INSTITUT FÜR INFORMATIK
DER LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN

Diplomarbeit

Virtualisierung einer Praktikumsinfrastruktur zur Ausbildung im Bereich vernetzter Systeme

Tobias Lindinger

Aufgabensteller: Prof. Dr. Heinz-Gerd Hegering

Betreuer: Dr. Helmut Reiser
Nils Otto v.d. gentschen Felde

Abgabetermin: 14. Mai 2006

INSTITUT FÜR INFORMATIK
DER LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN

Diplomarbeit

Virtualisierung einer Praktikumsinfrastruktur zur Ausbildung im Bereich vernetzter Systeme

Tobias Lindinger

Aufgabensteller: Prof. Dr. Heinz-Gerd Hegering

Betreuer: Dr. Helmut Reiser
Nils Otto v.d. gentschen Felde

Abgabetermin: 14. Mai 2006

Hiermit versichere ich, dass ich die vorliegende Diplomarbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

München, den 14. Mai 2006

. . . . . .

(Unterschrift des Kandidaten)

Durch die Verfügbarkeit immer leistungfähigerer Prozessoren und immer billiger werdende Speicherbausteine sind immer mehr Rechner überproportional mit Rechenleistung ausgestattet, die im Praxiseinsatz nicht genutzt wird. Durch Virtualisierungslösungen ist es möglich, dass sich viele PCs mit niedrigen Ansprüchen an die Hardware eine physische Plattform teilen. Dadurch entfällt die Anschaffung mehrerer Rechner zugunsten einem leistungsfähigen Server. In großen Maßstäben können auf diese Weise einfach Kosten gespart werden und die Durchführung einzelner Managementaufgaben, wie das Erstellen von Backups, wird wesentlich erleichtert. Virtualisierungstechnologien eigenen sich daher insbesondere auch für die Realisierung von Infrastrukturen, die zur Ausbildung im Bereich der EDV oder Informatik eingesetzt werden. Ziel dieser Arbeit ist die Erstellung einer solchen Infrastruktur. Angefangen von der Planung und Konzeption der Architektur bis hin zur Implementierung soll gezeigt werden, wie Schritt für Schritt ein existierendes Netzwerk virtuell erzeugt werden kann. Als Beispiel dienen zwei zu virtualisierende Szenarien des an der LMU und TU München angebotenen IT-Sicherheit Praktikums. Da die zur Zeit zur Ausbildung verwendeten Rechner und Netzkomponenten bereits relativ alt und störanfällig sind, soll das Praktikum zukünftig anhand der zu implementierenden virtuellen Umgebung abgehalten werden. Neben der reinen Implementierung der Szenarien, müssen sich daher auch Gedanken über das Management und den Zugang zu den virtuellen Maschinen gemacht werden.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einführung

Seit einiger Zeit häufen sich im Praktikum IT-Sicherheit der Lehr- und Forschungseinheit für Kommunikationssysteme und Systemprogrammierung an der Ludwig-Maximilians-Universität München Probleme mit der im Praktikum verwendeten Hardware. Insbesondere gehen immer wieder einzelne Komponenten der verwendeten Arbeitsplatzrechner kaputt. Dies hat zur Folge, dass die Betreuer oft mehr damit beschäftigt sind Hardware-Probleme zu lösen, als dass sie ihrer eigentlichen Aufgabe nachkommen könnten, die Studenten bei der Lösung der praktischen Aufgaben zu unterstützen.
Des Weiteren ist die Nachfrage bezüglich des Praktikums relativ hoch. Es gibt meist mehr Bewerber als freie Plätze zur Verfügung stehen. Das erste Problem ließe sich zwar durch die Anschaffung neuer PCs beheben, kostet aber Geld und schafft auch nicht mehr Praktikumsplätze, denn für zusätzliche Arbeitsplätze sind schlichtweg keine Räumlichkeiten mehr vorhanden.
Ein Ansatz zur Lösung beider Probleme ist eine Virtualisierung des Praktikums. Dabei werden alle im Praktikum benötigten Hardware-Komponenten softwareseitig auf einem Serversystem erzeugt und sind über das Netz zugreifbar. Wenn alle einzelnen Komponenten, angefangen vom Netzwerkkabel über Switches, Hubs und Netzwerkkarten sowie Rechner und Server richtig konfiguriert sind, kann im Zusammenspiel ein komplexes Netz entstehen. Von außen, das heißt über das Netz betrachtet, ist die Virtualisierung transparent; die simulierten Komponenten scheinen real zu existieren. Die einzelnen virtuellen Rechner lassen sich über das Netz bedienen wie jeder physikalisch existierende Rechner auch. So sollte es auch möglich sein die Aufgaben des Praktikums auf solch einem virtuellen System zu bearbeiten.
Auf diese Art könnten vorhandene PCs im CIP-Pool oder zu Hause verwendet werden, um auf den virtuellen Systemen des Praktikums zu arbeiten. Man benötigt keine weiteren Rechner und separate Räumlichkeiten mehr für das Praktikum. Lediglich die Serverhardware zur Bereitstellung der virtuellen Netzwerke muss untergebracht werden. In einer vorausgehenden Arbeit [Lind 05] wurde die Machbarkeit des Vorhabens bereits untersucht und als realisierbar eingestuft. Ziel der Untersuchung war neben der generellen Machbarkeit auch bereits die Klärung einiger Details zur Realisierung. So sollte analysiert werden, welche Virtualisierungssoftware für das Projekt geeignet ist, welche Hardware benötigt wird, wie das Management des virtuellen Netzes realisiert werden kann und welche Änderungen dazu grundsätzlich am Ablauf des Praktikums notwendig sind. Die Ergebnisse waren:

- Als Virtualisierungssoftware könnte User Mode Linux zum Einsatz kommen.
- Als Hardware eignet sich bereits ein modernes Zweiprozessorsystem mit viel Arbeitsspeicher.
- Die Änderungen am Praktikum sind minimal.

Als Managementlösung sollte VNUML [VNUM 05], ein Managementtool speziell zum Einsatz mit User Mode Linux [Dike 05] geschaffen, verwendet werden. Zum Zeitpunkt der Untersuchung war die Entscheidung für User Mode Linux sicherlich richtig. In der Zwischenzeit hat sich Xen, eine andere Virtualisierungslösung jedoch so rasant weiterentwickelt und wird aller Voraussicht nach auch in Zukunft noch verbessert werden, so dass die Entscheidung für User Mode Linux kaum mehr haltbar ist.
Aus diesem Grund wird diese Entscheidung in dieser Arbeit noch einmal hinterfragt, der Schwerpunkt liegt aber primär auf der Planung und Verwirklichung des Vorhabens. Dazu wird im folgenden Kapitel das Praktikum IT-Sicherheit detaillierter dargestellt um eine konkrete Anforderungsliste an das Virtualisierungstool erstellen zu können. Anschließend wird das Thema Virtualisierung vertieft behandelt, indem zunächst die gängigen Ansätze beschrieben und verbreitete Produkte vorgestellt werden. Die Hauptaufgabe dieser Arbeit ist die Planung und Realisierung der im Praktikum eingesetzten Szenarien als virtuelle Umgebung. Im Kapitel Planung wird die grundlegende Architektur der virtuellen Szenarien vorgestellt und diskutiert, während im Kapitel Realisierung Details behandelt werden, die für eine Implementierung entscheidend sind, nicht jedoch zum Verständnis der groben Funktionsweise. Die beiden letzten Kapitel bewerten die Realisierung und die verwendeten Komponenten hinsichtlich mehrerer Kriterien und versuchen eine Abschätzung des Verhaltens der virtuellen Netzwerke im Alltagseinsatz.

Im Anhang befinden sich neben allen Konfigurationsdateien und Skripten auch eine Installationsanleitung, anhand der das Szenario auf anderen Maschinen installiert werden kann, sowie eine kurze Anleitung zur Benutzung.

2 Grundlagen

2.1 Topologien

Die Aufgabe dieses Kapitels ist es, das Praktikum IT-Sicherheit, welches im Weiteren als Beispielnetz dienen wird, kurz vorzustellen. Dabei werden zunächst die Zielsetzungen des Praktikums erläutert und anschließend die verwendeten Netztopologien samt der darin zu lösenden Aufgaben vorgestellt.

2.1.1 Das Praktikum IT-Sicherheit

Das Praktikum IT-Sicherheit ist ein an der LMU München angebotenes Praktikum, das von der Lehr- und Forschungseinheit für Kommunikationssysteme und Systemprogrammierung durchgeführt wird. Teilnehmen können Studenten der LMU und TU München mit abgelegtem Vordiplom. Ziel des Praktikums ist es, grundlegende Kenntnisse im Bereich der Vernetzung von Computersystemen und deren Absicherung gegenüber Unberechtigten zu vermitteln. Dazu findet einmal pro Woche eine Theorieeinheit statt, deren Inhalt an zwei Vormittagen anhand praktischer Aufgaben vertieft wird. Aufgrund mangelnden Platzes werden die Teilnehmer des Praktikums in den praktischen Einheiten in zwei Gruppen aufgeteilt, die ihre Aufgaben an verschiedenen Vormittagen bearbeiten. Die verwendete Infrastruktur ist für beide Gruppen dieselbe. Dies wird durch den Einsatz zweier parallel installierter Betriebssysteme möglich. Die Dateisysteme sind verschlüsselt, so dass Lösungen, die von einer Gruppe bereits erarbeitet wurden, für die andere Gruppe nicht sichtbar sind. Es arbeiten jeweils zwei Studenten als Team an einem Rechner. Zeitweise arbeiten zwei solcher Teams zusammen, um Client- und Serveranwendungen zu testen. Durchgeführt wird das Praktikum auf dem Betriebssystem Linux. Wie an fast allen Rechnern am Institut, kommt hier die Distribution SuSE zum Einsatz.
Der Schwerpunkt der durchzuführenden Aufgaben liegt in den Bereichen

- Grundlagen von TCP/IP Netzwerken
- Hacking-Angriffe: Portscans, Spoofing, DoS, Passwort-Cracker, Rootkits...
- Statische und dynamische Paketfilter
- Verschlüsselung: Symmetrische ,asymmetrische und hybride Verfahren, Prüfsummen, digitale Signaturen, Zertifikate, VPN
- grundlegende Netzwerk Dienste: DNS, HTTP, SMTP, SSH,...
- Application Level Gateways und Proxys
- Firewallarchitekturen
- Intrusion Detection

Diese Themenbereiche werden in zwei verschiedenen Netzwerkszenarien bearbeitet, deren Aufbau nachfolgend kurz erläutert wird.

2.1.2 Szenarien

Szenario 1

Abbildung 2.1 zeigt das erste Szenario im Praktikum IT-Sicherheit. Die Praktikumsrechner `pcsec`*XX* sind in Fünfergruppen mit je einem Hub verbunden. Ein Rechner pro Gruppe dient als Router ins Kernnetz, in dem ein

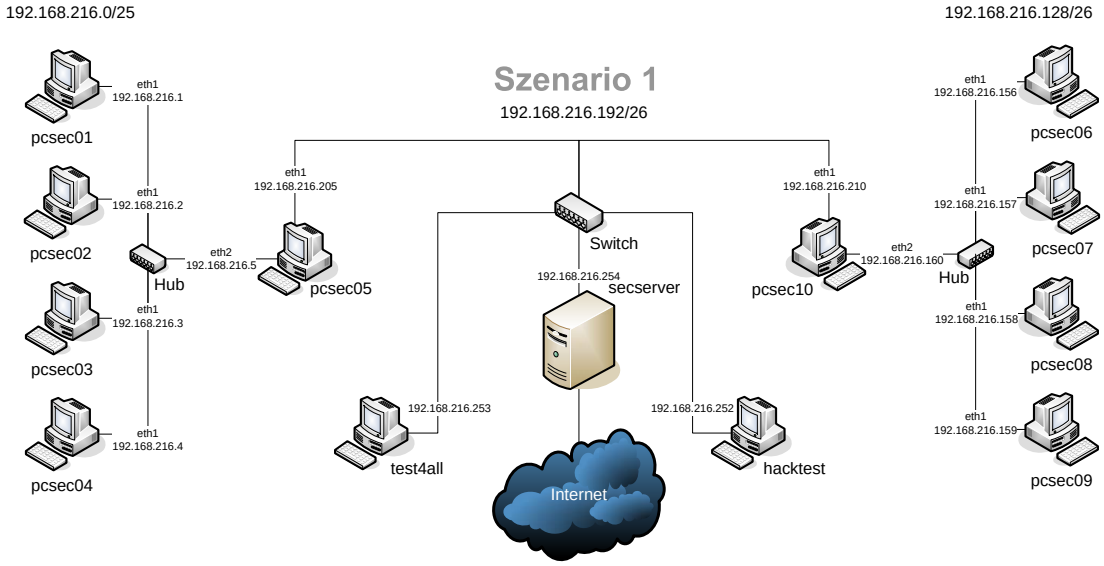

Abbildung 2.1: Szenario 1: Hub-Topologie

Server namens `secserver` Dienste bereitstellt. Der `secserver` arbeitet in diesem Szenario beispielsweise als HTTP-Proxy und NFS-Server zur Bereitstellung der SuSE-CDs. Des Weiteren sind im Kernnetz die Rechner `test4all`, ein Testrechner für unterschiedliche Aufgaben, sowie `hacktest`, ein Rechner mit Rootkit[1], angeschlossen.
Die wichtigsten Aufgaben der Studenten in diesem Szenario sind das Konfigurieren der Netzwerkkarten mit IP-Adresse und Netzmaske sowie das Einrichten des Routing im Netzwerk. Zusätzlich soll das Gefahrenpotential eines Hubs erkannt werden, indem mit den Tools `ngrep` und `tcpdump` Lauschangriffe auf Passwörter in unverschlüsselten Protokolle wie FTP (File Transfer Protokoll) und Telnet gestartet werden. Da die Hubs den Datenverkehr nicht ausschließlich an den jeweiligen Zielrechner weiterleiten, sondern weiterzuleitende Datenpakete auf allen Ports ausgeben, sollte ihr Einsatz in größeren Netzwerken wohl bedacht werden. Diese Gefahren beinhalten aber auch viele billigere Switches, wie sie zum Beispiel in DSL-Routern zum Einsatz kommen. Im Falle zu hoher Netzlast, welche sehr einfach erzeugt werden kann, läuft die interne Adresstabelle über und der Switch muss, wenn er keine Daten verwerfen will, im Hub-Modus weiterarbeiten. Daher sind die in diesem Versuch gewonnen Ergebnisse durchaus von Bedeutung, auch wenn Hubs mittlerweile durch billiger werdende Switches vom Markt verdrängt werden. Weitere behandelte Punkte sind DoS[2]-Werkzeuge, Rootkits, Portscans, sowie der Securityscanner `Nessus`.

Szenario 2

Im zweiten Szenario (Abb. 2.2) entfallen die Hubs. Die Rechner hängen in Teams zu zwei Rechnern am zentralen Switch. Ein Rechner pro Team dient als Router für den jeweils anderen. Das Kernnetz bleibt unverändert. Nach dem Anpassen der Netzwerkkonfiguration auf die veränderte Umgebung werden statische

[1] Sammlung von Softwarewerkzeugen, die nach dem Einbruch in ein Computersystem auf dem kompromittierten System installiert werden, um künftige Aktionen des Hackers zu vereinfachen beziehungsweise zu verbergen.
[2] Denial of Service; Außerstandsetzung eines Netzwerkdienstes durch Überlastung

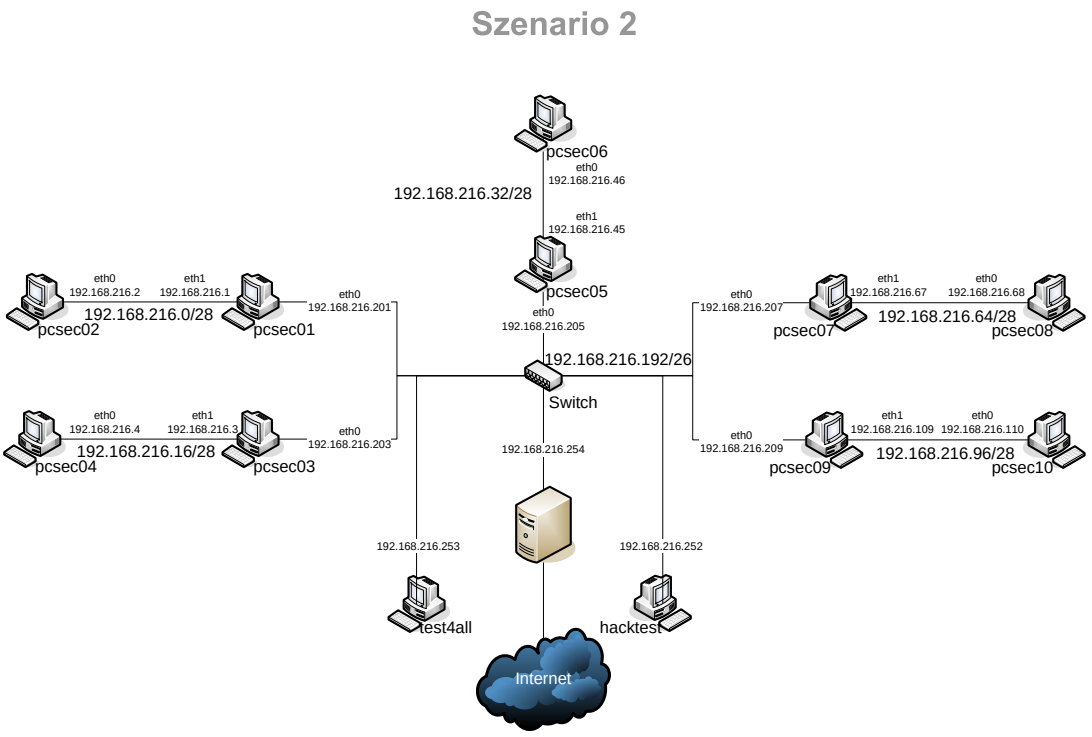

Abbildung 2.2: Szenario 2: Switch-Topologie

und dynamische Paketfilter Firewalls sowie Antispoofing und NAT (Network Address Translation) behandelt. Weitere Themen sind VPN[3], die Standarddienste DNS (Domain Name System), FTP, SSH (Secure Shell), Telnet, SMTP (Simple Mail Transfer Protocol) und HTTP (Hyper Text Transfer Protocol), sowie ihre passenden Application Level Gateways und Intrusion Detection Systeme.
All diese Aufgaben sind typische Client-Serveranwendungen, daher sollen die Studenten in diesem Szenario auch im Team mit einem zugewiesenem Partnerrechner arbeiten. Ziel dieser Aufgaben ist es, neben einem grundlegenden Verständnis für das Funktionieren von Netzwerken, einen Einblick in die wichtigsten Netzwerkdienste zu geben. Zusätzlich zu den konkret behandelten Themenbereichen soll im Praktikum generell das Sicherheitsbewusstsein beim Entwurf von Netzarchitekturen und bei der Installation von Diensten erhöht werden, denn nur durch Verstehen der Gesamtarchitektur und aller verwendeten Dienste, ist es möglich sichere Netze aufzubauen.

2.2 Virtualisierung

Im Bereich der Informatik existieren mittlerweile viele und sehr unterschiedliche Konzepte und Technologien, die alle mit Virtualisierung umschrieben werden. Eine eindeutige, stichfeste Definition für den Begriff „Virtualisierung" zu finden ist daher nicht möglich. Einen Versuch wagt die online Enzyklopädie WIKIPEDIA:
„[...] Virtualisierung bezeichnet Methoden, die es erlauben Ressourcen eines Computers aufzuteilen. Primäres Ziel ist, dem Benutzer eine Abstraktionschicht zur Verfügung zu stellen, die ihn von der eigentlichen Hardware - Rechenleistung und Speicherplatz - isoliert. Eine logische Schicht wird zwischen Anwender und Ressource eingeführt, um physische Gegebenheiten der Hardware zu verstecken. Dabei wird jedem Anwender (so gut es geht) vorgemacht, dass er der alleinige Nutzer einer Ressource sei. Die für den Anwender unsichtbare bzw. transparente Verwaltung der Ressource ist dabei in der Regel die Aufgabe des Betriebssystems. [...]" [WIK 05,

[3] Virtual Private Network; die Funktionsweise von VPNs erläutert Abschnitt 2.4

Stichwort: Virtualisierung]
Laut dieser Definition wäre bereits das 1959 aufgekommene Multiprogramming als Virtualisierung zu werten. Als Multiprogramming bezeichnet man die Fähigkeit eines Betriebssystems mehrere Prozesse annähernd gleichzeitig ausführen zu können, indem die einzelnen Prozesse in sehr kurzen Zeitabständen alternierend in Bruchstücken abgearbeitet werden. Ziel des Verfahrens ist es nicht nur dem Nutzer gleichzeitig laufende Programme zu simulieren, sondern auch die Hardware des Systems optimal auszulasten. Dies erreicht man hauptsächlich durch das Suspendieren eines auf Daten wartenden Prozesses. Während der Wartezeit wird ein rechenbereiter Prozess aktiv und kann in der ansonsten verlorene Zeitspanne den Prozessor nutzen, was den Durchsatz eines Systems erhöhen kann. Die angeführte Definition für Virtualisierung aus dem WIKIPEDIA ist damit voll erfüllt. Diese Arbeit beschäftigt sich jedoch mit dem Erstellen virtueller Netzwerke und damit zwangsläufig mit dem Erstellen virtueller Maschinen. Daher muss die Definition von Virtualisierung konkretisiert werden, wenn man sie nur auf komplette Rechner anwenden und allgemeine Fälle, wie den obigen, ausschließen möchte. Eine passendere Definition wäre etwa: *„Virtualisierung bezeichnet den Vorgang, mehrere Betriebssysteminstanzen unabhängig von einander parallel auf einer physischen Umgebung laufen zu lassen."*
Die Vorstellung dahinter ähnelt einem Multiprogramming auf Ebene der Betriebssysteme, die in diesem Kontext als Programme anzusehen sind. Die Definition ist also nur ein Spezialfall der allgemeinen Variante. Damit bleibt diese auch für den konkreten Fall uneingeschränkt gültig.
Wie aktuell das Thema Virtualisierung in der Informatik ist, zeigt sich durch immer wieder neue entstehende Firmen, die Produkte rund um Virtualisierungslösungen auf den Markt bringen. Jüngstes Beispiel ist die neu gegründete Firma Parallels, die Anfang September 2005 ihre gleichnamige Software [PAR 05b] veröffentlichte und damit ein weiteres Konkurrenzprodukt zu VMware [VMW 05c] und MS Virtual PC [MSV 05] bietet. Einen detaillierteren Überblick zu verfügbaren Virtualisierungslösungen auf Softwarebasis bietet der Abschnitt 2.2.2. Zuvor erläutert das Kapitel 2.2.1 die prinzipiellen Techniken der gängigen Virtualisierungssoftware. Der Markt beschränkt sich allerdings nicht nur auf Softwarelösungen. Auch Hardwarehersteller, allen voran die Hersteller von Prozessoren, haben erkannt, dass sich mit Virtualisierungslösungen hohe Gewinne erwirtschaften lassen. Da höhere Prozessortaktraten physikalisch zum heutigen Zeitpunkt kaum mehr möglich sind, ist die Integration von Befehlssätzen zur Virtualisierungsunterstützung in Hardware eine effiziente Möglichkeit neue Prozessoren zu vermarkten. Wie hardwaregestützte Virtualisierung technisch funktioniert und wie man sie nutzen kann, erklärt Abschnitt 2.3.

2.2.1 Technische Vorgehensweisen zur Virtualisierung

Trotz der großen Vielfalt an softwarebasierten Virtualisierungslösungen haben sich drei prinzipielle Techniken entwickelt, mit der mehr oder weniger alle Produkte arbeiten: Virtualisierung aller Hardwarekomponenten, Modifikation des Gast Betriebssystems und Paravirtualisierung. Die genannten Techniken werden im Folgenden eingehender beschrieben. Beachten muss man bei der Klassifizierung eines Produktes aber, dass die zugrunde liegende Technik oft nicht eindeutig einer Architektur zugeordnet werden kann sondern es sich um ein Mischform aus mehreren Ansätzen handelt. Dies liegt an den Vor- und Nachteilen der einzelnen Verfahren, aus denen jedes Produkt möglichst das Optimum erreichen möchte.

Hardwarevirtualisierung und Emulation

Die Virtualisierung einzelner Hardwarekomponenten, die anschließend zu einem vollständigen virtuellen Rechner zusammengesetzt werden können, ist die verbreitetste Technik auf dem Markt. Dabei werden alle benötigten Hardwarekomponenten einer Rechnerarchitektur in Software nachgebildet. Beispiele sind Festplatten, SCSI-Controller, Prozessoren, BIOS und Grafikkarten. Dabei unterscheidet man zwei Varianten der Nachbildung: Emulation und Virtualisierung. Während virtuelle Geräte versuchen, möglichst viele Befehle direkt an physikalische Komponenten durchzureichen, behandeln Emulatoren alle Aufrufe zunächst selbst. Dies hat zur Folge, dass Emulatoren eine deutlich geringere Performanz haben als virtuelle Komponenten, da auch sie die Befehle letztendlich an die Hardware weiterleiten müssen. Zuvor fällt aber eine zeitaufwendige Übersetzung der Befehle aus der Sprache des Emulators in die der Hardwarekomponente an. Diesem Nachteil in der Effizienz der Vorgehensweise steht jedoch ein Vorteil in der Vielfalt der unterstützten Architekturen gegenüber. Während virtualisierte Hardware nur auf der Architektur verwendet werden kann, auf der sie selbst aufsetzt,

bietet emulierte Hardware beispielsweise die Möglichkeit nach oben hin die Schnittstelle einer Macintosh-Architektur bereitzustellen, obwohl die zugrunde liegende Architektur die eines x86-Systems ist.
Lange Zeit ging man davon aus, dass die x86-Architektur nicht virtualisierbar ist. In der Tat wurde die Architektur nicht für diesen Zweck geschaffen. Dass es dennoch möglich ist, zeigte 1999 die Firma VMware mit ihrem Produkt VMware Workstation [VMW 05d]. Bei der Implementierung von VMware mussten einige Aspekte beachtet werden, mit denen auch alle übrigen Vorgehensweisen bei der Virtualisierung zurecht kommen müssen. Hauptproblem ist die Behandlung von Systemaufrufen in der virtuellen Maschine. Während ein Systemaufruf im Wirtbetriebssystem in den Kernelmode des Wirts schaltet und den Aufruf wie gewünscht behandelt, schaltet ein Systemaufruf im Gastbetriebssystem ohne zusätzliche Behandlung falscherweise ebenfalls in den Kernelmode des Wirts anstatt in den des Gastes. Systemaufrufe können nur im Kontext des Betriebssystems ablaufen. Versucht ein nicht privilegiertes Programm selbst einen Systemaufruf direkt zu tätigen, würde der Befehl vom Prozessor verweigert werden, da er sich im unprivilegierten „User Mode" statt im „Kernel Mode" befindet. Leider werden im Fall von virtuellen Maschinen nicht alle sensitiven Befehle korrekt abgefangen, da Befehle existieren, die je nach Modus in dem sie ausgeführt werden, andere Ergebnisse liefern. Dies hat zur Folge, dass ein beliebiges Programm, also auch das Betriebssystem der virtuellen Maschine, diese Befehle in dem Glauben ausführen kann, es befände sich im „Kernel Mode". In Wirklichkeit werden die Befehle jedoch im „User Mode" ausgeführt und liefern nicht das gewünschte Ergebnis. Wird der Prozessor des Gastbetriebssystems emuliert, ist es kein Problem diesen Fall entsprechend zu behandeln. VMware und viele andere Produkte benutzen aber aus Performanzgründen nur virtuelle Prozessoren. Aus diesem Grund müssen sensitive Befehle, das heißt Aufrufe, die den Zustand der Maschine verändern, vor ihrer Ausführung durch den virtuellen Prozessor abgefangen und entsprechend behandelt werden. Dies erreicht man durch ein Verfahren, das sich „Scan Before Execution" (SBE) oder kurz „Prescan" nennt. Dabei werden alle Prozessorinstruktionen vor ihrer Ausführung auf solche Befehle hin untersucht. Dieses Vorgehen ist nur zur Laufzeit möglich, da die x86-Architektur selbstmodifizierenden Code unterstützt und zum Beispiel Nutzereingaben nicht vorhergesehen werden können. Trifft der Scanner auf einen kritischen Befehl, substituiert er ihn durch „Int3" und setzt dadurch einen Softwarebreakpoint. Der Prozessor übergibt beim Erreichen dieser Stelle die Kontrolle an die Steuersoftware der virtuellen Maschine, wo der Aufruf behandelt wird.
Neben den Systemaufrufen muss auch die Verwaltung des Arbeitsspeichers angepasst werden. Im Wesentlichen bleibt das Pageingverfahren unverändert. Es muss lediglich verhindert werden, dass die virtuelle Maschine direkten Zugriff auf die realen Speicherbeschreiungstabellen erhält. Dies löst man durch die Einführung eines weiteren Adressraumes, auf den die virtuelle Maschine direkten Zugriff erhält. Die Übersetzung in reale Adressen erfolgt durch eine weitere Indirektionsschicht.
Grundsätzlich ist die Technik der Hardwarevirtualisierung eine sehr flexible Technik, mit der virtuelle Rechner durch das Hinzufügen oder Entfernen von Hardwarekomponenten an eigene Bedürfnisse angepasst werden können. Diese Flexibilität erkauft man sich aber durch deutliche Abstriche in der Performanz. Eine Möglichkeit Performanzverluste in Grenzen zu halten ist weitgehend auf den Einsatz von Emulatoren zu verzichten. Werden dennoch Emulatoren eingesetzt, dann meist nicht für alle Hardwarekomponenten, sondern nur für solche, die einen deutlichen Zugewinn an Flexibilität bringen. Ein Beispiel hierfür ist der bereits genannte Einsatz von Betriebssystemen auf fremden Hardwarearchitekturen.

Virtuelle Betriebssysteme

Virtuelle Betriebssysteme sind das Gegenstück zu Emulatoren. Nicht die Hardwarekomponenten werden virtuell erzeugt, sondern das Betriebssystem selbst wird virtualisiert. Für diese Technik ist es notwendig, dass die Quellen des Betriebssystems öffentlich verfügbar sind, denn sie müssen bei der Anpassung an ihren Verwendungszweck als virtuelles Betriebssystem modifiziert werden. Virtuelle Betriebssysteme beschränken sich daher im Wesentlichen auf das frei verfügbare Linux. Die auf diese Weise virtualisierte Variante von Linux nennt sich User Mode Linux. User Mode Linux [Dike 05], im Folgenden auch kurz UML genannt, wird von einer kleinen Gruppe von Entwicklern um Jeff Dike entwickelt. Das Projekt ist, da es direkt mit den Linux-Kernel-Sourcen arbeitet, frei unter der GPL (General Public License) erhältlich und wird auf Sourceforge gehostet. Ziel der Entwickler ist es nicht, mit Hardwareemulatoren, wie zum Beispiel Vmware zu konkurrieren. Sie wollen User Mode Linux als virtuelles Betriebssystem verstanden wissen. Ihr Ziel ist es, eine Art Sandbox für bestimmte Anwendungen, zum Beispiel Webserver, zu entwickeln. So könnte ein Provider mehrere von einander getrennte Webserver auf einem einzigen Server laufen lassen und seinen Kunden jeweils die Administrationsrechte für einen virtuellen Server gewähren. User Mode Linux eignet sich daher hervorragend für den

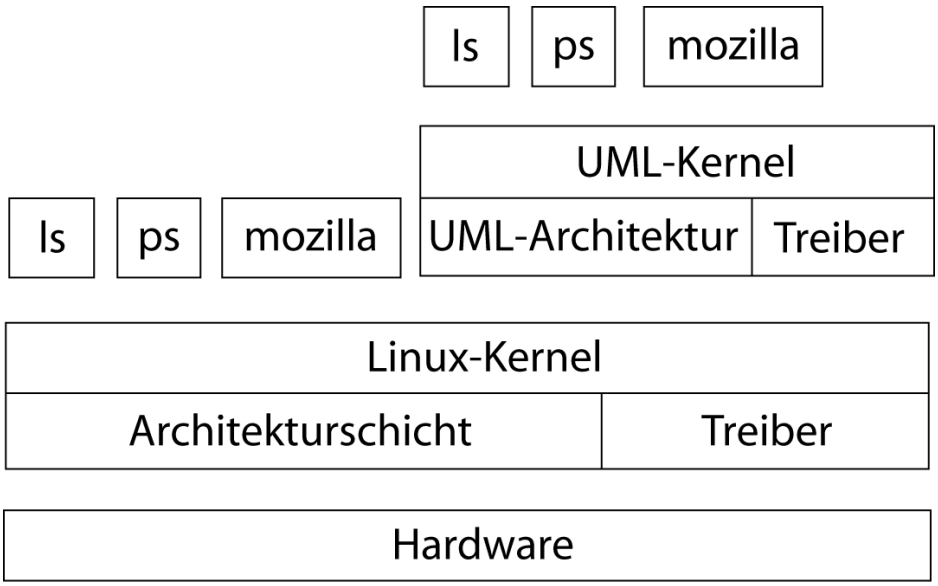

Abbildung 2.3: User Mode Linux Architektur

Zweck, viele virtuelle Instanzen von Betriebssystemen zu erzeugen, sofern diese keine allzu rechenintensiven Aufgaben zu bewältigen haben, da User Mode Linux selbst relativ geringe Rechenleistung zur Verwaltung der virtuellen Maschinen benötigt. Im Folgenden wird die Technik, die zur Virtualisierung eines Betriebssystems notwendig ist, kurz am Beispiel von User Mode Linux dargestellt.
Linux läuft bereits heute auf vielen unterschiedlichen Plattformen. Die Anpassung an eine neue Plattform ist vergleichsweise einfach zu implementieren, denn die dazu nötigen Änderungen betreffen nur einen sehr kleinen Teil der Quellen von Linux. Es muss lediglich die Architekturschicht neu implementiert werden. Genau diesen Weg geht User Mode Linux. Es implementiert die arch-Schnittstelle des Linux-Kernels neu und ist daher in der Lage auf einer virtuellen Plattform zu laufen. Dabei werden nicht einzelne Hardwarekomponenten in Software nachgebaut und zu einem lauffähigen Gesamtsystem zusammengesetzt, wie dies etwa VMware löst, sondern der User Mode Linux Kernel läuft als normales Programm im User Mode. Durch die Architekturschicht werden Systemaufrufe wie etwa Festplattenzugriffe abgefangen und entsprechend behandelt. Oberhalb der arch-Schnittstelle ist die Virtualisierung nicht mehr bemerkbar, d.h. es laufen wie in jedem „normalen" Linux ein Scheduler, Rechteverwaltung und diverse andere Prozesse. Einen groben Überblick über die Architektur gibt Abb. 2.3. Die Abbildung zeigt deutlich, dass der User Mode Linux Kernel auf der selben Hirachiestufe läuft wie normale Anwendungsprogramme. Als Beispiel sind hier `ls`, `ps` und `mozilla` aufgeführt. Erst oberhalb der eigentlichen Programmebene laufen die Prozesse des virtuellen Linux-Systems. Sie sind eigentlich für das zugrunde liegende Host-Linux nicht sichtbar. Eigentlich aber nur deshalb, weil zwar die Architektur der verschiedenen Hirachieschichten verhindern sollte, dass diese Programme unterhalb der User Mode Linux Architektur sichtbar sind, aber User Mode Linux für jeden virtuell gestarteten Prozess zur Ausführung einen Klon im Host-Linux erzeugt. Insofern sind die virtuellen Prozesse doch nicht komplett abgesondert. Daraus resultieren einige Performanz- und Sicherheitsprobleme, die sich aber mittels des SKAS-Patches beheben lassen. Seine Wirkung wird später in diesem Abschnitt behandelt.

Systemaufrufe Wie in allen Betriebssystemen treten auch bei virtuellen Betriebssystemen Systemaufrufe auf. Diese müssen entsprechend behandelt werden, da ein Systemaufruf, der von einem Prozess innerhalb des virtuellen Betriebssystems initiiert wird, nicht wie gewollt in den Kernel-Mode des virtuellen System schaltet, sondern in den Kernel-Mode des darunterliegenden Hostsystems. Der Systemaufruf würde also im Kontext des falschen Betriebssystems ausgeführt werden.
Das Problem löst man mit Hilfe eines Tracing-Threads, der nichts anderes zu tun hat, als auf Systemaufrufe der UML zu warten. Der Tracing-Thread wird nach einem Systemaufruf vom Kernel des Hosts über die eigentlich zu Debugging-Zwecken geschaffene ptrace-Schnittstelle informiert, sobald in den Kernel Mode geschalten

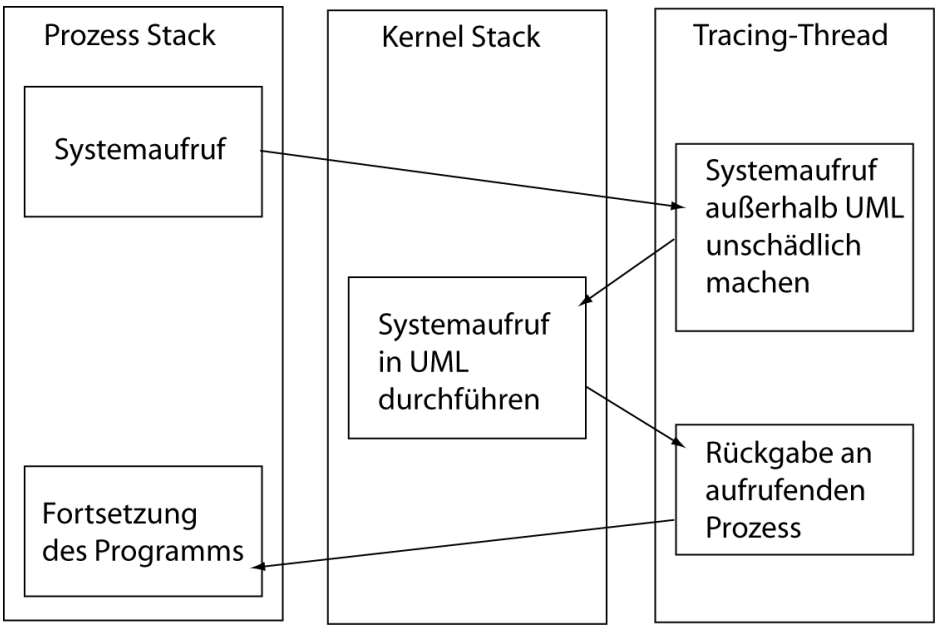

Abbildung 2.4: Systemaufrufe in User Mode Linux

werden soll. Darauf hin neutralisiert der Tracing-Thread den Systemaufruf mit Hilfe eines getpid-Aufrufes und behandelt den Systemaufruf im Kontext des UML-Kernel. Anschließend gibt er die Kontrolle über den Programmfluss zurück an die UML, wo der aufrufende Prozess weiter abgearbeitet wird. Der gesamte Ablauf wird schematisch in Abb. 2.4 dargestellt.
Beim Systemaufruf innerhalb eines User Mode Linux Prozesses geht die Kontrolle an den im Hostsystem laufenden Tracing-Thread, der den Systemaufruf im Host neutralisiert. Stattdessen übergibt er den Aufruf an den Kernel des virtuellen Systems, also User Mode Linux. Nach der Abarbeitung gelangt die Kontrolle des Programmflusses entgegengesetzt der ursprünglichen Aufrufreihenfolge über die Stacks zuerst wieder zum Tracing-Thread und anschließend in den aufrufenden Prozess, der weiter abgearbeitet wird.

Der SKAS-Patch Durch die bisher vorgestellte Architektur von User Mode Linux entsteht ein gravierendes Sicherheits-Problem. Es existieren keine von einander separierten Adressräume. So kann Prozess A ohne Einschränkung auf Daten von Prozess B zugreifen und diese sogar manipulieren; nicht einmal der Speicherbereich des Kernels ist vor solchen Angriffen geschützt. Dies zu lösen ist Ziel des SKAS-Patch (Seperate Kernel Address Space) [ska 05]. Mit ihm wird die Möglichkeit geschaffen, mehrere von einander getrennte Adressräume für einen einzigen Prozess zu erzeugen. Des Weiteren erweitert er die ptrace-Schnittstelle um einige zusätzliche Kommandos, so dass es Prozessen nun möglich ist, sich selbst zu unterbrechen. Durch diese beiden Design-Anpassungen verändert sich die Architektur des Systems nachhaltig: Es existiert nur noch ein einziger Prozess zur Ausführung von virtuellen Prozessen innerhalb der UML. Dieser Prozess kann sich nun selbst zu Scheduling-Zwecken per ptrace-Kommando unterbrechen, seinen aktiven Adressraum ändern und dadurch einen anderen UML-Prozess zur Ausführung bringen. Ähnlich verläuft auch ein Wechsel in den Kernel Mode von User Mode Linux. Als Resultat sind auch die im tt-Mode - so nennt sich der ungepatchte Modus von UML - erzeugten doppelten Prozesse nicht mehr notwendig. Für den Host sichtbar sind pro UML insgesamt nur noch vier Prozesse, wie dies in Abb. 2.5 erkennbar ist.
Ein positiver Seiteneffekt des SKAS-Patches ist die deutlich bessere Performanz der virtuellen . Durch die Möglichkeit der Prozesse sich selbst zu unterbrechen, benötigt man keinen Tracing-Thread mehr und kann sich aus diesem Grund zwei der vier Prozesswechsel bei einem Systemaufruf sparen. Statt bei einem Systemaufruf wie in Abb. 2.4 an den Tracing-Thread zu geben, erhält beim Einsatz des SKAS-Patches unmittelbar der User Mode Linux Kernel die Kontrolle.
Auf der Projekthomepage von User Mode Linux ist von einer Verdopplung der Performanz [Prat 04] bei der Kompilierung der Kernelquellen bei Verwendung des SKAS-Patches die Rede. In einigen Ausnahmefällen soll sogar eine Vervierfachung der Geschwindigkeit gemessen worden sein.

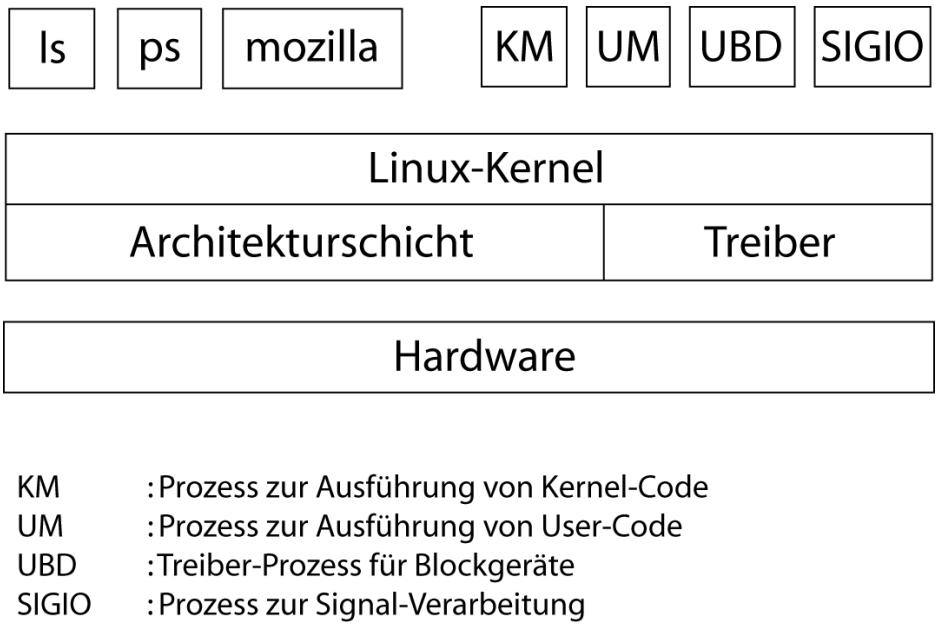

Abbildung 2.5: User Mode Linux Architektur mit SKAS-Patch

Zur Zeit liegt der SKAS-Patch in der Version 3 vor. Um ihn nutzen zu können muss er manuell in die Kernelsourcen des Hostsystems eingebunden und kompiliert werden. Eine feste Integration des Patches in den Kernel ist erst für die überarbeitete Version 4 vorgesehen. Eine manuelle Integration bedeutet derzeit zwar noch viel Arbeit, da der komplette Kernel neu übersetzt werden muss, bringt aber für das System Sicherheit und erhöht die Performanz. Negative Eigenschaften des SKAS-Patches wurden bisher nicht beobachtet.

Paravirtualisierung

Das Konzept der Paravirtualisierung ist die jüngste aller hier vorgestellten Technologien. Erstmals großflächigere Verbreitung in i386 Systemen erlangte sie in VMwares ESX-Server [VMW 05a]. Paravirtualisierung unterscheidet sich vom Ansatz der beiden übrigen Methoden dadurch, dass auf ein Wirtbetriebssystem verzichtet. Statt dessen kommt ein so genannter Hypervisor zum Einsatz, der die Rolle eines privilegierten Systems übernimmt. Er bietet nicht die Funktionalität eines Betriebssystems sondern ist erheblich schlanker und leistungsfähiger gebaut. Der Hypervisor, oft auch Virtual Machine Monitor (VMM) genannt, liegt als zusätzliche Schicht zwischen Hardwarearchitektur und Betriebssystem. Er selbst enthält keine Treiber zum Zugriff auf Hardware, diese werden weiterhin von den Betriebssystemen mitgebracht. Diese greifen jedoch nicht direkt auf Hardwarekomponenten zu, sondern über den VMM. Dieser bietet im Wesentlichen die selben Schnittstellen wie die Hardwarearchitektur selbst, allerdings angereichert um Virtualisierungsunterstützung. Dies hat zur Folge, dass bei manchen Implementierungen kleine Anpassungen am Betriebssystem notwendig sind, um auf der veränderten Architektur ausführbar zu sein. Neben der Zugriffskontrolle auf Hardware gehört es auch zu den Aufgaben des VMM Speicherverwaltung und Scheduling durchzuführen.
Paravirtualisierung ist derzeit die Effizienteste aller betrachteten Technologien. Zudem verbessert sich die Technik in diesem Bereich momentan am stärksten. Ein frei erhältlicher Vertreter dieser Technologie ist das Projekt Xen, welches später in diesem Kapitel noch eingehend behandelt wird.

2.2.2 Produkte

Aufgabe dieses Abschnitts ist es, die verfügbaren Virtualisierungsprodukte vorzustellen und ihre Vor- und Nachteile zu diskutieren, denn die Wahl der richtigen Virtualisierungssoftware hängt sehr stark von ihrem Einsatzzweck ab. Lösungen, die auf Hardwarevirtualisierung setzen, haben derzeit den größten Marktanteil. Zu ihnen gehören unter anderem die kommerziellen Produkte Vmware Workstation [VMW 05d] bzw. Vmware

GSX Server [VMW 05b] und MS Virtual PC bzw. MS Virtual Server [MSV 05]. Hinzu kommen die frei entwickelten oder wissenschaftlichen Projekte FAUmachine [FAU 05], QEMU [QEM 05] und Bochs [BOC 05]. Während die kommerziellen Lösungen vor allem Wert auf Benutzerfreundlichkeit und Performanz legen, steht bei den restlichen Projekten der Blickpunkt eher auf der Entwicklung neuer technischer Aspekte. Beispiele hierfür sind Automatisierung von Installationen und das Ausführen architekturfremder Betriebssysteme. Generell gilt: Je größer die Vielfalt der unterstützten Betriebssysteme, desto mehr leidet die Performanz eines Produktes und desto weniger virtuelle Instanzen virtueller Maschinen sind auf einem Referenzsystem möglich. Grundsätzlich ist diese Klasse der Virtualisierungslösungen, einschließlich der kommerziellen Lösungen, eher für den Einsatz einer niedrigeren Anzahl virtueller Maschinen geeignet. So wird das Testen und Entwickeln von Software auf verschieden Betriebssystemen und Plattformen mit virtuellen Maschinen wesentlich einfacher. Es wird lediglich ein realer Rechner benötigt um alle Tests durchzuführen, während zuvor für jede zu testende Konfiguration ein Rechner benötigt wurde.
Anders sieht die Situation bei den Produkten User Mode Linux [Dike 05], Vmware ESX Server [VMW 05c], Parallels [PAR 05b] und Xen [Univ 05a] aus. Alle Produkte, von denen nur User Mode Linux und Xen frei erhältlich sind, sind für den Einsatz vieler virtueller Maschinen konzipiert. Dabei verwendet User Mode Linux das Konzept eines virtuellen Betriebssystems, Xen und Vmware ESX Server setzen auf Paravirtualisierung. Parallels ist laut eigenen Angaben ein Hypervisor für den Einsatz am Desktop. Während User Mode Linux in seinem Einsatzzweck technisch auf Linux begrenzt ist, beeinflusst das Konzept der Paravirtualisierung die Wahl des Betriebssystems nicht. Tabelle 2.1 fasst die Unterschiede der einzelnen Produkte zusammen. Eine wichtige Eigenschaft der Tools ist auch die Möglichkeit sich in vorhandene Netze zu integrieren oder eigene schaffen. Dabei Arbeiten alle Produkte mit virtuellen Netzwerkkarten, die sich auf verschiedenste Varianten mit emulierten Switches oder Bridges verbinden lassen. Einige der Lösungen unterstützen sogar Hubs, indem beim Start der Softwareswitches bzw. Bridges ein spezieller Parameter gesetzt wird. Dieser Punkt erscheint nebensächlich, da Hubs längst ausgedient haben, in den Praktikumszenarien werden Hubs zum Aufbau der Netzwerke jedoch unbedingt benötigt.

2.2.3 Xen

Dieser Abschnitt befasst sich näher mit dem Hypervisor Xen. Xen ist aufgrund seiner Leistungsfähigkeit sehr interessant für das Vorhaben, das Praktikum IT-Sicherheit zu virtualisieren. Eine weitere in Frage kommende Virtualisierungslösung für das Projekt ist User Mode Linux. User Mode Linux wurde jedoch schon in einer vorausgehenden Arbeit [Lind 05] eingehend untersucht. Xen kam zum Zeitpunkt dieser Arbeit für das Vorhaben nicht in Betracht, da es sich noch im Entwicklungszustand befand. Mittlerweile ist Xen jedoch eine durchaus ernst zu nehmende Lösung für den Einsatzzweck. Aus diesem Grund wird Xen hier detaillierter behandelt, um anschließend das passende Werkzeug für die Virtualisierung wählen zu können.

Entstehung des Projektes Xen

Xen [Univ 05a] wurde ursprünglich an der Universität Cambridge entwickelt. Mittlerweile steht hinter der Entwicklung von Xen die Firma XenSource [XS 05], die das Projekt öffentlich und in Zusammenarbeit mit bekannten Firmen der Branche unter der Leitung von „XenMaster" Ian Pratt weiterentwickelt. Entstanden ist Xen nicht - wie man glauben möchte - als eigenständiges Projekt mit dem Ziel eine möglichst effiziente Virtualisierunsarchitektur zu schaffen, sondern es entstammt einem viel größeren Projekt namens Xenoserver [Univ 05b], das ebenfalls an der Universität Cambridge ins Leben gerufen wurde und mittlerweile sogar von den Deutsche Telekom Laboratories [DTL 05] unterstützt wird.
Die Idee, die hinter Xenoserver steht, ist das Erstellen einer öffentlichen Infrastruktur zur Verwendung in weit verteilten Systemen. Die Infrastruktur soll, wenn es nach den Plänen der Entwickler geht, den gesamten Globus umspannen und jedem zahlenden Menschen die Möglichkeit geben, auf ihr eigene Programme auszuführen. Durch die große Ausdehnung des Netzwerks soll sichergestellt werden, dass Benutzer ihre Programme möglichst nahe am eigenen Standort ausführen können und damit hohe Latenzzeiten und Flaschenhälse im Netzwerk möglichst vermieden werden. Um diese Lokalität ständig garantieren zu können, soll es zusätzlich ermöglicht werden, ganze personalisierte Dateisysteme und Dienste innerhalb von 45 Sekunden über den kompletten Globus auf eine andere Maschine migrieren zu können.

	Vmware Workstation Vmware GSX Server	MS Virtual PC MS Virtual Server	Parallels	FAUmachine	QEMU	Bochs	UML	Xen	Vmware ESX Server
Kommerziell	ja	ja	ja	nein	nein	nein	nein	nein	ja
Technik [a]	V	V	H	E	E	E	B	H	H
Performanz	+	+	+	+	-	-	+	++	+
Anzahl VMs [b]	o	o	o	o	-	-	+	++	+
Betriebssysteme [c]	Linux, Solaris, Netware, FreeBSD, DOS, Windows	Windows	Linux, FreeBSD, OS/2, eComStation, DOS, Windows	AROS, DOS, FreeBSD, NetBSD, OpenBSD, Linux, MenuetOS, OS/2, QNX, ReactOS, Syllable, Unununium, ZETA, Windows	BeOS, DOS, FreeDOS, NetBSD, QNX, SkyOS, OS/2, Linux, Minix, ReaktOS, Solaris, RTEMS, Etherboot, Windows,	Linux, Minix, OpenBSD, FreeBSD, GNU, SCO, DOS, FreeDOS Windows	Linux	Linux [d], BSD	Linux, Solaris Netware FreeBSD DOS, Windows
Netzarchitekturen [e]	Switch, Bridge	Switch Bridge	Switch Bridge	Switch Bridge Hub	Switch Bridge Hub	Switch Bridge Hub	Switch Bridge Hub	Switch Bridge Hub	Switch Bridge

Tabelle 2.1: Virtualisierungsprodukte und ihre Eigenschaften

[a] V=Hardwarevirtualisierung, E=Emulator, B=virtuelles Betriebssystem, H=Hypervisor
[b] auf einem Referenzsystem
[c] Herstellerangabe, nicht getestet
[d] ab Xen 3.0 und Vanderpool/Pacifica kann jedes für die x86-Architektur geeignete Betriebssystem verwendet werden
[e] Einige Kombinationen sind nur unter Linux möglich.

Dies kann logischerweise nur dann funktionieren, wenn die zugrunde liegende Serverhardware auf allen Maschinen identisch ist, da andernfalls zum Beispiel Probleme mit nicht vorhandenen Treibern oder fehlenden Ressourcen entstehen können. Da es kaum möglich erscheint überall auf der Welt über einen längeren Zeitraum identische Hardware bereitzustellen, kam man auf die Idee, die einzelnen Anfragen der Nutzergemeinde in virtuellen Maschinen abzuarbeiten. Ein zusätzlicher Vorteil dieser Architektur ist der Gewinn an Sicherheit, wenn Aufgaben in strikt getrennten Instanzen berechnet werden und sich nicht gegenseitig kompromittieren können. Leider existierten bis dato jedoch nur Virtualisierungslösungen, deren Performanz für das Projekt unzureichend erschienen. Dies war die Geburtsstunde des Kindprojektes Xen. Mit ihm sollte eine Möglichkeit geschaffen werden ohne allzu große Performanzverluste mehrere virtuelle Maschinen auf einem physikalischen Server auszuführen. Aufgrund der Architektur von Xen werden die einzelnen virtuellen Maschinen von einander isoliert und eine einfache Migration auf andere Server wird ermöglicht.
Xen arbeitet als Hypervisor; das heißt alle virtuellen Instanzen sehen bis auf wenige Ausnahmen die identische vom Hypervisor bereitgestellte Hardware. Somit ist es möglich Betriebssystemimages von einem Server auf einen anderen zu migrieren und innerhalb einer virtuellen Maschine - in der Xen Sprache „domU" genannt - zur Ausführung zu bringen. Mehr Informationen zur Architektur von Xen bietet der nächste Abschnitt.
Trotz der eigentlichen Aufgabe von Xen eine einheitliche Hardware-Schnittstelle bereitzustellen, ist es durch die unabhängig vom Xenoserver Projekt laufende Entwicklung des Hypervisors möglich, sehr an die eigenen Bedürfnisse angepasste domUs zu erstellen. Somit ist Xen auch eigenständig als ein sehr nützliches Projekt anzusehen. Wie sehr Virtualisierung den Zahn der Zeit trifft, zeigen auch die eifrigen Bemühungen der beiden großen Chiphersteller AMD und Intel Virtualisierungsunterstützung direkt in die Prozessoren zu integrieren. Genaueres liefert hierzu der Abschnitt 2.3.

Architektur von Xen

Xen wählt für seine Architektur eine Kombination aus Paravirtualisierung sowie hardwaregestützter Virtualisierung und Modifikation des Gastbetriebssystems [Prat 05] [Gers 05]. Der Schwerpunkt liegt jedoch klar auf der Seite der Paravirtualisierung. Es wird nur emuliert was unbedingt notwendig ist, gleichzeitig müssen kleinere Änderungen am Gastbetriebssystem vorgenommen werden, um das System an die veränderte Umgebung des Hypervisors anzupassen. Die Größenordnung der Anpassung von Betriebssystemen an Xen beläuft sich bei Linux auf ca. 3000 Zeilen Quelltext, bei Windows XP sind es rund 4600 für die allernötigsten Funktionen. Die Portierung von XP wurde nie vollendet, da sie von Microsoft Research lediglich zu Testzwecken vorgenommen wurde. Das angepasste Windows ist bis heute nicht öffentlich erhältlich, ob sich das in Zukunft ändern wird, darf bezweifelt werden, da Microsoft selbst umfangreiche Virtualisierungslösungen anbietet und diese natürlich auch vermarkten will. Durch prozessorunterstützte Virtualisierung wird die Portierung in Zukunft sowieso unnötig sein. Alle für die x86-Architektur geeigneten Betriebssysteme, also auch Windows, können mit ihrer Hilfe nativ auf Xen laufen.
Xen selbst ist ein Hypervisor und verwendet das Konzept der Paravirtualisierung. Das bedeutet, dass Xen selbst nur eine Schnittstelle, die der x86-Architektur sehr ähnlich ist, zur Verfügung stellt und die getätigten Aufrufe an die zugrunde liegende Hardware durchreicht. Auf der modifizierten x86-Schnittstelle - auch x86/Xen genannt - laufen die eigentlichen Betriebssysteme. Abbildung 2.6 verdeutlicht die Funktionsweise. Der Xen Virtual Maschine Monitor legt sich als zusätzliche Schicht über die zu abstrahierende Hardware. Über das so genannte „Safe Hardware Interface" sind Zugriffe auf Hardware möglich. Der VMM sorgt dafür, dass jede Komponente nur von einer Maschine zur selben Zeit verwendet wird. Im Falle des Prozessors - es werden auch Mehrprozessorsysteme unterstützt - sieht jede Maschine einen virtuellen Prozessor, der, sobald die Maschine aktiv wird, mit einem physischen Prozessor verbunden wird. Peripheriegeräte sind grundsätzlich einer Maschine fest zugeordnet. Oberhalb des VMM laufen alle virtuellen Maschinen, die in der Xen-Sprache als Domain bezeichnet. Alle Domains sind gleichberechtigt in ihrer Funktion. Lediglich eine Domain, „Domain0" oder kurz „dom0" genannt, übernimmt die privilegierte Aufgabe der Steuerung des Hypervisors und der andern Maschinen, „domU" genannt.
Zur Steuerung stellt Xen eine in Python geschriebene Verwaltungssoftware bereit, mit der es möglich ist, einzelne Domains neu zu starten, zu pausieren, aufzuwecken, zu zerstören oder neue Domains zu erstellen. Des Weiteren können spezielle Parameter des Hypervisors wie zum Beispiel die Art des Scheduling-Algorithmus gesetzt werden. Einen umfassenden Einblick in das Management der virtuellen Maschinen gibt das Kapitel 3.4.
Im Normalfall hat die Steuerdomain über den Hypervisor Zugriff auf sämtliche physikalische Hardwarekom-

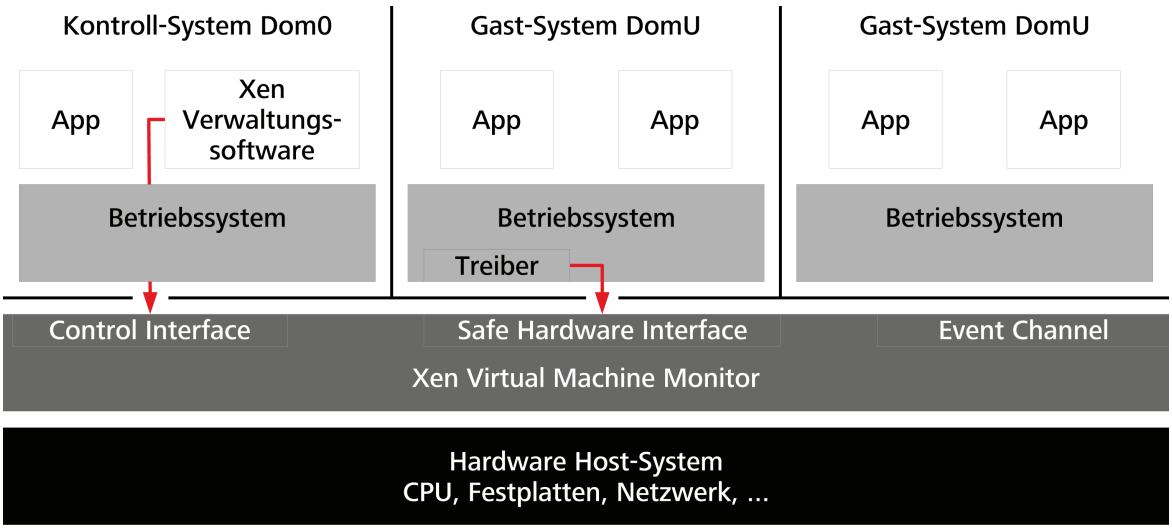

Abbildung 2.6: Die Architektur von Xen [Gers 05]

ponenten des Systems. Aus diesem Grund existieren für „Xenlinux", wie das an Xen angepasste Linux genannt wird, zwei verschiedene Kernelversionen. Der Kernel für die Steuerdomain dom0, der alle benötigten Treiber zum Zugriff auf physikalische Hardware besitzt, sowie ein reduzierter Kernel, der diese Funktionalität nicht bietet. Je kleiner der Kernel und je weniger Funktion und Fehlerquellen er enthält, desto stabiler und performanter läuft er. Soll ein unprivilegiertes System dennoch die Kontrolle über Hardware erhalten, lassen sich jederzeit neue Kernel erstellen, die die gewünschten Treiber enthalten. Eine andere Lösung, die SuSE Linux verwendet, besteht aus nur einem Kernel der bei Bedarf im Stande ist, sämtliche vorhandenen Treiber als Modul nachzuladen. Da nicht privilegierte Maschinen im Betrieb trotzdem auf bestimmte virtuelle Komponenten zugreifen müssen, enthalten sie sogenannte „Frontend Geräte Treiber" mit denen sie Geräte nutzen können, die ein oder mehrere „Backends" in anderen Systemen zur Verfügung stellen. Die genauere Funktionsweise von Frontend und Backend Zugriffen wird nachfolgend noch erläutert.
Um den Hypervisor vor unerlaubten Zugriffen der Betriebssysteme zu schützen bedient sich Xen des Ring-Konzeptes der x86-Architektur. Ringe stellen verschiedene Zugriffsebenen dar. Es existieren insgesamt vier Ringe von denen aber im Allgemeinen nur zwei genutzt werden. Ring eins, besser bekannt als „Kernel Mode", ist für das Betriebssystem reserviert und Ring 3, bekannt als „User Mode", steht Anwendungen zur Verfügung. Xen modifiziert diese Anordnung indem es den Hypervisor auf Ring 0 setzt und die Betriebssysteme auf Ring 1 verweist. Diese Änderung wird von Abbildung 2.7 dargestellt. Damit ist sichergestellt, dass der Hypervisor die höchste Priorität besitzt. Die Betriebssysteme selbst dürfen daher nicht mehr alle Befehle selbst auf dem Prozessor ausführen. Versuchen sie es dennoch, wird die Ausführung des Befehls durch den Prozessor verweigert und ein spezieller Exceptionhandler bearbeitet den Systemaufruf, welcher auch Hypercall genannt wird.
Ein weiterer wichtiger Aspekt ist die Verwaltung und Trennung der verschiedenen Adressräume. Der Aufbau ist bei Xen folgendermaßen organisiert: Im obersten Bereich bei 4 GB liegt Xen selbst, darunter die Betriebssysteme. Der Bereich zwischen 0 und 3 GB ist für Anwendungen reserviert. Abbildung 2.7 verdeutlicht die Lage der verschiedenen Adressräume grafisch. Auf Speicherbereiche der Anwendungen kann von allen Hierarchien zugegriffen werden, auf den Speicher der Betriebssysteme haben lediglich Hypervisor und Betriebssystem von den Ringen 0 und 1 Zugriff. Der Bereich des Hypervisors Xen ist nur im Ring 0 verfügbar.
Leider funktioniert die Trennung der einzelnen Hierarchien auf diese Weise nur bei 32Bit-Architekturen. Bei 64Bit-Architekturen versagt sie, da es dort nicht möglich ist obere Grenzen für Speichersegmentierung anzugeben. Hier werden der Hypervisor auf Ring 0 und Betriebssystem sowie Anwendungen auf Ring 3 ausgeführt. Um unerlaubte Speicherzugriffe zu unterbinden, wurde eine Kontrolle in das Pageingverfahren implementiert. Es existieren zwei unterschiedliche Pagetables: eine mit Usermodeseiten, die andere mit Kernel- und Usermodeseiten. Nur bei Systemaufrufen verwendet der VMM zweitere und verhindert somit Manipulation an den Kerneln.
Ein anderer interessanter und auch sehr wichtiger Aspekt ist die Wahl des Schedulingalgorithmus. Bei normalen Schedulingalgorithmen kommt es lediglich auf Fairness und Leistung an. Will man jedoch zeitkritische Anwendungen bedienen, wird es schwieriger. Genau mit diesem Problem ist Xen konfrontiert, denn das Ein-

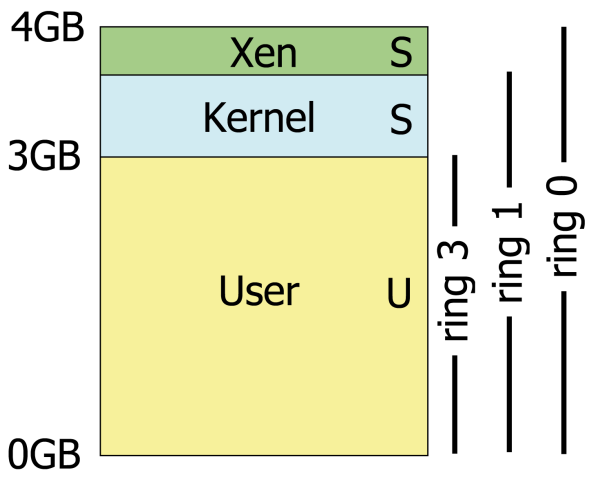

Abbildung 2.7: Speicherverteilung in x86 Systemen bei 32 Bit [Gers 05]

halten von TCP-Timingvorschriften kann im weiteren Sinne als solche gesehen werden. Wird eine Maschine nach dem Erhalt eines TCP-Paketes suspendiert und bis zum Ablauf eines Timeouts nicht reaktiviert, wird die Verbindung beendet. Aus diesem Grund wird in Xen als Schedulingalgorithmus für die Gastsysteme der von Kenneth J. Duda und David R. Cheriton 1999 entwickelte Borrowed Virtual Time (BVT) Scheduler verwendet [KJD 05] [Gers 05]. Dazu stellt Xen für jede virtuelle Maschine drei Zeiten zur Verfügung. Die *real time*, sie repräsentiert die Anzahl der vergangenen Nanosekunden seit dem Start der VM, die *virtual time*, welche nur verstreicht, wenn die Maschine aktiv ist, sowie die *wall-clock time*, ein Offset zur *real time*, mit der die aktuelle Uhrzeit berechnet werden kann ohne die *virtual time* zu beeinflussen. Grundlage für den Algorithmus ist das Zeitscheibenverfahren Round Robin und die bisher benötigte Rechenzeit eines Prozesses. Zusätzlich erhält jedes System einen Bonus, der zur Behandlung wichtiger Interrupts eingelöst werden kann. Damit ist das Verfahren zwar kurzzeitig ungerecht, langfristig jedoch ausgeglichen.
Hier ein Beispiel: Drei virtuelle Maschinen werden ausgeführt. Die Dauer einer Zeitscheibe betrage 5 Zeiteinheiten. Zum Zeitpunkt t=6 trete bei VM1 ein Interrupt auf. Dieser kann durch den eingeräumten Bonus von 7 Zeiteinheiten bereits nach Suspendierung von VM2 zum Zeitpunkt t=10 behandelt werden. Abbildung 2.8 illustriert den Ablauf. Bei Round Robin werden die rechenbereiten Prozesse streng nacheinander ausgeführt. Im Gegensatz dazu erhält VM1 mit dem Eintreffen eines Interrupts zum Zeitpunkt 6 einen Bonus von 7 Zeiteinheiten. Daher kann VM1 bereits nach Ausführung von VM2 zum Zeitpunkt 10 die Unterbrechung behandeln, da seine *virtual time* jetzt niedriger ist als die aller anderen Prozesse. Nach der Abarbeitung wird der Bonus wieder entfernt.
Das Verfahren ist laut der Xenentwickler so leistungsstark in der Performanz-Isolation, dass selbst leistungshungrige Maschinen oder DoS-Attacken auf einzelne Maschinen die Arbeit der restlichen virtuellen Maschinen nur minimal beeinflussen [Gers 05].

Versionsunterschiede

Zu Beginn dieser Arbeit stand als einzig stabile Xen-Version das Release 2.0.7 zur Verfügung, da sich die Version 3.0 noch im Entwicklungsstatus befand. Die Versuche mit Xen beruhen daher alle auf dieser Version von Xen. Die Ergebnisse sollten jedoch auch für die in der Entwicklung befindliche Version 3.0 von Xen gültig sein. Trotzdem soll hier schon im Voraus ein Überblick über die wesentlichen Unterschiede der beiden Versionen gegeben werden. Im Allgemeinen handelt es sich hierbei um die Integration von zusätzlichen Eigenschaften in die Version 3.0
Zeitgleich mit der Entwicklung der Version 3.0 von Xen arbeitete man an einer Portierung der Xen Architektur von x86-32 auf x86-64 mit dem Ziel, diese in 3.0 zu integrieren. Mit der Unterstützung von 64Bit-Architekturen wird die Möglichkeit geschaffen, durch den Hypervisor mehr als 4 GB Arbeitsspeicher zu adressieren, was bei 32Bit-Systemen ohne spezielle Speichererweiterungsstrategien das Maximum darstellt. Nunmehr sind theoretisch 16 EB (ExaByte) möglich, von denen Xen jedoch nur 1 TB verwenden kann, was

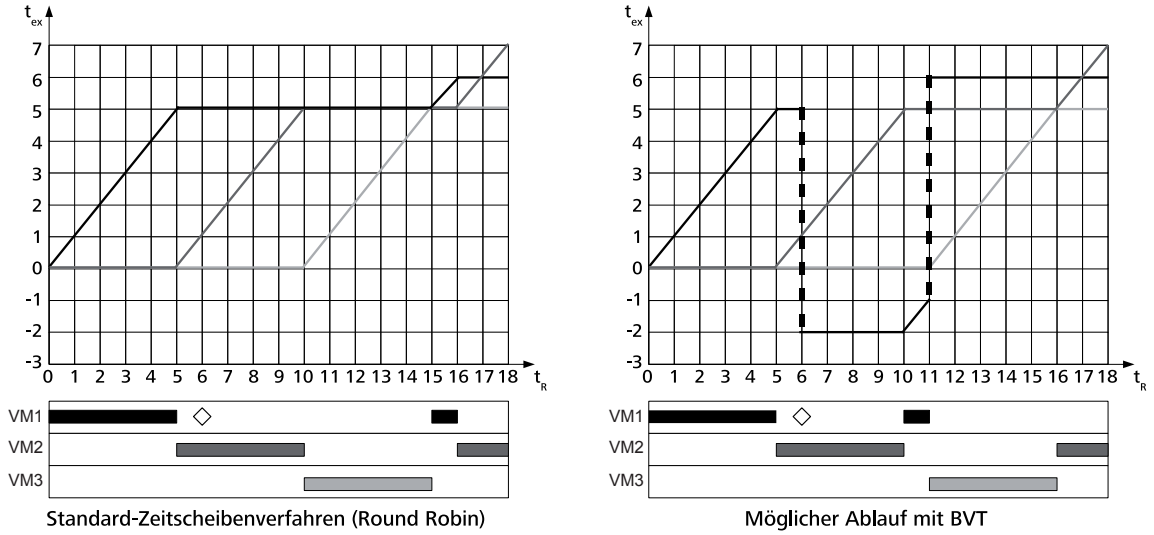

Abbildung 2.8: Vergleich der Schedulingverfahren Round Robin und BVT [Gers 05]

derzeit selbst für die Virtualisierung von sehr anspruchsvollen Topologien ausreichend ist. Des Weiteren wird der Hypervisor in zukünftigen Versionen automatisch Loadbalancing betreiben, indem er virtuelle Maschinen bei Bedarf auf andere Prozessoren migriert, um ausgewogene Last zu erreichen.
Ein weiterer sehr interessanter Ansatz ist das Cluster Storage System Parallax [PAR 05a]. Es soll die Möglichkeit bieten mit Copy on Write (CoW) Dateisystemen umzugehen sowie Snapshots zu erstellen. Als Snapshots bezeichnet man das Festschreiben eines Zustandes des Dateisystems zu dem jederzeit zurückgekehrt werden kann. Dadurch können Änderungen, die unabsichtlich oder nur zu Testzwecken vorgenommen wurden sehr einfach rückgängig gemacht werden. Mit der Copy on Write Technik ist es möglich, ein Dateisystem mehreren Instanzen schreibend zur Verfügung zu stellen. Dabei wird für jede Instanz eine separate Datei erstellt, in der lediglich die Änderungen gegenüber dem original Dateisystem abgespeichert werden. Beide Technologien haben zum Ziel, Speicherplatz zu sparen. Bei Parallax sollen auch rekursiv Snapshots von CoWs und CoWs von Snapshots möglich sein. Damit ist das Tool dem Linux eigenen Logical Volume Manager (LVM), der lediglich „Snapshots" von Dateisystemen erstellen kann und damit eigentlich CoW meint, deutlich überlegen. Parallax wird in der Version 3.0 von Xen enthalten sein. Allerdings sind noch nicht alle Features des Projekts komplett implementiert, einfache lokale Anwendungen ohne Migration von Domains auf andere Server funktionieren aber bereits.
Die interessanteste Neuerung von Xen 3.0 ist die Unterstützung von Befehlssätzen zur prozessorunterstützten Virtualisierung. Durch sie ist es möglich einige Aufgaben, die früher der Hypervisor übernehmen musste, direkt auf der Hardware des Prozessors auszuführen. Hierdurch erreicht man zum einen Geschwindigkeitsvorteile in der Ausführung, zum anderen entfällt das Anpassen der Gastbetriebssysteme an Xen. Es können also closed Source Betriebssysteme wie zum Beispiel Windows XP auf Xen ausgeführt werden. Volle Unterstützung für hardwaregestützte Virtualisierung wird es jedoch erst in Xen 3.0.1 geben. Zusätzlich benötigt man einen Prozessor, der die entsprechenden Befehlserweiterungen unterstützt. Intel nennt diese Technologie Vanderpool (VT), AMD die seine Pacifica. Abschnitt 2.3 geht detaillierter auf hardwaregestützte Virtualisierung ein.

Hardware

Spricht man im Zusammenhang von Virtualisierung über Hardwarekomponenten, so muss man zwei grundsätzliche Arten von Hardware unterscheiden. Physikalische und virtuelle Komponenten. Während erstere wirklich existieren, werden zweitere lediglich emuliert bzw. virtualisiert. Xen verwendete zwei verschiedene Treiber-

Arten um auf sie zuzugreifen. Native Treiber für physikalische Geräte und Frontend Treiber für virtuelle. Physikalische Geräte sind immer einer virtuellen Maschine fest zugeordnet. Alle Geräte, die der Maschine dom0 beim Start nicht explizit über Bootparameter entzogen werden, sind dieser zugeordnet. Alle anderen Geräte können beim Start von virtuellen Maschinen an diese gebunden werden. Eine Domain kann, sofern sie selbst geeignete Hardware besitzt, als Backend für andere Maschinen arbeiten, das heißt diese Komponenten anderen Systemen zur Verfügung stellen. Es existieren zwei Arten unterschiedlicher Backends: Netz-Backends, die Anschluss an virtuelle Bridges bieten und Blockgeräte-Backends, die Speicherplatz bereitstellen. Andere Komponenten können nur gemeinsam verwendet werden, sofern Linux dies unterstützt. Ein Beispiel hierfür ist die Verwendung eines Druckers über CUPS (Common Unix Printing System) oder das Verwenden eines fremden Modems durch Routing. Die unterschiedlichen Hardwarearten werden im Folgenden genauer betrachtet.

Physikalische Komponenten

Der Zugriff auf physikalische Geräte erfolgt über die Linux eigenen Treiber. Diese können entweder fest in den Kernel einkompiliert sein oder als ladbare Module zur Verfügung stehen. Da normalerweise alle physikalischen Geräte der Domain dom0 zugeordnet werden, sind diese auch nur in deren Kernel enthalten. Die Kernel der nicht privilegierten Systeme enthalten diese nativen Treiber nicht. Es lassen sich aber in beide Kernelversionen benötigte native Linux-Treiber einbinden. Dazu benötigt man lediglich die Quellen des Linux-Kernel mit dem passenden Xen-Patch. Beides ist in der Quelltextversion von Xen enthalten. Als erstes konfiguriert man den gewünschten Kernel, zum Beispiel mit

```
make menuconfig ARCH=xen,
```

anschließend wählt man die benötigten Treiber aus, speichert die Konfiguration und übersetzt und installiert den Kernel mittels

```
make ARCH=xen && make install ARCH=xen.
```

Jetzt kann der neue Kernel verwendet werden. Gegebenenfalls müssen zuvor noch die erzeugten Module in das Dateisystem der virtuellen Maschine kopiert werden. Der Zugriff einer Maschine auf essentielle Hardware ist exklusiv. Das heißt, dass diese Hardwarekomponenten immer fest einer Maschine zugeordnet sind, andere Maschinen haben auf eine vergebene Komponente keinen Zugriff. Damit dies sichergestellt ist, können die virtuellen Maschinen nicht direkt auf die Hardwareschicht zugreifen, sondern müssen dies über das „Safe Hardware Interface" von Xen erledigen. Dies zeigt auch Abbildung 2.9.
Jede Domain, die direkten Zugang zu Speichermedien hat, kann einzelne Dateien oder Partitionen anderen Maschinen als Blockgerät, einer Art virtuelle Festplatte, zur Verfügung stellen. Das funktioniert aber nur mit physikalischen Geräten, das Exportieren von Blockgeräten aus Blockgeräten ist nicht möglich. Der Sinn dieser Einschränkung ist klar ersichtlich: Die Wege, bis ein Aufruf seine Hardware erreicht, sollen möglichst kurz sein, möglichst wenig Mehraufwand verursachen und auf diese Weise ein performantes System gewährleisten.

Virtuelle Komponenten

Wie bereits erwähnt, existieren zwei Arten an virtuellen, peripheren Hardwarekomponenten: Blockgeräte und Netzwerkkarten.
Im Folgenden werden Maschinen, die Blockgeräte oder Netzwerkkarten exportieren, also anderen Maschinen bereitstellen, als Server bezeichnet. Entsprechend heißen Maschinen, die diese Dienste in Anspruch nehmen Clients.
Blockgeräte sind Teile eines physikalischen Massenspeichers, die anderen Maschinen als virtuelle Festplatten zugänglich gemacht werden können. Auf ihnen liegt zum Beispiel die Root-Partition eines Linux-Dateisystems. Als Blockgeräte eignen sich Partitionen und Image-Dateien, wobei letztere auch über den Linux eigenen LVM (Logical Volume Manager) bereitgestellt werden können. Für die virtuelle Maschine ist es nicht erkennbar, welche Art Massenspeicher sich hinter der virtuellen Festplatte verbirgt. Der Zugriff auf die Festplatte erfolgt

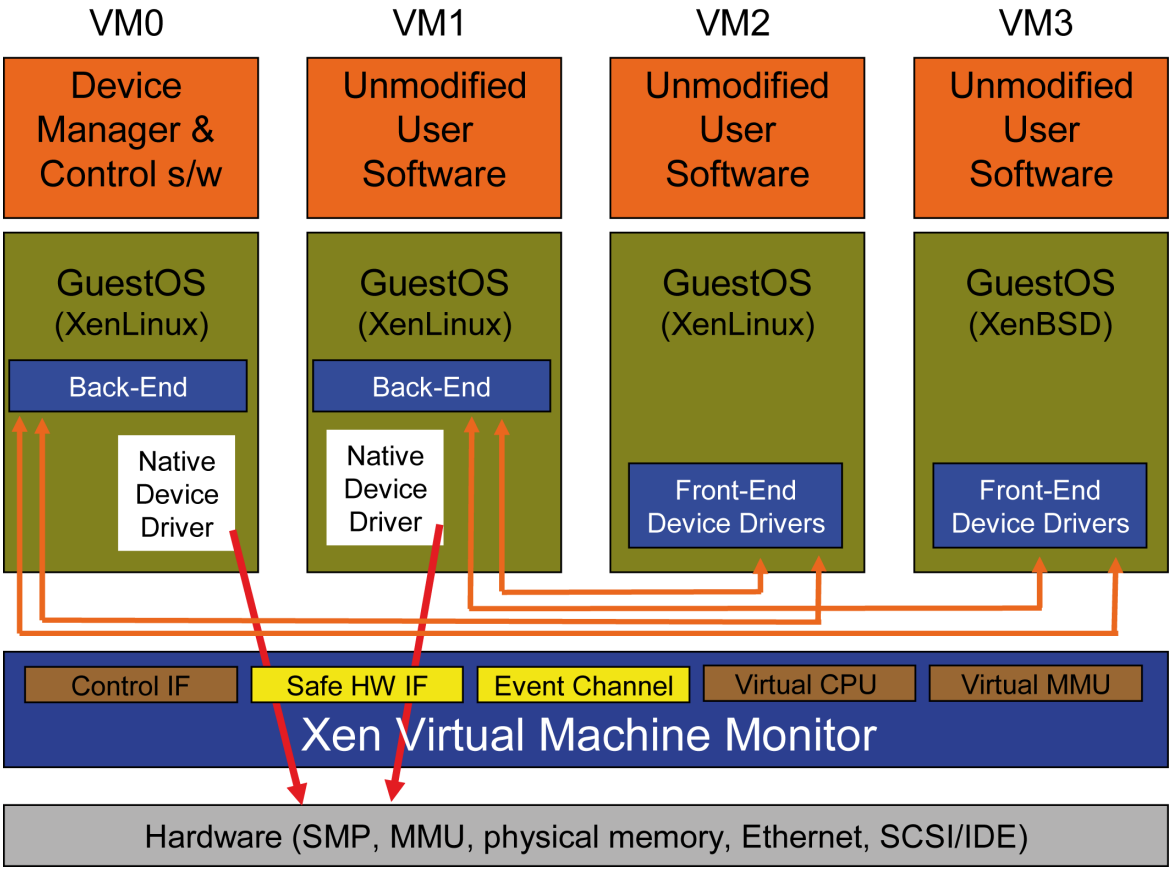

Abbildung 2.9: Front- und Backend Zugriffe unter Xen [Prat 05]

über Frontend Treiber des Clients. Der Frontend Treiber greift jedoch nicht selbst auf den physikalischen Datenträger zu, sondern kontaktiert lediglich das verbundene Backend. Als Backend wird die Dienstschnittstelle des Server bezeichnet, über die die Hardware zur Verfügung gestellt wird. Der eigentliche Hardwarezugriff wird anschließend vom Kernel des Servers ausgeführt. Dazu benötigt diese entsprechende native Treiber. Abbildung 2.9 stellt die Zugriffe zwischen Front- und Backend grafisch dar.
Eine Alternative zu Blockgeräten sind netzwerkbasierte Dateisysteme via NFS (Network File System). Sie sind zwar im Normalfall etwas langsamer im Zugriff als Blockgeräte, vereinfachen dafür aber die Migration einzelner Domains auf andere Server. Der Grund hierfür liegt in der Architektur des Internet Protokolls (IP). Während ein NFS-Server mit öffentlicher IP-Adresse von jedem Punkt im Netzwerk erreichbar ist, ist das für die proprietäre Schnittstelle des Backend Dienstes nicht zwangsläufig der Fall. Sie kann eventuell nur auf dem lokalen System verfügbar sein. Beim Verwenden eines NFS-Servers steht das Dateisystem bei Migrationen damit zu jedem Zeitpunkt an jedem Ort im Netzwerk bereit, während lokale Dateisysteme auf Blockgeräten erst kopiert werden müssten.

Die zweite Art virtueller Komponenten sind Netzwerkkarten. Jede virtuelle Netzwerkkarte in einer virtuellen Maschine ist logisch gesehen über ein Crossoverkabel mit einer zweiten virtuellen Netzwerkkarte in einem Backend verbunden. Die Netzwerkkarten im Backend sind jedoch nichts anderes als die Ports einer Software-Bridge. Als Bridge Implementation kommt das Standard Linux Kernel Bridging zum Einsatz. Dabei wird eine Bridge emuliert, der die virtuellen Netzwerkkarten im Backend-System als Ports hinzugefügt wurden. Die Standardkonfiguration von Xen erzeugt beim Start der Kontrollsoftware automatisch eine Bridge namens `xen-br0`. Für jede gestartete Maschine mit virtueller Netzwerkkarte wird dieser Bridge eine zusätzlich erzeugte Netzwerkkarte hinzugefügt. Zwischen den einzelnen virtuellen Maschinen arbeitet die Softwarebridge wie eine reale Bridge. Sie leitet Ethernetframes von einem Port exakt an den Port weiter, mit dem das Zielsystem verbunden ist. Das Verhalten der Standardkonfiguration lässt sich einfach den persönlichen Bedürfnissen anpassen. Es können nach Belieben mehrere unabhängige Bridges erstellt werden, auch die Zuordnung der Netzwerkkarten bei virtuellen Maschinen mit mehreren Netzwerkkarten auf bestimmte Bridges lässt sich anpassen. Um einen Hub zu realisieren, kann man den Parameter ”setageing” der Bridge auf Null setzten. Der Parameter bestimmt die Lebensdauer des internen ARP-Caches. Der Wert Null sorgt dafür, dass die Bridge alle Einträge im ARP-Cache sofort wieder verwirft. Damit ist nicht bekannt welcher Port welcher virtuellen Maschine zugeordnet ist. Als Konsequenz wird bei jedem Versuch ein Datenpaket an eine virtuelle Maschine weiterzuleiten das Paket auf allen Ports ausgegeben um die Nachricht erfolgreich absetzen zu können. Das Weiterleiten an alle Ports wird normalerweise nur im Lernmodus der Bridge durchgeführt. Schon nach der ersten Antwort des Zielsystems ist der ihm zugeordnete Port bekannt und wird im Cache gespeichert. Die Kommunikation mit diesem System läuft anschließend nur noch über diesen Port ab. Da sich der Cache bei dem gesetzten Parameterwert jedoch niemals füllt, bleibt die Bridge ständig im Lernmodus und arbeitet de facto als Hub.
Durch den Einsatz von `iptables` und `ebtables` auf dem Backendsystem lassen sich recht einfach und effektiv Paketfilter-Firewalls sowie OSI Level zwei basierte Firewalls erstellen. Fügt man der Bridge eine IP-Adresse hinzu, wird diese vom Linux-Kernel zusätzlich als normales Interface behandelt, auf dem Pakete das System betreten können. Bei aktiviertem Routing werden Pakete von anderen Netzwerkkarten über die Bridge weitergeleitet und umgekehrt. Das interne Verhalten unterscheidet sich nicht von dem bei normalen Netzwerkkarten. Auf diese Weise greifen auch die Filtertabellen von `iptables`. Diese Architektur macht selbst das Erstellen von komplexen Netzwerken möglich. Um ein so entstandenes virtuelles Netz ohne den Einsatz von Routing an ein physikalisches Netz anzubinden, ist es möglich auch physikalische Netzwerkkarten an eine Bridge zu binden. Damit werden alle anderen Clients der Bridge im physikalischen Netzwerk direkt erreichbar. Für Unbeteiligte ist sogar nicht einmal ersichtlich, dass es sich bei den Geräten um virtuelle Maschinen handelt.

2.3 Integration von Virtualisierungsunterstützung in Prozessoren

Ein wesentlicher Nachteil der Virtualisierung durch Softwarelösungen ist der Verlust an Performanz, bedingt durch den ständigen Kontextwechsel zwischen VMM und virtueller Maschine bei Systemaufrufen. Des Wei-

teren bleibt ein ungutes Gefühl, wenn Betriebssystem und Anwendungen mit den selben Privilegien im Ring 3 ausgeführt werden. Zwar kontrolliert der VMM sämtliche Zugriffe auf Speicher und Prozessor, doch wer garantiert, dass bei der Implementierung des Hypervisors an wirklich alle Ausnahmen gedacht wurde? Zusätzlich entsteht das Problem, dass Betriebssysteme darauf konzipiert sind, im Ring 0 zu laufen. Sollen sie wie bei Xen im Ring 1 oder 3 laufen, muss eine Anpassung erfolgen. Dies ist, wie bereits ausgeführt, mit Closed Source Betriebssystemen nicht oder nur schwer möglich. Eine Integration ausgewählter Funktionen des Hypervisor in den Prozessor kann diese Probleme elegant lösen. Sowohl Intel als auch AMD, die beiden größten Hersteller von x86-kompatiblen Prozessoren, haben entsprechende Produkte im Angebot bzw. in der Entwicklung. Anfang 2005 präsentierte Intel erstmals Pläne Virtualisierungsunterstützung nach Vorbild der Mainframes direkt in Prozessoren zu integrieren. Die Entwicklung fand von Anfang an in enger Zusammenarbeit mit den Marktführern der Virtualisierungsbranche statt. Zur Gruppe gehörten neben den beiden Chipherstellern Intel und AMD auch die Softwarehäuser VMware, XenSource und Microsoft. Mit diesem Zusammenschluss sollte eine optimale und kompatible Lösung der einzelnen Produkte erzielt werden. Intel nennt seine Technologie Vanderpool (VT) [INTE 05], AMD die seine Pacifica [AMD 05]. Was die beiden konkurrierenden Techniken bieten und worin sie sich unterscheiden, zeigt dieser Abschnitt.

2.3.1 Intel Vanderpool (VT)

Die Virtualisierungsunterstützung im Prozessor wird durch einen Befehlssatz bereitgestellt, den Intel VMX (Virtual Maschine Extension) taufte. Bei VMX existieren zwei Modi: VMX root und VMX nonroot. Ersterer Modus ist für die Ausführung des Hypervisors gedacht. Alle virtuellen Maschinen laufen im Modus VMX nonroot. Das Verhalten des Prozessors im Modus VMX root ist im Wesentlichen identisch mit dem herkömmlicher Prozessoren ergänzt mit einigen speziellen Befehlen, die zur Virtualisierung notwendig sind. Der Modus VMX nonroot hingegen entspricht einer limitierten Prozessorumgebung, in der das Ausführen bestimmter Befehle nicht möglich ist. Um den VMX-Befehlssatz zu aktivieren, muss der VMM zunächst das Bit 13 im CR4, auch CR4.VMXE genannt, setzen. Anschließend startet das Kommando VMXON die Befehlserweiterung VMX. VMX unterstützt nur den Page-protected Mode zur Speicherverwaltung, die Modi Unpaged-protected Mode sowie der Real Address Mode werden in Kombination mit VMX nicht unterstützt. Enthalten die Register CR0.PE und CR0.PG dennoch entsprechende Werte, wird VMX nicht aktiv. Nach dem Start befindet sich der Prozessor immer im Modus VMX root, so dass ein VMM, der VMX aktiviert hat die Kontrolle über das gesamte System erlangt. Zur Ausführung von virtuellen Maschinen wird in den Modus VMX nonroot geschaltet. Hierfür steht die VMX transition VMentry zur Verfügung. Der Befehl VMexit aktiviert den Modus VMX root wieder und gibt die Kontrolle zurück an den Hypervisor. Auf diese Weise hat der Hypervisor volle Kontrolle über das System, während die virtuellen Maschinen in ihren Aktionen begrenzt sind, obwohl deren Betriebssystem im Ring 0 läuft, der bei den Prozessoren auch Current Privilege Level (CPL) 0 heißt. Damit entfällt die Anpassung des Betriebssystems an andere CPLs. Da es auch kein softwarelesbares Register gibt, das den Modus VMX nonroot verraten würde, ist die Virtualisierung für das Betriebssystem transparent.
Versucht ein Betriebssystem einen privilegierten Befehl auszuführen, erfolgt seitens des Prozessors ein VMexit, so dass der VMM den Aufruf ordnungsgemäß behandeln kann. Dabei wird eine Art Prozesswechsel durchgeführt, der Ähnlichkeit mit dem des Schedulings hat. Dies funktioniert nur ordentlich, wenn sich der Prozessor beim nächsten Aktivieren eines Prozess im selben Zustand befindet, wie vor dem Wechsel. Aus diesem Grund wurde der Virtual Maschine Control State (VMCS) eingeführt. Er besitzt ähnliche Aufgaben wie das Programm Status Wort (PSW) beim Scheduling. VMCS stellt eine Kontrollstruktur bereit, in der zum Beispiel der Status eines Prozessors abgelegt werden kann. Das VMCS ist ein 4KB großer Block im Hauptspeicher des Rechners, der über den VMCS-Pointer des physischen Prozessors referenziert wird. Pro virtuellem Prozessor kann ein VMCS angelegt werden, das mittels VMPTRLD gefolgt von der Adresse des VMCS aktiv wird. VMPTRLD steht für Load Pointer to Virtual Maschine Control Structure und setzt den VMCS-Pointer eines physischen Prozessors. Entsprechend speichert VMPTRST den VMCS-Pointer und VMCLEAR deaktiviert das VMCS. Das VMCS besteht aus drei Bereichen:

- revision identifier, gibt die Version der VMCS-Struktur an.
- abort indicator, speichert einen Abbruch-Wert
- data, enthält den Prozessorstatus und lässt sich weiter unterteilen in:
  - Guest-state area

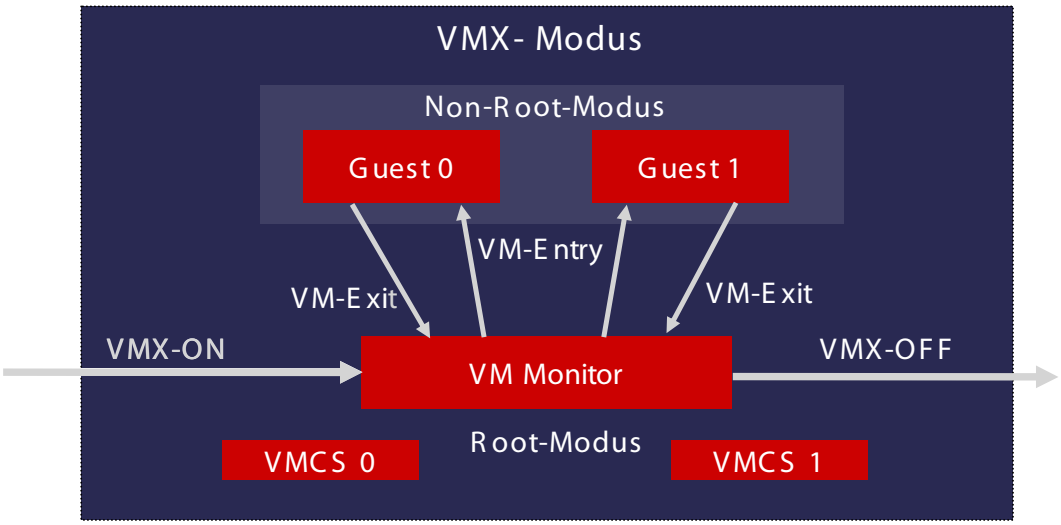

Abbildung 2.10: Vereinfachter Kontextwechsel zwischen VMX root und VMX nonroot bei Intel Vanderpool [Kers 05]

- Host-state area
- VM-execution control fields
- VM-exit control fields
- VM-entry control fields
- VM-exit information fields

Beispielsweise enthält die Guest-state area unter anderem die gesicherten Prozessorregister und VM-exit information fields den Grund der VMX transition. Die genauen Bedeutungen der einzelnen Felder können in der Spezifikation von Vanderpool [INTE 05] nachgelesen werden.
Um einen VMCS zu verwenden, benutzt man das Kommando VMLAUNCH. Er startet eine virtuelle Maschine unter der Kontrolle von VMCS. Um eine Maschine nach einer VMX transition erneut auszuführen, dient das schnellere Kommando VMRESUME. Bei der Verwendung ist darauf zu achten, dass ein VMCS nie von mehreren Prozessoren gleichzeitig verwendet wird. Aus diesem Grund muss bei der Migration von Maschinen auf andere Prozessoren, um zum Beispiel Loadbalancing durchzuführen, der erste Prozessor vor der Migration den VMCS mittels VMCLEAR freigeben. Nach der Migration kann mittels VMLAUNCH und im Folgenden VMRESUME mit dem selben VMCS weitergearbeitet werden.

Ein vereinfachter Wechsel zwischen den Modi VMX root und VMX nonroot mittels den Befehlen VMentry und VMexit wird von Abbildung 2.10 dargestellt. Nicht dargestellt wird dagegen die Aktivierung der verschiedenen VMCS-Instanzen vor dem Wechsel in den Modus VMX root mittels VMentry. Hierzu ist bei bestehendem VMCS die Instruktion VMRESUME, ansonsten das Kommando VMLAUNCH auszuführen.
Auch die Geschwindigkeit steigt mit dem Einsatz von Vanderpool. So sind in Vergleichsmessungen mit Vanderpool um ein vielfaches weniger Aktionen des VMM nötig als ohne. Dies ist auch in der Grafik 2.11 ersichtlich. Die Tests SYSmark Internet und SYSmark Office enthalten eine Reihe für Internet und Office Anwendungen typischen Aktionen. Zu beobachten ist in beiden Tests vor allem eine signifikante Abnahme der Instruktionen im Bereich Interrupt Handling bei der Verwendung von Vanderpool. In den anderen Bereichen fällt der Unterschied nicht so groß aus, es sind jedoch trotzdem positive Veränderungen messbar. Vanderpool wurde von Anfang an darauf ausgelegt erweiterungsfähig zu sein. Noch virtualisiert Vanderpool nur den Prozessor. Das Langfristige Ziel ist es jedoch klar, die gesamte i386 Plattform zu virtualisieren. Speichercontroller, Netzwerkkarten, Storage-Subsystem ja sogar Grafikkarten sollen in Zukunft Virtualisierungsunterstützung

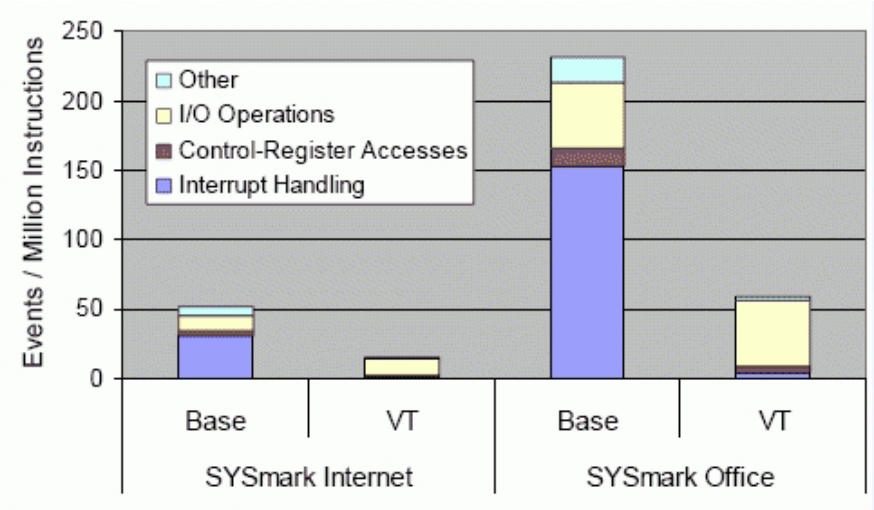

Abbildung 2.11: Interaktionshäufigkeit des VMM mit und ohne Vanderpool

bieten. Damit könnte in letzter Konsequenz sogar der Einsatz eines Virtual Machine Monitors in Software, der zur Zeit immer noch nötig ist, der Vergangenheit angehören. Bis es soweit ist, werden aber noch einige Jahre vergehen.

2.3.2 AMD Pacifica

AMDs Lösung zur Virtualisierungsunterstützung in Prozessoren mit dem Namen Pacifica ist weitgehend identisch mit Vanderpool. AMD geht jedoch einen Schritt weiter als die Konkurrenz und stellt zusätzliche Funktionen bereit. Genauer genommen ist Vanderpool damit nur eine Teilmenge von Pacifica. Die Grundfunktionen von Pacifica, die auch Vanderpool bietet, sind bis auf die Namen der Register und Befehle weitgehend identisch. So nennt AMD seinen Befehlssatz Secure Virtual Maschine (SVM) und aktiviert den „Guest-Mode" mit dem Befehl VMRUN. Beendet wird er mit #VMEXIT. Das dem VMCS entsprechende VMCB, was für Virtual Maschine Control Block steht, existiert ebenfalls. Diese Unterschiede in der Namensgebung ziehen sich durch die gesamte Architektur. Im Weiteren werden daher nur noch die zusätzlichen Funktionen behandelt.
Der wichtigste Unterschied ist die zusätzliche Virtualisierung des Speicher-Controllers mit Pacifica. Dies ist bei AMDs Prozessoren weniger schwierig zu lösen als bei Intel, da die Prozessoren AMD Athlon und AMD Opteron bereits über einen integrierten Speicher-Controller verfügen. Während bei Intel bei jedem Speicherzugriff einer virtuellen Maschine auf eine nicht im Arbeitsspeicher befindliche Seite der VMM aktiv werden muss, um die Speicheradresse zu übersetzen und damit zwangsläufig einen VMexit auslöst, werden bei AMDs Pacifica solche Speicherzugriffe von der Hardware abgefangen und übersetzt. Damit muss der Guest-Mode nicht verlassen werden und die Performanz erhöht sich. Ein zweiter Unterschied ist die Unterstützung von DMA (Direct Memory Access) Zugriffen aus virtuellen Maschinen über den sogenannten Device Exclusion Vector (DEV). Damit können DMA-fähige Geräte ohne Hilfe des Prozessors direkt auf Speicherbereiche der virtuellen Maschine zugreifen. Der DEV verhindert dabei den Zugriff auf den physikalischen Speicher des Gesamtsystems. Ein letzter Unterschied besteht in der Integration der Trusted-Computing-Technologie „Presido" in Pacifica. Daher trägt AMDs Technologie auch den Beinamen Secure Virtual Maschine Architecture. Für die eigentliche Virtualisierung sind diese Sicherheitsfunktionen unnötig, jedoch wird es mit ihrer Hilfe möglich, Software, die TPM (Trusted Platform Module)-Funktionen benötigt, in virtuellen Maschinen zu starten. Mit diesen Funktionen ist AMD seinem Konkurrenten Intel bereits einen kleinen Schritt in die Zukunft voraus. Allerdings sind bisher noch keine Prozessoren, die Pacifica enthalten, auf dem Markt erhältlich. Intel hat seit 14. November 2005 mit den Pentium 4 Modellen 662 und 672 bereits erste Produkte im Handel.

2.4 Virtual Private Networks (VPN)

Virtual Private Networks - kurz VPN - sind, wie der Name schon sagt, virtuelle, private Netze. Sie entstehen, wenn ein öffentliches Netz dazu verwendet wird, zwei private Netze - im Minimalfall auch nur zwei Rechner - miteinander zu verbinden. Ein virtuelles privates Netz ist eine Netzinfrastruktur, bei der Komponenten eines privaten Netzes über ein öffentliches Netz wie dem Internet miteinander kommunizieren, wobei sie die Illusion besitzen, das Netz zu ihrer alleinigen Verfügung zu haben [Ecke 04]. Dabei kommen im Normalfall kryptographische Verfahren zum Einsatz um die Integrität, Vertraulichkeit und Echtheit der übertragenen Daten zu gewährleisten. Der Einsatz von kryptographischen Verfahren ist aber optional und kann eventuell auch nur eine Untermenge der genannten Punkte realisieren. Typisch für VPNs ist jedoch, dass das Aufbringen spezieller Strukturen auf eine vorhandene, öffentliche Infrastruktur logisch gesehen ein abgekapseltes, virtuelles Netz erzeugt. VPNs zeigen daher ähnliche Strukturen, wie sie auch bei der Virtualisierung von Hardwarekomponenten im Abschnitt 2.2 zu finden sind.
Mittlerweile existieren mehrere unterschiedliche Implementierungen von VPNs. Die bekanntesten Lösungen implementieren dabei meist einen der drei folgenden Standards:

- IPSec - IP Security
- SSL - Secure Socket Layer / TLS - Transport Layer Security -VPNs
- PPTP - Point-to-Point Tunneling Protocol / L2TP - Layer 2 Tunneling Protocol

2.4.1 IPSec

IPSec [The 98] ist eine kryptographische Erweiterung des IP-Protokolls, die ursprünglich für IPv6 entwickelt wurde, aber mittlerweile auch für IPv4 existiert. Ziel bei der Entwicklung von IPSec war es, eine Sicherheitsarchitektur für die Kommunikation über IP-Netzwerke zu erstellen. IPSec ist in fast allen gängigen Betriebssystemen implementiert oder kann im Falle einiger Linux Distributionen sehr einfach über deren Paketverwaltung nachinstalliert werden. Der Standard IPSec gilt als sehr sicher, ist jedoch auch sehr komplex und damit fehleranfällig in seiner Konfiguration. Zudem ist es mit IPSec nicht möglich andere LAN-Protokolle als IP, also zum Beispiel IPX oder AppleTalk, zu übertragen.
IPSec bietet zwei Betriebsmodi sowie zwei Sicherungsarten, die miteinander kombiniert werden können. Als Modus kommt je nach Verwendungszweck entweder der Tunnelmode oder der Transportmode zum Einsatz. Sollen zwei Netze miteinander verbunden werden, wählt man meistens den Tunnelmode, bei der Verbindung zweier Rechner verwendet man primär den Transportmode. Diese Zuordnung ist allerdings nicht zwingend. Bei der Verwendung des Tunnelmodes wird das gesamte IP-Paket in ein neues Paket gepackt und ein unabhängiger IP-Header davorgesetzt. Somit kann das Paket „normal" im Internet geroutet werden, ohne dass die internen, eventuell sogar privaten IP-Adressen direkt sichtbar sind. Im Transportmode bleibt der ursprüngliche Header - IP-Adresse und weiter Optionen - erhalten. Die verwendeten Sicherungsarten sind Authentication Header (AH) und Encapsulation Security Payload (ESP). AH stellt nur die Integrität und Echtheit der Daten sicher, indem es Hashwerte (Hash Message Authentication Codes, HAMAC) über die zu sichernden Daten berechnet und diese verschlüsselt im AH-Header speichert. Sollen zusätzlich die Nutzdaten verschlüsselt werden, also Vertraulichkeit gewährleistet werden, so kommt ESP zum Einsatz. Den genauen Aufbau der IPSec-Pakete zeigt Abbildung 2.12.

Um den Paketinhalt verschlüsseln und kryptographische Prüfsummen berechnen zu können müssen Server und Client im Besitz zueinander passender Schlüssel sein. Der Schlüsselaustausch zwischen den beiden involvierten Systemen kann auf mehreren Wegen stattfinden. Die eleganteste Lösung ist der Einsatz des Internet-Key-Exchange-Protokoll (IKE), das neben dem Austausch der Schlüssel über das Diffie-Hellman Verfahren auch gleich die Aushandlung aller zur Kommunikation benötigten Parameter (Security Associations, SA) übernimmt. Da IKE aber eine eigene Zertifikatsverwaltung voraussetzt, ist es in einfach gelagerten Fällen bequemer Pre-Shared Keys (PSK) zu verwenden, die einmal manuell allen Teilnehmern mitgeteilt werden müssen. Dieses Verfahren benötigt zwar keine Zertifikatsverwaltung, hat aber das Problem, dass zum Beispiel beim Ausscheiden eines Mitarbeiters aus der Firma die Schlüssel auf allen anderen Clients ausgetauscht werden

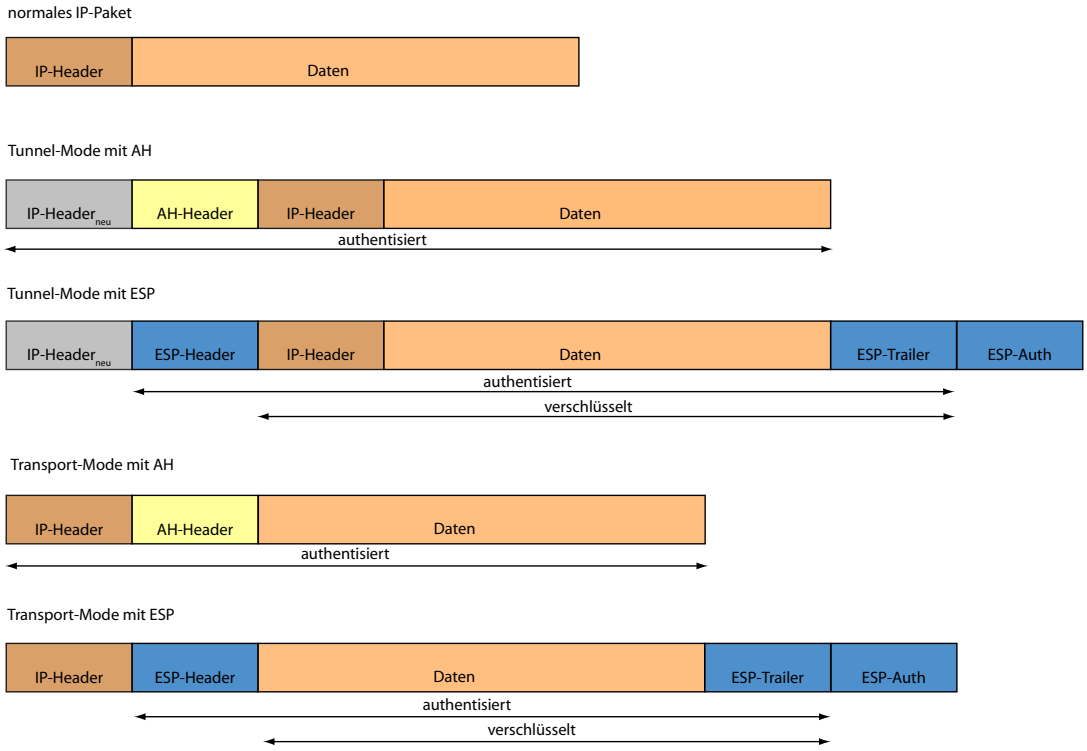

Abbildung 2.12: IPSec im Tunnel-/Transport-Mode mit AH oder ESP

müssen, um die Privatsphäre zu waren. Der Einsatz von PSKs ist daher hauptsächlich für private Zwecke interessant.
Da der IPSec Standard vor dem Aufkommen von DSL und DSL-Routern verabschiedet wurde, entstehen heute beim Einsatz von IPSec und NAT-Routern - wie es DSL-Router sind - einige Probleme. Da beim Einsatz von NAT die Quelladresse im IP-Header und gegebenenfalls auch der Quellport in den TCP/UDP-Headern der Pakete geändert wird, stimmen bei Verwendung von AH die HAMAC-Werte nicht mehr überein und das Paket wird von der Gegenseite verworfen, da es manipuliert wurde. Kommt ESP zum Einsatz, ist zwar das Ändern der Quell-IP-Adresse im neu erzeugten IP-Header kein Problem, die Ports im TCP/UDP-Header können allerdings nicht umgeschrieben werden, da sich der TCP/UDP-Header im Datenbereich des IP-Paketes befindet und dieser bei ESP verschlüsselt ist. Zu all diesen Problemen kommt es in der Praxis oft aber gar nicht, weil schon IKE an NAT scheitert. Viele IKE-Implementierungen arbeiten nur auf UDP Port 500 korrekt. Wird dieser Port durch NAT geändert, so schlägt der Schlüsselaustausch fehl. Die meisten neueren DSL-Router unterstützen daher ein IPSec-Passthrough bei dem der Wert des IKE-Ports nicht verändert wird. Auf diese Weise kann aber lediglich ein Client hinter dem Router eine IPSec Verbindung nach draußen aufbauen, für einen zweiten Client müsste wieder eine Portumsetzung erfolgen.
Beide Probleme lassen sich mit Hilfe vom NAT-Traversal lösen, jedoch nur bei der Verwendung von ESP. Beim Einsatz von NAT-Traversal wird zusätzlich zum IP-Header auch ein neuer UDP-Header erzeugt und das ursprüngliche IP-Paket in den UDP-Datenbereich gesetzt. Der neue UDP-Header ist ebenso wie der zusätzliche IP-Header im Tunnelmode weder verschlüsselt noch authentisiert (siehe Abbildung 2.13). Damit können NAT-Router sowohl IP-Adresse als auch Port des Paketes ändern, ohne die kryptographischen Prüfsummen zu verletzen. Auf der Gegenseite muss ein NAT-Traversal fähiger Router das ursprüngliche IPSec-Paket wieder entpacken, damit es weiter verarbeitet werden kann. Das Verfahren funktioniert ebenfalls, wenn sich mehrere IPSec-Clients hinter dem selben Router befinden und sich zum selben IPSec-Gateway verbinden. Ein Nachteil dieses Verfahrens ist aber der im Vergleich zum nativen IPSec klar gesteigerte Aufwand. IPSec wird damit noch komplizierter, als es in seiner nativen Architektur schon ist.
Bekannte Implementierungen von IPSec sind das seit 2004 nicht mehr weiterentwickelte FreeS/WAN [fre 06b]

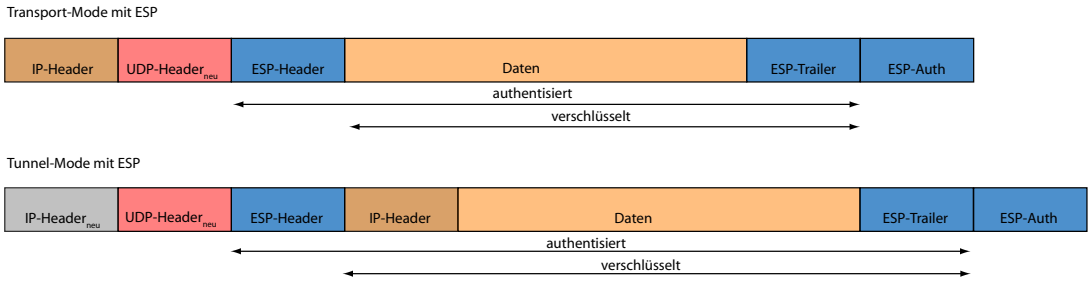

Abbildung 2.13: IPSec mit ESP und NAT-Traversal im Tunnel-/Transport-Mode

und die daraus entstandenen Projekte Openswan [ope 06b] und Strongswan [str 06].

2.4.2 SSL-VPN

Das Problem von IPSec mit NAT-Routern und dessen Komplexität sind die Gründe für die Entwicklung der SSL-VPNs. SSL [ssl 96] kam ursprünglich nur zur Sicherung der Verbindung zu Webservern wie etwa beim Homebanking oder beim Abruf einiger POP (Post Office Protocol) und IMAP (Internet Message Access Protocol) Konten zum Einsatz. Mittlerweile hat man erkannt, dass sich SSL mit einigen Anpassungen auch für den Einsatz in VPNs eignet. Da die für SSL benötigten Bibliotheken wie zum Beispiel OpenSSL [ope 06a] schon in der Vergangenheit sehr oft getestet und auf Schwachstellen untersucht worden sind, können sie als sehr sicher eingestuft werden. Die Technik zur Wahrung der Integrität, Echtheit und Vertraulichkeit übernimmt SSL-VPN weitgehend von IPSec. Einige SSL-VPNs verwenden statt SSL auch TLS. TLS [The 06] ist die überarbeitete und standardisierte Variante des von Netscape entwickelten SSL 3.0. TLS meldet sich daher auch als SSL 3.1. Sowohl SSL als auch TLS arbeiten ursprünglich auf Anwendungsebene, das heißt OSI-Schicht vier. Um SSL/TLS auch auf Schicht drei verwenden zu können, müssen sowohl auf dem Client als auch auf dem Server Umsetzer laufen, die die zu verschickenden IP-Pakete auf der Schicht vier in den Datenbereich eines neuen Paketes packen. Mit der selben Technik ist es auch möglich Pakete auf der OSI-Schicht zwei zu tunneln (siehe Abbildung 2.14). Auf diese Weise erreichen nicht nur reine Ethernet-Pakete, wie sie zum Beispiel ARP/RARP ((Reverse) Address Resolution Protokoll) verwendet, die Gegenseite, sondern ebenfalls nicht IP-Protokolle wie IPX oder AppleTalk. Die verbundenen Rechner verhalten sich damit so, als wären sie lokal am selben Switch angeschlossen. Der Einsatz von NAT-Routern stellt bei der Verwendung von SSL-VPNs ebenfalls kein Problem mehr da, da IP-Adresse und Quellport im IP-Header beziehungsweise TCP/UDP-Header des Pakets nicht per HAMAC geschützt werden. Im Gegensatz zum ursprünglichen SSL erfolgt jedoch beim Verbindungsaufbau sowohl eine serverseitige als auch eine clientseitige Authentisierung über Zertifikate oder PSK. Auch die Authentisierung über Passwörter ist möglich. Eine der bekanntesten und noch dazu freien Implementierungen eines SSL-VPNs ist OpenVPN. OpenVPN [Ope 06c] ist für beinahe alle großen Plattformen erhältlich und arbeitet sehr stabil. Dank ausführlicher Dokumentation lassen sich auch individuelle Konfigurationen mit einigen Vorkenntnissen schnell und problemlos realisieren.

2.4.3 PPTP & L2TP

Eine weitere Technik zum Erstellen von VPN-Verbindungen stellt PPTP (Point-to-Point Tunneling Protocol)[The 99] dar. PPTP wurde von einem Herstellerkonsortium, bestehend aus Ascend Communications, Microsoft Corporation, 3Com und anderen Mitgliedern, entwickelt. Technisch basiert PPTP auf PPP (Point-to-Point Protocol), das in Deutschland zur Einwahl ins Internet via Modem oder ISDN eingesetzt wird. Die einzelnen PPP-Pakete werden bei PPTP in GRE (Generic Routing Encapsulation Protocol) Pakete verpackt, welche über IP verschickt werden. PPP übernimmt dabei unter anderem die Aufgaben Authentisierung und Verschlüsselung. Für die Aushandlung der dafür benötigten Schlüssel wurde anfangs MSCHAPv1 (MicroSoft Challenge Handshake Authentication Protocol), ein Challenge Response Verfahren eingesetzt, bei dem eine vom Server im Klartext

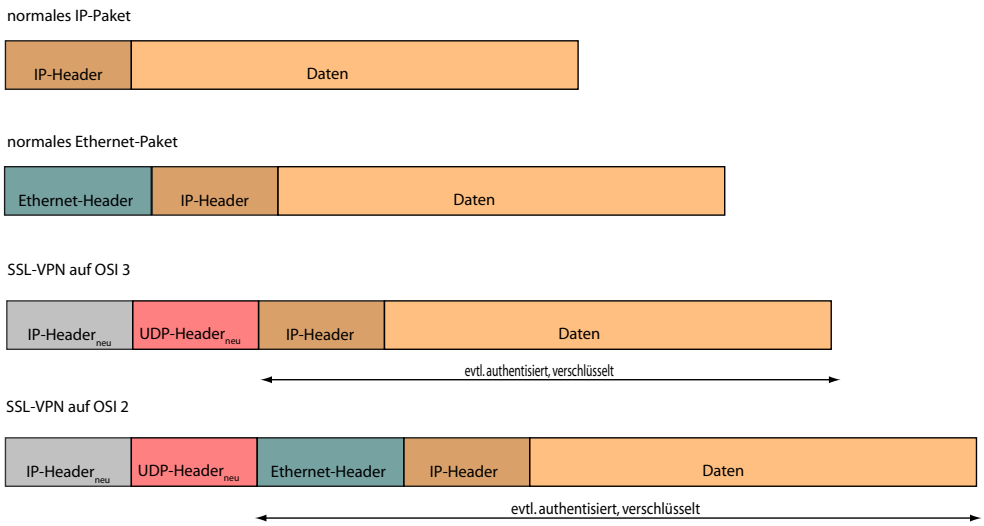

Abbildung 2.14: SSL-VPN auf OSI-Schicht zwei und drei

verschickte Challenge (Herausforderung) vom Client mit einem Passwort verschlüsselt und zurückgeschickt wird. Das Verfahren besaß jedoch einige Schwachstellen, so dass es wenig später von MSCHAPv2 abgelöst wurde. Auch MSCHAPv2 wurde 2001 von einer Gruppe Studenten gebrochen. Sie zeigten wie man das Verfahren innerhalb weniger Stunden knacken kann. Microsoft reagierte darauf, indem in alle Betriebssysteme ab Windows 2000 das Extensible Authentication Protocol (EAP) integriert wurde. EAP erweitert PPTP um die Unterstützung zur Authentisierung über Zertifikate und MD5-Hashes. Da GRE jedoch wie IPSec Probleme mit NAT-Routern besitzt, sofern diese kein PPTP-Passthrough unterstützen, war auch diese Lösung nicht befriedigend. Seit Windows 2000 existiert mit L2TP auch eine VPN-Variante basierend auf PPP, die keine eigene Verschlüsselung mit sich bringt, sondern die für diese Aufgabe Teile von IPSec verwendet.

3 Planung

Aufgabe dieses Kapitels ist es, einen Plan für die Architektur des virtuellen Netzes zu entwickeln, anhand welchem später die praktische Realisierung vorgenommen werden kann. Dazu zählen unter anderem das Anpassen der Netztopologien an eine virtuelle Infrastruktur sowie eine Vergrößerung der beiden Szenarien auf je 40 Praktikumsrechner. Ebenfalls zu klären ist die Frage, wie und durch wen das Management der Infrastruktur und der Clients erfolgen soll. Ein letzter Punkt befasst sich mit Sicherheitsanforderungen. Diese sind in diesem speziellen Fall nicht nur notwendig, wie in jedem anderen Serversystem, sondern aus zwei Gründen äußerst wichtig. Zum einen arbeiten die Teilnehmer des Praktikums mit sicherheitskritischen Softwaretools, angefangen beim einfachen Portscanner bis hin zu Tools, die Maschinen böswillig vom Netz trennen können. Schutz für die zentralen Netzwerkkomponenten ist also dringend nötig. Der zweite Punkt ist eher abstrakter Natur. Wie soll den Studenten Sicherheitsbewusstsein in der Informatik vermittelt werden, wenn nicht einmal die Sicherheit des Ausbildungsnetzwerks sichergestellt ist?

3.1 Machbarkeit

In einer vorausgehenden Arbeit [Lind 05] wurde bereits die Machbarkeit einer Virtualisierung des Praktikums untersucht und als realisierbar eingestuft. Allerdings war die Entwicklung von Xen zu Beginn der Untersuchung noch zu wenig fortgeschritten, um diese Technologie ernsthaft für die Virtualisierung der Praktikumsumgebung in Erwägung zu ziehen. Daher ging die Analyse von einer Umsetzung des Vorhabens mit User Mode Linux [Dike 05] aus. Erste Tests zu Beginn dieser Arbeit im kleineren Rahmen zeigten jedoch, dass Xen inzwischen durchaus für die Umsetzung geeignet ist und User Mode Linux sowohl in Performanz sowie Entwicklungsspielraum übertrifft. Hinsichtlich der Architektur unterscheiden sich User Mode Linux und Xen zwar deutlich (siehe Abschnitt 2.2.1), benötigen für ihre Funktionalität aber ähnliche Ressourcen. Beispielsweise benötigen beide Systeme Dateisysteme, die sich bis auf die an den jeweiligen Kernel angepassten Kernelmodule nicht unterscheiden. Die Installation von Xen dauert zwar insgesamt länger, da mehrere verschiedene Kernel übersetzt werden müssen, ist aber deutlich besser dokumentiert und im Gegensatz zu User Mode Linux auch von Anfängern durchführbar. In allen veröffentlichten Benchmarks (siehe Abb. 4.7, Seite 56) erzielt Xen gegenüber User Mode Linux die deutlich besseren Ergebnisse mit dem kleineren Overhead bezogen auf ein normales Linuxsystem. Daher ist damit zu rechnen, dass die Virtualisierung der Umgebung mit der in [Lind 05] veranschlagten Hardware auch mit Xen realisiert werden kann. Erste Tests auf einem Pentium 4, 3 GHz mit 2 GB RAM sowie SATA Festplatte verliefen Erfolg versprechend. Fünfundzwanzig virtuelle Maschinen ließen sich auf dem Testsystem problemlos starten und liefen stabil.
Die Netzwerkarchitektur von Xen unterscheidet sich deutlich von der, die User Mode Linux verwendet. Während User Mode Linux auf einen eigens für diesen Zweck entwickelten Switch namens `uml_switch` setzt, verwendet Xen die in den Linux Kernel integrierte Bridge. Dies hat Geschwindigkeitsvorteile für Xen zur Folge, da das anfallende Routing und Bridging zusammen mit den normal anfallenden Netzwerkpaketen direkt im Kernel behandelt werden kann und nicht von zusätzlichen Daemons ausgeführt werden muss. Ein weiterer Vorteil dieser Variante ist, dass auf diese Weise auch die Firewalls der Steuerdomain durchlaufen werden. Somit ist ein Filtern des Verkehrs im virtuellen Netz durch die Steuerdomain möglich.
In Summe bringt Xen mit seiner rundum durchdachteren Architektur für dieses Vorhaben nur Vorteile. User Mode Linux muss man jedoch zu Gute halten, dass es nie für den Einsatz als virtuelle Maschine geschaffen wurde, sondern immer als virtuelles Betriebssystem zur Ausführung einzelner Programme wie zum Beispiel Webserver in einer Sandbox gedacht war. Die nachfolgenden Kapitel beschreiben daher die Planung und Realisierung des Projekts mit Hilfe von Xen.

3.2 Erweiterung der Topologien auf 40 Maschinen

Als ersten Schritt der Planung müssen zunächst die beiden Szenarien (vgl. Abb. 2.1 Seite 4 und 2.2 Seite 5) auf 40 Maschinen erweitert werden, da das Praktikum in Zukunft nicht mehr in zwei Gruppen durchgeführt wird und jeweils zwei Studenten als Team zwei Rechner erhalten sollen. Für die Ausweitung der Szenarien wird das bisherige Schema der Vernetzung beibehalten, es wird lediglich die Anzahl der einzelnen Subnetze erhöht. Hierfür sind jedoch andere Subnetzmasken erforderlich als in den alten Szenarien, da der verwendete Adressraum ansonsten zu groß werden würde.
Bisher kam im gesamten Praktikumsnetz der IP-Bereich 192.168.216.0/24 zum Einsatz. Im ersten Szenario (Abbildung 2.1 Seite 4) wurde dieser aufgeteilt in die Subnetze

- 192.168.216.0/25 für `pcsec01` bis `pcsec05`
- 192.168.216.128/26 für `pcsec06` bis `pcsec10`
- 192.168.216.192/26 für das zentrale Netz.

Im zweiten Szenario (Abbildung 2.2 Seite 5) existierten die Subnetze

- 192.168.216.0/28 für `pcsec01` und `pcsec02`
- 192.168.216.16/28 für `pcsec03` und `pcsec04`
- 192.168.216.32/28 für `pcsec05` und `pcsec06`
- 192.168.216.64/28 für `pcsec07` und `pcsec08`
- 192.168.216.96/28 für `pcsec09` und `pcsec10`
- 192.168.216.192/26 für das zentrale Netz.

Um den IP-Bereich 192.168.216.0/24 trotz größerer Anzahl an Subnetzen beibehalten zu können, müssen die neuen Subnetze entsprechend kleiner gestaltet werden. Theoretisch kann bereits mit einer drei Bit langen Hostadresse, also Netzmaske 255.255.255.248 ein sechs Rechner starkes Subnetz adressiert werden, genug für unsere fünf Rechner großen Teilnetze im ersten Szenario. Der IP-Adressbereich lässt jedoch auch vier Bit lange Hostadressen (Subnetzmaske 255.255.255.240) zu, weshalb im ersten Szenario folgende Subnetzbildung (Abbildung 3.1) gewählt wird:

- 192.168.216.0/28 für `pcsec01` bis `pcsec05`
- 192.168.216.16/28 für `pcsec06` bis `pcsec10`
- 192.168.216.32/28 für `pcsec11` bis `pcsec15`
- 192.168.216.48/28 für `pcsec16` bis `pcsec20`
- 192.168.216.64/28 für `pcsec21` bis `pcsec25`
- 192.168.216.80/28 für `pcsec26` bis `pcsec30`
- 192.168.216.96/28 für `pcsec31` bis `pcsec35`
- 192.168.216.112/28 für `pcsec36` bis `pcsec40`
- 192.168.216.128/25 für das zentrale Netz.

Im zweiten Szenario müssen zwei Rechner pro Subnetz adressiert werden können. Eine zwei Bit lange Hostadresse würde also genügen. Auch hier lässt der IP-Bereich 192.168.216.0/24 aber drei Bit lange Hostadressen (Subnetzmaske 255.255.255.248) zu. Die Adressierung im zweiten Szenario (Abbildung 3.2) ergibt sich damit wie folgt:

- 192.168.216.0/29 für `pcsec01` und `pcsec02`
- 192.168.216.8/29 für `pcsec03` und `pcsec04`
- 192.168.216.16/29 für `pcsec05` und `pcsec06`

- 192.168.216.24/29 für `pcsec07` und `pcsec08`
- 192.168.216.32/29 für `pcsec09` und `pcsec10`
- 192.168.216.40/29 für `pcsec11` und `pcsec12`
- 192.168.216.48/29 für `pcsec13` und `pcsec14`
- 192.168.216.56/29 für `pcsec15` und `pcsec16`
- 192.168.216.64/29 für `pcsec17` und `pcsec18`
- 192.168.216.72/29 für `pcsec19` und `pcsec20`
- 192.168.216.80/29 für `pcsec21` und `pcsec22`
- 192.168.216.88/29 für `pcsec23` und `pcsec24`
- 192.168.216.96/29 für `pcsec25` und `pcsec26`
- 192.168.216.104/29 für `pcsec27` und `pcsec28`
- 192.168.216.112/29 für `pcsec29` und `pcsec30`
- 192.168.216.120/29 für `pcsec31` und `pcsec32`
- 192.168.216.128/29 für `pcsec33` und `pcsec34`
- 192.168.216.136/29 für `pcsec35` und `pcsec36`
- 192.168.216.144/29 für `pcsec37` und `pcsec38`
- 192.168.216.152/29 für `pcsec39` und `pcsec40`
- 192.168.216.192/26 für das zentrale Netz.

Alle Praktikumsrechner sind zusätzlich zum Praktikumsnetz über ihre Netzwerkkarte `eth0` an ein Managementnetz angeschlossen und sind über den IP-Bereich 192.168.10.0/24 adressierbar. Der Rechner `pcsec`*XX* kann über die Adresse `192.168.10.`*XX* erreicht werden. Dies ist unbedingt notwendig um die Rechner initial zu erreichen, da die übrigen Netzwerkkarten der Rechner im Gegensatz zu `eth0` nicht vorkonfiguriert sind. Diese Aufgabe sollen die Teilnehmer selbst erledigen. Als Gateway für das Managementnetz dient ein Login-Server, der aus dem Internet zur Zeit unter der Adresse `141.84.218.191` bzw. `secplogin.nm.ifi.lmu.de` erreichbar ist. Dem weiteren Aufbau des Managementnetzes widmet sich der Abschnitt 3.4. Die Abbildungen 3.1 und 3.2 zeigen die beiden Szenarien, wie sie im Folgenden umgesetzt werden.

3.3 Grundlegende Komponenten zur Realisierung

Hat man die Architektur und Funktionsweise von Xen verstanden, drängt sich ein Konzept für die grundlegende Umsetzung eines virtuellen Netzwerkes förmlich auf. Virtuelle Maschinen werden mit Hilfe von Xen erstellt, ihre Vernetzung übernehmen Bridges, die mit Hilfe des Linux Kernel Bridging erstellt werden können. Als Massenspeicher für die virtuellen Dateisysteme sollen „normale Images" - eine Partition in einer Datei - zum Einsatz kommen. Diese sind zwar bekannt für Leistungseinbußen bei hohem Lese- und Schreibaufkommen, sind aber dafür deutlich einfacher, sicherer und flexibler zu handhaben als beispielsweise LVM (Logical Volume Manager) basierte Images oder Festplattenpartitionen. Der Einsatz von CoW (Copy on Write)-Files, wie sie in [Lind 05] unter User Mode Linux eingesetzt wurden, ist mit Xen derzeit nicht möglich, da sich entsprechende Tools noch im Entwicklungsstadium befinden. CoW bezeichnet eine Art der Bereitstellung von Images, bei der viele ähnliche Images lediglich die Differenz zu einem Standard-Image speichern. Das Standard-Image ist für alle Instanzen lesbar, während Schreibzugriffe pro Instanz in eine separate Datei umgelenkt werden und die Inhalte im Standard-Image verschatten. Auf diese Weise benötigen viele ähnliche Images nur sehr wenig Speicherplatz auf der Festplatte.
Im Praktikum existieren nur sehr wenige Aufgaben, welche festplattenintensiv sind. Hierunter fallen unter anderem die Installation von Softwarepaketen und das Kompilieren von Software aus Quelltextpaketen, so dass der Vorteil der einfacheren Handhabbarkeit von Images gegenüber LVM oder anderen Techniken überwiegt.

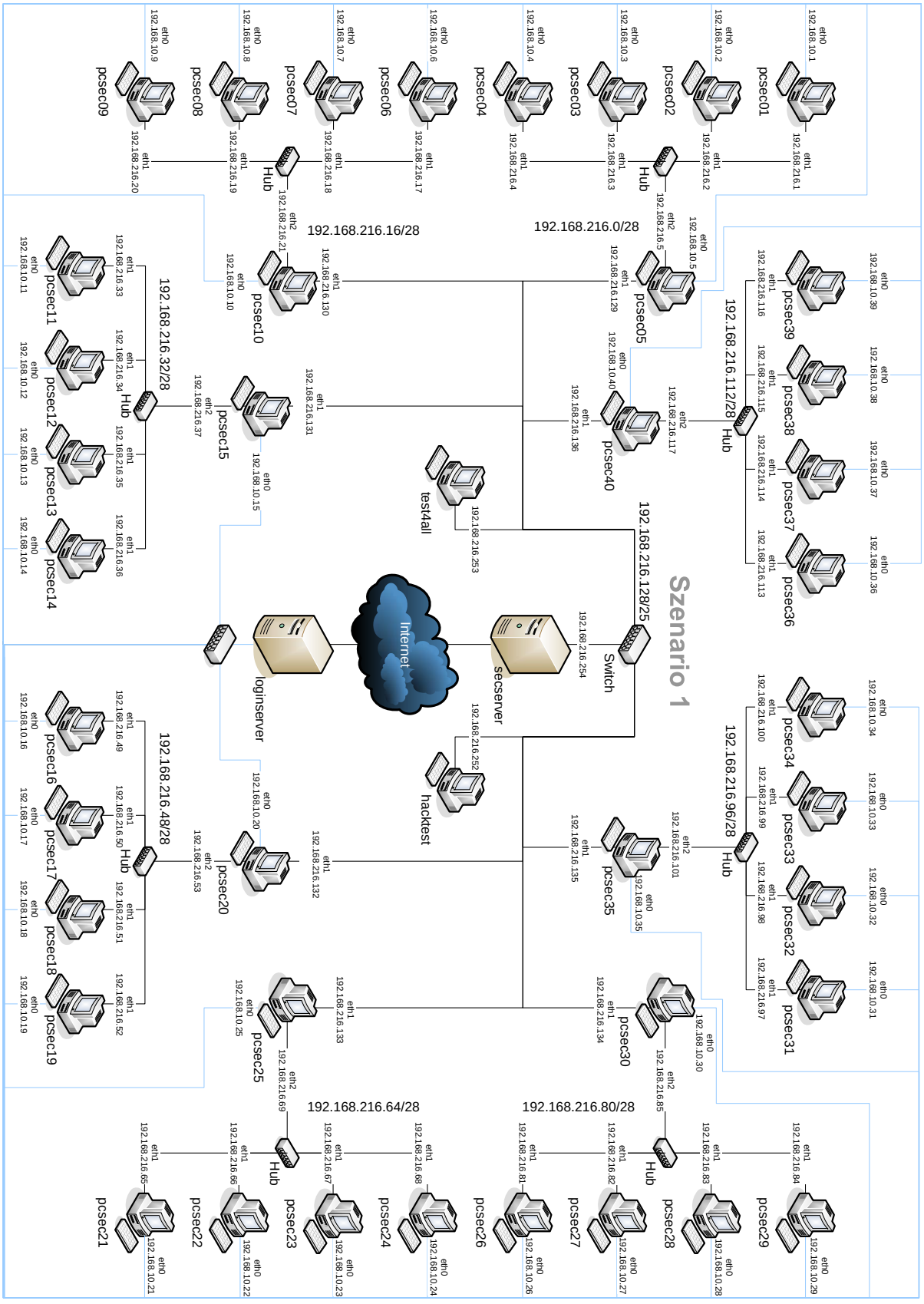

Abbildung 3.1: Szenario 1

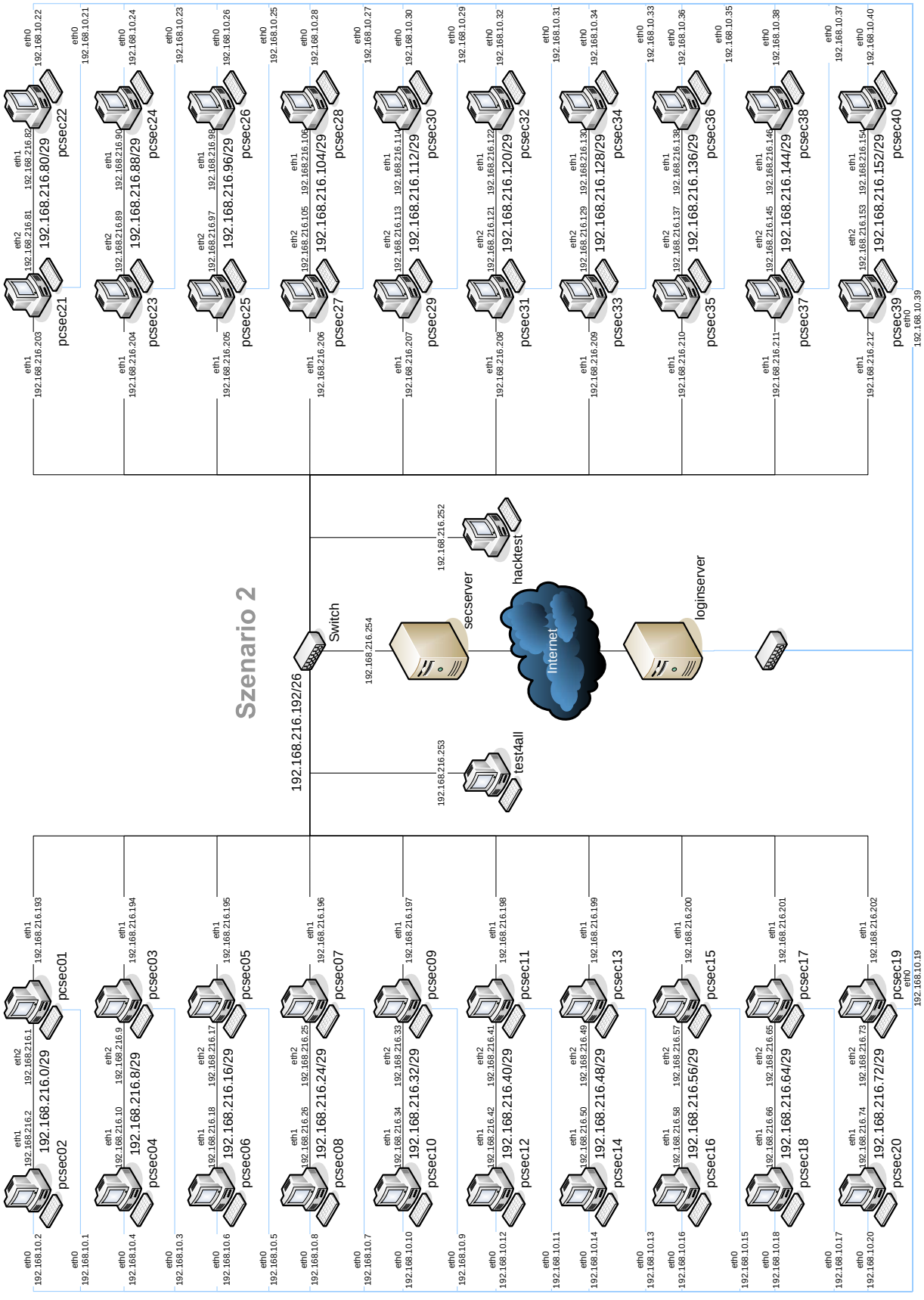

Abbildung 3.2: Szenario 2

Die Dateisysteme der virtuellen Maschinen werden daher mit Images realisiert. Dem entsprechend muss genug Speicherplatz im Serversystem für 44 Images[1] und deren Backups verfügbar sein. Bei einer Größe der Images von etwa 3,2 GB sollten mindestens 280 GB auf lokalen Festplatten bereitgestellt werden.
Die Analyse zur Machbarkeit [Lind 05] empfiehlt für ein Serversystem mindestens 4 GB Hauptspeicher, ein Zweiprozessorsystem und entsprechend schnellen Massenspeicher. Als Grundlage für die Realisierung dient daher eine gut ausgebaute Workstation von Fujitsu-Siemens, das Modell CELSIUS v810. Der Rechner ist ein Dual Opteron System mit 4 GB RAM, GBit-LAN, sowie vier SATA-Platten mit je 160 GB. Da das System auf den 64-Bit fähigen AMD Opteron Prozessoren basiert, kann das System später unter Einsatz von Xen 3 auf mehr als 4 GB RAM aufgerüstet werden. Für die erste Realisierung kommt das 32-Bit fähige Xen 2.0.7 zum Einsatz.

3.4 Management

Das Management von Rechnern, speziell das Management virtueller Maschinen, umfasst einen relativ großen Aufgabenbereich. So müssen sich alle VMMs (Virtual Maschine Monitor) mit der Aufgabe befassen, neue Maschinen zu erzeugen und bestehende zu verwalten, wozu beispielsweise das Pausieren und Fortsetzen von virtuellen Maschinen aber auch ihr Beenden gehört. Ein weiterer Aspekt ist das Anlegen von Backups und die Durchführung von Wiederherstellungen einzelner Dateien oder ganzer Images aus Backups. In größeren Realisierungen wie unserem Praktikumsnetz fallen auch Managementaufgaben am Netzwerk selbst an. Erschwert wird dieser Vorgang zusätzlich dadurch, dass Benutzer jeweils nur für bestimmte Teilaufgaben berechtigt sein sollen. Um ein Konzept für den Aufbau einer Management-Lösung entwickeln zu können, müssen daher als erstes die Anforderungen geklärt werden.

3.4.1 Anforderungen an das Management

In unserem Fall ergeben sich konkret folgende Anforderungen an das Management: Jedem Teilnehmer muss es möglich sein, seine eigene virtuelle Maschine im Falle eines Softwareproblems neu zu starten. Dies ist auch dann erforderlich, wenn die Maschine über das Netzwerk nicht mehr erreichbar ist. Dies ist zum Beispiel bei Versuchen mit Firewalls sehr leicht möglich. Ein versehentlich nicht geöffneter Port 22 macht die virtuelle Maschine für den Teilnehmer über SSH unerreichbar. Da SSH jedoch der einzige vorkonfigurierte Zugang sein wird über den die Rechner erreichbar sein sollen, ist eine virtuelle Maschine ohne Zugang zur Steuerdomain nicht mehr administrierbar. Ebenso soll eine Möglichkeit geschaffen werden, Backups und Restores der eigenen Maschine durchzuführen. Diese Aufgaben sind mit den entsprechenden Tools leicht zu bewerkstelligen. Allerdings werden zum Einsatz dieser Tools Rootrechte am Serversystem benötigt. Da diese aus Sicherheitsgründen für das gesamte System nicht eingeräumt werden können, muss eine andere Lösung gefunden werden. Ein weiterer zu beachtender Aspekt ist, dass alle unkonfigurierten Maschinen für die Teilnehmer erreichbar sein müssen. Da dies über das Praktikumsnetz nicht möglich ist, muss ein Managementnetz erstellt werden, das diese Funktionalität bietet.
Das Management gliedert sich aus diesen Gründen in zwei Bereiche. Zum einen muss ein Managementnetz erstellt werden, über das die virtuellen Maschinen konfiguriert werden können, zum anderen muss ein Tool zur Durchführung administrativer Aufgaben gefunden beziehungsweise implementiert werden.

3.4.2 Aufbau eines Managementnetzes

Der Aufbau des Managementnetzes ist verhältnismäßig einfach. Jeder Rechner, der über das Managementnetz erreichbar sein soll, erhält eine zusätzliche Netzwerkkarte, die im System zwingend als `eth0` eingebunden sein muss. In der Konfigurationsdatei für virtuelle Maschinen erlaubt es Xen für diesen Netzadapter eine Konfiguration anzugeben, die beim Start der virtuellen Maschine gesetzt wird, solange im System keine abweichende Konfiguration vorhanden ist. Um sich an einer virtuellen Maschine anmelden zu können, muss ein entsprechender Dienst wie SSH oder Telnet aktiv sein. Da Telnet Passwörter bei der Authentisierung und Nutzdaten unverschlüsselt überträgt, verbietet sich der Einsatz von Telnet in einem Sicherheitspraktikum von

[1] 40 `pcsec` Rechner sowie die Rechner `secserver`, `test4all`, `hacktest` und `secplogin`

selbst. Die Wahl fällt daher auf SSH. Die Rechner sind mit einer solchen Konfiguration bereits aus dem Managementnetz erreichbar.
Ein Problem bleibt damit aber noch ungelöst. Wie erreicht ein Teilnehmer von seinem PC zu Hause oder aus dem CIP-Pool das Managementnetz? Durch die Verwendung privater Adressbereiche im Managementnetz scheidet einfaches Routing über den Login-Server aus, da die privaten Adressen im Internet nicht geroutet werden. Ein anderer Lösungsansatz ist das Login mit Hilfe einer Secure Shell auf dem Login-Server von wo aus das Managementnetz erreichbar ist. Diese Variante hat den Nachteil, dass die Teilnehmer, selbst wenn sie nur unprivilegierte Accounts haben, das Login direkt auf dem Eingangsrouter des virtuellen Netzwerks durchführen würden. Dies wäre die klassische Ausgangssituation für eine „man in the middle" Attacke.
Eine Lösung dieses Problems ist der Einsatz eines VPNs. Auf dem Login-Server kann ein VPN-Server gestartet werden, der Verbindungen aus dem Internet akzeptiert und das interne Managementnetz verfügbar macht. Um auf den virtuellen Maschinen keine Route zu den externen Rechnern angeben zu müssen, bietet sich ein bridged-VPN an, das auf der OSI-Schicht zwei arbeitet. Damit verhalten sich die externen Rechner, als wären sie lokal am selben Switch angeschlossen wie die internen Maschinen. Der Einsatz von Firewalls zwischen internem und externem Netz bleibt jedoch weiter möglich. Die VPN-Technik bringt noch einen weiteren Vorteil mit sich. Während bei reinem SSH-Zugriff auf die Maschinen das Setzen eines X-Forwarders (ssh -X) die einzige Möglichkeit ist grafische Oberflächen zu verwenden, ermöglicht der VPN Zugriff zusätzlich den Einsatz anderer Remote Desktop Architekturen. Am bekanntesten sind hier diverse VNC (Virtual Network Computing)-Varianten. Noch besser in der Performanz ist FreeNX [Fre 06a]. Performanz ist auch das Hauptargument gegen den Einsatz von X-Forwarding basierend auf SSH. Der Fensteraufbau eines Programms über das Internet dauert mit dieser Technik selbst bei DSL-Leitungen oft mehrere Minuten, während mit FreeNX beispielsweise KDE-Sessions in einer Performanz möglich sind, als säße man lokal vor dem Rechner.
Eine Vorgabe bei der Planung des Aufbaus der virtuellen Infrastruktur war von Anfang an, dass die Aufgaben im Praktikum IT-Sicherheit vom CIP-Pool aus bearbeitet werden können müssen. Leider treten beim Einsatz von OpenVPN unter SELinux, welches im CIP-Pool installiert ist, Probleme auf. OpenVPN benötigt zum Verbindungsaufbau zwangsweise Rootrechte, die den Studenten nicht eingeräumt werden können. Daher ist es nötig einen alternativen Zugang zum Praktikumsnetz bereitzustellen. Diese Alternative ist ein SSH-Zugang zu den Rechnern im virtuellen Netz. Wie bereits erläutert, soll es den Teilnehmern nicht möglich sein, ein Login auf dem Gateway durchzuführen, bei dem sie eine Shell auf dem System erlangen. Aus diesem Grund wird nicht ein Zugang zum Login-Server selbst realisiert, sondern nur ein SSH-Tunnel über den Login-Server als Gateway an einen Endpunkt im Managementnetz. Möglich ist dies durch die Verwendung von Schlüsseln zur Authentisierung. In der Datei `authorized_keys` auf dem Server kann zusammen mit dem Public-Key ein Kommando eingetragen werden, das ausgeführt wird, wenn eine Verbindung hergestellt wird. Die Verbindung - und damit der Tunnel - bleibt so lange aktiv, wie das Kommando zur Abarbeitung benötigt. Anschließend wird die Verbindung automatisch getrennt. Die Ausführung von anderen Programmen wie den vorgegebenen ist nicht möglich, sofern der benutzte Account kein Passwort besitzt. Trägt man nun als Kommando eine endlose Schleife ein, die in ihrem Rumpf nur einen sleep-Befehl enthält, so bleibt ein aufgebauter Tunnel genau so lange bestehen, bis die Schleife manuell abgebrochen wird.

```
command='echo connection established; while :; do sleep 10; done'
```

Während der Tunnel existiert, sind durch ihn sowohl Verbindungen via SSH und NX zu den Praktikumsrechnern möglich, als auch Verbindungen zum Virtual Machine Management Tool (siehe Abschnitt 3.4.3) über Port 80. Die genaue Syntax zum Aufbau des Tunnels kann dem Anhang C entnommen werden.

3.4.3 Managementtools zur Steuerung virtueller Maschinen

xm

Xen enthält in der Standardinstallation bereits ein Programm zur Kontrolle aller laufenden Domains. Mit Hilfe des xm-Kommandos sind alle wesentlichen Managementaufgaben auf Kommandozeilenebene durchführbar. Die allgemeine Syntax des Programms ist

```
xm command [switches] [arguments] [variables].
```

command steht dabei unter anderem für eines der Schlüsselwörter `create`, `destroy`, `shutdown`, `pause`, `unpause`, `list`, `console`, `balloon`, `migrate` oder `help`.
Die einzelnen Kommandos haben folgende Wirkung:

- `create` bringt eine neue Domain zur Ausführung.
- `destroy` zerstört eine Domain sofort.
- `shutdown` fährt eine virtuelle Maschine herunter und beendet sie anschließend.
- `pause` friert den Zustand einer Maschine ein.
- `unpause` bringt eine Maschine im Zustand paused wieder zur Ausführung.
- `list` zeigt eine Liste aller existierenden Domains.
- `console` verbindet die Konsole auf die angegebene Maschine.
- `balloon` vergrößert oder verkleinert den Arbeitsspeicher einer Domain.
- `migrate` migriert eine Domain über das Netz auf eine andere physische Maschine.
- `help` liefert eine detaillierte Hilfe zu den aufgeführten Kommandos sowie ihren Optionen.

Nahezu alle Kommandos benötigen als Schalter den Namen oder die ID der Maschine, auf die sie angewendet werden sollen. Ausnahmen sind `list` und `help`. Die Argumente der einzelnen Kommandos können mittels `xm help` erfragt werden. Variablen sind benutzerorientiert und kommen hauptsächlich im Kontext von `xm create` zum Einsatz. Der Grund hierfür sind die unzähligen Argumente, die für die Erstellung einer virtuellen Maschine benötigt werden. Als Beispiel seien hier nur die Angabe des gewünschten Xen-Kernels, die Größe des Arbeitsspeichers, Konfiguration von Netzwerk und Massenspeichermedien sowie Bootparameter des Kernels genannt. Um nicht jedes Mal alle Parameter erneut von Hand eingeben zu müssen, können Konfigurationsdateien angelegt werden, die `xm create` direkt übergeben werden können. Die Konfigurationsdateien sind wie `xm` selbst Python-Skripte, die in der Regel nur entsprechende Variablenbelegungen enthalten.
Um bei vielen ähnlichen Konfigurationen nicht für jede Maschine eine eigene Konfigurationsdatei anlegen zu müssen, können dem Skript Argumente übergeben werden, die bei der Ausführung zur Verfügung stehen. Auf diese Weise können die Argumente für `xm create` mit Hilfe von benutzerdefinierten Variablen berechnet werden. Diese Möglichkeit erleichtert Änderungen an der Konfiguration der virtuellen Maschinen, die alle Maschinen betreffen, erheblich, da nur noch eine Datei editiert werden muss, während bei Einzelkonfigurationen unter Umständen sehr viele identische Änderungen an mehreren Dateien durchgeführt werden müssen. Beispiele für Konfigurationen von Xen-Maschinen sind im Anhang B ab Seite 92 zu finden. Das nachfolgende Beispiel (siehe Listing 3.1) zeigt exemplarisch am Beispiel der Netzwerkkartenkonfiguration wie entsprechende Parameter mit Hilfe von Python berechnet und gesetzt werden kann.

Listing 3.1: Dynamische Konfiguration der Netzwerkkarten im ersten Szenario

```
# Define network interfaces.

# Number of network interfaces. Default is 1.
if (int(vmid) % 5 == 0):
        nics=3
else:
        nics=2

# Optionally define mac and/or bridge for the network interfaces.
# Random MACs are assigned if not given.
vif = ['mac=aa:00:00:00:ff:%02d,␣bridge=brmgnt' % int(vmid),
        'mac=aa:00:00:00:%02d:%02d,␣bridge=brsec%02d' % ((((int(vmid)-1)/5)+1),
            int(vmid),(((int(vmid)-1)/5)+1)) ]
if (int(vmid) % 5 == 0):
        vif.insert(1,'mac=aa:00:00:00:00:%02d,␣bridge=brsec00' % int(vmid))
```

Um das xm Kommando verwenden zu können, muss lediglich der Xen Daemon xend laufen. Dieser ist ebenfalls in Python geschrieben und stellt die Schnittstelle zu Xen dar. Diese Schnittstelle steht jedoch nur in der privilegierten Domain0 zur Verfügung. Eine Steuerung von anderen Maschinen aus ist daher nicht möglich. Für den Einsatz im Praktikum ist xm trotz seiner relativ einfachen Bedienung und großem Funktionsumfang nicht direkt einsetzbar. Das liegt an zwei Dingen. Zum einen erfordert das Ausführen von xm Zugang zur Domain0 sowie Rootrechte. Jeder einzelne Teilnehmer könnte damit gewollt oder ungewollt das Serversystem des Praktikums beschädigen. Der zweite Grund ist die fehlende Möglichkeit der Autorisierung und damit die Steuerung auf einzelne Maschinen zu beschränken.

xensv

Ebenfalls in Xen integriert ist der Webserver xensv, der die Fähigkeiten von xm mit einigen Einschränkungen grafisch über ein Webinterface zur Verfügung stellt. Leider wird er bereits seit über einem Jahr nicht mehr weiterentwickelt. So erstaunt es auch nicht, dass die Installation fehlschlägt und xensv nur manuell in Betrieb gesetzt werden kann. Die Tatsache, dass dieser Fehler überhaupt behoben werden kann, ist nur der offenen Architektur zu verdanken, denn auch er ist komplett in Python geschrieben. Dies betrifft nicht nur den Server selbst, sondern auch die Art und Weise, wie der nötige HTML Code serverseitig generiert wird. Alle Komponenten werden mittels Python erstellt. Die Verbindung von Server und HTML Codegenerierung sowie der Einsatz von mehrstufigen Vererbungshierarchien innerhalb von xensv machen den Quelltext nur sehr schwer verständlich. Für den Einsatz im Praktikum ist der Server nicht geeignet. Er setzt zwar im Gegensatz zu xm keinen direkten Zugang zum Serversystem voraus, Nutzertrennung und Autorisierung beherrscht aber auch er nicht. Ein weiterer Kritikpunkt ist, dass auch der Server Rootrechte benötigt, um seine Aufgabe erfolgreich zu erfüllen. Auch wenn der Server nur sehr klein und übersichtlich ist, entstehen dadurch potenzielle Angriffspunkte am Serversystem, die im Falle eines erfolgreichen Angriffs zur Folge haben, dass ein Angreifer Rootrechte im System erlangen könnte.

Virtual Maschine Management Tool (VMMT)

Da keine anderen Managementwerkzeuge neben xm und xensv existieren, bleibt für das Praktikum nur, eine eigene Managementlösung zu entwickeln. Die beiden vorgestellten Tools sind trotz alledem nicht nutzlos. Sie zeigen die Möglichkeiten und Schnittstellen zum Zugriff auf Routinen im Xen Daemon auf und erleichtern die Aufgabe dadurch enorm. Aufgrund der Architektur ist auch der selbst entwickelte Webserver gezwungen Python einzusetzen, um auf den bestehenden Schnittstellen aufsetzen zu können. Um den Code knapp und übersichtlich zu halten sowie unnötige Implementierungsarbeit sparen zu können, soll der weit verbreitete Webserver Apache zum Einsatz kommen, für den auch ein Modul zur serverseitigen Ausführung von Python Code existiert. Das Modul nennt sich mod_python und ist in SuSE 9.3 bereits als RPM enthalten. mod_python unterstützt neben der serverseitigen Generierung von HTML Code auch die Erstellung eigener Authentifzierungs-Handler, welche vom Apache beim Zugriff auf definierte Verzeichnisse verwendet werden können. Zur Authentisierung sollen die DES-Hashes der Root-Passwörter in den Images der virtuellen Maschinen dienen. Das hat den Vorteil, dass nicht zwei verschiedene Passwörter zum Anmelden und Administrieren der selben Maschine notwendig sind.
Da der Apache Server bereits in der Standardkonfiguration mit minimalsten Rechten läuft, geht von ihm keine große Gefahr für die Domain0 aus. Für die nötigen Managementaktionen werden jedoch zwingend Rootrechte benötigt, denn das Einhängen von Dateisystemen, das Lesen der Root Passworthashes aus /etc/shadow sowie das Durchführen von Backup- und Restore-Operationen sind mit den beschränkten Rechten des Apache Users nicht möglich. Entsprechende Aktionen können jedoch mit Hilfe von sudo, ermöglicht werden. sudo erlaubt es unprivilegierten Nutzern definierte Programme, eventuell auch mit eingeschränkten Parametern, mit Rootrechten auszuführen. Damit hält sich das Risiko zu hoher Privilegien für öffentlich zugängliche Dienste in Grenzen und es können dennoch alle Aufgaben durch den Server ausgeführt werden.

3.5 Sicherheitsanforderungen an die Architektur des Gesamtsystems

Die Anforderungen an ein virtuelles Rechnernetz sind neben der grundsätzlichen Funktionalität auch Sicherheit vor Angriffen und falschem Benutzen sowie Stabilität, welche nur durch eine handfeste Architektur gegeben ist. Das virtuelle Netz gilt nur dann als sicher, wenn kein unbefugter Zugriff auf das Serversystem und die Kontrolldomain erfolgen kann. Das virtuelle Netz selbst soll sich von einem realen Netz nicht unterscheiden, das heißt vorhandene Unterschiede — hauptsächlich im Routing — müssen entfernt werden. Dies ist nicht nur nötig um einen möglichst detailgetreuen Nachbau des realen Netzes zu bekommen, sondern auch sicherheitskritisch, da alle Bridges innerhalb der Steuerdomain betrieben werden und der gesamte Verkehr des virtuellen Netzes damit die Domain0 physisch erreicht. Auch ein Ausbrechen aus der virtuellen Maschine darf nicht möglich sein. Dies sicherzustellen ist jedoch Aufgabe der Xen-Entwickler. Für die Sicherheit der Implementierung ist es daher im Wesentlichen nur wichtig, streng auf eine Trennung zwischen Serversystem und virtuellem Netz zu achten. Dieser Punkt wurde im Folgenden bei der Erstellung eines Plans für die Realisierung konsequent umgesetzt.

3.6 Physischer Aufbau des virtuellen Netzes

Den groben Aufbau des virtuellen Netzes zeigt Abbildung 3.3, welche der Übersicht halber nur einen Ausschnitt aus dem virtuellen Netz des ersten Szenarios darstellt. Dargestellt werden das Kernnetz mit `Secserver`, `test4all`, `hacktest`, Managementnetz und Login-Server sowie VPN. Für die eigentlichen Subnetze mit den `pcsec` Rechnern steht exemplarisch das blau eingefärbte Subnetz bestehend aus den Rechnern `pcsec01` bis `pcsec05`. Die übrigen Subnetze sind analog aufgebaut. Auf ihre Darstellung wurde der Übersichtlichkeit halber verzichtet. Grün dargestellt sind alle virtuellen Maschinen aus dem zentralen Netz sowie der Login-Server. Gelb dargestellt sind alle Bridges, die je nach Konfiguration als Switch oder Hub arbeiten und mit den Netzwerkkarten (orange) verbunden werden können. Bei allen Netzwerkkarten, ausgenommen den Karten `eth0` in den Rechnern Domain0 und `secplogin`, welche auch farblich abgesetzt sind, handelt es sich ausschließlich um virtuelle Adapter, die von Xen, genauer gesagt durch das Back- und Frontend-Treiber System, bereitgestellt werden. Die beiden übrigen Netzwerkkarten sind reale Adapter. Beheimatet wird das Szenario innerhalb der Domain0, der Steuerdomain. Dabei laufen die virtuellen Maschinen natürlich nicht innerhalb der Domain0, sondern, wie im Architekturteil (Abschnitt 2.2.3) gezeigt, nebeneinander auf der Xen-Plattform. Die Zeichnung 3.3 zeigt den Aufbau des Systems aus anderer Sichtweise: Es ist Aufgabe der Domain0 alle virtuellen Maschinen zu steuern und ihnen benötigte virtuelle Komponenten wie Dateisysteme, Netzwerkkarten und Bridges mit Ausnahme von `br0` bereitzustellen. Die Domain0 bildet somit die Grundlage für den Betrieb weiterer virtueller Maschinen. Eine Darstellung im Sinne der Architektur mit Front- und Backendsystemen ist wegen ihrer Komplexität in dieser Größenordnung nur schwer nachvollziehbar und wird aus diesem Grund hier nicht weiter aufgeführt.
Als einziger Zugangspunkt in das virtuelle Netz dient die Netzwerkkarte `eth0` des Login-Servers. Die Schnittstelle soll aus dem Internet via SSH und OpenVPN erreichbar sein. Der Endpunkt des VPNs ist das Tap-Interface `tap0`. Es ist zusammen mit der Schnittstelle `eth1` im Managementnetz über die Bridge `br0` verbunden. Das Managementnetz dehnt sich damit bis auf die VPN-Clients aus, da der Verkehr auf der OSI Schicht zwei über die Bridge weitergeleitet wird. (Siehe hierzu auch Abschnitt 3.4.2.)
Um nicht eine dritte Netzwerkkarte in das Serversystem einbauen zu müssen, wird der ausgehende Netzwerkverkehr des Servers `secserver` über das selbe physische Interface geleitet, wie der des Login-Servers `secplogin`. Dazu ist es nötig, den Login-Server mit NAT-Funktionalität auszustatten, da die internen IP-Adressen des Praktikumnetzes extern nicht bekannt sind und ohne Adressumsetzung keine Verbindung ins Internet möglich wäre. Dies wird schematisch auch in Abbildung 3.3 sichtbar.
Um Zugang zum Webserver des VMMT innerhalb der Domain0 zu erlangen, fungiert der Login-Server zusätzlich als einfacher HTTP-Proxy, der Anfragen an das VMMT auf Port 80 an den Webserver der Domain0 transparent weiterleitet. Das Vorhandensein der Domain0 wird dadurch verdeckt und die Trennung zwischen virtuellem Netz und physischer Hardware bleibt gewahrt.

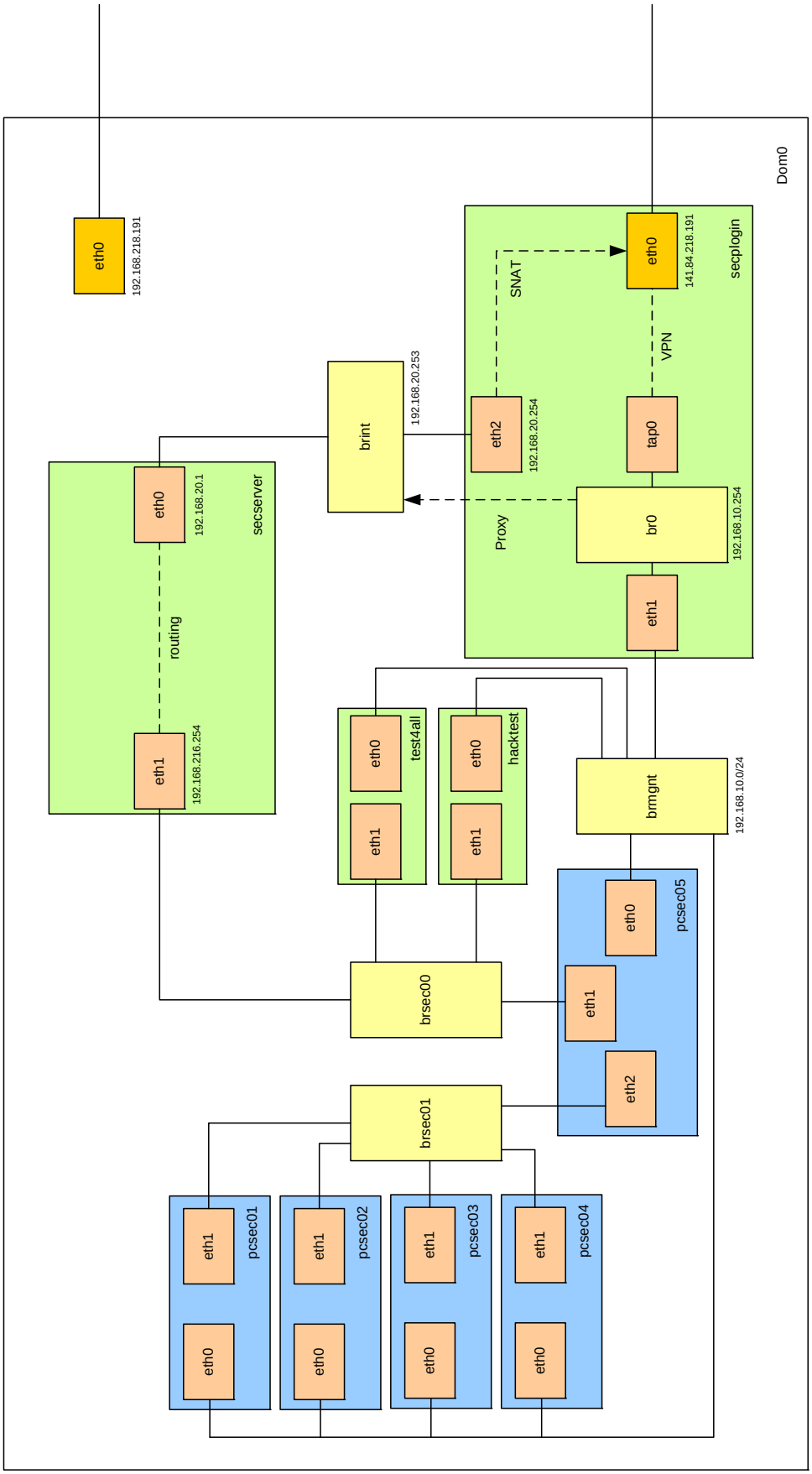

Abbildung 3.3: physischer Aufbau des virtuellen Netzes

3.7 Durchsetzung der Sicherheitsrichtlinien im Betrieb

Die Einhaltung der oben genannten Punkte erfordert weitgehende Maßnahmen. Die wohl weitreichenste Maßnahme zur Absicherung der Steuerdomain ist die Einführung eines virtuellen Login-Servers, der über eine eigene physikalische Netzwerkkarte verfügt. Über die Netzwerkkarte `eth0` des Login-Servers läuft, wie in Abbildung 3.3 zu sehen ist, der komplette Netzwerkverkehr zur Steuerung der virtuellen Maschinen. Der physikalische Netzadapter der Steuerdomain `eth0` muss nicht mehr öffentlich zugänglich sein und kann in das private Subnetz des Lehrstuhls gelegt werden, wo er von außen nicht erreichbar ist. Dadurch erreicht man eine vollkommene Trennung zwischen virtuellem Netzwerk und Steuerdomain. Ein Angriff auf das virtuelle Netz betrifft somit nur noch das virtuelle Netz und alle angeschlossenen virtuellen Maschinen, nicht aber die Domain0.
Das öffentliche Interface des Login-Servers wird mit einer Firewall streng abgeschirmt, so dass es von außen nur noch über SSH und VPN zugänglich ist. Der Endpunkt des VPNs liegt auf der Netzwerkkarte `tap0`, welche zur Bridge `br0` im Login-Server gehört. Damit arbeitet `tap0` streng genommen nicht mehr als eigenständige Netzwerkkarte, sondern nur noch als Port der Bridge. Jedes eingehende Paket wird damit entweder über die Bridge und die Netzwerkkarte `eth1`, welche ebenfalls nur als Port der Bridge arbeitet, weiter zur Managementbridge `brmgnt` gleitet oder erreicht über `br0` den Login-Server selbst. Alle Aktivitäten setzen entsprechende Regeln der lokalen Firewall im Login-Server voraus, da auch Pakete, die von der Bridge, also auf OSI Schicht zwei, weitergeleitet werden die FORWARD Kette eines Paketfilters auf OSI Schicht drei durchlaufen. Bei Paketen, die für den Login-Server selbst bestimmt sind, wird die INPUT Kette durchlaufen, ausgehende Pakete werden entsprechend von OUTPUT behandelt. Weitere Details zur Architektur und Funktionsweise der implementierten Firewalls geben die Kapitel 4.3.2 und 4.1.2.
Die Netzwerkkarte `eth2` im Login-Server dient lediglich dazu, den `Secserver` an das externe Netz anzubinden. Da der `Secserver` in den Szenarien 1 (Abb. 3.1 Seite 30) und 2 (Abb. 3.2 Seite 31) logisch gesehen einen eigenen Zugang zum Internet besitzt, muss das Vorhandensein der Netzwerkkarte `eth2` im Login-Server verborgen werden. Dies ist nur möglich, wenn das Routing von `eth2` und `br0` unterbunden wird. Um den `Secserver` mit einer privaten Adresse betreiben zu können, wird hier SNAT (Source Network Address Translation) eingesetzt, bei dem die Quelladresse des zu verschickenden Pakets im Login-Server auf die Adresse des Login-Servers geändert wird. Bei den Antwortpaketen wird der Vorgang entsprechend umgekehrt und die Zieladresse von der Adresse des Login-Servers auf die private Adresse des `Secservers` geändert.
Um den Zugriff auf den in der Steuerdomain laufenden Webserver möglich zu machen, sind zusätzliche Maßnahmen nötig. Eine Möglichkeit besteht darin, den Webserver im privaten Netz der Steuerdomain, also im privaten Netz des Lehrstuhls anzubinden. Dies würde die Architektur des Netzwerks unverändert lassen, scheitert jedoch daran, dass die Studenten keinen Zugang zum internen Netz des Lehrstuhls haben und auch nicht bekommen sollen. Eine elegante Alternative ist es daher, einer ausgewählten Bridge innerhalb der Domain0 eine IP-Adresse zu zuweisen. Die Bridge arbeitet dadurch zusätzlich zu ihrer Funktionsweise als Bridge auch als Netzwerkinterface der Domain0, über die der Webserver für das VMMT erreichbar ist. Um das Vorhandensein der Domain0 weiter verschleiern zu können, wird als Bridge `brint` ausgewählt. Diese wird von zwei den Studenten nicht zugänglichen Rechnern flankiert und kann am einfachsten mit Firewalls geschützt werden. Zusätzlich wird im Login-Server ein Apache Webserver installiert, der ausschließlich auf der Schnittstelle `br0` erreichbar ist. Der Webserver stellt selbst keine Daten zur Verfügung sondern arbeitet im Wesentlichen als Proxy. Er leitet alle ankommenden Anfragen an den Webserver der Steuerdomain weiter. Der Webserver mit dem Virtual Maschine Management Tool scheint dadurch auf dem Login-Server zu laufen. Das Vorhandensein einer Steuerdomain bleibt weiterhin transparent und der Webserver ist auch im Falle abgestürzter Arbeitsrechner zu deren Steuerung erreichbar, da der Endpunkt des VPNs und die Schnittstelle, auf der der Proxy arbeitet, auf der selben Bridge liegen und die Erreichbarkeit des Webservers auch bei nicht arbeitenden Clients gewährleistet ist. Der Zugriff über einen SSH-Tunnel auf das VMMT funktioniert ebenfalls.

4 Realisierung

Zur Erstellung der beiden Topologien (Abb. 3.1 Seite 30 und Abb. 3.2 Seite 31) ist weit mehr nötig als das bloße Erzeugen vieler einzelner Maschinen. Um ein komplexes Netzwerk mit all seinen Eigenschaften nachzubilden und gleichzeitig den zusätzlich Anforderungen, die durch die Virtualisierung entstehen, nachzukommen, ist neben dem Erstellen der Netzinfrastruktur und deren Konfiguration auch auf die Trennung von virtueller Umgebung und physischer Realisierung zu achten. Dies beginnt bei der bereits vorgestellten Architektur des Systems (Abschnitt 3.6), betrifft auch den Einsatz von Firewalls (Abschnitt 4.3.2) und endet bei der Vergabe entsprechender Zugriffsrechte auf einzelnen Komponenten (Abschnitt 4.2.1). Letztere können sowohl physischer oder virtueller Natur sein. Dieses Kapitel wird im Folgenden alle Punkte erläutern, die für eine Realisierung der Szenarien mit all den geforderten Eigenschaften notwendig sind.

4.1 Topologien

Bereits beim Erstellen der virtuellen Maschinen und Szenarien zeigt sich, dass die Realisierung des Vorhabens nicht Schritt für Schritt nach den oben genannten Gliederungspunkten erfolgen kann, da zwischen den Komponenten Abhängigkeiten bestehen. Dem zufolge sind auch die Pläne für die Architektur, wie sie im vorhergehenden Kapitel zum Teil schon vorgestellt wurden, in einer Art Wasserfallmodell entstanden. In einen anfänglichen Plan wurden Stück für Stück die einzelnen Komponenten integriert, aneinander angepasst und so lange überarbeitet, bis das Modell realisierbar schien und alle Anforderungen erfüllt waren. Ein Beispiel für die Verzahnung der einzelnen Komponenten ist die Erstellung eines Managementnetzes, die bereits bei der Erstellung der virtuellen Maschinen mit einer zusätzlichen Netzwerkkarte bedacht werden muss sowie das Erzeugen zusätzlicher Bridges für die Infrastruktur und die Einführung eines virtuellen Login-Servers zur Beibehaltung der Trennung von realen und virtuellen Maschinen (Abschnitt 4.3.1). Als erstes widmen wir uns aber der Erstellung der einzelnen virtuellen Maschinen.

4.1.1 Erstellen virtueller Maschinen

Da alle Rechner über unterschiedliche Hardwareausstattung verfügen, beziehungsweise bei identischer Hardware doch wenigstens an unterschiedliche Bridges angeschlossen sind und verschiedene Konfigurationen zum Beispiel im Bereich des Managementnetzes erforderlich sind, ist es nicht möglich eine statische Konfigurationsdatei für die Konfiguration aller Rechner anzugeben. Da die Konfigurationen der `pcsec`-Rechner aber doch sehr ähnlich sind, können die Werte der einzelnen Schalter in den Konfigurationsdateien (vgl. B.18, B.19) mit Hilfe von benutzerdefinierten Variablen dynamisch berechnet werden (siehe auch 3.4.3). Lediglich für die Rechner `secplogin`, `secserver`, `test4all` und `hacktest` werden eigene Konfigurationen benötigt. Die Konfigurationen selbst werden jeweils in einer eigenen Datei abgespeichert, die den Namen des jeweiligen Rechners trägt. Die wichtigsten Einstellungen in der Konfigurationsdatei sind die Werte

- `kernel` gibt den zu verwendenden Xen-Linux Kernel an.
  Beispiel: `kernel = /boot/linux-2.6.11.12-xenU`
- `memory` gibt die Größe des Arbeitsspeichers in MB an.
  Beispiel: `memory = 70`
- `name` gibt den Namen der Domain an, nicht zu verwechseln mit `hostname`!
- `nics` gibt die Anzahl der zu erstellenden Netzwerkkarten an.
- `vif` definiert die Netzwerkkarten genauer. `vif` ist vom Typ `Python dict`. Das folgende Beispiel erzeugt zwei Netzwerkkarten mit angegebener MAC Adresse, verbunden mit der gewünschten Bridge.

Weitere Angaben sind möglich und können der Referenz entnommen werden. Werden nicht alle Werte angegeben, werden Standardwerte gesetzt. Beispiel:
`vif = ['mac=00:11:22:33:44:55:66, bridge=myBridge1',`
`'mac=11:22:33:44:55:66:77, bridge=myBridge2']`

- `disk` beschreibt die zu erzeugenden IDE und SATA Geräte. Beispiel:
  `disk = ['file:/home/images/myImage.img,sda1,w',`
  `'file:/home/SuSE/SuSE-9.3.iso,hdc,r']`
  erzeugt die Partition `sda1` mit Inhalt des Images `myImage.img` und erlaubt schreibenden Zugriff, sowie das schreibgeschützte Laufwerk `hdc` mit der SuSE-DVD als Inhalt.
- `ip` gibt die IP-Adresse von `eth0` an.
  Beispiel: `ip= '192.168.0.1'`
- `netmask` gibt die Subnetzmaske von `eth0` an.
  Beispiel: `netmask = '255.255.255.0'`
- `hostname` gibt den Rechnernamen an.
  Beispiel: `hostname = 'PC01'`
- `root` gibt analog zum normalen Start eines Linuxsystems das Root-Device an.
  Beispiel: `root = '/dev/sda1'`
- `extra` enthält Argumente, die dem Kernel direkt beim Start übergeben werden.
  Beispiel: `extra= '5'`
  (Bootet in den Runlevel 5)

Für die Funktionsweise der mit Xen erzeugten Konfiguration von `eth0` ist es erforderlich, dass `eth0` innerhalb der startenden Maschine nicht konfiguriert ist. Eventuell vorhandene Konfigurationen im startenden Linux überschreiben die hier gesetzten Werte. Der Start einer Maschine erfolgt mit

`xm create` *`Konfigurationsdatei`*

Die Konfigurationsdateien der einzelnen Rechner befinden sich im Anhang B, Listings B.15, B.16, B.17, B.18, B.19.

Die dynamischen Konfigurationen der Clients werden mit Hilfe von Python im Konfigurationsskript berechnet. Hierzu dient die übergebene Variable „vmid", die die Nummer des zu erzeugenden Rechners als String enthält. Das Kommando

```
xm create pcsec vmid=8
```

würde also den Rechner `pcsec08` erzeugen. Da die Konfigurationen abhängig vom gewählten Szenario sind und der Übersicht halber auf eine zweite benutzerdefinierte Variable zur Unterscheidung der Szenarien verzichtet wird, sind zwei verschiedene Konfigurationen nötig. Auch sie sind im Anhang zu finden (Listings B.18, B.19).

4.1.2 Erstellen der Szenarien

Das Starten und Stoppen der Szenarien erfordert neben dem Starten und Stoppen der einzelnen Rechner eine ganze Reihe weiterer Aktionen. Um diese einfach und bequem Ausführen zu können, bieten sich Skripte an. Die Startskripte (siehe Listings B.20, B.22) erzeugen zunächst die benötigten Bridges und konfigurieren diese dem Szenario entsprechend. Anschließend werden szenarioabhängige Dateien in die Images der virtuellen Maschinen kopiert. Hierbei handelt es sich um die Konfigurationsdateien des Nameservers im `Secserver` sowie die Routing-Informationen und Netzwerkkonfiguration für die Rechner `secserver`, `test4all` und `hacktest`.

Nachdem alle Rechner gestartet worden sind, werden die Firewallregeln erstellt, die nötig sind um den Betrieb der Bridges zu erlauben. Da ein Datenpaket, das von der Bridge auf Ethernetebene weitergeleitet wird, trotzdem für den Paketfilter `iptables` sichtbar ist und dessen `FORWARD` Queue durchläuft, müssen Regeln der Art

`iptables -A FORWARD -i` *`bridge`* `-o` *`bridge`* `-j ACCEPT`

erzeugt werden, um die Weiterleitung von Datenpaketen, die auf ein und derselben Bridge eintreffen und ausgehen, zu erlauben. Den exakten Weg eines Paketes durch `iptables` und `ebtables` in Link und Network Layer veranschaulicht die Grafik 4.2.

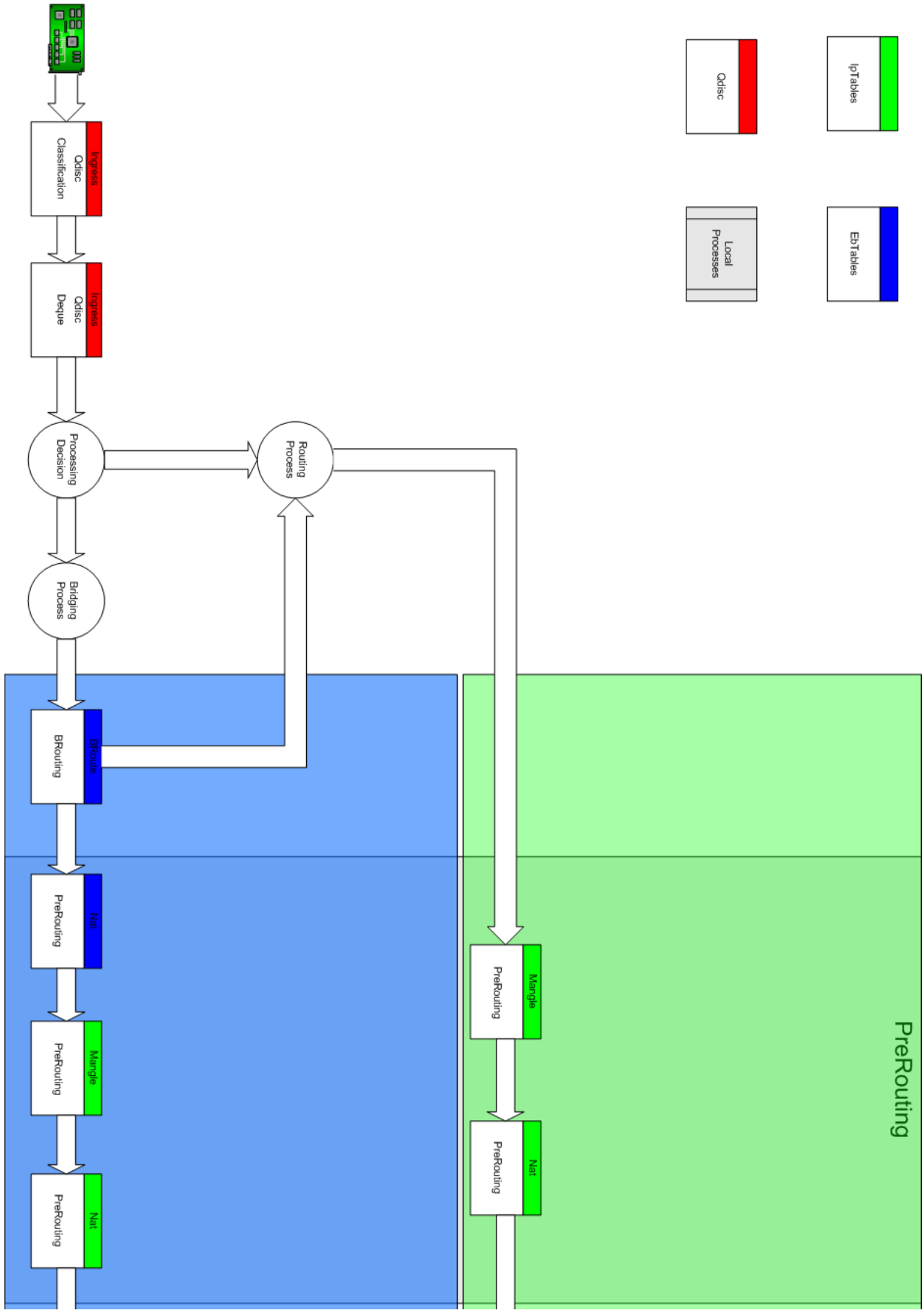

Abbildung 4.1: Wege eines Datenpaketes bei Routing und Bridging: PreRouting

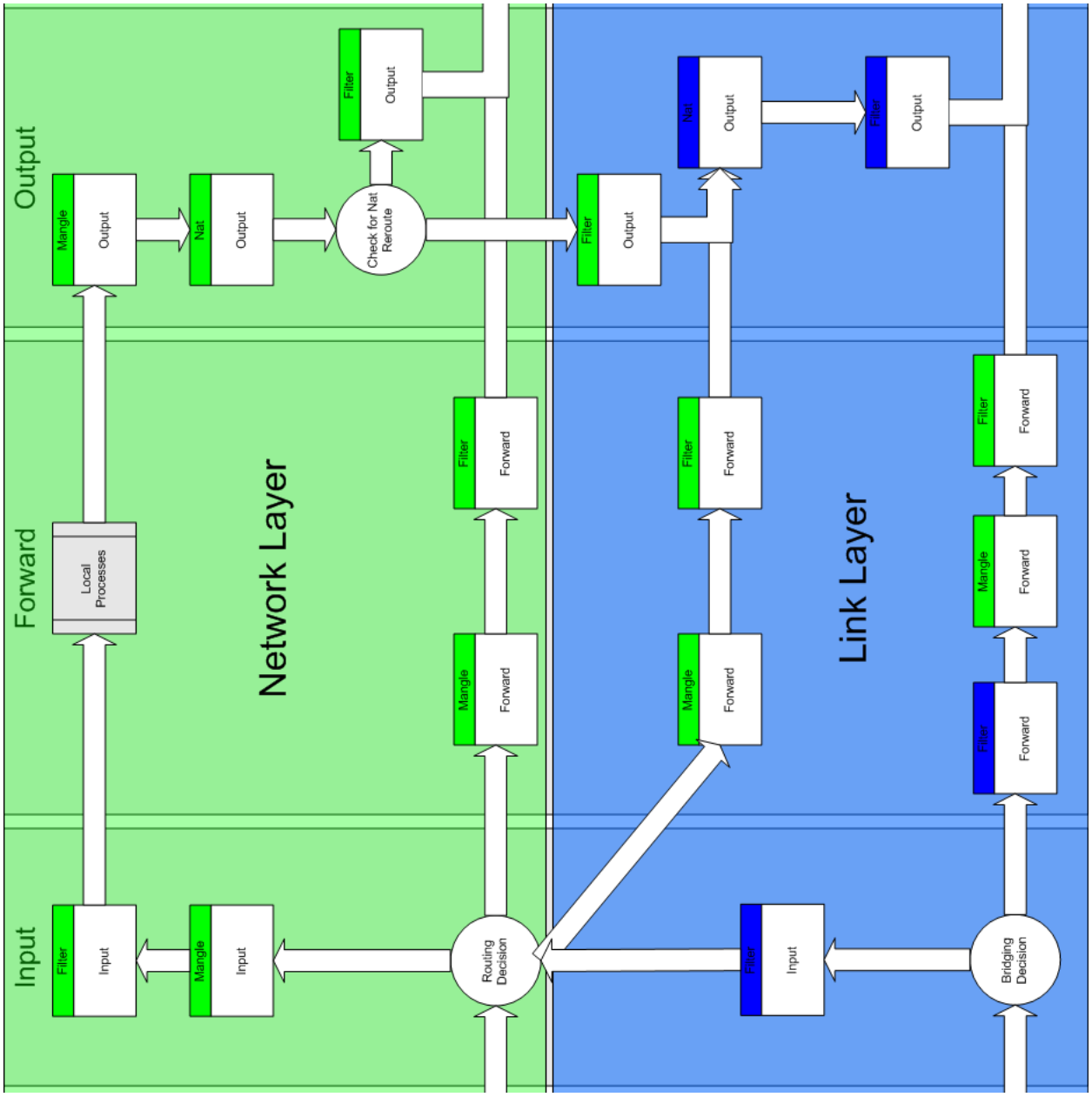

Abbildung 4.2: Wege eines Datenpaketes bei Routing und Bridging: Routing [Snyd 06]

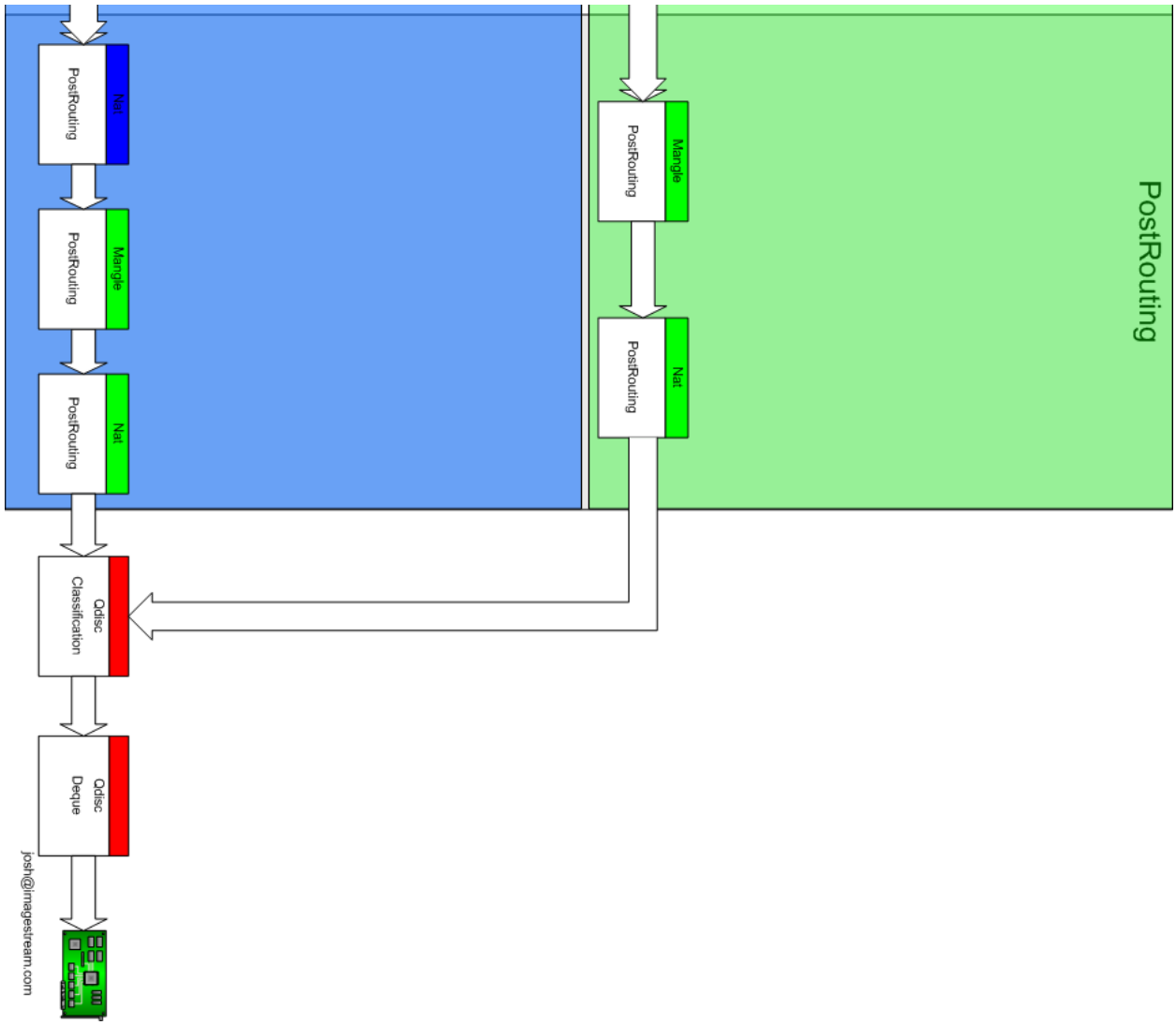

Abbildung 4.3: Wege eines Datenpaketes bei Routing und Bridging: PostRouting

Die für iptables sichtbare Schnittstelle ist dabei immer die Bridge, niemals die darauf verbundenen Netzwerkkarten. Soll die Entscheidung, ob ein Paket weitergeleitet wird oder nicht von den verbundenen Netzwerkkarten, also vom Port der Bridge, abhängen auf dem ein Paket eintrifft oder ausgeht, so muss das Modul physdev verwendet werden. Mit Hilfe des physdev-Moduls kann der Port der Bridge bestimmt werden, auf dem ein Paket das System erreicht hat beziehungsweise das System verlassen wird. Der Einsatz von physdev vereinfacht die folgende Aufgabe erheblich: Den Clients soll es nicht erlaubt sein über das Managementnetzwerk miteinander zu kommunizieren. Das Managementnetz soll lediglich dazu dienen, die Rechner über das VPN zu konfigurieren. Eine Möglichkeit dies zu realisieren wäre der Einsatz von Firewalls auf wie ebtables. Mit ihnen können Datenpakete anhand von Quell- und Ziel-MAC-Adresse gefiltert werden. Da leider die MAC-Adressen der VPN-Clients nicht bekannt sind und sich die MAC-Adressen der pcsec-Rechner unter Linux mit nur einem einzigen Kommando ändern lassen, ist diese Maßnahme zum einen sehr aufwändig zum anderen beinahe wirkungslos. Einfacher und auch nicht umgehbar ist der Einsatz des physdev-Moduls. Die beiden Anweisungen

```
iptables -A FORWARD -m physdev
          --physdev-in port
          --physdev-out ! port
          -j ACCEPT

iptables -A FORWARD -m physdev
          --physdev-in ! port
          --physdev-out port
          -j ACCEPT
```

in der Firewall-Konfiguration der Domain0 erlauben nur den Verkehr, der von den Netzwerkports der pcsec Rechner kommt und die Bridge über den Netzwerkport des Login-Servers wieder verlässt, sowie umgekehrt. Dies hat sogar noch den Vorteil, dass sich die VPN-Clients untereinander nicht direkt erreichen können und damit voreinander geschützt sind. In der angegebenen Regel steht Port ersatzweise für den Namen der Netzwerkschnittstelle, die mit eth1 des Login-Servers verbunden ist. Leider kann der Name des Ports nicht statisch bestimmt werden, sondern muss zur Laufzeit ermittelt werden, da er die ID des Login-Servers enthält, welche erst beim Start des Login-Servers festgelegt wird. Für das Erstellen der Szenarios bedeutet dies keinen großen Aufwand, da die ID mittels xm domid einfach erfragt werden kann. Eine potenzielle Fehlerquelle ergibt sich jedoch, wenn der Login-Server während der Aktivität eines Szenarios gestoppt und neu gestartet wird. Beim Start weist Xen der virtuellen Maschine eine neue ID zu, was eine Anpassung der Firewallregeln nötig macht. Das Virtual Maschine Management Tool berücksichtigt diesen Fall und ändert die Konfiguration entsprechend, bei der Benutzung von xm create muss dieser Schritt allerdings manuell erfolgen.
Für den erfolgreichen Start der Szenarien ist es unbedingt erforderlich ausreichend dynamische IRQs zur Verfügung zu haben. Der Standardwert an verfügbaren dynamischen IRQs liegt bei 128. Da jede virtuelle Netzwerkkarte in einer virtuellen Maschine jedoch einen Gegenspieler im Backend hat und dieser Adapter einen dynamischen IRQ benötigt, reicht dieser Wert nicht aus um alle Adapter zu versorgen. Eine Erhöhung dieses Wertes ist nicht ohne weiteres möglich, da die Anzahl aller IRQs in einem i386 System auf 256 begrenzt ist und die restlichen 128 IRQs physikalische sind. Um den Wert der dynamischen IRQs zu erhöhen, muss daher jeweils gleichzeitig die Anzahl der physikalischen IRQs entsprechend reduziert werden. Die Änderungen müssen in der Datei include/asm-xen/asm-i386/mach-xen/irq_vectors.h in den Kernelquellen des Xen0-Kernels vorgenommen werden. Anschließend muss der Kernel neu übersetzt werden.
Ein weiteres Problem stellt die Anzahl der verfügbaren Loop-Devices dar. Die Anzahl der Loop-Devices wurde beim Booten über den Parameter max_loop zwar schon erhöht, allerdings sind im Dateisystem nur 16 Device-Nodes angelegt, an die die Loop-Geräte gebunden werden können. Der Befehl

```
mknod -m660 /dev/loopX b 7 X
```

legt einen weiteren Knoten X an und weist ihm passende Berechtigungen zu. Dies muss für alle $X \in \{16..256\}$ durchgeführt werden, um alle 256 Knoten verfügbar zu machen.

4.2 Management

Für das Management der virtuellen Szenarien stehen zwei Tools zur Verfügung. Zum einen das Virtual Maschine Management Tool (vgl. Abschnitt 4.2.1), das alle gängigen Managementfunktionen, die die virtuellen Maschinen selbst betrifft, unterstützt und Dank eingebauter Authentisierung und Autorisierung sowohl Teilnehmern als auch Betreuern zur Verfügung steht. Für alle anderen Managementfunktionen kann ein SSH-Zugang benutzt werden. SSH steht allerdings nur Betreuern zur Verfügung und verlangt fundierte Kenntnisse in der Funktionsweise des Systems.

4.2.1 Virtual Maschine Management Tool

Das Virtual Maschine Management Tool (VMMT) ist eine mit Pythonskripten erstellte Webseite, auf die man über einen Apache Server, der in der Steuerdomain läuft, zugreifen kann (siehe Abb. 4.4). Zur Erstellung wurde das Modul mod_python verwendet, das es erlaubt serverseitige Pythonskripte auszuführen. Python ist die am besten für die Realisierung des Management Tools geeignete Sprache, da der Xen Daemon `xend` und `xm` selbst in dieser Sprache geschrieben sind und somit alle Schnittstellen zu Xen ohne Umwege realisierbar sind. Das VMMT besteht im Wesentlichen aus drei Dateien:

- index.psp
- request.py
- auth.py

`index.psp` ist für die Anzeige des aktuellen Status der einzelnen Rechner verantwortlich, `request.py` führt einzelne Anweisung aus und `auth.py` kümmert sich um die Autorisierung und Authentisierung. Je nach der Menge des generierten HTML-Codes eignen sich für die einzelnen Aufgaben unterschiedliche PythonHandler. Wird viel HTML ausgegeben sind Python Server Pages (.psp) die erste Wahl. PSP sind HTML Dateien in die ähnlich zu php einzelne Anweisungen oder Blöcke Python-Code eingefügt werden können. Diese werden vor der Auslieferung der Seite an den Browser serverseitig ausgeführt. Soll wenig oder garkeine Ausgabe erfolgen, bieten sich reine Python Dateien (.py) an. Da `index.psp` eine große Tabelle mit Statusinformationen ausgibt, wurde sie als PSP realisiert. `request.py` gibt nur Erfolgs- oder Misserfolgsmeldungen der durchgeführten Operationen aus und `auth.py` erzeugt gar keine Ausgabe, für beide wurde daher das .py-Format gewählt.
Die Dateien `auth.py` und `request.py` liegen im selben Verzeichnis. Beim Zugriff auf `request.py` durch den Server wird zunächst der AuthenHandler `auth.py` aufgerufen. `auth.py` kann je nach eingegebenem Nutzernamen und Passwort den Zugriff auf `request.py` erlauben und damit die Durchführung der Managementaktion gewähren, oder aber den Zugriff verweigern. Zur Autorisierung dienen als Nuztername der Name der zu administrierenden Maschine und als Passwort das zugehörige Root-Passwort. Alternativ kann auch der Benutzername ”root” in Verbindung mit dem Root-Passwort der Steuerdomain zur Steuerung aller Maschinen verwendet werden. Um die eingegebenen Passwörter mit den Passworthashes aus `/etc/shadow` der Images abgleichen zu können, ist es nötig eine Implementierung des verwendeten Hashalgorithmus unter Python verwenden zu können. Dies ist für den Standardalgorithmus DES (Data Encryption Standard) der Fall, nicht jedoch für den von SuSE verwendeten Blowfish-Algorithmus. Daher ist es wichtig, bei der Installation der Steuerdomain und der Erzeugung der Images diesen Algorithmus auf DES zu ändern.
Ein weiteres Problem stellen die für die meisten Managementaufgaben benötigten Rechte dar. Der Apache Server läuft in der Standardeinstellung mit sehr geringen Rechten, die zum Beispiel das Einhängen der Images der virtuellen Maschinen in das Dateisystem des Servers nicht erlauben. Da der Server auch nicht pauschal höhere Rechte erhalten soll, werden diese Befehle mit Hilfe von `sudo` ohne Eingabe eines Passwortes legitimiert.
Das Virtual Maschine Management Tool erlaubt es laufende Maschinen herunter zu fahren, zu zerstören, zu rebooten, zu pausieren und fortzusetzen. Für nicht aktive Maschinen kann ein Backup erstellt beziehungsweise ein Backup oder das ursprüngliche Image zurückgespielt werden. Natürlich können nicht aktive Maschinen auch gestartet werden. Alle Aktionen liefern über ein Popup-Fenster Statusmeldungen über Erfolg oder Misserfolg.

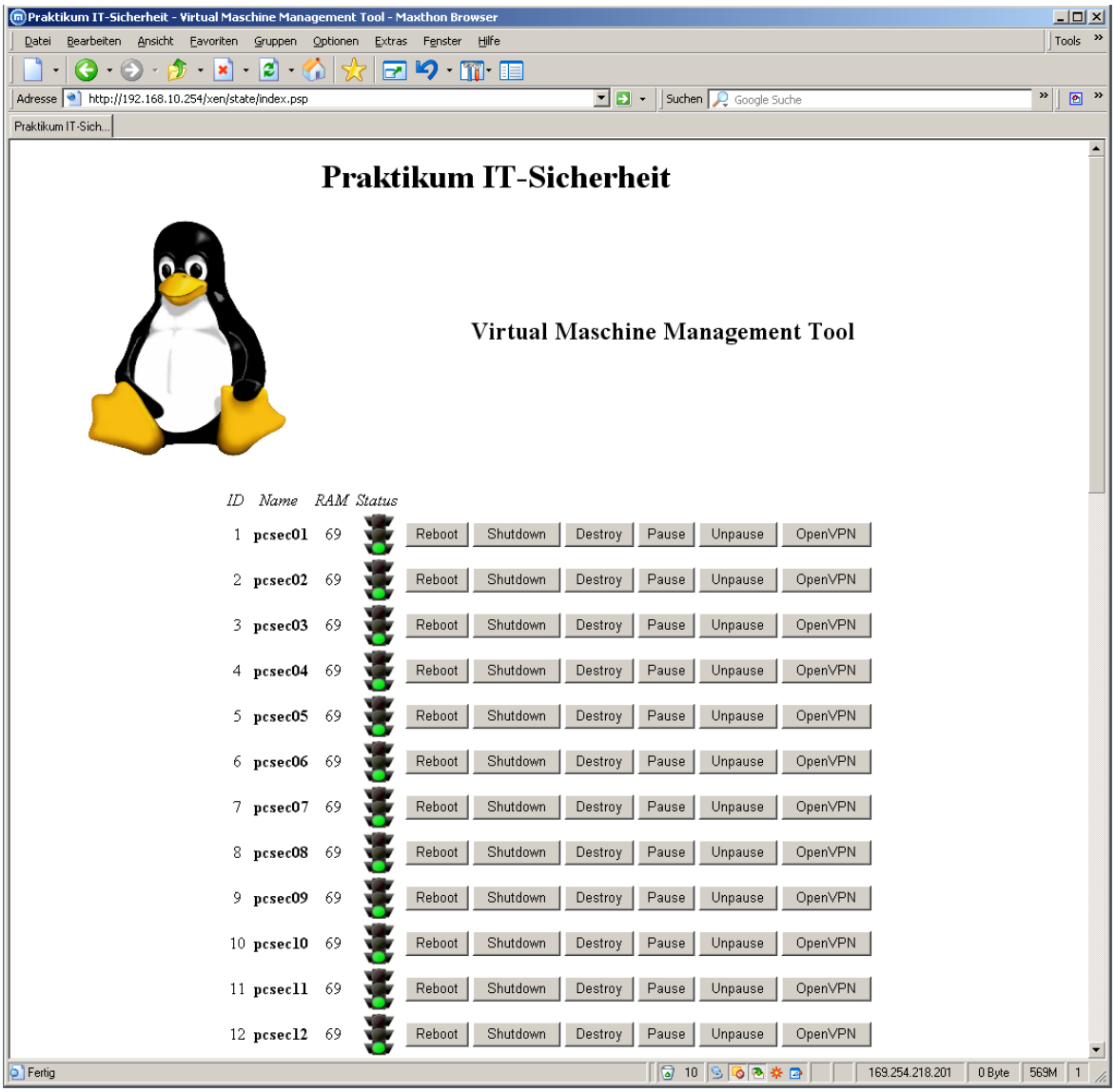

Abbildung 4.4: Virtual Maschine Management Tool

Da der Webserver des VMMT auf der Domain0 läuft, diese für die Teilnehmer aber nicht zugänglich ist, muss eine Art transparenter Proxy auf dem Login-Server realisiert werden, um diesen Umstand zu verbergen und den Webserver erreichbar zu machen. Ursprünglich sollte die Erreichbarkeit des VMMT aus dem Praktikumsnetz mit Hilfe von DNAT im Login-Server erreicht werden. Von den virtuellen Arbeitsrechnern oder den VPN-Clients wäre dies auch problemlos möglich gewesen. Durch die Unterstützung von Zugriffen auf das VMMT über SSH-Tunnel ist dieses Vorgehen nicht mehr möglich. Der Grund hierfür ist, dass DNAT von iptables im PREROUTING durchgeführt wird (siehe auch Abb. 4.2). PREROUTIG ist eine der ersten Aktionen, die ausgeführt wird, wenn ein Paket das System betritt. DNAT funktioniert daher nur, wenn ein Datenpaket die INPUT- oder FORWARD-Queue der Firewall durchläuft. Bei OpenVPN ist das der Fall. Die Pakete betreten das System logisch gesehen über die Bridge `br0`, genauer gesagt über deren Port `tap0` und durchlaufen daher den PREROUTING-Schritt. Beim Einsatz von SSH-Tunneln funktioniert das Vorgehen dagegen nicht. Die Pakete treffen hier zunächst auf `eth0` via SSH ein und werden vom SSH-Prozess verarbeitet. Der SSH-Prozess erzeugt aus den Paketen selbstständig die weiterzuleitenden Pakete, indem er den Inhalt der SSH-Pakete entpackt. Diese verlassen das System anschließend über die OUTPUT-Queue. Da es sich bei der Zieladresse `192.168.10.254` um eine lokale Schnittstelle handelt, wird der Vorgang intern über das Loopback-Interface durchgeführt, welches mit den Paketen nichts anfangen kann und sie verwirft. Der komplette Weg eines Paketes von externen Rechnern zum VMMT unter Verwendung von Adressumsetzung wird von Abbildung 4.5 dargestellt. Nicht relevante Firewalldurchläufe sind zur besseren Lesbarkeit der Grafik nicht eingezeichnet.

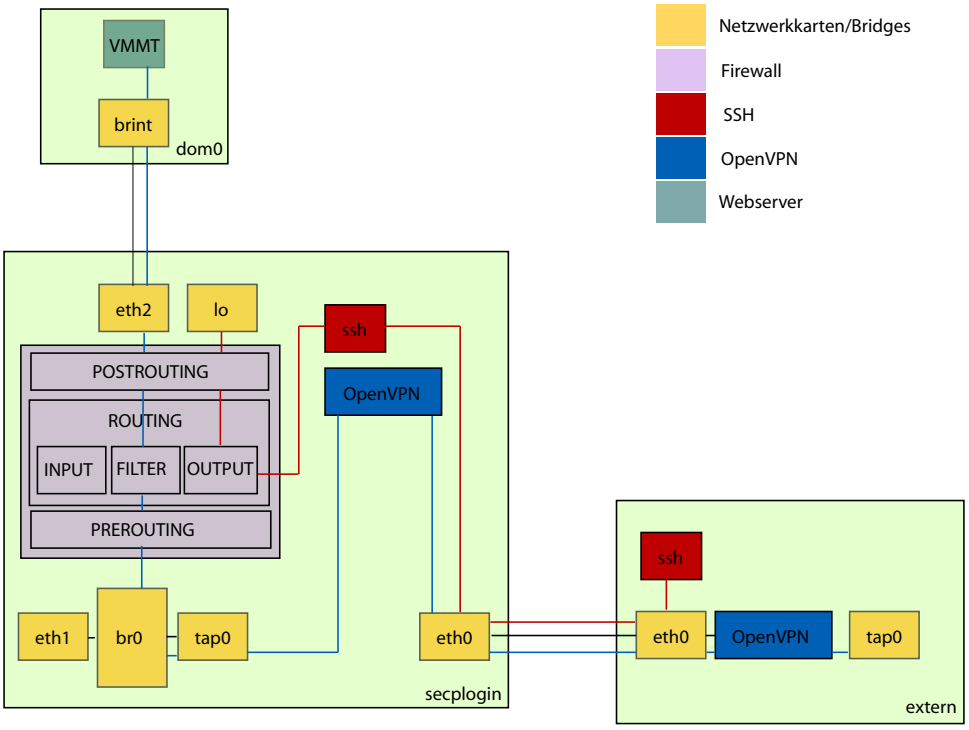

Abbildung 4.5: Zugriffe auf das VMMT via NAT

Eine Alternative den Webserver für das Praktikumsnetz zugänglich zu machen, ist der Einsatz eines transparenten Proxys, wie er mit Apache und dem Modul mod_proxy möglich ist. Der Webserver leitet damit Zugriffe auf bestimmte Verzeichnisse an andere Server weiter und schickt deren Antwort an den Client zurück. Entsprechend eingerichtet kann auch jeder Zugriff weitergeleitet werden, indem der Proxy für das Root-Verzeichnis des Servers aktiviert wird. Für den Client ist der Vorgang transparent. Im Browser steht immer noch die eingegebene Adresse, obwohl die Kommunikation eigentlich mit einem anderen Webserver abläuft. Der für den

Client sichtbare Webserver arbeitet hierbei lediglich als transparenter Proxy. Abbildung 4.6 zeigt den Weg eines Paketes von einem externen Rechner zum VMMT unter Zuhilfenahme von mod_proxy.

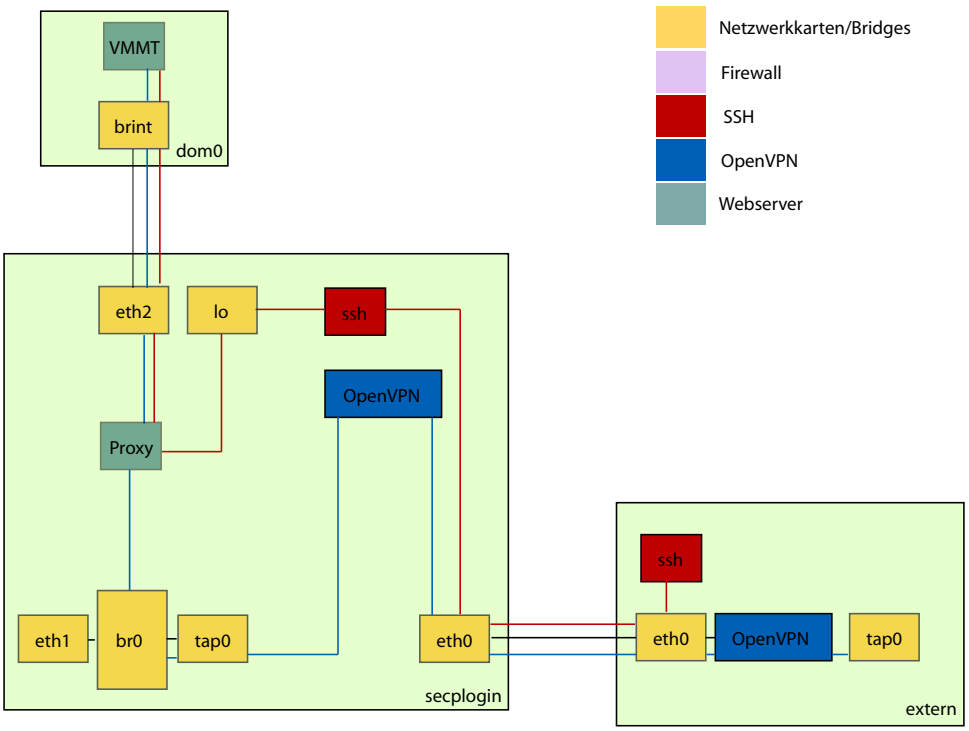

Abbildung 4.6: Zugriffe auf das VMMT mit dem Apache Modul mod_proxy

4.2.2 Secure Shell

Für Wartungsarbeiten an den einzelnen virtuellen Maschinen oder an der Steuerdomain selbst steht ein SSH-Zugang bereit, der entweder über die Steuerdomain oder den Login-Server erfolgt. Der Zugang zur Domain0 ist Betreuern vorbehalten. Wird der Zugang über den Login-Server gewählt, so besteht wie geplant keine Möglichkeit auf die Domain0 einzuwirken.
Zum Erstellen oder Beenden von Szenarien stehen im Verzeichnis `/home/config` der Steuerdomain Start- und Stopp-Skripte bereit. Zur manuellen Administration der Maschinen auf der Konsole dient das `xm` Kommando. Bei Problemen im Bereich des Netzwerks helfen die Programme `ifconfig`, `route` und `brctl`.

4.3 Sicherheit

Dieser Abschnitt soll die Sicherheit der Realisierungen gegenüber unbefugten Zugriffen und aktiven Angriffen zeigen. Dazu betrachten wir sowohl die Architektur des Systems als Ganzes als auch einzelne Komponenten. Ein separater Abschnitt widmet sich dem Thema Firewalls.

4.3.1 Trennung von physischen und virtuellen Komponenten

Bei der Architektur wurde streng darauf geachtet, virtuelles Netz und die zur Realisierung benötigten physischen Komponenten zu trennen. Es darf keinem Teilnehmer des Praktikums gelingen aus dem virtuellen

Szenario auszubrechen und Rechte in der Steuerdomain zu erlangen. Das Erlangen von Rechten in der Steuerdomain ist auf mehreren Wegen vorstellbar. Möglichkeiten hierzu bieten alle Verbindungen die virtuelle Maschinen zu Xen und Domain0 besitzen. Dies reicht vom Backend/Frontend-Treibersystem zur Bereitstellung von virtuellen Netzwerkkarten und Speichermedien bis zur Kompromittierung fremder Speicherbereiche über Bugs im Xen-Kernel sowie Angriffe über physische Netzwerkkarten. Gegen Angriffe, die auf Implementierungsfehlern in Xen selbst beruhen, kann in diesem Fall leider nichts unternommen werden, außer bekannt gewordene Schwachstellen zu schließen. Der Betrieb mit geringeren Rechten, wie dies bei dem Apache Server realisiert wurde (vgl. Abschnitt 4.2.1), ist mit einem Hypervisor, der zwangsläufig die privilegierteste Komponente sein muss, nicht möglich. Es bleiben also nur unerlaubte Zugriffe über Netzwerkschnittstellen und Bridges, die es zu verhindern gilt. Generell kann eine Kommunikation über IP nur mit Geräten erfolgen, welche auch eine IP-Adresse besitzen. Daher wurde auf die Vergabe von IP-Adressen wo möglich verzichtet. Ausgenommen hiervon sind die Bridge `brint`, sowie die Schnittstellen `eth0` in Domain0 und Login-Server. Für diese Ausnahmen gelten daher umso restriktivere Firewallregeln. Die Kommunikation über Ethernet ist grundsätzlich mit allen Bridges und Netzwerkkarten möglich, kann aber ebenfalls über Firewalls auf das nötigste begrenzt werden, da auch beim Einsatz von Bridges die Regeln einer Paketfilterfirewall greifen. Näheres illustriert Grafik 4.2.
Alle den Teilnehmern zugänglichen Bereiche der Architektur sind ausschließlich virtuell. Die einzige Schnittstelle des virtuellen Netzes nach außen, ist die physische Netzwerkkarte `eth0` des Login-Servers. Diese wird aber von einer virtuellen Maschine (dem Login-Server) kontrolliert und bietet somit keinen Zugang zur physischen Plattform. Die physische Plattform ist nur über eine separate Netzwerkkarte, die im privaten Subnetz des Lehrstuhls liegt, erreichbar. Die einzigen Schnittstellen zwischen virtuellem Netzwerk und Virtualisierungsplattform sind die Schnittstellen des Hypervisors zur Verwaltung der virtuellen Maschinen. Durch sie ist aber lediglich einseitige Kommunikation möglich. Das heißt, Managementaktionen können grundsätzlich nur von der Domain0 angestoßen werden. Eine Kontaktaufnahme seitens der virtuellen Maschinen ist nicht möglich. Um die Steuerung von einzelnen Maschinen aus dem virtuellen Netz zu ermöglichen, musste eine Möglichkeit geschaffen werden, dieses eigentlich wünschenswerte Verhalten kontrolliert zu umgehen. Die Realisierung der Steuerung der virtuellen Maschinen aus dem Managementnetz erfolgt über das Virtual Maschine Management Tool (vgl. Abschnitt 4.2.1). Die dazu notwendige Kommunikation läuft über die Bridge `brint` und das Protokoll HTTP. Möglich ist nur der Aufruf ausgewählter Funktionen mit vorausgehender Autorisierung. Der dafür notwendige Kanal zur Kommunikation wird durch den Einsatz von Firewalls so schmal wie möglich gehalten und mit allen verfügbaren Mitteln abgeschirmt und versteckt (siehe auch Abschnitt 3.7).

4.3.2 Firewall

Zur Absicherung der Szenarien und der darunter liegenden Hardware sind insgesamt drei Firewalls notwendig. Eine zum Schutz der Domain0 (Listing 4.1), eine für den Login-Server (Listing 4.2), die neben Schutz vor Angriffen auch NAT-Funktionen bieten muss, sowie eine zur Absicherung des `Secservers` (Listing 4.3). Alle implementierten Firewalls verwerfen in der Standardeinstellung alle nicht ausdrücklich erlaubten Datenpakete. Die Firewalls wurden als dynamisch Paketfilter-Firewalls mit `iptables` unter Verwendung des `state` Moduls realisiert. Ausgehende Verbindungen sind auf nahezu allen Interfaces erlaubt, eingehende Verbindungen werden nur für spezielle Dienste auf ausgewählten Schnittstellen zugelassen.
Das Weiterleiten von Paketen ist ebenfalls auf die notwendigen Netzwerkkarten und Bridges beschränkt. Die Firewall der Steuerdomain unterbindet in der Voreinstellung grundsätzlich jedes Routing. Die benötigten Routingregeln für das Weiterleiten innerhalb der Bridges werden beim Starten eines Szenarios vom Startskript passend erzeugt. (Siehe hierzu auch Abschnitt 4.1.2)
Eine zentrale Rolle nimmt die Bridge `brint` ein. Sie ist die einzige Bridge mit IP-Adresse und ist somit eine aktive Verbindung zur Steuerdomain. Ihre Absicherung ist mehrstufig gewährleistet. Zum einen ist die Bridge nur über die ebenfalls abgesicherten Maschinen `secserver` und `secplogin` erreichbar, zum anderen ist sie innerhalb der Domain0 durch eine Firewall abgesichert, die ausschließlich Kommunikation über Port 80 (HTTP) mit Quelladresse `192.168.20.254` erlaubt.
Die Firewall des Login-Servers bietet zusätzlich zu ihrer reinen Filterfunktion auch NAT-Funktionalität an. Für weiterzuleitende Pakete, die auf `eth2` eingehen und über `eth0` in das externe Netz weitergeleitet werden sollen, kommt SNAT zum Einsatz, da die privaten Adressen des Praktikumnetzes außerhalb weder bekannt sind, noch geroutet werden. Auf diese Weise ist die Bridge nicht nur vor direkten Zugriffen geschützt, sondern es wird ebenfalls die Architektur des Systems verschleiert. Der Webserver scheint logisch auf `eth1` des

Login-Servers zu laufen.
Die Firewall des Login-Servers verhindert externe Angriffe auf das virtuelle Netz, indem nur die TCP Ports 22 (SSH) und 1194 (OpenVPN) für eingehenden Verkehr geöffnet sind. Nicht einmal ICMP Anfragen werden beantwortet. Dies schützt die Maschine vor Portscans aus dem Internet, da die meisten Portscanner vor einem Scan das Vorhandensein eines Rechners mit ICMP Nachrichten sicherstellen. Bleibt die Antwort auf einen `icmp-request` aus, geht der Angreifer in der Regel davon aus, dass die Maschine nicht existiert und sucht sich ein anderes Opfer.

Listing 4.1: Firewall der Domain0

```
#! /bin/sh

##############################
# Firewall-Skript für Domain0 #
##############################

# Default: Alles verweigern

iptables -P INPUT DROP
iptables -P OUTPUT DROP
iptables -P FORWARD DROP

# Bereits existierende Regeln löschen
iptables -F
iptables -X

# Neue Pakete, die keine Verbindung aufbauen verwerfen
iptables -A INPUT -p tcp ! --syn -m state --state NEW -j LOG
iptables -A OUTPUT -p tcp ! --syn -m state --state NEW -j LOG
iptables -A INPUT -p tcp ! --syn -m state --state NEW -j DROP
iptables -A OUTPUT -p tcp ! --syn -m state --state NEW -j DROP

iptables -A INPUT -m state  --state ESTABLISHED,RELATED -j ACCEPT
iptables -A OUTPUT -m state  --state ESTABLISHED,RELATED -j ACCEPT


####################
# Freischaltregeln #
####################

# localhost

# lokal alles erlauben
iptables -A INPUT -i lo -j ACCEPT
iptables -A OUTPUT -o lo -j ACCEPT

# eth0

# Ausgehenden, lokal erzeugten Verkehr auf eth0 erlauben
iptables -A OUTPUT -o eth0 -j ACCEPT

# eingehend ident/auth TCP/113 auf REJECT setzen um Timeouts zu verhindern
iptables -A INPUT -i eth0 -p tcp --dport 113 -j REJECT

# eingehend ICMP auf eth0 erlauben
iptables -A INPUT -i eth0 -p icmp -j ACCEPT

# eingehend HTTP auf eth0 erlauben
iptables -A INPUT -i eth0 -p tcp --dport 80 -m state --state NEW -j ACCEPT
```

```
# eingehend SSH auf eth0 erlauben
iptables -A INPUT -i eth0 -p tcp --dport 22 -m state --state NEW -j ACCEPT

# Alle anderen Pakete loggen
iptables -A INPUT -j LOG
iptables -A OUTPUT -j LOG
iptables -A FORWARD -j LOG
```

Listing 4.2: Firewall des Login-Servers

```
#! /bin/sh

###################################
# Firewall-Skript für Login-Server #
###################################

# Default: Alles verweigern

iptables -P INPUT DROP
iptables -P OUTPUT DROP
iptables -P FORWARD DROP

# Bereits existierende Regeln löschen
iptables -F
iptables -X
iptables -Z

iptables -t nat -F
iptables -t nat -X
iptables -t nat -Z


# Neue Pakete, die keine Verbindung aufbauen verwerfen
iptables -A INPUT -p tcp ! --syn -m state --state NEW -j LOG
iptables -A OUTPUT -p tcp ! --syn -m state --state NEW -j LOG
iptables -A FORWARD -p tcp ! --syn -m state --state NEW -j LOG
iptables -A INPUT -p tcp ! --syn -m state --state NEW -j DROP
iptables -A OUTPUT -p tcp ! --syn -m state --state NEW -j DROP
iptables -A FORWARD -p tcp ! --syn -m state --state NEW -j DROP

iptables -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
iptables -A OUTPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
iptables -A FORWARD -m state --state ESTABLISHED,RELATED -j ACCEPT


####################
# Freischaltregeln #
####################

# localhost

# lokal alles erlauben
iptables -A INPUT -i lo -j ACCEPT
iptables -A OUTPUT -o lo -j ACCEPT

# eth0

# Ausgehenden, lokal erzeugten Verkehr auf eth0 erlauben
iptables -A OUTPUT -o eth0 -j ACCEPT

# eingehend SSH auf eth0 erlauben
```

```
iptables -A INPUT -i eth0 -p tcp --dport 22 -m state --state NEW -j ACCEPT

# eingehend VPN auf eth0 erlauben
iptables -A INPUT -i eth0 -p tcp --dport 1194 -m state --state NEW -j ACCEPT

# br0

# Ausgehenden, lokal erzeugten Verkehr auf br0 erlauben
iptables -A OUTPUT -o br0 -j ACCEPT

# eingehend ICMP erlauben
iptables -A INPUT -i br0 -p icmp -j ACCEPT

# eingehend SSH erlauben
iptables -A INPUT -i br0 -p tcp --dport 22 -m state --state NEW -j ACCEPT

# eingehend HTTP erlauben
iptables -A INPUT -i br0 -p tcp -d 192.168.10.254 --dport 80 -m state --state NEW
     -j ACCEPT

# Bridging auf br0 erlauben
iptables -A FORWARD -i br0 -o br0 -j ACCEPT

# eth2

# ausgehenden, lokal erzeugten Verkehr auf eth2 erlauben
iptables -A OUTPUT -o eth2 -j ACCEPT

# eingehend ICMP erlauben
iptables -A INPUT -i eth2 -s 192.168.20.0/24 -p icmp -j ACCEPT

# SNAT für Verbindungen vom Secserver aktivieren
iptables -A FORWARD -i eth2 -o eth0 -j ACCEPT
iptables -t nat -A POSTROUTING -o eth0 -j SNAT --to-source 141.84.218.191


# Alle anderen Pakete loggen
iptables -A INPUT -j LOG
iptables -A OUTPUT -j LOG
iptables -A FORWARD -j LOG
```

Listing 4.3: Firewall des Secservers

```
#! /bin/sh

################################
# Firewall-Skript für Secserver #
################################

# Default: Alles verweigern

iptables -P INPUT DROP
iptables -P OUTPUT DROP
iptables -P FORWARD DROP

# Bereits existierende Regeln löschen
iptables -F
iptables -X

# Neue Pakete, die keine Verbindung aufbauen verwerfen
iptables -A INPUT -p tcp ! --syn -m state --state NEW -j LOG
```

```
iptables -A OUTPUT -p tcp ! --syn -m state --state NEW -j LOG
iptables -A FORWARD -p tcp ! --syn -m state --state NEW -j LOG
iptables -A INPUT -p tcp ! --syn -m state --state NEW -j DROP
iptables -A OUTPUT -p tcp ! --syn -m state --state NEW -j DROP
iptables -A FORWARD -p tcp ! --syn -m state --state NEW -j DROP

iptables -A INPUT -m state  --state ESTABLISHED,RELATED -j ACCEPT
iptables -A OUTPUT -m state  --state ESTABLISHED,RELATED -j ACCEPT
iptables -A FORWARD -m state  --state ESTABLISHED,RELATED -j ACCEPT


####################
# Freischaltregeln #
####################

# localhost

# lokal alles erlauben
iptables -A INPUT -i lo -j ACCEPT
iptables -A OUTPUT -o lo -j ACCEPT

# eth0

# Ausgehenden, lokal erzeugten Verkehr auf eth0 erlauben
iptables -A OUTPUT -o eth0 -j ACCEPT

# eingehend ICMP auf eth0 erlauben
iptables -A INPUT -i eth0 -p icmp -j ACCEPT

# eingehend SSH auf eth0 erlauben
iptables -A INPUT -i eth0 -p tcp --dport 22 -m state --state NEW -j ACCEPT

# eth1

# Ausgehenden, lokal erzeugten Verkehr auf eth1 erlauben
iptables -A OUTPUT -o eth1 -j ACCEPT

# eingehend ICMP auf eth1 erlauben
iptables -A INPUT -i eth1 -p icmp -j ACCEPT

# eingehend DNS auf eth1 erlauben
iptables -A INPUT -i eth1 -p udp --dport 53 -j ACCEPT

# eingehend Portmapper erlauben
iptables -A INPUT -i eth1 -p tcp --dport 111 -m state --state NEW -j ACCEPT
iptables -A INPUT -i eth1 -p udp --dport 111 -j ACCEPT

# eingehend NFS erlauben
iptables -A INPUT -i eth1 -p tcp --dport 2049 -m state --state NEW -j ACCEPT
iptables -A INPUT -i eth1 -p udp --dport 2049 -j ACCEPT

# eingehend mountd erlauben
iptables -A INPUT -i eth1 -p tcp --dport 1002 -m state --state NEW -j ACCEPT
iptables -A INPUT -i eth1 -p udp --dport 1002 -j ACCEPT

# eingehend nlockmgr erlauben
iptables -A INPUT -i eth1 -p tcp --dport 1024 -m state --state NEW -j ACCEPT
iptables -A INPUT -i eth1 -p udp --dport 1026 -j ACCEPT

# Verkehr ins Internet von den Clients zum Login-Server weiterleiten
iptables -A FORWARD -i eth1 -j ACCEPT
```

```
# Alle anderen Pakete loggen
iptables -A INPUT -j LOG
iptables -A OUTPUT -j LOG
iptables -A FORWARD -j LOG
```

4.4 Tests

Bereits zum Beginn dieser Arbeit, noch vor der Entwurfsphase, wurden umfangreiche Tests mit Xen durchgeführt. Diese beschäftigten sich sowohl mit der Architektur von virtuellen Maschinen als auch mit deren Konfiguration. In einzelnen Versuchen wurde ausprobiert wie viel Arbeitsspeicher zum Betrieb der virtuellen Maschinen notwendig ist oder welche Art von Images - Loop-Device, native Partitionen, LVM, oder NFS - am performantesten arbeiten und am einfachsten zu handhaben sind. Stück für Stück wuchs so der Plan für die Architektur der beiden Szenarien. Doch nicht nur vor der Entwurfsphase wurde getestet, sondern auch während und nach der Realisierung. So sollte während der Entwicklung bereits Funktionalität, Performanz und Sicherheit des Vorhabens sichergestellt werden.

4.4.1 Funktionalität

Kriterien für den Punkt Funktionalität sind vor allem das Funktionieren des Managementnetzes sowie des Managementtools und die Möglichkeit alle im Praktikum behandelten Themen in der virtualisierten Umgebung zu behandeln. Erstere Kriterien wurden nach der Implementierung der einzelnen Komponenten getestet und funktionierten einwandfrei. Sowohl die Verbindung zum VPN-Gateway als auch die Verbindung über das VPN in das virtuelle Netz arbeiteten trotz ihrer Komplexität auf Anhieb. Auch wenn es theoretisch kein Problem darstellen durfte, so war es doch erstaunlich, dass die Einwahl in ein virtuelles Netzwerk über eine virtuelle Maschine als Gateway mit Hilfe eines Virtual Private Networks trotz so vieler virtueller Komponenten reibungslos funktionierte und auch noch sehr hohe Stabilität zeigte. Die VPN Verbindung überstand während der Entwicklung sogar Verbindungsabbrüche eines WLANs sowie DSL-Reconnects. Damit waren SSH-Verbindungen über das VPN zu den virtuellen Maschinen stabiler als die nativen SSH-Verbindung zum Serversystem. Auch die Steuerung der virtuellen Maschinen über das entwickelte Managementinterface zeigte am Ende nach einigen Schwierigkeiten während der Implementierung keine Schwächen mehr.
Die Eignung der virtuellen Umgebung zur Bearbeitung der Praktikumsaufgaben wurde wegen des dafür nötigen Zeitaufwands nicht vollständig getestet. Alle hardware- und kernelnahen Aufgaben, wie das Mithören des Netzwerkverkehrs an den Hubs oder die Implementierung von Firewalls mit den `iptables/netfilter` Modulen funktionierten ohne Probleme. Da sich die virtuellen Maschinen auf Anwendungsebene nicht anders verhalten als native Linux Systeme, sollten die Aufgaben, die auf reiner Anwendungssoftware basieren, kein Problem darstellen. Für endgültige Gewissheit bleibt ein erster Testlauf im Sommersemester 2006 abzuwarten.

4.4.2 Performanz

Die Performanz der Implementierung überzeugte bei einzelnen Tests, über ein Verhalten im Praxiseinsatz kann aber nur spekuliert werden. So dauerte das Erzeugen eines Szenarios mit Hilfe der Startskripte auf dem verwendeten Zweiprozessorsystem ca. 12 Minuten. Da alle Rechner gleichzeitig starten, erscheint die Dauer von 12 Minuten zum Hochfahren einer Maschine lange, umgerechnet benötigt jede Maschine zum Booten jedoch nur ungefähr 16 Sekunden. Mit dieser Zeit kann kein natives modernes Linuxsystem mit ähnlicher Funktionalität konkurrieren, zumal den einzelnen Maschinen im Durchschnitt nur ca. 80 MB Hauptspeicher zur Verfügung stehen, denn mehr als 4 GB Hauptspeicher können unter Xen 2.0.7, das mit 32-Bit arbeitet, nicht adressiert werden. Analog zu den Berichten auf der Xen Homepage (siehe Abb. 4.7) zeigt sich hier, dass durch die Architektur von Xen nur ein sehr geringer Zusatzaufwand bei der Behandlung von Systemaufrufen entsteht, der durch effiziente Auslastung des Systems durch Scheduling leicht kompensiert wird. Zu nativen Linux Systemen ist fast kein Unterschied zu bemerken, andere Virtualisierungslösungen dagegen schneiden zum Teil massiv schlechter ab. Wie hoch die Last des Systems im Praxiseinsatz wirklich ausfällt, lässt sich

schwer voraussagen. Dies hängt unter anderem stark von der Verteilung der Last über einen größeren Zeitraum ab. Wollen alle Studenten ihre Aufgaben zur selben Zeit bearbeiten, kann es zu Einbrüchen in der Performanz kommen. Da ein Großteil der Aufgaben lediglich die Anpassung von Konfigurationsdateien umfasst, dürfte die Last des Systems insgesamt nicht allzu hoch werden. Ein weiterer zu berücksichtigender Aspekt ist die Art der Managementverbindung. Die Verwendung von SSH ist logischerweise Ressourcen schonender als der Einsatz von NX, bei dem ein X-Server betrieben werden muss und sehr viele Daten übertragen werden müssen. Je nach Vorliebe der Studenten für die eine oder die andere Variante ergibt sich hier zusätzlich ein Unsicherheitsfaktor zur Abschätzung der Gesamtlast, von der die Performanz entscheidend abhängen wird. Gewissheit wird letztendlich auch nur der Einsatz in der Praxis bringen.

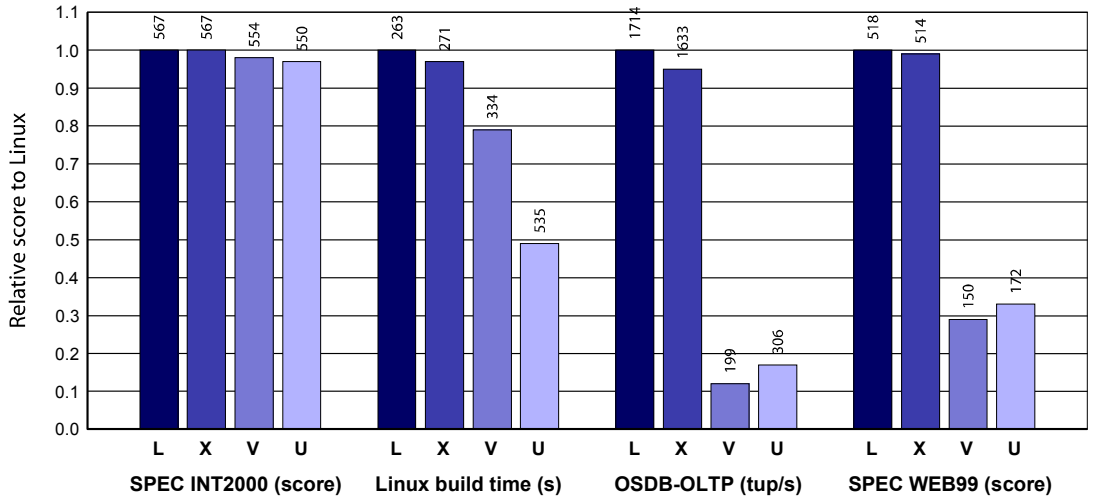

Abbildung 4.7: Vergleich der Leistungsfähigkeit verschiedener Virtualisierungslösungen [Prat 04]

4.4.3 Sicherheit

Um die Sicherheit der virtuellen Maschinen gewährleisten zu können, wurden alle eingesetzten Firewalls mit Hilfe von Portscannern getestet sowie alle auf den SuSE-Servern verfügbaren sicherheitsrelevanten Patches eingespielt. Die Architektur des Gesamtsystems wurde bereits von Anfang an so konzipiert, dass die Sicherheit der einzelnen Maschinen gegeben ist.

4.4.4 Stärken & Schwächen

In allen Phasen der Entwicklung ist die Stabilität von Xen positiv aufgefallen . Fehler, die in einer Maschine auftraten, beeinflussten nie das Verhalten einer anderen Maschine. Fehler von Xen während der Laufzeit wurden nie beobachtet. Auf der anderen Seite ist die Konfiguration der virtuellen Maschinen, deren Dateisysteme, sowie die Vernetzung der virtuellen Maschinen sehr zeitaufwendig. Es traten während der gesamten Implementierung immer wieder Fehler auf, deren Ursache zunächst unklar war. Erst durch tagelange Internetrecherchen und viel Ausprobieren konnten sie lokalisiert und behoben werden. Oft schien es, die auftretenden Fehler würden die Realisierung der virtuellen Szenarien unmöglich machen. Einer der schlimmsten auftretenden Fehler war der Mangel an dynamischen IRQs innerhalb der Steuerdomain (siehe Abschnitt 4.1.2). Das Starten der 39. virtuellen Maschine führte ohne Angabe von Fehlermeldungen sofort zum Absturz des kompletten Systems, egal in welcher Reihenfolge die Maschinen gestartet wurden. Erst ein Eintrag in das Xen Bugzilla brachte durch Zufall die Lösung, da der unmittelbar benachbarte Bug in der Suchliste die selbe Ursache und das gleiche Symptom zeigte aber in einer anderen Ausgangssituation auftrat. Für den benachbarten

Bug existierte ein Workaround, der empfahl die IRQ Verteilung anzupassen. Ein auf gut Glück durchgeführter Versuch hatte auch tatsächlich Erfolg und löste das Problem. Dieses Beispiel ist jedoch nur eines unter vielen. Nahezu jede der erarbeiteten Einstellungen mussten mit viel Ausprobieren gefunden werden, da keine Anleitungen für Szenarien dieser Größenordnung existieren.
Eine andere Schwachstelle ist das Fensterhandling des Webservers, das erst nach Ausführung einer Managementaktion Statusmeldungen erzeugt. Im Falle von Backup- oder Restore-Operationen geschieht dies oft erst 10 Minuten nach Anstoßen der Operation, ohne die Anzeige eines Fortschritts der laufenden Aktion. Die Implementierung eines eleganten Fensterhandlings innerhalb dieser Arbeit ist leider nicht möglich, da sie tiefgehende Kenntnisse in der Erstellung dynamischer Webseiten erfordert und somit den Zeitrahmen dieser Diplomarbeit deutlich sprengen würde. Diese Arbeit ist jedoch ohne allzu großen Aufwand zur Einarbeitung in Xen und das System realisierbar. Eine Möglichkeit eine komfortablere Oberfläche für das VMMT zu implementieren wäre daher zum Beispiel im Rahmen eines weiteren Fortgeschrittenen Praktikums möglich.

5 Zusammenfassung & Ausblick

Zusammenfassend bleibt festzustellen, dass mit moderner Virtualisierungssoftware wie Xen und leistungsfähiger Hardware mittlerweile sehr große und komplexe Realisierungen möglich sind. Die zusätzlichen Entwicklungen auf diesem Gebiet, wie die Entwicklung von Prozessor gestützter Virtualisierung, zeigen, dass der Bedarf an Entwicklung von Virtualisierungsprodukten besteht. Für die Zukunft sind weitere Hardware basierte Produkte zur Unterstützung von Virtualisierungstechnologien bereits angekündigt. Die momentan vielversprechenste Entwicklung sind Grafikkarten mit Virtualisierungsunterstützung. Damit ist es möglich die Ausgabe jeder virtuellen Maschine eines Systems ohne Einsatz von spezieller Software wie VNC, NX oder RDP auf dem Monitor zu betrachten. Dies war bisher ein Privileg der Domain0.
Trotz der Fortschritte im Bereich der Hard- und Software, existiert zur Zeit leider noch zu wenig tief gehende Dokumentation, die ein erfolgreiches Arbeiten mit den Produkten wesentlich vereinfachen würde. Das für Xen existierende Manual ermöglicht zwar den Einstieg in die Arbeit mit Xen, enthält aber leider nur Beschreibungen für die notwendigsten Einstellungen. Da auch die Ausgaben von `--help` der `xm`-Kommandos recht knapp ausfallen, sind weitergehende Konfigurationen sehr mühsam oder gar nicht möglich, obwohl die Software längst nicht optimal ausgenutzt wird.
Dem entsprechend mühsam gestaltete sich die Implementierung der virtuellen Szenarien mit Xen. Auf der anderen Seite zeigt diese Implementierung das enorme Potential, das in Virtualisierungssoftware, insbesondere Xen, steckt. Als alle größeren technischen Probleme ausgeräumt waren, liefen alle Komponenten beinahe beunruhigend stabil. Zum Ende der Implementierung hatte das System mit aktivem Szenario eine Up-Time von über drei Wochen, ohne Probleme zu verursachen, obwohl in dieser Zeit immer wieder kleinere Konfigurationsänderungen gemacht werden mussten.
Die Machbarkeit von Virtualisierungen dieser Größenordnung mit verhältnismäßig einfacher Hardware und akzeptabler Performanz zeigt wieder einmal, wie viel Rechenleistung in modernen PCs steckt und in vielen Fällen brach liegt. Virtualisierung ist aus diesem Grund auch für die Zukunft eine nicht zu vernachlässigende Technologie, was Wirtschaftlichkeit betrifft. Durch Xen und andere Virtualisierungsprodukte ist es möglich, ungenutzte Rechenkapazität in Servern zu nutzen, welche zuvor brach lag, weil Sicherheitsanforderungen es verboten, mehrere wichtige Dienste auf dem selben physischen Rechner auszuführen.
Trotz der immer wieder auftretenden gravierenden Schwierigkeiten in der Implementierungsphase, war die Bearbeitung dieser Aufgabe immer interessant, da das Ziel mit der Virtualisierung der Szenarien im Praktikum IT-Sicherheit zum einen klar definiert war, auf der anderen Seite aber sehr großen Freiraum für eigene Ideen beinhaltete. Zudem umfasste die Aufgabe neben der reinen Virtualisierung ein sehr großes Spektrum an zusätzlichen Aufgaben, angefangen beim Entwurf und der Planung von größeren Netzwerken über das Berücksichtigen von Sicherheitsaspekten im Aufbau und der Implementierung von Firewalls bis hin zur Administration von Linux-Systemen, der Programmierung von Managementtools mit Python und VPN-Technik. Dass das Zusammenspiel so vieler Komponenten nicht von Anfang an reibungslos funktionierte, erstaunt daher wenig. Exakt dieser Punkt zeichnet Informatik in der Praxis jedoch aus. Sie wäre als alleinstehende Wissenschaft nie entstanden. Informatik war und ist immer die Hilfswissenschaft, die es anderen Wissenschaften ermöglicht Probleme zu lösen. Das Lösen von Problemen im Bereich der EDV ist daher nicht nur Hauptaufgabe fast aller Informatiker, sondern die Berechtigungsgrundlage für ihre Existenz. Ohne dem Auftreten von Problemen während der Implementierung, hätte die Aufgabe einen großen Teil ihres Reizes verloren, denn erst durch die unzählbaren Versuche entstand das Verständnis für die Details der Thematik. Dieses Wissen kann kein noch so gutes Buch, Tutorial oder die Ausarbeitung dieser Diplomarbeit ersetzen. Ein indianische Sprichwort sagt: „*Tell me, I will forget. Show me, I may remember. Involve me, I will understand.*" Exakt so trifft es auch hier zu, denn wären detaillierte Anleitungen zur Realisierung großer virtueller Netzwerke verfügbar, dann wäre deren Implementierung zwar Arbeit, könnte aber im Prinzip von jedem durchgeführt werden, ohne dass dieser danach wüsste was er eigentlich getan hat.
Trotz der erfolgreichen Implementierung der Szenarien bleiben Ansatzpunkte für Verbesserungen oder Erweiterungen. Bereits angesprochen wurde in diesem Zusammenhang die Entwicklung einer komfortableren Management Oberfläche in Abschnitt 4.4.4, welche im Rahmen eines Fortgeschrittenen-Praktikums imple-

mentiert werden könnte. Ein anderer Punkt ist die Migration des Projektes auf Xen 3, wenn ein Probedurchgang im Sommersemester 2006 erfolgreich verlaufen ist. Mit der Verwendung von Xen 3 dürften sich einige Geschwindigkeitsvorteile für das System ergeben. Zum einen unterstützt Xen 3 sowohl prozessorgestützte Virtualisierung durch Vanderpool und Pacifica, zum anderen die Adressierung von mehr Arbeitsspeicher durch den Hypervisor. Beide Varianten erfordern allerdings die Anschaffung neuer beziehungsweise zusätzlicher Hardwarekomponenten. Eventuell sind bis zum Ende des Testlaufs auch weitere Technologien, wie CoW-Dateisysteme für Xen, welche sich zur Zeit noch in der Entwicklungsphase befinden verfügbar. So ließe sich sehr viel Festplattenplatz für Images und Backups sparen und der Zugriff auf die Images erheblich beschleunigen, da Teile der Images im Arbeitsspeicher gehalten werden könnten. Der Einsatz dieser Technologien dürfte allerdings nicht mit einer einfachen Migration erledigt sein. Die Implementierung dieser Änderungen greift an vielen Stellen so tief in das System ein, dass aus der Migration sehr schnell eine Neuimplementierung werden könnte.

A Installation

Ziel dieses Anhangs ist es, die Installation und Konfiguration aller für die Virtualisierung benötigter Komponenten in einer Schritt für Schritt Anleitung zu geben. Diese soll zum einen das Verständnis über den Aufbau der einzelnen Komponenten vertiefen und zum anderen dabei helfen, im Falle eines Falles Fehler zu beheben oder eine Migration auf andere Hardware vorzunehmen.

A.1 Installation des Betriebssystems

Als Betriebssystem kommt SuSE 9.3 Professional zum Einsatz. Der Einsatz von SuSE 10.0 mit Xen 2.07 ist aufgrund einiger zu neuer, systemnaher Bibliotheken nicht möglich! Die Server-Hardware ist ein Siemens CELSIUS V810 und besteht aus einem Zweiprozessor AMD Opteron System mit 4GB RAM. Das System besitzt vier SATA-Festplatten der Größe 160 GB. Diese könnten vom internen RAID-Controller verwaltet werden, sollen aber aus Sicherheitsgründen als Software-RAID konfiguriert werden, da der RAID-Controller selbst auf der Hauptplatine sitzt und im Falle eines Defektes nicht ausgetauscht werden kann. Die Daten auf den Festplatten wären damit, obwohl sie wahrscheinlich intakt sind, hoffnungslos verloren. SuSE 9.3 wird trotz der 64-Bit fähigen Opteron Prozessoren im 32-Bit Modus installiert, da das verwendete Xen 2.0.7 noch keine 64-Bit Systeme unterstützt.
Beim Start von der SuSE 9.3 DVD wählt man im Dialog des Bootloaders Grub als erstes die gewünschte Sprache und als Modus 32-Bit. Anschließend startet man den Menüpunkt „Installation". Die Standardvorgaben des Installationsprogrammes YaST sind in den meisten Fällen in Ordnung, lediglich die Konfiguration der Festplatten und die zu installierenden Softwarekomponenten werden von Hand angepasst. Dazu wählt man in der Übersicht nacheinander die Punkte „Partitionierung" - „Partitionen nach eigenen Vorstellungen anlegen" - „Erweiterte Einstellungen, manuelle Aufteilung (Partitionierung)". Im so genannten „Expertenmodus" werden auf den Festplatten `sda` und `sdb` über die Schaltfläche „Anlegen" je eine 10 GB große primäre Partition des Typs `0xFD Linux Raid` und eine den Rest der Platten füllende primäre Partition des selben Typs angelegt. Die Festplatten `sdc` und `sdd` erhalten jeweils eine primäre Partition der vollen Größe, ebenfalls vom Typ `0xFD Linux Raid`. Anschließend legt man nacheinander für je zwei Partitionen der selben Größe über „RAID" - „RAID anlegen..." drei RAID 1 Partitionen an und formatiert diese mit dem Dateisystem ext3. Die entstehende 10 GB große logische Partition ist für das Betriebssystem bestimmt und erhält daher den Mountpunkt „`/`" zugewiesen. Von den beiden anderen logischen Partitionen mountet man die kleinere nach „`/home`" und die größere nach „`/home/images`".
Die Auswahl der Softwarekomponenten ändert man über den Menüpunkt „Software-Auswahl". Dort wählt man den Punkt „Standard-System mit KDE" und anschließend die Schaltfläche „Erweiterte Auswahl. Neben den vorgewählten Komponenten installiert man zusätzlich „C/C++ Compiler und Werkzeuge" sowie folgende einzelne Pakete:

- curl
- curl-devel
- zlib
- zlib-devel
- iproute2
- bridge-utils
- phyton
- python-devel

- python-twisted
- apache2-mod_python
- apache2

Eventuell abhängige Pakete gibt man bei Nachfrage zur Installation frei.

Achtung: Die Installation von Xen via YaST darf nicht durchgeführt werden, da es sich um eine SuSE spezifische Variante handelt und diese nicht auf dem letzten Stand der Version 2.0.7 ist.

Der Rest der Installation verläuft wie gewohnt. Nach erfolgtem Neustart kann die Installation noch den eigenen Bedürfnissen angepasst werden. Wichtig sind vor allem eine funktionierende Netzwerkkonfiguration sowie die Wahl von DES als Hashalgorithmus bei der Authentisierung.
Da die beiliegende SuSE-Firewall auf Xen ohne IPv6 Unterstützung des Kernels Fehler produziert, wird sie durch eine eigene Firewall [4.1] ersetzt. Das Firewallskript wird dazu nach /etc/init.d/ kopiert. Über den YaST-Runlevel Editor kann anschließend die systemeigene Firewall deaktiviert und die eigene aktiviert werden.

A.2 Installation von Xen

Zum Einsatz kommt Xen 2.0.7, da Xen 3.0 zu Beginn dieser Arbeit noch sehr instabil war. Zur Installation dieser Version auf SuSE 9.3 benötigt man ein Installationspaket von Xen. Dieses gibt es wahlweise als Quelltextarchiv oder als Binaries. Da für die Anpassung der Xen-Linux Kernel an die Szenarien die Kernel passend konfiguriert werden müssen, benötigen wir zwangsweise die Quelltextvariante. Der Link auf dieses Archiv wurde allerdings mit dem Erscheinen der Xen Version 3 von der Xen Webseite [Univ 05a] gelöscht.
Um Xen zu installieren, entpackt man zunächst mit

```
tar zxvf xen.2.0.7-src.tgz
```

das Archiv und führt anschließend im Verzeichnis xen-2.0 das Kommando

```
make world
```

aus. Damit wird Xen kompiliert sowie die Quellen eines Linuxkernels heruntergeladen, für Xen gepatched und ebenfalls übersetzt. Ein anschließendes

```
make install
```

installiert Xen und kopiert je einen Xen0- und einen XenU-Kernel nach `/boot`.
Um das System mit dem Xen0 Kernel starten zu können, muss man folgenden Eintrag in die Konfigurationsdatei von Grub `/boot/grub/menu.1st` schreiben:

```
title Xen 2.0.7 auf SuSE 9.3
kernel (hd0,0)/boot/xen-2.0.7.gz
        dom0_mem=262144
        physdev_dom0_hide=(02:09.0)
module (hd0,0)/boot/vmlinuz-2.6.11.12-xen0
        root=/dev/md0 ro
        console=tty0
        max_loop=256
```

Die erste Zeile enthält nach dem Schlüsselwort `title` die Bezeichnung des Menüpunktes, wie sie im Bootmenü erscheint.
`kernel (hd0,0)/boot/xen-2.0.7.gz` lädt den Hypervisor aus der Datei `/boot/xen-2.0.7.gz` der ersten Partition auf der ersten Festplatte. Der Parameter `dom0_mem` weist der Kontrolldomain Domain0 256 MB Arbeitsspeicher zu. Die Angabe erfolgt in kb. Der Rest des Arbeitsspeichers ist somit für die übrigen virtuellen Maschinen reserviert. `physdev_dom0_hide` versteckt ein Gerät mit angegebener PCI-Adresse vor der Kontrolldomain. Das Gerät kann später von einer anderen virtuellen Maschine verwendet werden. Näheres dazu erläutert Abschnitt A.4.
Die letzte Zeile startet den Linux Kernel der Kontrolldomain, anders als üblich aber eingeleitet mit dem Schlüsselwort `module`, da der Hypervisor unter Xen das Betriebssystem repräsentiert und damit das Schlüsselwort `kernel` schon belegt ist.
Vor dem ersten Start benennt man noch das Verzeichnis `/lib/tls` in `/lib/tls.disabled` um. Damit wird das `thread lokal storage` deaktiviert. Dies ist nötig, weil die entsprechenden Bibliotheken nicht virtualisierungstauglich sind und zu einer erheblichen Verlangsamung des Systems führen können, wenn viele Programme, die mehrere Threads erzeugen, ausgeführt werden. Diese Programme laufen nach der Deaktivierung von `tls` ohne Probleme, da automatisch ein Fallback auf eine ältere Variante der Threadbehandlung durchgeführt wird. Damit ist das System bereit für den ersten Start. Da in der Kernelkonfiguration der Xen0- und XenU-Kernel aber einige notwendige Punkte deaktiviert sind, empfiehlt es sich, diese unter dem nativen Linux zu konfigurieren und kompilieren, da diese Aufgabe mit 4GB RAM und Zweiprozessorunterstützung wesentlich performanter durchführbar ist als in einem virtualisierten Einprozessorsystem mit 256MB RAM. Dazu wechselt man im Verzeichnis `xen-2.0` in das Unterverzeichnis `linux-2.6.11-xen0` und startet dort zur Konfiguration des Xen0-Kernels

```
make menuconfig ARCH=xen
```

Im Konfigurationsdialog aktiviert man unter Device Drivers - Networking Support - Networking Options - Network packet filtering (replaces ipchains) - IP: Netfilter Configuration folgende Punkte um alle benötigten Komponenten zur Konfiguration der Firewalls zu aktivieren:

- Connection tracking (required for masq/NAT)
- FTP protocol support
- IP tables support (required for filtering/masq/NAT)
- IP range match support
- Connection state match support
- Physdev match support
- Packet filtering
- REJECT target support
- LOG target support
- FULL NAT
- MASQUERADE target support
- REDIRECT target support
- Packet mangling

Zum Betrieb des VPNs im Login-Server benötigt man ein TAP-Device. Dies aktiviert man unter Device Drivers - Networking Support:

- Universal TUN/TAP device driver support

Um Unterstützung für Network File Systems und SMB zu aktivieren, wählt man unter Filesystems - Network File Systems die folgenden Punkte:

- NFS filesystem support

- Provide NFSv3 client support
- Provide NFSv4 client support (EXPERIMENTAL)
- NFS server support
- Provide NFSv3 server support
- Provide NFSv4 server support (EXPERIMENTAL)
- SMB file system support (to mount Windows shares etc.)

Falls möglich wählt man $< M >$ für Modul um den gewünschten Punkt zu aktivieren. Einige Komponenten lassen sich jedoch nicht als Modul einbinden, diese werden mit $< * >$ fest in den Kernel integriert. Die selben Änderungen nimmt man auch am XenU-Kernel vor. Lediglich TUN/TAP support sowie NFS server support sind dort nicht notwendig und können weggelassen werden. Anschließend werden die Kernel jeweils mit

```
make ARCH=xen
```

übersetzt und mit

```
make install ARCH=xen
```

installiert.

```
make modules_install ARCH=xen
```

installiert die Module für einen Kernel.

A.3 Erstellen der Rootdateisysteme

Alle Dateisysteme werden in Imagedateien bereitgestellt. Da sich die Inhalte der Images für die einzelnen virtuellen Rechner nur minimal unterscheiden, erstellen wir zunächst ein fertiges Image, das anschließend kopiert und angepasst werden kann. Als erstes erstellt man mit

```
dd if=/dev/zero of=/home/images/default.img ms=1M count=3328
```

eine 3,25 GB große, leere Datei. In ihr legt man mit Hilfe von

```
mkfs -t ext3 /home/images/default.img
```

ein ext3 Dateisystem an und mountet es mit

```
mount -o loop /home/images/default.img /home/images/mnt_default
```

in ein zuvor angelegtes Verzeichnis `mnt_default`. Nun erzeugt man mit Hilfe von YaST und dem Menüpunkt „Installation in Verzeichnis" in diesem Verzeichnis ein Standard SuSE System mit KDE. Folgende Komponenten sollten jedoch nicht installiert werden:

- Kernel und Kernel-Module
- Bootloader, wie z.B. Grub

Kernel und Module haben wir bereits selber erzeugt, ein Bootloader ist von der Architektur her nicht erforderlich. Zusätzlich installieren wir das Packet `FreeNX`, einen freier NX-Server. „YaST und SuSEconfig nach dem Systemstart ausführen" aktiviert man.

Nach der Installation verschiebt man im Zielort wie auch schon im nativen System das tls Verzeichnis mit

```
mv lib/tls lib/tls.disabled
```

Anschließend kopiert man das Verzeichnis `/var/adm/YaST` aus der Domain0 mit samt allen Unterverzeichnissen an die entsprechende Stelle im Image.

```
cp -r /var/adm/YaST /home/images/mnt_default/var/adm/
```

Ohne diesen Schritt fehlen YaST die nötigen Informationen zur Version von SuSE Linux und es können keine Online Updates durchgeführt werden. Zu guter Letzt kopiert man noch die erzeugten Module mit

```
cp -r /lib/modules/2611.12-xenU /home/images/mnt_default/lib/modules/
```

in das Dateisystem. Alle weiteren Konfigurationen werden am laufenden System durchgeführt. Daher muss das Image nun mit

```
umount /home/images/default.img
```

aus dem Dateisystem ausgehängt werden. Um eine virtuelle Maschine starten zu können, muss als erstes der Xen Daemon `xend` mit

```
xend start
```

gestartet werden. Als nächstes benötigen wir eine Standardkonfiguration für die virtuelle Setup-Maschine. Diese könnte folgendermaßen aussehen und in `/home/config/default` gespeichert sein:

```
kernel='/boot/vmlinuz-2.6.12.11-xenU'
memory=128
name='Setup Domain'
disk=['file:/home/images/default.img,sda1,w']
vif=['bridge=xen-br0']
root='/dev/sda1 ro'
extra='5'
```

Der Befehl

```
xm create -c /home/config/default
```

startet die Maschine und verbindet die aufrufende Konsole mit der startenden Domain. Dort läuft als erstes ein initiales Setup durch, in dessen Verlauf man wie bei der Installation des Betriebssystems auf dem Server, den Hashalgorithmus zur Speicherung der Passwörter auf DES setzt. Während des notwendigen Neustarts schlagen einige hardwarenahe Dienste fehl. Diese sind aber in der Regel für ein laufendes Xen-System nicht notwendig und scheitern nur deshalb, weil die entsprechende Hardware für die virtuelle Maschine nicht sichtbar ist. Um die Fehlermeldungen zu beseitigen, löscht man mit Hilfe von YaST im Expertenmodus des Runlevel-Editors folgende Einträge aus allen Runleveln:

- SuSEfirewall2_init
- SuSEfirewall2_setup
- earlykbd
- kbd

- boot.clock
- boot.device-mapper
- powersaved

Dabei müssen unter Umständen andere, abhängige Dienste ebenfalls deaktiviert werden. Als nächsten Schritt konfigurieren wir die für das online Update benötigte Netzwerkverbindung. Da das Interface gebridged ist, das heißt im selben Subnetz liegt wie die physikalische Netzwerkkarte der Domain0, weist man dem virtuellen Interface eine Adresse aus dem Subnetz der Steuerdomain zu. Defaultgateway- und Nameservereinstellungen übernimmt man von der Steuerdomain. Bei funktionierender Verbindung kann das YaST Online Update, kurz YOU, gestartet werden. Installiert werden alle Updates, die sicherheitsrelevant und daher schon automatisch ausgewählt sind. Nach erfolgtem Update kann die Maschine wieder heruntergefahren werden. Da während des ersten Starts der virtuellen Maschine automatisch Schlüssel für SSH erzeugt wurden, diese aber in allen Maschinen unterschiedlich sein müssen, müssen diese nun wieder aus dem Image gelöscht werden. Dazu mounten wir das Image wieder und löschen alle Dateien im Verzeichnis `etc/ssh/` die mit `ssh` beginnen mit dem Kommando

```
rm /home/images/mnt_server/etc/ssh/ssh*
```

Anschließend kann das Image wieder ausgehängt und pro virtuellem Rechner eine Kopie angelegt werden.

A.4 Konfiguration des Login-Servers

Der Login-Server unterscheidet sich von allen anderen virtuellen Maschinen dadurch, dass er selbst über eine physische Netzwerkkarte verfügt. Daher muss er, wie die Steuerdomain auch, mit einem privilegierten Xen0-Kernel ausgeführt werden. Die XenU-Module im Image des Login-Servers müssen gegen die des Xen0-Kernels ausgetauscht werden. Um der Maschine die physische Netzwerkkarte zugänglich zu machen, muss die Variable

```
pci=['02,09,0']
```

in der Konfigurationsdatei des Login-Servers für `xm create` wie angegeben belegt werden. Die notwendigen Parameter für die Adressierung der Karte liefert der Befehl `lspci`.

Achtung: Der Befehl muss im normalen SuSE Linux ausgeführt werden, da die Netzwerkkarte in unserem Xen-System schon nicht mehr sichtbar ist!

Nach dem Start des Login-Servers werden zunächst die Netzwerkkarten konfiguriert. Dazu kopiert man die Datei `/etc/sysconfig/network/ifcfg.template` nach `ifcfg-eth0`, `ifcfg-eth1` und `ifcfg-eth2`. Den Wert „`STARTMODE`" setzt man in den erzeugten Kopien auf `'auto'`, um die Konfiguration beim Starten der Maschine automatisch zu laden. Dieser Schritt ist notwendig, da YaST die virtuellen Netzwerkkarten nicht automatisch erkennt. Nach diesem Schritt können die Netzwerkkarten entweder per Hand oder wie gewohnt per YaST mit folgenden Werten konfiguriert werden:

- eth0
  - IP-Adresse: 141.84.218.144
  - Netzmaske: 255.255.255.128
- eth1
  - IP-Adresse: 192.168.10.254
  - Netzmaske: 255.255.255.0

- eth2
  - IP-Adresse: 192.168.20.254
  - Netzmaske: 255.255.255.0
- Routing & DNS
  - Defaultgateway: 141.84.218.254
  - Route: 192.168.216.0/24 gw 192.168.20.1
  - DNS: 129.187.214.135
  - Routing: enabled
  - Hostname: login
  - Domain: nm.ifi.lmu.de

Anschließend wird mit Hilfe von YaST das Paket OpenVPN installiert. OpenVPN benötigt zur Komprimierung der übertragenen Datenpakete das Paket lzo, welches automatisch installiert wird. Die Konfiguration [B.1] von OpenVPN muss wie im Anhang gezeigt angepasst werden, um den OpenVPN-Server auf dem TCP-Port 1194 der Netzwerkkarte `eth0` zu starten.
Zum Betrieb müssen die in der Konfigurationsdatei angegebenen Schlüssel und Zertifikate müssen noch erstellt werden. Dazu kopiert man den Inhalt des Verzeichnisses `/usr/share/doc/packages/openvpn` nach `/etc/openvpn` und wechselt in dieses Verzeichnis. Im Unterverzeichnis `easy-rsa` trägt man zunächst einige persönliche Daten wie etwa Standort des Servers in die Konfigurationsdatei vars ein. Anschließend führt man folgende Kommandos aus um die Master Certificate Authority (CA) zu erstellen.

```
. ./vars
./clean-all
./build-ca
```

Den noch fehlenden Schlüssel für den Server generiert

```
./build-key-server server,
```

die Diffie-Hellman Parameter werden durch

```
./build-dh
```

erzeugt. Die Schlüssel für die Clients werden ähnlich den Schlüsseln des Servers generiert. Da im Betrieb nicht mehrere Clients den selben Schlüssel verwenden dürfen, muss für jeden Client ein eigener Schlüssel erstellt werden. Der Aufruf

```
./build-key pcsec01
```

generiert einen Client-Schlüssel und Speichert ihn in der Datei `pcsec01`. Während des Erzeugens der Schlüssel muss man bestätigen, das der erzeugte Schlüssel von der oben erstellten CA signiert und somit gültig wird. Dieser Schritt muss für jeden Rechner wiederholt werden.
Da der Endpunkt des VPNs das Tap-Device `tap0` ist, unser Managementnetzwerk aber erst mit `eth1` beginnt, müssen die beiden Netzwerkendpunkte noch über eine Bridge verbunden werden. Das erledigt B.2. Das Skript erzeugt das Gerät `tap0` sowie die Bridge `br0`, fügt die Geräte `eth1` und `tap0` der Bridge hinzu und überträgt die IP-Adresse von `eth1` auf die Bridge. Der Aufruf von

```
modprobe tun
/etc/openvpn/sample-scripts/bridge-start
openvpn /etc/openvpn/server.conf
```

lädt den Tun-/Tap-Treiber, konfiguriert die Bridge und startet den VPN Server. Um den Start künftig automatisch beim Booten durchzuführen, verlinkt man noch das schon vorhandene Skript `/etc/init.d/openvpn` in den Runlevel 5. Da das Init-Skript von einem VPN ohne Bridge ausgeht, müssen in den Start und Stopp Anweisungen des Skripts die Aufrufe

`/etc/openvpn/sample-scripts/bridge-start` [B.2]

beziehungsweise

`/etc/openvpn/sample-scripts/bridge-stop` [B.3]

eingefügt werden. Das Skript `bridge-stop` ist zuständig für die Entfernung von Tap-Gerät und Bridge und sorgt dafür, dass `eth1` wieder seine ursprüngliche Adresse erhält. Das modifizierte Init-Skript befindet sich in B.4. Abschließend wird die SuSE-Firewall durch die eigens für den Login-Server geschriebene Firewall ersetzt. Dazu wird das Firewall-Skript aus 4.2 nach `/etc/init.d/` kopiert und in den Runlevel 5 verlinkt.

Als letzten Schritt muss noch der Apache Webserver auf dem System installiert werden.
In der Konfigurationsdatei `/etc/sysconfig/apache2` aktiviert man danach die Module

- proxy
- proxy_connect
- proxy_http
- rewrite

und trägt in der Konfigurationsdatei `/etc/apache2/default-server.conf` folgende Weiterleitung auf den richtigen Webserver ein.

```
ProxyPass / http://192.168.20.253/
ProxyPassReverse / http://192.168.20.253/
```

A.5 Konfiguration des Secservers

Nach dem Starten des Secservers konfiguriert man zuerst analog zur Vorgehensweise beim Login-Server die Netzwerkkarten wie folgt:

- eth0
  - IP-Adresse: 192.168.20.1
  - Netzmaske: 255.255.255.0
- eth1
  - IP-Adresse: 192.168.216.254
  - Netzmaske: 255.255.255.0
- Routing & DNS
  - Defaultgateway: 192.168.20.254
  - DNS: localhost
  - Routing: enabled
  - Hostname: secserver

  - Domain: secp.nm.ifi.lmu.de

Die Einträge in den Routingtabellen [B.5, B.6] sind je nach Szenario verschieden und werden dynamisch beim Starten der Maschine gesetzt.
Anschließend wird der Nameserver Bind installiert, konfiguriert und in den Runlevel 5 verlinkt. Der Secserver soll neben einem DNS-Forward für externe Namensauflösung auch primärer DNS für die Namensauflösung im internen Netz sein.
Die Konfigurationsdatei `/etc/named.conf` [B.7] muss daher um die Zonen `secp.nm.ifi.lmu.de` [B.8, B.9] beziehungsweise `secp.nm.informatik.uni-muenchen.de` [B.10, B.11] erweitert werden. Der Inhalt der Konfigurationsdateien für die jeweiligen Zonen hängt vom aktiven Szenario ab und muss daher wie die Routen beim Start eines Szenarios dynamisch erzeugt werden. Als Firewall kommt 4.3 zum Einsatz. Diese wird analog zum Login-Server aktiviert.

A.6 Konfiguration des Rechners test4all

Die Netzwerkkonfiguration für den Rechner test4all ist wie folgt:

- eth0
  - Managementinterface! Die Karte wird von Xen beim Starten automatisch konfiguriert. **Keine** Konfigurationsdatei `ifcfg-eth0` anlegen!
- eth1
  - IP-Adresse: 192.168.216.253
  - Netzmaske: 255.255.255.0
- Routing & DNS
  - Defaultgateway: 192.168.20.254
  - DNS: 192.168.20.254
  - Hostname: test4all
  - Domain: secp.nm.ifi.lmu.de

Im Rechner `test4all` ist ebenfalls ein DNS-Server aktiv. Er dient lediglich für Experimente bei der Konfiguration von DNS-Servern im Praktikum und beantwortet nur Anfragen,
die im Namespace `informatik.uni-muenchen.de` liegen. Der Server arbeitet lediglich als Forwarder und bezieht seine Informationen vom DNS des Secservers. Die Installation von bind läuft analog zum Secserver ab, die Konfigurationsdatei `/etc/named.conf` ist ebenfalls im B.14. Die Konfigurationen der Routingdateien [B.12, B.13] sind bis auf ein abweichendes Defaultgateway identisch mit der des Secservers.

A.7 Konfiguration des Rechners hacktest

Die Netzwerkkonfiguration für den Rechner `hacktest` ist wie folgt:

- eth0
  - Managementinterface! Die Karte wird von Xen beim Starten automatisch konfiguriert. **Keine** Konfigurationsdatei `ifcfg-eth0` anlegen!
- eth1
  - IP-Adresse: 192.168.216.252
  - Netzmaske: 255.255.255.0
- Routing & DNS

- Defaultgateway: 192.168.20.252
- DNS: 192.168.20.254
- Hostname: hacktest
- Domain: secp.nm.ifi.lmu.de

Der Rechner *hacktest* beheimatet ein Rootkit, das von den Studenten im Laufe des Praktikums gefunden werden muss. Da die Version 2.6 des Linux Kernels kaum mehr Möglichkeiten für die Integration eines LKM-Rootkits bietet, soll in diesem Rechner ein Kernel der 2.4er Version verwendet werden. Dieser wird bei der Installation von Xen allerdings nicht mehr automatisch erzeugt. Der Aufruf

```
KERNELS=*2.4* make dist
```

im Verzeichnis der Xen-Sourcen erledigt diese Aufgabe. Die erzeugten Kernel enthalten allerdings wie schon die 2.6er Kernel nicht alle benötigten Module und müssen nachkonfiguriert werden. Anschließend ist es nötig, die Kernel Module im Image auszutauschen. Da diese Variante des Kernels nicht optimiert für SuSE 9.3 ist scheitert ein Start der virtuellen Maschine in dieser Konfiguration. Abhilfe schafft das Setzen des Eintrags `RUN_PARALLEL=no` in `/etc/sysconfig/boot`.

A.8 Virtual Maschine Management Tool

Für die Inbetriebnahme des Virtual Maschine Management Tools ist ein installierter `Apache` Server samt eingebundenem Python Modul `mod_python` erforderlich. Beide Komponenten lassen sich mit YaST installieren und konfigurieren. Um `mod_python` für ausgewählte Verzeichnisse zu aktivieren, muss die Datei `/etc/apache2/default-server.conf` editiert werden. Ihr werden folgende Direktiven hinzugefügt:

```
 <Directory ''/srv/www/htdocs/xen/state/''>
AddHandler mod_python .psp
PythonHandler mod_python
# PythonDebug On
</Directory>

 <Directory ''/srv/www/htdocs/xen/action/''>
AddHandler mod_python .py
SetHandler mod_python
PythonHandler mod_python.publisher
PythonHandler auth
PythonAuthenHandler auth
AuthType Basic
AuthName ''Restricted Area''
required valid-user
# PythonDebug On
</Directory>
```

Anschließend werden die benötigten Dateien in ihre Zielverzeichnisse kopiert.

- index.html → /src/www/htdocs/
- index.psp → /src/www/htdocs/state/
- request.py → /src/www/htdocs/action/

- auth.py → /src/www/htdocs/action/

B Skripte & Konfigurationsdateien

B.1 OpenVPN

Listing B.1: Konfigurationsdatei des Openvpn Servers

```
################################################
# Sample OpenVPN 2.0 config file for           #
# multi-client server.                         #
#                                              #
# This file is for the server side             #
# of a many-clients <-> one-server             #
# OpenVPN configuration.                       #
#                                              #
# OpenVPN also supports                        #
# single-machine <-> single-machine            #
# configurations (See the Examples page        #
# on the web site for more info).              #
#                                              #
# This config should work on Windows           #
# or Linux/BSD systems.  Remember on           #
# Windows to quote pathnames and use           #
# double backslashes, e.g.:                    #
# "C:\\Program Files\\OpenVPN\\config\\foo.key" #
#                                              #
# Comments are preceded with '#' or ';'        #
################################################

# Which local IP address should OpenVPN
# listen on? (optional)
local 141.84.218.144

# Which TCP/UDP port should OpenVPN listen on?
# If you want to run multiple OpenVPN instances
# on the same machine, use a different port
# number for each one.  You will need to
# open up this port on your firewall.
port 1194

# TCP or UDP server?
proto tcp
;proto udp

# "dev tun" will create a routed IP tunnel,
# "dev tap" will create an ethernet tunnel.
# Use "dev tap" if you are ethernet bridging.
# If you want to control access policies
# over the VPN, you must create firewall
# rules for the the TUN/TAP interface.
# On non-Windows systems, you can give
# an explicit unit number, such as tun0.
# On Windows, use "dev-node" for this.
# On most systems, the VPN will not function
```

```
# unless you partially or fully disable
# the firewall for the TUN/TAP interface.
dev tap0
;dev tun

# Windows needs the TAP-Win32 adapter name
# from the Network Connections panel if you
# have more than one.  On XP SP2 or higher,
# you may need to selectively disable the
# Windows firewall for the TAP adapter.
# Non-Windows systems usually don't need this.
;dev-node tap0

# SSL/TLS root certificate (ca), certificate
# (cert), and private key (key).  Each client
# and the server must have their own cert and
# key file.  The server and all clients will
# use the same ca file.
#
# See the "easy-rsa" directory for a series
# of scripts for generating RSA certificates
# and private keys.  Remember to use
# a unique Common Name for the server
# and each of the client certificates.
#
# Any X509 key management system can be used.
# OpenVPN can also use a PKCS #12 formatted key file
# (see "pkcs12" directive in man page).
ca /etc/openvpn/easy-rsa/keys/ca.crt
cert /etc/openvpn/easy-rsa/keys/server.crt
key /etc/openvpn/easy-rsa/keys/server.key  # This file should be kept secret

# Diffie hellman parameters.
# Generate your own with:
#   openssl dhparam -out dh1024.pem 1024
# Substitute 2048 for 1024 if you are using
# 2048 bit keys.
dh /etc/openvpn/easy-rsa/keys/dh1024.pem

# Configure server mode and supply a VPN subnet
# for OpenVPN to draw client addresses from.
# The server will take 10.8.0.1 for itself,
# the rest will be made available to clients.
# Each client will be able to reach the server
# on 10.8.0.1. Comment this line out if you are
# ethernet bridging. See the man page for more info.
;server 192.168.10.0 255.255.255.0

# Maintain a record of client <-> virtual IP address
# associations in this file.  If OpenVPN goes down or
# is restarted, reconnecting clients can be assigned
# the same virtual IP address from the pool that was
# previously assigned.
ifconfig-pool-persist ipp.txt

# Configure server mode for ethernet bridging.
# You must first use your OS's bridging capability
# to bridge the TAP interface with the ethernet
# NIC interface.  Then you must manually set the
# IP/netmask on the bridge interface, here we
# assume 10.8.0.4/255.255.255.0.  Finally we
```

```
# must set aside an IP range in this subnet
# (start=10.8.0.50 end=10.8.0.100) to allocate
# to connecting clients.  Leave this line commented
# out unless you are ethernet bridging.
server-bridge 192.168.10.254 255.255.255.0 192.168.10.100 192.168.10.180

# Push routes to the client to allow it
# to reach other private subnets behind
# the server.  Remember that these
# private subnets will also need
# to know to route the OpenVPN client
# address pool (10.8.0.0/255.255.255.0)
# back to the OpenVPN server.
;push "route␣192.168.10.0␣255.255.255.0"
;push "route␣192.168.20.0␣255.255.255.0"

# To assign specific IP addresses to specific
# clients or if a connecting client has a private
# subnet behind it that should also have VPN access,
# use the subdirectory "ccd" for client-specific
# configuration files (see man page for more info).

# EXAMPLE: Suppose the client
# having the certificate common name "Thelonious"
# also has a small subnet behind his connecting
# machine, such as 192.168.40.128/255.255.255.248.
# First, uncomment out these lines:
;client-config-dir ccd
;route 192.168.40.128 255.255.255.248
# Then create a file ccd/Thelonious with this line:
#   iroute 192.168.40.128 255.255.255.248
# This will allow Thelonious' private subnet to
# access the VPN.  This example will only work
# if you are routing, not bridging, i.e. you are
# using "dev tun" and "server" directives.

# EXAMPLE: Suppose you want to give
# Thelonious a fixed VPN IP address of 10.9.0.1.
# First uncomment out these lines:
;client-config-dir ccd
;route 10.9.0.0 255.255.255.252
# Then add this line to ccd/Thelonious:
#   ifconfig-push 10.9.0.1 10.9.0.2

# Suppose that you want to enable different
# firewall access policies for different groups
# of clients.  There are two methods:
# (1) Run multiple OpenVPN daemons, one for each
#     group, and firewall the TUN/TAP interface
#     for each group/daemon appropriately.
# (2) (Advanced) Create a script to dynamically
#     modify the firewall in response to access
#     from different clients.  See man
#     page for more info on learn-address script.
;learn-address ./script

# If enabled, this directive will configure
# all clients to redirect their default
# network gateway through the VPN, causing
# all IP traffic such as web browsing and
# and DNS lookups to go through the VPN
```

```
# (The OpenVPN server machine may need to NAT
# the TUN/TAP interface to the internet in
# order for this to work properly).
# CAVEAT: May break client's network config if
# client's local DHCP server packets get routed
# through the tunnel.  Solution: make sure
# client's local DHCP server is reachable via
# a more specific route than the default route
# of 0.0.0.0/0.0.0.0.
;push "redirect-gateway"

# Certain Windows-specific network settings
# can be pushed to clients, such as DNS
# or WINS server addresses.  CAVEAT:
# http://openvpn.net/faq.html#dhcpcaveats
;push "dhcp-option␣DNS␣10.8.0.1"
;push "dhcp-option␣WINS␣10.8.0.1"

# Uncomment this directive to allow different
# clients to be able to "see" each other.
# By default, clients will only see the server.
# To force clients to only see the server, you
# will also need to appropriately firewall the
# server's TUN/TAP interface.
;client-to-client

# Uncomment this directive if multiple clients
# might connect with the same certificate/key
# files or common names.  This is recommended
# only for testing purposes.  For production use,
# each client should have its own certificate/key
# pair.
#
# IF YOU HAVE NOT GENERATED INDIVIDUAL
# CERTIFICATE/KEY PAIRS FOR EACH CLIENT,
# EACH HAVING ITS OWN UNIQUE "COMMON NAME",
# UNCOMMENT THIS LINE OUT.
;duplicate-cn

# The keepalive directive causes ping-like
# messages to be sent back and forth over
# the link so that each side knows when
# the other side has gone down.
# Ping every 10 seconds, assume that remote
# peer is down if no ping received during
# a 120 second time period.
keepalive 10 120

# For extra security beyond that provided
# by SSL/TLS, create an "HMAC firewall"
# to help block DoS attacks and UDP port flooding.
#
# Generate with:
#   openvpn --genkey --secret ta.key
#
# The server and each client must have
# a copy of this key.
# The second parameter should be '0'
# on the server and '1' on the clients.
;tls-auth ta.key 0 # This file is secret
```

```
# Select a cryptographic cipher.
# This config item must be copied to
# the client config file as well.
cipher BF-CBC        # Blowfish (default)
;cipher AES-128-CBC   # AES
;cipher DES-EDE3-CBC  # Triple-DES

# Enable compression on the VPN link.
# If you enable it here, you must also
# enable it in the client config file.
comp-lzo

# The maximum number of concurrently connected
# clients we want to allow.
;max-clients 100

# It's a good idea to reduce the OpenVPN
# daemon's privileges after initialization.
#
# You can uncomment this out on
# non-Windows systems.
user nobody
group nobody

# The persist options will try to avoid
# accessing certain resources on restart
# that may no longer be accessible because
# of the privilege downgrade.
persist-key
persist-tun

# Output a short status file showing
# current connections, truncated
# and rewritten every minute.
status openvpn-status.log

# By default, log messages will go to the syslog (or
# on Windows, if running as a service, they will go to
# the "\Program Files\OpenVPN\log" directory).
# Use log or log-append to override this default.
# "log" will truncate the log file on OpenVPN startup,
# while "log-append" will append to it.  Use one
# or the other (but not both).
;log         openvpn.log
;log-append  openvpn.log

# Set the appropriate level of log
# file verbosity.
#
# 0 is silent, except for fatal errors
# 4 is reasonable for general usage
# 5 and 6 can help to debug connection problems
# 9 is extremely verbose
verb 3

# Silence repeating messages.  At most 20
# sequential messages of the same message
# category will be output to the log.
;mute 20
```

Listing B.2: Shell-Skript zum Erzeugen und Konfigurieren der Bridge für den Openvpn Server

```
#!/bin/bash

################################
# Set up Ethernet bridge on Linux
################################

# Define Bridge Interface
br="br0"

# Define list of TAP interfaces to be bridged,
# for example tap="tap0 tap1 tap2".
tap="tap0"

# Define physical ethernet interface to be bridged
# with TAP interface(s) above.
eth="eth1"
eth_ip="192.168.10.254"
eth_netmask="255.255.255.0"
eth_broadcast="192.168.10.255"

for t in $tap; do
    openvpn --mktun --dev $t
done

brctl addbr $br
brctl addif $br $eth

for t in $tap; do
    brctl addif $br $t
done

for t in $tap; do
    ifconfig $t 0.0.0.0 promisc up
done

ifconfig $eth 0.0.0.0 promisc up

ifconfig $br $eth_ip netmask $eth_netmask broadcast $eth_broadcast
```

Listing B.3: Shell-Skript zum Entfernen der Bridge des Openvpn Servers

```
#!/bin/bash

##################################
# Tear Down Ethernet bridge on Linux
##################################

# Define Bridge Interface
br="br0"

# Define list of TAP interfaces to be bridged together
tap="tap0"

ifconfig $br down
brctl delbr $br

for t in $tap; do
    openvpn --rmtun --dev $t
done
```

```
ifconfig eth1 192.168.10.254 up
```

Listing B.4: Init-Skript zum Start des Openvpn Servers

```
#! /bin/sh
# Copyright (c) 2003 SuSE Linux AG
#
# Author: Peter Poeml <poeml@suse.de>
#
# inspired by the init script contributed to the OpenVPN project by
# Douglas Keller <doug@voidstar.dyndns.org>
#
# /etc/init.d/openvpn
#   and its symbolic link
# /usr/sbin/rcopenvpn
#
### BEGIN INIT INFO
# Provides:                     openvpn
# Required-Start:               $local_fs $remote_fs $network
# X-UnitedLinux-Should-Start:   $syslog
# Required-Stop:                $local_fs $remote_fs $network
# X-UnitedLinux-Should-Stop:    $syslog
# Default-Start:                3 5
# Default-Stop:                 0 1 2 6
# Short-Description:            OpenVPN tunnel
# Description:                  Start OpenVPN tunnel
### END INIT INFO


# test -s /etc/sysconfig/openvpn && \
#       . /etc/sysconfig/openvpn

DAEMON="OpenVPN"
openvpn=/usr/sbin/openvpn
confdir=/etc/openvpn
piddir=/var/run/openvpn
test -d $piddir || mkdir $piddir

test -x $openvpn || exit 5

# Shell functions sourced from /etc/rc.status:
#      rc_check         check and set local and overall rc status
#      rc_status        check and set local and overall rc status
#      rc_status -v     ditto but be verbose in local rc status
#      rc_status -v -r  ditto and clear the local rc status
#      rc_failed        set local and overall rc status to failed
#      rc_failed <num>  set local and overall rc status to <num><num>
#      rc_reset         clear local rc status (overall remains)
#      rc_exit          exit appropriate to overall rc status
. /etc/rc.status

# First reset status of this service
rc_reset

# Return values acc. to LSB for all commands but status:
# 0 - success
# 1 - generic or unspecified error
# 2 - invalid or excess argument(s)
# 3 - unimplemented feature (e.g. "reload")
# 4 - insufficient privilege
# 5 - program is not installed
```

```
# 6 - program is not configured
# 7 - program is not running
#
# Note that starting an already running service, stopping
# or restarting a not-running service as well as the restart
# with force-reload (in case signalling is not supported) are
# considered a success.

shopt -s nullglob
ret=true

case "$1" in
    start)
        echo -n "Starting␣$DAEMON␣"

        /sbin/modprobe tun &>/dev/null
        /etc/openvpn/sample-scripts/bridge-start

        for conf in $confdir/*.conf; do
                pidfile=$(basename ${conf%%.conf}).pid
                $openvpn --daemon \
                        --writepid $piddir/$pidfile \
                        --config $conf \
                        --cd $confdir \
                        || ret=false
        done

        # Remember status and be verbose
        $ret
        rc_status -v
        ;;
    stop)
        echo -n "Shutting␣down␣$DAEMON␣"

        /etc/openvpn/sample-scripts/bridge-stop

        ## Stop daemon with killproc(8) and if this fails
        ## set echo the echo return value.

        for i in $piddir/*.pid; do
                killproc -p $i -TERM $openvpn || ret=false
        done

        # Remember status and be verbose
        $ret
        rc_status -v
        ;;
    try-restart)
        ## Do a restart only if the service was active before.
        ## Note: try-restart is now part of LSB (as of 1.9).
        ## RH has a similar command named condrestart.
        $0 status
        if test $? = 0; then
                $0 restart
        else
                rc_reset        # Not running is not a failure.
        fi
        # Remember status and be quiet
        rc_status
        ;;
    restart)
```

```
        ## Stop the service and regardless of whether it was
        ## running or not, start it again.
        $0 stop
        sleep 3
        $0 start

        # Remember status and be quiet
        rc_status
        ;;
    reload)
        for i in $piddir/*.pid; do
                killproc -p $i -HUP $openvpn || ret=false
        done
        rc_status -v
        ;;
    reopen)
        for i in $piddir/*.pid; do
                killproc -p $i -USR1 $openvpn || ret=false
        done
        rc_status -v
        ;;
    status)
        echo -n "Checking for $DAEMON: "
        running=false
        for i in $piddir/*.pid; do
                running=true
                killproc -p $i -USR2 $openvpn || { rv=$?; ret=false; }
        done
        if $running; then
                $ret
                rc_status -v
                echo Status written to /var/log/messages
        else
                rc_failed 3
                rc_status -v
        fi
        ;;
    *)
        echo "Usage: $0 {start|stop|status|try-restart|restart|reload|reopen}"
        exit 1
esac
rc_exit
```

B.2 DNS & Routing

Listing B.5: Routingeinträge des Server im Szenario 1

```
192.168.216.0 192.168.216.129 255.255.255.240
192.168.216.16 192.168.216.130 255.255.255.240
192.168.216.32 192.168.216.131 255.255.255.240
192.168.216.48 192.168.216.132 255.255.255.240
192.168.216.64 192.168.216.133 255.255.255.240
192.168.216.80 192.168.216.134 255.255.255.240
192.168.216.96 192.168.216.135 255.255.255.240
192.168.216.112 192.168.216.136 255.255.255.240
default 192.168.20.254 - -
```

Listing B.6: Routingeinträge des Server im Szenario 2

```
192.168.216.0 192.168.216.193 255.255.255.248
192.168.216.8 192.168.216.194 255.255.255.248
192.168.216.16 192.168.216.195 255.255.255.248
192.168.216.24 192.168.216.196 255.255.255.248
192.168.216.32 192.168.216.197 255.255.255.248
192.168.216.40 192.168.216.198 255.255.255.248
192.168.216.48 192.168.216.199 255.255.255.248
192.168.216.56 192.168.216.200 255.255.255.248
192.168.216.64 192.168.216.201 255.255.255.248
192.168.216.72 192.168.216.202 255.255.255.248
192.168.216.80 192.168.216.203 255.255.255.248
192.168.216.88 192.168.216.204 255.255.255.248
192.168.216.96 192.168.216.205 255.255.255.248
192.168.216.104 192.168.216.206 255.255.255.248
192.168.216.112 192.168.216.207 255.255.255.248
192.168.216.120 192.168.216.208 255.255.255.248
192.168.216.128 192.168.216.209 255.255.255.248
192.168.216.136 192.168.216.210 255.255.255.248
192.168.216.144 192.168.216.211 255.255.255.248
192.168.216.152 192.168.216.212 255.255.255.248
default 192.168.20.254 - -
```

Listing B.7: Konfigurationsdatei des Nameservers bind im Secserver

```
# Copyright (c) 2001-2004 SuSE Linux AG, Nuernberg, Germany.
# All rights reserved.
#
# Author: Frank Bodammer, Lars Mueller <lmuelle@suse.de>
#
# /etc/named.conf
#
# This is a sample configuration file for the name server BIND 9.  It works as
# a caching only name server without modification.
#
# A sample configuration for setting up your own domain can be found in
# /usr/share/doc/packages/bind/sample-config.
#
# A description of all available options can be found in
# /usr/share/doc/packages/bind/misc/options.

options {

        # The directory statement defines the name server's working directory
```

```
        directory "/var/lib/named";

        # Write dump and statistics file to the log subdirectory.  The
        # pathenames are relative to the chroot jail.

        dump-file "/var/log/named_dump.db";
        statistics-file "/var/log/named.stats";

        # The forwarders record contains a list of servers to which queries
        # should be forwarded.  Enable this line and modify the IP address to
        # your provider's name server.  Up to three servers may be listed.

        #forwarders { 192.0.2.1; 192.0.2.2; };

        # Enable the next entry to prefer usage of the name server declared in
        # the forwarders section.

        #forward first;

        # The listen-on record contains a list of local network interfaces to
        # listen on.  Optionally the port can be specified.  Default is to
        # listen on all interfaces found on your system.  The default port is
        # 53.

        #listen-on port 53 { 127.0.0.1; };

        # The listen-on-v6 record enables or disables listening on IPv6
        # interfaces.  Allowed values are 'any' and 'none' or a list of
        # addresses.

        listen-on-v6 { any; };

        # The next three statements may be needed if a firewall stands between
        # the local server and the internet.

        #query-source address * port 53;
        #transfer-source * port 53;
        #notify-source * port 53;

        # The allow-query record contains a list of networks or IP addresses
        # to accept and deny queries from. The default is to allow queries
        # from all hosts.

        #allow-query { 127.0.0.1; };

        # If notify is set to yes (default), notify messages are sent to other
        # name servers when the the zone data is changed.  Instead of setting
        # a global 'notify' statement in the 'options' section, a separate
        # 'notify' can be added to each zone definition.

        notify no;
        include "/etc/named.d/forwarders.conf";
};

# To configure named's logging remove the leading '#' characters of the
# following examples.
#logging {
#       # Log queries to a file limited to a size of 100 MB.
#       channel query_logging {
#               file "/var/log/named_querylog"
#                       versions 3 size 100M;
```

```
#               print-time yes;                 // timestamp log entries
#       };
#       category queries {
#               query_logging;
#       };
#
#       # Or log this kind alternatively to syslog.
#       channel syslog_queries {
#               syslog user;
#               severity info;
#       };
#       category queries { syslog_queries; };
#
#       # Log general name server errors to syslog.
#       channel syslog_errors {
#               syslog user;
#               severity error;
#       };
#       category default { syslog_errors;  };
#
#       # Don't log lame server messages.
#       category lame-servers { null; };
#};

# The following zone definitions don't need any modification.  The first one
# is the definition of the root name servers.  The second one defines
# localhost while the third defines the reverse lookup for localhost.

zone "." in {
        type hint;
        file "root.hint";
};

zone "localhost" in {
        type master;
        file "localhost.zone";
};

zone "0.0.127.in-addr.arpa" in {
        type master;
        file "127.0.0.zone";
};

# Include the meta include file generated by createNamedConfInclude.  This
# includes all files as configured in NAMED_CONF_INCLUDE_FILES from
# /etc/sysconfig/named

include "/etc/named.conf.include";
logging {
        category default { log_syslog; };
        channel log_syslog { syslog; };
};
zone "secp.nm.ifi.lmu.de" in {
        file "master/secp.nm.ifi.lmu.de";
        type master;
};
zone "secp.nm.informatik.uni-muenchen.de" in {
        file "master/secp.nm.informatik.uni-muenchen.de";
        type master;
};
```

```
# You can insert further zone records for your own domains below or create
# single files in /etc/named.d/ and add the file names to
# NAMED_CONF_INCLUDE_FILES.
# See /usr/share/doc/packages/bind/README.SUSE for more details.
```

Listing B.8: Zonendatei des Nameservers bind im Secserver für die Zone secp.nm.ifi.lmu.de, erstes Szenario

```
$TTL 2d
@               IN SOA          localhost.      root.secserver.secp.nm.ifi.lmu.de
    . (
                                2006030202      ; serial
                                3h              ; refresh
                                1h              ; retry
                                1w              ; expiry
                                1d )            ; minimum

secp.nm.ifi.lmu.de.     IN NS           secserver.secp.nm.ifi.lmu.de.
secserver       IN A            192.168.216.254
test4all        IN A            192.168.216.253
hacktest        IN A            192.168.216.252
pcsec01         IN A            192.168.216.1
pcsec02         IN A            192.168.216.2
pcsec03         IN A            192.168.216.3
pcsec04         IN A            192.168.216.4
pcsec05         IN A            192.168.216.129
pcsec05-eth2    IN A            192.168.216.5
pcsec06         IN A            192.168.216.17
pcsec07         IN A            192.168.216.18
pcsec08         IN A            192.168.216.19
pcsec09         IN A            192.168.216.20
pcsec10         IN A            192.168.216.130
pcsec10-eth2    IN A            192.168.216.21
pcsec11         IN A            192.168.216.33
pcsec12         IN A            192.168.216.34
pcsec13         IN A            192.168.216.35
pcsec14         IN A            192.168.216.36
pcsec15         IN A            192.168.216.131
pcsec15-eth2    IN A            192.168.216.37
pcsec16         IN A            192.168.216.49
pcsec17         IN A            192.168.216.50
pcsec18         IN A            192.168.216.51
pcsec19         IN A            192.168.216.52
pcsec20         IN A            192.168.216.132
pcsec20-eth2    IN A            192.168.216.53
pcsec21         IN A            192.168.216.65
pcsec22         IN A            192.168.216.66
pcsec23         IN A            192.168.216.67
pcsec24         IN A            192.168.216.68
pcsec25         IN A            192.168.216.133
pcsec25-eth2    IN A            192.168.216.69
pcsec26         IN A            192.168.216.81
pcsec27         IN A            192.168.216.82
pcsec28         IN A            192.168.216.83
pcsec29         IN A            192.168.216.84
pcsec30         IN A            192.168.216.134
pcsec30-eth2    IN A            192.168.216.85
pcsec31         IN A            192.168.216.97
pcsec32         IN A            192.168.216.98
pcsec33         IN A            192.168.216.99
```

```
pcsec34         IN A            192.168.216.100
pcsec35         IN A            192.168.216.135
pcsec35-eth2    IN A            192.168.216.101
pcsec36         IN A            192.168.216.113
pcsec37         IN A            192.168.216.114
pcsec38         IN A            192.168.216.115
pcsec39         IN A            192.168.216.116
pcsec40         IN A            192.168.216.136
pcsec40-eth2    IN A            192.168.216.117
```

Listing B.9: Zonendatei des Nameservers bind im Secserver für die Zone secp.nm.ifi.lmu.de, zweites Szenario

```
$TTL 2d
@               IN SOA          localhost.      root.secserver.secp.nm.ifi.lmu.de
    . (
                                2006030202      ; serial
                                3h              ; refresh
                                1h              ; retry
                                1w              ; expiry
                                1d )            ; minimum

secp.nm.ifi.lmu.de.     IN NS           secserver.secp.nm.ifi.lmu.de.
secserver       IN A            192.168.216.254
test4all        IN A            192.168.216.253
hacktest        IN A            192.168.216.252
pcsec01         IN A            192.168.216.193
pcsec01-eth2    IN A            192.168.216.1
pcsec02         IN A            192.168.216.2
pcsec03         IN A            192.168.216.194
pcsec03-eth2    IN A            192.168.216.8
pcsec04         IN A            192.168.216.10
pcsec05         IN A            192.168.216.195
pcsec05-eth2    IN A            192.168.216.17
pcsec06         IN A            192.168.216.18
pcsec07         IN A            192.168.216.196
pcsec07-eth2    IN A            192.168.216.25
pcsec08         IN A            192.168.216.26
pcsec09         IN A            192.168.216.197
pcsec09-eth2    IN A            192.168.216.33
pcsec10         IN A            192.168.216.34
pcsec11         IN A            192.168.216.198
pcsec11-eth2    IN A            192.168.216.41
pcsec12         IN A            192.168.216.42
pcsec13         IN A            192.168.216.199
pcsec13-eth2    IN A            192.168.216.49
pcsec14         IN A            192.168.216.50
pcsec15         IN A            192.168.216.200
pcsec15-eth2    IN A            192.168.216.57
pcsec16         IN A            192.168.216.58
pcsec17         IN A            192.168.216.201
pcsec17-eth2    IN A            192.168.216.65
pcsec18         IN A            192.168.216.66
pcsec19         IN A            192.168.216.202
pcsec19-eth2    IN A            192.168.216.73
pcsec20         IN A            192.168.216.74
pcsec21         IN A            192.168.216.203
pcsec21-eth2    IN A            192.168.216.81
pcsec22         IN A            192.168.216.82
pcsec23         IN A            192.168.216.204
pcsec23-eth2    IN A            192.168.216.89
```

```
pcsec24         IN A            192.168.216.90
pcsec25         IN A            192.168.216.205
pcsec25-eth2    IN A            192.168.216.97
pcsec26         IN A            192.168.216.98
pcsec27         IN A            192.168.216.206
pcsec27-eth2    IN A            192.168.216.105
pcsec28         IN A            192.168.216.106
pcsec29         IN A            192.168.216.207
pcsec29-eth2    IN A            192.168.216.113
pcsec30         IN A            192.168.216.114
pcsec31         IN A            192.168.216.208
pcsec31-eth2    IN A            192.168.216.121
pcsec32         IN A            192.168.216.122
pcsec33         IN A            192.168.216.209
pcsec33-eth2    IN A            192.168.216.129
pcsec34         IN A            192.168.216.130
pcsec35         IN A            192.168.216.210
pcsec35-eth2    IN A            192.168.216.137
pcsec36         IN A            192.168.216.138
pcsec37         IN A            192.168.216.211
pcsec37-eth2    IN A            192.168.216.145
pcsec38         IN A            192.168.216.146
pcsec39         IN A            192.168.216.212
pcsec39-eth2    IN A            192.168.216.153
pcsec40         IN A            192.168.216.154
```

Listing B.10: Zonendatei des Nameservers bind im Secserver für die Zone secp.nm.informatik.uni-muenchen.de, erstes Szenario

```
$TTL 2d
@               IN SOA          localhost.      root.secserver.secp.nm.ifi.lmu.de
    . (
                                2006030202      ; serial
                                3h              ; refresh
                                1h              ; retry
                                1w              ; expiry
                                1d )            ; minimum

secp.nm.informatik.uni-muenchen.de.     IN NS           secserver.secp.nm.ifi.lmu
    .de.
secserver       IN A            192.168.216.254
test4all        IN A            192.168.216.253
hacktest        IN A            192.168.216.252
pcsec01         IN A            192.168.216.1
pcsec02         IN A            192.168.216.2
pcsec03         IN A            192.168.216.3
pcsec04         IN A            192.168.216.4
pcsec05         IN A            192.168.216.129
pcsec05-eth2    IN A            192.168.216.5
pcsec06         IN A            192.168.216.17
pcsec07         IN A            192.168.216.18
pcsec08         IN A            192.168.216.19
pcsec09         IN A            192.168.216.20
pcsec10         IN A            192.168.216.130
pcsec10-eth2    IN A            192.168.216.21
pcsec11         IN A            192.168.216.33
pcsec12         IN A            192.168.216.34
pcsec13         IN A            192.168.216.35
pcsec14         IN A            192.168.216.36
pcsec15         IN A            192.168.216.131
```

```
pcsec15-eth2    IN A            192.168.216.37
pcsec16         IN A            192.168.216.49
pcsec17         IN A            192.168.216.50
pcsec18         IN A            192.168.216.51
pcsec19         IN A            192.168.216.52
pcsec20         IN A            192.168.216.132
pcsec20-eth2    IN A            192.168.216.53
pcsec21         IN A            192.168.216.65
pcsec22         IN A            192.168.216.66
pcsec23         IN A            192.168.216.67
pcsec24         IN A            192.168.216.68
pcsec25         IN A            192.168.216.133
pcsec25-eth2    IN A            192.168.216.69
pcsec26         IN A            192.168.216.81
pcsec27         IN A            192.168.216.82
pcsec28         IN A            192.168.216.83
pcsec29         IN A            192.168.216.84
pcsec30         IN A            192.168.216.134
pcsec30-eth2    IN A            192.168.216.85
pcsec31         IN A            192.168.216.97
pcsec32         IN A            192.168.216.98
pcsec33         IN A            192.168.216.99
pcsec34         IN A            192.168.216.100
pcsec35         IN A            192.168.216.135
pcsec35-eth2    IN A            192.168.216.101
pcsec36         IN A            192.168.216.113
pcsec37         IN A            192.168.216.114
pcsec38         IN A            192.168.216.115
pcsec39         IN A            192.168.216.116
pcsec40         IN A            192.168.216.136
pcsec40-eth2    IN A            192.168.216.117
```

Listing B.11: Zonendatei des Nameservers bind im Secserver für die Zone secp.nm.informatik.uni-muenchen.de, zweites Szenario

```
$TTL 2d
@               IN SOA          localhost.      root.secserver.secp.nm.ifi.lmu.de
    . (
                                2006030202      ; serial
                                3h              ; refresh
                                1h              ; retry
                                1w              ; expiry
                                1d )            ; minimum

secp.nm.informatik.uni-muenchen.de.     IN NS           secserver.secp.nm.ifi.lmu
    .de.
secserver       IN A            192.168.216.254
test4all        IN A            192.168.216.253
hacktest        IN A            192.168.216.252
pcsec01         IN A            192.168.216.193
pcsec01-eth2    IN A            192.168.216.1
pcsec02         IN A            192.168.216.2
pcsec03         IN A            192.168.216.194
pcsec03-eth2    IN A            192.168.216.8
pcsec04         IN A            192.168.216.10
pcsec05         IN A            192.168.216.195
pcsec05-eth2    IN A            192.168.216.17
pcsec06         IN A            192.168.216.18
pcsec07         IN A            192.168.216.196
pcsec07-eth2    IN A            192.168.216.25
```

```
pcsec08         IN A            192.168.216.26
pcsec09         IN A            192.168.216.197
pcsec09-eth2    IN A            192.168.216.33
pcsec10         IN A            192.168.216.34
pcsec11         IN A            192.168.216.198
pcsec11-eth2    IN A            192.168.216.41
pcsec12         IN A            192.168.216.42
pcsec13         IN A            192.168.216.199
pcsec13-eth2    IN A            192.168.216.49
pcsec14         IN A            192.168.216.50
pcsec15         IN A            192.168.216.200
pcsec15-eth2    IN A            192.168.216.57
pcsec16         IN A            192.168.216.58
pcsec17         IN A            192.168.216.201
pcsec17-eth2    IN A            192.168.216.65
pcsec18         IN A            192.168.216.66
pcsec19         IN A            192.168.216.202
pcsec19-eth2    IN A            192.168.216.73
pcsec20         IN A            192.168.216.74
pcsec21         IN A            192.168.216.203
pcsec21-eth2    IN A            192.168.216.81
pcsec22         IN A            192.168.216.82
pcsec23         IN A            192.168.216.204
pcsec23-eth2    IN A            192.168.216.89
pcsec24         IN A            192.168.216.90
pcsec25         IN A            192.168.216.205
pcsec25-eth2    IN A            192.168.216.97
pcsec26         IN A            192.168.216.98
pcsec27         IN A            192.168.216.206
pcsec27-eth2    IN A            192.168.216.105
pcsec28         IN A            192.168.216.106
pcsec29         IN A            192.168.216.207
pcsec29-eth2    IN A            192.168.216.113
pcsec30         IN A            192.168.216.114
pcsec31         IN A            192.168.216.208
pcsec31-eth2    IN A            192.168.216.121
pcsec32         IN A            192.168.216.122
pcsec33         IN A            192.168.216.209
pcsec33-eth2    IN A            192.168.216.129
pcsec34         IN A            192.168.216.130
pcsec35         IN A            192.168.216.210
pcsec35-eth2    IN A            192.168.216.137
pcsec36         IN A            192.168.216.138
pcsec37         IN A            192.168.216.211
pcsec37-eth2    IN A            192.168.216.145
pcsec38         IN A            192.168.216.146
pcsec39         IN A            192.168.216.212
pcsec39-eth2    IN A            192.168.216.153
pcsec40         IN A            192.168.216.154
```

Listing B.12: Routingeinträge für die Rechner test4all und hacktest im Szenario 1

```
192.168.216.0 192.168.216.129 255.255.255.240
192.168.216.16 192.168.216.130 255.255.255.240
192.168.216.32 192.168.216.131 255.255.255.240
192.168.216.48 192.168.216.132 255.255.255.240
192.168.216.64 192.168.216.133 255.255.255.240
192.168.216.80 192.168.216.134 255.255.255.240
192.168.216.96 192.168.216.135 255.255.255.240
192.168.216.112 192.168.216.136 255.255.255.240
default 192.168.216.254 - -
```

Listing B.13: Routingeinträge für die Rechner test4all und hacktest im Szenario 2

```
192.168.216.0 192.168.216.193 255.255.255.248
192.168.216.8 192.168.216.194 255.255.255.248
192.168.216.16 192.168.216.195 255.255.255.248
192.168.216.24 192.168.216.196 255.255.255.248
192.168.216.32 192.168.216.197 255.255.255.248
192.168.216.40 192.168.216.198 255.255.255.248
192.168.216.48 192.168.216.199 255.255.255.248
192.168.216.56 192.168.216.200 255.255.255.248
192.168.216.64 192.168.216.201 255.255.255.248
192.168.216.72 192.168.216.202 255.255.255.248
192.168.216.80 192.168.216.203 255.255.255.248
192.168.216.88 192.168.216.204 255.255.255.248
192.168.216.96 192.168.216.205 255.255.255.248
192.168.216.104 192.168.216.206 255.255.255.248
192.168.216.112 192.168.216.207 255.255.255.248
192.168.216.120 192.168.216.208 255.255.255.248
192.168.216.128 192.168.216.209 255.255.255.248
192.168.216.136 192.168.216.210 255.255.255.248
192.168.216.144 192.168.216.211 255.255.255.248
192.168.216.152 192.168.216.212 255.255.255.248
default 192.168.216.254 - -
```

Listing B.14: Konfigurationsdatei des Nameservers bind im test4all

```
# Copyright (c) 2001-2004 SuSE Linux AG, Nuernberg, Germany.
# All rights reserved.
#
# Author: Frank Bodammer, Lars Mueller <lmuelle@suse.de>
#
# /etc/named.conf
#
# This is a sample configuration file for the name server BIND 9.  It works as
# a caching only name server without modification.
#
# A sample configuration for setting up your own domain can be found in
# /usr/share/doc/packages/bind/sample-config.
#
# A description of all available options can be found in
# /usr/share/doc/packages/bind/misc/options.

options {

        # The directory statement defines the name server's working directory

        directory "/var/lib/named";

        # Write dump and statistics file to the log subdirectory.  The
        # pathenames are relative to the chroot jail.

        dump-file "/var/log/named_dump.db";
        statistics-file "/var/log/named.stats";

        # The forwarders record contains a list of servers to which queries
        # should be forwarded.  Enable this line and modify the IP address to
        # your provider's name server.  Up to three servers may be listed.

        #forwarders { 192.0.2.1; 192.0.2.2; };

        # Enable the next entry to prefer usage of the name server declared in
        # the forwarders section.
```

```
        #forward first;

        # The listen-on record contains a list of local network interfaces to
        # listen on.  Optionally the port can be specified.  Default is to
        # listen on all interfaces found on your system.  The default port is
        # 53.

        #listen-on port 53 { 127.0.0.1; };

        # The listen-on-v6 record enables or disables listening on IPv6
        # interfaces.  Allowed values are 'any' and 'none' or a list of
        # addresses.

        listen-on-v6 { any; };

        # The next three statements may be needed if a firewall stands between
        # the local server and the internet.

        #query-source address * port 53;
        #transfer-source * port 53;
        #notify-source * port 53;

        # The allow-query record contains a list of networks or IP addresses
        # to accept and deny queries from. The default is to allow queries
        # from all hosts.

        #allow-query { 127.0.0.1; };

        # If notify is set to yes (default), notify messages are sent to other
        # name servers when the the zone data is changed.  Instead of setting
        # a global 'notify' statement in the 'options' section, a separate
        # 'notify' can be added to each zone definition.

        notify no;
};

# To configure named's logging remove the leading '#' characters of the
# following examples.
#logging {
#       # Log queries to a file limited to a size of 100 MB.
#       channel query_logging {
#               file "/var/log/named_querylog"
#                       versions 3 size 100M;
#               print-time yes;                 // timestamp log entries
#       };
#       category queries {
#               query_logging;
#       };
#
#       # Or log this kind alternatively to syslog.
#       channel syslog_queries {
#               syslog user;
#               severity info;
#       };
#       category queries { syslog_queries; };
#
#       # Log general name server errors to syslog.
#       channel syslog_errors {
#               syslog user;
#               severity error;
```

```
#       };
#       category default { syslog_errors;  };
#
#       # Don't log lame server messages.
#       category lame-servers { null; };
#};

# The following zone definitions don't need any modification.  The first one
# is the definition of the root name servers.  The second one defines
# localhost while the third defines the reverse lookup for localhost.

zone "." in {
        type hint;
        file "root.hint";
};

zone "localhost" in {
        type master;
        file "localhost.zone";
};

zone "0.0.127.in-addr.arpa" in {
        type master;
        file "127.0.0.zone";
};

zone "uni-muenchen.de" {
        type forward;
        forward only;
        forwarders {
                192.168.216.254;
        };
};


# Include the meta include file generated by createNamedConfInclude.  This
# includes all files as configured in NAMED_CONF_INCLUDE_FILES from
# /etc/sysconfig/named

include "/etc/named.conf.include";
logging {
        category default { log_syslog; };
        channel log_syslog { syslog; };
};

# You can insert further zone records for your own domains below or create
# single files in /etc/named.d/ and add the file names to
# NAMED_CONF_INCLUDE_FILES.
# See /usr/share/doc/packages/bind/README.SUSE for more details.
```

B.3 Maschinenkonfigurationen

Listing B.15: Konfigurationsdatei des Secservers

```
#  -*- mode: python; -*-
#============================================================================
# Python configuration setup for 'xm create'.
# This script sets the parameters used when a domain is created using 'xm create
    '.
# You use a separate script for each domain you want to create, or
# you can set the parameters for the domain on the xm command line.
#============================================================================

#----------------------------------------------------------------------------
# Kernel image file.
kernel = "/boot/vmlinuz-2.6.11.12-xenU"

# Optional ramdisk.
#ramdisk = "/boot/initrd.gz"

# The domain build function. Default is 'linux'.
#builder='linux'

# Initial memory allocation (in megabytes) for the new domain.
memory = 70

# A name for your domain. All domains must have different names.
name = "server"

# Which CPU to start domain on?
#cpu = -1   # leave to Xen to pick

#----------------------------------------------------------------------------
# Define network interfaces.

# Number of network interfaces. Default is 1.
nics=2

# Optionally define mac and/or bridge for the network interfaces.
# Random MACs are assigned if not given.
vif = [ 'mac=aa:00:00:00:ff:5c,␣bridge=brint',
        'mac=aa:00:00:00:00:5c,␣bridge=brsec00' ]

#----------------------------------------------------------------------------
# Define the disk devices you want the domain to have access to, and
# what you want them accessible as.
# Each disk entry is of the form phy:UNAME,DEV,MODE
# where UNAME is the device, DEV is the device name the domain will see,
# and MODE is r for read-only, w for read-write.

disk = [ 'file:/home/config/server.img,sda1,w',
         'file:/home/SuSE/SuSE-9.3.iso,hdc,r']

#----------------------------------------------------------------------------
# Set the kernel command line for the new domain.
# You only need to define the IP parameters and hostname if the domain's
# IP config doesn't, e.g. in ifcfg-eth0 or via DHCP.
# You can use 'extra' to set the runlevel and custom environment
# variables used by custom rc scripts (e.g. VMID=, usr= ).
```

```
ip="192.168.20.1"
# Set netmask.
netmask="255.255.255.0"
# Set default gateway.
#gateway=""
# Set the hostname.
hostname= "server"

# Set root device.
root = "/dev/sda1"

# Sets runlevel 5.
extra = "5"

#============================================================================
```

Listing B.16: Konfigurationsdatei des Rechners test4all

```
#  -*- mode: python; -*-
#============================================================================
# Python configuration setup for 'xm create'.
# This script sets the parameters used when a domain is created using 'xm create
    '.
# You use a separate script for each domain you want to create, or
# you can set the parameters for the domain on the xm command line.
#============================================================================

#----------------------------------------------------------------------------
# Kernel image file.
kernel = "/boot/vmlinuz-2.6.11.12-xenU"

# Optional ramdisk.
#ramdisk = "/boot/initrd.gz"

# The domain build function. Default is 'linux'.
#builder='linux'

# Initial memory allocation (in megabytes) for the new domain.
memory = 70

# A name for your domain. All domains must have different names.
name = "test4all"

# Which CPU to start domain on?
#cpu = -1   # leave to Xen to pick

#----------------------------------------------------------------------------
# Define network interfaces.

# Number of network interfaces. Default is 1.
nics=2

# Optionally define mac and/or bridge for the network interfaces.
# Random MACs are assigned if not given.
vif = [ 'mac=aa:00:00:00:ff:4a,␣bridge=brmgnt',
        'mac=aa:00:00:00:00:4a,␣bridge=brsec00' ]

#----------------------------------------------------------------------------
# Define the disk devices you want the domain to have access to, and
# what you want them accessible as.
```

```
# Each disk entry is of the form phy:UNAME,DEV,MODE
# where UNAME is the device, DEV is the device name the domain will see,
# and MODE is r for read-only, w for read-write.

disk = [ 'file:/home/config/test4all.img,sda1,w',
         'file:/home/SuSE/SuSE-9.3.iso,hdc,r']

#---------------------------------------------------------------------------
# Set the kernel command line for the new domain.
# You only need to define the IP parameters and hostname if the domain's
# IP config doesn't, e.g. in ifcfg-eth0 or via DHCP.
# You can use 'extra' to set the runlevel and custom environment
# variables used by custom rc scripts (e.g. VMID=, usr= ).

# Set ip-adress
ip="192.168.10.253"
# Set netmask.
netmask="255.255.255.0"
# Set default gateway.
#gateway=""
# Set the hostname.
hostname= "test4all"

# Set root device.
root = "/dev/sda1"

# Sets runlevel 5.
extra = "5"

#===========================================================================
```

Listing B.17: Konfigurationsdatei des Rechners hacktest

```
#  -*- mode: python; -*-
#============================================================================
# Python configuration setup for 'xm create'.
# This script sets the parameters used when a domain is created using 'xm create
    '.
# You use a separate script for each domain you want to create, or
# you can set the parameters for the domain on the xm command line.
#============================================================================

#----------------------------------------------------------------------------
# Kernel image file.
kernel = "/boot/vmlinuz-2.6.11.12-xenU"

# Optional ramdisk.
#ramdisk = "/boot/initrd.gz"

# The domain build function. Default is 'linux'.
#builder='linux'

# Initial memory allocation (in megabytes) for the new domain.
memory = 70

# A name for your domain. All domains must have different names.
name = "hacktest"

# Which CPU to start domain on?
#cpu = -1   # leave to Xen to pick
```

```
#-----------------------------------------------------------------------------
# Define network interfaces.

# Number of network interfaces. Default is 1.
nics=2

# Optionally define mac and/or bridge for the network interfaces.
# Random MACs are assigned if not given.
vif = [ 'mac=aa:00:00:00:ff:ac,␣bridge=brmgnt',
        'mac=aa:00:00:00:00:ac,␣bridge=brsec00' ]

#-----------------------------------------------------------------------------
# Define the disk devices you want the domain to have access to, and
# what you want them accessible as.
# Each disk entry is of the form phy:UNAME,DEV,MODE
# where UNAME is the device, DEV is the device name the domain will see,
# and MODE is r for read-only, w for read-write.

disk = [ 'file:/home/config/hacktest.img,sda1,w',
         'file:/home/SuSE/SuSE-9.3.iso,hdc,r']

#-----------------------------------------------------------------------------
# Set the kernel command line for the new domain.
# You only need to define the IP parameters and hostname if the domain's
# IP config doesn't, e.g. in ifcfg-eth0 or via DHCP.
# You can use 'extra' to set the runlevel and custom environment
# variables used by custom rc scripts (e.g. VMID=, usr= ).

# Set if you want dhcp to allocate the IP address.
#dhcp="dhcp"
# Set ip-adress
ip="192.168.10.252"
# Set netmask.
netmask="255.255.255.0"
# Set default gateway.
#gateway="192.168.10.254"
# Set the hostname.
hostname= "hacktest"

# Set root device.
root = "/dev/sda1"

# Root device for nfs.
#root = "/dev/nfs"
# The nfs server.
#nfs_server = '169.254.1.0'
# Root directory on the nfs server.
#nfs_root   = '/full/path/to/root/directory'

# Sets runlevel 5.
extra = "5"

#-----------------------------------------------------------------------------
# Set according to whether you want the domain restarted when it exits.
# The default is 'onreboot', which restarts the domain when it shuts down
# with exit code reboot.
# Other values are 'always', and 'never'.

#restart = 'onreboot'
```

```
#============================================================================
```

Listing B.18: Konfigurationsdatei der Clients im ersten Szenario

```
#  -*- mode: python; -*-
#============================================================================
# Python configuration setup for 'xm create'.
# This script sets the parameters used when a domain is created using 'xm create
    '.
# You use a separate script for each domain you want to create, or
# you can set the parameters for the domain on the xm command line.
#============================================================================

#----------------------------------------------------------------------------
# Kernel image file.
kernel = "/boot/vmlinuz-2.6.11.12-xenU"

# Optional ramdisk.
#ramdisk = "/boot/initrd.gz"

# The domain build function. Default is 'linux'.
#builder='linux'

# Initial memory allocation (in megabytes) for the new domain.
memory = 70

# A name for your domain. All domains must have different names.
name = "pcsec%02d" % int(vmid)

# Which CPU to start domain on?
#cpu = -1   # leave to Xen to pick

#----------------------------------------------------------------------------
# Define network interfaces.

# Number of network interfaces. Default is 1.
if (int(vmid) % 5 == 0):
        nics=3
else:
        nics=2

# Optionally define mac and/or bridge for the network interfaces.
# Random MACs are assigned if not given.
vif = ['mac=aa:00:00:00:ff:%02d,␣bridge=brmgnt' % int(vmid),
        'mac=aa:00:00:00:%02d:%02d,␣bridge=brsec%02d' % ((((int(vmid)-1)/5)+1),
            int(vmid),(((int(vmid)-1)/5)+1)) ]
if (int(vmid) % 5 == 0):
        vif.insert(1,'mac=aa:00:00:00:00:%02d,␣bridge=brsec00' % int(vmid))


#----------------------------------------------------------------------------
# Define the disk devices you want the domain to have access to, and
# what you want them accessible as.
# Each disk entry is of the form phy:UNAME,DEV,MODE
# where UNAME is the device, DEV is the device name the domain will see,
# and MODE is r for read-only, w for read-write.

disk = [ 'file:/home/images/pcsec%02d.img,sda1,w' % int(vmid),
         'file:/home/SuSE/SuSE-9.3.iso,hdc,r']

#----------------------------------------------------------------------------
```

```
# Set the kernel command line for the new domain.
# You only need to define the IP parameters and hostname if the domain's
# IP config doesn't, e.g. in ifcfg-eth0 or via DHCP.
# You can use 'extra' to set the runlevel and custom environment
# variables used by custom rc scripts (e.g. VMID=, usr= ).

# Set ip-adress
ip="192.168.10.%s" % vmid
# Set netmask.
netmask="255.255.255.0"
# Set default gateway.
#gateway=""
# Set the hostname.
hostname= "pcsec%02d" % int(vmid)

# Set root device.
root = "/dev/sda1"

# Sets runlevel 5.
extra = "5"

#============================================================================
```

Listing B.19: Konfigurationsdatei der Clients im zweiten Szenario

```
#  -*- mode: python; -*-
#============================================================================
# Python configuration setup for 'xm create'.
# This script sets the parameters used when a domain is created using 'xm create
    '.
# You use a separate script for each domain you want to create, or
# you can set the parameters for the domain on the xm command line.
#============================================================================

#----------------------------------------------------------------------------
# Kernel image file.
kernel = "/boot/vmlinuz-2.6.11.12-xenU"

# Optional ramdisk.
#ramdisk = "/boot/initrd.gz"

# The domain build function. Default is 'linux'.
#builder='linux'

# Initial memory allocation (in megabytes) for the new domain.
memory = 70

# A name for your domain. All domains must have different names.
name = "pcsec%02d" % int(vmid)

# Which CPU to start domain on?
#cpu = -1   # leave to Xen to pick

#----------------------------------------------------------------------------
# Define network interfaces.

# Number of network interfaces. Default is 1.
if (int(vmid) % 2 == 1):
        nics=3
else:
        nics=2
```

```
# Optionally define mac and/or bridge for the network interfaces.
# Random MACs are assigned if not given.
vif = ['mac=aa:00:00:00:ff:%02d,␣bridge=brmgnt' % int(vmid),
        'mac=aa:00:00:00:%02d:%02d,␣bridge=brsec%02d' % ((((int(vmid)-1)/2)+1),
            int(vmid),(((int(vmid)-1)/2)+1)) ]
if (int(vmid) % 2 == 1):
        vif.insert(1,'mac=aa:00:00:00:00:%02d,␣bridge=brsec00' % int(vmid))


#============================================================================
# Define the disk devices you want the domain to have access to, and
# what you want them accessible as.
# Each disk entry is of the form phy:UNAME,DEV,MODE
# where UNAME is the device, DEV is the device name the domain will see,
# and MODE is r for read-only, w for read-write.

disk = [ 'file:/home/images/pcsec%02d.img,sda1,w' % int(vmid),
         'file:/home/SuSE/SuSE-9.3.iso,hdc,r']

#============================================================================
# Set the kernel command line for the new domain.
# You only need to define the IP parameters and hostname if the domain's
# IP config doesn't, e.g. in ifcfg-eth0 or via DHCP.
# You can use 'extra' to set the runlevel and custom environment
# variables used by custom rc scripts (e.g. VMID=, usr= ).

# Set ip-adress
ip="192.168.10.%s" % vmid
# Set netmask.
netmask="255.255.255.0"
# Set default gateway.
#gateway=""
# Set the hostname.
hostname= "pcsec%02d" % int(vmid)

# Set root device.
root = "/dev/sda1"

# Sets runlevel 5.
extra = "5"

#============================================================================
```

B.4 Szenarien

Listing B.20: Startskript für das Szenario 1

```
#! /bin/sh

# Schalter fuer Webserver auf Szenario1 setzen
echo 1 > /home/config/szenario

# Management-Bridge erzeugen
brctl addbr brmgnt

# Interne Bridge erzeugen
brctl addbr brint

# Andere Bridges erzeugen
i=0
while test $i -le 8
        do
                brctl addbr brsec0$i
                brctl setageing brsec0$i 0
                ifconfig brsec0$i up
                let i=$i+1
        done

# Virtuelle Maschinen starten
i=1
while test $i -le 40
        do
                echo starte Rechner pcsec$i
                xm create szenario_1/pcsec vmid=$i
                let i=$i+1
        done

echo Netzwerkkonfiguration fÃ¼r Hackest setzen
mount -o loop /home/images/hacktest.img /home/images/mnt_hacktest
cp /home/config/szenario_1/routes.sz1 /home/images/mnt_hacktest/etc/sysconfig/
    network/routes
cp /home/config/szenario_1/ifcfg-eth1.hacktest.sz1 /home/images/mnt_hacktest/etc/
    sysconfig/network/ifcfg-eth1
umount /home/images/hacktest.img
echo starte Rechner hacktest
xm create hacktest

echo Netzwerkkonfiguration fÃ¼r test4all setzen
mount -o loop /home/images/test4all.img /home/images/mnt_test4all
cp /home/config/szenario_1/routes.sz1 /home/images/mnt_test4all/etc/sysconfig/
    network/routes
cp /home/config/szenario_1/ifcfg-eth1.test4all.sz1 /home/images/mnt_test4all/etc/
    sysconfig/network/ifcfg-eth1
umount /home/images/test4all.img
echo starte Rechner test4all
xm create test4all

echo Netzwerkkonfiguration fÃ¼r Secserver setzen
mount -o loop /home/images/server.img /home/images/mnt_server
cp /home/config/szenario_1/routes.server.sz1 /home/images/mnt_server/etc/
    sysconfig/network/routes
cp /home/config/szenario_1/ifcfg-eth1.server.sz1 /home/images/mnt_server/etc/
    sysconfig/network/ifcfg-eth1
```

```
cp /home/config/szenario_1/secp.nm.ifi.lmu.de.server.sz1 /home/images/mnt_server/
    var/lib/named/master/secp.nm.ifi.lmu.de
cp /home/config/szenario_1/secp.nm.informatik.uni-muenchen.de.server.sz1 /home/
    images/mnt_server/var/lib/named/master/secp.nm.informatik.uni-muenchen.de
umount /home/images/server.img
echo starte Rechner Secserver
xm create server

echo starte Rechner login
xm create login


echo Firewall Regeln erzeugen

/etc/init.d/firewall.sh.domain0

i=0
while test $i -le 8
        do
                ifconfig brsec0$i up
                iptables -A FORWARD -i brsec0$i -o brsec0$i -j ACCEPT
                let i=$i+1
        done

idLogin=`xm domid login`
port=vif${idLogin}.0
ifconfig brmgnt up

iptables -A FORWARD -m physdev --physdev-in $port --physdev-out ! $port -j ACCEPT
iptables -A FORWARD -m physdev --physdev-in ! $port --physdev-out $port -j ACCEPT

ifconfig brint 192.168.20.253 up
iptables -A FORWARD -i brint -o brint -j ACCEPT
iptables -A INPUT -i brint -s 192.168.20.254 -p tcp --dport 80 -m state --state
    NEW -j ACCEPT
```

Listing B.21: Stoppskript für das Szenario 1

```
#! /bin/sh

# Rechner runterfahren
i=1
while test $i -le 40
        do
                if test $i -le 9
                        then xm shutdown pcsec0$i
                        else xm shutdown pcsec$i
                fi
                let i=$i+1
        done
xm shutdown hacktest
xm shutdown test4all
xm shutdown server
xm shutdown login

# Bridges entfernen
i=0
while test $i -le 8
        do
                ifconfig brsec0$i down
                brctl delbr brsec0$i
```

```
                let i=$i+1
        done

# Management Bridge entfernen
ifconfig brmgnt down
brctl delbr brmgnt

# Interne Bridge entfernen
ifconfig brint down
brctl delbr brint

# Firewall wieder herstellen
/etc/init.d/firewall.sh.domain0
```

Listing B.22: Startskript für das Szenario 2

```
#! /bin/sh

# Schalter fuer Webserver auf Szenario1 setzen
echo 2 > /home/config/szenario

# Management-Bridge erzeugen
brctl addbr brmgnt

# Interne Bridge erzeugen
brctl addbr brint

# Andere Bridges erzeugen
i=0
while test $i -le 20
        do
                if test $i -le 9
                        then brctl addbr brsec0$i
                        else brctl addbr brsec$i
                fi
                let i=$i+1
        done

# Virtuelle Maschinen starten
i=1
while test $i -le 40
        do
                echo starte Rechner pcsec$i
                xm create szenario_2/pcsec vmid=$i
                let i=$i+1
        done

echo Netzwerkkonfiguration fÃ¼r Hackest setzen
mount -o loop /home/images/hacktest.img /home/images/mnt_hacktest
cp /home/config/szenario_2/routes.sz2 /home/images/mnt_hacktest/etc/sysconfig/
    network/routes
cp /home/config/szenario_2/ifcfg-eth1.hacktest.sz2 /home/images/mnt_hacktest/etc/
    sysconfig/network/ifcfg-eth1
umount /home/images/hacktest.img
echo starte Rechner hacktest
xm create hacktest

echo Netzwerkkonfiguration fÃ¼r test4all setzen
mount -o loop /home/images/test4all.img /home/images/mnt_test4all
cp /home/config/szenario_2/routes.sz2 /home/images/mnt_test4all/etc/sysconfig/
    network/routes
```

```
cp /home/config/szenario_2/ifcfg-eth1.test4all.sz2 /home/images/mnt_test4all/etc/
    sysconfig/network/ifcfg-eth1
umount /home/images/test4all.img
echo starte Rechner test4all
xm create test4all

echo Netzwerkkonfiguration fÃ¼r Secserver setzen
mount -o loop /home/images/server.img /home/images/mnt_server
cp /home/config/szenario_2/routes.server.sz2 /home/images/mnt_server/etc/
    sysconfig/network/routes
cp /home/config/szenario_2/ifcfg-eth1.server.sz2 /home/images/mnt_server/etc/
    sysconfig/network/ifcfg-eth1
cp /home/config/szenario_2/secp.nm.ifi.lmu.de.server.sz2 /home/images/mnt_server/
    var/lib/named/master/secp.nm.ifi.lmu.de
cp /home/config/szenario_2/secp.nm.informatik.uni-muenchen.de.server.sz2 /home/
    images/mnt_server/var/lib/named/master/secp.nm.informatik.uni-muenchen.de
umount /home/images/server.img
echo starte Rechner Secserver
xm create server

echo starte Rechner login
xm create login


echo Firewall Regeln erzeugen

/etc/init.d/firewall.sh.domain0

i=0
while test $i -le 20
        do
                if test $i -le 9
                        then j=0$i
                        else j=$i
                fi
                ifconfig brsec$j up
                iptables -A FORWARD -i brsec$j -o brsec$j -j ACCEPT
                let i=$i+1
        done

idLogin=`xm domid login`
port=vif${idLogin}.0
ifconfig brmgnt up

iptables -A FORWARD -m physdev --physdev-in $port --physdev-out ! $port -j ACCEPT
iptables -A FORWARD -m physdev --physdev-in ! $port --physdev-out $port -j ACCEPT

ifconfig brint 192.168.20.253 up
iptables -A FORWARD -i brint -o brint -j ACCEPT
iptables -A INPUT -i brint -s 192.168.20.254 -p tcp --dport 80 -m state --state
    NEW -j ACCEPT
```

Listing B.23: Stoppskript für das Szenario 2

```
# Rechner runterfahren
i=1
while test $i -le 40
        do
                if test $i -le 9
                        then xm shutdown pcsec0$i
                        else xm shutdown pcsec$i
```

```
                fi
                let i=$i+1
        done

xm shutdown hacktest
xm shutdown test4all
xm shutdown server
xm shutdown login

# Bridges entfernen
i=0
while test $i -le 20
        do
                if test $i -le 9
                        then ifconfig brsec0$i down
                             brctl delbr brsec0$i
                        else ifconfig brsec$i down
                             brctl delbr brsec$i
                fi
                let i=$i+1
        done

# Management Bridge entfernen
ifconfig brmgnt down
brctl delbr brmgnt

# Interne Bridge entfernen
ifconfig brint down
brctl delbr brint

# Firewall Regeln zurksetzen
/etc/init.d/firewall.sh.domain0
```

B.5 Virtual Maschine Management Tool

Listing B.24: index.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org
    /TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

        <head>
                <meta http-equiv="content-type" content="text/html;charset=utf-8"
                     />
                <title>Praktikum IT-Sicherheit</title>
                <meta name="author" content="Tobias Lindinger" />
                <meta http-equiv="refresh" content="0;URL=/xen/state/index.psp" /
                    >
        </head>

        <body>
                <p></p>
        </body>

</html>
```

Listing B.25: index.psp

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org
    /TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

        <head>
                <meta http-equiv="content-type" content="text/html;charset=utf-8"
                     />
                <meta name="generator" content="Adobe GoLive" />
                <title>Praktikum IT-Sicherheit - Virtual Maschine Management Tool
                    </title>
                <script language="JavaScript" type="text/javascript">
                <!-- Begin
                        function popUp(URL){
                                day = new Date();
                                id = day.getTime();
                                eval("page" + id + " = window.open(URL, '" + id +
                                     "', 'toolbar=0,scrollbars=0,location=0,
                                    statusbar=0,menubar=0,resizable=0,width=800,
                                    height=500,left=650,top=500');");
                        }
                // End -->
                </script>
        </head>

        <body>
                <p></p>
                <table width="90%" border="0" cellspacing="2" cellpadding="0">
                        <tr>
                                <td colspan="3">
                                        <div align="center">
                                                <h1>Praktikum IT-Sicherheit</h1>
                                        </div>
                                </td>
```

```
                        </tr>
                        <tr>
                                <td>
                                        <div align="center">
                                                <img src="tux.jpg" alt="tux"
                                                    height="237" width="200" />
                                        </div>
                                </td>
                                <td colspan="2">
                                        <div align="center">
                                                <h2>Virtual Maschine Management
                                                    Tool</h2>
                                        </div>
                                </td>
                        </tr>
                </table>
                <div align="center">

<form id="FormName" action="(EmptyReference!)" method="get" name="FormName">

<% from xen.xend.XendClient import server %>
<% from xen.xend import sxp %>
<% from xen.xend import PrettyPrint %>
<% import types %>
<%

def getDomInfoHash ( domain ):
        domInfoHash = {}
        try:
                domInfoHash = sxp2hash( server.xend_domain( domain ) )
                domInfoHash['dom'] = domain
        except:
                domInfoHash['name'] = "Error getting domain details"
        return domInfoHash

def sxp2hash( s ):
        sxphash = {}

        for child in sxp.children( s ):
                if isinstance( child, types.ListType ) and len( child ) > 1:
                        if isinstance( child[1], types.ListType ) and len( child
                            ) > 1:
                                sxphash[ child[0] ] = sxp2hash( child[1] )
                        else:
                                sxphash[ child[0] ] = child[1]
        return sxphash

def writeExDom( domInfoHash ):

        req.write('<tr>')

        req.write('<td>')
        req.write(str(domInfoHash['id']))
        req.write('</td>')

        req.write('<td>')
        req.write('<b>')
        req.write(str(domInfoHash['name']))
        req.write('</b>')
        req.write('</td>')
```

```
        req.write('<td>')
        req.write(str(domInfoHash['memory']))
        req.write('</td>')

        req.write('<td>')
        #req.write(str(domInfoHash['state']))
        if domInfoHash['state']=="-b---" or domInfoHash['state']=="r----":
                req.write('<img src="ampel-gruen.gif" alt="ampel-gruen" />')
        if domInfoHash['state']=="--p--":
                req.write('<img src="ampel-gelb.gif" alt="ampel-gruen" />')
        req.write('</td>')

        req.write('<td>')
        req.write('<button name="' + domInfoHash['name'] + '-reboot"  type="
            button" onClick="javascript:popUp('+"'"+'../action/request.py/
            reboot?target='+domInfoHash['id']+"'"+')">Reboot</button>')
        req.write('</td>')

        req.write('<td>')
        req.write('<button name="' + domInfoHash['name'] + '-destroy" type="
            button" onClick="javascript:popUp('+"'"+'../action/request.py/
            shutdown?target='+domInfoHash['id']+"'"+')">Shutdown</button>
            ')
        req.write('</td>')

        req.write('<td>')
        req.write('<button name="' + domInfoHash['name'] + '-destroy" type="
            button" onClick="javascript:popUp('+"'"+'../action/request.py/
            destroy?target='+domInfoHash['id']+"'"+')">Destroy</button>')
        req.write('</td>')

        req.write('<td>')
        req.write('<button name="' + domInfoHash['name'] + '-destroy" type="
            button" onClick="javascript:popUp('+"'"+'../action/request.py/
            pause?target='+domInfoHash['id']+"'"+')">Pause</button>')
        req.write('</td>')

        req.write('<td>')
        req.write('<button name="' + domInfoHash['name'] + '-destroy" type="
            button" onClick="javascript:popUp('+"'"+'../action/request.py/
            unpause?target='+domInfoHash['id']+"'"+')">Unpause</button>')
        req.write('</td>')

        req.write('<td>')
        req.write('<button name="' + domInfoHash['name'] + '-ovpn" type="button"
            onClick="javascript:popUp('+"'"+'../keys/'+domInfoHash['name']+
            "+".tar.gz'"+')">OpenVPN</button>')
        req.write('</td>')


        req.write('</tr>')


def writeNonexDom( name ):

        req.write('<tr>')

        req.write('<td>')
        req.write('--')
        req.write('</td>')
```

```
        req.write('<td>')
        req.write('<b>')
        req.write( str( name ) )
        req.write('</b>')
        req.write('</td>')

        req.write('<td>')
        req.write('--')
        req.write('</td>')

        req.write('<td>')
        req.write('<img src="ampel-rot.gif" alt="ampel-rot" />')
        req.write('</td>')

        req.write('<td>')
        req.write('<button name="' + name + '-create" type="button" onClick="
            javascript:popUp('+"'"+'../action/request.py/create?target='+
            str(name)+"'"+')">Create</button>')
        req.write('</td>')

        req.write('<td>')
        req.write('<button name="' + name + '-backup" type="button" onClick="
            javascript:popUp('+"'"+'../action/request.py/backup?target='+
            str(name)+"'"+')">Backup</button>')
        req.write('</td>')

        req.write('<td>')
        req.write('<button name="' + name + '-restore" type="button" onClick="
            javascript:popUp('+"'"+'../action/request.py/restore?target='+
            str(name)+"'"+')">Restore</button>')
        req.write('</td>')

        req.write('<td>')
        req.write('<button name="' + name + '-cleanImage" type="button" onClick="
            javascript:popUp('+"'"+'../action/request.py/cleanImage?target='
            +str(name)+"'"+')">Clean Image</button>')
        req.write('</td>')

        req.write('<td>')
        req.write('</td>')

        req.write('</tr>')

domains = server.xend_domains()
domains.sort()
list=range(0,45)
for domain in domains:
        domInfoHash = getDomInfoHash( domain )
        tmp=str(domInfoHash['name'])
        nr=0
        if tmp[:5]=="pcsec":
                nr=tmp[5:]
                # if nr[0:1]=="0":
                #       nr=nr[1:2]
                nr=int(nr)
        if tmp=="server":
                nr=41
        if tmp=="hacktest":
                nr=42
        if tmp=="test4all":
                nr=43
```

```
        if tmp=="login":
                nr=44
        if nr==0:
                continue
        list[nr]=domInfoHash
req.write('<table cellspacing="2">')
req.write('<tr>')
req.write('<td><i>ID</i></td>')
req.write('<td><i>Name</i></td>')
req.write('<td><i>RAM</i></td>')
req.write('<td><i>Status</i></td>')
req.write('</tr>')
for domain in list:
        if domain in range(0,45):
                if domain == 0:
                        continue
                if domain <= 40:
                        name="pcsec" + str(domain)
                if domain <= 9:
                        name="pcsec0" + str(domain)
                if domain == 41:
                        name="server"
                if domain == 42:
                        name="hacktest"
                if domain == 43:
                        name="test4all"
                if domain == 44:
                        name="login"

                writeNonexDom( name )
        else:
                writeExDom( domain )


req.write('</table>')

%>




</form>
</div>
                <p></p>
        </body>

</html>
```

Listing B.26: request.py

```
def domid(target):
        import subprocess
        try:
                p=subprocess.Popen("/usr/bin/sudo␣/usr/sbin/xm␣domid␣" + target,
                    shell=True, stdout=subprocess.PIPE, close_fds=True)
                r=p.wait()
                if r=='0':
                        result=p.stdout.readlines[0]
                else:
                        result=-1
        except:
```

```
                result=-1
        return result


def reboot(target="nothing"):
        result=writePre()
        from xen.xend.XendClient import server
        if  target != 'nothing':
                try:
                        server.xend_domain_shutdown( int ( target ), "reboot" )
                except:
                        result=result + ' Rebooting Domain' + target + ' failed'
                result=result + ' Rebooting Domain ' + target + ' succeeded'
        else:
                result=result + ' Rebooting Domain' + target + ' failed'
        return result


def shutdown(target="nothing"):
        result=writePre()
        from xen.xend.XendClient import server
        if  target != 'nothing':
                try:
                        server.xend_domain_shutdown( int ( target ), "halt" )
                except:
                        result=result + ' Shutting down Domain' + target + '
                            failed'
                result=result + ' Shutting down Domain ' + target + ' succeeded'
        else:
                result=result + ' Shutting down Domain' + target + ' failed'
        result=result + ' ' + writePost()
        return result


def destroy(target="nothing"):
        result=writePre()
        from xen.xend.XendClient import server
        if  target != 'nothing':
                try:
                        server.xend_domain_destroy( int ( target ), "halt" )
                except:
                        result=result + ' Destroying Domain' + target + ' failed'
                result=result + ' Destroying Domain ' + target + ' succeeded'
        else:
                result=result + ' Destroying Domain' + target + ' failed'
        return result


def pause(target="nothing"):
        result=writePre()
        from xen.xend.XendClient import server
        if  target != 'nothing':
                try:
                        server.xend_domain_pause( int ( target ) )
                except:
                        result=result + ' Pausing Domain' + target + ' failed'
                result=result + ' Pausing Domain ' + target + ' succeeded'
        else:
                result=result + ' Pausing Domain' + target + ' failed'
        return result
```

```
def unpause(target="nothing"):
        result=writePre()
        from xen.xend.XendClient import server
        if  target != 'nothing':
                try:
                        server.xend_domain_unpause( int ( target )  )
                except:
                        result=result + ' Unpausing Domain' + target + ' failed'
                result=result + ' Unpausing Domain ' + target + ' succeeded'
        else:
                result=result + ' Unpausing Domain' + target + ' failed'
        return result


def create(target="nothing"):
        result=writePre()
        import subprocess
        if  target != 'nothing':
               try:
                        p=subprocess.Popen("/usr/bin/sudo /usr/bin/less /home/
                            config/szenario", shell=True, stdout=subprocess.PIPE,
                             close_fds=True)
                        if p.wait()==0:
                                r=p.stdout.read()
                        else:
                                result=result + ' No szenario specified; 
                                    operation aborted!'
                                result=result + writePost()
                                return result
                        if target[:5]=='pcsec':
                              p=subprocess.Popen("/usr/bin/sudo /usr/sbin/xm 
                                  create /home/config/szenario_" + r[0] + "/pcsec
                                  vmid=" + str(int(target[5:])), shell=True,
                                  stdout=subprocess.PIPE, close_fds=True)
                              r=p.wait()
                        else:
                              p=subprocess.Popen("/usr/bin/sudo /usr/sbin/xm 
                                  create /home/config/" + target, shell=True,
                                  stdout=subprocess.PIPE, close_fds=True)
                              r=p.wait()
                        if r==0:
                                result=result + ' Creating domain ' + target + ' 
                                    succeeded'
                                if target[:5]=='login':
                                        port=getid('login')
                                        p=subprocess.Popen("usr/bin/sudo /usr/
                                            sbin/iptables -R FORWARD 2 -m physdev
                                            --physdev-in " + port + " --physdev-
                                            out ! " + port + " -j ACCEPT", shell=
                                            True, stdout=subprocess.PIPE,
                                            close_fds=True)
                                        r1==p.wait()
                                        p=subprocess.Popen("usr/bin/sudo /usr/
                                            sbin/iptables -R FORWARD 3 -m physdev
                                            --physdev-in ! " + port + " --
                                            physdev-out " + port + " -j ACCEPT",
                                            shell=True, stdout=subprocess.PIPE,
                                            close_fds=True)
                                        r2==p.wait()
                                        if r1==0 and r2==0:
```

```
                                        result=result + '<br /> Firewall
                                            rules adapted successfully'
                                else:
                                        result=result + '<br /> Adapting
                                            firewall rules failed'
                        else:
                                result=result + ' Creating Domain ' + target + '
                                    failed '
                except:
                        result=result + ' Creating Domain ' + target + ' failed
                            !!!!!!!!!!!!!!'
        else:
                result=result + ' Creating Domain ' + target + ' failed'
        result=result + writePost()
        return result


def backup(target="nothing"):
        result=writePre()
        import subprocess
        if target != 'nothing':
                try:
                        p=subprocess.Popen("/usr/bin/sudo /bin/tar zcf /home/
                            backup/" + target + ".tar.gz /home/images/" + target
                            + ".img", shell=True, stdout=subprocess.PIPE,
                            close_fds=True)
                        if p.wait()==0:
                                result=result + 'Backup Creation of ' + target +
                                    ' succeeded'
                                result=result + writePost()
                                return result
                        else:
                                result=result + ' Backup of ' + target + ' failed
                                    '
                except:
                        result=result + ' Backup Creation of ' + target + '
                            failed'
        else:
                result=result + ' Backup Creation of ' + target + ' failed'
        result=result + writePost()
        return result


def restore(target="nothing"):
        result=writePre()
        import subprocess
        if target != 'nothing':
                try:
                                        p=subprocess.Popen("/usr/bin/sudo /bin/
                                            tar zxf /home/backup/" + target + "
                                            .tar.gz", shell=True, stdout=
                                            subprocess.PIPE, close_fds=True)
                        if p.wait()==0:
                                result=result + 'Restore of ' + target + '
                                    succeeded'
                                result=result + writePost()
                                return result
                        else:
                                result=result + ' Restore of ' +  target + '
                                    failed'
                except:
```

```
                        result=result + ' Restore of ' + target + ' failed'
        else:
                result=result + ' Restore of ' + target + ' failed'
        result=result + writePost()
        return result


def cleanImage(target="nothing"):
        result=writePre()
        import subprocess
        if target!= 'nothing':
                try:
                        if target[:5]=='pcsec':
                                p=subprocess.Popen("/usr/bin/sudo /bin/cp /home/
                                    images/pcsec-default.img /home/images/" +
                                    target + ".img" , shell=True, stdout=
                                    subprocess.PIPE, close_fds=True)
                                if p.wait()==0:
                                                result=result + ' Clean Image
                                                    restored successfully'
                                                        result=result +
                                                            writePost()
                                                        return result
                        else:
                                                    result='operation not
                                                        supported for special
                                                        target ' + target
                except:
                        result=result + ' Restoring  Clean Image failed'
        else:
                result=result + ' Restoring  Clean Image failed'
        result=result + writePost()
        return result

def writePre( ):
        return '<html>'

def writePost( ):
        return '</html>'
```

Listing B.27: auth.py

```
from mod_python import apache
import subprocess
import crypt
from xen.xend.XendClient import server
from xen.xend import sxp
from xen.xend import PrettyPrint
import types

def getDomInfoHash ( domain ):
  domInfoHash = {}
  try:
    domInfoHash = sxp2hash( server.xend_domain( domain ) )
    domInfoHash['dom'] = domain
  except:
    domInfoHash['name'] = "Error getting domain details"
  return domInfoHash

def sxp2hash( s ):
```

```
  sxphash = {}
  for child in sxp.children( s ):
    if isinstance( child, types.ListType ) and len( child ) > 1:
      if isinstance( child[1], types.ListType ) and len( child ) > 1:
        sxphash[ child[0] ] = sxp2hash( child[1] )
      else:
        sxphash[ child[0] ] = child[1]
  return sxphash


def authenhandler(req):
  password = ''
  pw=req.get_basic_auth_pw()
  user=req.user
  if req.uri[-7:]=='.tar.gz' and pw=='secp':
    return apache.HTTP_UNAUTHORIZED
  if req.uri[-7:]=='.tar.gz':
    id=req.uri[-14:-7]
  else:
    id=req.args[7:]
  check=0
  if req.uri[-6:]=='create' or req.uri[-6:]=='backup' or req.uri[-7:]=='restore'
      or req.uri[-10:]=='cleanImage':
    if user==id:
      check=1
  if user=='root':
     p=subprocess.Popen("/usr/bin/sudo␣/usr/bin/less␣/etc/shadow", shell=True,
         stdout=subprocess.PIPE, close_fds=True)
     p.wait()
     get=p.stdout
     for line in get.readlines():
       if line[:4] == 'root':
         password = line[5:18]
         if password=='':
           return apache.HTTP_UNAUTHORIZED
         if password == crypt.crypt(pw,password[:2]):
           return apache.OK
         else:
           return apache.HTTP_UNAUTHORIZED
  domains = server.xend_domains()
  for domain in domains:
    domInfoHash = getDomInfoHash( domain )
    if id==user:
      check=1
    if str(domInfoHash['id']) == id:
      if user==str(domInfoHash['name']):
        check=1
      else:
        return apache.HTTP_UNAUTHORIZED
  if not check:
    return apache.HTTP_UNAUTHORIZED
  p=subprocess.Popen("/usr/bin/sudo␣/bin/mkdir␣/home/images/mnt_" + user, shell=
      True, stdout=subprocess.PIPE, close_fds=True)
  p.wait()
  p=subprocess.Popen("/usr/bin/sudo␣/bin/mount␣-o␣nosuid\,nodev\,loop␣/home/
      images/" + user + ".img␣/home/images/mnt_" + user, shell=True, stdout=
      subprocess.PIPE, close_fds=True)
  p.wait()
  p=subprocess.Popen("/usr/bin/sudo␣/usr/bin/less␣/home/images/mnt_" + user + "/
      etc/shadow", shell=True, stdout=subprocess.PIPE, close_fds=True)
  p.wait()
```

```
  get=p.stdout
  for line in get.readlines():
    if line[:4] == 'root':
      password = line[5:18]
  p=subprocess.Popen("/usr/bin/sudo␣/bin/umount␣/home/images/mnt_" + user, shell=
      True, stdout=subprocess.PIPE, close_fds=True)
  p.wait()
  if password=='':
    return apache.HTTP_UNAUTHORIZED
  if password == crypt.crypt(pw,password[:2]):
    return apache.OK
  else:
    return apache.HTTP_UNAUTHORIZED
```

C Hinweise zur Benutzung der Infrastruktur

MNM
TEAM
Munich Network Management Team

Ludwig-Maximilians-Universität München
und Technische Universität München
Prof. Dr. H.-G. Hegering

Praktikum IT-Sicherheit
Hinweise für die Benutzung der virtuellen Infrastruktur

Seit dem Sommersemester 2006 wird das Praktikum IT-Sicherheit nicht mehr auf real existierenden Rechnern durchgeführt, sondern innerhalb einer virtuellen Umgebung. Die einzelnen Rechner sowie das gesamte Netzwerk wurden mit Hilfe von Xen komplett virtualisiert. Die Funktionalität der Rechner wird hierdurch nicht beeinflusst, der Unterschied zu existierenden Rechnern ist lediglich, dass virtuelle Maschinen nicht direkt bedient werden können, sondern ein dritter Rechner z.B. im CIP-Pool oder zu Hause zur Steuerung verwendet wird.

1. **Verbindung ins Praktikumsnetz**

   Um auf den Rechnern `pcsec`*XX* des Praktikums arbeiten zu können sind einige Vorbereitungen nötig. Als derzeit einfachster Zugang in das Praktikumsnetz dient OpenVPN. Unter `http://openvpn.net` existieren Clients für alle gängigen Windows und Linux Betriebssysteme. Häufig ist OpenVPN aber schon in Linux Distributionen integriert. Eine Installationsanleitung befindet sich auf der Projekthomepage. Die Konfigurationsdatei für den Client sowie die zur Verbindung nötigen Schlüssel werden in der Vorlesung vergeben. Die Einwahl in das Netz via VPN ist von allen Rechnern im Internet aus möglich, sofern keine lokale Firewall die Kommunikation auf dem TCP-Port 1194 verbietet. Nach der Einwahl in das Praktikumsnetz kann der Praktikumsrechner `pcsec`*XX* unter der Adresse `192.168.10.`*XX* erreicht werden. Eine Kommunikation der Praktikumsrechner untereinander sowie der VPN-Clients untereinander ist nicht möglich. Der Zugriff von Praktikumsrechnern auf externe VPN-Clients ist möglich. Dies ermöglicht z.B. das Mounten von Dateifreigaben via NFS oder Samba zur Archivierung von Logfiles oder Konfigurationsdateien für Ausarbeitungen.

   Achtung: Die Freigaben sind grundsätzlich für alle Praktikumsrechner sichtbar, für die Sicherheit der freigegebenen Daten ist jeder selbst verantwortlich!

   Wegen technischer Probleme funktioniert OpenVPN im CIP-Pool derzeit nicht. Als Alternative kann hier mit den folgenden Kommandos ein SSH-Tunnel von einem beliebigen, freien, lokalen Port (hier: 1234) zu einem beliebigen Port (hier: 22) auf der virtuellen Maschine (hier: 192.168.10.1) über den Loginserver aufgebaut werden.

   ```
   ssh -g -L 1234:192.168.10.1:22 login@secplogin.nm.ifi.lmu.de
   ```

   Die Authentisierung am Loginserver ist ausschließlich über ein Zertifikat möglich, dass von der Webseite des Praktikums heruntergeladen werden kann. Anschließend kann in einem zweiten Schritt der aufgebaute Tunnel benutzt werden.

   ```
   ssh -X -p 1234 root@localhost
   ```

   Der oben stehende Befehl veranlasst SSH eine Verbindung als Root zum lokalen Tunnelende aufzubauen und aktiviert das X-Forwarding. Entsprechend modifiziert sind auch Verbindungen mit anderen Protokollen möglich.

2. **Verbindung zu den Praktikumsrechnern**

   Die einzige Möglichkeit zum Arbeiten auf den Praktikumsrechnern ist SSH. Im Bereich SSH existieren

unzählige Clients für nahezu alle Betriebssysteme. Eine bekannte Variante ist OpenSSH. OpenSSH gibt es unter `http://sshwindows.sourceforge.net` für Windows und unter `http://www.openssh.com` für viele andere Betriebssysteme.
Das vorgegebene root-Passwort auf allen PRaktikumsrechnern lautet `secp`. Ändern Sie dieses Passwort nach dem ersten Anmelden unbedingt!

3. **Ich will X!**

   Auch das Arbeiten mit grafischer Ausgabe ist möglich. Zur Ausgabe einzelner Fenster kann das X-Forwarding von SSH mit dem Schalter `-X` gesetzt werden. Dies ist aufgrund der Komplexität des X-Protokolls nur für sehr schnelle Verbindungen (LAN) empfohlen, da der Fensteraufbau ansonsten zur Geduldsprobe wird. Da Windows von sich aus das X-Protokoll nicht versteht muss zur Anwendung dieser Variante unter Windows Cygwin mit X11 Unterstützung installiert sein.
   Eine andere Vorgehensweise ist der Einsatz von NX. NX ist eine sehr performante Implementierung zur Herstellung von Remote Desktop Verbindungen, ähnlich zu VNC oder RDC. Ein NX-Server ist bereits in den Praktikumsrechnern installiert, er muss jedoch vor der Benutzung noch mit dem Kommando

   ```
   nxsetup --install --setup-nomachine-key
   ```

   aktiviert werden. NX-Clients liegen jeder aktuellen Linux Distribution bei. Komfortablere Clients gibt es unter `http://www.nomachine.com`.

4. **Hilfe, mein Rechner streikt!**

   Die virtuellen Praktikumsrechner können wie jeder physische Linux Rechner auch mit den bekannten Kommandos rebootet und heruntergefahren werden. Sollte eine Maschine nicht auf Eingaben reagieren, kann sie über ein webbasiertes Managementinterface neu gestartet werden. Hier können auch Backups der eigenen Maschine angelegt werden. Zur Durchführung dieser Aktionen ist die Autorisierung via Nutzername und Passwort nötig. Der Nutzername entspricht dem Namen des zu administrierenden Rechners, das Passwort ist das dazugehörige Root-Passwort. (Beispiel: Nutzername `pcsec01` und Passwort `secp`) Das Managementinterface ist unter http://192.168.10.254:80 aus dem VPN oder mit entsprechendem SSH-Tunnel erreichbar.

5. **Tipps & Tricks**

   Hier folgen noch einige wichtige Hinweise zur Vermeidung von Problemen.

   - Der Parallelbetrieb von Cygwin, NX-Client und OpenSSH bereitet unter Windows einige Probleme. Diese werden durch unterschiedliche Versionen der Datei `cygwin1.dll` in den Installationsverzeichnissen der genannten Programme verursacht. Um die Tools fehlerfrei miteinander verwenden zu können, müssen alle installierten Versionen der Datei `cygwin1.dll` gegen die aktuellste, installierte Version ausgetauscht werden.
   - Legen Sie niemals eine Konfiguration für die Netzwerkkarte `eth0` in den virtuellen Maschinen an! Das Interface wird beim Booten automatisch richtig konfiguriert. Manuelle Konfigurationen überschreiben die automatische Konfiguration und können die Maschine im schlimmsten Fall unerreichbar machen. In diesem Fall hilft nur noch das Zurückspielen eines hoffentlich angelegten Backups oder eines sauberen Images.
   - Selbst erstellte Firewalls sollten vor der Verlinkung in die Runlevelverzeichnisse unbedingt getestet werden. Lässt die aktivierte Firewall kein SSH auf `eth0` zu, bricht die Managementverbindung zur Maschine ab. Eine Verbindung zur Maschine über SSH zu Konfigurationszwecken ist in diesem Fall nicht mehr möglich. Um eine manuell aktivierte Firewall zurück zusetzen, genügt ein Neustart des Rechners über das Managementinterface. Beim Booten automatisch gestartete Firewalls können nicht zurückgesetzt werden. In diesem Fall hilft wie im oben genannten Punkt nur das Zurückspielen eines Backups.

Literaturverzeichnis

[AMD 05] AMD: *Secure Virtual Machine Architecture Reference Manual*, Mai 2005, http://enterprise.amd.com/downloadables/Pacifica_Spec.pdf .

[BOC 05] *Bochs Projekt Homepage*, Dezember 2005, http://bochs.sourceforge.net/ .

[Dike 05] DIKE, JEFF: *User Mode Linux Projekt Homepage*, Dezember 2005, http://user-mode-linux.sourceforge.net/ .

[DTL 05] *Deutsche Telekom Laboratories*, Dezember 2005, http://www.telekom.de/laboratories .

[Ecke 04] ECKERT, CLAUDIA: *IT-Sicherheit*. Oldenbourg Verlag, 3. Auflage, Juni 2004. ISBN 3-486-20000-3.

[FAU 05] *FAUmachine Projekt Homepage*, Dezember 2005, http://www3.informatik.uni-erlangen.de/Research/FAUmachine/ .

[Fre 06a] *FreeNX Projekt Homepage*, Februar 2006, http://freenx.berlios.de/ .

[fre 06b] *FreeS/WAN Projekt Homepage*, April 2006, http://www.freeswan.org/ .

[Gers 05] GERSTEL, MARKUS: *Hauptseminar der TU München: Ansätze für Betriebssysteme der Zukunft, WS 2005/06, Virtualisierungsansätze mit Schwerpunkt Xen*, Dezember 2005, http://www13.informatik.tu-muenchen.de/lehre/seminare/WS0506/hauptsem/Ausarbeitung02.pdf .

[INTE 05] INTEL®: *Virtualization Technology Specification for the IA-32 Intel® Architecture*, April 2005, ftp://download.intel.com/technology/computing/vptech/C97063-002.pdf .

[Kers 05] KERSTEN, CHRISTIAN: *Blockseminar der Universität Saarbrücken: Aktuelle Hardwaretechnologien und ihr Einfluss auf die Betriebssystementwicklung, Hardware-Virtualisierung auf der IA-32*, April 2005, http://hs-sonne.cs.uni-sb.de:8080/lehrstuhl/SS2005/Seminar_Aktuelle_Technologien/library/04F_-_Kersten_-_Hardware-Virtualisierung.pdf .
.

[KJD 05] KENNETH J. DUDA, DAVID R. CHERITON: *Borrowed-Virtual-Time (BVT) scheduling: supporting latency-sensitive threads in a general-purpose scheduler*, Dezember 2005, http://www-dsg.stanford.edu/bvt/bvt.ps .

[Lind 05] LINDINGER, TOBIAS: *Machbarkeitsanalyse zur Virtualisierung des IT-Sicherheit Praktikums*, Oktober 2005, http://www.nm.ifi.lmu.de/pub/Fopras/lind05/PDF-Version/lind05.pdf .

[Linu 06] LINUX MAGAZIN: *Virtualisierung - Geteilte Server sind besser genutzte Server*. Linux New Media AG, April 2006. Ausgabe 01/2006.

[Maga 06] MAGAZIN FÜR COMPUTERTECHNIK C'T: *Das Netz im Netz*. Heise Verlag, April 2006. Ausgabe 07/2006.

[MSV 05] *Microsoft Virtual PC Homepage*, Dezember 2005, http://www.microsoft.com/windows/virtualpc/default.mspx
.

[ope 06a] *OpenSSL Projekt Homepage*, April 2006, http://www.openssl.org .

[ope 06b] *Openswan Projekt Homepage*, April 2006, http://www.openswan.org/ .

[Ope 06c] *OpenVPN Projekt Homepage*, April 2006, http://openvpn.net/ .

[PAR 05a] *Parallax, ein Cluster Storage System für die Verwendung von CoW und Snapshots*, Dezember 2005, http://www.cl.cam.ac.uk/ akw27/papers/hotos-parallax.pdf .

[PAR 05b] *Parallels Homepage*, Dezember 2005, http://www.parallels.com/ .

[Prat 04] PRATT, IAN: *Performance of Xen compared to native Linux, VMware and User Mode Linux*, Dezember 2004, http://www.cl.cam.ac.uk/Research/SRG/netos/xen/performance.html .

[Prat 05] PRATT, IAN: *Xen Status Report*, Dezember 2005, http://www.cl.cam.ac.uk/netos/papers/ian-status.ppt .

[QEM 05] *QEMU Projekt Homepage*, Dezember 2005, http://fabrice.bellard.free.fr/qemu/ .

[ska 05] *SKAS-Patch for UML Homepage*, 2005, http://www.user-mode-linux.org/ blaisorblade/ .

[Snyd 06] SNYDER, JOSHUA: *IP packet flow on a Linux bridging firewall*, März 2006, http://ebtables.sourceforge.net/br_fw_ia/PacketFlow.png .

[ssl 96] *SSL Spezifikation*, November 1996, http://www.netscape.com/eng/ssl3/ .

[str 06] *Strongswan Projekt Homepage*, April 2006, http://www.strongswan.org/ .

[The 98] THE INTERNET ENGINEERING TASK FORCE (IETF): *Der Standard IPSEC, RFC 2401*, November 1998, http://www.ietf.org/rfc/rfc2401.txt .

[The 99] THE INTERNET ENGINEERING TASK FORCE (IETF): *Der Standard PPTP, RFC 2637*, Juli 1999, http://www.ietf.org/rfc/rfc2637.txt .

[The 06] THE INTERNET ENGINEERING TASK FORCE (IETF): *Der Standard TLS, RFC 4346*, April 2006, http://www1.ietf.org/rfc/rfc4346.txt .

[Univ 05a] UNIVERSITY OF CAMBRIDGE: *Xen Projekt Homepage*, Dezember 2005, http://www.cl.cam.ac.uk/Research/SRG/netos/xen/ .

[Univ 05b] UNIVERSITY OF CAMBRIDGE: *Xenoserver Projekt Homepage*, Dezember 2005, http://www.cl.cam.ac.uk/Research/SRG/netos/xeno/ .

[VMW 05a] *VMware ESX Server*, Dezember 2005, http://www.vmware.com/de/products/server/esx_features.html .

[VMW 05b] *VMware GSX Server*, Dezember 2005, http://www.vmware.com/de/products/server/gsx_features.html .

[VMW 05c] *VMware Homepage*, Dezember 2005, http://www.vmware.com/ .

[VMW 05d] *VMware Worksation*, Dezember 2005, http://www.vmware.com/de/products/desktop/ws_features.html .

[VNUM 05] VNUML: *VNUML Projekt Homepage*, Oktober 2005, http://jungla.dit.upm.es/ vnuml/ .

[WIK 05] *WIKIPEDIA, die freie Enzyklopädie*, Dezember 2005, http://de.wikipedia.org/ .

[XEN 05] *Xen Wiki*, Dezember 2005, http://wiki.xensource.com/xenwiki/XenDocs .

[XS 05] XEN SOURCE, INC.: *Xen Projekt Homepage*, Dezember 2005, http://www.xensource.com/ .