in ('taken','deferred');
```

Abfrage der geschlossenen Trouble-Tickets

```
select * from tickets where status in ('completed','cancelled');
```

Abfrage der offenen Trouble-Tickets anderer Systemadministratoren

```
select * from tickets where admin!='$loginuser' and admin!='' and
status in ('taken','deferred') group by admin;
```

Abfrage aller administrativen Domänen

```
select * from admdom;
```

Kategorisierte Abfrage der verwendeten Alarmmodule je Zielsystem

```
select * from checks order by amodule,dst;
```

Abfrage der letzten 30 Protokolleinträge

```
select * from logs order by id desc limit 0,30;
```

5.5. SMS-Gateway: Mr.SMS

5.5.1. Eingesetzte Hardware

Die Implementation nutzt zur SMS-Kommunikation ein GSM-Modem vom Typ „Falcom A2D-1-900/1800“ ab. Dieses Modem wird über eine RS232/V.24-Schnittstelle angesteuert und kann Kurznachrichten über den SMS mittels ETSI GSM 07.07-Befehlen versenden und empfangen. Die Ansteuerung des Modems wird von einer Perl-Instanz namens `Mr.SMS` übernommen.

5.5.2. ETSI GSM-Befehlssatz

Der vom GSM-Modem verwendete Befehlssatz ähnelt stark dem Hayes AT-Befehlssatz, wie er seit langer Zeit zur Steuerung von Modems im Telefonnetz verwendet wird. Zur Berücksichtigung der speziellen Anforderungen im Funkbereich hat das ETSI (*European Telecommunications Standards Institute*) ([etsi]) einen erweiterten GSM-Befehlssatz 07.07 unter der Bezeichnung *„AT command set for GSM Mobile Equipment“* veröffentlicht.

Anhand der folgenden Beispielsitzungen sollen die verwendeten Befehle erklärt werden. Alle implementierten Befehle sind in Tabelle 5.5.2 zusammengefasst.

Befehl	Antwort vom Modem	Bedeutung
AT&F		Zurücksetzen des Modems auf Werkseinstellungen
	OK	Initialisierung beendet
AT+CREG?		Anmeldung am GSM-Netz überprüfen
	+CREG: 0,1	Modem am GSM-Netz korrekt angemeldet
	OK	Modem bereit
AT+CSCA=+491722270000		SMSC einstellen
	OK	Modem bereit
AT+CMGF=1		Nachrichtenformat auf „Text“ setzen
	OK	Modem bereit
AT+CNMI=0,0		Automatische Benachrichtigung bei eingehenden Kurznachrichten abschalten. Ansonsten würde das Modem jede eingegangene Nachricht mit einer +CMGF:-Meldung quittieren und damit die Kommunikation stören.
	OK	Modem bereit
AT+CMGL=ALL		Empfangene Nachrichten aus der SIM-Karte abrufen
	+CMGL: 1,0,+491721234567,0,0	Nachricht mit der Indexnummer 1 vom Absender „+491721234567“ erhalten.
	Dies ist ein Test.	Inhalt der Nachricht
	OK	Keine weiteren Nachrichten gespeichert. Modem bereit.
AT+CMGD=1		Nachricht Nr. 1 aus der SIM-Karte löschen
	OK	Nachricht gelöscht. Modem bereit.
AT+CMGS=+491721234567		Neue Nachricht an „+491721234567“ versenden
	>	Bereit für Eingabe
Antwort.[CTRL-Z]		Eingabe der Nachricht
	+CMGS: 92	Nachricht versendet. Referenznummer der Nachricht im GSM-Netz ist 92.
	OK	Modem bereit.

Tabelle 5.2.: Beispielsitzung des GSM-Modems bei der SMS-Kommunikation

5.5.3. Sicherheitsbetrachtungen

Fälschung der Absenderkennung

Die Absenderkennung lässt sich verhältnismäßig schwer beim Versand von Kurznachrichten von einem Mobiltelefon aus fälschen, da die Rufnummer über die SIM-Karte und Informationen beim GSM-Dienstanbieter festgelegt wird.

Allerdings gibt es weitere Zugänge über eine Einwahl mit einem Modem beim GSM-Dienstanbieter. In Deutschland werden diese unter anderem von D2-Mannesmann (UCP-Protokoll) und D1 der Deutschen Telekom (TAP-Protokoll) angeboten. Bei diesen Protokollen wird die Absenderkennung vom Anwender vorgegeben. Dort lassen sich also beliebige Kennungen eingeben. Ist einem Angreifer eine Mobiltelefonnummer eines Systemadministrators bekannt, könnte er beispielsweise die Alarmierung über seinen parallel durchgeführten Angriff über SMS stornieren. Dies setzt aber voraus, dass er die Rufnummer erhalten hat, die Befehle des SMS-Moduls kennt und niemand anders rechtzeitig auf eine Alarmierung reagiert. Bei allen Prüfungen, die in der Tabelle *checks* angegeben sind, werden mehrere Administratoren alarmiert. Der betreffende Administrator würde lediglich eine Rückmeldung an sein Mobiltelefon gesendet bekommen zu einer Anfrage, die er gar nicht gestellt hat. Ansonsten bliebe dieser Angriff unbemerkt.

Da dieser Angriff eine genaue Kenntnis des Überwachungssystems erfordert und der Angreifer auch keine Rückmeldung über seine Aktionen erhalten würde, wurde das Gefahrenpotential während der Testphase des Prototypen als gering eingestuft. Eine zusätzliche Authentisierung des Absenders über ein Kennwort wäre jederzeit leicht zu implementieren. Sicherheitshalber sollte man hier Einwegpasswörter (analog zum TAN-Verfahren, das von Banken im Online-Banking verwendet wird) verwenden, so dass jedes verwendete Passwort nach seiner Verwendung ungültig wird.

Software-Verfügbarkeit

Das SMS-Modul ist ein gekapseltes Perl-Skript. Wie auch die anderen Module **MrAnalyse** und **MrLog2SQL** wird die Hauptroutine von einem `eval()`-Aufruf gekapselt. Tritt zur Laufzeit ein Fehler auf, der nicht automatisch behoben werden kann, so wird der `eval()`-Aufruf beendet und die Variable `$@` enthält bei einem Fehler die entsprechende Fehlermeldung. Das Programm versendet in diesem Fall eine Fehlermeldung mittels E-Mail an den Systemadministrator und zeichnet die komplette Fehlermeldung in der Protokoll-Datei auf.

Außerdem wird das Unterbrechungssignal `INT` durch einen Signal-Handler abgefangen und der Administrator über den Abbruch per E-Mail informiert, bevor sich das Programm beendet. Zusätzlich überwacht ein PIKT-Modul, dass der Prozess **Mr.SMS** läuft und alarmiert ansonsten ebenfalls den Administrator. Nur ein spontaner Totalausfall des gesamten Systems (z.B. CPU-Defekt) würde unentdeckt bleiben. Das Überwachungssystem einschließlich der GSM-Hardware ist zusätzlich durch eine leistungsfähige unterbrechungsfreie Stromversorgung abgesichert, deren Funktion auch vom System laufend überwacht wird.

Hardware-Verfügbarkeit

Leider erwies sich das GSM-Modem in der ersten mehrwöchigen Testphase als relativ unzuverlässig. Häufig reagierte das Modem auf Versandanfragen von Kurzmitteilungen unvermittelt mit Fehlermeldungen. Dieser Fehler konnte mit einem Firmware-Update des Herstellers beseitigt werden.

Das Kommunikationsmodul `Mr.SMS` enthält mehrere Recovery- und Failover-Absicherungen gegen sporadische Fehler im GSM-Netz. Bei ERROR-Antworten des Modems wird das Modul jedoch aus Sicherheitsgründen immer abgebrochen. Würde das Modul bei einer ERROR-Meldung einen erneuten Versand versuchen, hätte dies eventuell fatale Folgen, da theoretisch mehrere hundert Kurznachrichten vom SMSC gespeichert werden, die dann vom Administrator mühsam einzeln auf dem Mobiltelefon gelöscht werden müssten, falls die Kurznachricht das Zielsystem doch erreicht. Bei einem Fehler bei der Nutzung des SMSC des Netzbetreibers D2 wurde durch den Fehlercode suggeriert, die Nachricht wäre nicht zugestellt worden. Die Nachricht erreichte den Empfänger allerdings trotzdem.

Es sollte erwähnt werden, dass die eingegangenen Kurznachrichten auf der SIM-Karte im Modem gespeichert werden. Die Anzahl der Schreibzyklen während der Lebensdauer einer SIM-Karte werden von den Netzbetreibern auf 10.000 bis 100.000 geschätzt. Am Ende der Implementationsphase stellte der Hersteller FALCOM eine Methode vor, bei der Kurznachrichten ohne Umweg über die SIM-Karte empfangen und versendet werden können. Dieses komplexere Verfahren wurde in dieser Arbeit nicht implementiert. Kartenfehler traten während der Testphase nicht auf.

GSM-Netz-Verfügbarkeit

Wie jedes andere Übertragungsmedium ist auch das GSM-Netz nicht vor Störungen absolut gesichert. In Überlastsituationen des SMS-Systems[5] kam es bereits zu Ausfällen oder starken Verzögerungen im Nachrichtenverkehr.

Das GSM-Netz arbeitet im Bereich von 900 MHz bis 1900 MHz. Mittels eines einfachen Störsenders mit einer Sendeleistung im Bereich von einigen Watt ist es bereits möglich, die Kommunikation zwischen Mobiltelefon bzw. GSM-Modem und dem GSM-Relais zu blockieren.

5.6. Interaktion der implementierten Komponenten

Nachdem alle implementierten Komponenten von Mr.Network einzeln beschrieben wurden, soll abschließend noch einmal am Beispiel von Abbildung 5.3 das Zusammenspiel beschrieben werden. Die Basis für die Situationsanalyse bildet die Sicherheits-Policy, die vom globalen Netzwerkadministrator formlos erstellt wird. Es ist die Aufgabe der lokalen Systemadministratoren, nach diesen Vorgaben die Sicherheitsüberprüfungen zu konfigurieren, durch die die Policy umgesetzt werden soll. Dazu werden zunächst verschiedene Monitoring-Komponenten auf den überwachten Systemen installiert, die regelmäßig die überwachten Parameter abfragen und mittels Syslog an das Überwachungssystem übermitteln. Dort empfängt der Prozess Mr.Log2SQL diese

[5] siehe http://www.heise.de/newsticker/data/jk-25.12.00-003

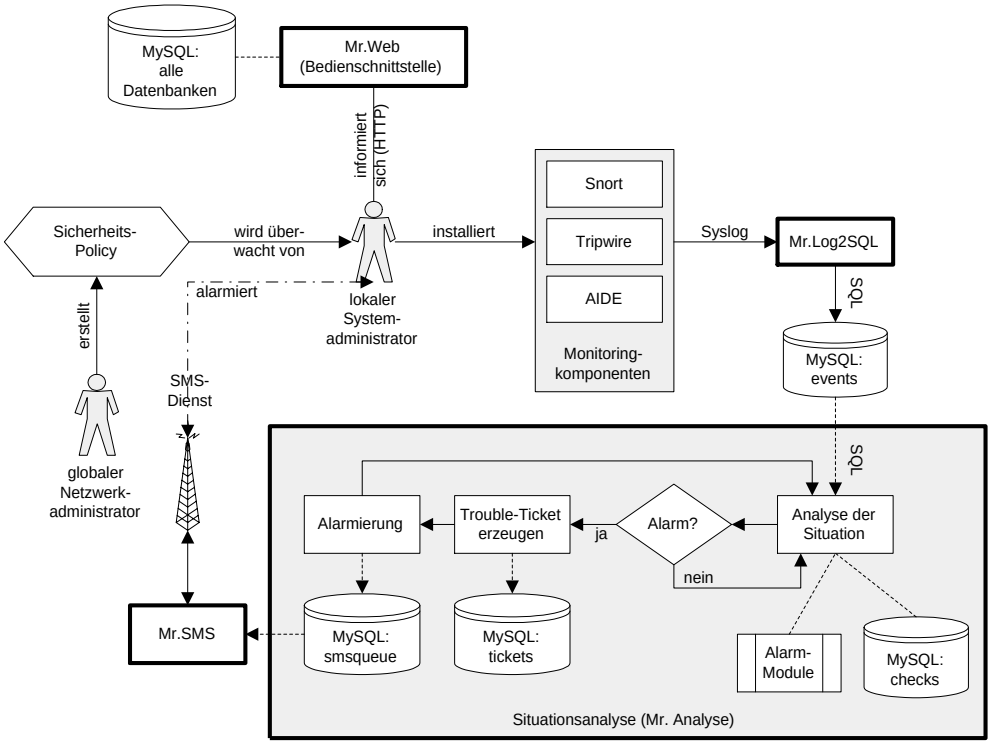

Abbildung 5.3.: Interaktion der implementierten Komponenten

Information und fügt sie in die Tabelle *events* der Datenbank ein. Asynchron zu diesen Vorgängen läuft periodisch der Prozess Mr.Analyse ab, der die Situation anhand der vorgegebenen Überprüfungen aus der Datenbanktabelle *checks* auswertet und für jede Überprüfung das passende Alarmmodul ausführt. Hat sich das Alarmmodul für eine Alarmierung entschieden, so wird ein Trouble-Ticket erzeugt und eine Nachricht an die Administratoren des IRTs (Incident Response Teams) gesendet.

5.7. Dateien

Dieser Abschnitt soll eine Übersicht über die in der Implementation verwendeten Dateien geben.

Datei	Bedeutung
mranalyse	Daemon-Prozess, der periodisch anhand der *checks* und *events* die aktuelle Situation analysiert und über Störungen alarmiert.
mrlog2sql	Hilfsprogramm zur Eintragung der Monitoring-Daten in die Tabelle *events*, sobald Informationen über Syslog an das zentrale Überwachungssystem übertragen werden.

mrsms	Daemon-Prozess zur Ansteuerung des GSM-Modems. Versendet periodisch anstehende Nachrichten aus der *smsmqueue* und reagiert auf eingehende Anfragen von Systemadministratoren.
AModule.pm	Objektklasse von Alarmmodulen, die vom Analyse-Prozess instanziiert werden.
MrDB.pm	Objektklasse von Hilfsmethoden zur Abwicklung von Datenbanktransaktion (u.a. für Trouble-Tickets)
MrWeb.pm	Objektklasse von Hilfsmethoden für das Web-Frontend
backup	Nächtlich ausgeführter Prozess, der die Inhalte der Datenbank extrahiert und sichert.
findmissingchecks	Hilfsprogramm zum Aufspüren von *events*, die noch von keiner Überprüfung behandelt werden.
modulchecker	Hilfsprogramm zum Testen von Alarmmodulen.
setadminpw	Hilfsprogramm zum Setzen des Zugangspassworts für einen Systemadministrator.
cgi/mrweb	Web-Frontend-CGI
html/analyserun	Vom Analyseprozess erzeugte Datei, die Auskunft über Startzeitpunkt und Dauer des letzten Analysedurchlaufs gibt. Wird verwendet zur Anzeige im Web-Frontend.
html/index.shtml	Hauptseite des Web-Frontends mit Frames.
html/mrweb.css	Stylesheet zur Definition von Attributen zur Darstellung der Webseiten im Browser.
html/nav.php3	Linker Frame des Web-Frontends.
html/titel.php3	Rechter Titel-Frame des Web-Frontends.

5.8. Tests

In den letzten Kapiteln wurde die Implementation im Detail erklärt. Natürlich sind trotz sorgfältiger Planung und Ausführung Fehler nicht immer auszuschließen. Deshalb war es nötig, das Überwachungssystem zu testen — insbesondere bei der Einführung neuer Alarmmodule oder Monitoring-Komponenten. Ein korrekter Weg wäre ein realer Test, der die Abschaltung von Netzwerkdiensten bedeuten würde. Bei manchen Diensten mag das für kurze Tests möglich sein, wie beim Überfüllen von Plattenpartitionen, aber wichtige Webserver abzuschalten, ist leider nicht immer möglich.

Um möglichst das Zusammenspiel aller Komponenten zu testen (und nicht nur jede Komponente einzeln), musste ein Weg gefunden werden, im System Veränderungen der Gesamtsituation zu simulieren. Wie man in Abbildung 5.3 erkennen kann, müssen die Ereignisse bewusst generiert werden, die die Monitoring-Komponenten über den Prozess `Mr.Log2SQL` in die Datenbanktabelle *events* schreiben. Es wäre nicht zweckdienlich, die Einträge in der Datenbank manuell zu verändern, da regelmäßig neue Informationen vom Monitoring erzeugt werden, die die eigenen Änderungen überschreiben. Außerdem könnte man so nicht die korrekte Funktion des Prozesses `Mr.Log2SQL` überprüfen.

Deshalb wurde der Weg gewählt, eine neue Überprüfung in der Tabelle *checks* anzulegen. Diese Überprüfung verweist dabei auf Informationen von einem nicht existenten

System. Der Testeintrag sah so aus:

Feld	Inhalt
id	9999
src	test
dst	test
amodule	DISK
apar1	/tmp
apar2	90
apar3	
aparams	
status	unknown
info	
admdomain	test
ticketid	
alertdelay	0
maintenance	n
alerttimes-monday	0...23
⋮	
alerttimes-sunday	0...23

Mit diesem Eintrag wird später überprüft, ob die Füllung der Partition `/tmp` auf dem System `test` unter 90% liegt. Zusätzlich war es notwendig, die administrative Domäne „test“ in der Tabelle *admdom* anzulegen:

Feld	Inhalt
id	8888
domain	test
admins	haas
info	Testdomäne

Mit diesen Einträgen war es dann möglich, einen Syslog-Eintrag an `Mr.Log2SQL` zu senden. Dazu wurde der UNIX-Befehl `logger` verwendet. Hier ein Beispielaufruf:

```
logger -p local0.emerg "Jan 1 00:00:00 none pikt[0]: EMERG: 968717835
test test DISK (/tmp,92) error '92%'"
```

Dieser Eintrag gibt vor, dass auf einem System namens `test` die Füllung der Partition `/tmp` bei 92% liegt. Wenn man diesen Wert mit dem Schwellwert „90%“ vergleicht, müsste dieser Wert zu einem Alarm führen. Auf diese Weise konnte überprüft werden, ob das Alarmmodul, der Prozess `Mr.Analyse` und die Alarmierungsroutinen einwandfrei funktionieren. Die Funktion des Prozesses `Mr.SMS` konnte durch einfügen von Datensätzen in die Tabelle *smsqueue* leichter überprüft werden.

Zur Überprüfung der Alarmmodule wurde ein zusätzliches Perl-Skript namens `modulchecker` programmiert, mit dem sich einzelne Überprüfungen aus der Tabelle *checks* ausführen lassen. So lassen sich verschiedene Entscheidungszweige eines Alarmmoduls ausprobieren.

Durch diese Vorgehensweise waren umfangreiche Tests auch während der Betriebsphase des Überwachungssystems möglich. Die einzig unberücksichtigten Teile durch diese Tests waren die Monitoring-Komponenten. Diese stellen aber keinen Teil der Implementation dar. Man müsste die Funktionsweise jeder einzelnen Komponente einzeln testen. Da es nicht in den Themenbereich dieser Arbeit gehört, die vorhandenen Sicherheitsprogramme Tests zu unterziehen, wird sich auf deren Funktion verlassen. Einfachere Tests wie die Füllung von Festplatten zu erhöhen oder Systeme zu belasten, wurden aber durchgeführt, um einzelne Monitoring-Instanzen zu testen.

6. Resümee

6.1. Ergebnisse

In dieser Arbeit wurden zunächst beispielhaft Angriffsmöglichkeiten eines mit dem Internet verbundenen Unternehmens beschrieben. Es wurden Auffälligkeiten und Gegenmaßnahmen aufgezeigt, damit die passive Sicherheit eigener Netzwerksysteme gegen Angriffe systematisch erhöht werden kann. Dabei wurde auch deutlich, dass sich auch beim Einsatz einer sorgfältig erstellten Sicherheitspolicy nicht alle Bedrohungen vollständig eliminieren lassen.

Um einen zusätzlichen Schutz zu erhalten, wurde eine aktive Absicherung der Systeme gefordert. Es stellte sich heraus, dass sich zentrale Teile der Sicherheitspolicy automatisch überwachen lassen. Bei der Implementation wurde außerdem Wert auf eine einfache Benutzbarkeit gelegt. Deshalb wurde zusätzlich zur grundlegenden Überwachung noch eine komfortable Alarmierung über den GSM-Kurznachrichtendienst und eine Verwaltung von Trouble-Tickets implementiert.

Das implementierte Programmpaket wurde etwas sechs Monate lang getestet und hat einige Anomalien und Systemstörungen alarmiert, die möglicherweise ohne das System zu spät oder gar nicht erkannt worden wären. An diesem Praxistest waren 13 Systemadministratoren beteiligt, die in 29 administrativen Domänen organisiert waren. Jede dieser Domänen bildete eine Menge der Systemadministratoren und formierte damit einzelne IRTs. Insgesamt wurden 44 Systeme in drei Niederlassungen von den Reporting-Instanzen

- AIDE und
- PIKT

periodisch überwacht. Daraus ergaben sich insgesamt 186 periodisch erneuerte Informationen, die als Grundlage für die Situationsanalyse herangezogen wurden. Diese Überprüfungen waren

- ID-Überprüfungen mittels AIDE,
- Temperaturüberwachungen,
- Überwachungen der Füllung von Partitionen,
- Verfügbarkeitsüberwachungen von Diensten auf Basis einfacher TCP-Verbindungen (ohne Protokollunterstützung) und
- die Überwachung der Verbindung zum Internet mit allen beteiligten Komponenten.

Für einen vollständigen Durchlauf aller Überprüfungen wurden 3,8 Sekunden benötigt.

6.2. Offene Probleme

Angesichts der Tatsache, dass die Implementation von der Firma Netlife als Grundlage eigener Sicherheitssysteme zeitnah benötigt wurde, fiel die Evaluationszeit für die Auswahl der Teilkomponenten leider vergleichsweise kurz aus. Auch deshalb konnten nicht alle Auswirkungen der Planung vollständig durchdacht werden. Aber auch unvorhergesehene Probleme erschwerten die Programmierung.

1. Von wichtigen Systemen wurden häufig mehrere Parameter überwacht. Beim Ausfall des gesamten Systems wurden so alle mit dem System verbundenen Alarme getriggert. Im Extremfall waren dies bis zu 10 Meldungen. Hier müsste eine geeignetere Alarmierung gefunden werden, die diese Alarme zusammenfasst.
2. Das GSM-Modem erwies sich während der Testzeit als nicht völlig zuverlässig. Es kam schätzungsweise einmal pro Woche zu einer nicht reproduzierbaren Fehlfunktion beim Versenden von Kurznachrichten.
3. Die bisherige Sicherheitsüberwachung verfolgt einen zentralisierten Ansatz. Würde das zentrale System ausfallen, wäre die gesamte Überwachung nicht mehr funktionsfähig. Hier könnte eine redundante Auslegung die Gesamtsicherheit erhöhen.

6.3. Erweiterungsmöglichkeiten

Die Implementation erhebt nicht den Anspruch, alle denkbaren Anwendungsfälle abzudecken. Im Laufe dieser Arbeit wurden einige Aspekte aufgezeigt, um die die Implementation erweitert werden könnte:

1. Die implementierte Kommunikation über SMS könnte durch eine komfortablere Methode ersetzt werden. Unter Verwendung des *Wireless Application Protocol* (WAP) könnte der Systemadministrator jederzeit von einem GSM-Mobiltelefon aus Status-Informationen abrufen oder Entscheidungsabläufe steuern. Außerdem könnte so die Sicherheit durch bessere Authentisierungsmöglichkeiten beim WAP erhöht werden.
2. Das Programmpaket `PIKT` erwies sich in einzelnen Überwachungen als zu umfangreich. Es wäre denkbar, die Monitoring-Instanzen durch eine Abfrage mittels des SNMP-Protokolls zu ersetzen. In ersten Tests erwies sich der Net-SNMPD als eine praktische Komponente zum Monitoring.
3. Es wäre theoretisch möglich, die verwendeten Perl-Module zu kompilieren. So könnte die Ablaufgeschwindigkeit der Situationsanalyse gesteigert werden.
4. Die Konfiguration der Überprüfungen findet bisher nur durch direkte Änderung der Datenbanktabellen statt. Hier wäre eine komfortablere Bedienung als Teil der Bedienschnittstelle sinnvoll.

A. Quellcode

A.1. Situationsanalyse: Mr.Analyse

```
1   #!/usr/bin/perl -w
2   #
3   # Mr. Analyse
4   #
5   # Network situation analysis tool. Its purpose is to analyse the
6   # current data (which was given by the monitoring instances)
7   # from the database's events table by means of the alarm modules
8   # supported in the AModule.pm class.
9   #
10  # Version 2.17
11  #

12  # Don't buffer outputs.
13  $|=1;

14  # Set the name of this process to let it be monitored by the PIKT system.
15  $0="mranalyse";

16  # Delay between the cycles of checking in seconds.
17  my $loopdelay=0;

18  # Path of the sendmail program and recipient user for sending emergency mails.
19  my $SENDMAIL="/usr/sbin/sendmail";
20  my $EMERGENCYADMIN="alertadmin@alertdomain.de";

21  # Force the Perl interpreter to be strict at using variables.
22  use strict;

23  # Include the DataDumper class providing a method to print any data structures
24  # in a human readable format for debugging.
25  use Data::Dumper;

26  # Include constants needed for flock.
27  use Fcntl ':flock';

28  # Include the AModule class containing the alarm modules.
29  use AModule;

30  # Include the MrDB class providing access to the MySQL database
31  use MrDB;

32  # Instantiate a MrDB object for getting access to the MrDB object class
33  # provided by MrDB.pm
34  my $MrDB = new MrDB;

35  # Instantiate an AModule object for getting access to the alarm module class
36  # provided by AModule.pm
```

```
37   my $AModule = new AModule;

38   ##############################################################################
39   #
40   # Signal handler for catching interrupt (INT) signals
41   #
42   # This handler logs the event of being interrupted and sends a mail
43   # to the EMERGENCY admin.
44   #
45   local $SIG{INT} = sub
46      {
47         $MrDB->log("analyse", "fatal", "Program terminated by signal INT");
48         $MrDB->dblog("Analyse process was killed.");
49         system("echo 'MrAnalyse died by INT' | mail $EMERGENCYADMIN");
50         exit;
51      };

52   ##############################################################################
53   #
54   # This perl script might also be run providing an optional argument giving
55   # the debug level (1-3). A debug level of 1 results in very few output
56   # and mainly describes the decision processes of Mr.Analyse. Debug levels
57   # of 2 or 3 result in very chatty output providing every single data
58   # structure.
59   #
60   if ($#ARGV==0)
61   {
62      print "Running Mr.Analyse in test mode...\n";
63      $MrDB->debug($ARGV[0]);
64      print "Consistency check...\n";
65      &consistency_check;
66      print "Testing checks...\n";
67      &dochecks;
68      exit;
69   }

70   ##############################################################################
71   #
72   # Trap manager for catching fatal errors when the programs fails.
73   #
74   eval '&main';
75   if($@)
76   {
77      &fatal("Fatal error caught of mranalyse: '$@'\n");
78   }
79   exit;

80   ##############################################################################
81   #
82   # The main routine called by the trap manager
83   #
84   sub main
85   {
86      # Consistency check the checks, admin and admdom database tables.
87      $MrDB->log("analyse", "info", "Consistency checking database...");
88      &consistency_check;

89      # Mr.Analyse now begins the main decision process.
90      $MrDB->log("analyse", "info", "Analyse is starting up!");
91      $MrDB->dblog("Analyse process started up.");

92      my $starttime;      # Local variables for measuring the time used to
```

```
 93      my $endtime;      # run through the checks.

 94      # Execute the main loop time and again
 95      while (1)
 96      {
 97         my $starttime=time;   # Get the start time.
 98         &dochecks;            # Execute all checks of the checks table.
 99         my $endtime=time;      # Get the end time.

100         # A file called 'analyserun' will be saved containing information
101         # about how long mranalyse took to stalk through the checks
102         # and when the time was mranalyse last did this checks.
103         umask 0113;     # Set the file permissions to: u=rw, g=rw, o=r

104         # Open the file for writing.
105         open (F, ">/usr/local/mrnetwork/html/analyserun")
106            or $MrDB->log("analyse", "warn", "Could not write status file");

107         # Try to get an exclusive lock.
108         # LOCK_EX means exclusive lock (read and write lock).
109         # LOCK_NB means non-blocking lock. If flock cannot immediately lock
110         # the file then this command will not wait for one.
111         unless (flock(F, LOCK_EX|LOCK_NB))
112         {
113            $MrDB->log("analyse","warn","Waiting for write lock to status file");
114         }

115         # Wait for getting the exlusive lock.
116         unless ( flock (F, LOCK_EX) )
117         {
118            &fatal("Could not lock analyserun file for writing.");
119         }

120         # Write the duration of the last dochecks run to the file.
121         print F ($endtime-$starttime)."\n";

122         # Write the ticks the last dochecks run occured to the file.
123         print F "$endtime\n";

124         # Unlock and close the file.
125         flock (F, LOCK_UN);
126         close F;

127         # Delay the execution of the next dochecks run to keep the system
128         # load low.
129         sleep $loopdelay;
130      }
131   }

132   ############################################################################
133   #
134   # DOCHECKS
135   #
136   # Browse through all checks fetched from the database's checks table
137   # and decide about the situation for each.
138   #
139   sub dochecks
140   {
141      # Query a list of all checks ordered by alarm module first and id second
142      # from the checks table.
143      my $qhandle = $MrDB->sql_query("select * from checks order by amodule,id");
```

```
144      # Get the SQL query's results
145      while (my $check = $MrDB->sql_result($qhandle))
146      {
147         $MrDB->dprint(1, "-----------------------------------");
148         $MrDB->dprint(1, "Now handling check #".$check->{id});
149         $MrDB->dprint(1, "Class is '".$check->{amodule}."'");

150         $MrDB->dprint(3, "Dumping tuple structure from 'checks'");
151         $MrDB->dprint(3, Dumper($check));

152         # Each check provides the name of the alarm module which needs to
153         # be executed to make a decision. Get the name of the alarm module.
154         my $amodule = $check->{amodule};

155         # Execute the alarm module by calling the matching method from the
156         # AModule object class. Provide the alarm module with the current
157         # checks entry from the database.
158         my $amodule_res = $AModule->$amodule($check)
159            or &fatal("Error executing alarm module '$AModule->$amodule'");

160         # The variable $amodule_res is now a hash reference containing
161         # these entries:
162         # - info (text message describing the result of the decision process)
163         # - decision ('alert', 'warn', 'ok' or 'unknown')
164         # - parameters (checked parameters formatted for display)
165         # - time (time on which the decision is based)

166         # Assign the database entries to local variables making the
167         # following script code more readable.
168         my $id=$check->{id};
169         my $src=$check->{src};
170         my $dst=$check->{dst};
171         my $apar1=$check->{apar1};
172         my $apar2=$check->{apar2};
173         my $apar3=$check->{apar3};
174         my $status=$check->{status};
175         my $maintenance=$check->{maintenance};

176         $MrDB->dprint(2, "AModule has decided '".$amodule_res->{decision}."'");
177         $MrDB->dprint(3, "AModule returned this hash:");
178         $MrDB->dprint(3, Dumper($amodule_res));
179
180         # Don't do anything if the check was set to 'maintenance' mode
181         # but writing the status back to the checks table.
182         if ($maintenance eq "y")
183         {
184            $MrDB->set_checks_info({
185               checkid=>$id,
186               aparams=>$amodule_res->{parameters},
187               info=>"This check was put offline for maintenance",
188               status=>"warn",
189               });
190            next;
191         }

192         # Handle the alarm module's decision.

193         # Is the decision an alert?
194         if ($amodule_res->{decision} eq "alert")
195         {
196            $MrDB->dprint(1, "Status is ALERT.");
197            $MrDB->dprint(2, "Is this a new alert? (is there yet no ticket?)");
```

```
198            # If there is an open ticket for this check we already know
199            # about the alert and don't have to bother.
200            if ($MrDB->is_open_ticket($check->{ticketid}))
201            {
202               $MrDB->dprint(1, "Ticket already created (".
203                  $check->{ticketid}."). Ignoring alert.");
204            }

205            # Otherwise this alert has yet been unnoticed and needs to
206            # be attended to.
207            else
208            {
209               $MrDB->dprint(1, "No ticket yet created. Handling new alert!");

210               # Assign @alertadmins an array of admin's names (as saved in the
211               # admins table) who are responsible for this check.
212               # $alertadmins becomes a comma seperated list of these names.
213               my @alertadmins = $MrDB->get_domain_admins($check->{admdomain});
214               my $alertadmins = join(',', @alertadmins);

215               # Check whether this alert is less than "alertdelay" minutes old.
216               # During the 'alertdelay' period we only set throw a warning.
217               my $age = $MrDB->timestamp_age($amodule_res->{time})/60;
218               if ( $age < $check->{alertdelay})
219               {
220                  $amodule_res->{decision}="warn";
221                  $amodule_res->{info}.=" (This message will turn into ".
222                     "an alert in ".int($check->{alertdelay}-$age)." minutes)";
223               }

224               # Check whether the alert can occur now according to the
225               # 'alerttimes' fields in the checks table.
226               elsif ( $MrDB->alerttimevalid($check) )
227               {
228                  # Alert is not delayed. Alert is commited!
229                  # First create a new trouble ticket.
230                  my $newticketid = $MrDB->create_ticket({
231                     status=>"alerted",
232                     info=>"Mr.Analyse created this ticket due to a decision ".
233                        "of the alarm module '$amodule' executed by check #$id",
234                     cause=>$amodule_res->{info},
235                     lastnotified=>$alertadmins
236                     });
237                  $MrDB->log("analyse", "info", "Ticket #$newticketid created.");
238                  $MrDB->dprint(2, "Ticket ID #$newticketid is being ".
239                     "entered into checks table");

240                  # Write the number of this new ticket to the checks table.
241                  $MrDB->sql_query("update checks set ticketid=$newticketid ".
242                     "where id=".$check->{id});

243                  # Send alert messages to the responsible administrators
244                  # (according to the administrative domain) that a new
245                  # ticket was created.
246                  $MrDB->alert_admins({
247                     admins=>[@alertadmins],
248                     info=>"New ticket #$newticketid: ".$amodule_res->{info},
249                     });
250                  $MrDB->log("analyse", "info", "Alerting admins: $alertadmins");
251                  $MrDB->dblog("New ticket #$newticketid. ".
252                     "Alerting '$alertadmins'.");
```

```
253             }
254             else
255             {
256                # Alert is suspended due to the 'alerttimes' in the
257                # checks table and will only bring up a warning.
258                $MrDB->dprint(2, "Alert must not accur now according to the ".
259                   "alerttime you specified in the checks table");
260                $amodule_res->{decision}="warn";
261                $amodule_res->{info}.=" [Alert is suspended]";
262             }
263          }
264       }

265       # Is the decision of the alarm module 'ok'?
266       elsif ($amodule_res->{decision} eq "ok")
267       {
268          # If we find an open ticket for this check then there has
269          # been an alert state formerly which now turned back to okay.
270          $MrDB->dprint(1, "Status is OK.");
271          if ($MrDB->is_open_ticket($check->{ticketid}))
272          {
273             $MrDB->dprint(1, "Still an open ticket (".$check->{ticketid}.
274                "). Revoking it.");
275             $MrDB->log("analyse", "info", "Revoking ticket #".
276                $check->{ticketid}.". Problem has vanished.");

277             # Send an all-clear message of this alert to the administrators
278             # who were last responsible for the appropriate trouble ticket.
279             my $ticket = $MrDB->get_ticket($check->{ticketid})
280                or &fatal("Could not get ticket #".$check->{ticketid}.
281                   " while revoking alert for check #$id");

282             # If the ticket was only 'alerted' then alert the whole admdomain
283             # as the ticket was revoked before anyone took it.
284             if ($ticket->{status} eq "alerted")
285             {
286                $MrDB->alert_admins({
287                   admins=>[$MrDB->get_domain_admins($check->{admdomain})],
288                   info=>"Ticket #".$check->{ticketid}." revoked because: ".
289                      $amodule_res->{info},
290                   });
291                $MrDB->log("analyse", "info", "Informing admins that the ".
292                   "ticket was revoked.");
293                $MrDB->dprint(2, "Informing all admins of domain '".
294                   $check->{admdomain}."' that the ticket was cancelled");
295                $MrDB->dblog("Ticket #".$check->{ticketid}." was revoked ".
296                   "before anyone took it.");
297             }

298             # But if the ticket is 'taken' or 'deferred' then alert the
299             # admin who this ticket was assigned to only.
300             elsif ($ticket->{status} eq "taken" ||
301                $ticket->{status} eq "deferred")
302             {
303                $MrDB->alert_admins({
304                   admins=>[$ticket->{admin}],
305                   info=>"Ticket #".$check->{ticketid}." revoked because: ".
306                      $amodule_res->{info},
307                   });
308                $MrDB->dprint(2, "Informing responsible admin '".
309                   $ticket->{admin}."' that the ticket was cancelled");
310                $MrDB->log("analyse", "info", "Informing admin ".
```

```
311                     $ticket->{admin}." (who took the ticket) that the ticket ".
312                        "was revoked.");
313                  $MrDB->dblog("Ticket #".$check->{ticketid}.
314                     " revoked which was assigned to '".$ticket->{admin}."'");
315               }

316               # Handle cases of inconsistent data.
317               else
318               {
319                  &fatal("Ticket's status is not alerted,taken or deferred. ".
320                     "It is '".$ticket->{status}."'. How come?");
321               }

322               # Cancel the ticket at last
323               $MrDB->cancel_ticket({
324                  id=>$check->{ticketid},
325                  info=>"Problem has vanished.",
326                  });
327               $MrDB->dprint(2, "Ticket cancelled.");
328            }

329            # The decision for this check is still 'ok'. Do nothing.
330            else
331            {
332               $MrDB->dprint(1, "No open ticket.");
333            }
334         }

335         # If the alarm module decided 'warn' then do nothing.
336         elsif ($amodule_res->{decision} eq "warn")
337         {
338            $MrDB->dprint(1, "We do not handle cases of 'warn'. Ignoring...");
339         }

340         # Handle unsupported decisions.
341         else
342         {
343            &fatal("Unexpected decision '".$amodule_res->{decision}."'");
344         }

345         # Set the status in the checks table to reflect the
346         # current check's result.
347         $MrDB->set_checks_info({
348            checkid=>$id,
349            aparams=>$amodule_res->{parameters},
350            info=>$amodule_res->{info},
351            status=>$amodule_res->{decision}
352            });
353      }
354   }   # end while(TRUE)

355   # Send email in case of unrecoverable errors
356   sub fatal
357   {
358      # not shown here
359   }

360   # Consistency check data entries in the database
361   sub consistency_check
362   {
363      # not shown here
364   }
```

A.2. Apache-Modperl-Authentisierungsmodul: AuthenMySQL

```
1  package Apache::AuthenMySQL;
2  use strict;
3  use Apache::Constants qw/:common/;
4  use vars qw/%ENV/;
5  use DBI;
6
7  $Apache::AuthenMySQL::VERSION = '1.0';
8  my $self="Apache::AuthenMySQL";
9
10 sub handler
11 {
12    # get request object
13    my $r = shift;
14
15    # service only the first internal request
16    return OK unless $r->is_initial_req;
17
18    # get password which the user entered in the browser
19    my($res, $sent_pwd) = $r->get_basic_auth_pw;
20
21    # decline if authentication method is not basic
22    return $res if $res;
23
24    # get user name
25    my $name = $r->connection->user;
26
27    # be sure username is reasonable
28    $name=~m/(\w+)/; $name=$1;
29
30    # blank user name would cause problems
31    unless($name)
32    {
33       $r->note_basic_auth_failure;
34       $r->log_reason("$self: no username supplied", $r->uri);
35       return AUTH_REQUIRED;
36    }
37
38    # load apache config vars
39    my $dir_config = $r->dir_config;
40    my $dbname = $dir_config->get("AuthenMySQL_dbname");
41    my $dbuser = $dir_config->get("AuthenMySQL_dbuser");
42    my $dbpw = $dir_config->get("AuthenMySQL_dbpw");
43    my $dbtable = $dir_config->get("AuthenMySQL_dbtable");
44
45    die "FATAL: Missing AuthenMySQL_dbname" unless $dbname;
46    die "FATAL: Missing AuthenMySQL_dbuser" unless $dbuser;
47    die "FATAL: Missing AuthenMySQL_dbpw" unless $dbpw;
48    die "FATAL: Missing AuthenMySQL_dbtable" unless $dbtable;
49
50    # get password from sql table
51    my $db = DBI->connect("DBI:mysql:".$dbname,$dbuser,$dbpw)
52       or die "FATAL: Error opening database pikt";
53    my $handle = $db->prepare("select * from $dbtable where name=?");
54    $handle->execute($name)
55       or die "FATAL: Error executing admins select query (".$handle->err.")";
56    my $result;
57    unless ($result = $handle->fetchrow_hashref)
58    {
```

```
59          $r->log_reason("User name '$name' not found in database. Access denied.");
60          return AUTH_REQUIRED;
61       }
62       my $sql_pw = $result->{password};
63       undef $handle;
64       $db->disconnect;
65       $handle->finish;
66
67       # check supplied password against the databases user table
68       if (crypt($sent_pwd, $sql_pw) eq $sql_pw)
69       {
70          return OK;
71       }
72       else
73       {
74          $r->log_reason("Passwords do not match. Access denied!", $r->uri);
75          return AUTH_REQUIRED;
76       }
77    }
```

Literaturverzeichnis

[bsi] *Bundesamt für Sicherheit in der Informationstechnik.*
`http://www.bsi.de`

[checkp] *Checkpoint Software Technologies Ltd.*
`http://www.checkpoint.com`

[comcrit] *Common Criteria - The Standard for Information Security.*
`http://www.commoncriteria.org`

[hackbib] CHAOS COMPUTER CLUB, HAMBURG:
Die Hackerbibel - Teil 1.
Grüne Kraft - Medienexperimente, Löhrbach, 1985.

[etsi] *European Telecommunications Standards Institute*.
`http://www.etsi.org`

[intman] RAINER FALK und MARKUS TROMMER:
Integrated Management of Network and Host Based Security Mechanisms.
Lecture Notes in Computer Science, Vol. 1438, 1998.

[intnet] HEINZ-GERD HEGERING und SEBASTIAN ABECK:
Integriertes Netz- und Systemmanagement.
Addison-Wesley, 1. Auflage, 1993.

[iana] *Internet Assigned Numbers Authority*.
`http://www.isi.edu/in-notes/iana/assignments`

[itsec] *The UK ITSEC scheme*.
`http://www.itsec.gov.uk`

[kerner] HELMUT KERNER:
Rechnernetze nach OSI.
Oldenbourg, 1995.

[lipx] ANDREAS STEFFEN:
Projektarbeit: Linux — IPsec mit PGP und X.509-Zertifikaten.
Universität Hannover, 2000.

[php] *PHP: Hypertext Processor.*
`http://www.php.net`

[pikt] ROBERT OSTERLUND: *PIKT*.
`http://pikt.uchicago.edu/pikt`

[modperl] LINCOLN STEIN und DOUG MACEACHERN:
Writing Apache Modules with Perl and C.
O'Reilly, 1999.

[hackexp] STUART MCCLURE, JOEL SCAMBRAY und GEORGE KURTZ:
Hacking Exposed.
O'Reilly, 1999.

[iaag] ANDREAS ENGEL und ANDREAS G. LESSIG:
Diplomarbeit: Internetgestützte Angriffe und ausgewählte Gegenmaßnahmen.
Universität Hamburg, 1999.

[lp] RANDAL L. SCHWARZ:
Learning Perl.
O'Reilly, 1994.

[lpt] NANCY WALSH:
Learning Perl/Tk.
O'Reilly, 1999.

[mysql] TCX DATAKONSULT AB:
MySQL.
`http://web.mysql.com`

[pcb] TOM CHRISTIANSEN und NATHAN TORKINGTON:
Perl Cookbook.
O'Reilly, 1999.

[ps] BIRGIT PFITZMANN:
Practical Security.
Universität Stravensis, 2000.

[pp] LARRY WALL, TOM CHRISTIANSEN und RANDAL L. SCHWARTZ:
Programming Perl.
O'Reilly, 1996.

[isoosi] HELMUT KERNER:
Rechnernetze nach ISO-OSI, CCITT.
Technische Universität Wien, 1989.

[kacwn] KLAUS-PETER KOSSAKOWSKI:
Diplomarbeit: Klassifikation und Abwehr von Computer-Würmern in Netzwerken.
Universität Hamburg, 1992.

[scsmfm] D. E. BELL und L.J. LAPADULA:
Secure Computer Systems: Mathematical Foundations and Model.
Mitre Corporation, 1973

[svsfs] STEFANIE FOX:
Spezifikation und Verifikation eines „Seperation of Duty"-Szenarios als verbindliche Telekooperation im Sinne des Gleichgewichtsmodells.
GMD – Forschungszentrum Informationstechnik GmbH, 1998.

[usah] EVI NEMETH ET AL:
UNIX System Administration Handbook.
Prentice Hall, 2001.

[vkez] KATHRIN SCHIER:
Dissertation: Vertrauenswürdige Kommunikation im elektronischen Zahlungsverkehr.
Universität Hamburg, 1999.

[vlb] RAMES ABDELHAMID:
Das Vieweg-LaTeX-Buch.
Vieweg, 2. Auflage, 1993.

[zkii] PETER FECHT:
Diplomarbeit: Zugriffskontrolle für Internet-Informationssysteme.
Universität Hamburg, 1998.

Index