Bundesamt
für Sicherheit in der
Informationstechnik

Sicherheitsanalyse von KVM (KVMSec)

Version: 1.0.1

Verfasser: OpenSource Security Ralf Spenneberg

Auftraggeber: Bundesamt für Sicherheit in der Informationstechnik

Bundesamt für Sicherheit in der Informationstechnik
Postfach 20 03 63
53133 Bonn
Tel.: +49 22899 9582-0
E-Mail: bsi-publikationen@bsi.bund.de
Internet: https://www.bsi.bund.de

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

ACPI Advanced Configuration and Power Interface
AES Advanced Encryption Standard
AH Authentication Header
API Application Programming Interface
APIC Advanced Programmable Interrupt Controller
APICv Advanced Programmable Interrupt Controller Virtualization
ARP Address Resolution Protocol
ASLR Address Space Layout Randomization
AVIC Advanced Virtual Interrupt Controller
BIOS Basic Input/Output System
BSD Berkeley Software Distribution
CBC Cipher Block Chaining
CDP Cisco Discovery Protocol
CDROM Compact Disc Read-Only Memory
cgroups Control Groups
CIFS Common Internet File System
CN Common Name
CPCP CentOS Profile for Cloud Providers
CPU Central Processing Unit
CPUID Central Processing Unit Identification
CR Control Register
CRC Cyclic Redundancy Check
CVE Common Vulnerabilities and Exposure
DAC Discretionary Access Control
DHCP Dynamic Host Configuration Protocol

KVM..... Kernel-based Virtual Machine
LUKS..... Linux Unified Key Setup
LXC..... LinuX Container
MAC..... Media Access Control
MB..... Megabyte
MCS..... Multi-Category Security
MDS..... Metadata Server Daemon
MLS..... Multi-Level Security
MMIO..... Memory Mapped Input/Output
MMU..... Memory Management Unit
MPLS..... Multiprotocol Label Switching
MPX..... Memory Protection Extension
MSI..... Message Signaled Interrupts
MSR..... Model-specific register
NAT..... Network Address Translation
NBD..... Network Block Device
NDP..... Neighbor Discovery Protocol
NFS..... Network File System
NMI..... Non-maskable interrupt
NUMA..... Non-Uniform Memory Access
NVRAM..... Non-volatile Random-Access Memory
OS..... Operation System
OSD..... Object Storage Daemon
OSI..... Open Systems Interconnection
PAE..... Physical Address Extension
PCI..... Peripheral Component Interconnect
PCIe..... Peripheral Component Interconnect Express
PCMCIA..... Personal Computer Memory Card International Association
PKI..... Public-Key-Infrastruktur
PMU..... Performance Management Unit
PPC..... PowerPC
PVI..... Protected-Mode Virtual Interrupts
QEMU..... Quick Emulator
RADOS..... Reliable Autonomic Distributed Object Store
RAID..... Redundant Array of Independent Disks
RAM..... Random Access Memory

Historie

1.0	23.1.2017	Initiale Version
1.0.1	26.6.2017	Alternativtexte für Abbildungen hinzugefügt

1 Projektübersicht

Virtualisierung hat sich zu einer der wesentlichen Vorgehensweisen entwickelt, um Serversysteme und IT-Dienste bereitzustellen und zu betreiben. Dies trifft sowohl auf kleine und mittelständische als auch auf große Unternehmen und Behörden zu. Die Abstraktion von physischer Hardware mittels Virtualisierung ist heute gelebte Praxis.

Die Gründe hierfür sind vielfältig. In erster Linie erlaubt die Abstraktion von physischer Hardware sowohl eine effiziente Auslastung der vorhandenen Hardwareressourcen als auch eine einfache Skalierung und hohe Flexibilität. Weitere Vorteile sind die unter dem Stichwort Green-IT zusammengefassten Aspekte des verringerten Platz-, Strom- und Kühlungsbedarfs. Darüber hinaus wird die Virtualisierung oftmals mit einer erhöhten Ausfallsicherheit und Fehlertoleranz in Zusammenhang gebracht.

Die Weiterentwicklung dieser Technik wird in erster Linie von den Hardwareherstellern, wie Intel und AMD, und den Anbietern der Virtualisierungsstacks vorangetrieben. Speziell bei den Anbietern der Virtualisierungsumgebungen ist es in den letzten Jahren zu einer gewissen Konsolidierung gekommen. So hat sich die Kernel-based Virtual Machine (KVM) als Basistechnologie im Open-Source-Bereich durchgesetzt. Mit libvirt steht eine einfache und einheitliche Schnittstelle zur Verwaltung von KVM zur Verfügung, die von unterschiedlichsten Werkzeugen genutzt wird und sich zu einem De-facto-Standard entwickelt hat.

Diese Studie hat daher zum Ziel, eine aussagekräftige und belastbare Beurteilung der Sicherheit einer virtualisierten Infrastruktur auf Basis von KVM zu erstellen. Hierzu werden die einzelnen Komponenten der Virtualisierungsumgebung einer typischen Einsatzumgebung vorgestellt sowie deren Kommunikationsbeziehungen untereinander ermittelt. Hierauf aufbauend wird die Architektur des Virtualisierungssystems mit Fokus auf sicherheitstechnischen Eigenschaften beschrieben. Eine Testumgebung wird genutzt, um Schwachstellen zu ermitteln und deren Auswirkungen demonstrieren zu können.

Den Schwerpunkt der Studie bildet die sicherheitstechnische Untersuchung der Softwarekomponenten, die für die Virtualisierung benötigt werden. Hierzu gehört insbesondere der Hypervisor, der sowohl auf konzeptionelle und architektonische Schwachstellen als auch auf sicherheitskritische Implementierungsfehler hin überprüft wird. Auch wird eine allgemeine Beurteilung der Qualität des entsprechenden Quelltextes durchgeführt. Darüber hinaus werden verschiedene Möglichkeiten zur Netzwerkanbindung und Speicherung persistenter Daten auf konzeptionelle und architektonische Schwächen hin untersucht. Ebenso werden die Management-Werkzeuge virsh, virt-manager und OpenStack untersucht.

Abschließend werden auf Basis der Untersuchungsergebnisse effektive und praxistaugliche Absicherungsmaßnahmen aufgezeigt.

2 Rahmenbedingungen

2.1 Bedrohungsszenario

Um eine systematische, zielgerichtete und praxisrelevante Untersuchung der Sicherheit von Virtualisierungskomponenten zu ermöglichen, wird im Folgenden ein Bedrohungsszenario beschrieben. Zusammen mit den in Abschnitt 2.2 dargelegten Sicherheitsanforderungen stellt es die Grundlage für die Beurteilung der Sicherheit der untersuchten Komponenten dar. Es wird systematisch nach Eigenschaften gesucht, die in diesem Bedrohungsszenario die Sicherheitsanforderungen verletzen können. Solche Eigenschaften stellen im Rahmen dieses Projektes Schwachstellen dar.[1]

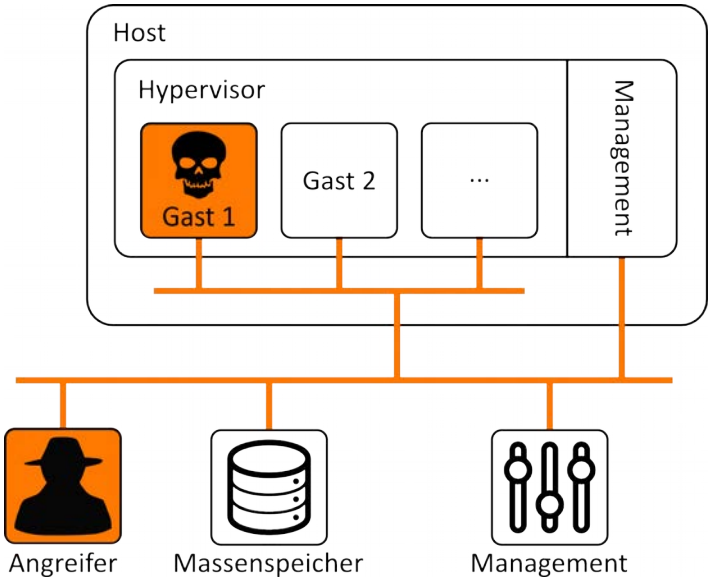

Abbildung 1: Bedrohungsszenario

Das ausgearbeitete Bedrohungsszenario und die Sicherheitsanforderungen sollen hierbei bewusst nicht die konkret eingesetzte Software oder die konkrete Art der Vernetzung implizieren. Vielmehr handelt es sich um ein abstraktes Modell der Virtualisierung und der Anforderungen an diese. Dies ermöglicht es, allgemeingültige Rahmenbedingungen für die verschiedenen konkret zu untersuchenden Soft- und Hardwarekomponenten zu formulieren.

Das Bedrohungsszenario ist in Abb. 1 dargestellt und setzt sich aus folgenden Annahmen zusammen:

1. Kompromittierte Gäste
   Der Angreifer hat die volle Kontrolle über ein oder mehrere Gastsysteme. Er ist insbesondere in der Lage, Befehle mit administrativen Rechten auszuführen. Ebenso ist es ihm möglich, im Kernel-Space des Gastsystems (Ring 0 der virtuellen CPU) uneingeschränkt Code auszuführen.
2. Kompromittiertes Netzwerk
   Der Angreifer hat weitgehende Kontrolle über das Netzwerk, mit dem die virtuellen Maschinen verbunden sind. Dies umfasst sowohl den „virtuellen" Teil des Netzes, also die Paketvermittlung, die durch den virtualisierenden Host durchgeführt wird, als auch das reale Netzwerk, an das der

1 Es werden auch solche Systemeigenschaften dokumentiert, die im Rahmen des Projekts festgestellt werden und auf Grundlage des dargestellten Bedrohungsszenarios und der Sicherheitsanforderungen keine Schwachstellen darstellen, jedoch unter anderen Rahmenbedingungen sicherheitskritisch sein könnten. Es wird jedoch außerhalb dieser Rahmenbedingungen nicht systematisch nach solchen Eigenschaften gesucht.

virtualisierende Host angeschlossen ist. Ebenso verläuft die Kommunikation zwischen Massenspeicher, Management und Host über das Netzwerk.[2]

3. Integre Virtualisierungsumgebung
   Alle weiteren Komponenten der Virtualisierungsumgebung sind nicht kompromittiert. Die Interaktion des Angreifers mit diesen Komponenten ist somit auf die Möglichkeiten beschränkt, die sich aus den Punkten 1 und 2 ergeben, und auf solche, die sich aus deren regulärer Nutzung ergeben.

Hierbei ist anzumerken, dass in den Untersuchungen der einzelnen Komponenten auch implizit von einem abgeschwächten Bedrohungsszenario ausgegangen werden kann. So muss der Angreifer z.B. für einige Angriffe auf die Netzwerkinfrastruktur nicht unbedingt einen Gast kontrollieren (Punkt 1), sondern er braucht lediglich Zugriff auf das Netzwerk (Punkt 2). Auch ist für einige Angriffe keine vollständige Kontrolle über einen Gast (Punkt 1) erforderlich, sondern lediglich ein unprivilegierter Zugriff.

Sofern Angriffe untersucht werden, an die weitere Voraussetzungen gestellt werden, werden diese an entsprechenden Stellen explizit benannt.

2.2 Sicherheitsanforderungen

Aufbauend auf dem Bedrohungsszenario stellt sich zur Beurteilung der Sicherheit des Virtualisierungssystems die Frage, welche sicherheitstechnischen Anforderungen an das Virtualisierungssystem gestellt werden. Im Rahmen dieses Projektes werden folgende Sicherheitsanforderungen angenommen:

1. Schutz der Datenverarbeitung und -speicherung
   Die Vertraulichkeit und Integrität der Datenverarbeitung und -speicherung nicht kompromittierter Gäste sowie die Verfügbarkeit von deren Diensten darf durch die Virtualisierung nicht geschwächt werden.
2. Schutz der Kommunikation
   Sowohl die Verfügbarkeit von Netzwerkdiensten nicht kompromittierter Gäste als auch die Vertraulichkeit und Integrität der Kommunikationsinhalte dieser Gäste mit anderen nicht kompromittierten Systemen, insbesondere
   - anderen Gästen,
   - dem Hypervisor/Hostsystem,
   - dem Massenspeicher,
   - der Management-Schnittstelle und
   - der Hardware (emuliert und real),

   darf durch die Virtualisierung nicht geschwächt werden.
3. Schutz der Verwaltbarkeit
   Die Verwaltung aller Gäste muss stets möglich sein. Die Vertraulichkeit und Integrität der hierzu übertragenen Daten muss sichergestellt werden. Ferner muss sichergestellt werden, dass nur autorisierte Dienste und Personen entsprechend ihren Berechtigungen Zugriff erhalten.
4. Schutz vor Privilegienausweitung im Gast
   Es darf nicht möglich sein, mithilfe der Virtualisierung die Privilegien innerhalb des Gastes auszuweiten.
5. Beschränkung der Netzwerkkommunikation
   Administrative Beschränkungen der Netzwerkkommunikation eines Gastes müssen auch dann eingehalten werden, wenn der Gast kompromittiert ist.

Allgemeine Anmerkungen

Allgemein ist zu beachten, dass lediglich virtualisierungsspezifische Gefährdungen untersucht werden. So wird beispielsweise nicht die Möglichkeit des Angreifers betrachtet, ein Gastsystem über eine Schwachstelle

2 Ein solches Netzwerk kann aus mehreren miteinander verbundenen oder voneinander separierten Teilnetzwerken bestehen.

eines seiner Dienste unmittelbar anzugreifen. Die Untersuchung stellt daher keine allumfassende Beschreibung der Sicherheit von Gastsystemen dar, sondern beurteilt die Gefährdungen, die durch die Virtualisierung im Rahmen des Bedrohungsszenarios entstehen.

Ferner ist zu beachten, dass keine Aussagen bezüglich kompromittierter Gäste getroffen werden. Dies bedeutet z. B., dass die Vertraulichkeit und Integrität von Daten eines Gastes, der unter der vollständigen Kontrolle eines Angreifers steht, durch Punkt 1 nicht garantiert wird. Die einzigen Ausnahmen bilden hier die Verhinderung einer Rechteausweitung im Gast, die durch die Virtualisierung erst ermöglicht wird (s. Anmerkungen zu Punkt 1) und der Schutz der Verwaltbarkeit (s. Punkt 3).

Anmerkungen zu Sicherheitsanforderung 1

Sicherheitsanforderung 1 fordert, dass der Angreifer weder lesend noch schreibend Zugriff auf die Daten erlangen kann, die von nicht kompromittierten Gästen verarbeitet und gespeichert werden. Dies setzt insbesondere voraus, dass der Angreifer keine Kontrolle über das Hostsystem erlangen kann, da dieses weitgehende Möglichkeiten hat, entsprechende Daten der Gastsysteme einzusehen und zu manipulieren. Somit impliziert Sicherheitsanforderung 1, dass es dem Angreifer nicht möglich ist, aus dem kompromittierten Gast auszubrechen.

Sicherheitsanforderung 1 garantiert jedoch nicht den Schutz jeglicher Information bezüglich des Verhaltens eines Gastes. Diesbezüglich muss insbesondere beachtet werden, dass sich die Gäste die realen Ressourcen, wie z. B. Arbeitsspeicher, CPU oder IO-Kapazität, typischerweise untereinander teilen. Somit ist es dem Angreifer mithilfe eines kompromittierten Gastes prinzipiell möglich, z. B. die Ressourcennutzung anderer Gäste zu ermitteln. Der Schutz solcher Informationen wird nicht von Sicherheitsanforderung 1 erfasst und wird im Rahmen dieses Projektes daher nicht systematisch analysiert. Darüber hinaus wird hier auch nicht betrachtet, ob der Angreifer Informationen über die Virtualisierungsumgebung gewinnen kann, sofern nicht eine andere Sicherheitsanforderung verletzt wird.[3]Des Weiteren ist nicht jede Beeinflussung der Performance von nicht kompromittierten Gästen durch den Angreifer als Einschränkung der Verfügbarkeit zu betrachten.

Anmerkungen zu Sicherheitsanforderung 2

Sicherheitsanforderung 2 garantiert die Vertraulichkeit und Integrität der virtualisierungsspezifischen Kommunikation. Dies umfasst insbesondere sowohl die Netzwerkkommunikation zwischen dem Host und dem Speicher der Festplattenabbilder als auch den Netzwerkverkehr zur Verwaltung der Virtualisierungsumgebung. Kommunikation, die nicht virtualisierungsspezifisch ist, wird von Sicherheitsanforderung 2 hingegen nicht erfasst. Tauscht ein Gast beispielsweise sensible Daten mittels einer ungeschützten HTTP-Verbindung über das Internet aus, so ist der hieraus resultierende Verlust der Vertraulichkeit dieser Daten keine Schwäche der Virtualisierungsumgebung. Für den Schutz der unmittelbar durch den Gast durchgeführten Kommunikation ist dieser selbst verantwortlich.

Anmerkungen zu Sicherheitsanforderung 3

Sicherheitsanforderung 3 garantiert, dass entsprechend legitimierte Dienste und Personen die Virtualisierungsumgebung stets verwalten können. Einem Angreifer darf es nicht möglich sein, Anweisungen an das Virtualisierungssystem einzusehen, abzuändern oder zu unterbinden. Daher muss eine sichere und verlässliche Kommunikation zwischen den Managementkomponenten und auch zu den virtuellen Maschinen gewährleistet sein.

Anmerkungen zu Sicherheitsanforderung 4

Sicherheitsanforderung 4 garantiert dem Gast, dass die von ihm getroffenen Maßnahmen zur Beschränkung von Privilegien auch innerhalb der Virtualisierung effektiv sind. Besitzt ein Angreifer einen

3 Ist ein Angreifer beispielsweise in der Lage, zu erkennen, dass ein kompromittierter Gast nicht unmittelbar auf realer Hardware ausgeführt, sondern virtualisiert wird, so ist dies in diesem Sinne keine Schwäche des Virtualisierungssystems. Gleiches gilt für die Ermittlung der Version der Virtualisierungssoftware oder für die Identifizierung des virtualisierenden Hosts.

unprivilegierten Zugriff auf einen Gast (abgeschwächtes Bedrohungsmodell), so darf ihm die Virtualisierung nicht dazu verhelfen, darüber hinausgehende Berechtigungen im Gast zu erlangen.[4]

Anmerkungen zu Sicherheitsanforderung 5

Sicherheitsanforderung 5 besagt, dass administrative Beschränkungen bezüglich der Netzwerkkommunikation, die durch die Virtualisierungsumgebung vorgegeben werden, einzuhalten sind. Dies gilt auch und insbesondere dann, wenn ein an der Kommunikation beteiligter Gast kompromittiert ist. Dem Angreifer darf es also nicht möglich sein, Beschränkungen der Kommunikationsmöglichkeiten des kompromittierten Gastes zu umgehen. Das Vorhandensein einer solchen Beschränkungsmöglichkeit wird durch Sicherheitsanforderung 4 jedoch nicht erzwungen. Das Fehlen einer solchen Möglichkeit ist in diesem Sinne keine Schwachstelle. Des Weiteren bezieht sich Sicherheitsanforderung 4 ausschließlich auf die Netzwerkkommunikation. Sie verlangt nicht, dass die Kommunikation über versteckte Kanäle, z. B. auf Basis gemeinsam genutzter Ressourcen, ausgeschlossen sein muss.

4 Angriffe zur Erweiterung der Privilegien innerhalb des Gastes sind z. B. [CVE-2010-0306] und [CVE-2010-0298].

3 Architekturbeschreibung

3.1 Zielsetzung

Zur Identifizierung sicherheitskritischer Komponenten und Kommunikationsbeziehungen wird im Folgenden die Architektur einer auf KVM basierenden Virtualisierungsumgebung dargestellt. Es werden sowohl die Architektur und Kommunikationsbeziehungen der unmittelbar für die Virtualisierung zuständigen Komponenten auf dem virtualisierenden Host als auch die Kommunikationsbeziehungen zu externen Komponenten betrachtet, wie zu Storage, Managementumgebung und weiteren KVM-Hosts. Als Managementumgebung wird hierbei OpenStack eingesetzt. Neben OpenStack werden auch virt-manager und virsh als leichtgewichtige Managementwerkzeuge betrachtet. Der Fokus liegt hierbei auf der Darstellung sicherheitstechnischer Aspekte, insbesondere der Schutzziele Vertraulichkeit, Integrität und Verfügbarkeit.

Die beschriebene Architektur dient als Grundlage für die Realisierung der Testumgebung, die in Abschnitt 4 beschrieben wird. Im Folgenden werden zunächst der Host mit Hypervisor und Gästen, anschließend die Managementumgebung betrachtet.

3.2 Überblick

Die technische Grundlage der in dieser Studie untersuchten Virtualisierung bilden die Komponenten KVM und QEMU. KVM ist Bestandteil des Linux-Kernels und stellt grundlegende Funktionen zur Verfügung, z. B. das Ausführen von Code im Kontext einer virtuellen CPU. QEMU ist eine auf diesen Funktionen[5] aufbauende Anwendung, deren Zweck unter anderem die Emulation von Hardware ist. Darüber hinaus werden zur Virtualisierung allgemeine Funktionen des Linux-Kernels genutzt, wie z. B. das Multi-Tasking, die Speicherverwaltung und der Zugriff auf reale Hardware mithilfe von Gerätetreibern.[6] Zur Erstellung und Verwaltung KVM-basierter virtueller Maschinen über ein Netzwerk hat sich libvirt als De-facto-Standard durchgesetzt. Daher wird libvirt im Rahmen dieser Studie als Schnittstelle zu Managementwerkzeugen untersucht.

Abb. 2 zeigt eine vereinfachte Darstellung der Architektur eines Hosts, der auf diesen Komponenten basiert. Die Abbildung zeigt außerdem wesentliche Vertrauensgrenzen des Systems zwischen verschieden privilegierten Codeabschnitten sowie die Kommunikation, die über diese Grenzen hinweg stattfindet.[7]

Der Dienst libvirtd bildet die Schnittstelle zum Netzwerk. Er wird über das Netzwerk angesprochen und kann virtuelle Maschinen erzeugen und mit diesen interagieren. Er dient als Vermittler zwischen QEMU[8] und netzwerkbasierten Management-komponenten. Somit ergibt sich eine Vertrauensgrenze zwischen dem Netzwerk und libvirtd, da auf Basis des zugrunde liegenden Bedrohungsszenarios (s. Abschnitt 2.1) angenommen wird, dass die Kommunikation über das Netzwerk prinzipiell nicht vertrauenswürdig ist. Ein weiterer Kommunikationspfad befindet sich zwischen libvirtd und den virtuellen Maschinen, die mittels QEMU ausgeführt werden. Üblicherweise verwaltet libvirtd mehrere virtuelle Maschinen und damit mehrere QEMU-Prozesse.

5 Es sei erwähnt, dass QEMU auch unabhängig von KVM betrieben werden kann. Hierbei wird die jeweilige virtuelle CPU vollständig mittels Software emuliert. Die dabei entstehenden Performancenachteile machen diese Betriebsart für die Servervirtualisierung jedoch uninteressant. Aus diesem Grund werden entsprechende Betriebsmodi im Rahmen dieser Studie nicht betrachtet.

6 Diese nicht virtualisierungsspezifischen Funktionen werden im Rahmen dieser Studie jedoch nur in begründeten Einzelfällen analysiert.

7 Es ist zu beachten, dass in dieser Architekturübersicht nicht alle Kommunikationspfade und Vertrauensgrenzen eingezeichnet sind. Ein detaillierteres Bild gibt Abschnitt 3.5.

8 libvirt ist darüber hinaus imstande, eine Reihe weiterer Hypervisoren zu verwalten, z. B. Xen oder VMWare. Dies bleibt aber aufgrund der Zielsetzung dieser Studie unbetrachtet.

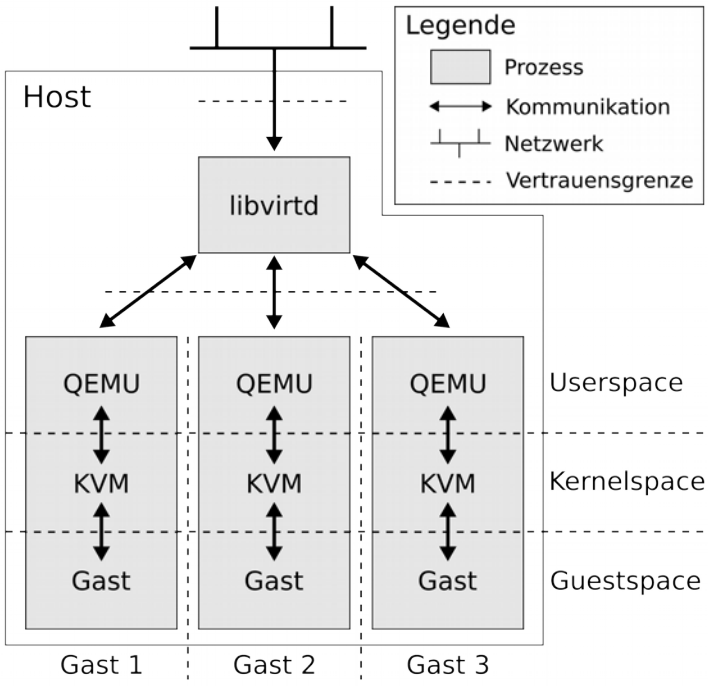

Abbildung 2: Vereinfachte Architektur des Hosts

QEMU ermöglicht seinerseits die Erzeugung und Verwaltung virtueller Maschinen. Ein QEMU-Prozess steuert hierbei stets genau eine virtuelle Maschine. Ein solcher Prozess unterscheidet sich aus Sicht des Linux-Kernels des Hosts nicht wesentlich von anderen Benutzerprozessen. Somit können die vorhandenen virtualisierungsunabhängigen Funktionen des Hosts genutzt werden. QEMU stellt ferner eine API zur Verfügung, anhand derer mit libvirtd interagiert werden kann. Dies ermöglicht sowohl das Auslesen von Informationen bezüglich der virtuellen Maschinen als auch deren Administration, d. h. das Stoppen und Starten virtueller Maschinen, das Hinzufügen und Entfernen virtueller Geräte und die Durchführung von Live-Migrationen.

Bei Ausführung mehrerer virtueller Maschinen werden die hierzu genutzten QEMU-Prozesse durch den Linux-Kernel des Hostsystems voneinander isoliert. Die QEMU-Prozesse können sich daher nicht direkt und unmittelbar gegenseitig beeinflussen.[9] Dem libvirtd-Prozess hingegen ist es im Rahmen seiner administrativen Funktion möglich, mit allen QEMU-Prozessen zu interagieren, wie oben beschrieben. Somit unterscheiden sich die Privilegien des libvirtd-Prozesses und der QEMU-Prozesse signifikant voneinander, sodass eine weitere Vertrauensgrenze gegeben ist.

Die QEMU-Prozesse umfassen ihrerseits mehrere Vertrauensgrenzen. Dies sind zum einen die in Abb. 2 horizontal dargestellten Grenzen zwischen den unterschiedlich privilegierten Codeabschnitten des jeweiligen Prozesses. Teile eines solchen Prozesses stellt die Anwendung QEMU. Diese übernimmt die Kommunikation mit libvirtd und regelmäßig auch die Emulation virtueller Geräte. Die Erstellung und Verwaltung virtueller CPUs, die die technische Grundlage für die Virtualisierung darstellen, ist ihr jedoch unmittelbar nicht möglich, da die hierzu notwendigen CPU-Instruktionen nur im privilegierten Kernelspace ausgeführt werden können.

Aus diesem Grund ist ein Vermittler zwischen Userspace und CPU innerhalb des Linux-Kernels erforderlich. Diese Rolle übernimmt das Kernelmodul KVM. Im Zuge der Virtualisierung interagiert QEMU

9 Es ist möglich, diese strikte Trennung ein Stück weit aufzuheben, indem gemeinsam genutzter Speicher verwendet wird. Die Einbindung von gemeinsam genutztem Speicher muss jedoch explizit konfiguriert werden und hat im Rahmen der Servervirtualisierung keine praktische Relevanz. Aus diesem Grund wird diese Möglichkeit nicht weiter betrachtet.

daher mit KVM, wodurch es beim Übergang der Programmausführung vom Userspace in den Kernelspace zu einer zeitweisen Erweiterung der Prozessprivilegien kommt. Das Kernelmodul ist seinerseits für die Isolation des Gastes vom Host und somit für die Speicherverwaltung und -separierung zuständig. Darüber hinaus startet und stoppt das Kernelmodul nach Anweisung durch QEMU die Ausführung virtueller Maschinen.

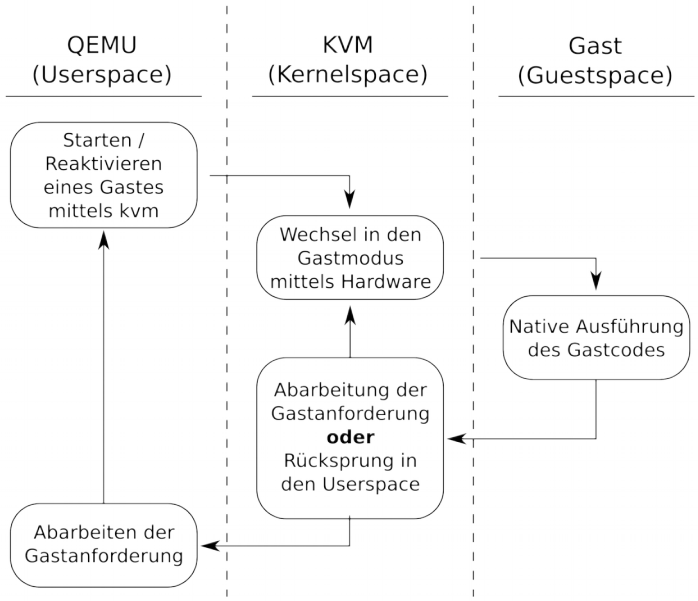

Abbildung 3: Ausführung eines Gastes

Die Isolation von Gast und Host wird realisiert, indem KVM die reale CPU anweist, Code in einem eingeschränkt privilegierten Kontext auszuführen. Innerhalb eines solchen Kontextes wird dem ausgeführten (Gast-)Code vorgetäuscht, uneingeschränkte Berechtigungen zu besitzen. Im Gegensatz zu sicherheitstechnisch unbedenklichen Anweisungen werden sicherheitskritische Befehle jedoch nicht von der CPU ausgeführt, sondern deren Ausführung an KVM delegiert. Statt zu einer unmittelbaren Ausführung sicherheitskritischer Anweisungen durch die Hardware kommt es somit zu einer kontrollierten Emulation durch Software. Hierbei ist insbesondere zu beachten, dass im Rahmen einer solchen Emulation ein Rücksprung vom Gast in den Kernelspace und damit ein Privilegienwechsel stattfindet. Im Zuge der Emulation virtueller Geräte durch QEMU folgt hierauf regelmäßig ein Wechsel in den entsprechenden Code innerhalb des Userspaces und somit eine weitere Überschreitung einer Vertrauensgrenze. Das Zusammenspiel von QEMU, KVM und Hardware wird in Abb. 3 skizziert.

Die Virtualisierung wird somit in den folgenden drei unterschiedlich privilegierten Kontexten durchgeführt:

- Guestspace
- Userspace
- Kernelspace

Die Bereitstellung von Geräten für die virtuelle Maschine erfolgt hierbei typischerweise auf Basis eines virtuellen PCI-Busses. Ein PCI-Gerät kann dem Gast auf drei verschiedenen Wegen bereitgestellt werden:

- Emulation oder Paravirtualisierung mittels QEMU
- Paravirtualisierung durch den Kernel des Hosts
- Durchreichung mittels Hardware

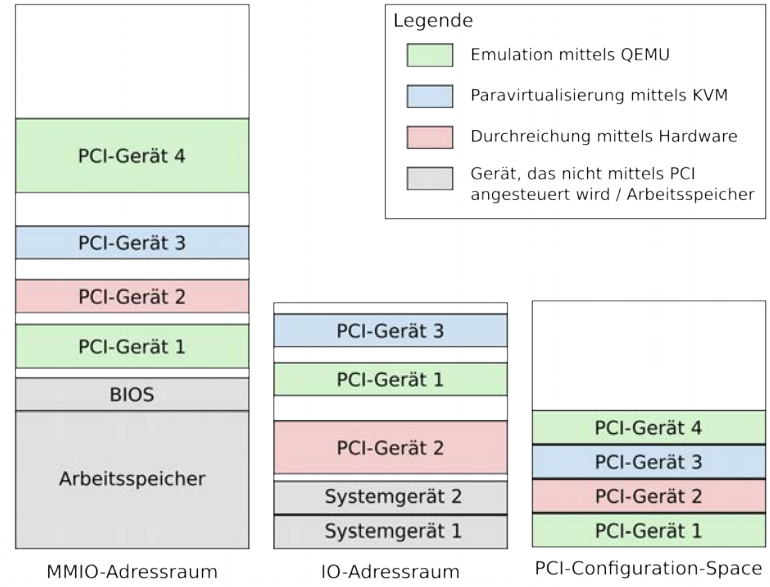

Abbildung 4: Adressräume der virtuellen Maschine

Im Falle der Durchreichung von realer Hardware in den Gast erfolgt der Zugriff der virtuellen Maschine auf die Hardware weitestgehend unabhängig von KVM und QEMU (s. Abschnitt 3.3). Anderenfalls wird der Zugriff auf das virtuelle Gerät abgefangen und von KVM oder QEMU verarbeitet (Trap-and-emulate-Verfahren). Aus Sicht des Gastes erscheint das Gerät in jedem Fall als real.

Technisch gesehen, handelt es sich bei einem Zugriff auf ein solches PCI-Gerät um eine Lese- oder Schreibanweisung auf den IO- bzw. den MMIO-Adressraum oder den PCI-Configuration-Space. Abb. 4 zeigt ein Beispiel für die Einbindung mehrerer Geräte in die Adressräume einer virtuellen Maschine.[10] Eine weiterführende Darstellung von PCI gibt [PCI].

3.3 Hardware

3.3.1 Codeausführung

Um die Performanceanforderungen erfüllen zu können, die an die Virtualisierung von Serversystemen gestellt werden, muss Gastcode nativ auf der CPU ausgeführt werden. Trotz einer solchen nativen Ausführung muss jedoch die Isolation der virtuellen Maschinen untereinander und vom Host gewahrt werden.

Um dies zu ermöglichen, unterstützen moderne CPUs der Unternehmen Intel und AMD die Technik VT-x bzw. AMD-V. Auch wenn sich diese Techniken im Detail unterscheiden, so sind ihre Konzepte identisch. Gastcode wird nativ auf der CPU in einem speziellen Modus ausgeführt. Dieser Modus erlaubt sowohl die Ausführung von unprivilegierten als auch von privilegierten Anweisungen. Auf diese Weise können Betriebssysteme in virtuellen Maschinen ausgeführt werden, ohne dass man ihre Betriebssysteme an die Virtualisierung anpassen muss. Während unprivilegierte Anweisungen hierbei unmittelbar durch die CPU verarbeitet werden, führen sicherheitskritische Anweisungen nicht zu einer unmittelbaren Ausführung. Stattdessen wird in den Code des Hypervisors zurückgesprungen. Diesem obliegt es dann, die vom Gast vorgegebene Anweisung zu emulieren, d. h., die entsprechende Operation unter Wahrung der Sicherheit des

10 Es sei angemerkt, dass die Nutzung aller drei Verfahren zur Einbindung von PCI-Geräten in einen einzelnen Gast hier zur Veranschaulichung gewählt wurde und in der Praxis selten anzutreffen ist.

Systems und der Isolation der Gäste auszuführen. Die wesentlichen Operationen, die anhand von VT-x und AMD-V abgefangen und somit emuliert werden können, sind:

- das Auslesen und Modifizieren des CPU-Zustands
  - HLT- und PAUSE-Instruktion
    Die Ausführung der HLT- oder PAUSE-Instruktion versetzt die CPU (zeitweise) in einen inaktiven Zustand. Die unkontrollierte Ausführung aus dem Gast heraus würde die Verfügbarkeit anderer Gäste beeinträchtigen.
  - XSAVES- und XRSTORS-Instruktion
    Die Instruktionen XSAVES und XRSTORS ermöglichen das Auslesen bzw. Setzen sicherheitsrelevanter Prozessorkonfigurationen. Eine unkontrollierte Ausführung aus dem Gast heraus ermöglicht die Störung der Funktionstüchtigkeit des Prozessors als auch die Modifikation der MMU. Somit wären Angriffe auf die Verfügbarkeit und Integrität des Gesamtsystems möglich.
  - RDMSR- und WRMSR-Instruktionen
    Der Zugriff auf maschinenspezifische Register (Machine Specific Registers, MSR) mittels der RDMSR- und WRMSR-Instruktion ermöglicht die Änderung verschiedener funktionskritischer Konfigurationen der CPU. Ein Zugriff auf diese Register kann unter Umständen die Funktionstüchtigkeit, Vertraulichkeit von Daten oder die Integrität des Gesamtsystems negativ beeinflussen.[11]
  - Zugriff auf die Konfigurationsregister CR0 und CR4
    Der schreibende Zugriff auf die globalen Konfigurationsregister CR0 und CR4 ermöglicht die Änderung sicherheitskritischer Konfigurationen der CPU. Hierüber ist beispielsweise eine globale Deaktivierung des Speicherschutzes möglich. Der uneingeschränkte Zugriff des Gastes auf diese Register würde somit die Funktionstüchtigkeit und die Integrität des Gesamtsystems beeinflussen.
  - Zugriff auf Debug-Register
    Ein lesender und insbesondere schreibender Zugriff auf Debug-Register der CPU durch den Gast würde unter anderem die Verfügbarkeit gefährden.[12]
- Erzeugung von Zufallswerten
  Für eine Reihe sicherheitsrelevanter Funktionen benötigt sowohl das Host- als auch das Gastsystem Zufallswerte mit möglichst großer Entropie. Die Erzeugung solcher Werte kann mithilfe der Instruktionen RDRAND und RDSEED erfolgen. Es kann jedoch nicht ausgeschlossen werden, dass es bei einer uneingeschränkten Nutzung durch den Gast zu einer schädlichen Beeinflussung anderer Gäste oder des Hosts kommt.
- Zugriff auf externe Ressourcen
  - Zugriff auf den IO-Adressraum
    Der Zugriff auf den IO-Adressraum ermöglicht den Zugriff auf die Hardware des Hosts. Ein solcher Zugriff könnte missbraucht werden, um unter andrem die Funktionstüchtigkeit des Gesamtsystems zu stören. Somit dürfen entsprechende Instruktionen (IN, INS, INSB, INSW, INSD, OUT, OUTS, OUTSB, OUTSW, OUTSD) vom Gast nicht unkontrolliert ausgeführt werden.
  - Zugriff auf den MMIO-Adressraum
    Analog zum Zugriffsschutz des IO-Adressraumes muss auch der Zugriff auf reale Hardware über den MMIO-Adressraum geschützt werden. Regelmäßig ist daher der unmittelbare und unkontrollierte Zugriff auf reale Hardware des Gastes gänzlich zu unterbinden. Eine Ausnahme hiervon bildet lediglich die abgesicherte Durchreichung von Hardware anhand einer IOMMU. Ferner muss es einem Gast ermöglicht werden, über den MMIO-Adressraum performant auf

11 [CVE-2014-3610], [CVE-2013-1797] und [CVE-2013-1796] beschreiben Fehler beim Zugriff auf modellspezifische Register, die zu einem Absturz des Hosts führen und durch einen Gast ausgelöst werden können.

12 [CVE-2009-3722] beschreibt einen Fehler, der aufgrund einer fehlerhaften Überprüfung beim Zugriff auf Debug-Register zum Absturz des Hostsystems führt und durch den Gast ausgelöst werden kann.

seinen Arbeitsspeicher und seine virtuellen Geräte zuzugreifen. Aus diesem Grund bedarf es einer feingranularen Kontrolle von Zugriffen auf den MMIO-Adressraum.

- Zugriff auf den PCI-Configuration-Space
  Ein weiterer Adressraum, dessen Zugriffe durch den Hypervisor kontrolliert werden müssen, ist der PCI-Configuration-Space. Dieser ermöglicht es, den PCI-Bus zu enumerieren, d. h., die angeschlossenen Geräte zu identifizieren und diese zu konfigurieren. Der Schutz des PCI-Configuration-Spaces realer Hardware ist daher für die Sicherheit des Systems unerlässlich. Ebenfalls muss zwingend ein virtueller PCI-Configuration-Space emuliert werden, um dem Betriebssystem des Gastes eine standardkonforme Interaktion mit virtuellen PCI-Geräten zu ermöglichen. Somit müssen Zugriffe auf den PCI-Configuration-Space kontrolliert werden.
- Konfiguration der MMU
  Um den Speicherschutz zu gewährleisten (s. Abschnitt 3.3.2), müssen Instruktionen zur Konfiguration der MMU kontrolliert werden.
  Hierzu gehören unter anderem lesende und schreibende Zugriffe auf die Konfigurationsregister CR2 und CR3. Sowohl direkte Zugriffe durch die Instruktion MOV als auch indirekte Zugriffe, z. B. mit der Instruktion LMSW, müssen somit überprüft werden. Ebenso dürfen Instruktionen zum Leeren des Translation Lookaside Buffers (TLB), wie INVLPG, INVPCID und INVEPT, nur

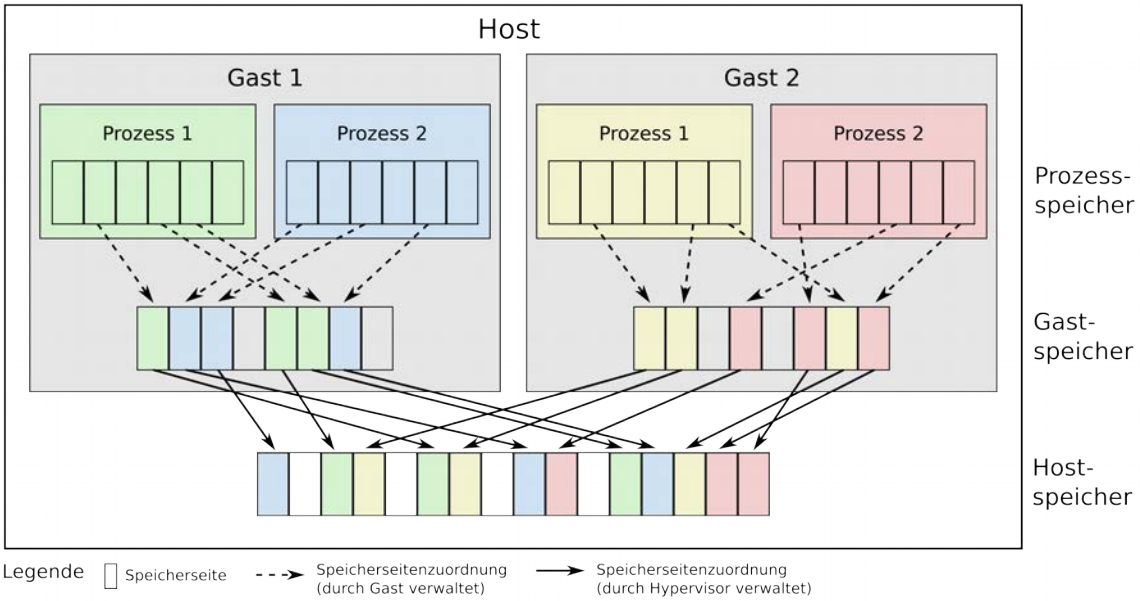

Abbildung 5: Speicherzugriffsmanagement

eingeschränkt durchgeführt werden.

3.3.2 Speicherschutz

Neben der kontrollierten Codeausführung durch den Gast ist zur Wahrung der Vertraulichkeit, Integrität und Verfügbarkeit des Gesamtsystems ein Speicherschutz erforderlich. Der Speicherschutz muss den lesenden und den schreibenden Zugriff eines Gastes auf dessen Speicher begrenzen, d. h., sowohl den Zugriff auf den Speicher des Hosts als auch auf den Speicher anderer Gäste unterbinden. Realisiert wird der

Speicherschutz mithilfe der MMU. Um die Sicherheit des Gesamtsystems zu gewährleisten, muss daher der Host eine Konfigurationsänderung der MMU, die der Gast vornimmt, überprüfen.

Ein entsprechender Zugriff des Gastes auf die Konfiguration der MMU ist erforderlich, da das Gastbetriebssystem selbst eine Speicherverwaltung durchführt und somit Zugriff auf die MMU verlangt. Die Herausforderung besteht also darin, Zugriffe des Gastes auf die Konfiguration der MMU einzuschränken, diesem aber gleichzeitig eine uneingeschränkte Zugriffsmöglichkeit vorzutäuschen. Abb. 5 skizziert das grundlegende Speicherzugriffsmanagement eines Hosts. Der Host hat hierbei die Pflicht, Speicherzugriffe aus dem Gast strikt auf den Speicher einzuschränken, der dem Gast zugeordnet ist.

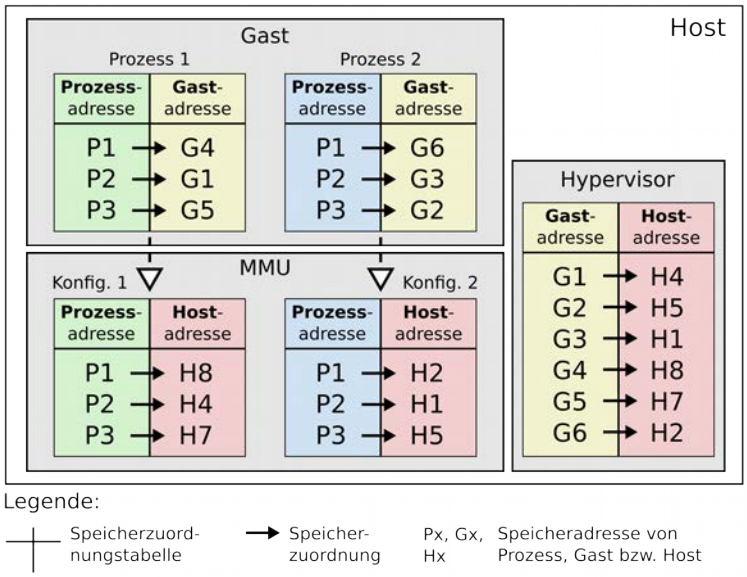

Abbildung 6: Aufbau des SoftMMU-Verfahrens

Um dies umzusetzen, sind zwei Ansätze praxisrelevant. Der klassische Ansatz, oft als SoftMMU bezeichnet, besteht darin, Änderungsanweisungen des Gastes bezüglich der Konfiguration der MMU durch den Hypervisor abzufangen. Hierauf fertigt der Host eine zweite Konfiguration an, die funktional identisch mit der vom Gast gewünschten Konfiguration ist, jedoch auf dem tatsächlichen physischen Adressbereich des Gastes basiert. Diese Schattenkonfiguration wird anstelle der vom Gast erstellten Konfiguration an die MMU übergeben. Hiermit ist eine Konfiguration der MMU nach Vorgaben des Gastes möglich, die gleichzeitig den Speicherschutz durch den Host berücksichtigt. Abb. 6 skizziert dieses Verfahren. Es ist primär in Software implementiert. Somit können auch MMUs genutzt werden, die nicht für die Virtualisierung angepasst sind. Der Nachteil dieses Vorgehens ist, dass es bei jeder vom Gast herbeigeführten Änderung der MMU-Konfiguration und somit bei jedem Programmwechsel (Multitasking) innerhalb des Gastes durchlaufen werden muss. Da hiermit zum Teil erhebliche Performanceeinbußen verbunden sind, besitzen moderne Prozessoren eine funktionserweiterte MMU.

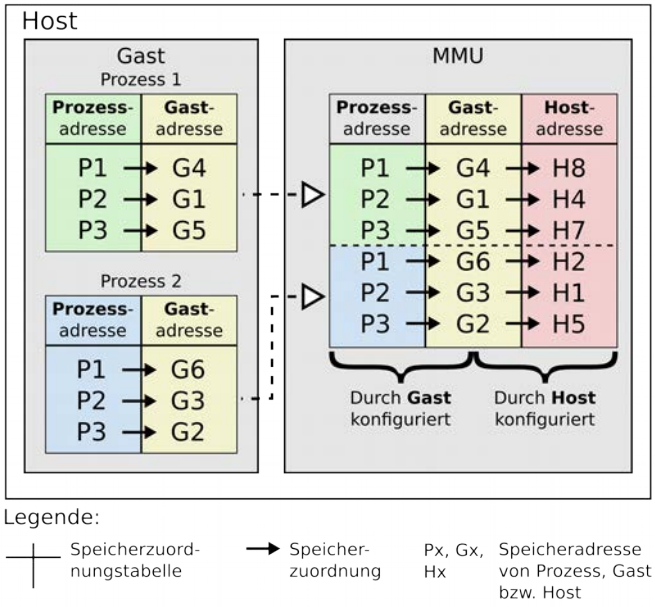

Abbildung 7: Verschachteltes Speichermanagement

Die in diesem Zusammenhang relevanten Techniken sind Extended Page Tables (EPT) von Intel und Rapid Virtualization Indexing (RVI) von AMD. Wenngleich sich beide Ansätze im Detail unterscheiden, so ist der Grundgedanke identisch. Anstatt ein klassisches eindimensionales Nachschlagewerk in der MMU zu verwenden, wird eine zweite Ebene eingeführt. Dies ermöglicht es dem Hypervisor, die MMU so weit zu konfigurieren, dass anschließende Änderungen durch den Gast selbstständig durchgeführt werden können, ohne dass dieser dabei den Speicherschutz unterlaufen kann. Dieses zweistufige Verfahren ist in Abb. 7 skizziert. Während die erste Stufe der erweiterten MMU, die der klassischen MMU entspricht, vom Gast selbstständig manipuliert werden kann, ist eine Änderung der zweiten Stufe allein dem Hypervisor vorbehalten. Der Vorteil dieses Verfahrens ist insbesondere die erhöhte Performance im Vergleich zur Nutzung einer SoftMMU (s. [VMWare]). Dies ergibt sich dadurch, dass die Anzahl der Kontextwechsel von dem Guestspace in den Kernelspace typischerweise stark reduziert wird.

3.3.3 Interruptbehandlung

Ein wesentlicher Kommunikationsmechanismus sind Interrupts. Diese ermöglichen die vorübergehende Unterbrechung des auf der CPU ausgeführten Programms. Hierzu wird eine Unterbrechungsanforderung (Interrupt Request, IRQ) an die CPU gesendet. Diese unterbricht daraufhin die reguläre Programmausführung und setzt die Befehlsabarbeitung in einer Unterbrechungsroutine (Interrupt Service Routine, ISR) fort. Nach Abarbeitung der Unterbrechungsroutine wird die Ausführung des ursprünglichen Programms wieder aufgenommen. Mit diesem Mechanismus kann somit auf Ereignisse reagiert werden, die asynchron zur Programmausführung auftreten. Unterbrechungsanforderungen an eine CPU können durch Hardwarekomponenten (z. B. PCI-Geräte) und in Mehrprozessorsystemen auch durch andere CPUs gestellt werden. Auf diesem Wege kann beispielsweise eine Netzwerkkarte das Betriebssystem, genauer gesagt den Treiber im Betriebssystem, informieren, sobald ein Netzwerkpaket empfangen wurde. Interrupts werden ferner genutzt, um durch periodische Unterbrechungen der Programmausführung ein präemptives Scheduling umzusetzen. Technisch umgesetzt wird dieser Mechanismus mittels eines Advanced Programmable Interrupt Controllers (APIC).[13]

13 Die Nutzung eines Programmable Interrupt Controllers (PIC), die die Vorgängertechnik des APCIs ist, wird zur Servervirtualisierung aus Performancegründen praktisch nicht mehr eingesetzt. Aus diesem Grund bleibt sie im Weiteren unberücksichtigt. Ferner wird auf Weiterentwicklungen oder Ergänzungen des APCIs (wie xAPIC,

Im Rahmen der Virtualisierung sind hierbei verschiedene Szenarien zu berücksichtigen. Zum einen kann ein reales Gerät des Hosts eine Unterbrechungsanforderung an eine CPU stellen. Sofern die CPU zum entsprechenden Zeitpunkt ein gewöhnliches Programm ausführt, kann der IRQ, wie oben beschrieben, unmittelbar verarbeitet werden. Führt die CPU hingegen einen Gast aus, so wird die Ausführung des Gastes unterbrochen, bevor die Unterbrechungsroutine des Hosts aufgerufen wird.[14] Somit ist sichergestellt, dass die Interaktion des Hosts mit realen Geräten vom Gast isoliert durchgeführt wird.

Darüber hinaus ist es aber für die Emulation von Hardware notwendig, dass auch virtuelle Geräte Interrupts auslösen können. Aus Sicht das Gastes scheint ein solcher Interrupt von der vermeintlichen Hardware erzeugt worden zu sein. Tatsächlich ist der Urheber des IRQs jedoch Software in Form des Hypervisors.

3.3.4 Hardwaredurchreichung

Neben der Emulation von Geräten ist es möglich, reale Geräte, die mittels PCI am Host angeschlossen sind, an virtuelle Maschinen durchzureichen. Um dies zu ermöglichen, ohne die Isolation des Gastes vom Host zu gefährden, muss die Kommunikation zwischen Gast und PCI-Gerät abgesichert werden. Hier ist insbesondere hervorzuheben, dass das PCI-Gerät eine aktive Komponente ist, vergleichbar mit der CPU. Es ist daher nicht nur der CPU möglich, Zugriffe auf das PCI-Gerät über die in Abschnitt 3.2 erläuterten Adressräume durchzuführen. Auch das PCI-Gerät selbst ist imstande, selbsttätig Benachrichtigungen in Form von Interrupts an die CPU zu versenden. Ein integraler Bestandteil von PCI ist es ferner, dem Gerät sowohl lesenden als auch schreibenden Zugriff auf den Arbeitsspeicher[15] zu geben. Die resultierenden Kommunikationswege innerhalb des Hosts sind in Abb. 8 dargestellt.

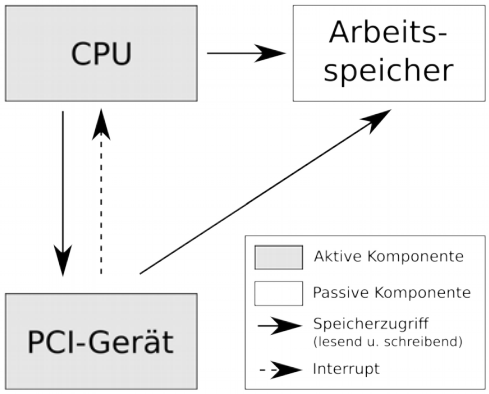

Abbildung 8: Zugriffsmöglichkeiten eines PCI-Geräts

Ein PCI-Gerät hat somit prinzipiell Zugriff auf alle wesentlichen Systemressourcen. Eine unmittelbare Interaktion zwischen einem Gast und einem PCI-Gerät würde ohne weitere Sicherungsmaßnahmen daher regelmäßig zu einem Verlust der Isolation zwischen Gast und Host führen. Soll ein Gerät dennoch an einen

x2APIC und APICv) im Rahmen dieser Studie nur eingegangen, sofern dies aus sicherheitstechnischen Gesichtspunkten sinnvoll ist.

14 Prinzipiell ist es auch möglich, die CPU so zu konfigurieren, dass der IRQ des realen Gerätes durch das Gastbetriebssystem verarbeitet wird. Da dies jedoch regelmäßig nicht zweckdienlich ist und zu erheblichen Sicherheitsbeeinträchtigungen des Systems führt, ist eine entsprechende Konfiguration in der Praxis nicht anzutreffen. Eine Ausnahme stellt jedoch die Technik APICv der Firma Intel im Zusammenhang mit der Durchreichung von Hardware dar.

15 Präziser ausgedrückt, besteht eine Zugriffsmöglichkeit sowohl auf den IO- als auch auf den MMIO-Adressraum. Somit ist es einem PCI-Gerät nicht nur möglich, auf den Arbeitsspeicher, sondern auch auf andere PCI-Geräte zuzugreifen.

Gast durchgereicht werden, so ist die Beschränkung der Zugriffsmöglichkeiten des jeweiligen Gerätes unabdingbar.

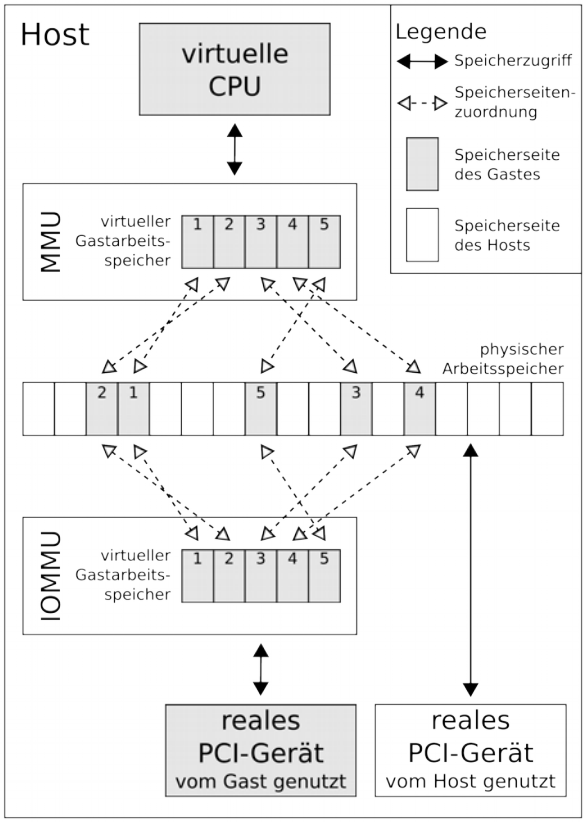

Abbildung 9: IOMMU

Um dies zu ermöglichen, bedarf es einer in die Hardware[16] integrierten IOMMU.[17] Sie übersetzt die Zieladressen, die das PCI-Gerät beim Zugriff auf den Arbeitsspeicher nutzt. Die Konfiguration der IOMMU und damit die Festlegung der Übersetzungsregeln obliegt dem Hypervisor. Er ist somit in der Lage, dem Gast und dem Gerät eine gleiche Sicht auf den Arbeitsspeicher zu geben und hierbei den Zugriff auf den Speicher einzuschränken. Die grundsätzliche Funktionsweise einer IOMMU ist in Abb. 9 dargestellt.

16 Die IOMMU ist nicht wie die MMU Bestandteil der CPU, sondern der PCI-Host-Bridge und somit eine Komponente des Chipsatzes. Die Frage, ob ein System die Durchreichung von Hardware unterstützt, ist daher nicht primär von der eingesetzten CPU abhängig, sondern von dem verwendeten Chipsatz.

17 Alternativ ist eine Durchreichung von Hardware mittels Modifizierung des Gastbetriebssystems ohne Verwendung einer IOMMU denkbar. Einen solchen Ansatz unterstützt beispielsweise der Hypervisor Xen. QEMU und KVM ermöglichen dies nicht.

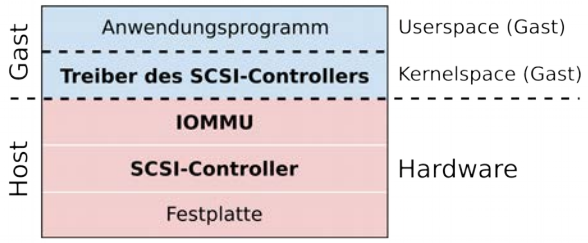

Abbildung 10: Durchreichung von Hardware

Auf diesem Wege ist es einem Gast möglich, weitestgehend ohne Interaktion des Hosts auf Hardware zuzugreifen. Der Zugriff erfolgt nicht mittelbar über den Hypervisor, sondern unmittelbar über den PCI-Bus. Im engeren Sinne handelt es sich bei diesem Vorgehen daher nicht um eine Virtualisierung, sondern um eine Isolation der Hardware. Abb. 10 veranschaulicht die Konfiguration eines Hosts, der einen SCSI-Controller an einen Gast durchreicht.

Darauf aufbauend existieren Techniken, die es ermöglichen, ein Gerät an mehrere Gäste durchzureichen. Insbesondere die Single Root Input/Output Virtualization (SR-IOV) ist hier von Bedeutung. SR-IOV ermöglicht es einer einzelnen physischen Netzwerkkarte, mehrere virtuelle Netzwerkkarten über den PCI-Bus zur Verfügung zu stellen. Sie verfügen über eigene spezifische MAC-Adressen und Warteschlangen zur Zwischenspeicherung der empfangenen und der zu sendenden Netzwerkpakete. Die virtuellen Netzwerkkarten können mit einer IOMMU, analog zu anderen PCI-Geräten, an Gäste durchgereicht werden. Der Gast greift somit mit einem nativen Treiber auf die Hardware zu. Hierbei ist es die Aufgabe der jeweiligen Netzwerkkarte, die Zugriffe der verschiedenen Gäste voneinander zu isolieren, obwohl sie dieselbe reale Hardware nutzen. Die Sicherheit einer auf SR-IOV aufbauenden Virtualisierung hängt somit auch von der eingesetzten Hardware ab.[18]

3.4 KVM

Aufbauend auf den Funktionen moderner CPUs stellt das Kernelmodul kvm.ko grundlegende Verfahren zur Virtualisierung bereit. Es wird ergänzt durch die beiden Module kvm_intel.ko und kvm_amd.ko, die die hardwarespezifischen Bestandteile für die Nutzung von VT-x bzw. AMD-V bereitstellen. Um diese Techniken nutzen zu können, wird ein Kernelmodul benötigt, da sie die Ausführung von Code im Kernelspace erfordern.

Die primäre Aufgabe von KVM besteht darin, Anwendungssoftware (z. B. QEMU) Zugriff auf die Virtualisierungsfunktionen der Hardware zu geben. Hierzu zählen insbesondere die Erstellung und Verwaltung virtueller CPUs und die Ausführung von Code auf ihnen. Ferner ermöglicht es KVM, das Speicherlayout der virtuellen Maschine zu spezifizieren und es mithilfe der MMU umzusetzen. KVM abstrahiert hierbei weitgehend von hardwarespezifischen Eigenschaften wie der von der CPU unterstützten Virtualisierungstechnik (VT-x, AMD-V) und der konkreten Methode der Speicherverwaltung (SoftMMU, EPT, RVI).

Des Weiteren ist KVM in der Lage, die virtuellen Interrupts mithilfe eines virtuellen Advanced Programmable Interrupt Controllers (APIC) (s. Abschnitt 3.3.3) zu erstellen und hiermit das Anwendungsprogramm zu entlasten.

Die Kommunikation zwischen dem Anwendungsprogramm (QEMU) und KVM erfolgt über die Gerätedatei /dev/kvm mithilfe des Systemaufrufs ioctl (s. [IOCTL]). Die Dateiberechtigungen legen fest,

18 Eine allgemeingültige Beurteilung der Sicherheit von SR-IOV ist daher nur bedingt möglich und wird im Rahmen dieser Studie nicht angestrebt.

welche Prozesse mit KVM interagieren dürfen. Die auf diesem Wege zur Verfügung gestellten Funktionen können grob in die folgenden Kategorien eingeteilt werden:

- Ermittlung der Fähigkeiten des Virtualisierungssystems
  Dies umfasst zum einen die Bestimmung der Techniken, die von der CPU unterstützt werden, z. B. der Fähigkeit, 64-Bit-Code auszuführen. Ferner kann die Anwendungssoftware feststellen, welche Funktionen die verwendete Version des Kernelmoduls unterstützt. Dies ermöglicht es, Anwendungssoftware zu erstellen, die mit verschiedenen Versionen von KVM kompatibel ist. Ebenso kann hierdurch der Funktionsumfang von KVM erweitert werden, ohne eine Inkompatibilität mit bestehender Anwendungssoftware einzuführen.
- Erstellung einer virtuellen Maschine
  Die Funktionen zur Einrichtung einer virtuellen Maschine ermöglichen es der Anwendungssoftware, die gewünschte Konfiguration des Gastes zu spezifizieren. Hierzu gehören insbesondere die Erstellung mehrerer virtueller CPUs für eine virtuelle Maschine, die Festlegung des Speicherbereichs, auf den der Gast Zugriff bekommen soll, und optional die Einbindung eines virtuellen APICs.
- Ausführung von Code innerhalb einer virtuellen Maschine
  Die zentrale Funktion von KVM ist das (Re-)Aktivieren einer virtuellen Maschine, d. h., die Fähigkeit, Code innerhalb des jeweiligen Gastkontextes auszuführen. Darüber hinaus ermöglicht es KVM der Anwendungssoftware, eigenständig virtuelle Interrupts zu erzeugen.
- Auslesen und Setzen des Zustandes einer virtuellen Maschine
  Ferner ermöglicht es KVM der Anwendungssoftware, den Zustand der virtuellen Maschine auszulesen. Dies umfasst insbesondere die Register der virtuellen CPUs. Ebenso ist es der Software möglich, diese zu modifizieren. Dies dient sowohl zur Hardwareemulation als auch zur persistenten Speicherung des Zustandes des Gastes und zu dessen späterer Reaktivierung.

Von zentraler Bedeutung sind insbesondere die Funktionen KVM_CREATE_VM, KVM_CREATE_VCPU, KVM_SET_USER_MEMORY_REGION und KVM_RUN. Die ersten beiden dienen zur Erzeugung einer virtuellen Maschine bzw. einer virtuellen CPU. Hierzu erstellt die Anwendungssoftware je einen Thread pro virtueller CPU. Jeder dieser Threads erzeugt nun mit der Funktion KVM_CREATE_VCPU eine virtuelle CPU und ist im Anschluss für deren Verwaltung zuständig. Mit der Funktion KVM_SET_USER_MEMORY_REGION blendet die Software Speicher in den Adressraum des Gastes ein. Dies ermöglicht es der Software, eigenen Speicher gegenüber dem Gast als vermeintlichen physischen Arbeitsspeicher auszugeben.

Schließlich wird der Gastcode mit der Funktion KVM_RUN im Kontext des jeweiligen Threads ausgeführt. Diese Ausführung geschieht aus Sicht der Anwendungssoftware blockierend. Wird im Kontext der virtuellen CPU ein Ereignis ausgelöst, das eine Reaktion von der Software erfordert, wie z. B. ein Zugriff auf virtuelle Hardware, so wird in den Anwendungscode des jeweiligen Threads zurückgesprungen. Hierbei übergibt KVM der Software eine ausführliche Beschreibung der Ursache für die Unterbrechung des Gastes. Der Software ist es somit möglich, das jeweilige Ereignis abzuarbeiten. Dies kann z. B. durch die Emulation virtueller Hardware geschehen. Im Anschluss daran wird die Ausführung im Kontext der virtuellen CPU durch einen erneuten Aufruf von KVM_RUN wieder aufgenommen. Eine ausführlichere Darstellung dieses Ablaufs erfolgt am Beispiel von QEMU in Abschnitt 3.5.1.

Abschließend sei erwähnt, dass KVM bezüglich des Hosts maximal privilegiert ist, da es vollständig im Kernelspace ausgeführt wird.

3.5 QEMU

3.5.1 Architektur

QEMU übernimmt einen Großteil der Funktionen, die für die Virtualisierung notwendig sind. Es nutzt hierzu KVM[19] und stellt eine Schnittstelle zur Steuerung zur Verfügung. Die Schnittstelle ermöglicht es, virtuelle Maschinen zu verwalten. Dies beinhaltet unter anderem:

- das Starten und Stoppen virtueller Maschinen
- das Erstellen und Wiederherstellen von Snapshots
- die Live-Migration virtueller Maschinen
- das Hinzufügen und Entfernen virtueller Geräte
- das Auslesen und Modifizieren des Arbeitsspeichers des Gastes

Der Zugriff auf diese Schnittstelle ermöglicht somit einen uneingeschränkten Zugriff auf den Gast. Im Falle eines nicht legitimierten Zugriffs sind daher sowohl die Vertraulichkeit als auch die Integrität und die Verfügbarkeit verletzt. Die Schnittstelle unterstützt keine Autorisierung, und ihr liegt kein Rollen- oder Benutzermodell zugrunde. Jede Anwendung, die mit dieser Schnittstelle von QEMU interagieren kann, besitzt daher weitreichende Privilegien bezüglich des jeweiligen Gastes.

QEMU kann diese Schnittstelle auf unterschiedlichen Wegen zur Verfügung stellen. Dies umfasst die Bereitstellung über TCP und somit die direkte Administration über ein Netzwerk. Die entsprechende Kommunikation ist allerdings nicht vor Einsicht oder Veränderung geschützt. Hieraus ergeben sich erhebliche Nachteile für die Sicherheit des Systems. Im Falle eines entsprechenden Einsatzes ist zum Schutz der virtuellen Maschinen daher zwingend eine verlässliche Zugriffsbeschränkung auf Netzwerkebene erforderlich. Aufgrund dieser Nachteile wird die Schnittstelle in produktiven Umgebungen in aller Regel nicht über das Netzwerk verfügbar gemacht. Stattdessen sichert sich libvirt einen exklusiven lokalen Zugriff. Realisiert wird dies durch die Nutzung von Pipes (s. [UNIX-PROG]), die libvirt beim Aufruf von QEMU erstellt. Darauf aufbauend bietet libvirt eine eigene abgesicherte Schnittstelle zum Zugriff über ein Netzwerk an. Aus diesen Gründen sind die potenziellen Sicherheitsrisiken, die durch das Fehlen einer Autorisierung und die ungeschützte Datenübertragung der QEMU-API entstehen, in der Praxis wenig relevant.

19 Auf den von KVM unabhängigen Betrieb von QEMU durch vollständige Emulation der CPU wird nicht eingegangen, da dieser aus Performancegründen für die Servervirtualisierung nicht relevant ist.

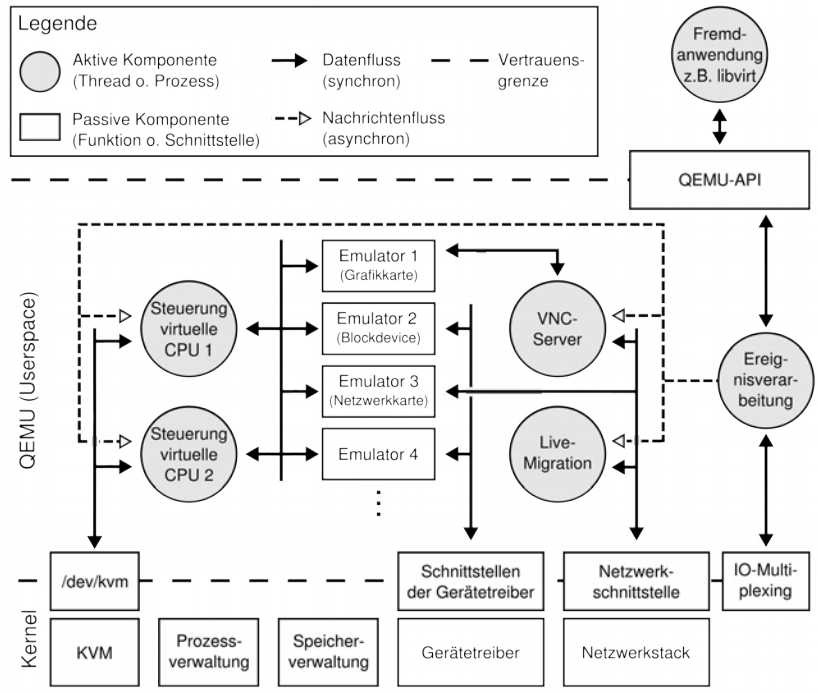

Abbildung 11: QEMU-Architektur

QEMU basiert auf einer modularen, ereignisbasierten und parallelen Architektur, die in Abb. 11 dargestellt ist. Ein QEMU-Prozess führt immer genau eine virtuelle Maschine aus. Sollen mehrere Gäste auf einem Host ausgeführt werden, so sind mehrere QEMU-Prozesse erforderlich. Jeder QEMU-Prozess verwaltet hierbei alle virtuellen CPUs des Gastes mithilfe von Threads. QEMU startet für jede CPU genau einen Thread, der für die Verwaltung der jeweiligen virtuellen CPU zuständig ist. Dies ermöglicht eine parallele Ausführung mehrerer virtueller CPUs und stellt dem Gast daher echte Multiprozessorfähigkeit zur Verfügung. Die CPU-Threads nutzen KVM zur Ausführung des Gastcodes. Der hierfür genutzte Aufruf von KVM_RUN (s. Abschnitt 3.4) ist blockierend. Der Thread wartet also darauf, dass die Ausführung des Gastcodes unterbrochen wird. Ein solche Unterbrechung geschieht regelmäßig beim Zugriff auf virtuelle Geräte, die durch QEMU bereitgestellt werden. Auf Grundlage der Informationen, die KVM über den Abbruchgrund bereitstellt, bestimmt QEMU das weitere Vorgehen.

Beim Zugriff auf einen virtuellen Adressraum (IO- oder MMIO-Adressraum) ermittelt der zuständige CPU-Thread anhand der jeweiligen Zieladresse den zuständigen Emulator (s. u.) und ruft diesen auf, damit er den lesenden oder schreibenden Zugriff verarbeitet. Die konkrete Funktionsweise des Emulators hängt von der emulierten Komponente ab. Nachdem der Emulator den Speicherzugriff abgearbeitet hat, wird KVM angewiesen, die Ausführung des Gastcodes wieder aufzunehmen.

Neben dem Zugriff auf virtuelle Geräte existieren weitere Ursachen dafür, die Ausführung von Gastcode zu unterbrechen. Zum Beispiel kann die Ausführung des Gastes durch einen anderen Thread von QEMU unterbrochen werden; der Gast könnte abgeschaltet werden oder es könnte ein Fehler bei der Virtualisierung auftreten. Ein vereinfachter Programmablaufplan eines CPU-Threads wird in Abb. 12 dargestellt.

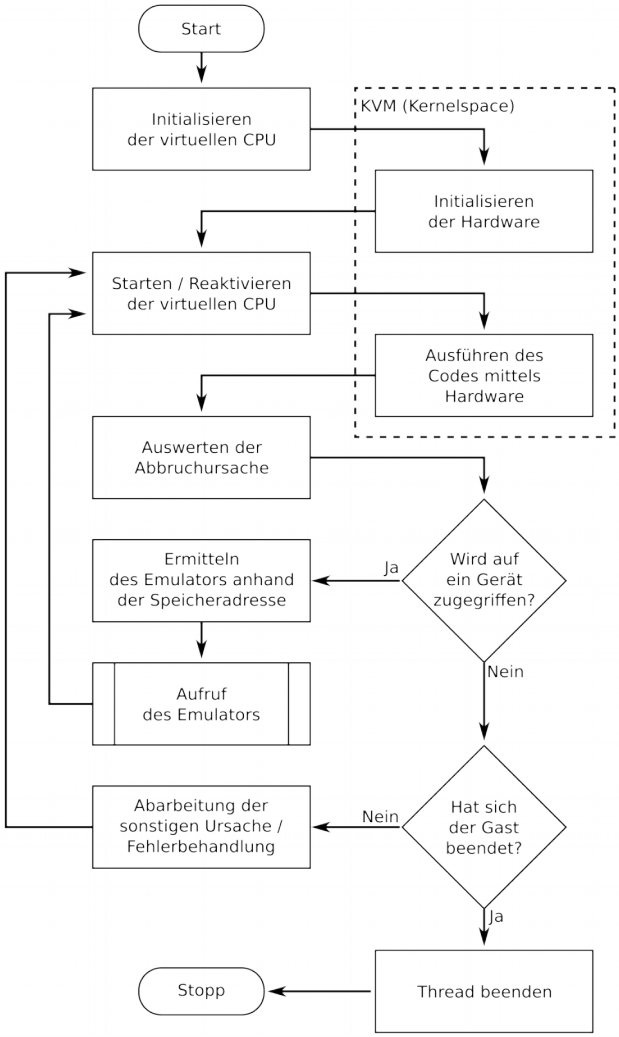

Abbildung 12: Vereinfachter Programmablaufplan eines CPU-Threads

Die Bereitstellung virtueller Geräte erfolgt anhand von Emulatoren. Diese sind modulare Bestandteile von QEMU und werden über eine weitestgehend einheitliche Schnittstelle von QEMU aufgerufen. Dieser Aufbau ermöglicht das vergleichsweise einfache Hinzufügen weiterer Emulatoren nach dem Baukastenprinzip. Der interne Aufbau der Emulatoren ist zweigeteilt. Zum einen bietet das Frontend des Emulators eine Schnittstelle zum Gast. Diese erlaubt es dem Gast mit dem Emulator und somit mit QEMU zu interagieren. In der Regel basiert das Frontend auf der Emulation eines PCI-Geräts. Daneben ermöglicht es das Backend, die Zugriffe über den Linux-Kernel an reale Geräte weiterzureichen. Es existieren verschiedene Arten von Backends, die die Anbindung an die Hardware auf unterschiedlichem Wege ermöglichen. Die wesentlichen sind zum Zugriff auf folgende Gerätearten bestimmt:

- Blockgeräte
- Zeichengeräte
- Netzwerkfunktionen
- virtuelle Konsole

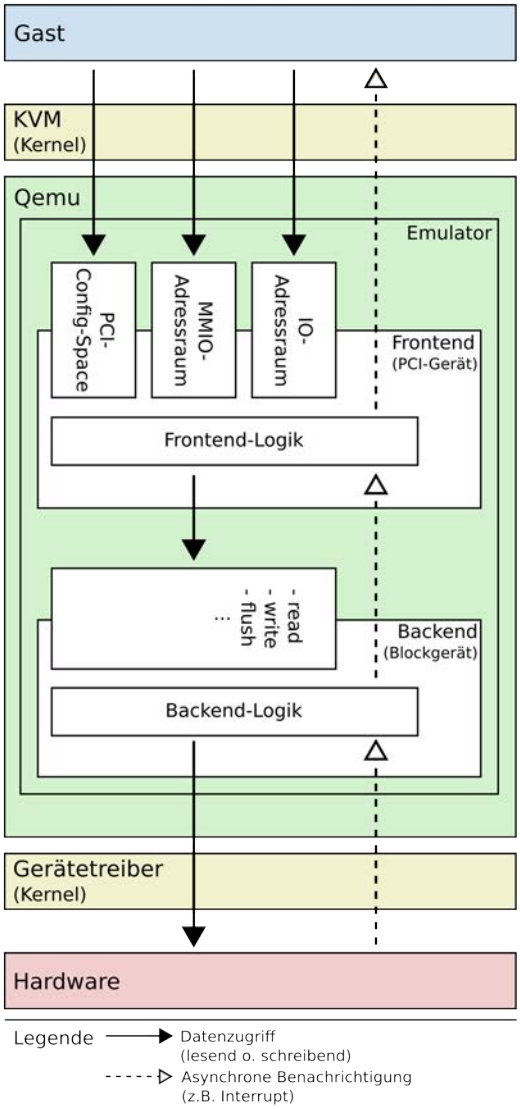

Abbildung 13: Emulator eines PCI-Geräts mit einem Blockgerät als Backend

In Abb. 13 wird der Aufbau anhand eines virtuellen PCI-Geräts mit Anbindung an ein Blockgerät skizziert. Konkret könnte es sich bei dem Frontend z. B. um einen IDE-Controller und bei dem Backend um eine lokale Datei handeln.

Der wesentliche Vorteil des zweiteiligen Aufbaus ist die Möglichkeit der flexiblen Kombination von Front- und Backend (s. auch Abschnitt 3.5.3). Beispielsweise kann ein Frontend, das dem Gast einen virtuellen IDE-Controller zur Verfügung stellt, ein Backend nutzen, das die Zugriffe auf eine lokal gespeicherte Datei umleitet. Dasselbe Frontend kann aber ebenso mit einem Backend zusammenarbeiten, das das Festplattenabbild über iSCSI zur Verfügung stellt. Aus Sicht des Gastes erfolgt der Zugriff in beiden Fällen über den virtuellen IDE-Controller. Der Gast erkennt keinen Unterschied. Ebenso ist es aber auch möglich, die Backends mit einem Frontend zu kombinieren, das den Zugriff des Gastes mithilfe von Paravirtualisierung (Virtio) ermöglicht. Eine erneute Implementierung der Anbindung an iSCSI für diese weitere Bereitstellungsart ist somit nicht notwendig.

Neben den Threads zur Steuerung der virtuellen CPUs existieren weitere Threads. Zum einen ist der Thread zur Ereignisverarbeitung (s. Abb. 11) zu erwähnen. Er dient zur Reaktion auf Ereignisse, die asynchron zur Ausführung des Gastcodes auftreten, wie z. B. das Eintreffen eines Netzwerkpaketes über eine reale Netzwerkkarte. Hierbei wird die eigentliche Abarbeitung solcher Ereignisse typischerweise nicht durch diesen Thread selbst ausgeführt. Stattdessen informiert er andere Threads über das Ereignis. Im Beispiel des empfangenen Netzwerkpaketes sorgt der Thread beispielsweise dafür, dass KVM die Abarbeitung von Gastcode unterbricht. Der CPU-Thread injiziert daraufhin bei der Wiederaufnahme der Gastausführung einen Interrupt in die virtuelle CPU und informiert somit den Gast über das eingegangene Netzwerkpaket.

Daneben existiert ein Thread für die Bereitstellung einer grafischen Konsole mittels VNC oder Spice. Er erhält seine Informationen von dem Emulator der Grafikkarte. Dieser Dienst wird üblicherweise über ein Netzwerk bereitgestellt. Er ist regelmäßig der einzige Weg, um unmittelbar über das Netzwerk auf QEMU zuzugreifen. Zwar existieren Sicherungsmaßnahmen, wie z. B. ein Passwortschutz, dennoch stellt der VNC- oder Spice-Dienst aufgrund seiner exponierten Stellung ein erhöhtes Sicherheitsrisiko dar. Aus diesem Grund wird der Zugriff aus dem Netzwerk üblicherweise beschränkt.

Des Weiteren existiert ein Thread zur Umsetzung der Live-Migration, der nur im Rahmen einer Migration aktiv wird. In dem QEMU-Prozess, der die zu migrierende virtuelle Maschine steuert, versendet dieser Thread unter anderem ein Abbild des Arbeitsspeichers. Ein zweiter QEMU-Prozess, dessen Migrations-Thread die Daten in Empfang nimmt, rekonstruiert den Zustand der virtuellen Maschine. QEMU bietet bezüglich der übertragenen Daten keinen Schutz der Vertraulichkeit oder Integrität. Wenngleich dies prinzipiell erhebliche Risiken birgt, so sind diese doch wenig praxisrelevant, da beim Einsatz von libvirt eine Absicherung der Datenübertragung durch den Prozess libvirtd stattfindet. Ebenso unterbindet libvirt auch zum Zeitpunkt der Migration die unmittelbare Erreichbarkeit des Migrations-Threads über das Netzwerk (s. Abschnitt 3.6.1).

Abschließend lässt sich festhalten, dass innerhalb des im Userspace ausgeführten Teils des QEMU-Prozesses alle Komponenten gleich privilegiert sind und vollen Zugriff auf den Zustand und insbesondere auf den Arbeitsspeicher des Gastes haben. Zwischen den Threads existieren somit keine Vertrauensgrenzen. Dies bedingt, dass eine Schwachstelle innerhalb einer dieser Komponenten (z. B. in einem Emulator) weitreichende Konsequenzen für den Gast haben kann. Ebenso besteht auch für den Host und somit für andere Gäste ein erhöhtes Bedrohungspotenzial, sofern es einem Angreifer gelingt, den QEMU-Prozess zu kompromittieren. Die Bedrohung erhöht sich, sofern der QEMU-Prozess mit root-Rechten gestartet wird. Die Widerstandsfähigkeit von QEMU ist daher wesentlich für die Sicherheit des gesamten Systems.

3.5.2 Virtuelle Infrastruktur

Die zentrale Aufgabe von QEMU ist die Emulation virtueller Geräte und Bussysteme. Aufbauend auf den virtuellen CPUs, die KVM zur Verfügung stellt, realisiert QEMU daher die Infrastruktur der virtuellen Maschine. QEMU stellt hierzu drei virtuelle Umgebungen mit den folgenden emulierten Chipsätzen zur Verfügung:

1. 440FX (82441FX) und PIIX3 (82371SB)
2. 82Q35 und ICH9 (82801IB)
3. ISA-only PC (entspricht Punkt 1 ohne PCI-Bus)

Standardmäßig wird die Variante aus Punkt 1 genutzt. Auch wenn der Chipsatz bereits im Jahr 1996 eingeführt wurde, so bietet er doch auch weiterhin eine sinnvolle Grundlage der Virtualisierung, da seine Treiber in alle relevanten Betriebssysteme integriert sind. Außerdem hat sich die Robustheit der Emulation im Laufe der Zeit bewährt. Sofern virtuelle Geräte nicht in den Chipsatz integriert sind, wie z. B. ein USB-3.0-Hostcontroller, können sie über den virtuellen PCI-Bus des Gastes hinzugefügt werden.

Die in Punkt 2 aufgeführte Kombination der Chips 82Q35 und ICH9 ist deutlich aktueller als die bereits beschriebene. Sie bietet aber nur wenige Vorteile, und ihr fehlen einige der Funktionen, die die erste Kombination bietet. Die Verwendung des Chipsatzes kann sinnvoll sein, sofern reale PCI-Express-Geräte

vom Gast genutzt werden sollen, da der Controller 440FX PCI-Express nicht unterstützt und eine entsprechende Durchreichung daher nicht ermöglicht.

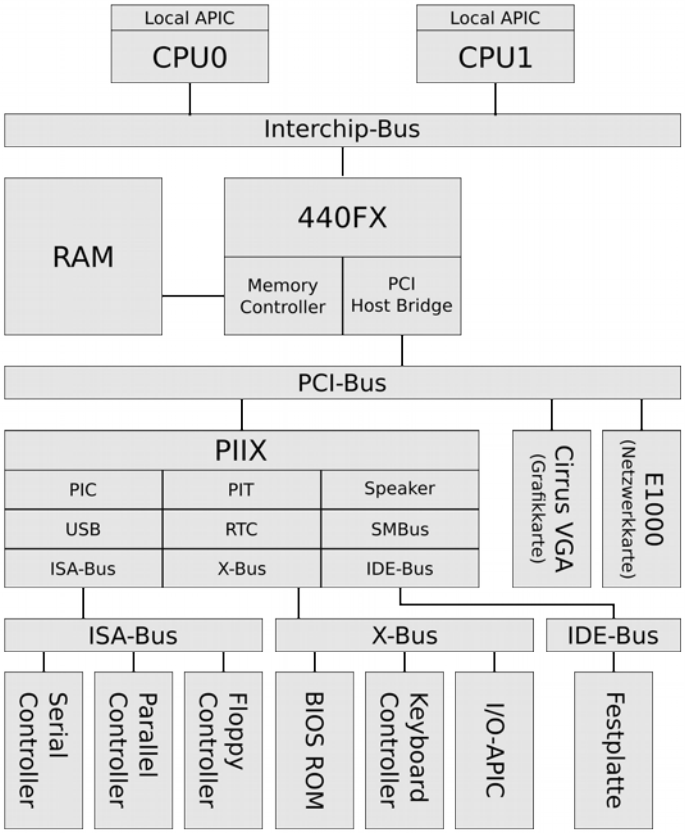

Abbildung 14: Architektur eines QEMU-Gastes

Die in Punkt 3 genannte Architektur entspricht im Wesentlichen den unter Punkt 1 genannten Komponenten. Ihr fehlt aber ein virtueller PCI-Bus. Aus diesem Grund ist diese Variante für den Einsatz in der Servervirtualisierung in aller Regel ungeeignet. Sie wird daher im Weiteren nicht berücksichtigt.

Abb. 14 skizziert eine virtuelle Umgebung mit zwei virtuellen CPUs basierend auf dem Chipsatz, bestehend aus Intel 440FX und PIIX3. Neben dem eigentlichen Chipsatz sind zur Veranschaulichung die virtuellen Geräte eingezeichnet, die QEMU standardmäßig erstellt. Sie erweitern die Grundfunktionen, die durch den Chipsatz bereitgestellt werden. In Anlehnung an ihre realen Gegenstücke wird eine Reihe von historischen Komponenten (wie z. B. die serielle Schnittstelle, das Diskettenlaufwerk und die Tastatur) über klassische Bussysteme angebunden.[20] Der bedeutendste Bus der virtuellen Maschine ist jedoch der PCI-Bus, der auch zur Anbindung hochperformanter virtueller Geräte genutzt werden kann. Zu nennen sind hier beispielsweise die Bereitstellung von Grafikkarten zur Darstellung einer Konsole oder die Bereitstellung von Netzwerkkarten, um die Dienste des Gastes an ein Netzwerk anzubinden. Zugriffe des Gastes werden, wie in Abschnitt 3.4 beschrieben, an QEMU weitergeleitet.

20 Ebenso wie ihre realen Gegenstücke ist eine Reihe dieser historischen Komponenten heute auch in der Virtualisierung kaum mehr anzutreffen.

3.5.3 Virtuelle Geräte

Wie bereits in Abschnitt 3.5.1 beschrieben, werden virtuelle Geräte durch Frontends und Backends realisiert. In diesem Kapitel werden verschiedene Frontends und Backends beschrieben und auf deren technische Umsetzung eingegangen.

3.5.3.1 Emulation

Das Frontend eines Emulators, d. h. die Schnittstelle zwischen Gast und QEMU, ahmt klassischerweise das Verhalten realer Hardware nach. Das Gastsystem kann somit den unveränderten Treiber der Hardware in der Virtualisierung nutzen. Eine Anpassung des Gastes oder von dessen Treibern ist daher nicht notwendig.

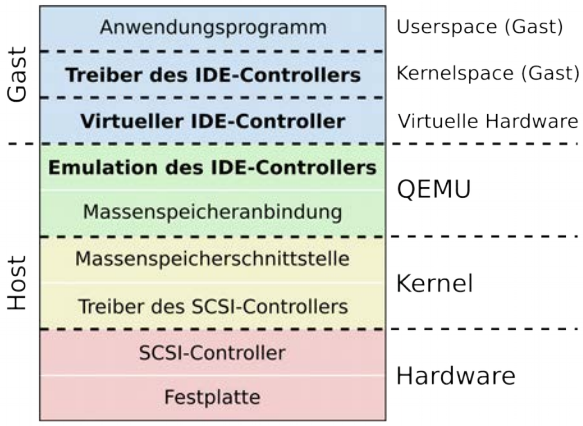

Abbildung 15: Emulation einer Festplatte durch QEMU

Abb. 15 zeigt den Aufbau am Beispiel der Emulation eines IDE-Festplattencontrollers. Hierbei stellt QEMU dem Gast einen emulierten Festplattencontroller so zur Verfügung, dass der Gast den originalen Treiber der emulierten Hardware für den Zugriff auf das Gerät nutzen kann. Auf die eigentliche Datenquelle greift QEMU mithilfe des Host-Kernels zu. Hierbei kann die Datenquelle unabhängig von der Anbindungsmethode des Gastes gewählt werden. In Abb. 15 wird beispielsweise eine lokale Datei auf einer SCSI-Festplatte als Quelle genutzt. Der Zugriff wird daher von dem entsprechenden Treiber des Hosts verarbeitet, indem dieser über den SCSI-Controller auf die tatsächliche Hardware zugreift.

3.5.3.2 Paravirtualisierung

Neben diesem klassischen Emulationsansatz existiert mit der Paravirtualisierung eine weitere Möglichkeit der Gerätebereitstellung. Hierbei werden dem Gast virtuelle Geräte bereitgestellt, die keine reale Entsprechung haben. Hierdurch ist es möglich, eine Schnittstelle zwischen Gast und Host zu schaffen, die für die Anforderungen der Virtualisierung optimiert ist. Das heißt, dass bei dieser Software-zu-Software-Schnittstelle auf performancereduzierende Eigenschaften verzichtet wird, die typisch für die Nachahmung einer Software-zu-Hardware-Schnittstelle sind. Hierdurch wird regelmäßig eine deutliche Performanceverbesserung gegenüber der Emulation von Geräten erzielt. Die Paravirtualisierung erfordert allerdings die Anpassung des Gastes durch entsprechende Treiber.

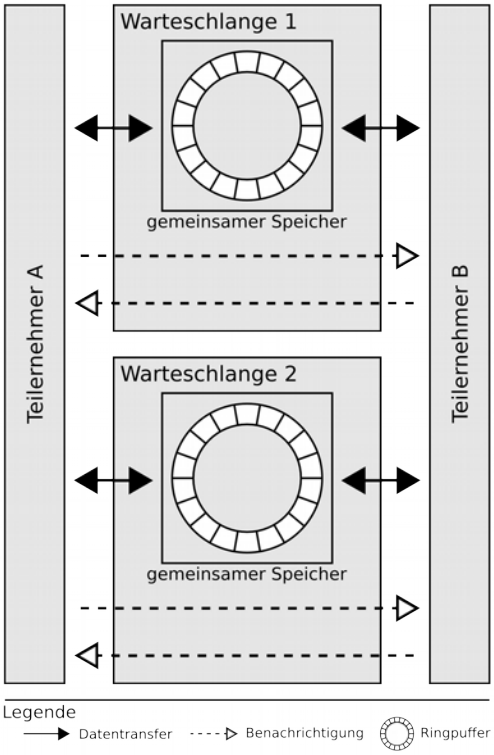

Abbildung 16: Kommunikation mittels Virtio

Technisch umgesetzt wird die Paravirtualisierung mit der offenen Schnittstelle Virtio ([OAS]). Deren Spezifikation setzt sich aus drei Teilen zusammen. Der erste Teil stellt eine generische Schnittstelle zur Übertragung von Daten vom Gast zum Host und umgekehrt zur Verfügung. Hierzu wird ein Ringpuffer in einem gemeinsam genutzten Speicherbereich verwendet, auf den beide Kommunikationsteilnehmer lesenden wie schreibenden Zugriff besitzen. Ferner ist es den Teilnehmern möglich, den Kommunikationspartner über einen einfachen asynchronen Benachrichtigungsmechanismus über Änderungen am Ringpuffer zu unterrichten. Die Kombination aus Ringpuffer und Benachrichtigungsmechanismus bildet eine Warteschlange. Zwischen den Kommunikationsteilnehmern können mehrere solcher Warteschlangen genutzt werden. Der grundlegende Aufbau mit Teilnehmern, die über zwei Warteschlangen kommunizieren, ist in Abb. 16 dargestellt.

Der zweite Teil der Spezifikation beschreibt, wie die Warteschlangen durch die Teilnehmer genutzt werden, z. B. wie der Benachrichtigungsmechanismus technisch umgesetzt wird. Wesentlich für diese Studie ist hierbei lediglich die Anbindung über einen virtuellen PCI-Bus.

Die Bedeutung und das Format der Daten sowie ein Protokoll, dass das kooperative Verhalten der Kommunikationsteilnehmer bestimmt, wird durch den dritten Teil der Spezifikation vorgegeben. Er schreibt z. B. vor, wie viele Warteschlangen ein SCSI-Controller hat, der auf Virtio aufbaut, und wie die Daten im Ringpuffer zu interpretieren sind. Die praxisrelevanten durch Virtio spezifizierten Geräte sind:

- allgemeines Blockgerät
- SCSI-Gerät
- Netzwerkkarte
- Arbeitsspeicher (Balloon)
- Konsole
- Zufallszahlengenerator
- Dateisystem-Host-Sharing

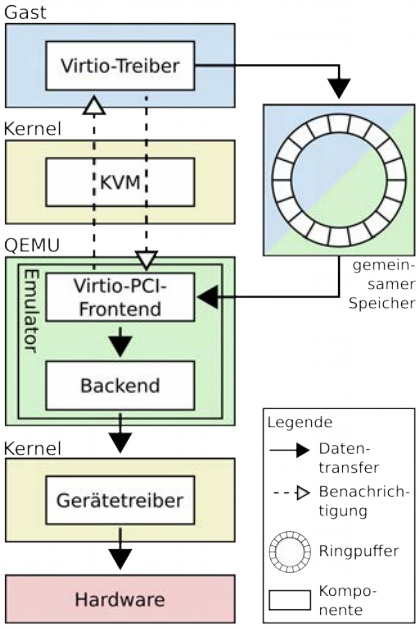

Abbildung 17: Architektur von Virtio

Abb. 17 stellt die Verwendung von Virtio durch QEMU im Rahmen einer Datenübertragung vom Gast zum Host anhand einer einzelnen Warteschlange dar. Der Vorgang könnte beispielsweise Teil der Kommunikation des Gastes mit einer virtuellen Netzwerkkarte sein, die paravirtualisiert zur Verfügung gestellt wird. Der Gast greift mithilfe des Virtio-Treibers auf den Ringpuffer im gemeinsam genutzten Speicher zu und hinterlegt dort die jeweiligen Daten. Danach benachrichtigt er das Virtio-PCI-Fontend, das von QEMU bereitgestellt wird, über die Änderung. Das Virtio-PCI-Fontend liest nun seinerseits die Daten aus dem Ringpuffer und interpretiert diese. Die Daten werden vom Frontend für das Backend aufbereitet und an dieses übersandt. Als letztes greift das Backend über den entsprechenden Gerätetreiber des Linux-Kernels auf die Hardware zu.

Dieser Vorgang ist aus Sicht des Hosts technisch weitgehend identisch mit der Emulation (s. Abschnitt 3.5.3.1). Der wesentliche Unterschied besteht in der vereinheitlichten und an die Anforderungen der Virtualisierung angepassten Kommunikation zwischen Gast und QEMU.[21]

Abb. 18 skizziert das Vorgehen am Beispiel der Bereitstellung einer virtuellen Festplatte mittels Virtio. Anders als im Beispiel aus Abb. 15 wird der Zugriff auf die virtuelle Hardware nicht mithilfe eines IDE-Controllers, sondern über die Virtio-Schnittstelle durchgeführt. Die weitere Verarbeitung erfolgt analog zu dem vorangegangenen Beispiel.

3.5.3.3 Backends

Die für diese Studie relevanten Backends dienen zur Anbindung der Frontends an Blockgeräte und an Netzwerke. Letzteres wird in Abschnitt 3.5.4 separat behandelt. Blockgeräte werden primär zur Bereitstellung der Massenspeicher, d. h. der virtuellen Festplatten, genutzt und bieten hierzu typischerweise sowohl schreibenden als auch lesenden Zugriff auf eine Datenquelle. Die von QEMU unterstützten und für die Virtualisierung von Serversystemen relevanten Blockgeräte sind:

21 Das Vorgehen zur Implementierung eines paravirtualisierten Geräts in QEMU ist mit dem Vorgehen zur Implementierung eines emulierten Geräts weitgehend identisch. Deshalb wird in dieser Studie die entsprechende Implementierung auch bei Nutzung der Paravirtualisierung als Emulator bezeichnet.

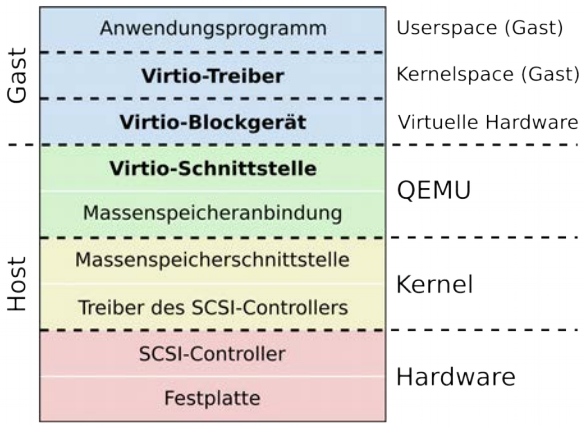

Abbildung 18: Nutzung von Virtio

- Datei
  Das Einbinden von Dateien als Backend bietet einen einfachen Weg, Datenträger zu verwalten. Sofern die Datei auf dem Host vorgehalten wird, ist hierzu keine weitere Infrastruktur erforderlich. Somit ist diese Art der Datenhaltung insbesondere in kleinen Virtualisierungsumgebungen zweckdienlich. Zu beachten ist, dass die Datei jedoch nicht zwingend auf dem Host gespeichert werden muss. Der Host kann auch entfernte Dateisysteme einbinden, so dass diese von QEMU wie lokal gespeicherte Dateien genutzt werden können. Dies ist unter anderem dann notwendig, wenn eine Live-Migration der virtuellen Maschinen durchgeführt werden soll, da beide beteiligten Hosts im Zuge der Migration Zugriff auf die Festplattenabbilder benötigen. Die Vertraulichkeit und Integrität lokal gespeicherter Dateien kann außer durch Dateizugriffsrechten über zusätzliche Maßnahmen geschützt werden, wie SELinux und AppArmor (s. Abschnitt 3.6.3.1). Beim Zugriff über ein entferntes Dateisystem hängen die vorhandenen Schutzmaßnahmen von dem Dateisystem und der Art der Einbindung durch den Host ab.
- iSCSI
  Die Anbindung an dedizierte Speicherhardware kann mittels iSCSI erfolgen. Hierbei werden die Zugriffe auf das Blockgerät von QEMU in SCSI-Befehle übersetzt und über das Netzwerk übertragen. QEMU nimmt dabei die Rolle eines iSCSI-Initiators ein. Es besteht hierbei die Möglichkeit, den Zugriff auf ein iSCSI-Target durch einen Benutzernamen und ein Kennwort abzusichern.
- nbd
  Ähnlich wie iSCSI ermöglicht Network Block Device (nbd) den Zugriff auf Blockgeräte über ein Netzwerk. nbd bezeichnet hierbei sowohl das zum Datenaustausch verwendete offene Netzwerkprotokoll (s. [NBD]) als auch die Serveranwendung, mit der Blockgeräte zur Verfügung gestellt werden. Das Protokoll unterstützt weder Verschlüsselung noch Integritätsschutz noch Authentifizierung.[22] Auch ergreift QEMU keine zusätzlichen Schutzmechanismen, wie z. B. die Nutzung von IPSec.
- ssh
  Wenngleich die Secure Shell (ssh) als entfernte Eingabeaufforderung konzipiert ist, so ermöglicht sie doch auch einen Datentransfer. QEMU setzt ssh in diesem Zusammenhang ein, um auf entfernte Dateien zuzugreifen und diese als Blockgeräte einzubinden. Die Kommunikation wird hierbei standardmäßig durch starke Kryptografie und Signierung abgesichert. Die konkret verwendeten Verfahren sind konfigurierbar. Ebenso ist die von ssh serverseitig verwendete Authentifizierungsmethode flexibel wählbar. QEMU weist sich gegenüber dem Server mithilfe der Anwendung ssh-agent aus, die auf dem Host betrieben werden muss.

22 nbd sieht zwar einen Schutz durch TLS vor, dieser wird jedoch als experimentell bezeichnet und nicht für den produktiven Betrieb empfohlen.

- Sheepdog
  Sheepdog ist ein modernes Protokoll zur Anbindung von Blockspeicher über ein Netzwerk. Anders als GlusterFS oder CephFS braucht es keine zentrale Datenbank zur Bereitstellung von Metadaten. Die Übertragung von Daten ist allerdings weder gegen Einsichtnahme noch gegen Veränderung geschützt.
- GlusterFS
  GlusterFS ist ein verteiltes Dateisystem, das von QEMU mittels TCP genutzt werden kann.[23] Der Fokus liegt hierbei auf einer hohen Skalierbarkeit und Ausfallsicherheit. Die Übertragung von Daten ist weder gegen Einsicht noch Veränderung geschützt. Eine detailliertere Betrachtung der Anbindung von GlusterFS an QEMU wird in Abschnitt 9.4 durchgeführt.
- CephFS
  CephFS ist ebenso wie GlusterFS ein verteiltes Dateisystem, das den Fokus auf hohe Skalierbarkeit und Ausfallsicherheit legt. CephFS bietet die Möglichkeit einer beidseitigen integritätsgeschützten Authentifizierung zwischen Server und Client. Ebenso kann die Übertragung der Nutzdaten vor Veränderungen geschützt werden. Eine Verschlüsselung wird hingegen nicht unterstützt. Eine detailliertere Betrachtung der Anbindung von CephFS an QEMU wird in Abschnitt 9.3 durchgeführt.
- HTTP(S), FTP(S) und TFTP
  Darüber hinaus unterstützt QEMU die klassischen Webprotokolle HTTP und FTP. Beide können auch in ihrer abgesicherten Form (HTTPS bzw. FTPS) genutzt werden. Auch TFTP wird unterstützt. Die Protokolle können ausschließlich genutzt werden, um lesenden Zugriff auf Datenträger zu erhalten. Eine Änderung der entsprechenden Dateien ist nicht möglich, womit die Anwendungsfälle stark limitiert werden. Die typische Verwendung ist der Zugriff auf Datenträger zur Installation von Software.

Die möglichen Kommunikationsverfahren unterscheiden sich also nicht nur bezüglich ihrer Funktionen, sondern auch bezüglich ihrer sicherheitstechnischen Eigenschaften. Insbesondere beim Einsatz von nicht oder nur geringfügig abgesicherten Protokollen sind daher zusätzliche Schutzmaßnahmen erforderlich. Diese können je nach Anforderung von einem physischen Zugangsschutz des Netzwerkes bis zum Einsatz zusätzlicher Protokolle, wie z. B. IPSec, reichen. Eine detaillierte Betrachtung der Sicherheit von QEMU in Zusammenhang mit GlusterFS und CephFS findet sich in Abschnitt 9.4 bzw. 9.3.

Ein weiterer wesentlicher Faktor bei der Absicherung von Massenspeichern ist das verwendete Dateiformat. Dieses kann weitgehend unabhängig von dem verwendeten Kommunikationsverfahren gewählt werden. Die einfachste strukturierte Art, die Daten auf dem Massenspeicher zu hinterlegen, besteht darin, Block für Block geordnet hintereinander auf das Gerät zu schreiben. Dieses Verfahren wird raw genannt und bedarf keiner Metainformationen. Es hat jedoch den Nachteil, dass typischerweise genau so viel Speicher vorgehalten werden muss, wie der virtuelle Datenträger beinhalten kann. Dies gilt auch dann, wenn der Datenträger nur zu einem geringen Teil gefüllt ist. Ferner unterstützt das Format raw weder Verschlüsselung noch eine Integritätssicherung.

Alternativ kann das Format QEMU Copy on Write 2 (qcow2) eingesetzt werden. Dieses ermöglicht ein dynamisches Wachstum des benötigten Speicherplatzes in Abhängigkeit vom tatsächlich genutzten Speicherplatz. Somit wird regelmäßig wesentlich weniger Speicher benötigt, als dem Gast zugestanden wird. Auch bietet qcow2 die Möglichkeit, Snapshots zu erstellen, und unterstützt Verschlüsselung. Hier ist jedoch anzumerken, dass die Sicherheit des genutzten Verschlüsselungsverfahrens mehrfach angezweifelt wurde (s. z. B. [BER]). Darüber hinaus besitzt qcow2 keinerlei Integritätsschutz. Selbst wenn die Verschlüsselung als ausreichend sicher angesehen werden würde, so ist es dem Angreifer prinzipiell weiterhin möglich, den Inhalt des Blockgeräts zu modifizieren (s. auch Abschnitt 9.2.2.2.3).

Eine Weiterentwicklung des Formats qcow2 ist QEMU Enhanced Disk (qed). Es bietet ähnliche Funktionen wie qcow2, ermöglicht jedoch in vielen Anwendungsszenarien eine bessere Performance. qed unterstützt jedoch weder Verschlüsselung noch Signierung.

23 Weitere Übertragungswege sind UNIX-Domain-Sockets und RDMA. Diese werden jedoch aufgrund fehlender praktischer Relevanz nicht betrachtet.

Darüber hinaus unterstützt QEMU eine Reihe weiterer Dateiformate. Diese werden jedoch typischerweise nicht im produktiven Betrieb eingesetzt und bleiben im Rahmen dieser Studie unberücksichtigt.

3.5.3.4 Vhost

Neben der Emulation oder Paravirtualisierung durch QEMU existiert eine weitere Möglichkeit der Gerätebereitstellung. Diese findet Anwendung, sofern hohe Datendurchsätze bewältigt werden müssen. Ein Nachteil der beschriebenen Gerätebereitstellung durch QEMU sind die Mehrkosten durch die Kontextwechsel von Kernel zu Userspace und umgekehrt. Diese entstehen nicht nur durch den unmittelbaren Zugriff des Gastes auf virtuelle Geräte, sondern auch bei dem resultierenden Zugriff des jeweiligen Emulator-Backends auf die reale Hardware mittels entsprechender Kernel-Schnittstellen. Ebenso ist regelmäßig ein mehrfaches Kopieren der Daten notwendig, die verarbeitet werden sollen. Der hierdurch entstehende Arbeitsaufwand kann die Performance des jeweiligen Geräts bei hohem Datendurchsatz signifikant reduzieren.

Um dies zu verhindern, kann QEMU die direkte Verarbeitung der Gastkommunikation durch die Kernel-Komponente Vhost veranlassen, sofern das jeweilige Gerät durch Vhost unterstützt wird. Der Datenaustausch zwischen Gast und virtuellem Gerät wird dann im Wesentlichen durch Vhost im Linux-Kernel realisiert. Der entsprechende Emulator innerhalb von QEMU wird allenfalls noch zur Verarbeitung performance-unkritischer Funktionen des virtuellen Geräts genutzt.

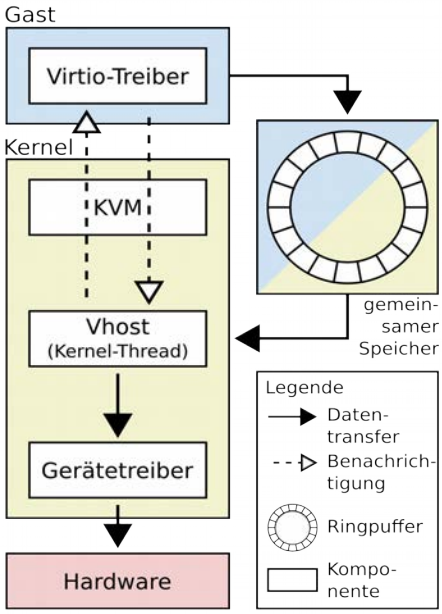

Abbildung 19: Architektur von Vhost

Die technische Realisierung beruht hierbei auf Virtio und erfordert somit die Nutzung der Virtio-Treiber durch den Gast. Im Unterschied zu dem Ansatz, der in Abschnitt 3.5.3.2 beschrieben wurde, ist jedoch nicht QEMU Kommunikationsteilnehmer, sondern ein Kernel-Thread, der zu Vhost gehört. Dieser vermittelt zwischen dem Gast und dem jeweiligen Gerätetreiber. Der Vorteil dieses Verfahrens besteht in der Vermeidung von Kontextwechseln zwischen dem Kernel und QEMU sowie von dem Kopieren der zu verarbeitenden Daten. Vhost unterstützt hierbei die folgenden Geräteklassen:

- Blockgeräte
- SCSI-Geräte
- Netzwerkkarten

Das Verfahren wird in Abb. 19 dargestellt.

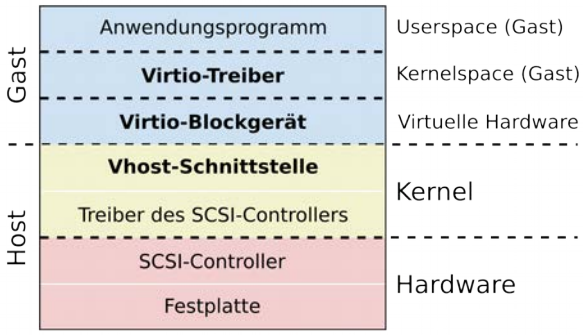

Abbildung 20: Nutzung von Vhost

Die Nutzung von Vhost ist die am weitesten optimierte Variante zur Bereitstellung virtueller Geräte bei gleichzeitiger Abstraktion der tatsächlichen physischen Hardware. Wie auch bei der Paravirtualisierung und der Emulation durch QEMU kann der Gast nicht unmittelbar auf die Hardware zugreifen. Eine Unterstützung des Gastes für die reale Hardware ist daher nicht erforderlich. Abb. 20 stellt ein Beispiel für die Bereitstellung eines virtuellen SCSI-Geräts mittels Vhost dar.

Unter sicherheitstechnischen Gesichtspunkten ist insbesondere festzuhalten, dass – im Gegensatz zur Emulation oder Paravirtualisierung durch QEMU – alle Operationen im Kernel durchgeführt werden. Dies bedeutet, dass die Anzahl der Operationen, die mit uneingeschränkten Rechten auf dem Host ausgeführt werden, deutlich erhöht ist.[24]

3.5.4 Netzwerkanbindung

3.5.4.1 Überblick

Neben der Anbindung an Blockgeräte, die in Abschnitt 3.5.3.3 beschrieben wurde, ist ein wesentlicher Aspekt der Servervirtualisierung die Anbindung des Gastes an ein Netzwerk. Wenngleich QEMU zu diesem Zweck eine Reihe von verschiedenen Backends anbietet, so ist für die Servervirtualisierung im produktiven Umfeld nur die Anbindung an ein TAP-Gerät von Bedeutung. Dazu gehören neben den klassischen TAP-Geräten auch macvtap-Geräte. Das Tap-Gerät stellt das Gegenstück zur virtuellen Netzwerkkarte des Gastes dar. Es ermöglicht den Datenaustausch zwischen Host und Gast auf Layer 2 des OSI-Modells.

Die Aufgabe des Hosts ist es, die Datenpakete, die vom Gast erzeugt werden, an das Netzwerk weiterzuleiten und Pakete, die über das Netzwerk empfangen werden, an den jeweiligen Gast weiterzureichen. Darüber hinaus ist es die Aufgabe des Hosts, den Netzwerkverkehr der virtuellen Maschinen voneinander zu separieren. Insbesondere dürfen Gästen nur den für sie bestimmten Netzwerkverkehr einsehen. Die Einsicht oder Manipulation von Verkehr anderer virtueller oder realer Maschinen soll unterbunden werden. Ebenso soll der Host verhindern, dass ein Gast eine Überlastsituation provoziert, die zu einem Verlust der Verfügbarkeit anderer Netzwerkteilnehmer führt.

Zur Realisierung der Netzwerkanbindung stehen die folgenden vier praxisrelevanten Ansätze zur Verfügung:

- macvtap
- Linux-Bridge

24 Ein Beispiel für einen sicherheitskritischen Fehler in der Vhost-Komponente, der einen Absturz des Hosts durch den Gast ermöglicht, ist [CVE-2013-0311].

- Open vSwitch
- Hardwaredurchreichung (sowohl mit als auch ohne SR-IOV)

Deren wesentlichen sicherheitstechnischen Eigenschaften sind in Tabelle 1 zusammengefasst.

Anbindungsart	Beschränkung interner Kom. (Gast zu Gast)	Beschränkung externer Kom. (Gast zu ext. System)	Netfilter anwendbar	Netzwerkkarte geteilt nutzbar	Live-Migration möglich
Bridge ohne VLAN-Filter	keine	keine	ja	ja	ja
Bridge mit VLAN-Filter	VLAN	VLAN	ja	ja	ja
macvtap (bridge)	keine	keine	nein	ja	ja
macvtap (vepa)	ext. Switch	keine	nein	ja	ja
macvtap (private)	ext. Router	keine	nein	ja	ja
macvtap (passthrough)	keine	keine	nein	nein	ja
Open vSwitch	ext. Control-Plane	ext. Control-Plane	nein	ja	ja
Durchreichung ohne SR-IOV	keine[25]	keine[25]	nein	nein	nein
Durchreichung mit SR-IOV	keine[25]	keine[25]	nein	ja	nein

Tabelle 1: Übersicht über die Methoden zur Netzwerkanbindung

Ein wesentliches Kriterium der jeweiligen Netzwerkanbindung ist, wie gut sich der Netzwerkverkehr administrativ beschränken lässt. Zu diesem Zweck bieten einige der Methoden integrierte Funktionen. Neben diesen integrierten Funktionen ist es möglich, mit der Komponente netfilter des Linux-Kernels den Netzwerkverkehr auf einer Bridge umfassend zu filtern. Beispielsweise ermöglicht es netfilter, die Kommunikation zwischen Gästen flexibel zu beschränken. Aus organisatorischen Gründen ist diese Möglichkeit in komplexen Umgebungen jedoch nur praxisrelevant, sofern eine Unterstützung des Managementwerkzeugs gegeben ist. Grund ist, dass bei manueller Anpassung des entsprechenden Regelwerks der administrative Arbeitsaufwand üblicherweise erheblich höher ist. Dies ist insbesondere dann der Fall, wenn eine Live-Migration der virtuellen Maschinen ermöglicht werden soll. Die durch die Managementwerkzeuge ermöglichten Beschränkungen der Netzwerkkommunikation werden in Kapitel 8 betrachtet.

3.5.4.2 Linux-Bridge

25 Es existieren Netzwerkkarten, die eine Filterung durch die Hardware erlauben. In einem solchen Fall sind die konkreten Filtermöglichkeiten hardwareabhängig.

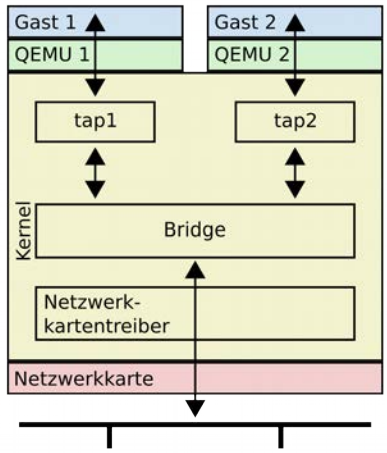

Abbildung 21: Linux-Bridge

Das klassische Mittel zur Verbindung mehrerer Netzwerkschnittstellen auf einem Linux-System ist die Nutzung einer Bridge. Diese verhält sich wie ein üblicher Layer-2-Switch. Standardmäßig werden daher alle Pakete anhand ihrer Ziel-MAC-Adresse uneingeschränkt weitergeleitet. In Abb. 21 wird die Anbindung zweier Gäste an ein externes Netzwerk mittels einer Bridge dargestellt.

Die Linux-Bridge unterstützt das Filtern anhand von VLAN-IDs. Hierbei kann einem angeschlossenen Gerät, typischerweise einem TAP-Gerät, eine VLAN-ID zugeordnet werden. Aus dem jeweiligen Gerät ausgeleitete Pakete werden automatisch mit einem entsprechenden VLAN-Header versehen. Ferner werden nur Pakete mit entsprechender Markierung dem Gerät zugeführt. Beides geschieht vollständig transparent für den Gast: Zum einen entfernt der Host die Markierungen eingehender Pakete vor der Übergabe, und zum anderen führt der Host eine entsprechende Markierung ausgehender Pakete zwangsweise durch.

3.5.4.3 Open vSwitch

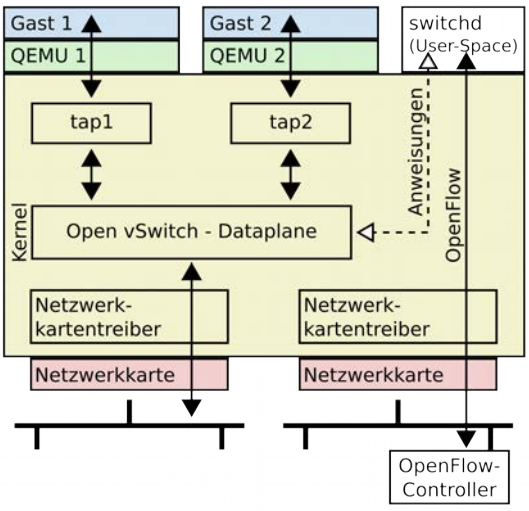

Abbildung 22: Open vSwitch

Eine weitere Anbindungsmöglichkeit ist die Anbindung mittels Open vSwitch. Die grundlegende Idee hinter Open vSwitch ist es, die Logik, die bestimmt, ob ein Paket weitervermittelt wird und wenn ja, wohin dieses zu versenden ist, von der eigentlichen Paketvermittlung weitestgehend unabhängig zu

implementieren. Dies ermöglicht die Realisierung einfach strukturierter, aber hochperformanter Hardware zur Paketvermittlung. Die Steuerung erfolgt hierbei durch externe Controller, die auf Standardhardware ausgeführt werden können. Im Weiteren wird die vermittelnde Komponente Dataplane und die steuernde Komponente Controlplane genannt. Eine Controlplane kann mehrere Dataplanes kontrollieren und ermöglicht die zentrale Verwaltung eines Regelwerks. Somit wird im Gegensatz zu klassischen Netzwerkinfrastrukturen ein zentrales Regelmanagment möglich. Im Rahmen der Virtualisierung ist daher die Realisierung einer Dataplane nicht nur mittels Hardware, sondern als Software sinnvoll.

Open vSwitch ist eine solche mittels Software realisierte Dataplane und Teil des Linux-Kernels. Diese Dataplane wird durch die Anwendung switchd gesteuert, die im Userspace des Hosts ausgeführt wird. switchd kommuniziert zum einen mit der Dataplane innerhalb des Kernels und zum anderen mit einem externen Controller. Letzterer stellt die Regeln zur Paketweiterleitung zur Verfügung. Abb. 22 zeigt eine Anbindung zweier virtueller Maschinen an ein externes Netzwerk mittels Open vSwitch. Das abgebildete Szenario trennt die Kommunikation der Gäste und den Datenverkehr zum OpenFlow-Controller über separate Netzwerke.

Anzumerken ist ferner, dass eine Filterung von Paketen durch netfilter nicht unterstützt wird. Die Beschränkung der Netzwerkkommunikation obliegt daher primär der Controlplane.

3.5.4.4 macvtap

Der Gerätetreiber macvtap ermöglicht es, virtuelle Netzwerkkarten auf Layer 2 des OSI-Referenzmodells unmittelbar an eine Netzwerkkarte zu binden. Es existiert somit keine weitere vermittelnde Netzwerkkomponente, wie z. B. eine Linux-Bridge. Die Weiterleitungsregeln für den Datenverkehr sind zum einen abhängig von dessen Ziel und Quelle, zum anderen von dem Modus, in dem sich das macvtap-Gerät befindet. Ziel und Quelle werden hierbei anhand der jeweiligen MAC-Adressen des Pakets identifiziert. Konkret werden Daten nach folgendem Regelwerk weitergeleitet:

- Extern zu Gast
  Externer Netzwerkverkehr, d. h. Pakete, die weder von den Gästen des Hosts noch von ihm selbst erzeugt wurden, wird wie von einem üblichen Switch weitergeleitet. Ein von extern eintreffendes Paket wird daher anhand seiner Ziel-MAC-Adresse zum jeweiligen Gast gesendet. Somit ist es den Gästen möglich, Daten von externen Geräten uneingeschränkt zu empfangen.
- Gast zu Extern
  Analog werden Pakete, die von Gästen erzeugt werden und deren Ziel ein externes Gerät ist, an das entsprechende Gerät versendet, das durch die Ziel-MAC-Adresse identifiziert wird. Somit ist es den Gästen möglich, Daten uneingeschränkt an externe Geräte zu versenden.
- Gast zu Gast
  - Modus bridge
    Pakete werden zwischen angeschlossenen Gästen auf Basis der jeweiligen Ziel-MAC-Adresse des Datenpakets verteilt. Die Paketvermittlung erfolgt also wie bei einem üblichen Switch. Somit können die Gäste uneingeschränkt untereinander kommunizieren. Eine Nutzung des realen Netzwerks erfolgt hierbei nicht.
  - Modus vepa
    Pakete werden ausnahmslos über die externe Schnittstelle versendet. Ein Paket von einem Gast zu einem anderen Gast wird somit zu dem externen Switch versendet. Sendet dieser das Paket zurück an den Host, wird es an den jeweiligen Gast weitergeleitet. Die Paketfilterung wird daher an den externen Switch abgetreten. Es ist jedoch zu beachten, dass eine große Anzahl von Switchen diesen Modus nicht unterstützen, auch wenn sie eine Filterfunktion besitzen.[26]

26 Hierbei ist problematisch, dass der physische Port des Switches, über den ein Paket eintrifft, ebenfalls der Port ist, über den das Paket wieder ausgeleitet werden muss. Diese Anforderung tritt in klassischen geswitchten Umgebungen nicht auf. Die entsprechende Fähigkeit des Switches wird üblicherweise als Hairpin-Mode bezeichnet.

- Modus private
  Der Modus unterbindet die unmittelbare Kommunikation auf Layer 2 zwischen Gästen. Ein Paket, das von einem Gast an einen anderen Gast gesendet wird, wird verworfen. Zu beachten ist jedoch, dass Ziel und Quelle des Pakets im Sinne einer geswitchten Umgebung anhand ihrer MAC-Adressen identifiziert werden. Eine mittelbare Kommunikation, z. B. über einen Router, wird daher nicht unterbunden.[27]
- Modus passthrough
  Dieser Modus dient dazu, eine reale Netzwerkschnittstelle exklusiv einem Gast zur Verfügung zu stellen. Dies ermöglicht es dem Gast, die Netzwerkkarte weitreichend zu konfigurieren. Da der Modus keine geteilte Nutzung von Netzwerkkarten vorsieht, ist er im Rahmen der Servervirtualisierung eher unüblich. Ferner ermöglicht er dem Gast Konfigurationsanpassungen, wie z. B. die Änderung der MAC-Adresse, die üblicherweise

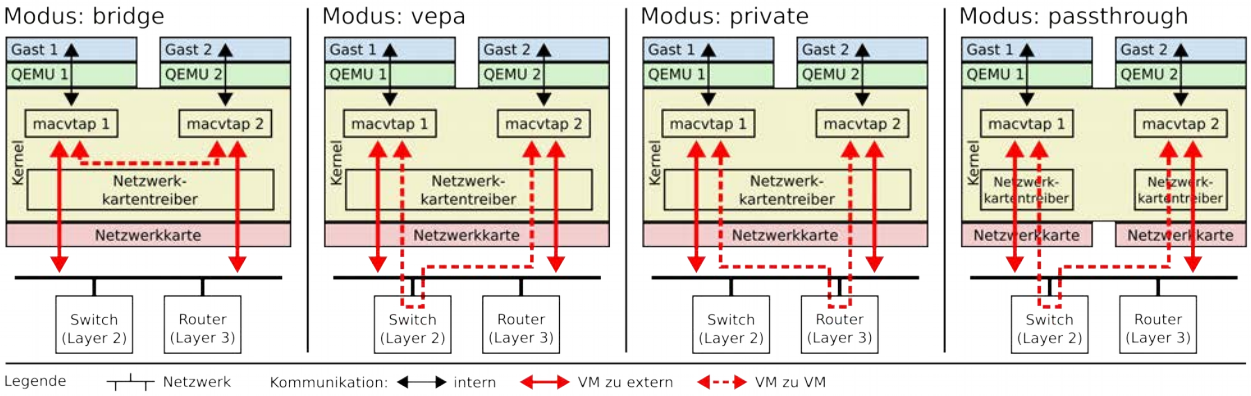

Abbildung 23: Modi von macvtap

unerwünscht sind.

Die Regeln zur Paketvermittlung sind in Abb. 23 dargestellt. Anzumerken ist ferner, dass eine Filterung von Paketen durch netfilter nicht unterstützt wird.

3.6 libvirt

3.6.1 Allgemeines

Auch wenn QEMU zusammen mit KVM es ermöglicht, virtuelle Maschinen zu betreiben, so reicht diese Lösung noch nicht für komplexe Virtualisierungsumgebungen aus – unter anderem deswegen, weil eine netzwerkfähige und praxistaugliche Möglichkeit zur Administration fehlt. Diesbezüglich hat sich libvirt als De-facto-Standard etabliert. libvirt ermöglicht es, mehrere QEMU-Prozesse und somit mehrere virtuelle Maschinen auf einem Host zu verwalten. Der Prozess, der dieses realisiert, ist libvirtd. Er wird dauerhaft auf dem jeweiligen Host ausgeführt und ermöglicht die netzwerkbasierte Verwaltung der virtuellen Maschinen. Er benötigt hierzu typischerweise Root-Rechte.

27 Der Name des Modus kann irreführend sein. Sind zwei Gäste in unterschiedlichen IP-Netzen und haben sie Zugang zu einem Router, der Zugriff auf beide Netze hat, so bestimmt der Router, ob eine Kommunikation zwischen den Gästen möglich ist. Es darf somit nicht davon ausgegangen werden, dass im Modus *private* eine Kommunikation zwischen Gästen grundsätzlich ausgeschlossen ist.

Es ergibt sich eine Vertrauensgrenze zwischen dem Netzwerk und libvirtd. Im Rahmen des zugrunde liegenden Bedrohungsszenarios (s. Abschnitt 2.1) wird angenommen, dass die Kommunikation über das Netzwerk prinzipiell nicht vertrauenswürdig ist. Daher ist insbesondere die Identität der Gegenstelle und deren Autorisierung zu überprüfen. Ferner müssen übergebene Daten verifiziert werden, besonders dann, wenn sie vor der Authentisierung der Gegenstelle entgegengenommen werden.

libvirt dient als Abstraktionsschicht von QEMU, die sowohl von virt-manager (s. Abschnitt 3.6.5.2) als auch von virsh (s. Abschnitt 3.6.5.1) als auch von OpenStack (s. Abschnitt 10.2) genutzt wird. Durch die Nutzung von libvirt ist es für die Managementwerkzeuge weitestgehend unerheblich, welcher Hypervisor (QEMU, XEN, LXC, VirtualBox, HyperV etc.) eingesetzt wird.

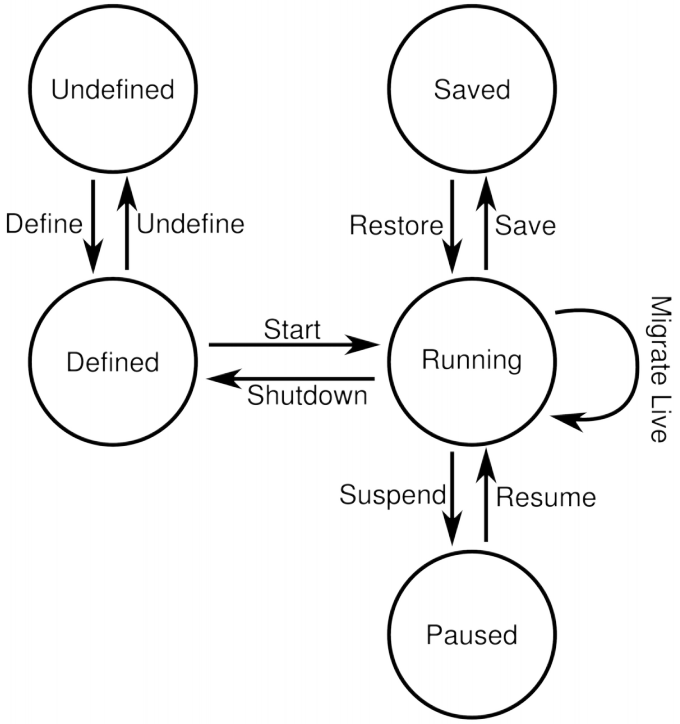

Abbildung 24: Lifecycle der VM

libvirt ermöglicht es insbesondere, den Zustand virtueller Maschinen zu verwalten. Eine von libvirt gesteuerte virtuelle Maschine befindet sich stets in einem der vier Zustände Defined, Running, Paused oder Saved. Im Zustand Defined ist der Gast libvirt bekannt, wird jedoch nicht ausgeführt. Wird er gestartet, wechselt er in den Zustand Running. Ferner ermöglicht es libvirt, den Gast zu pausieren. Dies kann zum einen dadurch geschehen, dass die virtuellen CPUs des Gastes angehalten werden. Hierdurch wechselt der Gast in den Zustand Paused. Er belegt hierdurch weiterhin alle von ihm benötigten Ressourcen, verbraucht jedoch keine Rechenzeit mehr. Zum anderen kann der komplette Zustand, also sowohl der Inhalt des Arbeitsspeichers als auch der Status der virtuellen Geräte, persistent gespeichert werden. Dies entspricht einem Wechsel in den Zustand Saved, in dem der Gast nicht ausgeführt wird und somit keine Ressourcen außer dem notwendigen persistenten Speicher benötigt. Darüber hinaus ermöglicht es libvirt, eine virtuelle Maschine restlos zu löschen und damit alle verwendeten Ressourcen freizugeben. In Abb. 24 wird dieser Sachverhalt in Form eines Zustandsdiagramms dargestellt.

libvirt unterstützt ferner die Migration eines Gastes von einem Host auf einen anderen. Die Migration erfolgt über einen Tunnel, der von dem libvirt-Daemon des Quellhosts aufgebaut wird. Es existieren zwei Arten der Migration: online und offline. Für beide Migrationsarten ist es erforderlich, dass beide Hosts auf die Datenträger des Gastes zugreifen können. Des Weiteren müssen auf dem Zielhost genügend Ressourcen zur Verfügung stehen, um den Gast betreiben zu können. Da der Gast seine Netzwerkkonfiguration bei der

Migration nicht ändert, muss der Zielhost auch auf die Netzwerke zugreifen können, an die der Gast angebunden ist.

Die Offline-Migration zeichnet aus, dass der Gast während der Migration heruntergefahren ist. Im Rahmen der Offline-Migration wird hauptsächlich die Konfiguration der virtuellen Maschine von einem Host auf einene anderen Host übertragen.

Bei Durchführung einer Online-Migration (oder Live-Migration) werden zusätzlich der Arbeitsspeicher und der Status der Geräte des Gastes während dessen Ausführung auf den Zielhost kopiert. Speicherseiten, die während des Kopiervorgangs verändert wurden, werden markiert und in weiteren Kopierdurchläufen übertragen. Schließlich wird der Gast auf dem Quellhost angehalten und auf dem Zielhost gestartet. Sollte es während des Verfahrens zu einem Fehler kommen, so wird der Vorgang abgebrochen und der Gast auf dem Quellhost unverändert weiterbetrieben. Das Verfahren ist somit weitgehend fehlertolerant. Die entsprechenden Funktionen werden von QEMU zur Verfügung gestellt (s. Abschnitt 3.5.1). Die Absicherung der Datenübermittlung obliegt hingegen libvirt.

Über libvirt können neben virtuellen Maschinen auch virtuelle Netze und der Zugang zu Massenspeichern verwaltet werden. Eine vollständige Dokumentation stellt [LIBVIRT] bereit.

Darüber hinaus können mittels libvirt folgende Funktionen verwaltet werden:

- Domänen (virtuelle Maschinen) inklusive CPU-Pinning
- Snapshots
- Hosts (Auslesen der Funktionalitäten, NUMA-Cells etc.)
- Geräte auf dem Host
- geheime Schlüssel (für Festplattenverschlüsselung, die Anbindung von iSCSI, Ceph etc.)
- Netzwerkkarten auf dem Host und in dem Gast
- virtuelle Netzwerke
- Firewallregeln auf dem Host
- Speicherpools und -volumes
- Ereignisse (ACPI-Events in virtuellen Maschinen etc.)
- Datenstromübertragung (für die Weiterleitung der seriellen Konsolen, VNC etc.)

Die Umsetzung dieser Funktionen geschieht mittels QEMU. Hierbei startet libvirt für jede virtuelle Maschine einen separaten QEMU-Prozess. Die Rechte, die einem solchen Prozess bei Erstellung einer virtuellen Maschine eingeräumt werden, können wie folgt gewählt werden:

- Übernahme der Rechte des ausführenden Benutzers
- Nutzung der Rechte eines gewöhnlichen Benutzers, der über die Konfiguration von libvirt ausgewählt werden kann
- Nutzung des Benutzers root

Der erste Ansatz ist nur möglich, sofern der Benutzer lokal am Host angemeldet ist. Dies ist für eine einfache lokale Virtualisierung auf dem Desktop ohne Root-Rechte geeignet und bietet den Benutzern des Systems eine voneinander unabhängige Verwaltung ihrer virtuellen Maschinen. Diese Variante hat im Bereich der Servervirtualisierung jedoch keine Relevanz.

Im zweiten Fall wird libvirt mit den Privilegien eines vorgegebenen gewöhnlichen Benutzers gestartet. Hierdurch erhält der QEMU-Prozess auf dem Host nur eingeschränkte Privilegien. Bei der Nutzung eines unprivilegierten Benutzers muss der Gast auf sämtliche Dateien auf dem Hostsystem zugreifen dürfen, die er benötigt. Da alle durch libvirt verwalteten QEMU-Instanzen hierbei mit den gleichen Rechten gestartet werden, können daher alle Prozesse grundsätzlich auf die gleichen Ressourcen zugreifen.

Beim dritten Ansatz wird der QEMU-Prozess mit Root-Rechten ausgeführt. Mithilfe der Bibliothek libcap-ng ist es jedoch trotzdem möglich, dessen Privilegien einzuschränken. Hierzu gibt der Prozess nach dem Starten und eventuell notwendigen Initialisierungstätigkeiten, aber vor der Ausführung des Gastes seine Berechtigungen ab.

3.6.2 Kommunikation

Folgende Transportmechanismen können verwendet werden, um mit libvirt von einem entfernten Rechner aus virtuelle Maschinen zu verwalten:[28]

- tcp
  Der Modus tcp nutzt standardmäßig eine unverschlüsselte TCP-Verbindung. Aus sicherheitstechnischen Gründen ist von der Verwendung über ein unsicheres Netzwerk abzuraten. Eine Ausnahme hiervon kann bestehen, wenn die Verbindung über ein zusätzliches Plug-in, z. B. SASL, abgesichert wird. Standardmäßig wird der Port 16509 verwendet.
- tls
  Der Transportmechanismus tls nutzt das Protokoll Transport Layer Security (TLS), um eine auf TCP basierende Netzwerkkommunikation abzusichern, d. h. gegen Einsicht und Veränderung durch Dritte zu schützen. Grundlage der Sicherung ist ein X509-Zertifikat, mit dem der Server sich gegenüber dem Client ausweist. Ferner besteht die Möglichkeit, dass auch der Server einen Identitätsnachweis vom Client in Form eines X509-Zertifikats verlangt. Standardmäßig wird der Port 16514 verwendet.
- ssh
  Bei Verwendung des Transportmechanismus ssh wird die Kommunikation über das Protokoll Secure Shell (ssh) übertragen. Die Verschlüsselung, Integritätssicherung und die gegenseitige Authentifizierung obliegt in diesem Fall ssh. Hierzu können asymmetrische Schlüssel genutzt werden. Aufgrund der umfangreichen Konfigurationsmöglichkeiten des Authentifizierungsvorgangs von ssh kann jedoch auch eine Vielzahl weiterer Verfahren eingesetzt werden. Standardmäßig wird der Port 22 genutzt.
- unix
  Der Transportmechanismus unix nutzt einen Unix-Socket zur Datenübertragung. Da ein Unix-Socket ausschließlich die Kommunikation zwischen Programmen auf demselben Host unterstützt, ist ein Informationsaustausch über ein Netzwerk damit ausgeschlossen. Eine Zugriffsbeschränkung wird zum einen umgesetzt, indem zwei Sockets erstellt werden. Der eine ermöglicht einen Vollzugriff, der andere lediglich einen lesenden Zugriff auf die Funktionen von libvirt. Ferner wird der Zugriff über die Dateirechte am jeweiligen Socket eingeschränkt. Standardmäßig werden für die Sockets die Dateien /var/run/libvirt/libvirt-sock bzw. /var/run/libvirt/libvirt-sock-ro verwendet.
- ext
  Der Transportmodus ext ermöglicht es, ein externes Programm zur Kommunikation einzusetzen. Die Daten werden von libvirt an dieses Programm übergeben oder von diesem in Empfang genommen. Die Datenübertragung obliegt dem externen Programm. Entsprechend hängt der Grad der Absicherung der Kommunikation ebenso von diesem ab.
- libssh2
  Der Transportmodus libssh2 entspricht weitestgehend dem Modus ssh. Im Gegensatz zum Modus ssh wird jedoch nicht der SSH-Server des Betriebssystems, sondern ein in libvirt integrierter SSH-Server genutzt.

libvirt bietet ferner die Möglichkeit, selbst eine Benutzerauthentifizierung durchzuführen. Praxisrelevant ist dies nur bei Nutzung eines Transportmechanismus, der eine entsprechende Überprüfung selbst nicht oder nicht in ausreichend sicherem Maße ermöglicht. Namentlich sind dies unix und tcp. Bei unix kommt regelmäßig polkit (früher Policykit) zum Einsatz. Es erlaubt eine flexible Konfiguration der Authentifizierung an lokalen virtuellen Maschinen. Da polkit keine Netzwerkunterstützung bietet, ist es im Rahmen der Authentifizierung für die Servervirtualisierung nicht relevant.[29] Der Transportmodus tcp lässt sich mit Simple Authentification and Security Layer (SASL) absichern. Dieses Framework ermöglicht sowohl eine Authentifizierung als auch eine verschlüsselte und integritätsgesicherte Datenübertragung.

28 Der Fokus der Studie bezüglich der möglichen Transportmechanismen liegt hierbei auf Nutzung von tls und ssh.

29 polkit kann aber sehr wohl auch bei einem netzwerkbasierten Zugriff zur Autorisierung eingesetzt werden.

Typischerweise weist sich der Benutzer mit einem Benutzernamen und einem Kennwort aus. Ebenso werden aber auch Single-Sign-on-Lösungen mittels Kerberos unterstützt.

Außer der Überprüfung der Benutzeridentität bietet libvirt die Möglichkeit, den Autorisierungsvorgang für die Benutzer zu konfigurieren. libvirt spezifiziert hierzu eine Softwareschnittstelle, mit der entsprechende Frameworks eingebunden werden können. Das einzige Framework, das von der zugrunde liegenden Version von libvirt unterstützt wird, ist polkit. Dies ermöglicht eine feingranulare Konfiguration der von libvirt zur Verfügung gestellten Funktionen.

Dies ist sicherheitsrelevant, da der Angreifer bei Kompromittierung des QEMU-Prozesses eines Gastes, z. B. durch einen Fehler in einem emulierten Treiber, prinzipiell auf alle anderen QEMU-Prozesse zugreifen kann. Es ist ihm grundsätzlich auch möglich, auf Ressourcen anderer Gäste, wie z. B. auf deren Festplattenabbilder, zuzugreifen.[30]

Zur Reduzierung der verbundenen Risiken stehen drei Lösungen zur Verfügung:
- SELinux mittels sVirt
- AppArmor mittels sVirt
- cgroups

Diese Schutzmaßnahmen beschränken den Zugriff der QEMU-Prozesse auf Geräte und Dateien, die für ihren Betrieb erforderlich sind. Dies dient dazu dem Gast den Zugriff auf Ressourcen anderer Gäste zu versagen (s. Abschnitt 7).

3.6.3 Integrierte Schutzmaßnahmen

3.6.3.1 sVirt

libvirt verfügt mit sVirt über einen Security-Treiber. Dieser unterstützt SELinux und AppArmor. In beiden Fällen ist keinerlei Konfiguration durch den Administrator erforderlich. Durch sVirt werden sämtliche Einstellungen automatisch und ohne Einwirkung von außen durchgeführt. AppArmor oder SELinux müssen hierzu auf dem System aktiv sein.

Das Ziel beider Verfahren ist, dass ein Angreifer auch nach Ausbruch aus einer virtuellen Maschine nicht auf andere virtuelle Maschinen oder den Host zugreifen kann. Insbesondere soll der Zugriff auf sämtliche auf dem Host befindliche Dateien, die nicht von dem Gast für die Ausführung benötigt werden, verhindert werden.

Bei Nutzung von SELinux wird jede virtuelle Maschine in ihrer eigenen SELinux-Domäne gestartet. Dabei werden die Kategorien der Domäne für jede virtuelle Maschine individuell und einzigartig gewählt. Die Kategorien können zur Laufzeit gewählt werden und erfordern keine zeitaufwendige Anpassung der SELinux-Richtlinie. Die in der Richtlinie hinterlegten Regeln erlauben der Domäne nur den Zugriff auf die Ressourcen mit identischem Label. Damit soll sichergestellt werden, dass eine virtuelle Maschine nicht auf die Ressourcen anderer virtueller Maschinen zugreifen kann. Gemeinsam genutzte Ressourcen und Read-Only-Ressourcen enthalten entsprechende Label. Die Zuweisung der Domäne kann automatisch durch libvirt oder manuell durch den Administrator in der Konfiguration der virtuellen Maschine erfolgen.

Bei Nutzung von AppArmor erzeugt der sVirt-Treiber vor dem Start der virtuellen Maschine automatisch ein AppArmor-Profil. AppArmor gestattet es einem Prozess, nur auf die in seinem Profil aufgeführten Ressourcen zuzugreifen. Alle weiteren Zugriffe werden unterbunden. Der libvirt-Daemon trägt die Ressourcen, die von der virtuellen Maschine benötigt werden, automatisch in das Profil ein.

30 [CVE-2011-1751] beschreibt einen Angriff, der einen Ausbruch aus einer virtuellen Maschine ermöglicht und damit auch prinzipiell den Zugriff auf Ressourcen anderer Gäste gestattet.

3.6.3.2 cgroups

Das cgroups-Framework erlaubt die Ressourcenverwaltung unter Linux. Hierdurch können einzelnen Prozessen Zugriffsrechte auf Geräte eingeräumt oder entzogen werden. Ferner können Leistungsmerkmale von Ressourcen garantiert oder beschränkt werden. QEMU nutzt cgroups für die folgenden Zwecke:

- Garantie und Beschränkung der CPU-Leistung einer virtuellen Maschine (cpu)
- Zuweisung einer virtuellen Maschine zu einer bestimmten CPU/Numa-Zelle (CPU-Pinning) (cpuset)
- Garantie und Beschränkung der Bandbreiten und Scheduler-Werte bei dem Zugriff auf Blockgeräte (blkio)
- Garantie und Beschränkung der Bandbreiten bei dem Zugriff auf das Netzwerk (net_cls)
- White- und Blacklisting von Geräten (devices)

Hierfür sind die cgroups-Controller verantwortlich. Deren Namen sind in Klammern hinter den Funktionen angegeben. libvirt unterstützt die Verwendung sämtlicher Controller mit Ausnahme von net_cls ohne zusätzlichen Konfigurationsaufwand.

3.6.4 Netzwerkanbindung

3.6.4.1 Virtuelle Netzwerke

Mithilfe von virtuellen Netzwerken ermöglicht libvirt sowohl die Vernetzung virtueller Maschinen untereinander als auch die Anbindung an reale Netzwerke. Ein solches virtuelles Netzwerk umfasst typischerweise einen IPv4- oder IPv6-Adressbereich. Ebenso ist ein Mischbetrieb von IPv4 und IPv6 möglich. Dem virtuellen Netzwerk ist ferner einer der Modi isolated, routed oder nat zugeordnet. Der Modus bestimmt hierbei, ob und wie virtuelle Maschinen mit externen Systemen kommunizieren können.

Das Ziel ist es hierbei, die Beschränkungen der in Abschnitt 3.5.4 beschriebenen Methoden zur Netzwerkanbindung zu umgehen. Letztere dienen lediglich dazu, einzelne Hosts mit realen Netzwerken auf Layer 2 zu verbinden. Sie stellen jedoch keine Infrastrukturdienste bereit, wie z. B. einen DHCP-Server zur automatischen Vergabe von IP-Adressen. Entsprechende Dienste müssen gegebenenfalls unabhängig vom Host bereitgestellt werden. Die durch libvirt realisierbaren virtuellen Netzwerke erlauben im Gegensatz hierzu die Verwaltung und automatische Vergabe von IP-Adressen. Dies erlaubt nicht nur die Verwaltung der Gäste an sich, sondern auch die Verwaltung von deren Netzwerkumgebung.

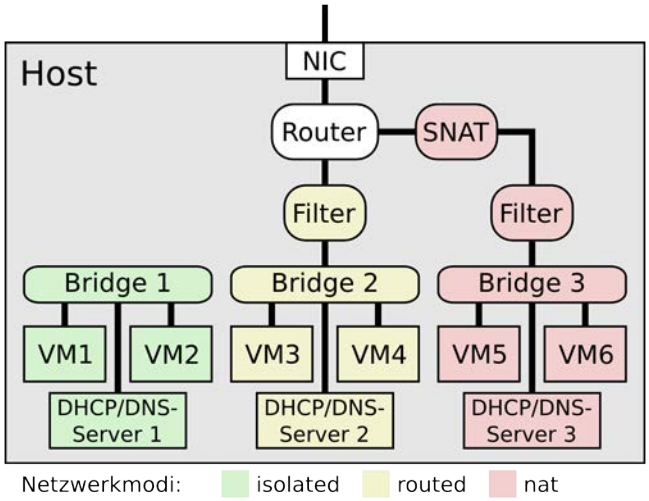

Abbildung 25: libvirt-Netzwerke

Die technische Umsetzung erfolgt über je eine Linux-Bridge pro Netzwerk. An die Linux-Bridge werden alle virtuellen Maschinen des Netzwerks angebunden. Es besteht somit grundsätzlich eine geswitchte Verbindung zwischen den virtuellen Maschinen eines Netzwerks. Der zu einer virtuellen Maschine eingehende bzw. von ihr ausgehende Netzwerkverkehr wird durch den Host gefiltert. Welche Filterregeln konkret angewendet werden, richtet sich nach dem Modus des Netzwerks. Ferner besitzt der Host für jedes Netzwerk eine eigene IP-Adresse und kann als Router zu anderen internen oder externen Netzwerken fungieren. Ob und wie eine solche Paketweiterleitung erfolgt, hängt ebenfalls vom Modus des jeweiligen Netzes ab.

Optional kann ein DHCP- und DNS-Server in das jeweilige Netzwerk eingebunden werden. Hierzu wird für jedes Netzwerk, für das ein entsprechender Dienst zur Verfügung gestellt werden soll, eine separate Instanz der Anwendung dnsmasq auf dem Host gestartet. Diese stellt den Gästen des Netzwerks DHCP- und DNS-Informationen gemäß der Netzwerkkonfiguration zur Verfügung. Abb. 25 stellt einen Host mit drei virtuellen Netzwerken unterschiedlicher Modi mit jeweils zwei virtuellen Maschinen dar.

Ein Netzwerk im Modus isolated ermöglicht lediglich die Kommunikation zwischen Gästen innerhalb desselben virtuellen Netzwerks des Hosts. Eine Kommunikation mit externen Systemen oder Gästen in anderen virtuellen Netzwerken wird unterbunden.

Im Gegensatz dazu ermöglicht ein Netzwerk im Modus routed eine Kommunikation sowohl mit anderen Gästen auf dem gleichen Host als auch mit externen Systemen. Hierzu übernimmt der Host die Rolle eines Routers. Pakete, die in das virtuelle Netzwerk hineingeleitet oder aus ihm hinausgeleitet werden, d. h. ihren Ursprung bzw. ihr Ziel in realen oder anderen virtuellen Netzwerken haben, werden nach den folgenden Regeln gefiltert:

1. Wird ein Paket in das Netzwerk hineingeleitet und hat das Paket eine IPv4- oder IPv6-Zieladresse, die dem Netzwerk zugeordnet ist, so wird es weitergeleitet.

2. Wird ein Paket aus dem Netzwerk hinausgeleitet und hat das Paket eine IPv4- oder IPv6-Quelladresse, die dem Netzwerk zugeordnet ist, so wird es weitergeleitet.
3. Ansonsten wird das Paket verworfen.

Dieses Regelwerk soll sicherstellen, dass die von den Gästen verwendeten IP-Adressen denen entsprechen, die dem Netzwerk zugeordnet sind.[31] Die Paketvermittlung erfolgt, sofern sie dem Regelwerk genügt, sowohl zwischen einem internen und einem externenNetzwerk als auch zwischen zwei internen Netzwerken.[32]

Ein Netzwerk im Modus nat entspricht weitgehend dem Netzwerk im Modus routed. Zusätzlich wird jedoch die IPv4-Adresse von Paketen, die aus dem Netzwerk ausgeleitet werden, maskiert, sodass aus Sicht des Empfängers der Host die Quelle der Kommunikation ist. Ein Verbindungsaufbau zu einem Gast des Netzes aus einem anderen Netz ist in diesem Modus nicht ohne Weiteres möglich.

Für alle drei Netzwerkmodi gilt, dass der Host netzwerkinterne Kommunikation nicht filtert. Die einzige Ausnahme hiervon bildet die optionale Überprüfung der MAC-Adressen. Hierbei werden Pakete, die vom Gast versendet werden, dahingehend überprüft, ob in den Paketen die MAC-Adresse des Gastes als Absenderadresse eingetragen ist.[33]

Darüber hinaus kann eine einzelne virtuelle Maschine mehr als eine virtuelle Netzwerkkarte besitzen. Eine solche virtuelle Maschine kann daher an mehrere virtuelle Netzwerke angebunden sein und die Rolle eines Routers oder Switches einnehmen. Dies ermöglicht die Erstellung komplexer virtueller Netzwerkumgebungen, die aus mehreren miteinander verbundenen virtuellen Netzwerken bestehen.[34]

Eine über das virtuelle Netzwerk hinausgehende Separation der virtuellen Maschinen findet nicht statt. Insbesondere wird der Routing-Vorgang vom Host für alle Kommunikationsvorgänge identisch angewandt. Allen Gästen, die grundsätzlich mit externen Systemen kommunizieren können, sich also nicht in einem Netz mit dem Modus isolated befinden, stehen damit die gleichen Kommunikationsmöglichkeiten zur Verfügung. Die Verteilung eines virtuellen Netzwerks über mehrere Hosts wird von libvirt nicht unterstützt.

3.6.4.2 Alternative Netzwerkanbindung

Neben der Möglichkeit, die Gäste mittels virtueller Netzwerke zu vernetzen, ist es ebenso möglich, die virtuellen Netzwerkschnittstellen der Gäste direkt an die Netzwerkinfrastruktur zu binden. Für die Servervirtualisierung sind insbesondere die Anbindung mithilfe von macvtap, Open vSwitch und die Hardwaredurchreichung der Netzwerkkarte von Relevanz.

Der erstgenannte Ansatz veranlasst libvirt, ein macvtap-Interface mit Bindung an eine bereits bestehende Netzwerkschnittstelle zu erzeugen. Dabei kann frei bestimmt werden, welchen macvtap-Modus die künftige Schnittstelle anwenden soll. (vgl. Abschnitt 3.5.4.4).

Ferner ist libvirt imstande, virtuelle Maschinen an einen Open vSwitch anzubinden. Die Konfiguration des Open vSwitches wird von libvirt hingegen nicht unterstützt. Es ist also ein von libvirt unabhängiger OpenFlow-Controller erforderlich.

31 Netzinterner Verkehr, d. h. Verkehr zwischen Gästen innerhalb desselben virtuellen Netzwerks, wird jedoch grundsätzlich weitergeleitet. Den Gästen steht es somit frei, für netzinterne Kommunikation frei gewählte IP-Adressen zu nutzen. Auch wird nicht die Zuordnung einer konkreten IP-Adresse zu einem Gast überprüft, sondern lediglich geprüft, ob die jeweilige Adresse zu dem jeweiligen Netzwerk gehört. Es steht dem Gast somit frei, mit externen Geräten mittels einer beliebigen IP-Adresse zu kommunizieren, sofern diese Adresse dem jeweiligen Adressbereich des Netzes angehört. IP-Spoofing-Angriffe werden somit nur bedingt verhindert.

32 Hierbei ist jedoch zu beachten, dass der Host im Modus *routed* und *nat* als Router ebenfalls ein Netzwerkteilnehmer ist. Werden auf dem Host Netzwerkdienste ausgeführt, so können diese typischerweise aus den Gastnetzen heraus aufgerufen werden.

33 Die Überprüfung von MAC-Adressen ist im Rahmen einer Standardinstallation oftmals deaktiviert.

34 In einer solchen Umgebung kann die Paketvermittlung zwischen einem virtuellen und einem realen Netzwerk daher nicht nur unmittelbar über den Host erfolgen, sondern auch mittelbar über weitere virtuelle Netzwerke.

Darüber hinaus ermöglicht libvirt die Durchreichung von Netzwerkhardware an eine virtuelle Maschine. Die Konfiguration der entsprechenden Hardware muss jedoch unabhängig von libvirt vorgenommen werden.

Allen drei alternativen Anbindungsmethoden ist gemeinsam, dass eine Filterung des Netzwerkverkehrs durch libvirt nicht automatisch veranlasst wird.

3.6.4.3 Filterung von Netzwerkverkehr

Sofern ein Gast über ein virtuelles Netzwerk oder eine Linux-Bridge an das Netzwerk angebunden ist, ermöglicht libvirt eine flexible Filterung seines Datenverkehrs. Hierbei ist es unbedeutend, ob die Anbindung mittels Linux-Bridge explizit gewählt wurde oder ob sie sich implizit durch die Nutzung eines virtuellen Netzwerks (s. Abschnitt 3.6.4.1) ergibt. Die gegebenenfalls durch den Modus des virtuellen Netzwerks bedingten Beschränkungen der Netzwerkkommunikation (s. Abschnitt 3.6.4.1) bleiben hiervon unberührt. Ferner können Filterregeln für jede virtuelle Netzwerkkarte individuell spezifiziert werden. Somit wird eine feingranulare und umfassende Beschränkung der Netzwerkkommunikation der Gäste untereinander wie auch zu realen Maschinen ermöglicht. Die Filterung geschieht hierbei durch den Host. Eine Abänderung oder Umgehung der jeweiligen Regeln durch den Gast wird somit verhindert. Auch ein kompromittierter Gast ist daher an die Filterregeln gebunden. Der Host agiert damit als Firewall.

Ein Regelwerk zur Filterung kann dabei aus mehreren Regeln bestehen, die nacheinander abgearbeitet werden müssen. Diese Regeln können die Weiterleitung eines Netzwerkpakets von einem oder zu einem Gast anhand einer Reihe von Paketeigenschaften erlauben oder verbieten. Die Filterung erlaubt somit eine feingranulare Beschränkung der Netzwerkkommunikation. Die Kommunikation kann somit auf das Mindestmaß beschränkt werden, das zur Diensterbringung der Gäste erforderlich ist.

Filterregeln können dabei unter Zuhilfenahme von Variablen spezifiziert werden. Insbesondere können IP- und MAC-Adressen sowie TCP- und UDP-Portnummern als Variablen in die Regel eingebunden werden. Dies ermöglicht es, Filtermechanismen unabhängig von Gästen zu spezifizieren. Solche abstrakten Filterregeln können durch eine entsprechende Belegung der jeweiligen Variablen an einen konkreten Gast angepasst werden. Auf diesem Wege ist auch eine Wiederverwendung der Filterregeln für mehrere Gäste möglich.

Mittels Filterregeln können die Eigenschaften der folgenden Protokolle untersucht werden:

- Ethernet inklusive VLAN (802.1Q)
- STP
- ARP
- RARP
- IPv4
- IPv6
- TCP
- UDP
- UDPLIGHT
- SCTP
- ICMP
- ICMPv6
- IGMP
- AH
- ESP

Darüber hinaus ist der Filtermechanismus imstande, die IPv4-Adresse des jeweiligen Gastes dynamisch zu ermitteln. Dies geschieht entweder über die Auswertung der IPv4-Absenderadresse des allgemeinen IPv4-Verkehrs oder über die Analyse von DHCP-Paketen. Hat der Filtermechanismus die IPv4-Adresse des Gastes erlernt, so steht diese Information den Filterregeln zur Verfügung.

libvirt beinhaltet ferner die folgenden vordefinierten Regeln:

- no-arp-spoofing: Alle ARP-Pakete, die eine falsche IP-MAC-Adressenzuordnung aufweisen, werden verworfen.
- no-ip-spoofing: Alle versandten IP-Pakete, die nicht die IP-Absenderadresse des Gastes aufweisen, werden verworfen.
- no-ip-multicast: Alle vom Gast versandten IP-Multicast-Pakete werden verworfen.
- clean-traffic: Umfasst die Regeln no-arp-spoofing und no-ip-spoofing.
- allow-dhcp: Erlaubt das Anfragen einer DHCP-Adresse.
- allow-dhcp-server: Erlaubt das Anfragen einer DHCP-Adresse von einem anzugebenden DHCP-Server.

3.6.5 Verwaltungswerkzeuge

3.6.5.1 virsh

Die Verwaltung eines Hosts, auf dem libvirtd ausgeführt wird, kann mit dem befehlszeilenorientierten Werkzeug virsh erfolgen. Es ist wie libvirtd ein Teil des libvirt-Projekts. Aus diesem Grund unterstützt virsh den vollen Funktionsumfang von libvirtd. Auch wenn es weniger intuitiv zu bedienen ist als beispielsweise der grafische virt-manager, so eignet es sich besonders zur Automatisierung von Verwaltungsaufgaben mithilfe von Skripten.

Zusammen mit virsh werden typischerweise auch die folgenden befehlszeilenorientierten Anwendungen genutzt:

- virt-install: Erstellen von virtuellen Maschinen
- virt-clone: Klonen von virtuellen Maschinen
- virt-convert: Konvertieren von virtuellen Maschinen anderer Virtualisierungsprodukte, wie z. B. VMware

virsh unterstützt die Authentifizierung via polkit und SASL und kann Verbindungen über alle von libvirt unterstützten Transportmechanismen (s. Abschnitt 3.6.2) aufbauen. Dies ermöglicht die Verwaltung von Hosts über ein Netzwerk. Ferner können Gäste mittels virsh von einem Host auf einen anderen migriert werden. Von der Migration abgesehen, ist virsh stets mit einem einzigen Host verbunden. Die gleichzeitige koordinierte Verwaltung mehrerer Hosts ist somit nicht möglich. virsh besitzt keine Benutzer- und Rollenverwaltung.

3.6.5.2 virt-manager

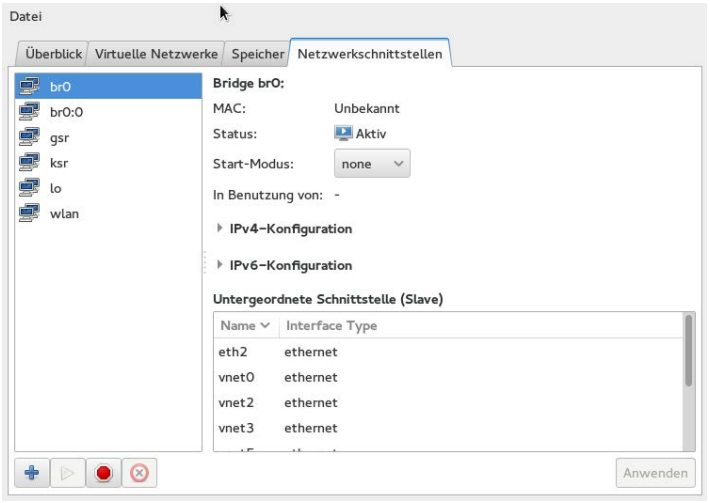

Abbildung 26: Verwaltungsfenster für Netzwerkkarten im virt-manager

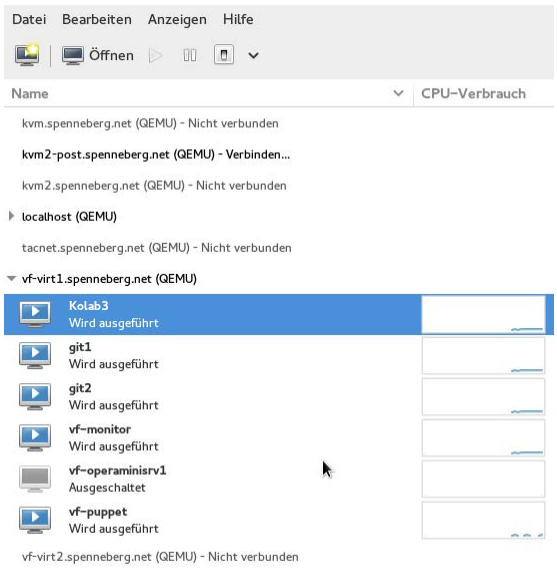

Abbildung 27: Übersichtsfenster des virt-managers

Ein weiteres leichtgewichtiges Managementwerkzeug ist die grafische Anwendung virt-manager. Sie ermöglicht die Administration einzelner Hosts und ihrer Geräte über ein Netzwerk anhand einer intuitiv bedienbaren grafischen Oberfläche. virt-manager ermöglicht sowohl das Erstellen, das Starten und Stoppen als auch eine umfassende Konfiguration virtueller Maschinen. Ferner sind in den virt-manager ein VNC- sowie ein Spice-Client integriert, um auf die Konsole der Gäste zugreifen zu können. Um virt-manager einsetzen zu können, muss auf dem jeweiligen Host der Dienst libvirtd ausgeführt werden und über ein Netzwerk erreichbar sein. virt-manager unterstützt hierzu alle in Abschnitt 3.6.2 aufgeführten Kommunikationsarten. Darüber hinaus werden keine weiteren Dienste oder weitere Infrastrukturen benötigt. Die hieraus resultierende einfache Installation macht ihn insbesondere für kleine und mittelgroße Virtualisierungsumgebungen interessant, die nur aus wenigen Hosts bestehen. Abb. 27 zeigt einen Screenshot des Übersichtsfensters der Anwendung, Abb. 26 zeigt die Verwaltung virtueller Netzwerkkarten eines Gastes.

4 Testumgebung

Zur Analyse der einzelnen Komponenten werden im Rahmen dieser Studie Testumgebungen verwendet. Dies erlaubt es, die Widerstandsfähigkeit der Virtualisierungsumgebung in einem möglichst praxisnahen Szenario unter den Rahmenbedingungen zu untersuchen, die in Kapitel 2 genannt werden.

Hierzu werden zwei separate Testumgebungen aufgebaut. Die erste Testumgebung dient zur Analyse der Virtualisierung mittels KVM, QEMU und libvirt. Hierzu gehören neben der eigentlichen Virtualisierung auch die Anbindung an einen Blockspeicher sowie die Anbindung an ein Gast-Netzwerk. Auch kann diese Testumgebung genutzt werden, um grundlegende Managementfunktionen, die von libvirt bereitgestellt werden, sowie die Interaktion von virsh und virt-manager mit dem Host zu untersuchen.

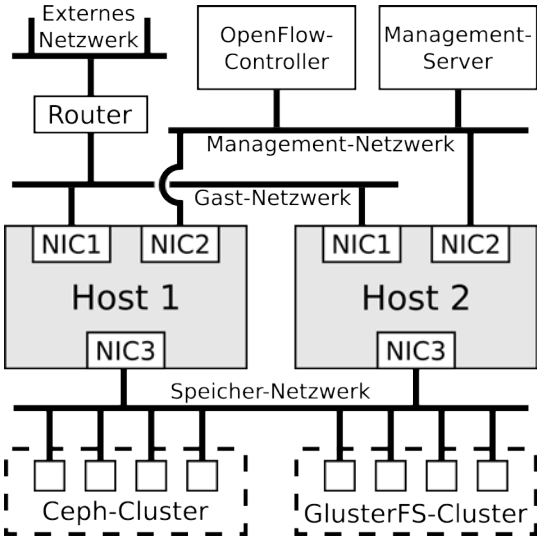

Abbildung 28: Infrastruktur der Testumgebung

Die Infrastruktur dieser Testumgebung wird in Abb. 28 dargestellt. Sie besteht aus zwei Hosts, die zur Virtualisierung mittels KVM, QEMU und libvirt genutzt werden. Deren Konfiguration kann, wie in den folgenden Abschnitten beschrieben, angepasst werden. Allen Konfigurationen ist gemeinsam, dass die Hosts über unterschiedliche Netzwerkkarten an drei Netzwerke angebunden sind.

Das Gast-Netzwerk dient zur Kommunikation der Gäste untereinander. Ob und in welcher Form eine Separation der Gastkommunikation stattfindet, hängt von der gewählten Konfiguration ab. Ferner wird den Gästen die Kommunikation mit externen Netzwerken, wie z. B. dem Internet, über einen Router ermöglicht. Der Router filtert den Netzwerkverkehr nicht.

Das Management-Netzwerk dient dazu, Managementfunktionen zu nutzen, die von libvirt bereitgestellt werden. Dies umfasst beispielsweise die Erstellung, die Aktivierung und das Entfernen virtueller Maschinen, ebenso wie den Zugriff auf eine grafische Konsole mittels VNC oder Spice. Um dies umzusetzen, existiert ein Management-Server, auf dem virsh sowie virt-manager installiert ist. Ferner existiert ein OpenFlow-Controller, der die Regeln für die Netzwerkkommunikation vorhält. Sofern der Host mittels Open vSwitch an das Gast-Netzwerk angebunden ist, erfragt er entsprechende Regeln beim OpenFlow-Controller. Andernfalls hat der OpenFlow-Controller keine Funktion.[35] Das Management-Netzwerk ist ein geswitchtes Netzwerk ohne Filterung. Insbesondere sind keine Zugriffsbeschränkungen bezüglich der

35 Aus sicherheitstechnischen Gründen wird die Kommunikation mit dem OpenFlow-Controller oftmals über ein separates Netzwerk geführt. Hierdurch kann ausgeschlossen werden, dass die durch den Gast mittelbar beeinflussbare Open-Flow-Kommunikation negativen Einfluss auf die Management-Kommunikation hat. Zur Vereinfachung der Testumgebung wird jedoch auf ein weiteres Netzwerk verzichtet.

Netzwerkteilnehmer implementiert. Ein Zugriff auf das Management-Netzwerk aus dem Gast heraus ist nicht vorgesehen und soll unterbunden werden.

Das Speicher-Netzwerk dient dazu, dem Gast Blockspeicher zur Verfügung zu stellen. Zu diesem Zweck ist sowohl ein CephFS-Cluster als auch ein GlusterFS-Cluster Bestandteil der Testumgebung. Der konkret verwendete Cluster wird durch die Konfiguration der Hosts bestimmt. Die Anbindung erfolgt nicht unmittelbar aus dem Gast heraus, sondern mittels QEMU. Das Speicher-Netzwerk ist ebenfalls ein geswitchtes Netzwerk ohne Filterung. Zugriffsbeschränkungen bezüglich der Netzwerkteilnehmer sind nicht implementiert. Ein unmittelbarer Zugriff auf das Speicher-Netzwerk aus dem Gast heraus ist nicht vorgesehen und soll unterbunden werden.

Die Anbindung der Gäste an das Netzwerk kann mit einem virtuellen Netzwerk erfolgen, das durch libvirt verwaltet wird. Innerhalb der Testumgebungen können Netzwerke aller drei Modi isolated, routed und nat erzeugt werden (vgl. Abschnitt 3.6.4.1). Der technische Aufbau ist in Abb. 29 dargestellt.

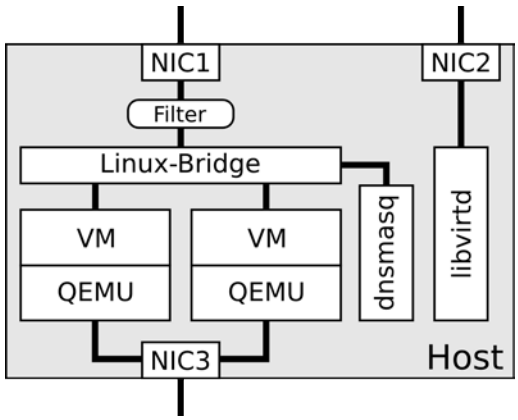

Abbildung 29: Anbindung über ein virtuelles Netzwerk

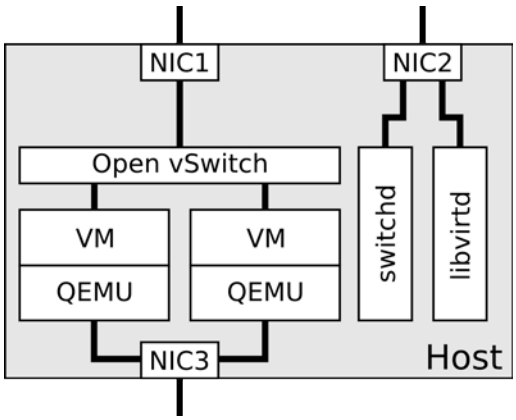

Abbildung 30: Anbindung mittels Open vSwitch

Darüber hinaus lässt sich die Testumgebung derart konfigurieren, dass die Anbindung der Gäste an das Netzwerk über Open vSwitch durchgeführt wird. Die Netzwerkverkehrsregeln werden hierbei mit einem externen OpenFlow-Controller verwaltet (vgl. Abschnitt 3.5.4.3). Der Fokus liegt hierbei auf der Analyse der grundlegenden Eigenschaften des Open vSwitches sowie auf der Interaktion zwischen Gast und Open vSwitch. Der OpenFlow-Controller wird lediglich dann betrachtet, wenn dies zur Analyse der Funktionsweise des Open vSwitches erforderlich ist. Des Weiteren werden den Gästen in dieser Konfiguration keine DHCP- oder DNS-Informationen zur Verfügung gestellt. Die Konfiguration ist in Abb. 30 dargestellt.

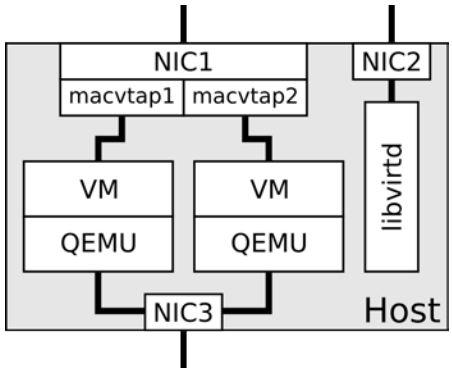

Abbildung 31: Anbindung mittels macvtap

Eine weitere von der Testumgebung unterstützte Anbindungsart (Abb. 31) ist die Verwendung von macvtap. Hierbei werden alle vier macvtap-Modi unterstützt. Eine Filterung sowie eine Separation der Datenströme über die Eigenschaften des jeweiligen macvtap-Modus hinaus erfolgen nicht. Der externe Switch unterstützt die Ausleitung von Paketen an demselben Netzwerkport, an dem diese empfangen werden (Hairpin-Mode).

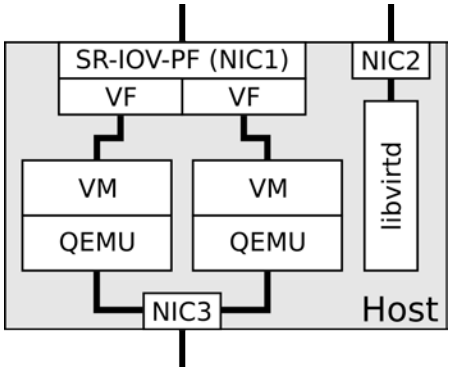

Abbildung 32: Anbindung mittels SR-IOV

Als weitere Anbindungsart verfügt einer der Hosts über eine SR-IOV-fähige Netzwerkkarte und eine IOMMU. Die Netzwerkkarte kann somit an Gäste durchgereicht werden. Dies ermöglicht es, die Funktionsweise der Durchreichung zu analysieren. Eine Analyse der verwendeten Netzwerkkarte wird jedoch nicht durchgeführt. Die entsprechende Konfiguration ist in Abb. 32 dargestellt.

Die Hosts verwenden hierbei folgende Software:

- CentOS 7[36] (x86_64, Kernel „Linux ovs_master.localdomain 3.10.0 #1 SMP Tue Feb 23 14:36:10 CET 2016“)
- QEMU 1.5.3 (105.el7_2.3)
- libvirt 1.2.17
- Open vSwitch 2.5.0

Ferner werden auf dem Management-System virsh 1.2.17 und virt-manager 1.2.1 eingesetzt.

36 Bei der Installation von CentOS 7 wird die Security Policy *CentOS Profile for Cloud Providers (CPCP)* aktiviert.

5 Sicherheitsanalyse von QEMU

5.1 Zielsetzung

Wesentliche sicherheitskritische Komponenten der Virtualisierungsumgebung sind der QEMU-Prozess und die Kernel-Komponente KVM, da diese quasi unmittelbar mit dem Gast kommunizieren. Sie sind somit im Kontext des Bedrohungsszenarios aus Abschnitt 2.1 besonders exponiert. Zudem besitzen beide Komponenten, besonders jedoch QEMU, einen großen Codeumfang und bieten damit eine vergleichsweise große Angriffsfläche. Aus diesem Grund wird im Folgenden eine Analyse des Quelltextes beider Komponenten durchgeführt. Diese umfasst primär das Aufzeigen von gegebenenfalls vorhandenen systematischen Mängeln der Implementierung. Sekundär sollen konkrete sicherheitskritische Implementierungsfehler aufgedeckt werden. Maßgeblich für die Analyse sind hierbei die in Abschnitt 2.2 aufgeführten Sicherheitsanforderungen. Insbesondere beschränkt sich die Untersuchung auf die Virtualisierung von Serversystemen, die auf der x86-Architektur basieren.

Eine systematische Untersuchung der Funktionstüchtigkeit der Softwarekomponenten wird nicht durchgeführt. So werden beispielsweise Emulatoren nicht dahingehend überpüft, ob diese das Verhalten der jeweiligen realen Hardware korrekt nachahmen. Darüber hinaus wird angenommen, dass die Schnittstellen zur Verwaltung des Gastes, insbesondere der Zugang zur Konsole des Gastes mittels VNC oder Spice, nur vertrauenswürdigen Personen zur Verfügung steht. Angriffsmöglichkeiten gegen entsprechende Schnittstellen werden daher nicht untersucht. Ferner bleiben von QEMU genutzte Softwarebibliotheken weitgehend unberücksichtigt. Zwar haben sicherheitskritische Fehler dieser Bibliotheken typischerweise identische Auswirkungen, wie entsprechende Fehler in QEMU selbst. Keine der eingesetzten Bibliotheken ist jedoch besonders exponiert oder übernimmt virtualisierungsspezifische Aufgaben. Eine Untersuchung der Bibliotheken kann somit prinzipiell unabhängig von QEMU vorgenommen werden.

Neben dem eigentlichen Quellcode werden ferner Härtungsmaßnahmen analysiert, die beim Übersetzungsprozess und zur Laufzeit von QEMU durch das CentOS-Projekt eingesetzt werden.

5.2 Methodik

Um die Güte des Quellcodes festzustellen und um sicherheitskritische Fehler zu finden, werden mehrere aufeinander aufbauende Arbeitsschritte durchlaufen.

Der erste Arbeitsschritt ist die Einstufung des Quelltextes in relevante und nicht relevante Bereiche. Als nicht relevant werden die Codebereiche eingeordnet, die weder direkt noch indirekt an der Virtualisierung beteiligt sind oder offensichtlich keine sicherheitskritischen Eigenschaften haben. Im Sinne eines Ausschlussverfahrens bleiben hierbei die potenziell relevanten Quelltextanteile übrig. Diese werden, abhängig von ihrer Funktion, zu Gruppen zusammengefasst. Eine Gruppe kann hierbei aus einer oder mehreren Quelltextdateien bestehen, in Einzelfällen auch aus Dateifragmenten. Als nicht relevant eingestufte Quelltextbestandteile werden von der weiteren Untersuchung ausgeschlossen.

Aufbauend auf der Kategorisierung des Quelltextes wird in der nächste Stufe der Quellcode mittels der GNU Compiler Collection (GCC) übersetzt. Die Konfiguration des Compilers wird hierbei unabhängig von der zu überprüfenden Anwendung gewählt. Die Ergebnisse werden nach Art und Schweregrad der Unregelmäßigkeiten bewertet.

Darüber hinaus wird mithilfe des Analysewerkzeugs flawfinder gezielt nach der Nutzung von Funktionen gesucht, deren Verwendung als besonders fehleranfällig gilt und damit ein erhöhtes Sicherheitsrisiko darstellt.

Die erzielten Befunde werden manuell inspiziert, sofern ihr Schweregrad und die sicherheitstechnische Relevanz des betroffenen Quelltextes dies gebieten. Hierzu wird stets eine Abwägung zwischen der

potenziellen sicherheitstechnischen Bedeutung des Befundes und dem zu erwartenden Untersuchungsaufwand vorgenommen.

5.3 Organisatorische Schutzmaßnahmen

Der Quelltext von QEMU wird mit der Software git verwaltet und ist öffentlich abrufbar[37]. Insgesamt verzeichnet das Projekt 84 Maintainer[38], die für den Quelltext oder Teile von ihm verantwortlich sind. Diese haben vermutlich schreibenden Zugriff auf das Quelltextrepositorium. Die Einbringung von Quelltext ist, typisch für Open-Source-Software, prinzipiell jedem möglich. Hierzu gibt das QEMU-Projekt in [QEMUCTR] einen Prozess vor, der zwingend einzuhalten ist. Dieser erfordert das Einreichen von Codeänderungen über die öffentliche Mailingliste des Projekts.[39] Somit ist bereits vor der Übernahme des Codes eine öffentliche Begutachtung möglich. Über die Aufnahme in das QEMU-Projekt entscheidet der jeweilige Maintainer, wobei er zumindest überprüfen muss, ob die durch den Prozess vorgeschriebenen Anforderungen an den Quellcode eingehalten werden. Um dies zu vereinfachen und die Praxistauglichkeit des Prozesses sicherzustellen, beschreibt [QEMUCTR] einige formale Anforderungen bezüglich der Einreichung von Quellcode.

Darüber hinaus gibt ein Coding Styleguide ([STYLE]) verbindliche Anforderungen an die syntaktische Beschaffenheit des Quelltextes vor. Er spezifiziert beispielsweise, wie Quelltext einzurücken ist und wie Variablennamen zu wählen sind, um ein einheitliches Erscheinungsbild und die Les- und Wartbarkeit des Quelltextes zu gewährleisten. Positiv ist hier insbesondere zu vermerken, dass das Projekt ein Skript zur Verfügung stellt, das diese Regeln[40] automatisiert überprüft und dessen Anwendung zwingend ist.[41]

Des Weiteren existiert eine verbindliche Coding Guideline ([CODING]). Sie enthält Regeln, die über die syntaktischen Eigenschaften des Codes hinausgehen. Im Fokus steht der Umgang mit besonders risikobehafteten Codekonstruktionen der Programmiersprache C. Dazu gehören Vorschriften, die beschreiben, wann vorzeichenbehaftete und wann vorzeichenlose Datentypen zu nutzen sind, um Typkonflikte möglichst zu vermeiden. Darüber hinaus wird die Nutzung von Funktionen zur Speicherverwaltung, wie z. B. malloc, explizit untersagt. Solche Funktionen werden durch projekteigene Funktionen ersetzt, mit dem Ziel, die Fehleranfälligkeit der dynamischen Speicherverwaltung zu reduzieren. Ferner werden Handlungsanweisungen zum Umgang mit Zeichenketten gegeben. Diese untersagen die Nutzung besonders risikobehafteter Funktionen, wie z. B. strcat und strncat, und stellen projekteigene Alternativen vor. Auch werden Regeln aufgestellt, die den Umgang mit Ausgabefunktionen mit einer variablen Parameteranzahl beschreiben. Diese schreiben die Verwendung bestimmter GCC-spezifischer Attribute vor, um einen Abgleich der Anzahl an Funktionsparametern in Format-Strings während der Programmübersetzung durchzuführen. Dieses Vorgehen verhindert, dass Format-String-Schwachstellen entstehen. Abschließend werden Regeln zur Fehlerbehandlung aufgestellt. Hier wird gefordert, dass die Beendigung von QEMU aufgrund eines Fehlers nur in Ausnahmefällen erfolgen soll. Insbesondere soll eine Programmbeendigung auf diesem Wege nicht durch den Gast provoziert werden können, da dies ansonsten zu einem Angriff auf dessen Verfügbarkeit missbraucht werden könnte.

Die Regeln der Coding Guideline sind grundsätzlich geeignet, um sicherheitskritische Schwachstellen zu vermeiden. Eine automatische Überprüfung dieser Regeln findet jedoch nur teilweise statt. Eine lückenlose Anwendung dieser Regeln kann daher nicht garantiert werden. Tatsächlich wird in Abschnitt 5.7 gezeigt, dass durchaus Verletzungen dieser Regeln existieren. Insbesondere die gemischte Nutzung von vorzeichenlosen und vorzeichenbehafteten Datentypen tritt gehäuft auf.

37 http://git.qemu.org/qemu.git
38 Diese sind in der Datei *MAINTAINER* aufgeführt.
39 Eine Ausnahme hiervon besteht lediglich für Code zur Behebung sicherheitskritischer Fehler, für die ein Responsible-Disclosure-Verfahren vorgesehen ist.
40 Tatsächlich werden durch das Skript wesentlich mehr Regeln abgearbeitet, als im Coding Styleguide aufgeführt sind.
41 Das Erzwingen von syntaktischer Konformität verhindert jedoch weitestgehend, dass man Rückschlüsse auf die Codequalität durch Analyse syntaktischer Codemerkmale ziehen kann.

Es lässt sich somit festhalten, dass ein Prozess existiert, der neben formalen Anforderungen zur Einreichung von Code auch Anforderungen an dessen Struktur und Aufbau stellt. Ein wesentlicher Teil dieser Regeln hat zum Zweck, sicherheitskritische Programmierfehler zu vermeiden. Die Regeln sind zum überwiegenden Teil sowohl zweckdienlich als auch praktikabel. Die Einhaltung wird teilweise automatisiert überprüft.

5.4 Härtungsmaßnahmen

5.4.1 Programmübersetzung

Da die Fehlerfreiheit komplexer Programme in der Praxis nicht sicherzustellen ist, sind – insbesondere bei Verwendung der Programmiersprache C – Maßnahmen angeraten, die die Anwendung härten. Als wirkungsvoll haben sich hier der Einsatz von Stack Protection und des NX-Bits erwiesen. Zur Laufzeit können diese Maßnahmen sinnvoll durch Address Space Layout Randomization (ASLR) ergänzt werden. Damit diese Maßnahmen die Ausnutzung von Implementierungsfehlern effektiv unterbinden können, bedarf es jedoch einer analogen Härtung der von QEMU genutzten Softwarebibliotheken.

Im Folgenden wird daher untersucht, ob sowohl QEMU als auch die genutzten Bibliotheken Stack Protection und das NX-Bit nutzen und für den Einsatz von ASLR vorbereitet sind. Es ist jedoch zu beachten, dass dies keine Quellcodeeigenschaften im engeren Sinne sind. Alternative Linux-Distributionen können QEMU anders als CentOS 7 übersetzen und gegebenenfalls auf die aufgeführten Härtungsmaßnahmen verzichten. Dies gilt analog ebenso für die verwendeten Bibliotheken. Auch kann die Anzahl der verwendeten Bibliotheken oder deren Version sich bei alternativen Distributionen unterscheiden.

Feststellen lässt sich, dass im Rahmen von CentOS 7 sowohl QEMU selbst als auch alle verwendeten Bibliotheken das NX-Bit verwenden. Es wird somit strikt zwischen modifizierbaren Daten und ausführbaren Codeabschnitten unterschieden und eine Ausführung von injiziertem Code unterbunden.[42] Darüber hinaus nutzen sowohl QEMU selbst als auch die verwendeten Bibliotheken Stack Protection. Ausnahmen bilden jedoch insgesamt 15 Bibliotheken, die entsprechend in Anhang C markiert sind. Keine der Funktionen dieser Bibliotheken ist jedoch unmittelbar aus dem Gast aufrufbar. Somit ist festzuhalten, dass einer Änderung des Programmflusses durch Manipulationen des Stacks entgegengewirkt wird. Auch wenn Stack Protection solche Manipulationen nicht unter allen Umständen erkennen und verhindern kann und es Schutzlücken durch die genannten Bibliotheken gibt, so kann dennoch von einem weitgehend effektiven Schutz vor entsprechenden Angriffen ausgegangen werden.

Ferner ist der QEMU-Quellcode positionsunabhängig übersetzt worden. Dies ermöglicht es dem Betriebssystem, die Positionierung des Programms und seiner Bibliotheken im Rahmen von Address Space Layout Randomization (ASLR) zufällig und für den Angreifer unvorhersehbar zu wählen. Sofern es dem Angreifer trotz der oben beschriebenen Härtungsmaßnahmen möglich wäre, den Programmfluss zu seinem Vorteil zu verändern, so erschwert ASLR die zweckdienliche Ausnutzung erheblich (vgl. [SSP]).[43]

Als positiv ist daher zu bewerten, dass eine Reihe von Härtungsmaßnahmen bereits bei der Übersetzung von QEMU angewandt werden. Insbesondere in Kombination wehren diese Maßnahmen eine große Anzahl von Angriffsarten ab oder erschweren diese zumindest erheblich. Dies führt dazu, dass ein sicherheitskritischer Fehler in der Implementierung von QEMU nicht zwingend ausgenutzt werden kann, auch wenn der Fehler an sich hierzu geeignet ist.

42 Dies setzt voraus, dass die verwendete Architektur eine solche Funktion unterstützt. Bei modernen x86-Systemen ist dies gegeben, auf anderen Architekturen ist dies aber eventuell nicht der Fall.

43 Zu beachten ist, dass ASLR durch den Linux-Kernel zur Laufzeit der Anwendung realisiert wird. Um effektiv zu sein, muss ASLR zum Startzeitpunkt von QEMU aktiviert sein. CentOS 7 aktiviert ASLR automatisch im Rahmen des Bootvorgangs.

5.4.2 seccomp

Darüber hinaus kann QEMU zur Laufzeit seccomp (s. [SECCOMP]) nutzen. seccomp ermöglicht es einer Anwendung, die Menge der ihr erlaubten Operationen selbst einzuschränken. Hierzu erzeugt die Anwendung im Rahmen ihrer Initialisierung ein Regelwerk und übergibt dieses dem Betriebssystem. Das Regelwerk umfasst hierbei die Menge an Systemaufrufen (Syscalls), die die Anwendung im Rahmen ihres Betriebs durchführen möchte. Sobald das Regelwerk aktiviert worden ist, verbietet das Betriebssystem der Anwendung, über diese Menge hinausgehende Systemaufrufe zu nutzen. Es handelt sich somit um einen Whitelist-Ansatz. Das Betriebssystem erlaubt es nicht, das Regelwerk nach dessen Aktivierung zu modifizieren.

Auch wenn ein solcher Whitelist-Ansatz eine potenziell effektive Möglichkeit ist, ungewolltes Verhalten der Anwendung zu unterbinden, so zeigt sich in der Praxis doch regelmäßig eine Reihe von Problemen. Besonders hervorzuheben ist die sehr grobgranulare Konfiguration durch das Regelwerk. Wird ein Systemaufruf an einer beliebigen Stelle innerhalb der Anwendung benötigt, so muss seine Ausführung für die gesamte Anwendung erlaubt werden. Die Privilegierung einer einzelnen Teilkomponente der Anwendung ist nicht möglich. Ferner besteht das Regelwerk in seiner klassischen Form lediglich aus einer Menge von erlaubten Systemaufrufen. Dies führt beispielsweise dazu, dass einer Anwendung, die auf Dateien zugreift, die Nutzung des Systemaufrufs open ermöglicht werden muss. Somit darf die Anwendung jedoch auf beliebige Dateien zugreifen, sofern dies nicht durch weitere Schutzmaßnahmen eingeschränkt wird, wie z. B. durch die Dateirechte innerhalb des Dateisystems. Neuere Versionen von seccomp bieten daher die Möglichkeit, neben dem jeweiligen Systemaufruf auch Filterregeln für deren Parameter zu spezifizieren. Somit kann die Erlaubnis zur Ausführung eines Systemaufrufs nicht nur pauschal, sondern auch auf Basis der jeweilig übergebenen Parameter erfolgen. Hierdurch ist es möglich, ein – im Vergleich zum ursprünglichen Ansatz – wesentlich feingranulareres Regelwerk zu erstellen, vergleichbar mit den Möglichkeiten von SELinux oder AppArmor (s. Abschnitt 7.5). Dies erhöht den Nutzen von seccomp deutlich, da somit eine deutlich präzisere Beschränkung auf tatsächlich notwendige Operationen ermöglicht wird. Nachteilig an diesem Ansatz ist jedoch der deutlich erhöhte Entwicklungsaufwand zur genauen Bestimmung ebensolcher Operationen und Parameterbelegungen.

QEMU nutzt seccomp, sofern es mit dem Parameter sandbox gestartet wird. In diesem Fall beschränkt QEMU im Rahmen seiner Initialisierung die Menge der erlaubten Systemaufrufe.[44] Das Regelwerk erlaubt hierbei die Nutzung von insgesamt 218 Systemaufrufen. Hierzu gehören potenziell sicherheitskritische Aufrufe, wie z. B. fork, execve, open, read, write, connect, accept und mmap. Ferner findet keine Filterung auf Basis der jeweiligen Aufrufparameter statt.

Auch ist zu beachten, dass libvirt, das den QEMU-Prozess in aller Regel startet (s. Abschnitt 3.6.1), den Parameter sandbox üblicherweise nicht setzt. Sofern keine entsprechende Anpassung der Konfiguration von libvirt erfolgt, kommt seccomp daher typischerweise nicht zur Anwendung.[45] Aus diesem Grund darf von einer gering verbreiteten Nutzung von seccomp ausgegangen werden. In diesem Zusammenhang stellt sich somit auch die Frage, ob das Regelwerk systematisch und kontinuierlich überprüft wird. Ist das Regelwerk nicht vollständig, so führt dies zu einer unbeabsichtigten Restriktion des QEMU-Prozesses. Dies kann zu Funktionsstörungen und somit zu einer Einschränkung der Verfügbarkeit des Gastes führen.[46]

Auch wenn die Unterstützung von seccomp prinzipiell zu begrüßen ist, so stellt sich die konkrete Umsetzung als fragwürdig heraus. Der Grund hierfür ist insbesondere die große Zahl an pauschal erlaubten, potenziell sicherheitskritischen Systemaufrufen. Der tatsächlich erzielte Sicherheitsgewinn ist daher bestenfalls als gering zu beurteilen. In Zusammenhang mit einer potenziellen Einschränkung der Verfügbarkeit ist bei Aktivierung von seccomp zumindest ein grundlegender Funktionstest aller Komponenten des Gastes durchzuführen, z. B. der Netzwerkkarte oder der Konnektivität mittels VNC oder

44 Konkret geschieht dies in der Datei *qemu-seccomp.c*.

45 Viele moderne Linux-Distributionen, wie z. B. CentOS 7, nutzen zur Absicherung von QEMU hingegen SELinux oder AppArmor (s. Abschnitt 7.5).

46 https://bugs.archlinux.org/task/48703 beschreibt eine solche Einschränkung der Verfügbarkeit.

Spice. Aufgrund des ungünstigen Verhältnisses von Nutzen zu Aufwand ist ein Verzicht auf die Nutzung von seccomp regelmäßig gerechtfertigt.

5.5 Coderelevanz

Da QEMU eine wesentliche Virtualisierungskomponente ist, die quasi unmittelbar[47] mit dem Gast interagiert, sind Programmfehler von QEMU potenziell besonders sicherheitskritisch für das Virtualisierungssystem. Insbesondere die Gefahr eines Ausbruchs aus dem Gast mit anschließender Kompromittierung anderer Gäste oder der Managementumgebung sind hier hervorzuheben, auch wenn diese Gefahr durch die in Abschnitt 5.4 beschriebenen Härtungsmaßnahmen stark reduziert wird.

Entsprechend dem in Abschnitt 5.2 beschriebenen Verfahren wird der Quelltext von QEMU in relevante und nicht relevante Bestandteile unterteilt. Die Analyse, die sich daran anschließt, beschränkt sich auf die Untersuchung relevanter Codeabschnitte. Erschwert wird diese Auswahl hierbei durch die fehlende Isolation der Komponenten voneinander zur Laufzeit. Da die gesamte Emulation durch QEMU – abgesehen von den durch KVM bereitgestellten Funktionen – mittels eines einzelnen Betriebssystemprozesses durchgeführt wird, sind die Funktionen von QEMU allesamt gleichberechtigt (s. Abschnitt 3.5.1). Existieren ausnutzbare Schwachstellen im QEMU-Code, so ist es irrelevant, welche Komponente den Fehler aufweist. Somit können auch Komponenten, die keine sicherheitskritischen Funktionen bereitstellen, Schwachstellen aufweisen, die für die Sicherheit des Systems fatal sind. Die Funktion einer Komponente lässt daher nicht auf die sicherheitstechnische Bedeutung von Fehlern schließen und kann daher keine Grundlage für die Beurteilung von deren Relevanz sein.

Hingegen bietet der modulare Aufbau von QEMU (s. Abschnitt 5.6) Kriterien zur Einschätzung der Relevanz von Codeabschnitten. Eine Reihe dieser Module sind auf einer x86-Plattform nicht ausführbar und damit für diese Studie nicht relevant. Darüber hinaus gibt es eine Reihe von Emulatoren, wie z. B. solche für Audiogeräte, die bei Virtualisierung von Serversystemen nicht angewendet werden. Hierbei muss jedoch beachtet werden, dass entsprechende Komponenten in Einzelfällen (z. B. bei unüblichen Anforderungen an den Gast oder bei nachlässiger Konfiguration) doch genutzt werden können.

Ein weiteres Kriterium zur Einschätzung der Relevanz ist, wie sehr eine Komponente exponiert ist, d. h., wie unmittelbar ein Angreifer mit ihr interagieren kann. Besonders exponierte Komponenten sind die Frontends von Emulatoren (s. Abschnitt 3.5.3.1) sowie die Anbindung an KVM. Backends werden hingegen aufgrund ihrer nur mittelbaren Interaktion mit dem Gast als weniger exponiert angesehen und daher nachrangig untersucht. Komponenten, die weder mittelbar noch unmittelbar mit dem Gast kommunizieren, werden als nicht relevant betrachtet. Letzteres trifft insbesondere auf initialisierenden Code zu, der vor der Ausführung des Gastes abgearbeitet wird.

5.6 Strukturierung des Quelltextes

5.6.1 Übersicht

QEMU ist fast vollständig in der Programmiersprache C verfasst. Insgesamt besteht das Projekt aus 1965 C-Quellcode-Dateien und 939 C-Header-Dateien. Die gesamte Anzahl an Zeilen in C-Quellcode-Dateien beträgt 1.018.231, die Anzahl an Zeilen in C-Header-Dateien beläuft sich auf 174.840[48]. Der Quelltext ist über

47 Zwar erfolgt die Interaktion mittels KVM (s. Abschnitt 3.2). KVM nimmt hierbei aber primär eine Vermittlerrolle ein. Daher kann die Kommunikation zwischen QEMU und dem Gast aus sicherheitstechnischen Gesichtspunkten als unmittelbar betrachtet werden.

48 Als Quellcode- und Header-Datei werden hierbei Daten mit der Endung *c* bzw. *h* betrachtet. Die Zeilenanzahl gibt jeweils die tatsächliche Anzahl an Zeilen, ohne Berücksichtigung von deren Inhalt wieder.

279 Verzeichnisse verteilt. Die Verzeichnisstruktur dient hierbei primär zur Organisation des Quelltextes anhand dessen Funktion.

Tabelle 2 listet alle Quellcodeverzeichnisse auf, d. h. Verzeichnisse mit mindestens einer Quellcodedatei. Die Tabelle führt zu jedem Verzeichnis die Anzahl der in ihm gespeicherten Quellcodedateien an. Hierbei werden gegebenenfalls auch die Inhalte von Unterverzeichnissen berücksichtigt. Darüber hinaus wird die Anzahl an Zeilen dieser Dateien aufgeführt. Ferner wird das Verhältnis der Zeilenanzahl zum gesamten Codeumfang prozentual aufgeführt. Zusätzlich wird der Zweck des gespeicherten Quellcodes erläutert und dessen Relevanz für diese Untersuchung bewertet. Verzeichnisse, die relevanten Code enthalten, sind hervorgehoben. Die Codeverteilung wird außerdem in Abb. 33 veranschaulicht.

Codeumfang (in %)	Codezeilen	Quellcode-dateien	Verzeichnis	Inhalt	Beinhaltet relevante Code-abschnitte?
36.65	373.182	619	hw	Emulator-Frontends	Ja[49]
6.57	66.905	694	tests	Automatisch ausführbare Testfälle	Nein, da zur Laufzeit nicht aktiv[50]
6.44	65.667	68	Wurzel-verzeichnis[51]	Grundlegende Bestandteile von QEMU	Ja
5.31	54.165	16	disas	Funktionen zum Disassemblieren von Maschinen-Code. Umfasst neben der i386-Architektur eine Reihe weiterer Architekturen.	Nein, da bei Verwendung von KVM weitestgehend ungenutzt
4.95	50.490	53	block	Backends zur Anbindung an Blockgeräte	Ja
4.07	41.444	19	target-arm	Architekturspezifischer Code zur Emulation der Architektur arm	Nein, da die Architektur nicht untersucht wird
3.84	39.191	22	target-ppc	Architekturspezifischer Code zur Emulation der Architektur PowerPC	Nein, da die Architektur nicht untersucht wird
3.45	35.194	12	target-mips	Architekturspezifischer Code zur Emulation der Architektur MIPS	Nein, da die Architektur nicht untersucht wird
2.93	29.856	20	linux-user	Code zur Ausführung einzelner Programme einer nicht nativen Architektur	Nein, da keine vollständige Betriebssystem-ausführung
2.49	25.395	20	target-i386	Architekturspezifischer Code zur Emulation der Architektur i386 (sowohl 32- als auch 64-Bit). Beinhaltet Anbindung an	Ja

49 Eine genauere Differenzierung findet in Abschnitt 5.6.5 statt.
50 Da die Testfälle jedoch über die Entwicklungsarbeit und die Codequalität von QEMU Aufschluss geben können, werden sie in Abschnitt 5.6.3 betrachtet.
51 Die Angaben beziehen sich ausschließlich auf das Wurzelverzeichnis selbst. Unterordner sind nicht berücksichtigt.

Codeumfang (in %)	Codezeilen	Quellcode-dateien	Verzeichnis	Inhalt	Beinhaltet relevante Code-abschnitte?
				KVM.	
2.41	24.632	13	tcg	Emulation ohne Hardwareunterstützung	Nein, da bei Verwendung von KVM nicht verwendet
1.98	20.225	37	ui	User Interface, insbesondere VNC- und Spice-Unterstützung	Nein, da nur administrativ erreichbar (s. Abschnitt 5.6.5.5)
1.56	15.925	50	util	Allgemeine Funktionen und Strukturen	Ja
1.45	14.772	16	migration	Code zur Live-Migration	Nein, da keine Interaktion mit dem Gast besteht[52]
1.39	14.165	15	target-s390x	Architekturspezifischer Code zur Emulation der Architektur S390	Nein, da die Architektur nicht untersucht wird
1.24	12.704	14	target-sparc	Architekturspezifischer Code zur Emulation der Architektur Sparc	Nein, da die Architektur nicht untersucht wird
1.12	11.449	5	target-tricore	Architekturspezifischer Code zur Emulation der Architektur TriCore	Nein, da die Architektur nicht untersucht wird
1.03	10.531	5	libdecnumber	Mathematische Bibliothek, die in Quellcodeform eingebunden wird	Nein, da Bibliotheken nicht untersucht werden (s. Abschnitt 5.1)
0.91	9.325	22	net	Backends zur Netzwerkanbindung	Ja
0.89	9.124	19	slirp	Code zur Erstellung eines lokalen virtuellen Netzwerkes unabhängig vom Linux-Kernel	Nein, da zur Servervirtualisierung nicht genutzt
0.86	8.825	14	audio	Backends zur Anbindung von Audiogeräten	Nein, da zur Servervirtualisierung nicht genutzt
0.75	7.722	1	fpu	Emulation einer Floating Point Unit (FPU)	Nein, da nicht zusammen mit KVM genutzt

52 Die Absicherung der Netzwerkkommunikation wird in Kapitel 8 untersucht.

Codeumfang (in %)	Codezeilen	Quellcode-dateien	Verzeichnis	Inhalt	Beinhaltet relevante Code-abschnitte?
0.67	6.830	9	qga	Softwareschnittstelle zur Kommunikation mit Managementwerkzeugen innerhalb des Gastes	Nein, da in der Praxis quasi nicht genutzt
0.64	6.581	8	target-cris	Architekturspezifischer Code zur Emulation der Architektur Cris	Nein, da die Architektur nicht untersucht wird
0.60	6.142	12	target-xtensa	Architekturspezifischer Code zur Emulation der Architektur xTensa	Nein, da die Architektur nicht untersucht wird
0.53	5.441	10	target-alpha	Architekturspezifischer Code zur Emulation der Architektur Alpha	Nein, da die Architektur nicht untersucht wird
0.51	5.244	12	crypto	Kryptofunktionen, die unter anderem zur Verschlüsselung von Blockgeräten eingesetzt werden können	Ja
0.49	5.020	6	target-m68k	Architekturspezifischer Code zur Emulation der Architektur m68k	Nein, da die Architektur nicht untersucht wird
0.42	4.309	8	bsd-user	Code zur Ausführung einzelner Programme einer nicht nativen Architektur auf einem BSD-System	Nein, da keine vollständige Betriebssystem-ausführung stattfindet
0.37	3.825	6	target-sh4	Architekturspezifischer Code zur Emulation der Architektur sh4	Nein, da die Architektur nicht untersucht wird
0.33	3.423	6	target-unicore32	Architekturspezifischer Code zur Emulation der Architektur unicore32	Nein, da die Architektur nicht untersucht wird
0.33	3.377	6	target-microblaze	Architekturspezifischer Code zur Emulation der Architektur microblaze	Nein, da die Architektur nicht untersucht wird
0.32	3.324	13	target-openrisc	Architekturspezifischer Code zur Emulation der Architektur openrisc	Nein, da die Architektur nicht untersucht wird
0.29	2.960	4	target-tilegx	Architekturspezifischer Code zur Emulation der Architektur tilegx	Nein, da die Architektur nicht untersucht wird
0.27	2.759	11	qapi	Schnittstelle zur Kommunikation mit Managementanwendungen	Nein, da nur administrativ zugänglich
0.26	2.702	11	qobject	Allgemeine Funktionen und Strukturen	Ja

Codeumfang (in %)	Codezeilen	Quellcode-dateien	Verzeichnis	Inhalt	Beinhaltet relevante Code-abschnitte?
0.26	2.650	5	qom	Schnittstelle zur Kommunikation mit Managementanwendungen	Nein, da nur administrativ zugänglich
0.22	2.299	7	target-lm32	Architekturspezifischer Code zur Emulation der Architektur lm32	Nein, da die Architektur nicht untersucht wird
0.20	2.119	10	backends	Sonstige Backends, die nicht zur Anbindung an Blockgeräte, Netzwerke, Audiogeräte oder Dateisysteme dienen	Ja[53]
0.16	1.696	5	fsdev	Backend zur Durchreichung von Dateisystemen wie 9p	Nein, da in der Servervirtualisierung nicht genutzt
0.14	1.442	4	contrib	Hilfsprogramme zur Nutzung von gemeinsamem Speicher zwischen Gast und Host	Nein, da in der Servervirtualisierung nicht genutzt
0.13	1.340	4	pc-bios	BIOS-Implementierungen für den Gast	Nein, da nur von geringer sicherheitstechnischer Bedeutung
0.12	1.309	5	target-moxie	Architekturspezifischer Code zur Emulation der Architektur moxie	Nein, da die Architektur nicht untersucht wird
0.10	1.051	5	replay	Code zum Testen von QEMU und zur Fehlersuche	Nein, da im produktiven Einsatz nicht genutzt
0.07	715	4	trace	Code zur Aufzeichnung der Aktivitäten von QEMU. Dient primär zur Fehlersuche.	Nein, da im produktiven Einsatz nicht genutzt
0.04	442	40	stubs	Testcode	Nein, da im produktiven Einsatz nicht genutzt
0.03	343	1	scripts	Skripte mit verschiedenen Verwendungszwecken, primär zur Fehlersuche und Installation von QEMU.	Nein, da zur Laufzeit des Gastes nicht aktiv

Tabelle 2: Quellcodeverzeichnisse von QEMU

53 Tatsächlich relevant sind jedoch nur der Zufallzahlengenerator und die Anbindung an TPM.

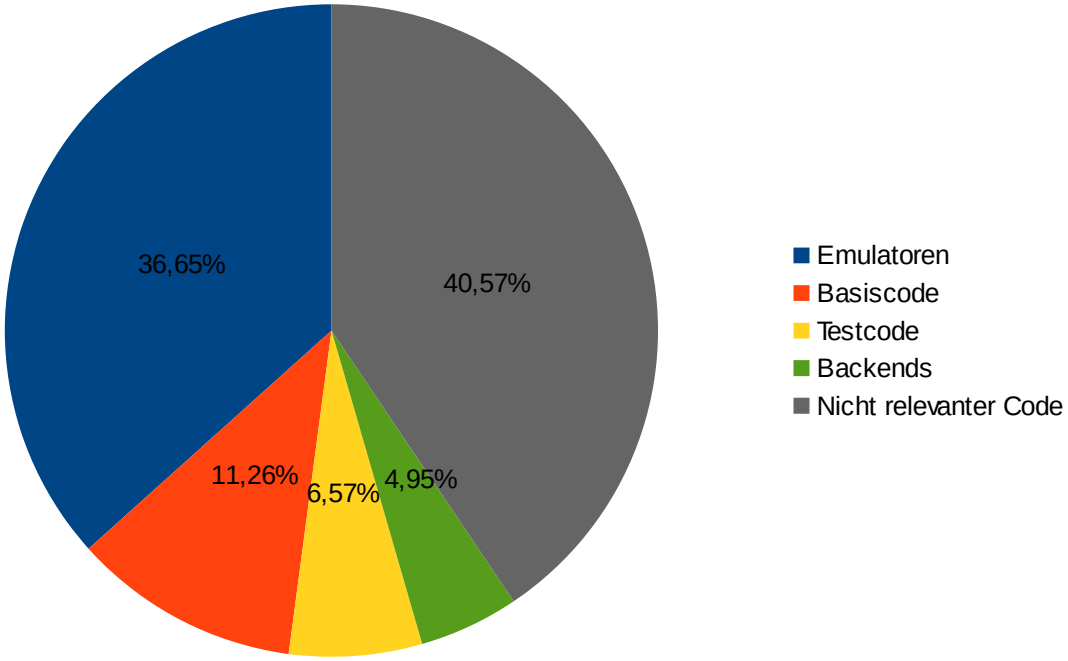

Abbildung 33: Anteile relevanter Codeabschnitte

5.6.2 Basiscode

Die mit 11,26 % (114.933 Zeilen) zweitgrößte relevante Codegruppe bilden allgemeine Funktionen von QEMU. Hierzu wird Quellcode gezählt, der sowohl im Wurzelverzeichnis als auch in den Verzeichnissen target-i386[54], util, crypto und qobject gespeichert ist. Mit Ausnahme des Codes, der im Verzeichnis target-i386 gespeichert ist, ist dieser weitgehend architekturneutral.

Die sicherheitstechnische Relevanz des Codes variiert dabei erheblich. So dient ein nicht unerheblicher Teil des Codes zur Initialisierung einer virtuellen Maschine und wird somit vor deren Start ausgeführt. Aufgrund dieses zeitlichen Ablaufs ist eine Beeinflussung aus dem Gast heraus regelmäßig ausgeschlossen. Des Weiteren umfasst diese Codegruppe eine Reihe von Hilfsfunktionen, z. B. zur Threadverwaltung, die zwar während der Ausführung des Gastes genutzt werden, jedoch nicht mit diesem kommunizieren und somit für diese Untersuchung nicht relevant sind.

Des Weiteren umfasst diese Codegruppe Code zur Anbindung an KVM. Hierzu gehören Funktionen, die nur durch KVM vom Gast getrennt sind und mit diesem quasi unmittelbar interagieren. Sicherheitskritische Fehler in diesen Codeabschnitten würden daher QEMU weitestgehend unabhängig von der Konfiguration und den eingesetzten Emulatoren angreifbar machen. Diese Codeabschnitte besitzen daher eine so weitreichende sicherheitstechnische Relevanz, dass sie teilweise in Abschnitt 5.9 manuell inspiziert werden.

5.6.3 Testcode

Der zweitgrößte Anteil an Quellcodeteilen mit 6,57 % (66.905 Zeilen) nimmt Testcode ein. Auch wenn dies auf eine hohe Testabdeckung und somit auf eine hohe Codequalität schließen lässt, so muss diese Schlussfolgerung doch relativiert werden. Tatsächlich entfallen ca. 42 % (28.141 Zeilen) des Testcodes auf das

54 Der vergleichsweise große Codeumfang im Verzeichnis *target-i386* geht auf die rein softwarebasierte Virtualisierung zurück. Der Code zur Anbindung an KVM umfasst hingegen nur einen geringen Anteil.

Testen verschiedener Prozessortypen. Falls eine hardwareunterstützte Virtualisierung mittels KVM verwendet wird, haben die hierdurch getesteten Codebestandteile von QEMU jedoch keine Relevanz. Der übrige Teil des Testcodes von ca. 57 % (38.367 Zeilen) dient zum Testen der weiteren Bestandteile von QEMU. Im Vordergrund steht hierbei das Testen der Emulatoren, wozu eine dedizierte Schnittstelle existiert. Diese ermöglicht es dem Testcode, auf die gleiche Art mit einem Emulator zu interagieren wie ein Gast, ohne einen solchen jedoch auszuführen. Dieser Ansatz ermöglicht ein systematisches, performantes und vergleichsweise einfaches Testen eines Emulators. Erschwert wird das Schreiben von Testcode jedoch dadurch, dass die Testschnittstelle ohne Weiteres nur mit der Programmiersprache C genutzt werden kann. Zwar kann das Entwickeln eines Emulators in der Sprache C aus verschiedenen Gründen sinnvoll sein. C ist jedoch, wenn es um das Verfassen von Testcode geht, eine verhältnismäßig ausdrucksschwache Sprache und daher nur bedingt geeignet. Eine zusätzliche Anbindung für weitere Programmiersprachen wäre hier wünschenswert. Es darf vermutet werden, dass dies die Bereitschaft der Entwickler erhöhen würde, umfassendere Tests zu programmieren.

Insgesamt existieren 97 Testsuites[55] mit jeweils einem oder mehreren Testfällen. Die Gesamtzahl der Testfälle beträgt ca. 716[56]. Hierbei ist jedoch zu beachten, dass die Anzahl der Testfälle pro Testsuite stark variiert.[57] So umfassen die oberen 20 % (19) der Testsuites mit den meisten Testfällen insgesamt 61 % (438) der Testfälle. Die unteren 20 % der Testsuites umfassen lediglich jeweils einen Testfall und somit nur 0,03 % der Testfälle. Tatsächlich enthalten lediglich 22,6 % (22) der Testsuites 10 oder mehr Testfälle. Es darf davon ausgegangen werden, dass Testsuites mit einer solch geringen Anzahl an Testfällen keine fundierte Aussage über die Güte der zu testenden Komponente liefern können. Tatsächlich zeigt eine grobe Sichtung der Testfälle, dass diese fast ausnahmslos mittels Positiv-Beispielen arbeiten. Es wird also primär auf ein korrektes Verhalten für zu erwartende Eingaben getestet. Das Testen der Sicherheit der jeweiligen Komponenten durch unerwartete Eingaben unterbleibt hingegen fast völlig. Somit ist festzuhalten, dass zwar in einem gewissen Umfang auf Funktionstüchtigkeit, jedoch nur sporadisch auf Sicherheit getestet wird.

Wenngleich also das Vorhandensein einer dedizierten Schnittstelle zu Softwaretests als positiv zu werten ist, so muss die Testabdeckung als schwach beurteilt werden.[58]

5.6.4 Backends

Eine weitere sicherheitsrelevante Codegruppe sind die Backends, die in den Verzeichnissen block, net und backends gespeichert sind. Mit 4,95 % (50.490 Zeilen) des Codeumfangs der Gesamtsoftware ist die Anbindung an Speichergeräte der größte Teil dieser Codegruppe. Der Grund ist die recht große Anzahl an unterstützten Dateiformaten und Speichertechniken von Blockgeräten (s. Abschnitt 3.5.3.3). Mit nur 0,91 % Codeanteil (9.325 Zeilen) nimmt die Netzwerkanbindung von QEMU einen recht geringen Codeumfang ein. Dieser Codeanteil umfasst neben der relevanten Anbindung mittels tap-Device auch eine Reihe anderer Anbindungsmethoden, die für die Servervirtualisierung nicht relevant sind. Ebenso sind eine Reihe dieser Methoden für verschiedene Betriebssysteme implementiert. Der tatsächlich für diese Studie relevante Codeumfang umfasst ca. 4.400 Zeilen. Einen noch geringeren Codeumfang besitzen die sonstigen relevanten Backends. Hier kann von ca. 1.500 Zeilen Quellcode ausgegangen werden.

55 Eine Testsuite bezeichnet hierbei eine Menge von Testfällen, die eine durch die Testsuite bestimmte Komponente von QEMU testet. Die Anzahl der Testsuites entspricht somit in etwa den getesteten Komponenten.

56 Die tatsächliche Anzahl an Testfällen kann geringfügig abweichen, da nicht ausgeschlossen werden kann, dass in Einzelfällen mehrere Testfälle als ein Testfall gezählt wurden.

57 Die Standardabweichung beträgt bei einem Mittelwert von 7,38 Testfällen pro Testsuite 11,2.

58 Es kann jedoch nicht abschließend ausgeschlossen werden, dass außerhalb des offiziellen Sourcecode-Repositoriums weiterer Testcode existiert.

5.6.5 Emulatoren

5.6.5.1 Übersicht

Mit insgesamt 36,65 % (373.182 Zeilen) des Codeumfangs besteht der mit Abstand größte Teil von QEMU aus der Implementierung von Emulatoren, die im Verzeichnis hw gespeichert sind. Typischerweise bestehen sie aus einem Frontend, das die eigentliche Emulation durchführt, und einem Backend, mit dem die Funktion des Emulators umgesetzt wird (s. Abschnitt 3.5.3). Sofern eine solche Trennung in Frontend und Backend nicht zweckdienlich ist, ist der Emulator inklusive seiner eigentlichen Funktion Teil dieser Codegruppe.

Ein wesentliches Merkmal ist, dass der Code quasi unmittelbar mit dem Gast interagiert und somit als besonders sicherheitskritisch angesehen werden muss. Zu beachten ist jedoch, dass bei der Virtualisierung eines konkreten Gastes immer nur eine kleine Anzahl an Emulatoren eingesetzt wird und somit ein Großteil dieses Codes nicht in Benutzung ist. In Tabelle 3 werden die Unterverzeichnisse aufgeführt und wird die Relevanz des enthaltenen Quelltextes beurteilt.

Codeumfang (in %)	Codezeilen	Quellcode-dateien	Verzeichnis	Inhalt	Beinhaltet relevante Code-abschnitte?
9,31	34.748	40	net	Frontend für Netzwerkschnittstellen, sowohl Emulation als auch Virtio	Ja
8,06	30.099	31	usb	Frontend für USB-Anbindung	Nein, da zur Anbindung von Eingabegeräten sinnvollere Alternativen existieren (s. Abschnitt 5.6.5.5)
7,50	28.020	42	arm	Frontend für arm-spezifische Hardware	Nein, da die Architektur nicht untersucht wird
6,84	25.532	32	display	Frontend für Grafikkarten	Ja
5,15	19.220	27	ppc	Frontend für PowerPC-spezifische Hardware	Nein, da die Architektur nicht untersucht wird
4,17	15.591	12	scsi	Frontend für SCSI-Geräte	Ja
4,01	14.989	32	timer	Zeitgeber	Nein, da Zeitgeber im Rahmen der Emulation von APIC durch KVM realisiert werden.
3,93	14.694	34	intc	Emulation von Interrupt-Controllern	Nein, da der Interrupt-Controller via APIC durch KVM

Codeumfang (in %)	Codezeilen	Quellcode-dateien	Verzeichnis	Inhalt	Beinhaltet relevante Code-abschnitte?
					emuliert wird
3,62	13.544	36	misc	Verschiedene Emulatoren, die nicht in andere Kategorien passen	Nein, da nicht für die Server-virtualisierung verwendet
3,41	12.738	18	i386	Frontend für i386-spezifische Hardware	Ja
3,30	12.352	18	audio	Emulation von Audio-Geräten	Nein, da Audio für die Servervirtuali-sierung nicht relevant ist
2,99	11.167	28	char	Emulation von Character Devices, wie z. B. seriellen Schnittstellen	Ja
2,88	10.777	15	block	Emulation von Blockgeräten, die nicht in die Kategorie SCSI oder IDE gehören, wie z. B. das Virtio-Blockgerät	Ja
2,56	9.571	13	ide	Emulation von IDE-Controllern	Ja
2,41	9.028	13	dma	Emulation von DMA-Controllern	Nein, da für die Servervirtuali-sierung nicht genutzt
2,33	8.704	14	9pfs	Emulation des Dateisystems 9p	Nein, da für die Servervirtuali-sierung nicht genutzt
2,30	8.600	15	s390x	Frontend für S390-spezifische Hardware	Nein, da die Architektur nicht untersucht wird
2,26	8.467	10	virtio	Emulatoren, die auf Virtio basieren	Ja
1,93	7.238	15	input	Emulation von Eingabegeräten, wie Human Interface Devices (HID) und Virtio-input	Ja
1,86	6.958	12	pci	Emulation des PCI-Busses	Ja
1,57	5.887	15	core	Allgemeine Strukturen und Funktionen zur Hardwareemulation	Ja
1,54	5.763	6	bt	Emulation von Bluetooth-Geräten	Nein, da für die Servervirtuali-sierung nicht genutzt
1,52	5.687	5	vfio	Hardwaredurchreichung	Ja
1,49	5.587	11	pci-host	Emulation des PCI-Host-Controllers	Ja

Codeumfang (in %)	Codezeilen	Quellcode-dateien	Verzeichnis	Inhalt	Beinhaltet relevante Code-abschnitte?
1,44	5.389	7	sd	Emulation von SD-Memory-Cards	Nein, da für die Servervirtualisierung nicht genutzt
1,43	5.363	7	xen	Emulationen spezifisch für den Hypervisor XEN	Nein, da XEN nicht untersucht wird
1,12	4.207	10	acpi	Emulation von ACPI	Ja
1,09	4.087	9	mips	Frontend für Mips-spezifische Hardware	Nein, da die Architektur nicht untersucht wird
0,71	2.660	10	i2c	Emulation von I^2C und SMBus	Nein, da für die Servervirtualisierung nicht genutzt
0,64	2.396	7	gpio	Emulation von allgemeinen IO-Ports	Nein, da für die Servervirtualisierung nicht genutzt
0,62	2.343	7	isa	Emulation eines ISA-Busses	Ja
0,54	2.034	5	ssi	Emulation einer synchronen seriellen Schnittstelle	Nein, da für die Servervirtualisierung nicht genutzt
0,52	1.946	5	nvram	Emulation von NVRAM	Nein, da für die Servervirtualisierung nicht genutzt
0,48	1.818	2	sparc	Frontend für Sparc-spezifische Hardware	Nein, da die Architektur nicht untersucht wird
0,47	1.766	3	tpm	Emulation eines TPM-Geräts	Ja
0,42	1.603	5	sh4	Frontend für sh4-spezifische Hardware	Nein, da die Architektur nicht untersucht wird
0,38	1.425	7	pci-bridge	Kopplung des PCI-Busses an andere Busse	Nein, da für die Servervirtualisierung nicht genutzt
0,32	1.199	5	m68k	Frontend für m68k-spezifische Hardware	Nein, da die Architektur nicht untersucht wird

Codeumfang (in %)	Codezeilen	Quellcode-dateien	Verzeichnis	Inhalt	Beinhaltet relevante Code-abschnitte?
0,32	1.226	3	alpha	Frontend für Alpha-spezifische Hardware	Nein, da die Architektur nicht untersucht wird
0,30	1.139	1	smbios	Emulation von SMBIOS-Funktionen	Ja
0,27	1.032	1	sparc64	Frontend für Sparc64-spezifische Hardware	Nein, da die Architektur nicht untersucht wird
0,24	903	4	watchdog	Emulation eines Watchdogs	Nein, da für die Servervirtualisierung nicht genutzt
0,21	793	3	xtensa	Frontend für xtensa-spezifische Hardware	Nein, da die Architektur nicht untersucht wird
0,20	775	2	ipack	Emulation von IndustryPack-Geräten	Nein, da für die Servervirtualisierung nicht genutzt
0,17	657	4	cpu	Emulation von in die CPU eingebetteter Hardware auf ARM-Plattformen	Nein, da die Architektur nicht untersucht wird
0,15	561	3	microblaze	Frontend für microblaze-spezifische Hardware	Nein, da die Architektur nicht untersucht wird
0,14	550	2	lm32	Frontend für lm32-spezifische Hardware	Nein, da die Architektur nicht untersucht wird
0,12	464	1	mem	Code für das Hotplugging von RAM-Riegeln	Nein, wenn Balloning, dann über Virtio
0,12	459	2	cris	Emulation von Hardware auf einem AXIS-Entwicklungsboard	Nein, da die Architektur nicht untersucht wird
0,10	403	2	xenpv	Emulation von paravirtualisierten Geräten bei Nutzung des Hypervisors XEN	Nein, da XEN nicht untersucht wird
0,08	314	3	openrisc	Frontend für OpenRisc-spezifische Hardware	Nein, da die Architektur nicht untersucht wird
0,07	288	2	pcmcia	Emulation eines PCMCIA-Controllers	Nein, da für die Servervirtualisierung nicht

Codeumfang (in %)	Codezeilen	Quellcode-dateien	Verzeichnis	Inhalt	Beinhaltet relevante Code-abschnitte?
					genutzt
0,04	156	1	moxie	Frontend für moxie-spezifische Hardware	Nein, da die Architektur nicht untersucht wird
0,03	140	1	unicore32	Frontend für unicore32-spezifische Hardware	Nein, da die Architektur nicht untersucht wird
0,03	125	1	tricore	Frontend für tricore-spezifische Hardware	Nein, da die Architektur nicht untersucht wird

Tabelle 3: Quellcodeverzeichnisse der Emulatoren

5.6.5.2 Chipsatz

Eine wesentliche Komponente einer jeden x86-Architektur ist der Chipsatz, der die Kommunikation mit Peripheriegeräten mithilfe von Datenbussen ermöglicht. Die von QEMU emulierbaren Chipsätze sind der FX440 und der Q35 (s. Abschnitt 3.5.2). Einer dieser beiden Chipsätze wird somit in jeder von QEMU betriebenen virtuellen Maschine genutzt.[59] Sicherheitskritische Fehler haben hier also besonders weitreichende Folgen. Darüber hinaus stellt QEMU eine Reihe von Funktionen zur Verfügung, mit denen Emulatoren auf die virtuellen Datenbusse, insbesondere PCI, zugreifen können. Tabelle 4 listet die wesentlichen Quellcodedateien hierzu auf.

Name	Quellcodedateien
FX440-Chipsatz	hw/pci-host/piix.c
Q35-Chipsatz	hw/pci-host/q35.c
Funktionen zum Zugriff auf den virtuellen PCI-Bus	hw/pci/pci.c hw/pci/msi.c, hw/pci/msix.c
Funktionen zum Zugriff auf den virtuellen ISA-Bus	hw/isa/isa-bus.c

Tabelle 4: Quellcodedateien zur Realisierung des Chipsatzes und der Datenbusse

5.6.5.3 Blockgeräte

Eine weitere wesentliche Geräteklasse sind Blockgeräte, deren Emulation auf mehreren Wegen geschehen kann. So kann dem Gast sowohl ein IDE- als auch ein SCSI-Bus zur Verfügung gestellt werden. Bei Erstem geschieht dies durch Emulation eines IDE-Controllers, typischerweise des Intel 82801. Bei Letzterem wird dies durch die Emulation eines SCSI-Controllers, typischerweise des Symbios Logic 53c895a, oder alternativ paravirtualisiert mittels Virtio erreicht. Darüber hinaus stehen zur Anbindung von Blockgeräten weitere Emulatoren, wie z. B. für den MegaRAID SAS 8708EM2, zur Verfügung, die jedoch aufgrund geringer

59 Da der Controller FX440 unter anderem von libvirt standardmäßig genutzt wird, wird er in der Praxis deutlich häufiger eingesetzt als der Q35.

Verbreitung unberücksichtigt bleiben. Ebenso bleiben als veraltet geltende Emulatoren unberücksichtigt. Tabelle 5 führt die relevanten Emulatoren auf.

Name	Bus	Gerätetyp	Beschreibung	Quellcodedateien
piix3-ide	IDE	Controller	Intel 82371SB – IDE-Controller	hw/ide/piix.c
piix4-ide	IDE	Controller	Intel 82371AB – IDE-Controller	hw/ide/piix.c
ich9-ahci	IDE	Controller	Intel 82801IR/IO/IH – SATA-Controller	hw/ide/ich.c
ide-cd	IDE	CDROM	Generische Emulation eines IDE-CD-ROMs	hw/ide/qdev.c
ide-hd	IDE	Festplatte	Generische Emulation einer IDE-Festplatte	hw/ide/qdev.c
lsi53c895a	SCSI	Controller	Symbios Logic 53c895a – SCSI-Controller	hw/scsi/lsi53c895a.c
scsi-cd	SCSI	CDROM	Generische Emulation eines SCSI-CD-ROMs	hw/scsi/scsi-disk.c
scsi-hd	SCSI	Festplatte	Generische Emulation einer SCSI-Festplatte	hw/scsi/scsi-disk.c
scsi-block	SCSI	Festplatte	Durchreichung eines allgemeinen Blockgeräts mittels SCSI-Befehlen	hw/scsi/scsi-disk.c
virtio-scsi-pci	SCSI	Festplatte	Durchreichung eines SCSI-Geräts	hw/scsi/virtio-scsi.c
vhost-scsi-pci	SCSI	Generisch	Durchreichung von SCSI-Geräten mittels Vhost	hw/scsi/vhost-scsi.c

Tabelle 5: Emulatoren von Blockgeräten

5.6.5.4 Netzwerkanbindung

Relevant sind ferner Emulatoren zur Netzwerkanbindung. Wenngleich QEMU hierzu eine große Anzahl an verschiedenen Emulatoren bereitstellt, wird jedoch in einem überwiegenden Teil der Anwendungsfälle einer der Emulatoren verwendet, die in Tabelle 6 aufgeführt sind. Darüber hinaus existierende Emulatoren von Netzwerkkarten bleiben unberücksichtigt.

Name	Beschreibung	Quellcodedateien
e1000	Intel e1000-Netzwerkkarte	hw/net/e1000.c
ne2k_pci	NE2000-Netzwerkkarte	hw/net/ne2000.c
pcnet	AMD PC-Net-II-Netzwerkkarte	hw/net/pcnet.c
rtl8139	RTL8139-Netzwerkkarte	hw/net/rtl8139.c
virtio-net-pci	Virtio-Netzwerkkarte	hw/net/virtio-net.c

Tabelle 6: Emulatoren von Netzwerkkarten

5.6.5.5 Eingabegeräte

Die Anbindung von Eingabegeräten, konkret von Tastatur und Maus, kann auf verschiedenen Wegen erfolgen. Eine Möglichkeit stellt hier der Universal Serial Bus (USB) dar. Die Nutzung von USB hat den Vorteil, dass neben Eingabegeräten auch weitere USB-Geräte an den Gast angeschlossen werden können. Hierbei können dem Gast sowohl virtuelle, also emulierte Geräte, als auch reale Geräte zur Verfügung gestellt werden. Wenngleich dies insbesondere in der Desktopvirtualisierung einen hohen Nutzen hat, so werden in der Servervirtualisierung regelmäßig nur Tastatur und Maus benötigt. Eine Ausnahme bildet hier lediglich die Anbindung eines virtuellen Tablets, um Positionierungsprobleme des Cursors zu vermeiden. Ein Nachteil bei der Nutzung von USB ist jedoch dessen Komplexität. Bei ausschließlicher Nutzung von

Maus und Tastatur steht hier also eine vergleichsweise simple Funktion einer komplexen Implementierung gegenüber.

Aus diesem Grund empfiehlt es sich, insbesondere bei erhöhten Sicherheitsanforderungen, gänzlich auf die Emulation eines USB-Busses zu verzichten. Maus und Tastatur können alternativ über eine virtuelle PS/2-Schnittstelle angebunden werden, deren Implementierung wesentlich weniger komplex ist. Auch können Eingabegeräte mittels Virtio paravirtualisiert werden. Dies bietet sich vor allem an, wenn bereits andere Komponenten paravirtualisiert werden. Ein Vorteil gegenüber der Anbindung mittels PS/2 ist auch, dass die Anbindung eines virtuellen Tablets möglich ist und somit die genannten Positionierungsprobleme des Cursors vermieden werden können.

Da somit der Einsatz von USB in der Servervirtualisierung nicht notwendig ist und einfach umgangen werden kann, wird USB nicht berücksichtigt. Die als relevant erachteten Emulatoren von Eingabegeräten sind in Tabelle 7 zusammengefasst.

Name	Beschreibung	Quellcodedateien
PS2-Tastatur- und Maus-Emulation	Anbindung von Tastatur und Maus mittels PS/2	hw/input/ps2.c
virtio-keyboard-pci	Anbindung einer Tastatur mittels Virtio	hw/input/virtio-input.c
virtio-mouse-pci	Anbindung einer Maus mittels Virtio	hw/input/virtio-input.c
virtio-tablet-pci	Anbindung eines Tablets mittels Virtio	hw/input/virtio-input.c

Tabelle 7: Emulatoren von Eingabegeräten

5.6.5.6 Grafikausgabe

Wenngleich die Administration von Gästen im Bereich der Servervirtualisierung ganz überwiegend durch einen Netzwerkdienst, wie z. B. ssh oder RDP, erfolgt, so ist der Zugang zum Gast über eine Konsole dennoch relevant. Verglichen mit der Desktopvirtualisierung sind die Performanceanforderungen in der Servervirtualisierung jedoch deutlich geringer. Dies gilt insbesondere für Gäste, die mittels einer textbasierten Konsole administriert werden. Zwar bestehen für Gäste, die durch eine grafische Konsole gesteuert werden, leicht erhöhte Leistungsanforderungen, aber diese sind geringfügig. Die Leistungsmerkmale einer virtuellen Grafikkarte sind in der Servervirtualisierung somit nebensächlich. Die Vielfalt der verschiedenen Emulatoren von QEMU birgt somit in der Servervirtualisierung keine signifikanten Vorteile. Wesentliches Auswahlkriterium ist vielmehr die Unterstützung durch das Gastbetriebssystem.[60] Aus diesem Grund beschränkt sich diese Studie auf nur zwei Emulatoren, die in Tabelle 8 aufgeführt sind.

Name	Beschreibung	Quellcodedateien
cirrus-vga	Unterstützt Standard-VGA-Auflösungen, die für Systeme mit Textkonsole ausreichend sind. Beim Einsatz grafischer Oberflächen sind diese jedoch deutlich in ihrer Auflösung beschränkt. Sehr weitreichende Softwareunterstützung.	hw/display/cirrus_vga.c
qxl	Kann auch höher auflösende Displays emulieren.	hw/display/qxl.c

Tabelle 8: Emulatoren für die grafische Ausgabe

60 Weil der paravirtualisierte Emulator virtio-vga nur von sehr wenigen Gastbetriebssystemen unterstützt wird, wird darauf verzichtet, ihn zu untersuchen.

5.6.5.7 Virtio

Von großer Bedeutung ist ferner die Paravirtualisierung basierend auf Virtio. Aufgrund des modularen Aufbaus von Virtio (s. Abschnitt 3.5.3.2) setzen sich Emulatoren auf Basis von Virtio aus verschiedenen Bestandteilen zusammen. Auch wenn diese Emulatoren bereits in den vorhergehenden Abschnitten aufgeführt wurden, so greifen sie auf allgemeine Komponenten zurück. Fehler in diesen grundlegenden Komponenten können sich daher auf verschiedene paravirtualisierte Geräte auswirken. Weil paravirtualisierte Geräte sehr häufig genutzt werden, insbesondere beim Einsatz von Linux als Gastsystem, sind sie für diese Untersuchung relevant. Tabelle 9 fasst die entsprechenden Komponenten zusammen.

Name	Quellcodedateien
Virtio-Anbindung	hw/virtio/virtio.c hw/virtio/virtio-pci.c hw/virtio/virtio-bus.c hw/virtio/virtio-mmio.c hw/virtio/dataplane/vring.c
Vhost-Anbindung	hw/virtio/vhost.c hw/virtio/vhost-backend.c hw/virtio/vhost-user.c

Tabelle 9: Bestandteile von Virtio und Vhost

5.6.5.8 Anbindung an KVM

Darüber hinaus existieren Hardwarekomponenten, die im Zusammenhang mit KVM genutzt werden. Diese ergänzen den Quellcode zur Anbindung an KVM im Verzeichnis target-i386 (s. Abschnitt 5.6.2). Die wesentlichen Quellcodedateien sind in Tabelle 10 aufgeführt.

Name	Quellcodedateien
Anbindung an KVM-Emulatoren	hw/i386/kvm/apic.c hw/i386/kvm/clock.c hw/i386/kvm/i8254.c.c hw/i386/kvm/i8259.c hw/i386/kvm/ioapic.c

Tabelle 10: KVM-spezifische Emulatoren

5.6.5.9 Sonstige Geräte

Außerdem existieren Emulatoren, die sich in keine der genannten Geräteklassen einordnen lassen, jedoch für diese Untersuchung relevant sind.

Zu dieser Gruppe gehört ein Emulator, der die dynamische Anpassung des Arbeitsspeichers mittels Virtio ermöglicht. Er ermöglicht es dem Gast, Arbeitsspeicher vom Host anzufordern und an diesen zurückzugeben. Somit kann der Speicherverbrauch, in vom Host bestimmten Grenzen, dem tatsächlichen Bedarf angepasst werden (Ballooning).

Darüber hinaus benötigt der Gast, insbesondere für kryptografische Operationen, Zufallswerte. Diese können ihm im Rahmen der Paravirtualisierung durch einen entsprechenden Emulator bereitgestellt werden.

Ferner kann die Sicherheit des Schlüsselmaterials des Gastes durch ein TPM-Gerät erhöht werden. Hierzu kann der Host ein solches Gerät emulieren oder ein reales Gerät an den Gast durchreichen.

Die entsprechenden Emulatoren sind in Tabelle 11 zusammengefasst.

Name	Beschreibung	Quellcodedateien
virtio-ballon-pci	Emulator zu dynamischen Anspassung der Speichergröße	hw/virtio/virtio-balloon.c
virtio-rng-pci	Emulator zur Anbindung eines Zufallszahlengenerators	hw/virtio/virtio-rng.c
tpm-tis	Emulator zur Anbindung von TPM	hw/tpm/tpm_passthrough .c hw/tpm/tpm_tis.c hw/tpm/tpm_util.c

Tabelle 11: Sonstige Emulatoren

5.7 Untersuchung mithilfe der GNU Compiler Collection

Entsprechend dem in Abschnitt 5.2 beschriebenen Verfahren wird der Quelltext von QEMU mittels GCC in Version 6.1.0 übersetzt. Der Vorgang wird hierbei auf Komponenten für die x86-Architektur beschränkt, die KVM nutzen. Hierbei werden die Einstellungen für den GCC unabhängig von den Vorgaben des QEMU-Projekts gewählt und alle sicherheitsrelevanten Überprüfungsroutinen aktiviert. Die vollständige Konfiguration des Übersetzungsvorgangs ist in Anhang B aufgeführt. Die hierbei auftretenden Warnmeldungen werden im Folgenden zusammengefasst und ausgewertet.

Warnungen, die durch Quellcode verursacht werden, der an sich kein Bestandteil von QEMU ist, sondern zu Softwarebibliotheken oder Schnittstellen gehört, werden hierbei nicht berücksichtigt. Doppelnennungen, die durch das mehrfache Inkludieren einer Quellcodedatei entstehen, werden nur einmalig gezählt.

Insgesamt werden 16.114 Warnungen während des Übersetzungsprozesses gemeldet. Auf Grundlage der insgesamt 1.018.231 Quellcodezeilen bedeutet dies ca. 16 Warnungen pro 1.000 Zeilen Code. Die Warnungen sind in Tabelle 12 nach Anzahl und Art zusammengefasst. Die Bezeichnungen werden hierbei vom GCC übernommen. Eine vollständige Beschreibung wird in [GCC] gegeben. Tabelle 13 fasst die Art der Warnungen in Gruppen zusammen und schlüsselt diese nach der jeweiligen Quellcodedatei auf, in der die Warnung auftritt. Hierbei werden solche Warnungen, die sich auf als nicht relevant eingestuften Quelltext beziehen, nicht berücksichtigt. Tabelle 13 bezieht sich somit auf 420 Quellcodedateien mit insgesamt 276.025 Zeilen.

Anzahl	Bezeichnung
8.269	sign-conversion
5.771	conversion
1.014	sign-compare
458	missing-field-initializers
187	override-init
164	pointer-arith
49	float-conversion
45	null-dereference
33	jump-misses-init
33	float-equal

Anzahl	Bezeichnung
32	strict-overflow
27	stack-protector
24	bad-function-cast
4	logical-op
4	format-nonliteral
2	shift-negative-value

Tabelle 12: Warnungen des GCC

Verzeichnis	Gesamtzahl an Warnungen	Konvertierungs-warnungen	Initialisierung	Umgang mit Zeigern	Sonstige
block	2.215	2.116	0	57	42
target-i386	1.826	1.774	49	0	3
Wurzelverzeichnis	1.491	1.442	10	27	12
hw/scsi	745	731	9	3	2
hw/net	674	652	11	10	1
hw/display	622	596	9	9	8
hw/ide	328	313	14	1	0
hw/virtio	313	288	6	18	1
hw/block	300	261	31	6	2
hw/i386	297	288	5	2	2
hw/pci	289	232	57	0	0
util	252	203	1	21	27
hw/vfio	233	228	0	5	0
hw/acpi	147	137	9	1	0
net	147	142	0	3	2
hw/char	147	132	14	1	0
hw/i386/kvm	108	107	1	0	0
hw/net/rocker	73	70	2	0	1
hw/core	63	59	0	4	0
hw/tpm	44	44	0	0	0
qobject	43	22	19	1	1
hw/smbios	42	42	0	0	0
backends	41	37	0	0	4

Verzeichnis	Gesamtzahl an Warnungen	Konvertierungs-warnungen	Initialisierung	Umgang mit Zeigern	Sonstige
hw/i2c	29	25	3	1	0
hw/isa	29	27	2	0	0
hw/pci-host	28	24	4	0	0
hw/virtio/dataplane	13	12	0	1	0
crypto	9	9	0	0	0
hw/pci-bridge	6	2	4	0	0
hw/block/dataplane	1	1	0	0	0

Tabelle 13: Warnungen von GCC, nach Verzeichnis aufgeschlüsselt

Auffallend ist, dass sich insgesamt 93,4 % (15.054) der Warnungen auf potenzielle Konvertierungsfehler (sign-conversion, conversion, sign-compare, float-conversion) beziehen. Die Bewertung dieser Ausgangslage erweist sich als schwierig. Zwar kann die Verknüpfung von vorzeichenbehafteten und vorzeichenlosen Datentypen zu einer Wertänderung der Operanden und somit zu einem ungewollten Programmverhalten führen (s. [ARTSSA]). Dies ist jedoch abhängig von dem konkreten Wert der Operanden. Sofern ein Angreifer den Wert der Operanden nicht beeinflussen kann, liegt regelmäßig keine Schwachstelle vor. Ist eine solche Einflussnahme hingegen möglich, so kann dies prinzipiell eine Reihe schwerwiegender Schwachstellen zur Folge haben. Hierzu gehören unter anderem fehlerhafte Wertbereichs- und Feldgrenzenüberprüfungen. Entsprechende Warnungen können daher auf sicherheitskritische Schwachstellen hindeuten. Ein Teil dieser Meldungen, der sich auf besonders sicherheitskritischen Code bezieht, wird daher in Abschnitt 5.9 manuell überprüft. Hierbei zeigt sich, dass an keiner der manuell überprüften Codestellen eine Schwachstelle erkennbar ist. Eine vollständige Überprüfung unterbleibt jedoch aufgrund des großen Umfangs entsprechender Meldungen. Festzuhalten ist aber, dass bezüglich der Verknüpfung von vorzeichenlosen und vorzeichenbehafteten Datentypen eine systematische Unzulänglichkeit besteht. Hierzu gehört auch, dass eine entsprechende Überprüfung im Rahmen des üblichen Übersetzungsprozesses unterbleibt. Beim Hinzufügen von Code dürften dem jeweiligen Entwickler entsprechende, gegebenenfalls vorhandene Unzulänglichkeiten regelmäßig nicht bewusst sein.

Die zweitgrößte Klasse an Warnungen (missing-field-initializers und override-init) mit 645 Meldungen bezieht sich auf potenzielle Fehler bei der Initialisierung von Variablen. Nicht initialisierte Variablen können unter Umständen zu einem unerwarteten Programmverhalten führen und in dessen Folge Abstürze verursachen. Einem geschickten Angreifer ist es unter Umständen auch möglich, entsprechende Variablen auf einen von ihm gewählten Wert zu setzen und somit den weiteren Programmverlauf in seinem Sinne gezielt zu beeinflussen.

Die drittgrößte Klasse an Warnungen (pointer-arith und null-dereference) mit 209 Meldungen bezieht sich auf einen potenziell fehlerhaften Umgang mit Zeigern. Diesbezüglich ist zwischen einer Dereferenzierung eines NULL-Zeigers und einer fehlerhaften Arithmetik mit Zeigern zu unterscheiden. Bei Ersterem besteht die Gefahr, einen NULL-Zeiger zu dereferenzieren, was zu einem Programmabbruch durch das Betriebssystem führt. Dies kann potenziell für einen Angriff auf die Verfügbarkeit des Gastes genutzt werden. Damit tatsächlich eine Schwachstelle vorliegt, muss ein Angreifer ein solches Verhalten jedoch auch mit eingeschränkten Rechten aus dem Gast heraus provozieren können. Dem Angreifer müsste es also möglich sein, den Fehlerfall mittelbar über das Gastbetriebssystem auszulösen. Die tatsächliche Gefahr kann daher als gering eingestuft werden. Fehler der Zeigerarithmetik erlauben demgegenüber weitaus

ernsthaftere Angriffsmöglichkeiten. Gelingt es einem Angreifer, gezielt einen Zeiger falsch berechnen zu lassen, so ist unter Umständen nicht nur die Verfügbarkeit gefährdet, sondern auch eine Manipulation des Speichers außerhalb des Gastes denkbar.

Die übrigen Warnungen (strict-overflow, stack-protector, bad-function-cast, logical-op, format-nonliteral, shift-negative-value) umfassen nur noch 93 Meldungen. Wenngleich auch diese auf potenziell sicherheitskritische Fehler hinweisen können, so bilden sie doch eine in Relation zum Umfang des überprüften Quelltextes so kleine Menge, dass sie keine negativen Rückschlüsse auf die Qualität des Quelltextes zulassen.

5.8 Untersuchung mithilfe von flawfinder

Mit dem Tool flawfinder wird der Code daraufhin untersucht, ob er unsichere Funktionen enthält, deren Nutzung leicht zu sicherheitskritischen Fehlern führt. flawfinder analysiert hierbei, ob die jeweilige Funktion auf eine potenziell unsichere Art und Weise verwendet wird (s. [FLAWF]). Den beanstandeten Befunden wird ein Schweregrad zwischen 0 und 5 zugeordnet. Der Schweregrad 0 bedeutet hierbei „kein Risiko“, der Schwerergrad 5 bedeutet „hohes Risiko“. Da Vorkommnisse des Schweregrades 0 keine sicherheitstechnische Relevanz besitzen, wird darauf verzichtet, sie zu untersuchen. Die Anzahl der Vorkommnisse der Schweregrade 1 bis 5 sind in Tabelle 14 aufgeführt.

Verzeichnis	Schwere-grad 5	Schwere-grad 4	Schwere-grad 3	Schwere-grad 2	Schwere-grad 1	Gesamt
Wurzelverzeichnis	2	56	30	140	181	409
hw/vfio	2			8	13	23
hw/i386	1	9		32	27	69
util	1	6	9	44	49	109
hw/net		39		122	15	176
block		24	5	166	128	323
hw/display		19		42	10	71
hw/tpm		18		2	2	22
hw/pci		15		12	13	40
target-i386		8		56	8	72
hw/scsi		8		47	7	62
hw/char		8		16	6	30
hw/i2c		7		3	9	19
net		6	6	68	50	130
hw/pci-host		6		2	2	10
hw/block		5		16	1	22
hw/acpi		4		21	13	38
qobject		3		13	12	28
hw/core		2		20	31	53
crypto		2		19	2	23
hw/ide		2		13	2	17

Verzeichnis	Schwere-grad 5	Schwere-grad 4	Schwere-grad 3	Schwere-grad 2	Schwere-grad 1	Gesamt
hw/virtio		2		7		9
hw/isa		2		4	3	9
backends		1		6	1	8
hw/pci-bridge		1		1	1	3
hw/smbios				9	4	13

Tabelle 14: Warnungen durch flawfinder

Aufgrund der teilweise erheblichen sicherheitstechnischen Bedeutung von Warnungen der Schweregrade 3 bis 5 werden diese gänzlich manuell kontrolliert. Auf Basis dieser manuellen Prüfung lässt sich zusammenfassend feststellen, dass keiner der Befunde mit dem Schweregrad 3 bis 5 tatsächlich ein ausnutzbarer, sicherheitskritischer Fehler ist.

Auf eine manuelle Überprüfung der Meldungen mit Schweregrad 1 und 2 wird verzichtet, da die zugrunde liegenden potenziellen Fehler erfahrungsgemäß selten sicherheitskritisch sind und es gehäuft zu Falsch-Positiv-Meldungen kommt.

Die Überprüfungen zeigen, dass zwar vereinzelt Verletzungen der Coding Guideline (s. Abschnitt 5.3) existieren, die Befunde aber in keinem Fall sicherheitskritische Schwachstellen sind. Aufgrund der systematischen Überprüfungen ist insbesondere das Vorhandensein von Format-String-Schwachstellen sehr unwahrscheinlich. Ferner kommen unsichere Funktionen zur Verarbeitung von Zeichenketten im geringen Umfang zum Einsatz. Für die überwiegende Menge an Fällen kann eine Beeinflussung durch den Gast praktisch ausgeschlossen werden. In den übrigen Fällen werden die Funktionen in einer unbedenklichen Art und Weise verwendet. Die weiteren Befunde, soweit untersucht, stellen im Rahmen des in Abschnitt 2.1 beschriebenen Bedrohungsszenarios ebenfalls keine Sicherheitsschwachstellen dar.

5.9 Manuelle Codeinspektion

Eine manuelle Inspektion des Quelltextes wurde zum einen für besonders relevante Meldungen von GCC und flawfinder durchgeführt. Die Ergebnisse der manuellen Inspektion werden in den Abschnitten 5.7 bzw. 5.8 erläutert.

Darüber hinaus wurde eine Inspektion des Quelltextes durchgeführt, der zwischen KVM und den Emulatoren vermittelt. Hierbei sind die folgenden potenziell sicherheitskritischen Unzulänglichkeiten festgestellt worden.

Die Funktion kvm_cpu_exec (s. u.), die die Ausführung des Gastes veranlasst und den von KVM angegebenen Abbruchgrund verarbeitet, berücksichtigt den Abbruchgrund KVM_EXIT_HYPERCALL nicht.

```
1   int kvm_cpu_exec(CPUState *cpu)
2   {
3       struct kvm_run *run = cpu->kvm_run;
4       int ret, run_ret;
5
6       DPRINTF("kvm_cpu_exec()\n");
7
8       if (kvm_arch_process_async_events(cpu)) {
9           cpu->exit_request = 0;
10          return EXCP_HLT;
11      }
12
13      qemu_mutex_unlock_iothread();
14
```

```
15    do {
16        MemTxAttrs attrs;
17
18        if (cpu->kvm_vcpu_dirty) {
19            kvm_arch_put_registers(cpu, KVM_PUT_RUNTIME_STATE);
20            cpu->kvm_vcpu_dirty = false;
21        }
22
23        kvm_arch_pre_run(cpu, run);
24        if (cpu->exit_request) {
25            DPRINTF("interrupt exit requested\n");
26            /*
27             * KVM requires us to reenter the kernel after IO exits to complete
28             * instruction emulation. This self-signal will ensure that we
29             * leave ASAP again.
30             */
31            qemu_cpu_kick_self();
32        }
33
34        run_ret = kvm_vcpu_ioctl(cpu, KVM_RUN, 0);
35
36        attrs = kvm_arch_post_run(cpu, run);
37
38        if (run_ret < 0) {
39            if (run_ret == -EINTR || run_ret == -EAGAIN) {
40                DPRINTF("io window exit\n");
41                ret = EXCP_INTERRUPT;
42                break;
43            }
44            fprintf(stderr, "error: kvm run failed %s\n",
45                    strerror(-run_ret));
46  #ifdef TARGET_PPC
47            if (run_ret == -EBUSY) {
48                fprintf(stderr,
49                        "This is probably because your SMT is enabled.\n"
50                        "VCPU can only run on primary threads with all "
51                        "secondary threads offline.\n");
52            }
53  #endif
54            ret = -1;
55            break;
56        }
57
58        trace_kvm_run_exit(cpu->cpu_index, run->exit_reason);
59        switch (run->exit_reason) {
60        case KVM_EXIT_IO:
61            DPRINTF("handle_io\n");
62            /* Called outside BQL */
63            kvm_handle_io(run->io.port, attrs,
64                          (uint8_t *)run + run->io.data_offset,
65                          run->io.direction,
66                          run->io.size,
67                          run->io.count);
68            ret = 0;
69            break;
70        case KVM_EXIT_MMIO:
71            DPRINTF("handle_mmio\n");
72            /* Called outside BQL */
73            address_space_rw(&address_space_memory,
74                             run->mmio.phys_addr, attrs,
75                             run->mmio.data,
76                             run->mmio.len,
```

```
77                          run->mmio.is_write);
78          ret = 0;
79          break;
80      case KVM_EXIT_IRQ_WINDOW_OPEN:
81          DPRINTF("irq_window_open\n");
82          ret = EXCP_INTERRUPT;
83          break;
84      case KVM_EXIT_SHUTDOWN:
85          DPRINTF("shutdown\n");
86          qemu_system_reset_request();
87          ret = EXCP_INTERRUPT;
88          break;
89      case KVM_EXIT_UNKNOWN:
90          fprintf(stderr, "KVM: unknown exit, hardware reason %" PRIx64 "\n",
91                  (uint64_t)run->hw.hardware_exit_reason);
92          ret = -1;
93          break;
94      case KVM_EXIT_INTERNAL_ERROR:
95          ret = kvm_handle_internal_error(cpu, run);
96          break;
97      case KVM_EXIT_SYSTEM_EVENT:
98          switch (run->system_event.type) {
99          case KVM_SYSTEM_EVENT_SHUTDOWN:
100             qemu_system_shutdown_request();
101             ret = EXCP_INTERRUPT;
102             break;
103         case KVM_SYSTEM_EVENT_RESET:
104             qemu_system_reset_request();
105             ret = EXCP_INTERRUPT;
106             break;
107         case KVM_SYSTEM_EVENT_CRASH:
108             qemu_mutex_lock_iothread();
109             qemu_system_guest_panicked();
110             qemu_mutex_unlock_iothread();
111             ret = 0;
112             break;
113         default:
114             DPRINTF("kvm_arch_handle_exit\n");
115             ret = kvm_arch_handle_exit(cpu, run);
116             break;
117         }
118         break;
119     default:
120         DPRINTF("kvm_arch_handle_exit\n");
121         ret = kvm_arch_handle_exit(cpu, run);
122         break;
123     }
124 } while (ret == 0);
125
126 qemu_mutex_lock_iothread();
127
128 if (ret < 0) {
129     cpu_dump_state(cpu, stderr, fprintf, CPU_DUMP_CODE);
130     vm_stop(RUN_STATE_INTERNAL_ERROR);
131 }
132
133 cpu->exit_request = 0;
134 return ret;
135}
```

Die grundlegenden Arbeitsschritte der Funktion sind:

1. Vorbereitung der Ausführung von Gastcode mit der Funktion kvm_arch_post_run
2. Ausführung von Gastcode mit einer entsprechenden Anweisung an /dev/kvm
3. Nachbearbeitung der Ausführung von Gastcode mit der Funktion kvm_arch_post_run
4. Auswerten des Abbruchgrundes und gegebenenfalls Aufruf eines Emulators

Nach der Abarbeitung eines Abbruchgrundes gibt es grundsätzlich drei unterschiedliche Vorgehensweisen. Erstens kann die Funktion durch Rücksprung an den Schleifenanfang (Zeile (Z) 15) erneut durchlaufen werden. Zweitens kann die Schleifenausführung beendet und die Funktion ohne eine Fehlerindikation beendet werden. Dies führt effektiv zu einem erneuten Aufruf durch die Funktion qemu_kvm_cpu_thread_fn. Drittens kann die Schleifenausführung beendet, die virtuelle Maschine gestoppt und die Funktion mit einer Fehlerindikation beendet werden. Dies führt effektiv zu einer sofortigen Beendigung des Gastes. Von dieser letzten Möglichkeit wird Gebrauch gemacht, sofern der Abbruchgrund KVM_EXIT_UNKNOWN ist (Z. 89) oder es aufgrund des Abbruchgrundes zum Aufruf der Funktion kvm_arch_handle_exit (Z. 115 und 121) kommt und diese einen negativen Wert zurückgibt. Durch das Verhalten der Funktion kvm_arch_handle_exit kommt es somit nicht nur im Fall eines Fehlers von KVM (KVM_EXIT_UNKNOWN), sondern auch im Fall eines nicht in QEMU kodierten Abbruchgrundes zu einer Beendigung des Gastes.

Der einzige Abbruchgrund, der von KVM auf der x86-Plattform definiert wird und der nicht durch QEMU explizit abgearbeitet wird, ist KVM_EXIT_HYPERCALL. Gäbe KVM diesen Abbruchgrund gegenüber QEMU an, so würde dies zu einer Beendigung des Gastes führen. Problematisch ist hierbei, dass dieser Abbruchgrund durch den Gast ausgelöst werden kann, wenn dieser einen Maschinenbefehl[61] aufruft, und dass dieser Befehl potenziell auch aus dem Userspace des Gastes aufgerufen werden kann. Somit besteht die Gefahr, dass ein Angreifer mit einfachen Benutzerrechten einen entsprechenden Maschinenbefehl aufruft und somit die sofortige Beendigung des Gastes herbeiführt. Der Grund, warum dies tatsächlich nicht möglich ist, besteht darin, dass KVM einen solchen Hypercall rein intern verarbeitet und auf einer x86-Plattform den entsprechenden Abbruchgrund nie an QEMU zurückgibt. Hier bleibt jedoch die Gefahr bestehen, dass sich das Verhalten von KVM in Zukunft so ändert, dass das oben beschriebene Angriffsszenario erlaubt würde. Daher empfiehlt es sich, den Abbruchgrund KVM_EXIT_HYPERCALL in QEMU zu berücksichtigen oder ihn aus der Schnittstelle von KVM zu entfernen.

Darüber hinaus arbeitet die Funktion memory_region_dispatch_write (s. u.) fehlerhaft.

```
136 MemTxResult memory_region_dispatch_write(MemoryRegion *mr,
137                                          hwaddr addr,
138                                          uint64_t data,
139                                          unsigned size,
140                                          MemTxAttrs attrs)
141 {
142     if (!memory_region_access_valid(mr, addr, size, true)) {
143         unassigned_mem_write(mr, addr, data, size);
144         return MEMTX_DECODE_ERROR;
145     }
146
147     adjust_endianness(mr, &data, size);
148
149     if (mr->ops->write) {
150         return access_with_adjusted_size(addr, &data, size,
151                                          mr->ops->impl.min_access_size,
152                                          mr->ops->impl.max_access_size,
153                                          memory_region_write_accessor, mr,
154                                          attrs);
155     } else if (mr->ops->write_with_attrs) {
156         return
157             access_with_adjusted_size(addr, &data, size,
158                                       mr->ops->impl.min_access_size,
159                                       mr->ops->impl.max_access_size,
160                                       memory_region_write_with_attrs_accessor,
```

61 Auf Hardware von Intel ist dies die Instruktion VMCALL, auf Hardware von AMD hingegen VMMCALL.

```
161                             mr, attrs);
162   } else {
163       return access_with_adjusted_size(addr, &data, size, 1, 4,
164                                 memory_region_oldmmio_write_accessor,
165                                 mr, attrs);
166   }
167}
```

Primärer Zweck der Funktion ist es, einen schreibenden Speicherzugriff des Gastes an den Emulator weiterzureichen. Da den Emulatoren verschiedene Schnittstellen zur Verfügung stehen, um den Speicherzugriff zu implementieren, muss die Funktion ausgewählt werden, die konkret aufgerufen werden soll. Diese Auswahl führt die Funktion memory_region_dispatch_write durch, indem sie sukzessive überprüft, ob der Emulator die jeweilige Schnittstelle implementiert hat. Dies erfolgt jedoch fehlerhaft: Die Funktion ruft für den Fall, dass der Emulator weder die Schnittstelle write noch write_with_attrs bereitstellt, die Schnittstelle oldmmio_write auf, ohne zu überprüfen, ob diese implementiert ist. Sind alle drei Funktionen nicht implementiert, z. B. weil der jeweilige Emulator keinen schreibenden Zugriff auf die jeweilige Speicherstelle vorsieht, so führt dies zu einem Zugriff auf einen NULL-Zeiger und somit zum Absturz von QEMU und somit auch des Gastes.

Die tatsächliche Gefahr kann jedoch als gering eingestuft werden, da in aller Regel angenommen werden darf, dass ein solcher Zugriff nicht aus dem Userspace des Gastes veranlasst werden kann. Ein Angreifer, der Code im Kernelspace ausführen kann, ist ohnehin imstande, die Funktionstüchtigkeit des Gastes zu stören, und kann daher keinen Vorteil aus diesem Fehler ziehen.

Die aufgezeigten Unzulänglichkeiten sind somit zwar potenziell sicherheitsrelevant, lassen sich aber unter realistischen Bedingungen nicht ausnutzen.

5.10 Zusammenfassung

Die wesentlichen Aspekte bezüglich der Widerstandsfähigkeit von QEMU lassen sich wie folgt zusammenfassen: Es existiert ein verbindlicher Prozess zur Einreichung von Quellcode in das QEMU-Projekt. Dieser Prozess stellt eine Reihe von Anforderungen an den Quelltext und schreibt dessen Begutachtung durch einen Projekt-Maintainer vor. Ein großer Teil dieser an den Quelltext gestellten Anforderungen haben den Zweck, sicherheitskritische Fehler zu vermeiden. Die hierzu aufgestellten, verbindlichen Regeln sind zum überwiegenden Teil sowohl praktikabel als auch zweckdienlich. Die automatisierte Überprüfung eines Teils dieser Anforderungen stellt ferner deren konsequente Anwendung sicher. Der vorhandene Prozess zur Aufnahme von Quellcode ist daher grundsätzlich geeignet, sicherheitskritische Implementierungsfehler zu vermeiden oder frühzeitig zu erkennen, und trägt daher signifikant zur Codequalität bei.

Ein weiterer Aspekt, der zur Codequalität von QEMU beiträgt, ist dessen modularer Aufbau. So sind die Bestandteile von QEMU konsequent in Form von Komponenten realisiert, die zumeist lediglich lose miteinander gekoppelt sind. Dies ermöglicht es, Komponenten durch alternative Komponenten auszutauschen oder gänzlich auf deren Verwendung zu verzichten. Hierdurch kann die Angriffsfläche minimiert werden. Darüber hinaus trägt die zweckdienliche Strukturierung des Quelltextes zu einer einfachen Les- und Wartbarkeit von QEMU bei.

Darüber hinaus ist die konsequente Unterstützung von Virtio nicht nur aus Gründen der Performance, sondern auch unter sicherheitstechnischen Gesichtspunkten positiv hervorzuheben. So vereinfacht die Wiederverwendung allgemeiner Virtio-Funktionen die Implementierung konkreter auf Virtio aufbauender Emulatoren und erhöht dadurch die Les- und Wartbarkeit des Codes. Darüber hinaus ist die Nachahmung realer Hardware nicht erforderlich, was regelmäßig zu einem einfacher strukturierten Datenaustausch zwischen Host und Gast führt.

Ferner besitzt QEMU eine dedizierte Schnittstelle zum Testen von Emulatoren. Diese ermöglicht es, neben dem Test der eigentlichen Funktion auch die Sicherheit von Emulatoren zu überprüfen. Tatsächlich wird

Letzteres jedoch nur in einem vernachlässigbar geringen Rahmen durchgeführt. Auch wenn somit ein großes Potenzial zur Überprüfung und Gewährleistung der Codequalität gegeben ist, so ist ein signifikanter Beitrag tatsächlich nicht erkennbar.

Im Rahmen von CentOS 7 werden des Weiteren sowohl zur Übersetzungs- als auch zur Laufzeit sinnvolle Härtungsmaßnahmen der Programmdatei ergriffen. Insbesondere die Nutzung von Stack Protection, des NX-Bits und von ASLR sowohl für QEMU selbst als auch mit wenigen Ausnahmen für die eingebundenen Softwarebibliotheken bietet einen soliden Schutz. Das Ausnutzen eines Programmfehlers wird somit für eine Reihe von Angriffsarten verhindert oder zumindest erheblich erschwert. Es ist somit positiv zu bewerten, dass gegebenenfalls vorhandene, potenziell sicherheitskritische Implementierungsfehler oftmals tatsächlich nicht ausnutzbar sind.

Darüber hinaus wird eine Reihe von Codeabschnitten, die für diese Studie relevant sind, von GCC und flawfinder beanstandet. Den überwiegenden Teil dieser Beanstandungen machen hierbei Konvertierungsvorgänge zwischen vorzeichenlosen und vorzeichenbehafteten Datentypen aus. Erfahrungsgemäß handelt es sich hierbei in aller Regel nicht um ausnutzbare Fehler, was auch durch die Überprüfung einzelner Befunde an besonders kritischen Codestellen bestätigt wird. Dennoch besteht bezüglich der beanstandeten Konvertierungsfehler zumindest ein potenzielles Sicherheitsrisiko. Aufgrund der großen Anzahl dieser Meldungen ist es im Rahmen dieser Untersuchung jedoch nicht möglich, diese in Gänze zu analysieren. Es verbleibt also eine gewisse Unsicherheit über den tatsächlichen Sachverhalt. Es kann jedoch sehr wohl festgestellt werden, dass diesbezüglich eine systematische Unzulänglichkeit und eine Einschränkung der Codequalität von QEMU vorliegt. Die Menge der darüber hinaus beanstandeten Codefragmente sind in Relation zum untersuchten Codeumfang gering. Stichprobenartig vorgenommene manuelle Überprüfungen lassen keine sicherheitskritischen Fehler erkennen.

Die vorgenommene manuelle Überprüfung der Anbindung von QEMU an KVM zeigt wenige Unzulänglichkeiten auf. Diese haben zwar ein sicherheitsrelevantes Potenzial, sind jedoch unter realistischen Bedingungen nicht ausnutzbar.

Abschließend lässt sich daher zusammenfassen, dass – abgesehen von wenigen Ausnahmen – keine grundlegenden Mängel festgestellt werden konnten. Es kann daher davon ausgegangen werden, dass QEMU den Anforderungen eines normalen Schutzbedarfs, wie er bei typischen, produktiv genutzten Virtualisierungsumgebungen anzutreffen ist, genügt. Sicherheitskritische Fehler können, trotz der aufgezeigten Codequalität und der getroffenen Härtungsmaßnahmen, jedoch nicht gänzlich ausgeschlossen werden. Bei einem Einsatz auf Systemen mit hohem Schutzbedarf ist daher eine weitergehende Absicherung, z. B. durch SELinux oder AppAmor, angeraten. Bei sehr hohem Schutzbedarf ist eine solche weitergehende Absicherung obligatorisch.

6 Sicherheitsanalyse von KVM

6.1 Zielsetzung

Gemäß der in Abschnitt 3.4 beschriebenen Architektur obliegt KVM die Durchführung verschiedener sicherheitskritischer Aufgaben. Dazu gehört die Initialisierung und Konfiguration der Hardware zur sicheren Ausführung von Gastcode. Eine weitere Aufgabe ist die Vermittlung zwischen Userspace-Programmen, wie z. B. QEMU, und der Hardware während der Ausführung des Gastes. Hierbei stellt KVM dem Userspace eine weitgehend plattformunabhängige Schnittstelle zur Verfügung. Die Anweisungen des Userspaces bezüglich der Virtualisierung müssen durch KVM daher für die konkrete Virtualisierungstechnik, wie z. B. VT-x oder AMD-V, aufbereitet und an diese weitergeleitet werden. Ferner müssen Laufzeitinformationen der Hardware, insbesondere Informationen zur Abbruchursache der Codeausführung, in einer plattformunabhängigen Weise an den Userspace übergeben werden.

Hierbei hat KVM stets eine ausreichende Isolation sowohl zwischen Host und Gast als auch zwischen Gästen untereinander sicherzustellen. Von wesentlicher Bedeutung für diese Isolation ist, dass der Gast die CPU nur eingeschränkt konfigurieren kann. Dies umfasst insbesondere die Control Register (CR), das Extended Feature Enable Register (EFER) sowie eine Reihe von maschinenspezifischen Registern.

Um eine solche unabhängige Konfiguration zu ermöglichen, sichern die Virtualisierungstechniken VT-x und AMD-V den Inhalt dieser Register bei einem Wechsel vom Gast zum Host und umgekehrt. So wird beim Abbruch der Ausführung eines Gastes beispielsweise der Inhalt von CR0 in einen dafür vorgesehenen Speicher geschrieben. Danach wird der Zustand des CR0 zurückgeschrieben, der vor der Ausführung des Gastes gesichert wurde. Somit befindet sich CR0 in dem gleichen Zustand wie vor dem Beginn der Ausführung des Gastes. Der Zustand des Hosts wurde auf diese Weise rekonstruiert. Bei erneutem Wechsel in den Gast wird dieses Verfahren analog wiederholt, sodass der gespeicherte Zustand des Gastes wiederhergestellt wird. Diese durch VT-x bzw. AMD-V implementierten Verfahren stellen eine weitgehende Trennung des Host- und Gastzustandes sicher. Die Durchführung obliegt hierbei der Hardware.

Dieses Verfahren hat jedoch Grenzen. So ist es eine wesentliche Aufgabe des Hypervisors, sicherzustellen, dass der Gast während seiner Ausführung die CPU nicht in einen illegitimen Zustand versetzt. Beispielsweise darf es dem Gast nicht möglich sein, die Virtualisierung zu deaktivieren oder auf Speicherbereiche zuzugreifen, die ihm nicht zugewiesen wurden. Es ist die Aufgabe des Hypervisors, dies sowohl durch eine entsprechende Initialisierung der Hardware als auch durch Kontrollmaßnahmen zur Laufzeit sicherzustellen.

Daher wird im Folgenden analysiert, wie KVM die Hardware bei Verwendung von VT-x und AMD-V konfiguriert. Dies umfasst sowohl die Initialisierung als auch die Konfiguration zur Laufzeit des Gastes. Die hierzu durchgeführte Quellcodeanalyse soll aufzeigen, ob eine effektive Isolation gegeben ist.

Von der Analyse ausgeschlossen ist jedoch die Unterstützung für eine verschachtelte Virtualisierung. Ferner werden Funktionen des Kernels, die nicht ausschließlich oder hauptsächlich zur Virtualisierung genutzt werden (wie z. B. das Scheduling und allgemeine Bestandteile der Speicherverwaltung), nicht betrachtet. Ebenso sind Bestandteile zum Debuggen sowie zum Tracen ausgeschlossen. Auch wird die HyperV-Kompatibilitätsschicht von KVM (s. [KVMHV]) nicht betrachtet. Es wird ferner angenommen, dass die Hardware[62] selbst vertrauenswürdig ist und sich spezifikationsgetreu verhält.

Grundlage für die Untersuchung ist der Linux-Kernel in Version 4.7.6 in unveränderter Form („Vanilla“).

62 Dies schließt den auf der CPU ausgeführten Microcode mit ein.

6.2 Sicherheitskritische Modulparameter

6.2.1 Allgemeines

Die zur Virtualisierung verwendeten Kernelmodule lassen sich teilweise parametrisieren und somit in ihrem Verhalten beeinflussen. Ein Großteil dieser Parameter dienen zur Performanceoptimierung und sind von einem sicherheitstechnischen Standpunkt aus weitestgehend unbedenklich. Aus diesem Grund wird auf eine vollständige Auflistung verzichtet. Parameter mit sicherheitsrelevanter Bedeutung werden im Folgenden mit kurzer Erläuterung aufgeführt. Zu beachten ist, dass die Parameterbelegung durch die jeweilige Linux-Distribution angepasst wird und somit von der aufgeführten Standardbelegung abweichen kann.

6.2.2 nested

Der Parameter nested bestimmt, ob dem Gast selbst Virtualisierungsfunktionen zur Verfügung gestellt werden sollen. Hierbei ist in dem Kernelmodul kvm_intel die Unterstützung standardmäßig deaktiviert, während das Modul kvm_amd diese hingegen standardmäßig aktiviert. Sofern eine solche Funktion vom Gast nicht benötigt wird, sollte die verschachtelte Virtualisierung deaktiviert werden, um die Angriffsfläche aus dem Gast heraus zu minimieren.[63]

6.2.3 ept

Ist der Parameter ept des Kernelmoduls kvm_intel gesetzt, so nutzt KVM eine verschachtelte Speicherverwaltung (s. Abschnitt 3.3.2), sofern dies von der Hardware unterstützt wird. Dies führt regelmäßig zu einer Performancesteigerung und hat zudem sicherheitstechnische Vorteile gegenüber der Emulation der MMU. So ist deren Emulation keineswegs trivial und bedingt eine komplexe Verarbeitung von Daten, die der Gast zur Verfügung stellt (s. Abschnitt 3.3.2). Der Parameter sollte daher stets auf true gesetzt sein, um den Umfang des tatsächlich genutzten Codes zu reduzieren und somit die Angriffsfläche zu verringern. EPT ist standardmäßig aktiviert.

6.2.4 npt

Der Parameter npt des Moduls kvm_amd entspricht dem Parameter ept des Moduls kvm_intel (s. Abschnitt 6.2.3). Entsprechende Anmerkungen gelten analog.

6.2.5 allow_unsafe_assigned_interrupts

Der Parameter allow_unsafe_assigned_interrupts des Kernelmoduls vfio_iommu ermöglicht die Durchreichung von Hardware auch dann, wenn die IOMMU kein Interrupt Remapping unterstützt. Da eine Durchreichung ohne Unterstützung von Interrupt Remapping ein signifikantes Sicherheitsrisiko bedeutet (s. Abschnitt 6.6), sollte der Parameter auf false gesetzt sein, was die Standardbelegung ist.

Zu beachten ist, dass die Durchreichung von Hardware scheitert, wenn die Hardware ein entsprechendes Remapping nicht unterstützt und der Parameter auf false gesetzt ist. Die in diesem Zuge vom Kernel erzeugte Fehlermeldung fordert dazu auf, den Parameter auf true zu setzen, ohne jedoch auf die damit verbundenen Risiken hinzuweisen.

63 Die verschachtelte Virtualisierung wird im Rahmen dieser Untersuchung nicht betrachtet (s. Abschnitt 3.1).

6.3 Herstellerunabhängige Eigenschaften

6.3.1 Zugriff auf maschinenspezifische Register

6.3.1.1 Lesender Zugriff

Die Funktion kvm_get_msr_common bietet eine allgemeine Behandlung von lesenden Zugriffen auf MSRs. Sie ermittelt anhand der vom Gast spezifizierten Adresse eines MSRs den Wert, der an diesen Gast zurückgegeben werden muss. Ausgeführt wird sie von den Rückruffunktionen des Maschinenbefehls RDMSR der Implementierungen von VT-x und AMD-V. Die Funktion kvm_get_msr_common realisiert somit die Verarbeitung allgemeiner MSRs der x86-Architektur. MSRs, die spezifisch für CPUs von Intel oder AMD sind, werden zusätzlich in der jeweiligen Implementierung umgesetzt.[64] Im Folgenden wird untersucht, ob ausnutzbare Implementierungsfehler vorliegen. Hierbei wird jedoch nicht überprüft, ob die zum Gast zurückgegebenen Werte sinnvoll und spezifikationsgetreu sind.

Die Komplexität der Ermittlung des zurückzugebenden Wertes ist von dem jeweils abgerufenen MSR abhängig. Im einfachsten Fall wird ein konstanter Wert zurückgegeben. Tabelle 15 listet alle MSRs auf, für die stets ein konstanter Wert zurückgegeben wird. Deren Implementierung ist trivial und weist keine Auffälligkeiten auf.

MSR	Wert
IA32_PLATFORM_ID	0
IA32_EBL_CR_POWERON	0
IA32_DEBUGCTLMSR	0
IA32_LASTBRANCHFROMIP,	0
IA32_LASTBRANCHTOIP	0
IA32_LASTINTFROMIP	0
IA32_LASTINTTOIP	0
K8_SYSCFG	0
K8_TSEG_ADDR	0
K8_TSEG_MASK	0
K7_HWCR	0
VM_HSAVE_PA	0
K8_INT_PENDING_MSG	0
AMD64_NB_CFG	0
FAM10H_MMIO_CONF_BASE	0
AMD64_BU_CFG2	0
IA32_PERF_CTL	0
IA32_UCODE_REV	0x100000000.
MTRRcap	0x508

64 Zugriffe auf MSRs, die grundsätzlich nicht abgefangen werden (s. Abschnitt 6.4.1.8 und 6.5.1.3), werden naheliegenderweise nicht durch die Funktion *kvm_get_msr_common* verarbeitet.

MSR	Wert
0xcd	3
EBC_FREQUENCY_ID	0x1000000
IA32_PERF_STATUS	0x40000001000
K7_CLK_CTL	0x20000000
IA32_BBL_CR_CTL3	0xbe702111

Tabelle 15: Konstante MSRs

Darüber hinaus existieren MSRs, deren Wert aus einem Verwaltungsspeicher von KVM ausgelesen wird und an den Gast zurückgegeben wird. Auch wenn diese Werte somit nicht konstant sein müssen, so ist ihre Implementierung ebenfalls trivial. Entsprechende MSRs sind in Tabelle 16 aufgeführt.[65] Keine der Implementierungen der in Tabelle 16 aufgeführten MSRs weist Auffälligkeiten auf.

MSR	Implementierung
IA32_TSC_ADJUST	vcpu->arch.ia32_tsc_adjust_msr
IA32_MISC_ENABLE	vcpu->arch.ia32_misc_enable_msr
IA32_SMBASE	vcpu->arch.smbase
EFER	vcpu->arch.efer
KVM_WALL_CLOCK, KVM_WALL_CLOCK_NEW	vcpu->kvm->arch.wall_clock
KVM_SYSTEM_TIME	vcpu->arch.time
KVM_SYSTEM_TIME_NEW	vcpu->arch.time
KVM_ASYNC_PF_EN	vcpu->arch.apf.msr_val
KVM_STEAL_TIME	vcpu->arch.st.msr_val
KVM_PV_EOI_EN	vcpu->arch.pv_eoi.msr_val
AMD64_OSVW_ID_LENGTH	vcpu->arch.osvw.length
AMD64_OSVW_STATUS	vcpu->arch.osvw.status

Tabelle 16: Allgemeine MSRs mit trivialer Implementierung des Lesezugriffs

Darüber hinausgehend existieren MSRs, deren Implementierung komplexer ist und deshalb in eigene Funktionen ausgelagert wird. Diese MSRs sind in Tabelle 17 dargestellt.

MSR	Wert	Kommentar
0x200 ... 0x2ff	kvm_mtrr_get_msr	Die Funktion kvm_mtrr_get_msr ermittelt den Rückgabewert anhand der MSR-Adresse und der Struktur vcpu->arch.mtrr_state. Keine Auffälligkeiten feststellbar.
IA32_APICBASE	kvm_get_apic_base	Die Funktion kvm_get_apic_base gibt schlicht den Wert vcpu->arch.apic_base zurück. Keine Auffälligkeiten feststellbar.
APIC_BASE bis APIC_BASE + 0x3ff	kvm_x2apic_msr_read	Die Funktion kvm_x2apic_msr_read

65 Es werden auch MSRs aufgelistet, die zusätzlich eine triviale Plausibilitätsüberprüfung durchführen.

MSR	Wert	Kommentar
		führt zu einem Rücksprung in den Userspace, sofern APIC nicht durch KVM emuliert wird.
IA32_TSCDEADLINE	kvm_get_lapic_tscdeadline_msr	Die Funktion kvm_get_lapic_tscdeadline_msr gibt den Wert apic->lapic_timer.tscdeadline zurück, sofern APIC im Kernel emuliert wird und entsprechend konfiguriert ist. Andernfalls wird 0 zurückgegeben. Keine Auffälligkeiten feststellbar.
K7_EVNTSEL0 bis K7_EVNTSEL3, K7_PERFCTR0 bis K7_PERFCTR3, P6_PERFCTR0 bis P6_PERFCTR1, P6_EVNTSEL0 bis P6_EVNTSEL1,	kvm_pmu_get_msr	Keine Auffälligkeiten feststellbar (s. unten).
IA32_P5_MC_ADDR, IA32_P5_MC_TYPE, IA32_MCG_CAP, IA32_MCG_CTL, IA32_MCG_STATUS, IA32_MC0_CTL bis IA32_MC31_CTL	get_msr_mce	Die Funktion get_msr_mce liest den zurückzugebenden Wert aus einer dafür vorgesehenen internen Datenstruktur. Keine Auffälligkeiten feststellbar.
HV_X64_MSR_GUEST_OS_ID bis HV_X64_MSR_SINT15, HV_X64_MSR_CRASH_P0 bis HV_X64_MSR_CRASH_P4, HV_X64_MSR_STIMER0_CONFIG bis HV_X64_MSR_STIMER3_COUNT, HV_X64_MSR_CRASH_CTL	kvm_hv_get_msr_common	Die Funktion kvm_hv_get_msr_common ist Bestandteil der Hyper-V-Emulation. Die Implementierung wird daher nicht untersucht. Zu beanstanden ist jedoch, dass die entsprechende Implementierung immer durch den Gast aufgerufen werden kann. Sofern die Hyper-V-Emulation nicht verwendet werden soll, vergrößert dies unnötigerweise die Angriffsfläche.

Tabelle 17: Allgemeine MSRs mit komplexer Implementierung des Lesezugriffs

Greift der Gast auf ein nicht aufgeführtes MSR zu, so wird mit der Funktion kvm_pmu_is_valid_msr überprüft, ob das MSR zur Performance Management Unit (PMU) gehört. Ist dies der Fall, so wird sein Wert durch die Funktion kvm_pmu_get_msr bestimmt. Letztere ruft, je nach CPU-Typ, die Funktion intel_pmu_get_msr oder amd_pmu_get_msr auf. Beide Implementierungen erzeugen den Rückgabewert auf Basis von Performancedaten, die der Linux-Kernel gesammelt hat. Auch wenn beide Implementierungen keine Auffälligkeiten zeigen, so ist zu bemängeln, dass sich dieser Zugriff auf die Performancedaten nicht deaktivieren lässt. Dies wäre wünschenswert, da die Implementierung nicht trivial ist und somit prinzipiell eine erhebliche Angriffsfläche bietet. Des Weiteren lassen sich mit der PMU präzise Performancedaten ermitteln, unter anderem bezüglich der CPU-Caches. Es kann nicht ausgeschlossen werden, dass somit versteckte Kommunikationskanäle (s. Abschnitt 6.7) oder Seitenkanalangriffe (s. Abschnitt 6.8) realisierbar sind.[66]

66 Einer Deaktivierung könnten allerdings Anforderungen des Gastes entgegenstehen. Zwar darf dieser die Unterstützung von PMU nur voraussetzen, wenn dies in dem entsprechenden CPUID-Register vermerkt ist (s.

Ist das vom Gast angeforderte MSR auch nicht zur PMU gehörig, so kennt KVM dieses MSR nicht. In diesem Fall wird überprüft, ob der Kernelparameter ignore_msrs auf true gesetzt ist. Ist dies der Fall, so wird der Wert 0 an den Gast zurückgegeben. Andernfalls wird innerhalb des Gastes eine Exception erzeugt. In beiden Fällen wird eine Warnmeldung erzeugt.

Ausnutzbare Schwächen sind nicht feststellbar. Zu kritisieren ist jedoch, dass einzelne MSRs und deren zugehörige Implementierung nicht deaktiviert werden können. Dies trifft insbesondere auf die MSRs zu, die zur PMU und zur Hyper-V-Emulation gehören.

6.3.1.2 Schreibender Zugriff

Eine allgemeine Behandlung von schreibenden Zugriffen auf MSRs bietet die Funktion kvm_set_msr_common. Sie ermittelt anhand der vom Gast spezifizierten Adresse das MSR, in das geschrieben werden soll. Neben dieser Adresse übergibt der Gast den zu schreibenden Wert. Ausgeführt wird die Funktion von den Rückruffunktionen des Maschinenbefehls WRMSR, der in den Implementierungen von VT-x und AMD-V enthalten ist. Die Funktion kvm_set_msr_common realisiert somit die Verarbeitung allgemeiner MSRs der x86-Architektur. MSRs, die spezifisch für CPUs von Intel oder AMD sind, werden zusätzlich in der jeweiligen Implementierung umgesetzt.[67] Im Folgenden wird untersucht, ob ausnutzbare Implementierungsfehler vorliegen. Es wird jedoch nicht untersucht, ob die Emulation der damit verbundenen Funktionen Spezifikationen verletzt.

Da die Modifikation eines MSRs nicht nur das Abspeichern eines Wertes umfassen kann, sondern teilweise komplexe Operationen nach sich zieht, sind die Implementierungen für verschiedene MSRs unterschiedlich komplex. Die folgende Auflistung führt MSRs auf, in die der Gast schreiben kann, bei denen jedoch KVM den übergebenen Wert verwirft.[68] Die zugehörige Implementierung ist somit trivial und nicht zu beanstanden.

- AMD64_NB_CFG
- IA32_UCODE_REV
- IA32_UCODE_WRITE
- VM_HSAVE_PA
- AMD64_PATCH_LOADER
- AMD64_BU_CFG2
- K7_CLK_CTL
- K7_HWCR
- FAM10H_MMIO_CONF_BASE

Für eine Reihe weiterer MSRs wird der vom Gast übergebene Wert von kvm_set_msr_common lediglich gespeichert, ohne dass sich unmittelbar eine Verarbeitung anschließt.[69] Entsprechende MSRs sind in Tabelle 20 aufgeführt. Ihre Implementierung ist trivial und nicht zu beanstanden.

S. 537 in [INTEL2]). Ob jedoch alle relevanten Betriebssysteme eine solche Überprüfung durchführen, ist fraglich.

67 Zugriffe auf MSRs, die grundsätzlich nicht abgefangen werden (s. Abschnitt 6.4.1.8 und 6.5.1.3), werden naheliegenderweise nicht durch die Funktion *kvm_set_msr_common* verarbeitet.

68 Bei einigen der aufgeführten MSRs wird eine Überprüfung des übergebenen Datums durchgeführt. Verläuft diese negativ, so wird eine Exception im Gast erzeugt. Eine Speicherung des Datums findet aber in keinem Fall statt.

69 Hier kann jedoch nicht ausgeschlossen werden, dass eine Verarbeitung des übergebenen Datums zu einem späteren Zeitpunkt erfolgt.

MSR	Implementierung
IA32_MISC_ENABLE	vcpu->arch.ia32_misc_enable_msr
A32_SMBASE	vcpu->arch.smbase
AMD64_OSVW_ID_LENGTH	vcpu->arch.osvw.length
AMD64_OSVW_STATUS	vcpu->arch.osvw.status

Tabelle 18: Allgemeine MSRs mit trivialer Implementierung für Schreibzugriffe

Darüber hinaus existieren MSRs, deren Implementierung eines schreibenden Zugriffs komplexer ist. Die entsprechenden MSRs werden in Tabelle 19 aufgeführt.

MSR	Funktion	Kommentar
EFER	set_efer	s. Abschnitt 6.3.2
0x200 bis 0x2ff	kvm_mtrr_set_msr	Keine Auffälligkeiten feststellbar.
IA32_APICBASE	kvm_set_apic_base	Keine Auffälligkeiten feststellbar.
APIC_BASE bis APIC_BASE + 0x3ff	kvm_x2apic_msr_write	
IA32_TSCDEADLINE	kvm_set_lapic_tscdeadline_msr	Die Funktion kvm_set_lapic_tscdeadline_msr startet einen nicht periodischen Timer. Keine Auffälligkeiten feststellbar.
IA32_TSC_ADJUST	adjust_tsc_offset_guest	Das übergebene Datum wird mit der Funktion adjust_tsc_offset_guest zum TSC-Offset addiert, sofern dem Gast dies erlaubt ist (X86_FEATURE_TSC_ADJUST). Andernfalls wird eine Exception ausgelöst. Keine Auffälligkeiten feststellbar.
KVM_WALL_CLOCK, KVM_WALL_CLOCK_NEW	kvm_write_wall_clock	Die Funktion kvm_write_wall_clock dient zur Messung der tatsächlich verstrichenen Zeit (s. [KVMMSR]). Keine Auffälligkeiten feststellbar.
KVM_SYSTEM_TIME KVM_SYSTEM_TIME_NEW		Übergibt dem Gast die tatsächliche Zeit (s. [KVMMSR]). Keine Auffälligkeiten feststellbar.
KVM_ASYNC_PF_EN	kvm_pv_enable_async_pf	Keine Auffälligkeiten feststellbar (Details s [KVMMSR]).
KVM_STEAL_TIME		Übergibt eine Angabe bezüglich der Dauer, während der der Gast nicht aktiv war (s.[KVMMSR]). Keine Auffälligkeiten feststellbar.
KVM_PV_EOI_EN	kvm_lapic_enable_pv_eoi	Keine Auffälligkeiten feststellbar (Details s [KVMMSR]).
IA32_MCG_CTL,	set_msr_mce	Die Funktion set_msr_mce

MSR	Funktion	Kommentar
IA32_MCG_STATUS, IA32_MC0_CTL bis IA32_MC31_CTL		speichert das übergebene Datum in entsprechenden Datenstrukturen. Keine Auffälligkeiten feststellbar.
MSR_K7_PERFCTR0 bis MSR_K7_PERFCTR3, MSR_P6_PERFCTR0 bis MSR_P6_PERFCTR1, MSR_K7_EVNTSEL0 bis MSR_K7_EVNTSEL3, MSR_P6_EVNTSEL0 bis MSR_P6_EVNTSEL1	kvm_pmu_set_msr	Keine Auffälligkeiten feststellbar (s. unten).
HV_X64_MSR_GUEST_OS_ID bis HV_X64_MSR_SINT15, HV_X64_MSR_CRASH_P0 bis HV_X64_MSR_CRASH_P4, HV_X64_MSR_CRASH_CTL, HV_X64_MSR_STIMER0_CONFIG bis HV_X64_MSR_STIMER3_COUNT	kvm_hv_set_msr_common	Bestandteil der Hyper-V-Emulation. Die Implementierung wird daher nicht untersucht. Zu beanstanden ist jedoch, dass die entsprechende Implementierung immer durch den Gast aufgerufen werden kann. Sofern die Hyper-V-Emulation nicht verwendet werden soll, vergrößert dies unnötigerweise die Angriffsfläche.

Tabelle 19: Allgemeine MSRs mit komplexer Implementierung für Schreibzugriffe

Wird versucht, in ein MSR zu schreiben, das nicht aufgeführt ist, so wird überprüft, ob das MSR zur PMU gehört. Ist dies der Fall, so wird die Funktion kvm_pmu_set_msr aufgerufen.

Diese ruft, je nach CPU-Typ, die Funktion intel_pmu_set_msr oder amd_pmu_set_msr auf. Beide Implementierungen konfigurieren die PMU-Zähler auf Basis des vom Gast übergebenen Datums. Wenngleich beide Implementierungen keine Auffälligkeiten zeigen, so ist zu bemängeln, dass sich der Zugriff, analog zum lesenden Zugriff auf die entsprechenden MSRs, nicht deaktivieren lässt. Dies wäre wünschenswert, da die Implementierung nicht trivial ist und somit prinzipiell eine nicht unerhebliche Angriffsfläche bietet.[70]

Wenn das vom Gast angeforderte MSR nicht zur PMU gehört, so ist es KVM nicht bekannt. In diesem Fall wird überprüft, ob der Kernelparameter ignore_msrs auf true gesetzt ist. Ist dies der Fall, so wird ohne weitere Verarbeitung in den Gast zurückgesprungen. Anderenfalls wird innerhalb des Gastes eine Exception erzeugt. In beiden Fällen wird eine Warnmeldung erzeugt.

Ausnutzbare Schwächen sind nicht feststellbar. Wie bei den lesenden Zugriffen auf die MSRs ist jedoch zu kritisieren, dass einzelne MSRs und die zugehörige Implementierung nicht deaktiviert werden können. Dies trifft insbesondere auf die MSRs zu, die zur PMU und zur Hyper-V-Emulation gehören.

6.3.2 Modifikation des Extended Feature Enable Registers

Die Modifikation des Extended Feature Enable Registers (EFER, s. Abschnitt 2.2.1 in [INTEL3A]) erfolgt mit der Funktion set_efer, die durch kvm_set_msr_common (s. Abschnitt 6.3.1.2) aufgerufen wird. Diese ruft je

70 Einer Deaktivierung könnten allerdings Anforderungen des Gastes entgegenstehen. Zwar darf der Gast das Vorhandensein einer PMU nur dann voraussetzen, wenn dies in dem entsprechenden CPUID-Register vermerkt ist (s. S. 537 in [INTEL2]). Ob jedoch alle relevanten Betriebssysteme eine solche Überprüfung durchführen, ist fraglich.

nach verwendeter CPU entweder die Funktion vmx_set_efer oder svm_set_efer auf. Bevor das vom Gast übergebene Datum in das Register EFER übertragen wird, führen set_efer bzw. vmx_set_efer eine Einschränkung des Datums durch. Tabelle 20 führt die Bits des Registers auf und gibt an, wann die Bits durch den Gast modifiziert werden können.[71] Alle weiteren (reservierten) Bits des Registers kann der Gast nicht abändern.

Bit	VT-x	AMD-V	Kommentar
System Call Extensions (SCE)	Ja	Ja	Die Nutzung von schnellen System Calls hat nur gastinterne Auswirkungen.
Long Mode Enable (LME)	Ja, sofern das Bit LMA auf dem Host gesetzt ist.	Ja, sofern das Bit LMA auf dem Host gesetzt ist oder die verschachtelte Speicherverwaltung aktiviert ist.	Sofern Paging aktiv ist, d. h. sofern das Bit PG in CR0 gesetzt ist, muss das LME-Bit vom Gast gesetzt sein (s. auch Abschnitt 4.1.2 in [INTEL3A]).
Long Mode Active (LMA)	Nein	Nein	Das Bit LMA ist kann nur gelesen werden. Ein Änderungsversuch durch den Gast wird ignoriert.
No-Execute Enable (NXE)	Ja	Ja	Die Möglichkeit, den Schreibschutz auf Speicherseiten zu entfernen, hat lediglich gastinterne Auswirkungen.
Secure Virtual Machine Enable (SVME)	Nein (Das Bit ist AMD-spezifisch)	Nein	Eine Änderung unter AMD-V wird in eine Schattenkopie des Registers EFER übernommen, sofern die verschachtelte Virtualisierung aktiviert ist. Somit kann dem Gast die Kontrolle über das Bit vorgetäuscht werden. Ansonsten führt eine Änderungsanfrage zu einer Exception im Gast.
Long Mode Segment Limit Enable (LMSLE)	Nein (Das Bit ist AMD-spezifisch.)	Nein	Der Kommentar zum Bit SVME gilt analog.
Fast FXSAVE/FXRSTOR (FFXSR)	Nein (Das Bit ist AMD-spezifisch.)	Ja, sofern Fast FXSAVE/FXRSTOR (s. S. 57 in [AMD2]) für den Gast aktiviert ist.	Die Nutzung von Fast FXSAVE/FXRSTOR hat lediglich gastinterne Auswirkungen.
Translation Cache Extension (TCE)	Nein	Nein	Das Bit TCE wird von KVM nicht unterstützt.

Tabelle 20: Mögliche Manipulationen des Registers EFER durch den Gast

Zusammenfassend lässt sich festhalten, dass die Möglichkeiten für den Gast, das Register EFER zu manipulieren, ausreichend beschränkt sind. Der Gast hat lediglich Zugriff auf die unbedenklichen Bits SCE, NXE und TCE. Er kann weder die Virtualisierung deaktivieren noch eine unsichere Änderung am Ausführungsmodus der CPU durchführen oder deren Cachingverhalten ändern.

71 Die Angaben stehen unter dem Vorbehalt, dass die physische CPU das entsprechende Bit unterstützt. Bei modernen CPUs kann dies als gegeben betrachtet werden.

6.4 VT-x

6.4.1 Initialisierung des Gastes

6.4.1.1 Pin-Based VM-Execution Controls

Das Register Pin-Based VM-Execution Controls steuert den Umgang mit Interrupts, die während der Ausführung von Code auf einer vCPU auftreten (s. Abschnitt 24.6.1 in [INTEL3C]). KVM setzt hierzu stets die Bits External-interrupt exiting und NMI exiting. Dies hat zur Folge, dass sowohl maskierbare als auch nicht maskierbare Interrupts zu einer Unterbrechung der Gastausführung führen. Von der Hardware erzeugte Interrupts werden daher an den Host und nicht an den Gast geliefert und sind für diesen nicht sichtbar. Von dieser Regel sind nur solche Interrupts ausgenommen, die mit APICv umgeleitet werden (s. Abschnitt 29.1 in [INTEL3C]). Sofern dies von der verwendeten Hardware unterstützt wird, werden ferner auch die Bits Virtual NMIs und Process posted interrupts gesetzt. Das hiermit verbundene Verhalten der vCPU ermöglicht ein performanteres Injizieren von Interrupts in den Gast. Für den Fall, dass APICv aktiviert ist, wird das Bit Process posted interrupts zweckmäßigerweise nicht gesetzt. Externe Interrupts werden daher stets abgefangen und vom Host anstatt vom Gast verarbeitet. Ausgenommen hiervon sind Interrupts von durchgereichter Hardware, sofern APICv aktiviert ist. Das Bit Activate VMX preemption timer wird in keinem Fall gesetzt.

6.4.1.2 Primary Process-Based VM-Execution Controls

Mit den Process-Based VM-Execution Controls (s. Abschnitt 24.6.2 in [INTEL3C]) werden wesentliche Aspekte der Virtualisierung konfiguriert. Insbesondere nutzt man sie zur Angabe der Maschinenbefehle, die abgefangen werden sollen. Hierzu gehören die Primary Process-Based VM-Execution Controls und die Secondary Process-Based VM-Execution Controls (s. Abschnitt 6.4.1.3).

Bei Verwendung einer SoftMMU (s. Abschnitt 3.3.2) setzt KVM die Bits CR3-load exiting, CR3-store exiting und INVLPG exiting und verhindert somit eine unkontrollierte Konfiguration der MMU. Falls EPT verwendet wird (s. Abschnitt 3.3.2), setzt KVM diese Bits nicht und ermöglicht es dem Gast, unkontrolliert auf CR3 zuzugreifen und den TLB mit dem Maschinenbefehl INVLPG zu leeren. Bei aktiviertem EPT ist es sinnvoll und unbedenklich, diese Bits nicht zu setzen.

KVM verhindert ferner, dass der Gast die Interrupt-Prioritäten verändert, indem er die Bits CR8-load exiting und CR8-store exiting setzt. Sofern die Hardware dies unterstützt, wird ferner das TPR-Shadowing (s. Abschnitt 29.3 in [INTEL3C]) aktiviert, indem das Bit Use TPR shadow gesetzt wird. In diesem Fall werden die Bits CR8-load exiting und CR8-store exiting nicht gesetzt und dem Gast somit ein unkontrollierter Zugriff auf CR8 gewährt. Letzteres ist sinnvoll und unbedenklich.

Darüber hinaus setzt KVM die Bits HLT exiting, RDPMC exiting, MWAIT exiting und MONITOR exiting. Hierdurch wird eine unkontrollierte Ausführung der Maschinenbefehle HLT, RDPMC, MWAIT und MONITOR unterbunden.

Allerdings setzt KVM das Bit RDTSC exiting nicht. KVM erlaubt es dem Gast somit, die Maschinenbefehle RDTSC und RDTSCP auszuführen und ermöglicht es ihm dadurch, Zeitstempel ohne Unterbrechung durch KVM erstellen zu lassen. Somit ist eine präzise logische Zeitmessung innerhalb des Gastes möglich. Hierdurch kann insbesondere ein Entropie-Pool zur Erzeugung von Zufallszahlen aufgebaut werden (s. [VMRNG]). KVM setzt ferner das Bit Use TSC offsetting und ist somit in der Lage, den Zeitstempel um einen

wählbaren Betrag zu verschieben.[72] Eine negative Beeinflussung des Entropie-Pools entsteht laut [VMRNG] hierdurch nicht.

Ferner unterbindet KVM den unkontrollierten Zugriff auf die Debug-Register der CPU, indem es das Bit MOV-DR exiting setzt. KVM ermöglicht es jedoch, den Gast in einem Debug-Modus auszuführen, der zum Nichtsetzen dieses Bits führt. Der damit verbundene unkontrollierte Zugriff auf die Debug-Register kann unter Umständen sicherheitskritisch sein. Es ist daher festzuhalten, dass die Nutzung des Debug-Modus im produktiven Betrieb nicht nur aus Performance-, sondern auch aus Sicherheitsgründen unterbleiben sollte.

KVM setzt außerdem das Bit Use I/O bitmaps, nicht jedoch das Bit Unconditional I/O exiting. Hierdurch wird erreicht, dass der Zugriff auf den IO-Adressraum nur für ausgewählte IO-Adressen abgefangen wird (s. Abschnitt 6.4.1.5).

Ferner setzt KVM das Bit Use MSR bitmaps, sofern dies von der Hardware unterstützt wird. Somit wird der Zugriff auf maschinenspezifische Register nicht per se unterbunden. Stattdessen wird ein unkontrollierter Zugriff auf MSRs selektiv durch die MSR-Bitmap (s. Abschnitt 6.4.1.8) gestattet. Wird ein selektives Abfangen entsprechender Zugriffe von der Hardware nicht unterstützt, so werden alle Zugriffe auf MSRs abgefangen.

Das ausschließlich zu Debug-Zwecken verwendete Bit Monitor trap flag (s. Abschnitt 25.5.2 in [INTEL3C]) wird von KVM nicht gesetzt. Ebenso wird von der Möglichkeit, den Maschinenbefehl PAUSE abzufangen, kein Gebrauch gemacht, indem das Bit PAUSE exiting nicht gesetzt wird.[73] Dies ist sicherheitstechnisch unbedenklich.

Darüber hinaus werden die Bits Interrupt-window exiting und NMI-window exiting, die zur Injektion von Interrupts in den Gast dienen, sinnvollerweise bei der Initialisierung nicht gesetzt.

Abschließend werden durch das Setzen des Bits Activate secondary controls die Secondary Process-Based VM-Execution Controls aktiviert, sofern dies von der Hardware unterstützt wird.

6.4.1.3 Secondary Process-Based VM-Execution Controls

Die Secondary Process-Based VM-Execution Controls erweitern die Konfigurationsmöglichkeiten der Primary Process-Based VM-Execution Controls (s. Abschnitt 6.4.1.2).

KVM setzt weder voraus, dass das Register in Gänze, noch dass einzelne Konfigurationsbits von der Hardware unterstützt werden. Die nachfolgend aufgeführten Konfigurationsoptionen werden von KVM daher nur dann gesetzt, wenn eine entsprechende Hardwareunterstützung gegeben ist.

KVM aktiviert die verschachtelte Speicherverwaltung (s. Abschnitt 3.3.2) durch das Setzen des Bits Enable EPT, sofern der Kernelparameter ept (s. Abschnitt 6.2.3) auf den Wert true gesetzt ist. Dies ist standardmäßig der Fall. Die Nutzung von EPT vereinfacht die Speicherverwaltung für KVM erheblich, da die komplexe Erstellung und Pflege einer SoftMMU entfällt. Die standardmäßige Nutzung der verschachtelten Speicherverwaltung erhöht damit nicht nur die Performance, sondern trägt auch zur Erhöhung der Sicherheit bei, weil sich der genutzte Codeumfang reduziert.

Ferner lässt KVM einen virtuellen Interrupt-Controller durch die Hardware emulieren. Hierzu setzt KVM die Bits Virtualize APIC accesses, Virtualize x2APIC mode, APIC-register virtualization und Virtual-interrupt delivery. Das erste Bit sorgt dafür, dass Zugriffe auf die virtuellen APIC-Speicherseiten (s. Abschnitt 29.4 in [INTEL3C]) abgefangen werden. Das zweite Bit veranlasst die Hardware, den Interrupt-Controller im Modus x2APIC (s. Abschnitt 10.12 in [INTEL3A]) zu emulieren. Durch das Setzen des dritten Konfigurationsbits fängt KVM Zugriffe auf die zu x2APIC gehörenden maschinenspezifische Register ab oder leitet diese auf Kopieren um. Das vierte Bit ermöglicht es KVM, virtuelle Interrupts ohne eine Ausführungsunterbrechung in den Gast zu injizieren (s. Abschnitt 29.2 in [INTEL3C]). Hierbei setzt KVM die genannten Bits, mit

72 Eine ähnliche Funktion hat das Bit *Use TSC scaling* in den *Secondary Process-Based VM-Execution Controls* (s. Abschnitt 6.4.1.3).

73 KVM fängt jedoch das mehrmalige aufeinanderfolgende Ausführen des Maschinenbefehls PAUSE ab (s. Abschnitt 6.4.1.3).

Ausnahme des Bits Virtualize APIC accesses, nur dann, sofern das TPR-Shadowing aktiviert ist (s. Abschnitt 29.3 in [INTEL3C]).

Darüber hinaus fängt KVM Manipulationen des Translation Lookaside Buffers (TLB) ab, die durch den Maschinenbefehl WBINVD ausgelöst werden, indem es dasBit WBINVD exiting setzt. Ferner wird dem Gast durch Setzen des Bits Enable INVPCID die Nutzung des Maschinenbefehls INVPCID erlaubt.

Außerdem wird dem Gast gestattet, den Maschinenbefehl RDTSCP zur Erstellung von Zeitstempeln zu nutzen, indem das Bit Enable RDTSCP gesetzt wird. Die Abarbeitung wird hierbei aufgrund des nicht gesetzten Bits RDTSC exiting innerhalb der Primary Process-Based VM-Execution Controls (s. Abschnitt 6.4.1.2) nicht abgefangen. In diesem Zusammenhang wird auch das Bit Use TSC scaling gesetzt, das es KVM erlaubt, beim Erstellen von Zeitstempeln einen wählbaren Faktor einfließen zu lassen (s. Abschnitt 24.6.5 in [INTEL3C]). Eine negative Beeinflussung des Entropie-Pools entsteht laut [VMRNG] hierdurch nicht.

Auch fängt KVM die Ausführung der Maschinenbefehle XSAVES und XRSTORS ab, indem es das Bit Enable XSAVES/XRSTORS setzt.

Zur effizienteren Unterstützung einer verschachtelten Virtualisierung setzt KVM die Bits Enable VPID und VMCS shadowing.

Ferner ermöglicht KVM es dem Gast, Code nicht nur im Protected Mode auszuführen. Dies wird durch die Nutzung des Unrestricted Guest Mode realisiert, der durch das Bit Unrestricted Guest (s. Abschnitt 31.2.1 in [INTEL3C]) aktiviert wird. Auf diesem Wege kann der Gast den CPU-Modus weitgehend ohne Eingriff von KVM ändern. Dies hat den Vorteil, dass eine Reihe von Maschinenbefehlen, sofern sie im Real-Mode ausgeführt werden, nicht von KVM emuliert werden müssen.

KVM nutzt darüber hinaus erweiterte Steuerungsmöglichkeiten, um auf aufeinanderfolgende Aufrufe des Maschinenbefehls PAUSE zu reagieren (s. Abschnitt 24.6.13 in [INTEL3C]). Dazu setzt es das Bit PAUSE-loop exiting.

In den Secondary Process-Based VM-Execution Controls nicht gesetzt sind unter anderem die Bits RDRAND exiting und RDSEED exiting. Hierdurch wird dem Gast eine unkontrollierte Nutzung der Maschinenbefehle RDRAND und RDSEED ermöglicht. Der Gast kann somit Zufallszahlen unmittelbar durch die CPU erstellen lassen (s. Abschnitt 7.3.17.1 in [INTEL1]). Dies unterstützt insbesondere den Aufbau eines Entropie-Pools (s. [VMRNG]).

Ebenso ist das Bit Descriptor-table exiting nicht gesetzt. Sowohl beim Einsatz einer SoftMMU als auch einer verschachtelten Speicherverwaltung (s. Abschnitt 3.3.2) begrenzt diese Einstellung den Speicher, auf den durch die Speichersegmentierung der vCPU zugegriffen werden kann. KVM braucht daher nicht zu kontrollieren, welche Speichersegmente der Gast nutzt.

Des Weiteren unterstützt KVM keine VM-Funktionen (s. Abschnitt 25.5.5 in [INTEL3C]). Aus diesem Grund ist das Bit Enable VM functions nicht gesetzt. Dadurch wird verhindert, dass der Gast unmittelbar Code des Hosts aufrufen kann, indem er den Maschinenbefehl VMFUNC verwendet.

6.4.1.4 Exception Bitmap

6.4.1.5 I/O-Bitmap

Anhand der I/O-Bitmap (s. Abschnitt 24.6.4 in [INTEL3C]) legt KVM fest, auf welche I/O-Adressbereiche der Gast unkontrolliert zugreifen darf. Wird dem Gast kein unkontrollierter Zugriff auf eine IO-Adresse gestattet, so werden die Maschinenbefehle IN, INS, INSB, INSW, INSD, OUT, OUTS, OUTSB, OUTSW und OUTSD abgefangen. Anderenfalls werden diese Maschinenbefehle unmittelbar von der CPU ausgeführt. Die Bitmap wird von KVM derart gesetzt, dass Zugriffe auf den gesamten Adressbereich abgefangen werden. Hiervon ausgenommen ist lediglich die Adresse 0x80. Traditionell wird diese Adresse zur Ausgabe von POST-Codes durch das BIOS genutzt. Nach dem Bootvorgang ist die Adresse daher praktisch funktionslos.

Früher wurde diese Adresse genutzt, um auf einfachem Wege zeitliche Verzögerungen zu implementieren. Auch wenn dieses Vorgehen allgemein als unkritisch betrachtet wird, so wäre es doch wünschenswert, dieses Verhalten z. B. über einen Parameter des Kernelmoduls abschalten zu können. Eine unmittelbare Gefahr ergibt sich jedoch nicht.

6.4.1.6 Zugriffsmasken für CR0 und CR4

Die Zugriffsmasken für CR0 und CR4 legen fest, auf welche Bits dieser Register der Gast unkontrolliert zugreifen darf (s. Abschnitt 24.6.6 in [INTEL3C]). Aufgrund der erheblichen sicherheitstechnischen Bedeutung der meisten Bits in CR0 und CR4 (s. Abschnitt 2.5 in [INTEL3A]) ist der Schutz dieser Register von besonderer Bedeutung.

KVM überlässt dem Gast einen uneingeschränkten Zugriff auf das Bit Task Switched (TS) des CR0 sowie auf die Bits Protected-Mode Virtual Interrupts (PVI), Debugging Extensions (DE), Performance-Monitoring Counter Enable (PCE), Operations System Support for FXSAVE and FXRSTOR Instructions (OSFXSR), Operation System Support for Unmasked SIMD Floating-Point Exceptions (OSXMMEXCPT) und Time Stamp Disable (TSD).

Das Bit TS beeinflusst die Sicherung von CPU-Registern bei einem Task Switch (s. Abschnitt 7.3 in [INTEL3A]). Die Konfiguration dieses Bits hat lediglich gastinterne Auswirkungen. Anhand des Bits PVI kann das (Gast-)Betriebssystem festlegen, ob externe Interrupts aus dem Userspace heraus aktiviert oder deaktiviert werden können (s. Abschnitt 20.4 in [INTEL3B]). Weil KVM das Bit External-interrupt exiting in den Primary Process-Based VM-Execution Controls (s. Abschnitt 6.4.1.2) setzt, werden externe Interrupts jedoch in keinem Fall an den Gast weitergeleitet. Die Konfiguration des Bits PVI hat daher lediglich gastinterne Auswirkungen. Mithilfe der Bits TSD und PCE kann das (Gast-)Betriebssystem spezifizieren, ob die Maschinenbefehle RDTSC (s. S. 545 f. in [INTEL2]) bzw. RDPMC (s. S. 537 ff. in [INTEL2]) auch aus dem Userspace heraus ausführbar sein sollen. Die Ausführung beider Maschinenbefehle hat lediglich gastinterne Auswirkungen. Das Bit DE bestimmt, ob die Debug-Register DR4 und DR5 nutzbar sein sollen. Diese Konfiguration hat jedoch keinen Einfluss darauf, ob entsprechende Zugriffe durch KVM abgefangen werden sollen (s. Abschnitt 6.4.1.2). Die Nutzung dieses Konfigurationsbits hat daher ebenfalls lediglich gastinterne Auswirkungen. Darüber hinaus gibt das Bit OSFXSR an, welche Register durch den Aufruf des Maschinenbefehls FXSAVE gesichert bzw. durch den Maschinenbefehl FXRSTOR geladen werden sollen. Unabhängig von der Konfiguration dieses Bits hat KVM auf alle infrage kommenden Register Zugriff. Die Konfiguration dieses Bits hat daher lediglich gastinterne Auswirkungen. Das Bit OSXMMEXCPT spezifiziert, wie auf Exceptions reagiert werden soll, die durch SSE ausgelöst werden. Die Konfiguration dieses Bits hat ebenso lediglich gastinterne Auswirkungen. Ferner wird bei Verwendung der verschachtelten Speicherverwaltung dem Host unkontrollierter Zugriff auf das Bit PGE des CR4 gegeben. Zwar hat das Bit PGE potenziell sicherheitskritischen Einfluss auf die Speicherverwaltung. Bei Verwendung der verschachtelten Speicherverwaltung sind seine Auswirkungen jedoch auf den Gast beschränkt. Die Konfiguration dieses Bits hat daher lediglich gastinterne Auswirkungen.

KVM konfiguriert die Zugriffsmasken darüber hinaus so, dass alle weiteren Zugriffe auf CR0 und CR4 abgefangen werden. Zusammenfassend ermöglicht es KVM dem Gast daher, nur auf solche Bits unkontrolliert zuzugreifen, die lediglich gastinterne Auswirkungen haben und somit die Sicherheit des Hosts nicht negativ beeinflussen.

6.4.1.7 CR3-Target Controls

Mit den CR3-Target-Controls (s. Abschnitt 24.6.7 in [INTEL3C]) kann der Hypervisor dem Gast eine partiell unkontrollierte Modifikation der Speicherverwaltung ermöglichen. Hierdurch können unter Umständen Performanceverbesserungen bei der Nutzung einer SoftMMU erzielt werden. KVM macht von dieser Möglichkeit jedoch keinen Gebrauch und fängt bei Nutzung einer SoftMMU jegliche

Konfigurationsänderung der MMU ab. Bei Verwendung von EPT wird dem Gast sinnvollerweise ein uneingeschränkter Zugriff auf CR3 eingeräumt.

6.4.1.8 MSR-Bitmap

Anhand der MSR-Bitmap (s. Abschnitt 24.6.9 in [INTEL3C]) bestimmt KVM aufgrund des gesetzten Bits Use MSR bitmaps in den Primary Process-Based VM-Execution Controls (s. Abschnitt 6.4.1.2), auf welche maschinenspezifischen Register unkontrolliert zugegriffen werden kann.

KVM erlaubt auf diese Weise den unkontrollierten Zugriff auf die MSRs IA32_SYSENTER_CS, IA32_SYSENTER_ESP, IA32_SYSENTER_EIP (s. Abschnitt 5.8.7 in [INTEL3A]). Dies ermöglicht dem Gast einen schnellen Kontextwechsel zwischen Userspace und Kernelspace (s. Abschnitt 5.8.7 in [INTEL3A] und Abschnitt 31.10.4.2 in [INTEL3C]). In diesem Zusammenhang wird auch ein unmittelbarer Zugriff auf die MSRs GS_BASE und KERNEL_GS_BASE (s. Abschnitt 31.10.4.4 in [INTEL3C]) gewährt.

Ferner wird eine unkontrollierte Nutzung der Intel Memory Protection Extension (MPX, s. Abschnitt 17 in [INTEL1]) durch den unmittelbaren Zugriff auf das MSR IA32_BNDCFGS (s. Abschnitt 17.4.3.1 in [INTEL1]) ermöglicht. Darüber hinaus wird ein unmittelbarer Zugriff auf das MSR FS_BASE ermöglicht.

Des Weiteren wird lesender Zugriff auf alle MSRs gegeben, die für die Konfiguration von x2APIC zuständig sind (s. Abschnitt 2.3.2 in [X2APIC])[74]. Ausgenommen hiervon sind lediglich das Local APIC ID Register (s. Abschnitt 10.4.6 in [INTEL3A]) und das Current Count Register (s. Abschnitt 10.5.4 in [INTEL3A]). Schreibender Zugriff wird auf das Task Priority Register (s. Abschnitt 10.8.3.1 [INTEL3A] und Abschnitt 29.1.2 in [INTEL3C]), das EOI Register (s. Abschnitt 10.8.5 in [INTEL3A] und Abschnitt 29.1.4 in [INTEL3C]) und das Register Self IPI (s. Abschnitt 10.12.11 in [INTEL3A] und Abschnitt 29.1.5 in [INTEL3C]) erlaubt. Der unkontrollierte Zugriff auf die aufgeführten MSRs hat lediglich gastinterne Auswirkungen und stellt keine Einschränkung der Sicherheit dar.

Auffallend ist, dass ein lesender Zugriff auf das MSR IA32_TSC nicht gegeben wird. Somit können Zeitstempel zwar mit dem Maschinenbefehl RDTSC, nicht jedoch mit RDMSR unkontrolliert abgefragt werden (s. auch Abschnitt 6.4.2.2). Dies führt dazu, dass Zeitstempel, die mit RDTSC ausgelesen wurden, wesentlich präziser sind als solche, die mit RDMSR ermittelt werden.

6.4.2 Laufzeitverhalten

6.4.2.1 Zugriff auf Control Register

6.4.2.1.1 CR0

Änderungen an CR0 (s. Abschnitt 2.5 [INTEL3A]) werden durch das Zusammenspiel der Funktionen handle_set_cr0, kvm_set_cr0, vmx_set_cr0 implementiert. Der Gast kann die Bits modifizieren, die in Tabelle 21 genannt sind.

Bit	Kommentar
Paging (PG)	Bestimmt, ob das Paging aktiviert sein soll (s. unten).
Protection Enabled (PE)	Bestimmt, ob die CPU im Real oder Protection Mode arbeiten soll (s. unten).

74 Konkret werden die MSRs 0x800 bis 0x8ff freigegeben. Dies verwundert, da die für x2APIC reservierten MSRs bis einschließlich 0xbff reichen (s. Abschnitt 2.3.2 in [X2APIC]). Ob es sich hierbei um einen Fehler oder ein gewolltes Verhalten handelt, ist nicht ersichtlich. Da die ausgeschlossenen MSRs aber bislang keine Funktion zugewiesen bekommen haben, hat dies praktisch keine Auswirkungen.

Bit	Kommentar
Numeric Error (NE)	Spezifiziert das Verhalten der Floating Point Unit (FPU) im Fehlerfall. Hat lediglich gastinterne Auswirkungen.
Alignment Mask (AM)	Spezifiziert das Verhalten von nicht ausgerichteten Speicherzugriffen (s. Abschnitt 5.10.5 in [INTEL3A]). Hat lediglich gastinterne Auswirkungen.
Monitor Coprocessor (MP)	Bestimmt das Verhalten des Maschinenbefehls MWAIT. Hat lediglich gastinterne Auswirkungen.
Write Protect (WP)	Bestimmt, ob im Kernelmode in schreibgeschützte Speicherseiten geschrieben werden kann (s. Abschnitt 4.6 in [INTEL3A]). Hat lediglich gastinterne Auswirkungen.
Task Switched (TS)	Bestimmt das Verhalten eines Prozesswechsels. Hat lediglich gastinterne Auswirkungen.
Emulation (EM)	Dient zur Aktivierung der FPU. Hat lediglich gastinterne Auswirkungen.

Tabelle 21: Modifizierbare Bits des CR0

Die Änderung der Bits PG und PE ist jedoch nur möglich, sofern die Funktion Unrestricted Guest (s. Abschnitt 25.6 in [INTEL3C]) der vCPU aktiv ist. Ist dies der Fall, so ist ein Verlassen des Protected Mode auf sichere Weise möglich.

Demgegenüber ermöglicht es KVM dem Gast nicht, die Nutzung des CPU-Caches mithilfe der Bits Write-through (NW) und Cache Disable (CD) zu deaktivieren.

Modifikationen des CR0 durch den Gast haben daher lediglich gastinterne Auswirkungen.

6.4.2.1.2 CR3

Änderungen an CR3 (s. Abschnitt 2.5 [INTEL3A]) werden durch die Funktionen kvm_set_cr3 durchgeführt, sofern der Schreibvorgang abgefangen wird. Dies ist bei Verwendung einer SoftMMU der Fall (s. Abschnitt 6.4.1.7). Die Neuberechnung der SoftMMU wird hierbei anhand der Funktion mmu_free_roots durchgeführt. Hierzu wird die vom Gast erstellte Konfiguration der MMU ausgewertet und eine Schattenkopie erstellt. Insbesondere aufgrund der großen Anzahl der Konfigurationsmöglichkeiten, die der Gast hat, ist dieser Vorgang daher in hohem Maße komplex. Aus sicherheitstechnischen Gründen ist daher von der Nutzung einer SoftMMU abzuraten.

6.4.2.1.3 CR4

Änderungen an CR4 (s. Abschnitt 2.5 [INTEL3A]) werden durch das Zusammenspiel der Funktionen kvm_set_cr4 und vmx_set_cr4 realisiert. Tabelle 22 zeigt, welche Bits des CR4 durch den Gast modifiziert werden können.

Bit	Kommentar	Durch Gast modifizierbar?
Virtual-8086 Mode Extensions (VME)	Bestimmt, wie Interrupts und Exceptions im Virtual 8086 Mode verarbeitet werden.	Ja, sofern sich der Gast im Protected Mode befindet.
Protected-Mode	Bestimmt, wie Interrupts im Protected Mode	Ja

Bit	Kommentar	Durch Gast modifizierbar?
Virtual Interrupts (PVI)	verarbeitet werden. Hat lediglich gastinterne Auswirkungen.	
Time Stamp Disable (TSD)	Bestimmt, ob der Maschinenbefehl RDTSC (s. Abschnitt 2.8.6 in [INTEL3A]) aus dem Userspace aufgerufen werden darf. Hat lediglich gastinterne Auswirkungen.	Ja
Debugging Extensions (DE)	Bestimmt, ob die Debug-Register 4 und 5 zur Verfügung stehen. Hat lediglich gastinterne Auswirkungen.	Ja
Page Size Extensions (PSE)	Ermöglicht Speicherseiten mit einer Größe von 4 MB statt 4 KB.	Ja, es sei denn, EPT ist aktiv, Paging (s. CR0) jedoch nicht. In diesem Fall wird PSE gesetzt.
Physical Address Extension (PAE)	Ermöglicht es, im Protected Mode ohne aktiven Long Mode mehr als 4 GB Speicher zu nutzen.	Ja, sofern EPT aktiv ist und der Gast Paging (s. CR0) nutzt.
Machine-Check Enable (MCE)	Sofern gesetzt, werden Hardwarefehler mittels einer entsprechenden Exception signalisiert (s. Abschnitt 15 in [INTEL3B]). Die Funktion darf auch bei aktivierter Virtualisierung nicht abgeschaltet werden, um mögliche Fehlerzustände nicht zu übersehen.	Nein, wird stets von der Hostkonfiguration übernommen.
Page Global Enable (PGE)	Ermöglicht es, Speicherseiten als global zu markieren (s. Abschnitt 4.10 in [INTEL3A]). Hat lediglich gastinterne Auswirkungen.	Ja
Performance-Monitoring Counter Enable (PCE)	Ermöglicht den Zugriff auf die Performance Monitoring Unit (PMU) über den Maschinenbefehl RDPMC (s. S. 537 in [INTEL2]). Hat lediglich gastinterne Auswirkungen. Da die Implementierung von PMU aber komplex ist, wäre eine Möglichkeit zur Deaktivierung wünschenswert.	Ja
Operating System Support for FXSAVE and FXRSTOR Instructions (OSFXSR)	Bestimmt, ob die Maschinenbefehle FXSAVE und FXRSTOR ausgeführt werden können. Hat lediglich gastinterne Auswirkungen.	Ja
Operating System Support for Unmasked SIMD Floating-Point Exceptions (OSXMMEXCPT)	Legt fest, wie auf bestimmte SIMD-Fehler reagiert werden soll. Hat lediglich gastinterne Auswirkungen.	Ja
User-Mode Instruction Prevention (UMIP)	Verhindert die Ausführung der Maschinenbefehle SGDT, SIDT, SLDT, SMSW und STR aus dem Userspace. Wird von KVM nicht unterstützt.	Nein, ist stets 0.
VMX-Enable Bit (VMXE)	Bestimmt den Zustand der Virtualisierungserweiterung. Darf nicht durch den Gast beeinflussbar sein.	Nein, ist stets 1.

Bit	Kommentar	Durch Gast modifizierbar?
SMX-Enable Bit (SMXE)	Ermöglicht die Nutzung von SMX-Anweisungen (s. Abschnitt 6 in [INTEL2]) als Teil von Intels Trusted Execution Technology. Wird von KVM nicht unterstützt.	Nein, ist stets 0.
FSGSBASE-Enable Bit (FSGSBASE)	Ermöglicht die Ausführung der Maschinenbefehle RDFSBASE, RDGSBASE, WRFSBASE und WRGSBASE. Hat lediglich gastinterne Auswirkungen.	Ja, es sei denn, der Gast wurde dahingehend konfiguriert, die Maschinenbefehle nicht zu unterstützen.
PCID-Enable Bit (PCIDE)	Ermöglicht die Nutzung eines Process-Context Identifier. Hat lediglich gastinterne Auswirkungen.	Ja
XSAVE and Processor Extended States-Enable Bit (OSXSAVE)	Ermöglicht die Nutzung der Maschinenbefehle XGETBV, XSAVE und XRSTOR. Hat lediglich gastinterne Auswirkungen.	Ja, es sei denn, der Gast wurde dahingehend konfiguriert, diese Funktion nicht zu unterstützen.
SMEP-Enable Bit (SMEP)	Ermöglicht es, Speicherseiten als „nicht ausführbar im Kernelspace“ zu markieren. Hat lediglich gastinterne Auswirkungen.	Ja, es sei denn, der Gast wurde dahingehend konfiguriert, diese Funktion nicht zu unterstützen.
SMAP-Enable Bit (SMAP)	Ermöglicht es, Speicherseiten vor dem Zugriff durch den Kernelspace zu schützen. Hat lediglich gastinterne Auswirkungen.	Ja, es sei denn, der Gast wurde dahingehend konfiguriert, diese Funktion nicht zu unterstützen.
Protection-Key-Enable Bit (PKE)	Ermöglicht einen schlüsselbasierten Schutz von Speicherseiten. Hat lediglich gastinterne Auswirkungen.	Ja, es sei denn, der Gast wurde dahingehend konfiguriert, diese Funktion nicht zu unterstützen.

Tabelle 22: Modifizierbare Bits des CR4

Zusammenfassend lässt sich festhalten, dass die von einem Gast modifizierbaren Bits entweder lediglich gastinterne Auswirkungen haben oder dass deren Änderung ausreichend stark eingeschränkt ist, um eine negative Beeinflussung des Hosts zu vermeiden.

6.4.2.2 Zugriff auf maschinenspezifische Register

6.4.2.2.1 Lesender Zugriff

Ergänzend zu der Umsetzung des lesenden Zugriffs auf allgemeine MSRs, die in Abschnitt 6.3.1.1 beschrieben wurde, implementiert die Funktion vmx_get_msr lesenden Zugriff auf MSRs, die spezifisch für CPUs von Intel sind. Im Folgenden wird diese Implementierung untersucht. Es wird hierbei jedoch nicht überprüft, ob die zum Gast zurückgegebenen Werte spezifikationsgetreu sind.

Ein Großteil der MSRs ist so implementiert, dass der Wert, den sie zurückgeben sollen, praktisch unmittelbar aus einer entsprechenden Verwaltungsstruktur ausgelesen wird (z. B. aus der VMCS).[75] Die entsprechenden MSRs sind Tabelle 23 aufgeführt. Ihre Implementierung ist trivial und nicht zu beanstanden. Ferner ist zu beachten, dass alle in Tabelle 23 aufgeführten MSRs – mit Ausnahme von IA32_XSS – ohnehin unmittelbar vom Gast ausgelesen werden dürfen (s. Abschnitt 6.4.1.8). Die entsprechenden Implementierungen werden somit regelmäßig nicht genutzt.

MSR	Implementierung
FS_BASE	vmcs_readl(GUEST_FS_BASE)
GS_BASE	vmcs_readl(GUEST_GS_BASE)
KERNEL_GS_BASE	vcpu->msr_guest_kernel_gs_base
IA32_SYSENTER_CS	vmcs_read32(GUEST_SYSENTER_CS)
IA32_SYSENTER_EIP	vmcs_readl(GUEST_SYSENTER_EIP)
IA32_SYSENTER_ESP	vmcs_readl(GUEST_SYSENTER_ESP)
IA32_BNDCFGS	vmcs_read64(GUEST_BNDCFGS)
IA32_XSS	vcpu->arch.ia32_xss

Tabelle 23: MSRs mit trivialer Implementierung des lesenden Zugriffs

Ferner existiert das MSR IA32_TSC, das mit der Funktion guest_read_tsc implementiert ist. Diese gibt einen Zeitstempel unter Berücksichtigung des TSC-Offsets (s. Abschnitt 6.4.1.2) und des TSC-Scalings (s. Abschnitt 6.4.1.3) zurück. Da der Zugriff auf das Register, anders als die Ausführung des Maschinenbefehls RDTSC, abgefangen wird, kann eine präzise Zeitmessung durch Zugriff auf das MSR IA32_TSC nicht garantiert werden. Es erscheint daher sinnvoll, den lesenden Zugriff auf das MSR IA32_TSC anhand der MSR-Bitmap (s. Abschnitt 6.4.1.8) unkontrolliert zu gestatten.

Darüber hinaus werden, sofern dem Gast selbst eine Virtualisierung erlaubt ist, die folgenden MSRs anhand der Funktion vmx_get_vmx_msr implementiert. Die Umsetzung der verschachtelten Virtualisierung bleibt im Rahmen dieser Studie jedoch unberücksichtigt.

- IA32_FEATURE_CONTROL
- IA32_VMX_BASIC
- IA32_VMX_PINBASED_CTLS
- IA32_VMX_PROCBASED_CTLS
- IA32_VMX_EXIT_CTLS
- IA32_VMX_ENTRY_CTLS
- IA32_VMX_MISC
- IA32_VMX_CR0_FIXED0
- IA32_VMX_CR0_FIXED1
- IA32_VMX_CR4_FIXED0
- IA32_VMX_CR4_FIXED1
- IA32_VMX_VMCS_ENUM
- IA32_VMX_PROCBASED_CTLS2
- IA32_VMX_EPT_VPID_CAP
- IA32_VMX_TRUE_PINBASED_CTLS
- IA32_VMX_TRUE_PROCBASED_CTLS
- IA32_VMX_TRUE_EXIT_CTLS
- IA32_VMX_TRUE_ENTRY_CTLS

75 Teilweise wird vor der Rückgabe des Wertes überprüft, ob die zugehörige Funktion für den Gast freigeschaltet ist.

- IA32_VMX_VMFUNC

Wird auf ein MSR lesend zugegriffen, das nicht aufgeführt ist, so wird die Funktion kvm_get_msr_common genutzt, die allgemeine MSRs implementiert (s. Abschnitt 6.3.1.1).

Es sind somit keine Auffälligkeiten in Hinblick auf die Implementierung des lesenden Zugriffs feststellbar. Die einzige Ausnahme bildet das anscheinend unnötige Abfangen des Zugriffs auf das MSR IA32_TSC.

6.4.2.2.2 Schreibender Zugriff

Ergänzend zu der Umsetzung des schreibenden Zugriffs auf allgemeine MSRs, die in Abschnitt 6.3.1.2 beschrieben wurde, implementiert die Funktion vmx_set_msr schreibenden Zugriff auf MSRs, die spezifisch für CPUs von Intel sind. Im Folgenden wird diese Implementierung untersucht. Hierbei wird jedoch nicht überprüft, ob auf den schreibenden Zugriff spezifikationsgetreu reagiert wird.

Ein Großteil der MSRs ist so implementiert, dass der vom Gast übergebene Wert praktisch unmittelbar in einer internen Verwaltungsstruktur gespeichert wird, z. B. in der VMCS.[76] Die entsprechenden MSRs sind in Tabelle 24 aufgeführt. Ihre Implementierung ist trivial und nicht zu beanstanden. Ferner ist zu beachten, dass alle in Tabelle 24 aufgeführten MSRs ohnehin unmittelbar vom Gast geschrieben werden dürfen (s. Abschnitt 6.4.1.8). Die entsprechenden Implementierungen werden somit regelmäßig nicht genutzt.

MSR	Implementierung
FS_BASE	vmcs_writel(GUEST_FS_BASE, data);
GS_BASE	vmcs_writel(GUEST_GS_BASE, data);
KERNEL_GS_BASE	vmx->msr_guest_kernel_gs_base = data;
IA32_SYSENTER_CS	vmcs_write32(GUEST_SYSENTER_CS, data);
IA32_SYSENTER_EIP	vmcs_writel(GUEST_SYSENTER_EIP, data);
IA32_SYSENTER_ESP	vmcs_writel(GUEST_SYSENTER_ESP, data);
IA32_BNDCFGS	vmcs_write64(GUEST_BNDCFGS, data);

Tabelle 24: MSRs mit trivialer Implementierung des schreibenden Zugriffs

Darüber hinaus existiert eine Reihe von MSRs, deren Implementierung des schreibenden Zugriffs komplexer ist. Diese sind in Tabelle 25 aufgeführt.

MSR	Funktion	Kommentar
IA32_TSC	kvm_write_tsc	Mit der Funktion kvm_write_tsc wird der TSC-Offset der VMCS gesetzt. Keine Auffälligkeiten feststellbar.
IA32_CR_PAT	-	Speichern des übergebenen Wertes in die VMCS nach vorheriger Validierung des Wertes (s. Abschnitt 11.12.2 in [INTEL3A]). Keine Auffälligkeiten feststellbar.
IA32_XSS	-	Speichern des übergebenem Wertes in vcpu->arch.ia32_xss, sofern der übergebene Wert 0 ist. Ansonsten wird eine Exception im Gast ausgelöst. Keine Auffälligkeiten feststellbar.

76 Teilweise wird vor dem Speichern des Wertes überprüft, ob die zugehörige Funktion für den Gast freigeschaltet ist.

Tabelle 25: MSRs mit komplexer Implementierung des schreibenden Zugriffs

Sollte das zu schreibende MSR nicht aufgeführt sein, wird die Funktion kvm_set_msr_common aufgerufen und der Schreibvorgang so durchgeführt, wie in Abschnitt 6.3.1.2 beschrieben.

Es sind somit keine Auffälligkeiten in Hinblick auf die Implementierung des schreibenden Zugriffs feststellbar.

6.5 AMD-V

6.5.1 Initialisierung des Gastes

6.5.1.1 Virtual Machine Control Block

Wesentliche Teile der Konfiguration der Virtualisierung mittels AMD-V werden anhand des Virtual Machine Control Block (VMCB, s. Abschnitt 15 in [AMD2] und Appendix B in [AMD2]) umgesetzt. KVM nutzt den VMCB, um Zugriffe auf die Control Register der CPU sowie lesende und schreibende Zugriffe auf CR0, CR3 und CR4 abzufangen (s. Abschnitte 3.1 und 15.9 in [AMD2]). Ferner wird das Bit Intercept Task Switch gesetzt. Hieraus resultiert, dass sowohl direkte Zugriffe auf die genannten Register durch die Maschinenbefehle MOV, LMSW, SMSW und CLTS als auch indirekte Modifikationen durch einen Task-Switch abgefangen werden (s. Abschnitt 15.9 in [AMD2]).

Eine Ausnahme hiervon besteht jedoch bei Nutzung der verschachtelten Speicherverwaltung (s. Abschnitt 3.3.2). Wird diese Funktion genutzt, so wird dem Gast ein uneingeschränkter lesender und schreibender Zugriff auf CR3 gewährt. In diesem Fall wird ferner eine unkontrollierte Ausführung des Maschinenbefehls INVLPG (s. u.) zugelassen. Dies ist zweckdienlich und unbedenklich.

Eine weitere Ausnahme stellt der schreibende Zugriff auf das Bit Task Switched (s. S. 44 in [AMD2]) und das Bit Monitor coprocessor (s. S. 43 f. in [AMD2]) des CR0 dar. KVM erlaubt diesen Zugriff, indem es das Bit Selective CR0 Write Intercept setzt. Der unkontrollierte Zugriff auf diese Bits hat lediglich gastinterne Auswirkungen und ist daher unbedenklich.

Darüber hinaus wird ein schreibender Zugriff auf CR8 abgefangen, sofern der Advanced Virtual Interrupt Controller (AVIC, s. Abschnitt 15.29 in [AMD2]) nicht aktiv ist. Bei aktivem AVIC wird dem Gast ein unkontrollierter Zugriff auf CR8 gewährt. Dies ist zweckdienlich und unbedenklich.

Darüber hinaus setzt KVM die entsprechenden Bits im VMCB, um die Invalidierung des TLBs mittels der Maschinenbefehle

- INVD (s. S. 353 in [AMD3])
- INVLPG (s. S. 354 in [AMD3])
- INVLPGA (s. S. 355 in [AMD3])
- WBINVD (s. S. 435 in [AMD3])

abzufangen. Ebenso wird der VMCB konfiguriert, um Zugriffe auf die globale Interruptsteuerung mittels der Maschinenbefehle

- STGI (s. S. 405 in [AMD3])
- CLGI (s. S. 345 in [AMD3])

abzufangen. Analog wird der VMCB durch KVM konfiguriert, um eine unkontrollierte Ausführung der Maschinenbefehle

- CPUID (s. S. 189 in [AMD3])
- RDPMC (s. S. 388 in [AMD3])
- HLT (s. S. 385 in [AMD3])

- MONITOR (s. S. 375 in [AMD3])
- MWAIT (s. S. 383 in [AMD3])
- XSETBV (s. S. 857 in [AMD4])

zu unterbinden. Ebenso wird der VMCB konfiguriert, um die folgenden Maschinenbefehle für eine verschachtelte Virtualisierung durchzuführen:

- VMRUN (s. S. 428 ff. in [AMD3])
- VMMCALL (s. S. 427 in [AMD3])
- VMLOAD (s. S. 425 f. in [AMD3])
- VMSAVE (s. S. 433 f. in [AMD3])
- SKINIT (s. S. 436 f. in [AMD3])

Darüber hinaus wird der VMCB konfiguriert, um sowohl maskierbare als auch nicht maskierbare Interrupts abzufangen.

Ferner werden alle lesenden und schreibenden Zugriffe auf die Debug-Register DR0 bis DR7 abgefangen.

6.5.1.2 I/O Permissions Map

Anhand der I/O Permission Map kann festgelegt werden, auf welche IO-Adressbereiche der Gast unkontrolliert zugreifen kann (s. Abschnitt 15.10.1 in [AMD2]). KVM aktiviert diese Überprüfung, indem es das Bit IOIO_PROT (s. Abschnitt 15.10.2 in [AMD2]) in dem VMCB setzt. In der I/O Permission Map werden alle zur Verfügung stehenden Bits gesetzt. Somit wird jeglicher lesender und schreibender Zugriff auf den IO-Adressbereich abgefangen.

6.5.1.3 MSR Permission Map

Anhand der MSR Permission Map wird bestimmt, auf welche maschinenspezifischen Register unmittelbar, d. h. ohne Kontrolle durch den Hypervisor, zugegriffen werden kann. KVM erlaubt unkontrollierte lesende und schreibende Zugriffe auf die MSRs STAR, LSTAR, CSTAR, SFMASK und SYSENTER_CS, um einen schnellen Kontextwechsel zwischen Userspace und Kernelspace innerhalb des Gastes zu ermöglichen (s. Abschnitt 6.1.1 und 6.1.2 in [AMD2]). In diesem Zusammenhang wird auch ein unmittelbarer Zugriff auf die MSRs GS_BASE und KERNEL_GS_BASE (s. Abschnitt 6.1.3 in [AMD2]) gewährt. Darüber hinaus wird ein unmittelbarer Zugriff auf das MSR FS_BASE ermöglicht. Die Verwendung der genannten Register hat lediglich gastinterne Auswirkungen und ist somit unbedenklich.

Ist der Debug-Mode aktiviert, so wird auch auf die MSRs LASTBRANCHFROMIP, LASTBRANCHTOIP, LASTINTFROMIP und LASTINTTOIP Zugriff gewährt (s. Abschnitt 3.2.4 in [AMD2]).

6.5.2 Laufzeitverhalten

6.5.2.1 Zugriff auf Control Register

Der Zugriff auf die Control Register CR0, CR3 und CR4 wird durch KVM umfassend eingeschränkt. Die Zugriffsmöglichkeiten des Gastes entsprechend im Wesentlichen denen, die in Abschnitt 6.4.1 beschrieben sind. Angriffsmöglichkeiten auf den Host oder andere Gäste durch Manipulation der Control Register durch den Gast können nicht erkannt werden.

6.5.2.2 Zugriff auf maschinenspezifische Register

6.5.2.2.1 Lesender Zugriff

Ergänzend zu der Umsetzung des lesenden Zugriffs auf allgemeine MSRs, die in Abschnitt 6.3.1.1 beschrieben wurde, implementiert die Funktion svm_get_msr lesenden Zugriff auf MSRs, die spezifisch für CPUs von AMD sind. Im Folgenden wird diese Implementierung untersucht. Hierbei wird jedoch nicht überprüft, ob die zum Gast zurückgegebenen Werte spezifikationsgetreu sind.

Ein Großteil der MSRs ist so implementiert, dass der zurückzugebende Wert praktisch unmittelbar aus einer entsprechenden Verwaltungsstruktur ausgelesen wird, z. B. aus dem VMCB.[77] Die entsprechenden MSRs sind in Tabelle 26 aufgeführt. Ihre Implementierung ist trivial und nicht zu beanstanden. Ferner ist zu beachten, dass der Gast auf einen Großteil der in Tabelle 26 aufgeführten MSRs unkontrolliert zugreifen kann (s. Abschnitt 6.5.1.3). Die entsprechenden Implementierungen werden somit regelmäßig nicht genutzt.

MSR	Implementierung
STAR	svm->vmcb->save.star
LSTAR	svm->vmcb->save.lstar
CSTAR	svm->vmcb->save.cstar
KERNEL_GS_BASE	svm->vmcb->save.kernel_gs_base
SYSCALL_MASK	svm->vmcb->save.sfmask
IA32_SYSENTER_CS	svm->vmcb->save.sysenter_cs
IA32_SYSENTER_EIP	svm->sysenter_eip
IA32_SYSENTER_ESP	svm->sysenter_esp
TSC_AUX	svm->tsc_aux
IA32_DEBUGCTLMSR	svm->vmcb->save.dbgctl
IA32_LASTBRANCHFROMIP	svm->vmcb->save.br_from
IA32_LASTBRANCHTOIP	svm->vmcb->save.br_to
IA32_LASTINTFROMIP	svm->vmcb->save.last_excp_from
IA32_LASTINTTOIP	svm->vmcb->save.last_excp_to
VM_HSAVE_PA	svm->nested.hsave_msr
VM_CR	svm->nested.vm_cr_msr
IA32_UCODE_REV	0x01000065
F15H_IC_CFG	0x1E[78]

Tabelle 26: AMD-spezifische MSRs mit trivialer Implementierung des lesenden Zugriffs

Ferner existiert das MSR IA32_TSC, das mithilfe der Funktion kvm_scale_tsc implementiert ist. Diese Funktion gibt einen Zeitstempel unter Berücksichtigung des TSC-Offsets (s. Abschnitt 6.5.1.1) zurück. Da der Zugriff auf das Register, anders als die Ausführung des Maschinenbefehls RDTSC, abgefangen wird, kann eine präzise Zeitmessung durch Zugriff auf das MSR IA32_TSC nicht garantiert werden. Es erscheint

77 Teilweise wird vor der Rückgabe des Wertes überprüft, ob die zugehörige Funktion für den Gast freigeschaltet ist.

78 Je nach Konfiguration des Gastes wird alternativ auch eine Exception im Gast ausgelöst.

daher sinnvoll, den lesenden Zugriff auf das MSR IA32_TSC anhand der MSR-Bitmap (s. Abschnitt 6.4.1.8) unkontrolliert zu gestatten.

Wird lesend auf ein MSR zugegriffen, das nicht aufgeführt ist, so wird die Funktion kvm_get_msr_common genutzt, die allgemeine MSRs implementiert (s. Abschnitt 6.3.1.1).

Es sind somit keine Auffälligkeiten in Hinblick auf die Implementierung des lesenden Zugriffs feststellbar. Die einzige Ausnahme bildet das anscheinend unnötige Abfangen des Zugriffs auf das MSR IA32_TSC.

6.5.2.2.2 Schreibender Zugriff

Ergänzend zu der Umsetzung des schreibenden Zugriff auf allgemeine MSRs, die in Abschnitt 6.3.1.2 beschrieben wurde, implementiert die Funktion svm_set_msr schreibenden Zugriff auf MSRs, die spezifisch für CPUs von AMD sind. Im Folgenden wird diese Implementierung untersucht. Hierbei wird jedoch nicht überprüft, ob auf den schreibenden Zugriff spezifikationsgetreu reagiert wird.

Ein Großteil der MSRs ist so implementiert, dass der vom Gast übergebene Wert praktisch unmittelbar in einer internen Verwaltungsstruktur gespeichert wird, z. B. in dem VMCB.[79] Die entsprechenden MSRs sind in Tabelle 27 aufgeführt. Ihre Implementierung ist trivial und nicht zu beanstanden. Ferner ist zu beachten, dass viele der in Tabelle 27 aufgeführten MSRs ohnehin unmittelbar vom Gast geschrieben werden dürfen (s. Abschnitt 6.5.1.3). Die entsprechenden Implementierungen werden somit regelmäßig nicht genutzt.

MSR	Speicher
STAR	svm->vmcb->save.star
LSTAR	svm->vmcb->save.lstar
CSTAR	svm->vmcb->save.cstar
KERNEL_GS_BASE	svm->vmcb->save.kernel_gs_base
SYSCALL_MASK	svm->vmcb->save.sfmask
IA32_SYSENTER_CS	svm->vmcb->save.sysenter_cs
IA32_SYSENTER_EIP	svm->vmcb->save.sysenter_eip
IA32_SYSENTER_ESP	svm->vmcb->save.sysenter_esp
VM_HSAVE_PA	svm->nested.hsave_msr

Tabelle 27: AMD-spezifische MSRs mit trivialer Implementierung des schreibenden Zugriffs

Darüber hinaus existiert eine Reihe von MSRs, deren Implementierung des schreibenden Zugriffs komplexer ist. Diese sind in Tabelle 28 aufgeführt.

MSR	Funktion	Kommentar
IA32_TSC	kvm_write_tsc	Mit der Funktion kvm_write_tsc wird der TSC-Offset des VMCB gesetzt. Keine Auffälligkeiten feststellbar.
IA32_DEBUGCTLMSR	-	Dient zur Aktivierung von Debug-Registern. Kann nur genutzt werden, wenn der Gast bei der Initialisierung entsprechend konfiguriert wird. Keine Auffälligkeiten feststellbar.
VM_CR	svm_set_vm_cr	Ist Teil der verschachtelten Virtualisierung und wird daher nicht betrachtet.

79 Teilweise wird vor dem Speichern des Wertes überprüft, ob die zugehörige Funktion für den Gast freigeschaltet ist.

Tabelle 28: AMD-spezifische MSRs mit komplexer Implementierung des schreibenden Zugriffs

Sollte das zu schreibende MSR nicht aufgeführt sein, dann wird die Funktion kvm_set_msr_common aufgerufen und der Schreibvorgang so durchgeführt, wie in Abschnitt 6.3.1.2 beschrieben.

Es sind somit keine Auffälligkeiten in Hinblick auf die Implementierung des schreibenden Zugriffs auf AMD-spezifische MSRs feststellbar.

6.6 Hardwaredurchreichung

KVM ermöglicht die Durchreichung von PCI-Geräten, sofern die Hardware eine IOMMU zur Verfügung stellt. KVM nutzt die IOMMU mit dem Ziel, die Hardware, die durchgereicht werden soll, von anderer Hardware des Hosts zu isolieren (s. Abschnitt 3.3.4). Hierbei stellt sich die Frage, ob diese Isolation in jedem Fall effektiv ist.

Problematisch ist hier, dass sich Betriebssystem und Hardware klassischerweise uneingeschränkt vertrauen. Dieses grundlegende Designkonzept von PCI stammt aus seinen Anfängen im Jahr 1991 und wurde über alle Erweiterungen des Standards beibehalten, auch bei PCI-Express. Erst die vergleichsweise jungen Techniken VT-d (s. [VT-D]) und AMD-IO (s. [AMD-IO]) schränken dieses Vertrauensverhältnis zwischen PCI-Gerät und (Gast-)Betriebssystem ein. Beide Techniken haben hierbei das Ziel, kompatibel zu dem bisherigen Standard und der verfügbaren Hardware zu sein. Der historisch gewachsene Funktionsumfang von PCI muss daher funktional unverändert für die Sicherheitsanforderungen der Hardwaredurchreichung eingeschränkt werden.

Als problematisch erweist sich hier unter anderem die Verarbeitung von Interrupts. Moderne PCI-Geräte übermitteln Interrupts nicht mehr über dedizierte Leiterbahnen, sondern mithilfe von Nachrichten, sogenannten Message Signaled Interrupts (MSI, s. Kapitel 9 in [PCI]). Dies erlaubt eine wesentlich flexiblere und einfachere Konfiguration der PCI-Geräte. Modernen PCI-Geräten können auf diesem Wege die Interrupt-Nummern zugewiesen werden, die sie verwenden sollen. Diese Flexibilität birgt jedoch eine Reihe von potenziellen Sicherheitsproblemen.

So zeigt [HASE] eine Reihe von Angriffen, die auf der Erzeugung bösartiger Interrupts basieren. Hierbei wird ein handelsübliches PCI-Gerät[80] in den Gast durchgereicht und von diesem absichtlich fehlerhaft konfiguriert. Das hierdurch provozierte Verhalten reicht von instabilen Zuständen des Hosts bis hin zur Ausführung von Schadcode auf dem Host. Eine der wesentlichen Ursachen hierfür ist die fehlerhafte Verarbeitung der Interruptanforderungen durch die Hardware.[81] Beispielsweise wird gezeigt, dass Interruptanforderungen, die ausschließlich zur Kommunikation zwischen den Prozessoren bestimmt sind, von den PCI-Geräten erzeugt werden können und anschließend von der Hardware verarbeitet werden. Zwar kann dieses Ergebnis nicht ohne Weiteres auf beliebige Serversysteme verallgemeinert werden. Es zeigt jedoch, dass die konkret verwendete Hardware eine erhebliche Bedeutung für die Sicherheit der Hardwaredurchreichung hat.

[HASE] legt nahe, dass die gezeigten Angriffe durch Nutzung von Interrupt Remapping (s. Abschnitt 2.5.2 in [VT-D]) abgewehrt werden können. Hardware, die diese Technik unterstützt, ermöglicht es dem Hypervisor, eine gerätespezifische Whitelist von erlaubten Interrupts zu verwalten. Interrupt Remapping ermöglicht also eine Isolation der PCI-Geräte bezüglich der Interrupterzeugung, vergleichbar mit der Isolation, die eine IOMMU für Speicherzugriffe vornehmen kann.

[PLS] zeigt hingegen, dass auch beim Einsatz von Interrupt Remapping instabile Systemzustände provoziert werden können. Ursächlich ist hier die Erzeugung von nicht standardkonformen Interruptanforderungen. [PLS] resümiert, dass diesbezüglich keine effektiven Abwehrmaßnahmen zur Verfügung stehen.

Außer dem Problem der sicheren Interruptbehandlung bestehen auch Gefahren beim Speicherzugriff. Zwar ermöglichen es VT-d und AMD-IO, mithilfe einer IOMMU den Zugriff des PCI-Geräts auf einen vorab

80 Konkret handelt es sich hierbei um eine e1000-Netzwerkkarte.
81 Des Weiteren werden Schwachstellen im Hypervisor XEN aufgezeigt.

bestimmten Speicherbereich einzuschränken. Die Identifizierung des sendenden PCI-Geräts geschieht jedoch regelmäßig anhand von Angaben, die das jeweilige PCI-Gerät selbst sendet (s. [PCI]).[82] Laut [FIT] kann sich ein PCI-Gerät somit als ein anderes Gerät ausgeben und auf diese Weise die Kommunikationsbeschränkungen der IOMMU zumindest teilweise[83] umgehen. Es ist anzunehmen, dass übliche PCI-Geräte eine solche dem PCI-Standard widersprechende Kommunikation typischerweise nicht ohne Weiteres ermöglichen. Allgemein kann dies jedoch nicht gänzlich ausgeschlossen werden. Insbesondere weil dem Gast ein uneingeschränkter Zugang zum durchgereichten Gerät zur Verfügung steht, kann eine Kompromittierung des Geräts, z. B. durch Ausnutzung von Schwachstellen der Geräte-Firmware oder durch ein modifiziertes Firmware-Update, nicht ausgeschlossen werden.

Eine potenzielle Gefährdung des Hosts ergibt sich, sofern das Verhalten des PCI-Geräts persistent (d. h. über die Dauer der Durchreichung hinaus) durch den Gast verändert werden kann. Außer durch die bereits genannte Möglichkeit einer Firmware-Manipulation kann dies durch dauerhafte Konfigurationsänderungen geschehen.[84]

Es ist somit festzuhalten, dass die durchzureichende Hardware großen Einfluss auf die Sicherheit des Hosts hat. Unter sicherheitstechnischen Gesichtspunkten empfiehlt es sich daher, ausschließlich Hardware durchzureichen, die für die Nutzung mit einem nicht vertrauenswürdigen Betriebssystem konzipiert wurde. Einem solchen PCI-Gerät kann eine erhöhte Widerstandsfähigkeit unterstellt werden. In der Praxis wird die Auswahl auf Hardware mit Virtualisierungsunterstützung (SR-IOV) beschränkt. Zur Feststellung der tatsächlichen Widerstandsfähigkeit muss jedoch stets die Hardware untersucht werden, die durchgereicht werden soll.

Darüber hinaus ermöglicht QEMU in Zusammenarbeit mit KVM keine Live-Migration von Gästen, sofern diese durchgereichte Hardware nutzen. Dies hat erhebliche sicherheitstechnische Auswirkungen. Nur eine Live-Migration – also eine für den Gast nicht ersichtliche und ohne seine Unterstützung durchführbare Verschiebung von einem Host auf einen anderen – ermöglicht Wartungen des Hosts ohne eine Unterbrechung der Gäste. Auf diesem Wege ist es möglich, zentrale Softwarekomponenten zu aktualisieren, z. B. QEMU oder den Linux-Kernel. Ohne Live-Migration ist hierzu typischerweise ein Herunterfahren des Gastes notwendig, was in der Regel den Anforderungen an die Virtualisierung widerspricht.

Aufgrund der beschriebenen Gefährdungen ist die Hardwaredurchreichung mittels KVM nur vertretbar, sofern

- höchstens ein normaler Schutzbedarf vorliegt und
- moderne virtualisierende Hardware mit IRQ-Remapping-Unterstützung verwendet wird und
- das durchzureichende PCI-Gerät speziell für die Virtualisierung entworfen wurde (SR-IOV) und
- eine sinnvolle Update-Strategie ohne die Notwendigkeit der Live-Migration existiert.

Anderenfalls – insbesondere bei hohem oder sehr hohem Schutzbedarf – sollte von einer Verwendung der Hardwaredurchreichung mittels KVM abgesehen werden.

6.7 Versteckte Kommunikationskanäle

Ein wesentlicher Aspekt bei der Nutzung der Virtualisierung ist, dass auf einfache Weise virtuelle Netzwerke erstellt werden und somit eine Kommunikation der Gäste sowohl mit externen Systemen als auch mit anderen Gästen ermöglicht werden kann (s. Abschnitt 3.5.4). Aus Sicherheitsgründen sollte diese Kommunikation jedoch beschränkt werden. Hierzu kann eine Reihe von Methoden zur Einschränkung der Netzwerkkommunikation genutzt werden (s. Kapitel 8). Das Unterbinden einer Kommunikation über versteckte Kommunikationskanäle ist hingegen wesentlich diffiziler. Hier stellt sich die Frage, ob die

82 Konkret sind dies Bus-Device-Function-Adressen (BDF-Adressen).
83 Die Kommunikation beschränkt sich dabei auf unidirektionale Nachrichten, wie z. B. Posted Memory Write Transactions (s. S. 62 in [PCI]).
84 Hierzu gehört beispielsweise auch das Installieren eines Boot-ROM für Netzwerkkarten. Dabei besteht die Gefahr, dass der eingebrachte Code bei einem Neustart des Hosts ausgeführt wird.

Kommunikation zwischen Gästen, die auf demselben Host ausgeführt werden, effektiv unterbunden werden kann. Dabei wird davon ausgegangen, dass die Kommunikation stets ein kooperativer und von den beteiligten Gästen beabsichtigter Informationsaustausch ist.

Die Basis eines versteckten Kommunikationskanals ist stets die gemeinsame Nutzung einer Ressource, die als Informationsträger genutzt wird. Ferner ist es erforderlich, dass ein Gast derart auf die gemeinsame Ressource einwirken kann, dass er deren Verhalten gegenüber einem anderen Gast gezielt beeinflussen kann. Ist dies gegeben, so kann ein Gast Informationen durch die Interaktion mit der Ressource modulieren. Ein anderer Gast kann diese Informationen durch Beobachtung der gemeinsamen Ressource demodulieren.

Ein einfaches Beispiel für einen solchen Vorgang gibt [HHU].[85] Hierbei werden Bits durch hochfrequenten Zugriff auf den Hauptspeicher bzw. durch Unterlassen eines solchen Zugriffs auf den Informationsträger „Speicheranbindung“ moduliert.[86] Auf diese Weise versendet ein Gast in äquidistanten zeitlichen Abständen Bits. Zeitgleich misst der empfangene Gast durch eigenen Zugriff auf den Hauptspeicher die Zugriffszeiten. Die zum Zugriff benötigte Zeit ist hierbei vom Verhalten des sendenden Gastes abhängig. Somit können die gesendeten Informationen demoduliert werden.

[RIS] verfeinert diesen Ansatz und überträgt ihn auf die Virtualisierung. Die angewendeten Methoden erlauben einen Informationsaustausch in Amazon EC2 zwischen zwei Gästen. Die Ergebnisse basieren daher auf der Virtualisierung mittels XEN. Die angewandte Methode sollte jedoch prinzipiell auch auf die Virtualisierung mittels KVM anwendbar sein. Auch wenn die erreichte Übertragungsrate mit unter 1 Bit pro Sekunde sehr gering ist, so wird doch gezeigt, dass ein Informationsaustausch über versteckte Kommunikationskanäle nicht nur unter Laborbedingungen realisierbar ist.

Eine weitere Steigerung der Bandbreite erreicht [ZWU]. Anstelle eines einfachen Speicherzugriffs werden Maschinenbefehle genutzt, die einen atomaren Zugriff ermöglichen. Die im Rahmen eines solchen atomaren Zugriffs durchgeführten Anpassungen des CPU-Caches verzögern die Speicherzugriffe gastübergreifend. Dieses zeitliche Verhalten wird als vergleichsweise effizienter Informationsträger genutzt.[87] Die in Amazon EC2 erreichte Datenrate liegt bei ca. 340 Bit pro Sekunde. Eine Übertragung auf eine KVM-basierte Virtualisierungsumgebung erscheint mit geringem Aufwand möglich.

[PER] zeigt ferner eine Informationsübertragung anhand von CPU-Caches mit ca. 100 Kilobyte pro Sekunde. Zwar erfolgt dies im Rahmen von Multitasking und ist nicht ohne Weiteres auf die Virtualisierung mittels KVM zu übertragen, es zeigt jedoch ein Steigerungspotenzial. [LLC] erzielt einen Datendurchsatz von ca. 1,2 Megabit pro Sekunde in einer Virtualisierungsumgebung, die auf XEN und VMware ESXi basiert. Ob diese Ergebnisse auf KVM übertragbar sind, ist ungewiss.

Um versteckte Kommunikationskanäle zu unterbinden, stehen prinzipiell zwei Ansätze zur Verfügung: Zum einen kann ihnen die Grundlage der gemeinsamen Ressource entzogen werden, zum anderen kann durch Störung eine Informationsübertragung verhindert werden. Bei dem ersten Ansatz wird auf die geteilte Nutzung von Ressourcen verzichtet. Dies kann z. B. geschehen, indem Gäste auf unterschiedlichen Hosts ausgeführt werden. Alternativ ist es ausreichend, wenn die Interaktion eines Gastes mit einer geteilten Ressource deren Verhalten gegenüber anderen Gästen nicht beeinflussen kann. Letzteres erweist sich in der Praxis aufgrund der Vielzahl an gemeinsam genutzten Ressourcen und technischen Beschränkungen der x86-Architektur jedoch als unrealistisch.

Mit der Störung eines versteckten Kommunikationskanals zur Unterbindung eines Informationsaustausches befassen sich unter anderem [GR1], [GR2] und [WHU]. Zwar betrachten diese Arbeiten nicht speziell die Virtualisierung, jedoch ist die Erkenntnis, dass eine ausreichende Störung nicht

85 Der Vorgang wird hierbei nicht im Rahmen der Virtualisierung, sondern im Kontext von Multitasking beschrieben. Das Verfahren lässt sich aber prinzipiell auf die Virtualisierung übertragen.

86 Hierbei ist es notwendig, dass die Speicheranbindung zwischen dem sendenden und dem empfangenden Gast geteilt wird, dass also keine Trennung im Rahmen von NUMA vorliegt. Ferner ist es erforderlich, dass beide Gäste echt zeitgleich ausgeführt werden.

87 Dieser Ansatz funktioniert auch, wenn NUMA genutzt wird und die vCPUs in verschiedenen NUMA-Zellen ausgeführt werden.

trivial umsetzbar ist, dennoch auf die Virtualisierung übertragbar. In diesem Kontext sind insbesondere moderne nachrichtentechnische Verfahren zur Übermittlung von Informationen über gestörte Kommunikationskanäle (s. z. B. [SIG]) zu beachten. Auch dürfen potenzielle Störmaßnahmen die eigentliche Ausführung der Gäste nicht über Gebühr belasten. Es darf daher bezweifelt werden, dass Störmaßnahmen effektiv zur Unterbindung von versteckten Kommunikationskanälen genutzt werden können.

Abschließend ist daher festzuhalten, dass bei Nutzung von KVM versteckte Kommunikationskanäle zwischen Gästen in der Praxis nur unterbunden werden können, indem die Gäste auf unterschiedlichen Hosts ausgeführt werden. Werden Gäste mit KVM auf demselben Host betrieben, so ist ihnen ein kooperativer Informationsaustausch möglich.

6.8 Seitenkanalangriffe

Eine weitere Gefahr für die Vertraulichkeit der Datenverarbeitung stellen Seitenkanalangriffe dar. Anders als bei versteckten Kommunikationskanälen (s. Abschnitt 6.7) findet bei einem Seitenkanalangriff kein kooperativer Informationsaustausch statt. Stattdessen ermittelt ein Gast Informationen über das Verhalten eines anderen Gastes ohne dessen Einverständnis.[88] Das Verhalten des angegriffenen Gastes ist somit nicht darauf ausgerichtet, Informationen preiszugeben. Auch kann der Angreifer das Verhalten des angegriffenen Gastes in der Regel nicht signifikant beeinflussen. Verglichen mit der Kommunikation über versteckte Kommunikationskanäle, erschwert dies die erfolgreiche Durchführung von Seitenkanalangriffen deutlich.

Die Voraussetzungen für Seitenkanalangriffe sind jedoch identisch mit denen für versteckte Kommunikationskanäle: Es wird eine geteilte Ressource gebraucht, deren Interaktion mit einem Gast eine Verhaltensänderung gegenüber einem anderen Gast verursacht. Somit ist es recht offensichtlich, dass zumindest einige grundlegende Informationen über das Verhalten anderer Gäste ermittelt werden können. So kann beispielsweise durch Messung der Zugriffszeiten auf den Hauptspeicher auf die Auslastung anderer Gäste geschlossen werden. Die konkrete Fragestellung lautet daher nicht, ob Informationen über das Verhalten anderer Gäste ermittelt werden können, sondern welche.

Ähnlich wie bei den versteckten Kommunikationskanälen zeigen sich hier die CPU-Caches als potenzielle Informationsträger. Als problematisch stellt sich hierbei heraus, dass die CPU-Caches, genauer gesagt der Layer-3-Cache, auf x86-Prozessoren stets geteilt genutzt werden. Greift eine vCPU auf eine Speicherseite zu, so wird sie in den Cache überführt und eine andere Speicherseite wird aus dem Cache verdrängt.

Diesbezüglich ist zu unterscheiden, ob der jeweilige Angriff eine gemeinsame Nutzung von Speicherseiten benötigt. Liegt eine solche gemeinsame Nutzung zwischen zwei Gästen vor, so kann ein Gast das Zugriffsverhalten des anderen Gastes auf eben diese Speicherseiten verhältnismäßig einfach ermitteln. Greift ein Gast auf eine gemeinsam genutzte Speicherseite zu und überführt er diese somit in den Cache, so wird ein nachfolgender Zugriff auf eben diese Speicherseite beschleunigt. Dies gilt auch, wenn der zweite Lesevorgang durch eine andere vCPU durchgeführt wird. [WAM] nutzt dies beispielsweise, um in einer auf VMware ESXi 5.5.0 basierenden Virtualisierungsumgebung den AES-Schlüssel eines anderen Gastes zu ermitteln. Hierbei wird ausgenutzt, dass die Lookup-Tabellen der AES-Implementierung[89] im gemeinsamen Speicher vorgehalten werden. Somit kann festgestellt werden, welche Elemente dieser Tabellen ausgelesen wurden, und dadurch kann auf den Schlüssel zurückgeschlossen werden. Die notwendige Voraussetzung, dass entsprechende Codebereiche in einem gemeinsamen Speicher vorgehalten werden, erreicht [WAM] durch Ausnutzung von Speicherdeduplizierungsfunktionen des Hypervisors.

Anders als XEN und VMware ESXi verfügt KVM selbst nicht über eine Funktion zur Speicherdeduplizierung. Der Linux-Kernel unterstützt jedoch das Kernel Samepage Merging (KSM). KSM identifiziert Speicherseiten mit gleichem Inhalt und modifiziert die Speicherverwaltung derart, dass die entsprechenden virtuellen Adressen auf die gleiche physische Speicherseite verweisen. Bei Verwendung von KSM ist daher davon auszugehen, dass Angriffe wie der von [WAM] demonstrierte auch bei Nutzung von KVM effektiv sind. Es ist

88 Seitenkanalangriffe, die nicht virtualisierungsspezifisch sind, werden nicht betrachtet.
89 Konkret wird OpenSSL 1.0.1 verwendet.

daher festzustellen, dass die Nutzung von KSM eine Gefahr für die Vertraulichkeit der vom Gast verarbeiteten Daten darstellt. Zwar wird KSM nur von wenigen Linux-Distributionen, wie z. B. CentOS 7, genutzt. Es sollte aber dennoch stets sichergestellt werden, dass KSM deaktiviert ist (s. Abschnitt 11.2.2).

[LLC] zeigt hingegen, dass Angriffe mithilfe von CPU-Caches prinzipiell auch dann möglich sind, wenn kein gemeinsamer Speicher genutzt wird. Die Angriffe wurden mit VMware ESXi durchgeführt. [LLC] legt jedoch nahe, dass diese Angriffe unabhängig vom Hypervisor durchgeführt werden können. Inwieweit die verwendeten Methoden praxistauglich sind, ist noch nicht gänzlich geklärt. Die von [LLC] beschriebenen Abwehrmaßnahmen umfassen entweder eine Änderung der angegriffenen Anwendung[90] oder aber eine Änderung der Hardware. Die einzig aufgeführte Abwehrmaßnahme, die mithilfe des Hypervisors umgesetzt werden kann, besteht darin, genaue Zeitmessungen zu unterbinden. Dies würde aber signifikante Probleme bei der Erstellung des Entropie-Pools des Gastes nach sich ziehen (s. [VMRNG]). Sollte sich [LLC] als praxistauglich erweisen, so ist anzunehmen, dass eine Abwehr durch eine Anpassung des Hypervisors allein nicht realistisch ist.

Abschließend kann daher festgehalten werden, dass Seitenkanalangriffe durch KVM zumindest nicht erleichtert werden. Insbesondere wird in aller Regel auf eine gemeinsame Nutzung von Speicherseiten verzichtet. Ob die zugrunde liegende x86-Architektur Schwächen aufweist, die Seitenkanalangriffe ermöglichen, wird derzeit aktiv erforscht und kann daher an dieser Stelle nicht abschließend beurteilt werden. Nach aktuellem Stand der Forschung darf somit davon ausgegangen werden, dass die Ermittlung von sicherheitskritischen Informationen, wie z. B. kryptografischen Schlüsseln, unter den in der Praxis zu erwartenden Rahmenbedingungen bei Verwendung von KVM nicht möglich ist.

6.9 Zusammenfassung

Abschließend ist festzuhalten, dass KVM sowohl bei Nutzung von VT-x als auch von AMD-V den Zugriff auf sicherheitskritische Register der CPU entweder gänzlich verwehrt oder aber sinnvoll einschränkt. Dies geschieht zum einen, indem dem Gast im Rahmen der Hardwareinitialisierung eine unkontrollierte Nutzung nur weniger nicht sicherheitskritischer Konfigurationsoptionen gestattet wird. Zum anderen werden darüber hinausgehende Änderungen während der Ausführung des Gastes von KVM abgefangen und in ausreichendem Maße validiert. Eine durch Rekonfiguration der CPU erzielte sicherheitskritische Verhaltensänderung von Speicherverwaltung, CPU-Caches und Virtualisierungsfunktion wird somit verhindert.

Negativ ist jedoch die recht große Anzahl von durch KVM emulierten maschinenspezifischen Registern. Zwar weisen viele dieser Register eine triviale, nicht zu beanstandende Implementierung auf. Einige realisieren jedoch komplexe und für die Virtualisierung nicht zwingend benötigte Funktionen. Auch wenn diesbezüglich keine sicherheitskritischen Schwachstellen festgestellt wurden, so resultiert hieraus doch unnötigerweise eine erheblich vergrößerte Angriffsfläche. Zu benennen sind hier insbesondere die Emulation der Hyper-V-Schnittstelle und der Performance Managment Unit. Zu bemängeln ist, dass sich die entsprechenden Funktionen nicht abschalten lassen und sich somit eine zumindest potenzielle Gefährdung administrativ nicht verhindern lässt.

Darüber hinaus zeigt die Durchreichung von Hardware in einen Gast mittels VT-d oder AMD-I/O einige prinzipielle Designschwächen. So ist die Sicherheit des Hosts unter anderem von der durchgereichten Hardware abhängig. Insbesondere das Durchreichen von Hardware, die nicht für die Virtualisierung konzipiert wurde, ist unter Umständen problematisch. Eine allgemeine abschließende Sicherheitsbewertung ist somit nicht möglich. Vielmehr muss die jeweils konkret verwendete Hardware betrachtet werden, um ihre Sicherheit sinnvoll beurteilen zu können. Darüber hinaus ermöglicht es KVM je nach Konfiguration, Hardware ohne effektive Isolation von Interrupts durchzureichen. Eine derartige Durchreichung führt regelmäßig zu einer signifikanten Gefährdung des Hosts. Ferner ist KVM nicht imstande, virtuelle Maschinen zu migrieren, sofern diese auf durchgereichte Hardware Zugriff haben. Eine Hardwaredurchreichung mithilfe von KVM sollte daher nur dann als vertretbar angesehen werden, sofern

90 Konkret handelt es sich dabei um verschiedene Versionen von GnuPG.

- höchstens ein normaler Schutzbedarf vorliegt und
- moderne virtualisierende Hardware mit IRQ-Remapping-Unterstützung verwendet wird und
- das durchzureichende PCI-Gerät speziell für die Virtualisierung entworfen wurde (SR-IOV) und
- eine sinnvolle Updatestrategie ohne die Notwendigkeit der Live-Migration existiert.

Anderenfalls – insbesondere bei hohem oder sehr hohem Schutzbedarf – sollte von der Verwendung der Hardwaredurchreichung mit KVM abgesehen werden.

Ferner muss regelmäßig davon ausgegangen werden, dass virtuelle Maschinen, die durch KVM auf demselben Host betrieben werden, miteinander kommunizieren können. Zwar kann die Kommunikation über die üblichen Kommunikationskanäle, wie z. B. über die Netzwerkanbindung, zwischen Gästen unterbunden werden. Der Informationsaustausch über versteckte Kommunikationskanäle ist jedoch in der Praxis nicht gänzlich zu verhindern. Ursache hierfür ist die gemeinsame Nutzung von Ressourcen, die der Virtualisierung praktisch inhärent ist. Es ist daher festzuhalten, dass virtuelle Maschinen, die von KVM auf demselben Host ausgeführt werden, kooperativ Informationen austauschen können. Der hierbei zu erzielende Datendurchsatz ist aufgrund der relativ jungen Forschung nicht mit Sicherheit abzuschätzen. Eine effektive und praxistaugliche Abwehr existiert nicht.

Die darüber hinausgehende Fragestellung, ob ein Gast Informationen zur Datenverarbeitung eines anderen Gastes auch ohne dessen Kooperation ermitteln kann, kann nicht abschließend geklärt werden. Der Grund hierfür ist die zwar recht aktive, aber noch junge Forschung. Festzuhalten ist jedoch, dass die Nutzung von Kernel Samepage Merging Seitenkanalangriffe mit geringem Aufwand ermöglicht. Von einem Einsatz des Kernel Samepage Merging ist daher in jedem Fall abzuraten. Weitere Auffälligkeiten, die Seitenkanalangriffe im Zusammenhang mit KVM begünstigen, sind nicht erkennbar. Nach dem aktuellen Stand der Forschung darf somit davon ausgegangen werden, dass die Ermittlung von sicherheitskritischen Informationen (wie z. B. kryptografischen Schlüsseln) unter den in der Praxis zu erwartenden Rahmenbedingungen bei Verwendung von KVM nicht möglich ist.

7 Sicherheitsanalyse von libvirt

7.1 Zielsetzung

Für die effektive Virtualisierung mit QEMU ist die einfache netzwerkbasierte Verwaltung der Gäste wesentlich. Der De-facto-Standard hierfür ist libvirt. Es steuert QEMU, das die Gäste ausführt (s. Abschnitt 3.6). Hierdurch ist es libvirt möglich, grundlegende Anweisungen an QEMU zu geben, wie z. B. die Ausführung eines Gastes zu beenden. Auch kann libvirt prinzipiell über QEMU auf interne Zustände der Gäste zugreifen. Aus diesem Grund ist es von entscheidender Bedeutung für die Sicherheit der Gäste, die entsprechenden Netzwerkschnittstellen und die Netzwerkkommunikation von libvirt vor unbefugten Zugriffen zu schützen.

Darüber hinaus unterstützt libvirt eine Reihe von Schutzmaßnahmen zur Härtung des Hosts. Grundsätzlich kann QEMU Schwachstellen aufweisen, die einen Ausbruch aus einem Gast ermöglichen. Die Härtungsmaßnahmen sollen die negativen Folgen eines derartigen Ausbruchs reduzieren oder abwehren. Konkret handelt es sich hierbei um Nutzung von SELinux mithilfe von sVirt und der cgroups.

Ferner ermöglicht libvirt ein Auditing, mit dem sowohl der Lebenszyklus der Gäste als auch Änderungen an der libvirt-Konfiguration des Hosts protokolliert werden können. Hierdurch können unbefugte Manipulationen potenziell erkannt werden.

Wesentliche Fragestellungen sind daher:

- Wie nutzt libvirt cgroups zur Härtung des Systems?
- Wie nutzt libvirt SELinux zur Härtung des Systems?
- Wie kann ein sicherer Fernzugriff auf den Host mithilfe von libvirt realisiert werden?
- Wie realisiert libvirt die Auditierung des Systems?

Die Untersuchung basiert auf dem in Abschnitt 4 beschriebenen Testsystem und wird im Kontext des Bedrohungsszenarios durchgeführt, das in Abschnitt 2.1 dargestellt wird. Die von libvirt implementierten Verfahren werden in Hinblick auf ihre grundsätzliche Funktionsweise hin analysiert. Eine vollständige Schwachstellenanalyse einschließlich einer Untersuchung des Quelltextes wird nicht durchgeführt.

7.2 cgroups

7.2.1 Allgemein

cgroups ist das seit der Version 2.6.24 vorhandene Ressourcenmanagement des Linux-Kernels. Ursprünglich als "process container" bezeichnet, können die Control-Groups (cgroups) Ressourcen (Arbeitsspeicher, CPU, I/O) limitieren, priorisieren, zählen (für Abrechnungszwecke) und isolieren.

Wenngleich cgroups selten genutzt werden, sind sie beim Einsatz etwa von KVM-Virtualisierung sehr interessant und können von libvirt ohne administrative Unterstützung verwendet werden. Mit cgroups lassen sich die Ressourcen eines virtuellen Gastes beschränken oder gegenüber anderen Gästen priorisieren.

cgroups bieten hierfür mehrere Subsysteme (Controller) für die unterschiedlichen Ressourcen an. CentOS 7 nutzt:

- cpuset – Mit diesem Controller können CPUs und Speicherknoten (bei einer NUMA-Architektur) einzelnen Prozessen zugewiesen werden. Dies ist auf einem NUMA-System sinnvoll, damit der Scheduler die Prozesse nicht unzweckmäßig verteilt.
- cpu – Mit diesem Controller kann der Zugriff auf die CPU verwaltet werden und zum Beispiel auf 20 % reduziert werden.

- cpuacct – Dieser Controller erlaubt die Überwachung der Nutzung der CPU.
- memory – Dieser Controller erlaubt die Überwachung und Beschränkung des verfügbaren Arbeitspeichers einer Task-Gruppe.
- devices – Dieser Controller erlaubt oder verbietet den Zugriff auf einzelne Geräte.
- freezer – Dieser Controller erlaubt das „Einfrieren“ oder „Auftauen“ einzelner Tasks.
- net_cls – Dieser Controller versieht den Netzwerkverkehr mit einer Class-ID, sodass der Netzwerkverkehr anschließend mithilfe der Traffic Control (tc) verwaltet werden kann. Dies erlaubt sowohl eine Priorisierung (QoS) als auch eine Beschränkung (Policying) des Netzwerkverkehrs.
- blkio – Dieser Controller überwacht, beschränkt und verteilt den Zugriff auf die Blockgeräte, wie z. B. Festplatten.
- perf_event – Dieser Controller erlaubt die Überwachung der Task-Gruppen mit dem perf-Werkzeug.
- hugetlb – Dieser Controller beschränkt die Anzahl der HugeTLB, die von einer Task-Gruppe genutzt werden.

Die statische Verwaltung der Task-Gruppen, der Controller und ihrer Konfiguration erfolgt in der Datei /etc/cgconfig.conf. Die Zuweisung der einzelnen Tasks zu den Task-Gruppen erfolgt manuell. Um die Tasks automatisch einzelnen Gruppen zuzuordnen, kann der cgroup Rules Daemon genutzt werden. Bei modernen Distributionen ist diese Funktion jedoch in den Systemd integriert.

7.2.2 cgroup-Nutzung durch libvirt

Der libvirt-Daemon erzeugt gemeinsam mit dem Systemd die folgende Hierarchie unterhalb von /sys/fs/cgroup/<controller>/:

```
$ROOT
 |
 +- system.slice
 |  |
 |  +- libvirtd.service
 |
 +- machine.slice
    |
    +- machine-qemu\x2dID1\x2dvm1name.scope
    |  |
    |  +- emulator
    |  +- vcpu0
    |  +- vcpu1
    +- machine-qemu\x2dID2\x2dvm2name.scope
    |  |
```

Die ID1 und ID2 sind Nummern der virtuellen Gäste vm1name und vm2name die von libvirt intern genutzt werden. Die Verwaltung der Parameter für die einzelnen Controller erfolgt dann zum Beispiel mit dem Kommando virsh:

```
# virsh blkiotune debian8
weight       : 1000
device_weight :
```

```
device_read_iops_sec:
device_write_iops_sec:
device_read_bytes_sec:
device_write_bytes_sec:
# virsh blkiotune debian8 --weight 500
```

Nicht alle verfügbaren Parameter können über libvirt verwaltet werden, aber die, die verwaltet werden können, sind ausreichend für den Kontext der Virtualisierung. Jeder virtuelle Gast wird in einer eigenen getrennten Task-Gruppe verwaltet. Dabei wird jeder Gast durch libvirt automatisch mit individuellen Werten versehen. Insbesondere das Subsystem Devices wird für den Gast auf die folgenden Geräte eingeschränkt:

```
c 136:* rw        - PTYs
c 1:3 rw - /dev/null
c 1:7 rw - /dev/full
c 1:5 rw - /dev/zero
c 1:8 rw - /dev/random
c 1:9 rw - /dev/urandom
c 5:2 rw - /dev/ptmx  PTY master multiplex
c 10:232 rw       - /dev/kvm
c 253:0 rw        - /dev/rtc0
c 10:228 rw       - /dev/hpet
c 10:196 rw       - /dev/vfio/vfio
```

Der Zugriff auf sämtliche weiteren Geräte wird einem Gast nicht erlaubt. Werden explizit Geräte des Hosts (z. B. eine Festplattenpartition) über die XML-Konfiguration an den Gast durchgereicht, so wird die Liste von dem libvirt-Daemon beim Start des Gastes entsprechend automatisch erweitert. Ein Angreifer, der einen Gast kompromittiert, hat jedoch nicht die Möglichkeit, selbst die Liste zu erweitern.

7.2.3 Manuelle Anpassung der cgroups

Um mehrere Gäste gemeinsamen Beschränkungen zu unterwerfen, können diese in der Hierarchie zusammengefasst werden. So können Gäste für Testzwecke einfach von den Produktionssystemen getrennt verwaltet werden. Hierzu bietet libvirt die Möglichkeit, einzelnen Gästen in der XML-Konfiguration eine gemeinsame Partition in der cgroup-Hierarchie zuzuweisen:

```
<resource>
  <partition>/machine/testing</partition>
</resource>
```

Die Systeme werden in einem gemeinsamen Unterverzeichnis /sys/fs/cgroup/<controller>/machine.slice/machine-testing.slice/ verwaltet. Dabei können dann in dem übergeordneten Verzeichnis die Beschränkungen des einzelnen Controllers für die gesamte Task-Gruppe festgelegt werden. Diese Beschränkungen vererben sich dann auf die Task-Gruppe.

7.2.4 Bewertung der cgroup-Funktionalität

Die libvirt-Bibliothek nutzt die cgroup-Funktionalität, sofern diese vorhanden ist, und weist jedem Gast eine eigene Task-Gruppe zu. Das Devices-Subsystem wird von libvirt automatisch so konfiguriert, dass der entsprechende Task nur auf die oben aufgeführten Geräte zugreifen darf. Explizit durchgereichte Geräte (z. B. Festplattenpartitionen) werden automatisch in die Liste aufgenommen. Der Zugriff auf alle weiteren Geräte ist verboten. Die cgroups können von dem Anwender genutzt werden, um die Ressourcen zu beschränken, die die Gäste nutzen dürfen. Damit kann Denial-of-Service-Situationen durch einzelne Gäste vorgebeugt werden. libvirt versieht die Gäste jedoch nicht standardmäßig mit Maximalwerten. Allerdings unterstützt libvirt die Konfiguration entsprechender Werte.

7.3 Fernzugriff

7.3.1 Allgemein

libvirt unterstützt den Zugriff aus der Ferne. Die Fernsteuerung erlaubt die Erzeugung und Verwaltung der Gäste und den Zugriff auf die Konsole mit mindestens den Werkzeugen virsh, virt-manager und virt-viewer.

Hierzu werden mehrere Kommunikationswege angeboten. Praktisch relevant sind jedoch nur die folgenden drei Wege. Die alternativen Varianten bieten entweder keine Sicherheit oder weisen keine hohe Verbreitung auf.

Analysiert werden der Zugriff via SSH, via TLS und via SASL/GSSAPI und Kerberos.

7.3.2 Fernzugriff via SSH

Standardmäßig wird ein Zugriff mithilfe von SSH ohne eine Anpassung des Systems unterstützt, sofern der Benutzer Root-Rechte hat. Wenn ein unprivilegierter Benutzer verwendet werden soll, muss er Mitglied in der Gruppe libvirt sein. Die Authentifizierung des Benutzers erfolgt über das PolicyKit. Die Verwendung alternativer UNIX-Gruppen für die Authentifizierung ist möglich.[91] Die Anmeldung kann durch ein Kennwort oder sicherer durch ein SSH-Schlüsselpaar erfolgen. Problematisch ist – insbesondere bei der Verwendung des Benutzers root –, dass gleichzeitig auch ein Vollzugriff auf der Kommandozeile möglich ist, der nicht trivial eingeschränkt werden kann.

7.3.3 Fernzugriff via TLS

Durch TLS erfolgt die Kommunikation direkt mit dem libvirt-Daemon. Ein Kommandozeilenzugriff ist nicht möglich. Jedoch muss dieser Zugriff konfiguriert werden.[92] Dazu ist zunächst eine Zertifikatsautorität zu erstellen. Mit dieser werden Zertifikate für den Host und für den Client ausgestellt. Diese Zertifikate müssen auf den Host und die Clients verteilt werden, die für die Fernverwaltung eingesetzt werden. Anschließend ist der libvirt-Daemon so zu konfigurieren, dass er direkt Netzwerkverbindungen entgegennimmt und den Client-Zertifikaten vertraut. Hierzu wird eine Whitelist im libvirt-Daemon hinterlegt. Jeder Client, der über ein vertrauenswürdiges Zertifikat und den passenden privaten Schlüssel verfügt, ist nun in der Lage, die Gäste zu verwalten. Der private Schlüssel kann zusätzlich durch eine Passphrase geschützt werden.

91 https://wiki.libvirt.org/page/SSHPolicyKitSetup
92 http://www.ibm.com/support/knowledgecenter/linuxonibm/liaat/liaatkvmsecsrmtls.htm

7.3.4 Fernzugriff via SASL/GSSAPI und Kerberos

Die Verbindung via TCP auf Port 16509 (Klartext) nutzt standardmäßig SASL für die Authentifizierung und die optionale Verschlüsselung. Dabei ist im Ausgangszustand der SASL-Mechanismus DIGEST-MD5 eingestellt. Er erwartet eine lokale Benutzerdatenbank. Alternativ kann der Mechanismus auf GSSAPI umgestellt und anschließend Kerberos genutzt werden. Der libvirt-Daemon erwartet hierfür einen eigenen Principal in der Kerberos-Datenbank libvirt/<host>@REALM. Sobald Kerberos eingerichtet wurde, kann die Gruppe der erlaubten Anwender über den Parameter sasl_allowed_username_list in der Konfigurationsdatei des libvirt-Daemons eingeschränkt werden. Standardmäßig darf jeder Principal mit einem gültigen Service-Granting-Ticket auf den libvirt-Daemon zugreifen und die Gäste verwalten.

7.3.5 Beurteilung der Verwaltung des Fernzugriffs

Alle drei Varianten stellen einen authentifizierten und verschlüsselten Fernzugriff zur Verfügung. Jedoch unterscheiden sie sich in dem Aufwand für die Konfiguration und in den Möglichkeiten, die sie einem Administrator bieten. Hier ist der Zugriff via SSH am einfachsten umzusetzen. Jedoch erhält der Anwender nicht nur die Möglichkeit, die Gäste zu verwalten, sondern er erhält vollständigen Kommandozeilenzugriff auf den Host. Speziell bei Nutzung des Benutzers root ist dies als sehr kritisch einzustufen. Die Nutzung des Zugriffs via TLS ist diesbezüglich sicherer. Die Kommunikation erfolgt lediglich mit dem libvirt-Daemon. Die Gruppe der Anwender kann über eine zertifikatsbasierte Whitelist eingeschränkt werden. Die Variante via SASL/GSSAPI und Kerberos bietet ebenso eine sichere Authentifizierung und eine Whitelist-Möglichkeit. Jedoch ist hier sicherzustellen, dass via Kerberos nicht auch ein Zugriff auf der Kommandozeile mithilfe der SSH möglich ist. Ansonsten unterscheidet sich diese Variante nicht von der ersten Variante mit der Kommunikation über SSH.

7.4 Auditing

7.4.1 Allgemein

Für die Sicherheit eines Systems ist neben der Durchführung der erforderlichen Härtungsmaßnahmen auch eine wirkungsvolle Überwachung erforderlich. Nur eine entsprechende Überwachung erlaubt im Verdachtsfall eine Analyse. Moderne Linux-Distributionen bieten mit dem Audit-System eine feingranulare Überwachung der Vorgänge. Dieses System besteht aus einer Kernel-Komponente und einem Userspace-Daemon. Die Kernel-Komponente erzeugt die Meldungen, die von dem Daemon in Protokollen gespeichert werden. Weitere Werkzeuge im Userspace erlauben die Suche, Analyse und die Erstellung von Berichten. CentOS 7 aktiviert standardmäßig das Auditing im Linux-Kernel und installiert ebenfalls den Audit-Daemon. Dieser ist auch für die Protokollierung der SELinux-Ereignisse verantwortlich.

7.4.2 libvirt-Audit-Unterstützung

libvirt verfügt über eine eigene Audit-Komponente. Diese kann ihre eigenen Meldungen erzeugen. Das Auditing wird in der Datei /etc/libvirt/libvirtd.conf konfiguriert. Standardmäßig ist es auf CentOS 7 aktiviert, sofern das Auditing auch im Host aktiv ist. Dies ist bei CentOS 7 standardmäßig der Fall.

libvirt generiert drei verschiedene Audit-Nachrichten:

- VIRT_CONTROL – Lifecycle (Shutdown/Start) von Gästen
- VIRT_RESOURCE – Änderungen einer Ressource (CPU, Speicher) eines Gastes
- VIRT_MACHINE_ID – Zuweisung eines SELinux-Labels an einen Gast

Für die Analyse der Protokolle steht der Befehl auvirt zur Verfügung. Er kann die Nachrichten anzeigen. Hierbei werden zunächst nur der Start- und der Stopp-Zeitpunkt des Gastes dargestellt. Mit der Option --vm vmname kann die Ausgabe auf einen bestimmten Gast beschränkt werden. Die Option --all-events gibt ausführlichere Informationen aus. So wird auch der Zugriff auf Ressourcen dargestellt.

Diese Unterstützung erlaubt es, auch im Nachhinein, den Lebenszyklus und die genutzten Ressourcen eines Gastes darzustellen.

7.4.3 Spezifische Überwachung

Das Audit-System kann auch genutzt werden, um den Zugriff auf zentrale Konfigurationsdateien der libvirt/KVM-Infrastruktur zu überwachen und deren Änderung zu protokollieren. Eine umfassende Auflistung entsprechender Regeln gibt [KVMS]. Diese müssen jedoch manuell konfiguriert werden und stehen standardmäßig nicht zur Verfügung.

7.4.4 Beurteilung der Audit-Unterstützung

Die Audit-Unterstützung erlaubt bereits in ihrer Standardkonfiguration eine lückenlose Protokollierung des Lebenszyklus der Gäste und der Ressourcen, auf die diese Gäste Zugriff erhalten haben. Mit einer manuellen Konfiguration des Audit-Systems durch eigene Regeln (siehe letzter Abschnitt) kann die Überwachung erweitert werden. In diesem Fall ist jedoch sicherzustellen, dass die Audit-Protokolle sicher gespeichert werden. Standardmäßig darf der Benutzer root diese Protokolle lesen, verändern und löschen. Er kann auch die Regeln für die Audit-Überwachung anpassen oder deaktivieren.

7.5 sVirt

7.5.1 Allgemein

Bei der Verwendung des Hypervisors KVM werden die einzelnen Gäste auf einem Host durch getrennte QEMU-Prozesse ausgeführt. Daher erfolgt deren Trennung und Rechteverwaltung wie bei jedem anderen Linux-Prozess. Das Linux-System prüft entsprechend der Discretionary Access Control (DAC), ob der Besitzer eines Prozesses auf eine bestimmte Ressource zugreifen darf. Verfügt er über die notwendigen Rechte, wird der Zugriff gewährt. Im gegenteiligen Fall wird der Zugriff abgelehnt. Bei der Verwendung der libvirt-Bibliothek und dem Start der Prozesse durch den libvirt-Daemon werden alle Prozesse mit demselben unprivilegierten Benutzer gestartet. Dieser Benutzer muss daher auf sämtliche Dateien zugreifen dürfen (z. B. Festplattenabbilder), die von den Gästen für ihre Funktion benötigt werden. Das DAC-System unterscheidet hierbei die einzelnen Gäste nicht weiter, da die Prozesse alle mit demselben unprivilegierten Benutzer ausgeführt werden. Sobald ein Angreifer aus einem Gast heraus den QEMU-Prozess kompromittieren kann, der den Gast erzeugt hat, ist er in der Lage, auf alle weiteren QEMU-Prozesse und deren Dateien lesend und schreibend zuzugreifen. Eine mögliche Lösung dieses Problems ist die Verwendung eines individuellen Benutzers für jeden einzelnen Gast. Hierbei ergeben sich jedoch mehrere Anforderungen. So muss der Benutzer bei der Erstellung des Gastes erzeugt werden. Bei dem Start des Gastes muss sichergestellt werden, dass die benötigten Dateien (Festplattenabbilder) durch diesen Benutzer gelesen und geschrieben werden können. Weiteren Benutzern muss der Zugriff verweigert werden.

sVirt löst dieses Problem durch Nutzung der Mandatory Access Control (MAC). Es stellt ein Framework zur Verfügung, das beliebige MAC-Systeme anbinden kann. So unterstützt sVirt aktuell sowohl SELinux als auch AppArmor. Diese Analyse betrachtet die Einbindung von SELinux, weil auf der hier betrachteten Plattform CentOS SELinux zum Einsatz kommt.

Die wesentliche Aufgabe von sVirt ist der Schutz des Hosts und der Gäste vor Angriffen, die durch Sicherheitslücken im Hypervisor möglich werden.

7.5.2 Nutzung von SELinux und sVirt

SELinux versieht jeden Prozess und jede Ressource mit einem Security-Context (Label). Dieser besteht aus einem SELinux-Benutzer, einer SELinux-Rolle, einem SELinux-Typ und einer Sensitivität (MLS) sowie einer Kategorie (MCS). Damit SELinux einem Prozess den Zugriff erlaubt sind der Typ, die Sensitivität und die Kategorie ausschlaggebend. Der Benutzer und die Rolle sind hier irrelevant. Dabei definieren Regeln, wann ein Subjekt auf ein Objekt zugreifen darf. Während bei der Sensitivität und der Kategorie lediglich sichergestellt sein muss, dass das Subjekt über ein Superset der Sensitivität und Kategorien der Ressource verfügt, sind für die Typen explizite Regeln (Type Enforcement) erforderlich. Grundsätzlich ist bei dem Type Enforcement von dem Standard Deny auszugehen. Das heißt, jeder Zugriff muss explizit erlaubt werden. Dabei besitzen dann alle Prozesse, die denselben Typ verwenden, identische Zugriffsrechte. Ein Prozess kann auf alle Ressourcen, die mit demselben Typ gelabelt wurden, mit identischen Rechten zugreifen. Der Typ dient somit zur abstrakten Gruppierung sowohl der Prozesse als auch der Ressourcen.

Damit sVirt verwendet werden kann, muss der Administrator lediglich sicherstellen, dass SELinux installiert wurde und sich im Zustand Enforcing befindet. Standardmäßig aktiviert CentOS 7 sVirt, wenn SELinux aktiv ist. Sobald SELinux global abgeschaltet wird, ist auch die Verwendung von sVirt deaktiviert. Im Zustand Permissive bietet SELinux keinen Schutz. In diesem Zustand werden die Regeln geladen und genutzt. Eine Regelverletzung führt jedoch nur zu einer Protokollierung und nicht zur Ablehnung des Zugriffs.

sVirt kann auch von Hand global deaktiviert werden. Ausschlaggebend ist die Datei /etc/libvirt/qemu.conf. Hier kann das sVirt-Plug-in ausgewählt und konfiguriert werden. Im Folgenden sind die Default-Werte angegeben:

```
security_driver = „selinux“

# Gäste werden durch SELinux mit spezifischem Kontext versehen
security_default_confined = 1

# Für einzelne Gäste kann die Trennung deaktiviert werden
security_require_confined = 0
```

7.5.3 Implementierung von sVirt

Um das Host-System grundsätzlich vor den Gästen zu schützen, verwendet SELinux zunächst eigene Typen für die Verwaltung der libvirt-Prozesse und der dazugehörigen Dateien. In der aktuellen CentOS-Version mit dem SELinux-Modul virt in der Version 1.5.0 sind das die folgenden Typen:

- virtd_t – libvirt-Daemon-Prozess
- virt_etc_t – Konfigurationsdateien; nur lesender Zugriff erlaubt
- virt_etc_rw_t – Konfigurationsdateien; schreibender Zugriff erlaubt
- virt_content_t – Daten für Gäste; schreibender Zugriff erlaubt
- virt_image_t – Gastabbilder inaktiver Gäste; schreibender Zugriff erlaubt
- virt_var_lib_t – variable Daten des Daemons

Die dynamische Erzeugung eines neuen SELinux-Typs ist aufwendig und erfordert den erneuten Bau der vollständigen SELinux-Policy und deren Reload durch den Kernel. Daher erfolgt die Trennung der Gäste nicht durch unterschiedliche Typen, sondern durch unterschiedliche Kategorien. SELinux unterstützt in der aktuellen Version bis zu 500.000 Kategorien, die dynamisch ohne den Reload der Policy erzeugt und auch wieder zerstört werden können. Für den Betrieb der Gäste nutzt sVirt daher für alle Gäste dieselben

Typen. Die Trennung der einzelnen Gäste erfolgt durch die Kategorien. sVirt verwendet die folgenden Typen für den Betrieb der Gäste. Im Folgenden symbolisiert MCS die MCS-Kategorie:

- svirt_t:MCS – der Gast-Prozess
- svirt_image_t:MCS – das Festplattenabbild eines laufenden Gastes. Nur svirt_t- Prozesse mit demselben MCS-Feld dürfen dieses Abbild lesen und schreiben.
- svirt_image_t:s0 – gemeinsam genutzte Festplattenabilder. Alle svirt_t-Prozesse dürfen diese Dateien bzw. Geräte lesen und schreiben.
- svirt_content_t:s0 – Alle svirt_t-Prozesse besitzen lesenden Zugriff auf diese Dateien und Geräte.
- virt_content_t:s0 – Default-Label eines Festplattenabbilds eines nicht aktiven Gastes. Kein svirt_t-Prozess darf auf diese Dateien zugreifen.

Die Erzeugung der MCS-Kategorie kann auf drei Arten erfolgen:

- dynamisch zur Laufzeit durch den libvirt-Daemon
- statisch durch den Anwender in der XML-Datei hinterlegt
- deaktiviert, sodass der Gast nicht in einer eigenen Kategorie läuft

7.5.3.1 Dynamische Erzeugung der MCS-Kategorie von SELinux

Die Default-Einstellung ist die dynamische Erzeugung der MCS-Kategorie durch den libvirt-Daemon. Jeder Gast läuft damit in einer eigenen isolierten und dynamisch erzeugten Kategorie. Die Festplattenabbilder werden vor dem Start und nach dem Beenden des Gastes entsprechend mit anderen Labeln versehen. Die Gäste werden so untereinander und von dem Hypervisor isoliert.

Hierzu wird der initiale Security Context für den Gast-Prozess aus der Datei /etc/selinux/targeted/contexts/virtual_domain_context gelesen. Er lautet bei CentOS 7 system_u:system_r:svirt_t:s0. Der initiale Security Context für die Festplattenabbilder wird aus der Datei /etc/selinux/targeted/contexts/virtual_image_context gelesen. Er lautet: system_u:object_r:svirt_image_t:s0. Sobald der Gast gestartet wird, erzeugt der libvirt-Daemon einen zufälligen MCS-Level, der aus zwei Kategorien besteht, versieht die Datei mit dem entsprechenden Label und startet den Gast-Prozess in dem dazugehörigen Kontext. Der erzeugte MCS-Level wird in der XML-Gastkonfigurationsdatei hinterlegt:

```
<seclabel type='dynamic' model='selinux' relabel='yes'>
  <label>system_u:system_r:svirt_t:s0:c73,c582</label>
<imagelabel>system_u:object_r:svirt_image_t:s0:c73,c582</imagelabel>
</seclabel>
```

Sowohl das genutzte Festplattenabbild als auch der Prozess nutzen den MCS-Label:

```
# ls -lZ /var/lib/libvirt/images/debian8.qcow2
-rw-r--r--. qemu qemu system_u:object_r:svirt_image_t:s0:c73,c582 /var/lib/libvirt/images/debian8.qcow2
# ps -efZ
...
system_u:system_r:svirt_t:s0:c73,c582 qemu 28773   1  0 Dez28 ?       00:00:54 /usr/libexec/qemu-kvm -name debian8 -S -machine ....
...
```

sVirt verfügt auch über Unterstützung für gemeinsam genutzte Festplattenabbilder. Hierzu muss die entsprechende Konfiguration in der XML-Datei um das Attribut <shareable/> erweitert werden:

```
<disk type='file' device='disk'>
   <driver name='qemu' type='qcow2'/>
```

```
    <source file='/var/lib/libvirt/images/share.qcow2'/>
    <target dev='hdb' bus='ide'/>
    <shareable/>
</disk>
```

Die entsprechende Datei erhält dann den Security Context system_u:object_r:svirt_image_t:s0 ohne einen MCS-Label. Aus SELinux-Sicht dürfen sämtliche Gäste auf dieses Festplattenabbild lesend und schreibend zugreifen. Somit erfolgt keine Trennung für gemeinsam genutzte Festplattenabbilder. Auch wenn der Zugriff zusätzlich mit dem Attribut <readonly/> eingeschränkt wird, erlaubt SELinux weiterhin einen lesenden sowie einen schreibenden Zugriff.

Die zusätzliche Angabe eines Base-Labels mit <baselabel> erlaubt auch die Nutzung eigener SELinux-Richtlinien. Die Kategorien werden weiterhin dynamisch erzeugt und gemeinsam mit dem Base-Label für den Betrieb der Gäste genutzt.

7.5.3.2 Statische Erzeugung der MCS-Kategorie von SELinux

Alternativ kann auch der Anwender die zu verwendenden Labels statisch vorgeben. Hierzu wird der Seclabel-Container in der XML-Datei mit den folgenden Werten gefüllt:

```
<seclabel type='static' model='selinux' relabel='yes'>
  <label>system_u:system_r:svirt_t:s0:c500,c501</label>
</seclabel>
```

Wenn der Parameter relabel='yes' nicht gesetzt wird, muss der Anwender die Festplattenabbilder manuell mit dem richtigen Typ und der richtigen Kategorie ausstatten. Hierzu stehen die Befehle chcon und chcat zur Verfügung. Durch die Verwendung eines Supersets an Kategorien für den Prozess sind auch komplexe Setups mit mehreren Festplattenabbildern möglich, die von verschiedenen Gästen gemeinsam genutzt werden.

7.5.3.3 Deaktivierte Unterstützung für den Gast

Die Trennung durch sVirt kann auch für jeden Gast einzeln deaktiviert werden. Hierzu wird als Seclabel in der XML-Konfigurationsdatei der folgende Wert verwendet:

```
<seclabel type='none' model='selinux'/>
```

Der Gast wird dann in dem SELinux-Security-Context system_u_system_r:virtd_t:s0-s0:c0.c1023 betrieben. Dieser Gast hat damit aus Sicht von SELinux Zugriff auf alle weiteren Gäste und auch auf den libvirt-Daemon selbst. Lediglich weitere Bereiche des Host-Systems sind durch SELinux vor einem Missbrauch durch den Gast geschützt.

7.5.4 sVirt und netzwerkbasierte Speicherung

Die Trennung der Gäste durch sVirt und SELinux ist nur lokal auf dem Host gegeben. Die üblicherweise in Kombination mit KVM eingesetzten Protokolle für den Zugriff auf entfernte Speichersysteme wie Ceph und GlusterFS unterstützen SELinux nicht. Lediglich NFSv4.2 kann SELinux-Security-Contexts unterstützen. Daher können sVirt und SELinux nur lokal eine vollständige Trennung der Gäste erreichen. Der Schutz der Dateisysteme und die Trennung der Gäste auf entfernten Speichersystemen muss deswegen durch alternative Methoden sichergestellt werden. Zu diesen Methoden zählen eine sinnvolle Authentifizierung und Autorisierung (s. Abschnitt 9).

7.5.5 Beurteilung der Trennung

In der Praxis ist die dynamische Erzeugung der Label die am häufigsten anzutreffende Konfiguration, da CentOS 7 diese per Default nutzt. Unabhängig hiervon unterscheidet sich die Trennung der Gäste bei dynamischer und statischer Vergabe der Labels nicht, wenn alle Gäste unterschiedliche Labels erhalten. Erhalten zwei Gäste statisch identische Labels, so verfügen sie über identische Zugriffsprivilegien.

Bei aktivierter Trennung via SELinux wird die Domäne svirt_t für die Gäste genutzt. Sie hat nur sehr eingeschränkte Rechte. Die entsprechenden Regeln sind auszugsweise in Anhang D angefügt. Sie beschränken sich auf die folgenden Bereiche:

- Aufbau und Entgegennahme von beliebigen TCP- und UDP-Netzwerkverbindungen
- Erlaubnis, in einer Secure-Boot-Umgebung nichtsignierten Code im Kernel auszuführen (compromise_kernel). Dies ist für PCI-Passthrough erforderlich.
- lesender Zugriff auf Ressourcen mit dem Typ virt_content_t
- lesender Zugriff auf Netzwerkinformationen und Heimatverzeichnisse der Nutzer, sowie allgemeine Dateien in /var
- schreibender Zugriff auf Ressourcen mit dem Typ virt_home_t, svirt_home_t, virt_cache_t, svirt_image_t, svirt_tmp_t, svirt_tmpfs_t, qemu_var_run_t, virt_log_t und virt_var_lib_t.
- ausführender Zugriff auf binäre Hilfsprogramme und die Shell. Diese werden aber weiterhin in der Domäne svirt_t ausgeführt.
- lesender Zugriff auf den Zufallszahlengenerator
- schreibender Zugriff auf die Geräte /dev/ksm, /dev/kvm, VFIO, Infiniband, Terminals
- einstellbar über SELinux-Booleans-Zugriff auf CIFS, NFS, USB, serielle und parallele Schnittstellen

Bei Nutzung von sVirt ist eine sinnvolle Trennung der Gäste gegeben. Das Regelwerk schränkt die erlaubten Zugriffe auf das erforderliche Maß ein. Lediglich die unbedingte Zuweisung der Capability compromise_kernel für den PCI-Passthrough kann als kritisch eingestuft werden. Hier wäre eine optionale Konfiguration über SELinux-Booleans wünschenswert. Jedoch ist die Verwaltung dieser Capability in aktuelleren Kerneln ersatzlos weggefallen, und ein Zugriff wird nun nicht mehr von SELinux abgewiesen.

Die Trennung mehrerer Gäste voneinander durch die dynamische oder auch statische Vergabe der MCS-Label funktioniert und ist wirksam.

Abschließend kann die Aktivierung von SELinux und sVirt empfohlen werden. Insbesondere die dynamische Vergabe von MCS-Labeln erhöht die Sicherheit des Hosts und ist transparent für den Benutzer. Falls es einem Angreifer gelingt, aus dem Hypervisor auszubrechen, ist der Angreifer in dem QEMU-Prozess in dem Security-Context system_u:system_r:svirt_t:s0:MCS gefangen. Ein Zugriff auf andere Gäste oder auch auf den libvirt-Daemon oder kritische Bestandteile des Systems ist dadurch nicht möglich. Ein Aufruf von Befehlen mit erweiterten Privilegien (su, ping etc.) ist ebenfalls nicht möglich.

7.6 Zusammenfassung

Zusammenfassend ist festzuhalten, dass libvirt anhand von cgroups und SELinux, das über sVirt angebunden ist, eine effektive Härtung des Hosts durchführt. Beide Funktionen werden zweckmäßig genutzt, um die Zugriffsberechtigungen des QEMU-Prozesses deutlich einzuschränken. So wird die Menge der Gerätedateien minimiert, auf die ein Gast zugreifen kann. Standardmäßig wird nur ein Zugriff auf wenige, nicht sicherheitskritische Gerätedateien gewährt. Darüber hinaus wird auch der Zugriff auf normale Dateien mithilfe von SELinux stark eingeschränkt. Insbesondere der Zugriff auf lokale gespeicherte Festplattenabbilder wird nur den QEMU-Prozessen gestattet, die diese Festplattenabbilder tatsächlich benötigen, um den jeweiligen Gast ausführen zu können. Festplattenabbilder, die nicht lokal vorgehalten werden, werden jedoch nicht durch libvirt geschützt. Die Umsetzung einer feingranularen Zugriffskontrolle bei Nutzung eines netzwerkbasierten Speicherbackends muss durch dieses realisiert werden. Auch wird eine ungewollte gegenseitige Beeinflussung der QEMU-Prozesse weitgehend verhindert.

Auch wenn QEMU Schwachstellen aufweisen würde, die einen Ausbruch aus dem Gast potenziell ermöglichen, verhindern die aufgeführten Schutzmaßnahmen doch weitgehend eine Kompromittierung des Hosts.

Die Nutzung sowohl von cgroups als auch von SELinux erfolgt durch libvirt standardmäßig, sofern die Funktionen auf dem System zur Verfügung stehen. Auf dem untersuchten CentOS-7-System ist dies der Fall. Eine diesbezügliche Konfiguration ist daher nicht erforderlich.

Darüber hinaus stellt libvirt zweckmäßige Funktionen zur Auditierung des Systems zur Verfügung. Hierdurch lässt sich insbesondere der Lebenszyklus der Gäste einwandfrei protokollieren. Zusätzlich kann das System genutzt werden, um nicht privilegierte Änderungen wesentlicher Konfigurationen der Virtualisierungsumgebung festzustellen.

Der netzwerkbasierte Zugriff auf libvirt kann auf mehreren Wegen erfolgen. Die wesentlichen Methoden sind hier der Zugang mithilfe von SSH, TLS oder SASL/GSSAPI. Zwar bieten alle drei eine prinzipiell sichere Anmeldung und eine lückenlose Verschlüsselung und Signierung der Kommunikation. Bei Anmeldung mithilfe von SSH steht dem Nutzer aber grundsätzlich ein Zugriff auf die Shell des Hosts zur Verfügung. Dies bedeutet in der Regel, dass dem Nutzer eine über das notwendige Maß hinausgehende Menge an Berechtigungen zugesprochen werden. TLS und SASL/GSSAPI teilen diese Eigenschaft nicht und sind daher insbesondere in komplexen Virtualisierungsumgebungen zu bevorzugen.

8 Sicherheitsanalyse der Netzwerkanbindung

8.1 Zielsetzung

Die im Folgenden aufgeführte Analyse behandelt die Sicherheit der Netzwerkkommunikation von Gästen in den Testumgebungen, die in Kapitel 4 spezifiziert wurden. Hierzu werden verschiedene Konfigurationen im Kontext der Rahmenbedingungen untersucht, die in Kapitel 2 beschrieben sind. Es wird somit insbesondere angenommen, dass ein Angreifer die volle Kontrolle über ein Gast-Betriebssystem erlangt hat.

Die folgenden Fragestellungen werden hierbei untersucht:

- Ist es einem Gast möglich, unberechtigt Netzwerkverkehr – d. h. Verkehr, der nicht an ihn adressiert ist und dessen Verarbeitung nicht aus technischen Gründen sinnvoll oder erforderlich ist – mitzulesen, zu manipulieren oder zu unterbinden?
- Ist es einem Gast möglich, unberechtigt mit Diensten der Virtualisierungsumgebung zu kommunizieren, d. h. mit solchen Diensten, zu dem ihm nicht explizit der Zugriff gestattet ist?
- Ist es einem Gast möglich, auf unberechtigte Weise – d. h. mit anderen Vermittlungsprotokollen als IPv4 und IPv6 und den hierzu eingesetzten Hilfsprotokollen wie ARP – mit anderen Systemen zu kommunizieren?

Nicht untersucht wird hierbei jedoch, ob es dem Gast möglich ist, eine Überlastsituation im Netzwerk herbeizuführen, die die Kommunikation anderer Systeme signifikant stört oder gänzlich unterbindet.[93] Des Weiteren wird die Absicherung der Kommunikation der Gäste mit externen Systemen über die Grenzen des Virtualisierungssystems hinaus nicht betrachtet.[94] Eine Sicherheitsanalyse der Kommunikation zwischen Diensten des Virtualisierungssystems über die oben beschriebenen Fragestellungen hinaus findet ferner in Kapitel 7 und 10 statt. Eine entsprechende gastunabhängige Kommunikation zwischen Virtualisierungsdiensten bleibt daher an dieser Stelle unberücksichtigt.

Das Ziel dieser Analyse ist es, möglichst allgemeingültige Aussagen über entsprechende Komponenten zu treffen, die auf zukünftige oder bereits bestehende Virtualisierungsumgebungen übertragbar sind. Aufgrund der Vielzahl an verschiedenen Konfigurationsmöglichkeiten der Netzwerkanbindung ist es jedoch nicht möglich, eine erschöpfende Untersuchung aller Einsatzszenarien vorzunehmen.

8.2 Allgemeine Sicherheitsaspekte

Welche Aspekte zu berücksichtigen sind, um für eine sichere Netzwerkanbindung einer Virtualisierungsumgebung zu sorgen, hängt typischerweise stark von der gewählten Anbindungsmethode und der konkreten Konfiguration ab. Es gibt jedoch eine Reihe allgemein zu berücksichtigender Sicherheitsaspekte, die im Folgenden erläutert werden.

Eine erhebliche Gefahr entsteht, wenn eine IP-Adresse des Gastnetzwerks an eine Netzwerkschnittstelle des Hosts vergeben wird. Besitzt der Host eine solche IP-Adresse, ist er somit nicht mehr nur eine passive Komponente im Sinne einer reinen Paketweiterleitung auf Layer 2, sondern aktiver Teilnehmer des Gast-Netzwerks auf Layer 3. In dem Fall ist es einem Gast prinzipiell möglich, eine IP-Kommunikation mit dem Host aufzunehmen. Hierbei besteht unter anderem die Gefahr, dass der Gast Zugang zu Managementdiensten der Virtualisierungsumgebung erhält. Auch wenn diese Dienste typischerweise eine Benutzerauthentifizierung durchführen, so besteht doch die Gefahr, dass der Gast auf entsprechende Dienste des Hosts zugreift.

93 Eine solche Untersuchung ist in ganz erheblichem Maße von der Hardware abhängig, die zur Virtualisierung und zur Netzanbindung eingesetzt wird. Hierzu sind allgemeingültige Aussagen nur sehr bedingt möglich.

94 Hier liegt die Annahme zugrunde, dass die Sicherung einer entsprechenden Kommunikation Aufgabe des Gastes und nicht der Virtualisierungsumgebung ist. Beispielsweise ist es die Aufgabe des Gastes, die Übermittlung sensitiver Informationen über das Internet zu verschlüsseln.

Eine unter Umständen noch schwerwiegendere Folge der Bindung einer IP-Adresse des Gast-Netzwerks an den Host kann jedoch sein, dass der Gast auf das Management- oder Storage-Netzwerk zugreift. Ein solcher Zugriff ist möglich, sofern der Host IP-Pakete weiterleitet (IP-Forwarding). Obwohl dies typischerweise die Aufgabe eines Routers ist, so übernimmt der Host jedoch auch regelmäßig entsprechende Funktionen. Dies ist beispielsweise bei Nutzung von libvirt-Netzwerken (s. Abschnitt 3.6.4.1) notwendig. Erschwerend kommt hinzu, dass im Zuge der Installation von libvirt typischerweise ein solches Netzwerk eingerichtet und hierbei das IP-Forwarding aktiviert wird. Bei Vergabe einer entsprechenden IP-Adresse an den Host besteht daher die erhebliche Gefahr, dass es dem Gast ermöglicht wird, aus dem Gast-Netzwerk auszubrechen.

Ebenso kann durch aktives IP-Forwarding die Trennung zwischen Management- und Storage-Netzwerk aufgehoben werden, da der Host die an den entsprechenden Netzwerkschnittstellen eintreffenden Pakete weiterleitet.

Jedoch besteht im Fall von aktivem IP-Forwarding auch dann eine Gefahr, wenn keine IP-Adresse des Gast-Netzwerks an den Host gebunden ist. Dies liegt an der grundlegenden Funktionsweise von IP-Forwarding des Linux-Kernels. Ist es aktiv, so wird ein Paket weitergeleitet, das auf Layer 2 an den Host, auf Layer 3 jedoch an einen anderen Netzwerkteilnehmer adressiert ist. Dies hat zur Folge, dass der Schnittstelle, auf der ein Paket empfangen wird, nicht zwingend eine IP-Adresse zugewiesen sein muss, damit Pakete an eine andere Netzwerkschnittstelle weitergeleitet werden. Tatsächlich muss der Host lediglich auf Layer 2 mittels seiner MAC-Adresse adressierbar sein. Letzteres ist jedoch quasi immer gegeben. In einem solchen Fall ist es einem Gast möglich, IP-Pakete über den Host an Netzwerkkomponenten im Management- oder Storage-Netzwerk zu versenden, sofern bei aktivem IP-Forwarding keine Härtungsmaßnahmen auf dem Host ergriffen wurden.[95]

Mögliche Härtungsmaßnahmen sind die Aktivierung von Filterregeln, sofern dies durch die jeweilige Anbindungsmethode des Netzwerks und der Managementumgebung unterstützt wird. libvirt unterstützt hierzu eine Reihe vorkonfigurierter Filterregeln, die zur Härtung eingesetzt werden können (s. Abschnitt 3.6.4.3). Alternativ können entsprechende Filter mit iptables oder ebtables eingerichtet werden. Hierbei kann es jedoch zu unerwarteten Wechselwirkungen mit der Managementumgebung kommen, die Funktionsstörungen oder eine negative Beeinflussung der Sicherheit zur Folge haben können. Die Installation komplexer Filterregeln mithilfe von iptables oder ebtables unter Umgehung der Managementumgebung sollte daher nur in Ausnahmefällen vorgenommen werden.

Eine weitere Härtungsmaßnahme, die insbesondere vor dem oben beschriebenen Weiterleitungsvorgang ohne Bindung an eine entsprechende IP-Adresse schützt, ist der Reverse-Path-Filter (rp-filter, s. [IP-SYSCTL]).[96] Diese Funktion des Linux-Kernels unterbindet, sofern sie aktiv ist, eine Weiterleitung von IP-Paketen, die sonst an Netzwerkschnittstellen ohne IP-Adresse empfangen werden könnten.[97]

Eine weitere Härtungsmaßnahme ist die selektive Abschaltung des IP-Forwardings. Sie ermöglicht es, die Weiterleitung für ausgewählte Netzwerkschnittstellen zu untersagen, für die übrigen Schnittstellen des Hosts aber aktiviert zu lassen. Dies empfiehlt sich in der Regel für Netzwerkschnittstellen, die mit der Kommunikation von Gästen in Berührung kommen.

Darüber hinaus ist zu beachten, dass Netzwerkdienste, die auf einem Linux-System ausgeführt werden, in der Regel über alle Netzwerkschnittstellen angesprochen werden können. Insbesondere kann ein Dienst, der sich explizit an eine IP-Adresse gebunden hat, auch über Netzwerkschnittstellen kontaktiert werden, die selbst keine IP-Adresse des entsprechenden IP-Netzwerks zugewiesen bekommen haben.[98] Analog ist auch das Versenden von ICMP-Nachrichten möglich, z. B. eines Pings oder von ARP-Anfragen. Durch entsprechende Anfragen kann der Gast unter Umständen die Existenz von Verwaltungsnetzen erkennen

95 Unter der Annahme, dass die Netzwerkkomponenten eine gegebenenfalls erzeugte Antwort des kontaktierten Systems nicht an den Gast zurückleiten, ist eine solche Kommunikation jedoch lediglich unidirektional.

96 Hierzu muss der Filter in den Modus *strict*, nicht jedoch in den Modus *loose* versetzt werden.

97 In den untersuchten Testumgebungen, die auf CentOS 7 basieren, befindet sich dieser Filter standardmäßig im Modus *strict*. Es existiert jedoch eine Reihe von Linux-Distributionen, bei denen dies nicht der Fall ist.

98 Eine Ausnahme bilden hier jedoch Dienste, die sich an nicht routbare IP-Adressen, wie z. B. 127.0.0.1, gebunden haben.

und Adressinformationen über diese ermitteln. Von einem sicherheitstechnischen Standpunkt aus sollten sowohl MAC- als auch IP-Adressen daher als dem Host und nicht seinen Netzwerkschnittstellen zugeordnet betrachtet werden.

8.3 Durchführung

Um die in Abschnitt 8.1 aufgeführten Fragestellungen zu überprüfen, ist in Tabelle 29 ein Katalog aus Testfällen beschrieben. Dieser wird auf die im Folgenden beschriebenen Konfigurationen angewandt. Die Testresultate werden als bestanden oder nicht bestanden klassifiziert. Sofern dies für die Bewertung der jeweiligen Konfiguration zweckdienlich ist, werden darüber hinaus weitere Tests durchgeführt, die im Rahmen der Konfigurationsbeschreibung erläutert werden.

Anhand der Testfälle 1 und 2 wird sichergestellt, dass das Gastsystem grundsätzlich in der Lage ist, mittels IPv4 oder IPv6 zu kommunizieren. Durch die Testfälle 3 bis 5 wird überprüft, ob eine strikte Trennung zwischen den verschiedenen Netzwerken in der Virtualisierungsumgebung gegeben ist. Mit Testfall 6 wird sichergestellt, dass Gäste ausschließlich mittels IPv4 oder IPv6 und den hierzu genutzten Protokollen Ethernet, ARP und RARP kommunizieren können. Die Testfälle 7 bis 12 überprüfen typische Angriffsszenarien, die die Sicherheit von IP-Netzwerken negativ beeinflussen können.

Ausgehend von den Ergebnissen der Tests werden Härtungsmaßnahmen für die jeweilige Konfiguration der Virtualisierungsumgebung vorgeschlagen. Ein erneuter Test überprüft gegebenenfalls die Effektivität dieser Maßnahmen.

	Testbezeichnung	Beschreibung des Tests
1	Vorhandene Konnektivität mittels IPv4	Der Test prüft, ob ein Gast grundsätzlich über IPv4 sowohl mit anderen Gästen als auch mit externen Systemen kommunizieren kann. Hierzu werden mit dem Systembefehl ping ICMP-Pings sowohl an einen anderen Gast als auch an ein externes System gesendet. Der Test ist nicht bestanden, wenn der Gast kein ICMP-Pong von dem anderen Gast oder dem externen System empfängt.
2	Vorhandene Konnektivität mittels IPv6	Der Test prüft, ob ein Gast grundsätzlich über IPv6 sowohl mit anderen Gästen als auch mit externen Systemen kommunizieren kann. Für die Durchführung wird der Systembefehl ping6 und entsprechend das Protokoll ICMPv6 genutzt. Ansonsten ist der Test analog zu Test 1.
3	Fehlende Trennung vom Management-Netzwerk	Es wird getestet, ob eine Netzwerkverbindung zwischen einem Gast und dem Management-Netzwerk besteht. Hierzu erzeugt der Gast eine ARP-Anfrage, um eine existierende IP-Adresse im Management-Netzwerk aufzulösen. Wird die Anfrage beantwortet, existiert keine strikte Trennung auf Layer 2. Darüber hinaus wird die Trennung auf Layer 3 überprüft. Ein ICMP-Ping wird an eine existierende IP-Adresse im Management-Netzwerk gerichtet. Als Ziel-MAC-Adresse wird die MAC-Adresse des Hosts verwendet. (Der Host fungiert als Router.) Der Test ist nicht bestanden, wenn ein vom Gast erzeugtes Netzwerkpaket in das Management-Netzwerk gelangt.
4	Fehlende Trennung vom Storage-Netzwerk	Es wird getestet, ob eine Netzwerkverbindung zwischen einem Gast und dem Storage-Netzwerk besteht. Die Testdurchführung erfolgt analog zu Test 3.

	Testbezeichnung	Beschreibung des Tests
5	Erreichbarkeit von Diensten des Hosts durch den Gast	Es wird überprüft, ob ein Gast mit Netzwerkdiensten eines Hosts kommunizieren kann. Hierzu wird getestet, ob der Gast Pakete zu einem Netzwerkdienst des Hosts senden kann, der ihn virtualisiert, oder zu einem fremden Host. Ausgenommen sind hiervon Dienste, die dem Gast explizit zur Verfügung gestellt werden.[99] Der Test ist nicht bestanden, wenn ein vom Gast erzeugtes Paket den Netzwerkdienst eines Hosts erreicht.
6	Kommunikation mittels beliebiger Protokolle über Ethernet	Die zur Kommunikation der Gäste genutzten Vermittlungsprotokolle sind IPv4 und IPv6. Ferner wird für IPv4-Netzwerke ARP benötigt. Darüber hinaus existieren auf Ethernet aufbauende Protokolle, wie z. B. das Spanning Tree Protocol (STP), das Cisco Discovery Protocol (CDP) oder das Protokoll Multi-Protocol Label Switching (MPLS). Ist es einem Gast möglich, beliebige Protokolle auf der Vermittlungsschicht zu nutzen, so sind ihm potenziell verschiedene Angriffe auf die Virtualisierungsumgebung möglich (z. B. Spanning-Tree-Attacks, CDP-Attacks und VLAN-Hopping). Im Rahmen dieses Tests versendet daher der Gast systematisch Ethernet-Pakete mit jeder möglichen Belegung des Feldes Type. Der Test gilt als nicht bestanden, falls ein Ethernet-Paket, dessen Feld Type nicht den Protokollen IPv4, IPv6, ARP oder RARP entspricht, einen anderen Gast erreicht oder den Host verlässt.
7	MAC-Address-Spoofing	Dieser Test überprüft, ob ein Gast Ethernet-Pakete mit einer beliebigen MAC-Absenderadresse versenden kann. Der Test gilt als nicht bestanden, wenn ein Paket mit einer MAC-Absenderadresse, die dem Gast nicht durch die Virtualisierungsumgebung zugewiesen wurde, den Host verlässt oder einen anderen Gast erreicht.
8	IP-Address-Spoofing	Dieser Test überprüft, ob ein Gast IP-Pakete mit einer beliebigen IP-Absenderadresse versenden kann. Der Test gilt als nicht bestanden, wenn ein Paket mit einer IP-Absenderadresse, die dem Gast nicht durch die Virtualisierungsumgebung zugewiesen wurde, den Host verlässt oder einen anderen Gast erreicht.
9	ARP-Spoofing	Dieser Test überprüft, ob ein Gast die Zuordnung zwischen IPv4-Adressen und MAC-Adressen anderer Systeme unberechtigt verändern kann. Der Test gilt als nicht bestanden, wenn ein ARP-Reply den Host verlässt oder einen anderen Gast erreicht und wenn die enthaltene IP-MAC-Adresszuordnung nicht der des versendenden Gastes entspricht.
10	ARP-Flooding	Dieser Test überprüft, ob die Virtualisierungsumgebung ein Überlaufen von MAC-Adresstabellen erkennen und verhindern kann. Hierzu versendet der Gast 1000 Ethernet-Pakete mit verschiedenen MAC-Absenderadressen. Der Test ist nicht bestanden, wenn alle 1000 Ethernet-Pakete den Host verlassen oder eine Bridge auf dem Host erreichen.

99 Eine solche Ausnahme ist beispielsweise ein DHCP-Dienst, den der Host dem Gast zur Verfügung stellt.

	Testbezeichnung	Beschreibung des Tests
11	NDP-Spoofing/RA-Spoofing	Dieser Test überprüft, ob ein Gast beliebige Neighbor-Discovery-Protocol-Nachrichten versenden kann. Der Test ist nicht bestanden, wenn ein Neighbor Advertisement mit einer anderen Link-Layer-Adresse als der des Gastes oder ein Router Advertisement den Host verlässt oder einen anderen Gast erreicht.
12	DHCP-Spoofing	Dieser Test überprüft, ob ein Gast DHCP-Anfragen beantworten kann. Für diesen Test wird ein DHCP-Server auf einem Gast gestartet. Der Test gilt als nicht bestanden, wenn DHCP-Antworten eines Gastes den Host verlassen oder einen anderen Gast erreichen.

Tabelle 29: Testfälle zur Prüfung der Netzwerksicherheit

8.4 libvirt

8.4.1 Linux-Bridge

8.4.1.1 Konfiguration

Auf beiden Hosts der Testumgebung (s. Kapitel 4) wird mit dem folgenden Skript eine Linux-Bridge mit dem Namen br0 erzeugt und die als eth0 bezeichnete physische Schnittstelle des Gast-Netzwerks angehängt. Hierbei ist eth0 aktiv, aber nicht konfiguriert. Weder an br0 noch an eth0 ist eine IPv4- oder IPv6-Adresse gebunden. Ferner ist für diese Schnittstelle die IPv6-Autokonfiguration deaktiviert.

```
#!/bin/bash
brctl addbr br0
brctl addif br0 eth0
ip link set up dev br0
```

Im Anschluss wird mit libvirt und der folgenden Beschreibung ein Netzwerk erstellt, das auf der Linux-Bridge br0 basiert.

```
<network>
        <name>bridgenet</name>
        <forward mode="bridge" />
        <bridge name="br0" />
</network>
```

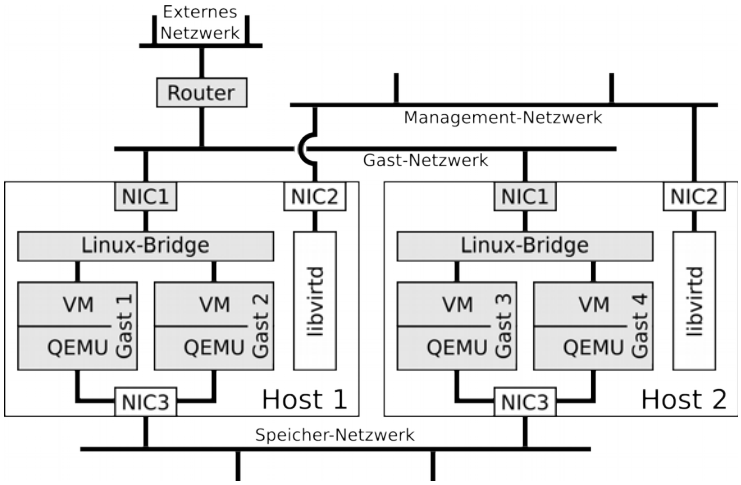

Abbildung 34: Konfiguration der Testumgebung mit Linux-Bridge

Auf beiden Hosts werden jeweils zwei Gäste gestartet und mit dem Netzwerk verbunden. Die Konfiguration ist in Abb. 34 dargestellt. Demgegenüber wird in Abb. 35 der resultierende logische Aufbau des Gast-Netzwerks skizziert. Hierbei entsprechen die virtuellen Switche der jeweiligen Linux-Bridge. Bei dem realen Switch handelt es sich um den (in Abb. 34 nicht eingezeichneten) Switch, an den die physischen Netzwerkschnittstellen der Hosts angeschlossen sind. Es handelt sich somit um ein kaskadiertes geswitchtes Netzwerk. Der Router entspricht dem Router in Abb. 34 und dient als Zugang zu externen Netzwerken.

8.4.1.2 Ergebnisse

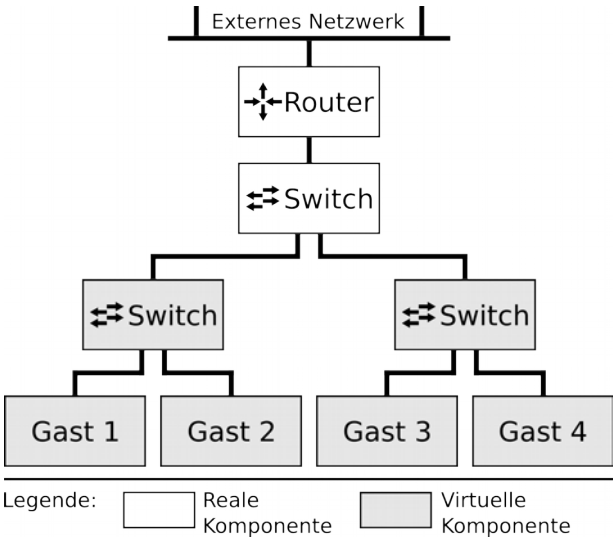

Abbildung 35: Logische Netzwerktopologie im Bridge-Modus

Die Testdurchführung, deren Ergebnisse in Tabelle 30 zusammengefasst sind, zeigt, dass die Konfiguration eine Kommunikation mittels IPv4 und IPv6 grundsätzlich ermöglicht.

Ferner besteht eine Trennung der Gäste vom Host und von dem Management- und dem Storage-Netzwerk. Dies liegt daran, dass die Linux-Bridge im Sinne eines Switches Netzwerkverkehr lediglich an die

angeschlossenen Schnittstellen verteilt, d. h. an die physische Schnittstelle des Gast-Netzwerks und an die Netzwerkschnittstellen der Gäste.

Die Testfälle 6 bis 12 werden allesamt nicht bestanden, da die Linux-Bridge keine Filterung des Netzwerkverkehrs vornimmt, sondern Ethernet-Pakete anhand ihrer Zieladresse vermittelt.

Es lässt sich daher festhalten, dass die Netzwerkanbindung mittels einer vorkonfigurierten Linux-Bridge ohne Härtungsmaßnahmen zwar eine grundlegende Trennung zwischen Gästen und Virtualisierungsumgebung aufweist, eine Reihe von netzwerkbasierten Angriffen jedoch möglich ist.

#	Testbezeichnung	Ergebnis	Bemerkung
1	Vorhandene Konnektivität mittels IPv4	bestanden	
2	Vorhandene Konnektivität mittels IPv6	bestanden	
3	Fehlende Trennung vom Management-Netzwerk	bestanden	Abgesichert durch Reverse-Path-Filter, s. Abschnitt 8.2
4	Fehlende Trennung vom Storage-Netzwerk	bestanden	Abgesichert durch Reverse-Path-Filter, s. Abschnitt 8.2
5	Erreichbarkeit von Diensten des Hosts durch den Gast	bestanden	Abgesichert durch Reverse-Path-Filter, s. Abschnitt 8.2
6	Kommunikation mittels beliebiger Protokolle über Ethernet	nicht bestanden	
7	MAC-Address-Spoofing	nicht bestanden	
8	IP-Address-Spoofing	nicht bestanden	
9	ARP-Spoofing	nicht bestanden	
10	ARP-Flooding	nicht bestanden	
11	NDP-Spoofing/ RA-Spoofing	nicht bestanden	
12	DHCP-Spoofing	nicht bestanden	

Tabelle 30: Testergebnisse der Linux-Bridge

8.4.1.3 Härtung

Um netzwerkbasierte Angriffe zu verhindern, wird der durch libvirt vordefinierte Filter clean-traffic (s. Abschnitt 3.6.4.3) für jeden Host aktiviert. Dies wird umgesetzt, indem die Beschreibung der Gäste um das folgende XML-Tag erweitert wird:

```
<filterref filter='clean-traffic'/>
```

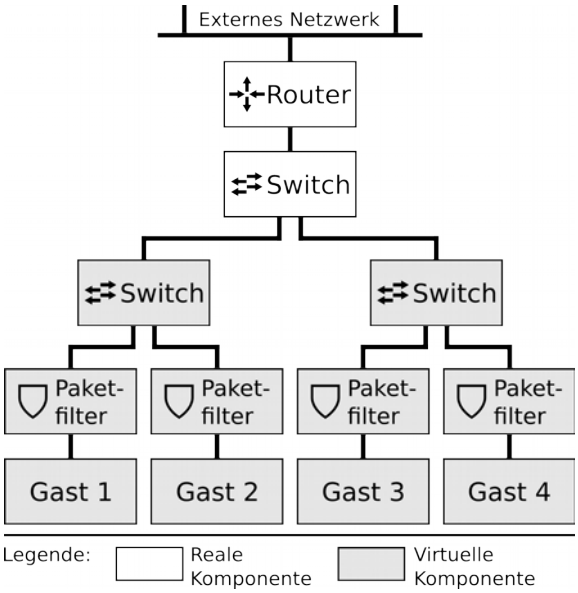

Abbildung 36: Logische Netzwerktopologie mittels Linux-Bridge und Paketfilterung

Die logische Netzwerktopologie verändert sich hierdurch von der in Abb. 35 gezeigten Topologie in diejenige Topologie, die in Abb. 36 dargestellt ist.

Die Auswirkungen der Aktivierung des Filters sind in Tabelle 31 aufgeführt. Es zeigt sich zum einen, dass die Tests 6 bis 11, im Gegensatz zu den entsprechenden vorangegangenen Tests, bestanden werden. Der Filter clean-traffic arbeitet in dieser Konfiguration also bestimmungsgemäß (s. Abschnitt 3.6.4.3). Ferner zeigt sich, dass die Filterung mittels libvirt auf Basis einer Linux-Bridge grundsätzlich funktioniert.

Anzumerken ist jedoch, dass eine Kommunikation mittels IPv6 nicht mehr möglich ist, weil die Unterstützung durch den Filter clean-traffic fehlt. Diese Einschränkung kann jedoch dadurch aufgehoben werden, dass das Regelwerk mithilfe von selbst definierten Filterregeln erweitert wird.

Ferner werden Angriffe via DHCP nicht durch den Filter clean-traffic unterbunden. Der Grund ist, dass das auf UDP basierende DHCP nicht von dem verwendeten Filter überwacht wird. Prinzipiell ist es jedoch möglich, den Filter entsprechend zu erweitern, um DHCP-basierte Angriffe zu unterbinden.[100]

#	Bedrohung	Ergebnis	Bemerkung
1	Vorhandene Konnektivität mittels IPv4	bestanden	
2	Vorhandene Konnektivität mittels IPv6	nicht bestanden	IPv6-Verkehr wird durch den Filter clean-traffic verworfen.
3	Fehlende Trennung vom Management-Netzwerk	bestanden	
4	Fehlende Trennung vom Storage-Netzwerk	bestanden	
5	Erreichbarkeit von Diensten des Hosts durch den Gast	bestanden	
6	Kommunikation mittels	bestanden	Nur IPv4, ARP und RARP sind erlaubt.

100Hierbei können insbesondere die vordefinierten Filter *enable-dhcp* und *enable-dhcp-server* hilfreich sein.

#	Bedrohung	Ergebnis	Bemerkung
	beliebiger Protokolle über Ethernet		
7	MAC-Address-Spoofing	bestanden	
8	IP-Address-Spoofing	bestanden	
9	ARP-Spoofing	bestanden	
10	ARP-Flooding	bestanden	
11	NDP-Spoofing/ RA-Spoofing	bestanden	IPv6-Verkehr wird durch den Filter clean-traffic verworfen.
12	DHCP-Spoofing	nicht bestanden	

Tabelle 31: Testergebnisse der Linux-Bridge nach Härtung

8.4.2 macvtap

8.4.2.1 Modus bridge

8.4.2.1.1 Konfiguration

Auf beiden Hosts der Testumgebung (s. Kapitel 4) wird mithilfe von libvirt ein Netzwerk erzeugt, das auf macvtap basiert. Hierbei wird der Modus bridge für die macvtap-Schnittstellen gewählt.

Folgende Beschreibung wird verwendet, um das Netzwerk zu erzeugen, wobei eth0 die physische Netzwerkschnittstelle des Hosts zum Gast-Netzwerk bezeichnet. Die Schnittstelle eth0 ist aktiviert, aber ansonsten nicht konfiguriert. Insbesondere ist der Netzwerkschnittstelle keine IPv4- und keine IPv6-Adresse zugeordnet. Ferner ist für diese Schnittstelle die IPv6-Autokonfiguration deaktiviert.

```
<network>
        <name>macvtapnet</name>
        <forward mode="bridge">
                        <interface dev="eth0"/>
        </forward>
</network>
```

Auf beiden Hosts werden jeweils zwei Gäste gestartet und mit dem Netzwerk verbunden.

Die Konfiguration der Testumgebung wird in Abb. 37 dargestellt. Die logische Netzwerktopologie entspricht hierbei der Konfiguration mithilfe der Linux-Bridge, die in Abb. 35 dargestellt ist. Hierbei werden die virtuellen Switche durch macvtap realisiert. Bei dem realen Switch handelt es sich um den (in Abb. 37 nicht eingezeichneten) Switch, an den die physischen Netzwerkschnittstellen der Hosts angeschlossen sind. Es liegt somit eine kaskadierte geswitchte Umgebung vor. Der Router entspricht dem Router in Abb. 37 und dient zum Zugang zu externen Netzen.

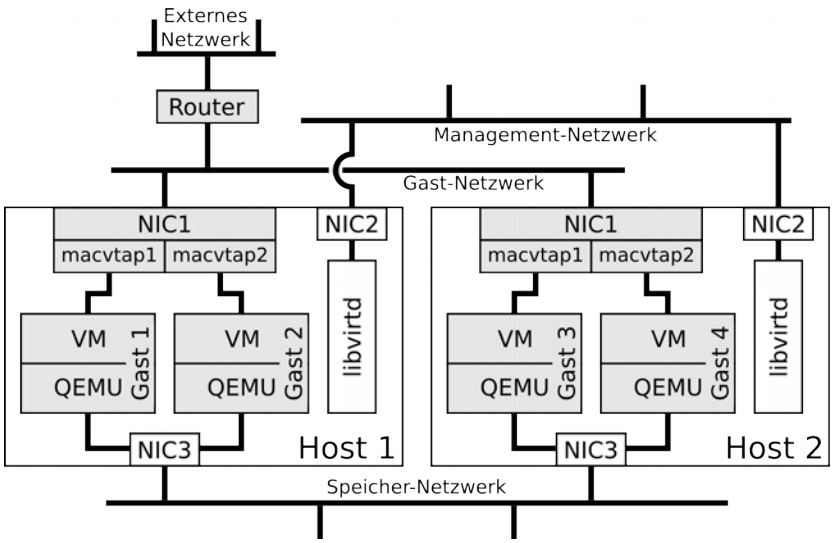

Abbildung 37: Konfiguration der Testumgebung mit macvtap

8.4.2.1.2 Ergebnisse

Die Durchführung des Tests, dessen Ergebnisse in Tabelle 32 zusammengefasst sind, zeigt, dass die Konfiguration eine Kommunikation mittels IPv4 und IPv6 grundsätzlich ermöglicht.

Ferner besteht eine Trennung der Gäste vom Host und vom Management- und Storage-Netzwerk. Der Grund hierfür ist, dass die macvtap-Schnittstellen Netzwerkkommunikation zwar untereinander vermitteln, die physische Netzwerkschnittstelle jedoch ausschließlich zur Aus- und Einleitung von Kommunikation in das reale Netzwerk bzw. aus ihm nutzen. Eine Kommunikation mit dem Host ist somit nicht unmittelbar möglich.[101]

Die Testfälle 6 bis 12 werden allesamt nicht bestanden, da macvtap im Modus bridge keine Filterung des Netzwerkverkehrs vornimmt, sondern Ethernet-Pakete anhand ihrer Zieladresse an den jeweiligen Empfänger vermittelt. Letzteres hat jedoch zur Folge, dass ein MAC-Spoofing nur bedingt möglich ist. So kann zwar ein Ethernet-Paket mit gefälschter Absenderadresse versendet werden. Ein eingehendes Ethernet-Paket mit entsprechender Zieladresse kann durch den Host jedoch typischerweise keiner macvtap-Schnittstelle zugeordnet werden und wird daher nicht an den Gast weitergegeben.

Es lässt sich daher festhalten, dass die Netzwerkanbindung mittels macvtap im Modus bridge ohne Härtungsmaßnahmen zwar eine grundlegende Trennung zwischen Gästen und Virtualisierungsumgebung aufweist, eine Reihe von netzwerkbasierten Angriffen jedoch möglich ist.

#	Bedrohung	Ergebnis	Bemerkung
1	Vorhandene Konnektivität mittels IPv4	bestanden	
2	Vorhandene Konnektivität mittels IPv6	bestanden	
3	Fehlende Trennung vom Management-Netzwerk	bestanden	
4	Fehlende Trennung vom Storage-Netzwerk	bestanden	

101 Sofern der reale Switch jedoch im Hairpin-Modus arbeitet oder der ausgeleitete Verkehr auf anderem Wege zum Host reflektiert wird, ist eine solche Kommunikation jedoch sehr wohl möglich.

#	Bedrohung	Ergebnis	Bemerkung
5	Erreichbarkeit von Diensten des Hosts durch den Gast	bestanden	
6	Kommunikation mittels beliebiger Protokolle über Ethernet	nicht bestanden	
7	MAC-Address-Spoofing	nicht bestanden	Antworten mit gefälschten MAC-Adressen werden von dem Host nicht an den Gast weitergeleitet.
8	IP-Address-Spoofing	nicht bestanden	
9	ARP-Spoofing	nicht bestanden	
10	ARP-Flooding	nicht bestanden	
11	NDP-Spoofing/ RA-Spoofing	nicht bestanden	
12	DHCP-Spoofing	nicht bestanden	

Tabelle 32: Testergebnisse von macvtap im Modus „bridge“

8.4.2.1.3 Härtung

Da die Filterung mithilfe von netfilter beim Einsatz von macvtap nicht möglich ist, stehen auch die Filtermöglichkeiten von libvirt (s. Abschnitt 3.6.4.3) nicht zur Verfügung. Anders als bei Realisierung eines geswitchten Gast-Netzwerks mithilfe einer Linux-Bridge (s. Abschnitt 8.4.1) stehen damit keine flexibel konfigurierbaren Filtermöglichkeiten auf dem Host zur Verfügung. Verkehr, der den Host verlässt oder zu ihm eingeht, kann grundsätzlich mit externen Systemen analysiert werden. Verkehr, der zwischen Gästen desselben Hosts vermittelt wird, wird jedoch nicht ausgeleitet (s. Abschnitt 3.5.4.4) und entzieht sich daher einer solchen Überprüfung. Eine Filterung der Kommunikation zwischen Gästen eines Hosts kann daher nicht von der Virtualisierungsumgebung, sondern lediglich durch die Gäste selbst durchgeführt werden.

8.4.2.2 Modus vepa

8.4.2.2.1 Konfiguration

Auf beiden Hosts der Testumgebung (s. Kapitel 4) wird mithilfe von libvirt ein Netzwerk erzeugt, das auf macvtap basiert. Hierbei wird der Modus vepa für die macvtap-Schnittstellen gewählt.

Folgende Beschreibung wird verwendet, um das Netzwerk zu erzeugen, wobei eth0 die physische Netzwerkschnittstelle des Hosts zum Gast-Netzwerk bezeichnet. Die Schnittstelle eth0 ist aktiviert, aber ansonsten nicht konfiguriert. Insbesondere ist der Netzwerkschnittstelle keine IPv4- und keine IPv6-Adresse zugeordnet. Ferner ist für diese Schnittstelle die IPv6-Autokonfiguration deaktiviert.

```
<network>
        <name>vepanet</name>
        <forward mode="vepa">
                        <interface dev="eth0"/>
        </forward>
```

```
</network>
```

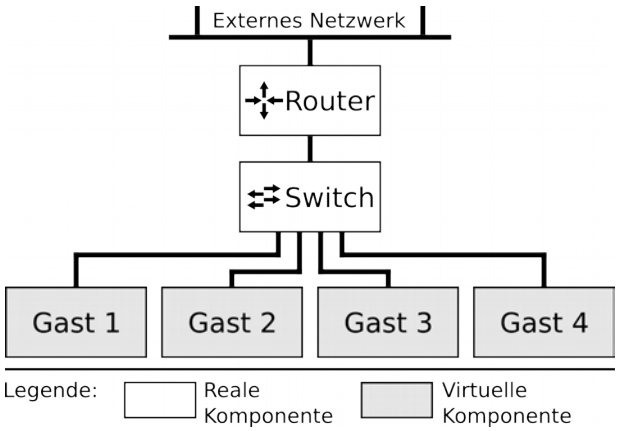

Abbildung 38: Logische Netzwerktopologie im Modus vepa

Auf beiden Hosts werden jeweils zwei Gäste gestartet und mit dem Netzwerk verbunden. Die physische Netzwerkschnittstelle des Hosts ist mit einem Switch verbunden, der im Hairpin-Modus arbeitet.

Die Konfiguration des Hosts entspricht, mit Ausnahme des macvtap-Modus, der Konfiguration, die in Abschnitt 8.4.2.1 beschrieben und in Abb. 37 dargestellt ist. Die logische Netzwerktopologie wird in Abb. 38 skizziert. Sie unterscheidet sich von der in Abb. 35 dargestellten Topologie insofern, als dass die virtuellen Switche nicht vorhanden sind und der gesamte Datenverkehr grundsätzlich über den realen Switch geleitet wird.

8.4.2.2.2 Ergebnisse

Die Testergebnisse des Modus vepa, die in Tabelle 33 zusammengefasst sind, entsprechen denen des Modus bridge (s. Abschnitt 8.4.2.1.2). Der Grund hierfür ist, dass sich die Modi lediglich in der Vermittlung des Netzwerkverkehrs zwischen Gästen unterscheiden (s. Abschnitt 3.5.4.4). Während der Modus bridge die entsprechende Kommunikation hostintern vermittelt, wird sie im Modus vepa stets ausgeleitet. Der reale Switch der Testumgebung reflektiert den Netzwerkverkehr ungefiltert, woraufhin der Host diesen an den jeweiligen Gast übergibt.

Aufgrund der notwendigen Reflexion besteht jedoch die Gefahr, dass eine Kommunikation der Gäste mit dem Host über die physische Netzwerkschnittstelle möglich ist, die den macvtap-Schnittstellen zugrunde liegt. Da an diese Schnittstelle jedoch im Rahmen der Testdurchführung keine IP-Adresse gebunden ist, wird eine solche Kommunikation verhindert.

Es lässt sich daher festhalten, dass die Netzwerkanbindung mithilfe von macvtap im Modus vepa ohne Härtungsmaßnahmen zwar eine grundlegende Trennung zwischen Gästen und Virtualisierungsumgebung aufweist, eine Reihe von netzwerkbasierten Angriffen jedoch möglich ist.

#	Bedrohung	Ergebnis	Bemerkung
1	Vorhandene Konnektivität mittels IPv4	bestanden	
2	Vorhandene Konnektivität mittels IPv6	bestanden	
3	Fehlende Trennung vom Management-Netzwerk	bestanden	

#	Bedrohung	Ergebnis	Bemerkung
4	Fehlende Trennung vom Storage-Netzwerk	bestanden	
5	Erreichbarkeit von Diensten des Hosts durch den Gast	bestanden	
6	Kommunikation mittels beliebiger Protokolle über Ethernet	nicht bestanden	
7	MAC-Address-Spoofing	nicht bestanden	Antworten zu gefälschten MAC-Adressen werden vom Host nicht an den Gast weitergegeben.
8	IP-Address-Spoofing	nicht bestanden	
9	ARP-Spoofing	nicht bestanden	
10	ARP-Flooding	nicht bestanden	
11	NDP-Spoofing/RA-Spoofing	nicht bestanden	
12	DHCP-Spoofing	nicht bestanden	

Tabelle 33: Testergebnisse von macvtap im Modus „vepa“

8.4.2.2.3 Härtung

Analog zum Modus bridge (s. Abschnitt 8.4.2.1.3) besteht beim Modus vepa nicht die Möglichkeit einer flexibel konfigurierbaren Filterung des Netzwerkverkehrs durch den Host. Anders als im Modus bridge wird jedoch sämtlicher Verkehr aus dem Host ausgeleitet (s. Abschnitt 3.5.4.4). Dies ermöglicht die Filterung der gesamten Kommunikation durch externe Netzwerkkomponenten, d. h. sowohl die Filterung der Kommunikation zwischen Gast und externen Netzwerkteilnehmern als auch die Filterung der Kommunikation zwischen Gästen.

macvtap im Modus vepa ist daher geeignet, die Filterung des gesamten Netzwerkverkehrs auf externe Komponenten (wie z. B. einen OpenFlow-fähigen Switch) zu übertragen. Die fehlende Filterung durch den Host kann somit kompensiert werden. Auch kann eine solche Trennung zwischen der eigentlichen Virtualisierung und der Vermittlung und Filterung der Netzwerkkommunikation zu einem erhöhten Sicherheitsniveau beitragen.

8.4.2.3 Modus private

8.4.2.3.1 Konfiguration

Auf beiden Hosts der Testumgebung (s. Abschnitt 4) wird mithilfe von libvirt ein Netzwerk erzeugt, das auf macvtap basiert. Hierbei wird der Modus private für die macvtap-Schnittstellen gewählt.

Folgende Beschreibung wird verwendet, um das Netzwerk zu erzeugen, wobei eth0 die physische Netzwerkschnittstelle des Hosts zum Gast-Netzwerk bezeichnet. Die Schnittstelle eth0 ist aktiviert, aber ansonsten nicht konfiguriert. Insbesondere ist der Netzwerkschnittstelle keine IPv4- und keine IPv6-Adresse zugeordnet. Ferner ist für die Schnittstelle die IPv6-Autokonfiguration deaktiviert.

```
<network>
```

```
        <name>privatenet</name>
        <forward mode="private">
                <interface dev="eth0"/>
        </forward>
</network>
```

Auf beiden Hosts werden jeweils zwei Gäste gestartet und mit dem Netzwerk verbunden. Die physische Netzwerkschnittstelle des Hosts ist mit einem Switch verbunden, der im Hairpin-Modus arbeitet.[102]

Die Konfiguration entspricht, mit Ausnahme des macvtap-Modus, der Konfiguration, die in Abschnitt 8.4.2.1 beschrieben und in Abb. 37 dargestellt ist. Die logische Netzwerktopologie entspricht weitestgehend der in Abb. 38 skizzierten Topologie. Pakete, die von einem Gast erzeugt werden, sollen jedoch im Modus private nicht an einen anderen Gast auf demselben Host weitergeleitet, sondern verworfen werden (s. Abschnitt 3.5.4.4). Ein Paket, das von dem in Abb. 38 dargestellten Gast 1 an Gast 2 geschickt wird, soll daher vom Host verworfen werden.[103]

8.4.2.3.2 Ergebnisse

Die Testdurchführung, deren Ergebnisse in Tabelle 34 zusammengefasst sind, zeigt, dass die Konfiguration eine grundsätzliche Kommunikation mittels IPv4 ermöglicht. Dagegen ist die Kommunikation mittels IPv6 aufgrund einer fehlgeschlagenen Autokonfiguration nicht ohne Weiteres möglich. Wird der Hairpin-Modus abgeschaltet, so funktioniert die Autokonfiguration.

Darüber hinaus ist die Isolation der Gäste untereinander keineswegs sichergestellt. Tatsächlich blockiert der Modus private die Auflösung von IPv6- und IPv4-Adressen zu MAC-Adressen, indem entsprechende ARP- bzw. ICMPv6-Nachrichten verworfen werden. Hierdurch sind Gäste untereinander nicht sichtbar, und eine Kommunikation kann nicht ohne Weiteres erfolgen. Allerdings lässt sich dieser Schutz umgehen, sofern der angeschlossene Switch im Hairpin-Modus arbeitet. Die Ursache hierfür ist, dass Pakete (außer im Rahmen der Adressauflösung) grundsätzlich ausgeleitet werden. Sofern die Übersetzung von IP- in MAC-Adressen auf den Gästen statisch hinterlegt wird und somit die Durchführung einer dynamischen Adressübersetzung entfällt, können Gäste Pakete aneinander versenden. Diese werden von der macvtap-Schnittstelle ausgeleitet, vom Switch reflektiert und letztendlich zum entsprechenden Gast weitergeleitet.

Erschwerend kommt hinzu, dass die statische Konfiguration der Adressübersetzung umgangen werden kann. Hierzu versendet ein Angreifer ein ARP-Paket, das seine MAC- und IP-Adresse enthält. Als Absender im Ethernet-Header trägt er hingegen eine gefälschte MAC-Adresse ein. Hierdurch ordnet die macvtap-Schnittstelle das ARP-Paket nicht dem versendenden Gast zu. Das Paket wird ausgeleitet und erreicht durch Reflexion die Gäste. Diese erstellen auf Basis des ARP-Pakets eine dynamische Zuordnung zwischen der IPv4- und der MAC-Adresse. Hierzu wird typischerweise ausschließlich auf die Adresszuordnung im ARP-Header zurückgegriffen. Die gefälschte MAC-Absenderadresse stört diesen Vorgang nicht. Somit ist es dem versendenden Gast gelungen, anderen Gästen die Zuordnung seiner IPv4-Adresse zu seiner MAC-Adresse mitzuteilen. Sofern ersterer seine eigene Zuordnungstabelle anpasst, ist somit eine Kommunikation zwischen den Gästen möglich und der Schutz durch die macvtap-Schnittstelle umgangen. Analog kann mit ICMPv6-Nachrichten bei Nutzung von IPv6 verfahren werden.

102 Die Aktivierung des Hairpin-Modus ist keine Voraussetzung für die Funktionstüchtigkeit von macvtap im Modus *private*. Eine Aktivierung wird aufgrund der starken Ähnlichkeit zum Modus *vepa* trotzdem vorgenommen, um etwaige Schwachstellen ermitteln zu können. Ferner beschreiben eine Reihe von Dokumentationen, wie z. B. [MACVTAP], dass die Schutzfunktionen des Modus *private* unabhängig von der Nutzung des Hairpin-Modus effektiv sind.

103 Dieses Verhalten ist jedoch auf die jeweilige Netzwerkschnittstelle beschränkt, an die macvtap gebunden ist. Eine Kommunikation zwischen Gast 1 und Gast 3 ist beispielsweise möglich, da diese von unterschiedlichen Hosts virtualisiert werden.

Darüber hinaus gelten die Ergebnisse zum Modus vepa (s. Abschnitt 8.4.2.2.2) analog.

#	Bedrohung	Ergebnis	Bemerkung
1	Vorhandene Konnektivität mittels IPv4	bestanden	
2	Vorhandene Konnektivität mittels IPv6	nicht bestanden	
3	Fehlende Trennung vom Management-Netzwerk	bestanden	
4	Fehlende Trennung vom Storage-Netzwerk	bestanden	
5	Erreichbarkeit von Diensten des Hosts durch den Gast	bestanden	
6	Kommunikation mittels beliebiger Protokolle über Ethernet	nicht bestanden	
7	MAC-Address-Spoofing	nicht bestanden	Antworten zu gefälschten MAC-Adressen werden von dem Host verworfen.
8	IP-Address-Spoofing	nicht bestanden	
9	ARP-Spoofing	nicht bestanden	
10	ARP-Flooding	nicht bestanden	
11	NDP-Spoofing/ RA-Spoofing	nicht bestanden	
12	DHCP-Spoofing	nicht bestanden	

Tabelle 34: Testergebnisse von macvtap im Modus „private“

8.4.2.3.3 Härtung

Analog zum Modus vepa besteht beim Modus private nicht die Möglichkeit einer effektiven Filterung des Netzwerkverkehrs durch den Host. Vergleichbar mit dem Modus vepa wird keine hostinterne Weiterleitung der Kommunikation vorgenommen, sondern sämtlicher Verkehr ausgeleitet. Somit gilt das in Abschnitt 8.4.2.2.3 Gesagte für den Modus private analog.

Darüber hinaus sollte aber auf den Einsatz eines externen Switches mit aktiviertem Hairpin-Modus verzichtet werden. Ein solcher Switch unterbindet nicht nur die Funktionstüchtigkeit von IPv6, sondern ermöglicht es den Gästen, untereinander zu kommunizieren, und hebt somit die Schutzfunktion des Modus private auf. Wird auf einen entsprechend konfigurierten Switch verzichtet, so ist sowohl IPv6 funktionstüchtig als auch eine Kommunikation zwischen Gästen unterbunden.

8.4.2.4 Modus passthrough

8.4.2.4.1 Konfiguration

Auf beiden Hosts der Testumgebung (s. Kapitel 4) wird mithilfe von libvirt ein Netzwerk erzeugt, das auf macvtap basiert. Hierbei wird der Modus passthrough für die macvtap-Schnittstellen gewählt.

Folgende Beschreibung wird verwendet, um das Netzwerk zu erzeugen, wobei eth0 die physische Netzwerkschnittstelle des Hosts zum Gast-Netzwerk bezeichnet.

```
<network>
        <name>passthroughnet</name>
        <forward mode="passthrough">
                <interface dev="eth0"/>
        </forward>
</network>
```

Auf beiden Hosts wird jeweils ein Gast[104] gestartet und mit dem Netzwerk verbunden.

8.4.2.4.2 Ergebnisse

Die in Tabelle 35 zusammengefassten Ergebnisse entsprechen denen des Modus vepa (s. Abschnitt 8.4.2.2.2).

#	Bedrohung	Ergebnis	Bemerkung
1	Vorhandene Konnektivität mittels IPv4	bestanden	
2	Vorhandene Konnektivität mittels IPv6	bestanden	
3	Fehlende Trennung vom Management-Netzwerk	bestanden	
4	Fehlende Trennung vom Storage-Netzwerk	bestanden	
5	Erreichbarkeit von Diensten des Hosts durch den Gast	bestanden	
6	Kommunikation mittels beliebiger Protokolle über Ethernet	nicht bestanden	
7	MAC-Address-Spoofing	nicht bestanden	
8	IP-Address-Spoofing	nicht bestanden	
9	ARP-Spoofing	nicht bestanden	
10	ARP-Flooding	nicht bestanden	
11	NDP-Spoofing/ RA-Spoofing	nicht bestanden	
12	DHCP-Spoofing	nicht bestanden	

Tabelle 35: Testergebnisse von macvtap im Modus „passthrough“

104 Der Umstand, dass pro Host nur ein Gast gestartet wird, ist der Tatsache geschuldet, das macvtap im Modus *passthrough* keine geteilte Nutzung einer Netzwerkschnittstelle ermöglicht (s. Abschnitt 3.5.4.4).

8.4.2.4.3 Härtung

Für die Härtung einer macvtap-Schnittstelle im Modus passthrough gelten die Aussagen analog, die bezüglich des Modus vepa gemacht wurden (s. Abschnitt 8.4.2.2.3).

8.4.3 Open vSwitch

8.4.3.1 Konfiguration

Auf beiden Hosts der Testumgebung (s. Kapitel 4) wird mithilfe des folgenden Skripts ein Open vSwitch mit dem Namen ovs-bridge erzeugt und die als eth0 bezeichnete physische Schnittstelle des Gast-Netzwerks angehängt. Hierbei ist eth0 aktiv, aber nicht konfiguriert. Weder an ovs-bridge noch an eth0 ist eine IPv4- oder IPv6-Adresse gebunden. Ferner ist für diese Schnittstellen die IPv6-Autokonfiguration deaktiviert.

```
#!/bin/bash
ovs-vsctl add-br ovs-bridge
ovs-vsctl add-port ovs-bridge eth0
ip link set up dev ovs-bridge
```

Im Anschluss wird mit libvirt und der folgenden Beschreibung ein Netzwerk erstellt. das auf der Open-vSwitch-Schnittstelle ovs-bridge basiert.

```
<network>
        <name>ovsnet</name>
        <forward mode='bridge'/>
        <bridge name='ovs-bridge'/>
        <virtualport type='openvswitch'/>
</network>
```

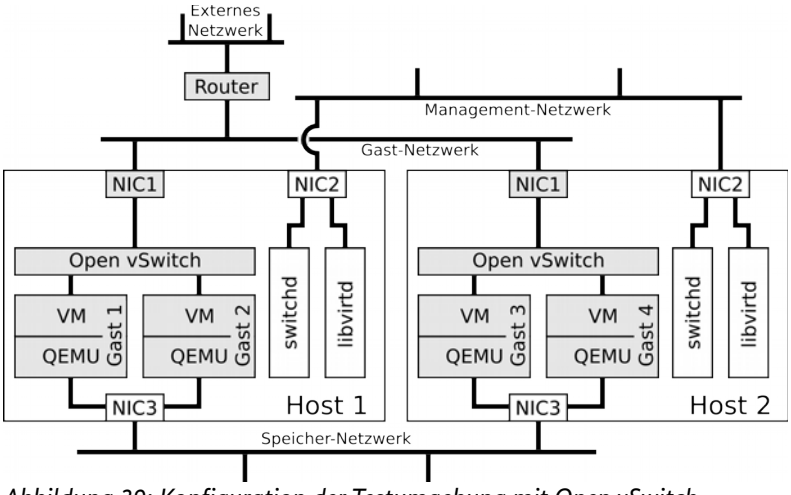

Abbildung 39: Konfiguration der Testumgebung mit Open vSwitch

Auf beiden Hosts werden jeweils zwei Gäste gestartet und mit dem Netzwerk verbunden. Die Konfiguration ist in Abb. 39 dargestellt.

Bei der gewählten Konfiguration wurde bewusst auf den Einsatz einer Control-Plane verzichtet. Zwar besteht der primäre Zweck des Einsatzes von Open vSwitch in der Auslagerung des Regelwerks zur Verkehrsverarbeitung. Um die grundlegende Vergleichbarkeit der Technologien herstellen zu können, wird an dieser Stelle aber auf eine Control-Plane verzichtet.

8.4.3.2 Ergebnisse

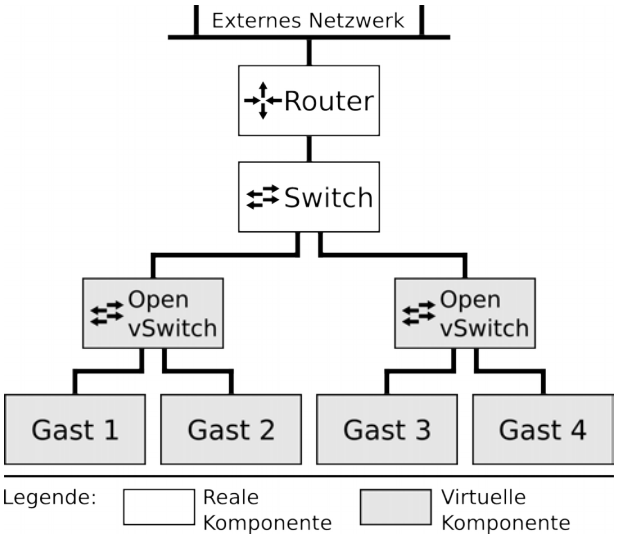

Abbildung 40: Logische Netzwerktopologie mit Open vSwitch

Die Testdurchführung, deren Ergebnisse in Tabelle 36 zusammengefasst sind, zeigt, dass die Konfiguration eine Kommunikation mithilfe von IPv4 und IPv6 grundsätzlich ermöglicht. Ferner besteht eine Trennung der Gäste vom Host und vom Management- und Storage-Netzwerk. Die Testfälle 6 bis 12 werden hingegen allesamt nicht bestanden.

Der Grund für die Ergebnisse ist, dass bei Erstellung der Open-vSwitch-Schnittstelle standardmäßig ein Regelwerk aktiviert wird, das aus einer einzigen Regel besteht. Diese Regel bewirkt, dass die Datenweiterleitung analog zu einem üblichen Switch auf Basis der MAC-Empfänger-Adresse durchgeführt wird. Das System verhält sich also aus sicherheitstechnischen Gesichtspunkten identisch zu einer Linux-Bridge ohne Härtungsmaßnahmen (s. Abschnitt 8.4.1.2).

Es lässt sich daher festhalten, dass die Netzwerkanbindung mit Open vSwitch ohne zusätzliche Konfiguration oder den Einsatz einer Control-Plane zwar eine grundlegende Trennung zwischen den Gästen und der Virtualisierungsumgebung aufweist, eine Reihe von netzwerkbasierten Angriffen jedoch möglich ist.

#	Bedrohung	Ergebnis	Bemerkung
1	Vorhandene Konnektivität mittels IPv4	bestanden	
2	Vorhandene Konnektivität mittels IPv6	bestanden	
3	Fehlende Trennung vom Management-Netzwerk	bestanden	
4	Fehlende Trennung vom Storage-Netzwerk	bestanden	

#	Bedrohung	Ergebnis	Bemerkung
5	Erreichbarkeit von Diensten des Hosts durch den Gast	bestanden	
6	Kommunikation mittels beliebiger Protokolle über Ethernet	nicht bestanden	
7	MAC-Address-Spoofing	nicht bestanden	
8	IP-Address-Spoofing	nicht bestanden	
9	ARP-Spoofing	nicht bestanden	
10	ARP-Flooding	nicht bestanden	
11	NDP-Spoofing/RA-Spoofing	nicht bestanden	
12	DHCP-Spoofing	nicht bestanden	

Tabelle 36: Testergebnisse von Open vSwitch

8.4.3.3 Härtung

Da Netzwerkverkehr, der über Open vSwitch vermittelt wird, nicht durch netfilter gefiltert werden kann, stehen die Filterfunktionen von libvirt zur Härtung nicht zur Verfügung. In der zugrunde liegenden Testumgebung hält dies libvirt jedoch nicht davon ab, Netzwerkbeschreibungen, die Filter enthalten, ohne Beanstandung entgegenzunehmen. Die Regeln werden jedoch von libvirt ignoriert und sind somit nicht effektiv.

Im Folgenden wird getestet, inwiefern es mit Open vSwitch dennoch möglich ist, die getesteten Netzwerkangriffe zu unterbinden. Hierzu wird ein einfaches Regelwerk erstellt, das einen Controller simuliert. Das Regelwerk ersetzt die oben beschriebene vordefinierte Standardregel und verleiht der Open-vSwitch-Schnittstelle Filterfunktionen.

Hierbei soll jedoch nicht unerwähnt bleiben, dass die Konfiguration von Open vSwitch oder allgemeiner von OpenFlow-fähigen Switchen in der Praxis durch komplexe Control-Planes durchgeführt wird. Die Konfiguration mithilfe eines Skripts wird im Rahmen dieser Untersuchung aber dennoch als sinnvoll erachtet, da somit die Ergebnisse unabhängig von einer konkreten Implementierung einer Control-Plane sind. Auch kann auf diesem Wege die grundsätzliche Möglichkeit einer entsprechenden Filterung untersucht werden.

Das verwendete Regelwerk ist in Anhang A aufgeführt. Nach Aktivierung des Regelwerks werden alle Testfälle, wie in Tabelle 37 dargestellt, bestanden.

#	Bedrohung	Ergebnis	Bemerkung
1	Vorhandene Konnektivität mittels IPv4	bestanden	
2	Vorhandene Konnektivität mittels IPv6	bestanden	
3	Fehlende Trennung vom Management-Netzwerk	bestanden	
4	Fehlende Trennung vom Storage-Netzwerk	bestanden	

#	Bedrohung	Ergebnis	Bemerkung
5	Erreichbarkeit von Diensten des Hosts durch den Gast	bestanden	
6	Kommunikation mittels beliebiger Protokolle über Ethernet	bestanden	
7	MAC-Address-Spoofing	bestanden	
8	IP-Address-Spoofing	bestanden	
9	ARP-Spoofing	bestanden	
10	ARP-Flooding	bestanden	
11	NDP-Spoofing/ RA-Spoofing	bestanden	
12	DHCP-Spoofing	bestanden	

Tabelle 37: Testergebnisse von Open vSwitch nach Härtung

8.4.4 Virtuelle Netzwerke mit libvirt

8.4.4.1 Modus isolated

8.4.4.1.1 Konfiguration

Auf beiden Hosts der Testumgebung (s. Kapitel 4) wird ein libvirt-Netzwerk im Modus isolated (s. Abschnitt 3.6.4.1) erstellt. die folgende Beschreibung wird verwendet, um das Netzwerk zu erzeugen.[105]

```
<network>
 <name>isolatednet</name>
 <bridge name="virbr0" />
 <ip address="192.168.152.1" netmask="255.255.255.0">
  <dhcp>
   <range start="192.168.152.2" end="192.168.152.254" />
  </dhcp>
 </ip>
 <ip family="ipv6" address="2001:db8:ca2:3::1" prefix="64" />
</network>
```

Im Rahmen der Netzwerkinitialisierung startet libvirt einen DHCP-Server auf dem Host (s. Abschnitt 3.6.4.1). Dieser Server verteilt gemäß der aufgeführten Netzwerkbeschreibung die IPv4-Adressen

105 Hierbei ist die physische Schnittstelle des externen Gast-Netzwerks aktiv, aber nicht konfiguriert. Insbesondere ist keine IP-Adresse an die Schnittstelle gebunden. Da der Modus *isolated* aber keine Ausleitung von Paketen erlaubt, sollte dies unerheblich für die Testdurchführung sein.

192.168.152.2 bis 192.168.152.254. Ferner ermöglicht der Host die IPv6-Autokonfiguration mit dem Netzwerkpräfix 2001:db8:ca2:3.

Auf beiden Hosts werden jeweils zwei Gäste gestartet und mit dem Netzwerk verbunden. Die Konfiguration ist in Abb. 41, die entsprechende logische Netzwerktopologie in Abb. 42 dargestellt. Hierbei entspricht der virtuelle Switch der Linux-Bridge.

8.4.4.1.2 Ergebnisse

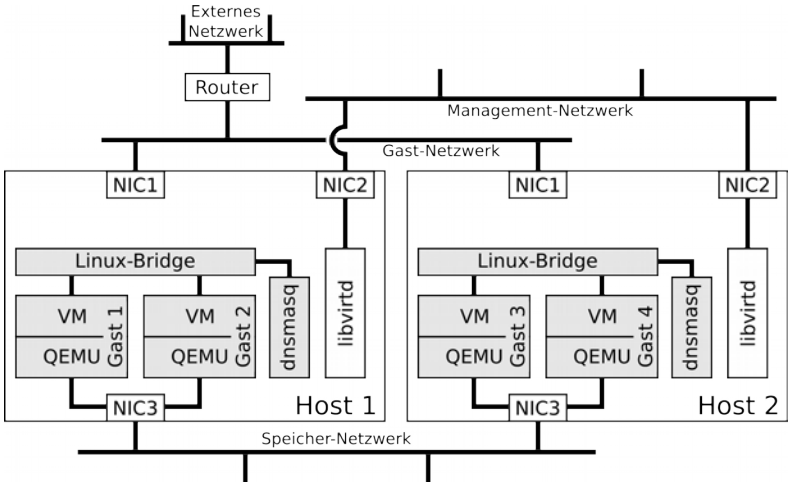

Abbildung 41: Konfiguration der Testumgebung mit einem libvirt-Netzwerk im Modus „isolated“

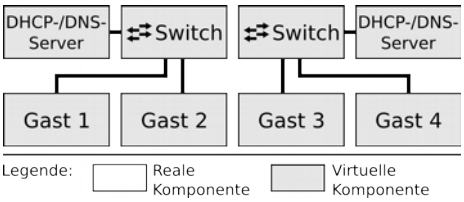

Abbildung 42: Logische Netzwerktopologie von libvirt-Netzwerken im Modus „isolated“

Die Testdurchführung, deren Ergebnisse in Tabelle 38 zusammengefasst sind, zeigt, dass die Konfiguration eine Kommunikation der Gäste über IPv4 und IPv6 grundsätzlich ermöglicht. Ferner besteht eine Trennung der Gäste vom Host und vom Management- und Storage-Netzwerk.

Des Weiteren wird den Gästen ein DHCP- und DNS-Dienst zur Verfügung gestellt, indem ein entsprechender Netzwerkdienst auf dem Host gestartet wird (s. Abschnitt 3.6.4.1). Um eine Kommunikation zwischen den Gästen und dem beschriebenen Dienst zu ermöglichen, bindet libvirt eine IP-Adresse des virtuellen Netzwerks an die Linux-Bridge. Anhand dieser Adresse können Gäste somit mit den auf dem Host betriebenen Diensten kommunizieren. Auch wenn dieses Vorgehen zweckdienlich ist, so birgt es doch auch eine Reihe potenzieller Risiken (s. Abschnitt 8.2). Diesbezüglich ist festzuhalten, dass durch die Nutzung eines libvirt-Netzwerks im Modus isolated das IP-Forwarding nicht aktiviert wird. Da diese Funktion aber unter Umständen anderweitig benötigt wird (wie z. B. bei paralleler Nutzung virtueller Netzwerke im Modus routed oder nat), kann in aller Regel nicht davon ausgegangen werden, dass diese Funktion tatsächlich deaktiviert ist. Um eine Weiterleitung von IP-Paketen zu verhindern, installiert libvirt Filterregeln mittels iptables. Diese blockieren die Weiterleitung von IP-Paketen über die Linux-Bridge hinaus.

Dagegen wird der Zugriff auf Netzwerkdienste des Hosts nicht auf eine solche strikte Art und Weise isoliert. Zwar werden sinnvollerweise Regeln mit iptables installiert, die den Zugriff auf den DNS- und DHCP-Dienst explizit zulassen, eine Beschränkung der Zugriffsmöglichkeiten führt libvirt jedoch nicht durch.

Die tatsächlich doch existierenden Beschränkungen sind nicht auf libvirt, sondern auf die aktivierten Security Policies von CentOS zurückzuführen. Durch die Aktivierung der Security Policy CentOS Profile for Cloud Providers (CPCP) im Rahmen der Betriebssysteminstallation (s. Kapitel 4) werden Regeln aktiviert, die lediglich einen Netzwerkzugriff auf den Dienst SSH zulassen. Auch wenn vordefinierte Sicherheitsregeln seitens CentOS zu begrüßen sind, so erscheint die Zusammenarbeit von libvirt mit dem Betriebssystem an dieser Stelle doch eher einem glücklichen Zufall und nicht einer durchdachten Sicherheitsstrategie

geschuldet zu sein. Insbesondere sorgt libvirt nicht für zusätzliche Schutzmaßnahmen, sofern die besagte Security Policy nicht aktiv ist. Auch mit aktivierter Security Policy ist jedoch ein Zugriff auf den Dienst SSH des Hosts möglich. Zwar bietet SSH eigene Methoden zur Zugriffsbeschränkung, dennoch ist es ein Mangel, dass der Gast auf diesen sicherheitskritischen Managementdienst zugreifen kann – insbesondere, da dieser genutzt wird, um die Kommunikation zu libvirt zu ermöglichen. Aufgrund fehlender strikter Beschränkungen beim Zugriff der Gäste auf Dienste des Hosts wird Testfall 5 nicht bestanden.

Ferner werden die Testfälle 6 bis 12 allesamt nicht bestanden. Der Grund dafür ist, dass libvirt zur Realisierung des virtuellen Netzwerks eine Linux-Bridge nutzt (s. Abschnitt 3.6.4.1). Dies bewirkt, dass die Kommunikation zwischen Gästen wie bei einem üblichen Switch auf Basis der MAC-Empfänger-Adresse durchgeführt wird. Das System verhält sich diesbezüglich also unter sicherheitstechnischen Gesichtspunkten genau so wie bei der Nutzung einer Linux-Bridge ohne Härtungsmaßnahmen (s. Abschnitt 8.4.1.2).

Über die in Abschnitt 8.3 beschriebenen Testfälle hinaus wird analog zu Testfall 1 geprüft, ob eine Kommunikation mit dem Gast-Netzwerk möglich ist. Eine solche Kommunikation wird unterbunden, wie es durch den Netzwerkmodus isolated bezweckt wird. Technisch ist dies mithilfe von iptables-Filterregeln umgesetzt.

Es lässt sich daher festhalten, dass die Netzwerkanbindung mithilfe eines libvirt-Netzwerks im Modus isolated ohne zusätzliche Härtungsmaßnahmen zwar eine isolierte Netzwerkumgebung erzeugt. Eine Interaktion mit Diensten des Hosts ist jedoch möglich. Ferner ist eine Reihe von netzwerkbasierten Angriffen möglich.

#	Bedrohung	Ergebnis	Bemerkung
1	Vorhandene Konnektivität mittels IPv4	bestanden	Der Test wurde auf die Prüfung beschränkt, ob andere Gäste des jeweiligen Netzwerks erreicht werden können.
2	Vorhandene Konnektivität mittels IPv6	bestanden	Der Test wurde auf die Prüfung beschränkt, ob andere Gäste des jeweiligen Netzwerks erreicht werden können.
3	Fehlende Trennung vom Management-Netzwerk	bestanden	Die Gäste können jedoch die IP-Adresse des Hosts im Management-Netzwerk ermitteln (s. Abschnitt 8.2).
4	Fehlende Trennung vom Storage-Netzwerk	bestanden	Die Gäste können jedoch die IP-Adresse des Hosts im Storage-Netzwerk ermitteln (s. Abschnitt 8.2).
5	Erreichbarkeit von Diensten des Hosts durch den Gast	nicht bestanden	Der Netzwerkdienst SSH des Hosts ist vom Gast aus erreichbar.
6	Kommunikation mittels beliebiger Protokolle über Ethernet	nicht bestanden	
7	MAC-Address-Spoofing	nicht bestanden	
8	IP-Address-Spoofing	nicht bestanden	
9	ARP-Spoofing	nicht bestanden	
10	ARP-Flooding	nicht bestanden	
11	NDP-Spoofing/ RA-Spoofing	nicht bestanden	
12	DHCP-Spoofing	nicht bestanden	

Tabelle 38: Testergebnisse des libvirt-Netzwerks im Modus „isolated“

8.4.4.1.3 Härtung

Zur Härtung der Konfiguration wird wie bei der Härtung einer Linux-Bridge (die in Abschnitt 8.4.1.3 beschrieben wurde) der Filter clean-traffic aktiviert. Die Testergebnisse weichen nur in Test 5 von den Ergebnissen in Abschnitt 8.4.1.3 ab. Libvirt verhindert also nicht, dass Dienste des Hosts erreicht werden können. Ein entsprechender Schutz kann jedoch anhand zusätzlicher individueller Filterregeln realisiert werden.

#	Bedrohung	Ergebnis	Bemerkung
1	Vorhandene Konnektivität mittels IPv4	bestanden	
2	Vorhandene Konnektivität mittels IPv6	nicht bestanden	IPv6-Verkehr wird durch den Filter clean-traffic verworfen.
3	Fehlende Trennung vom Management-Netzwerk	bestanden	
4	Fehlende Trennung vom Storage-Netzwerk	bestanden	
5	Erreichbarkeit von Diensten des Hosts durch den Gast	nicht bestanden	Der Netzwerkdienst SSH des Hosts ist vom Gast aus erreichbar.
6	Kommunikation mittels beliebiger Protokolle über Ethernet	bestanden	Nur IPv4, ARP und RARP sind erlaubt.
7	MAC-Address-Spoofing	bestanden	
8	IP-Address-Spoofing	bestanden	
9	ARP-Spoofing	bestanden	
10	ARP-Flooding	bestanden	
11	NDP-Spoofing/ RA-Spoofing	bestanden	IPv6-Verkehr wird durch den Filter clean-traffic verworfen.
12	DHCP-Spoofing	nicht bestanden	

Tabelle 39: Testergebnisse des libvirt-Netzwerks im Modus „isolated“ nach Härtung

8.4.4.2 Modus routed

8.4.4.2.1 Konfiguration

Auf beiden Hosts der Testumgebung (s. Kapitel 4) wird ein libvirt-Netzwerk im Modus routed erstellt. Die folgende Beschreibung wird verwendet, um das Netzwerk zu erzeugen. Hierbei ist die physische Schnittstelle des externen Gast-Netzwerks aktiv und mit IPv4- und IPv6-Adressen des Netzwerks konfiguriert.

```
<network>
        <name>routenet</name>
        <bridge name="virbr0" />
        <forward mode="route" />
```

```
        <ip address="10.100.100.1" netmask="255.255.255.0">
                <dhcp>
                        <range start="10.100.100.1" end="10.100.100.254" />
                </dhcp>
        </ip>
        <ip family="ipv6" address="2a01:198:5a1:100::1" prefix="64"/>
</network>
```

Im Rahmen der Netzwerkinitialisierung startet libvirt einen DHCP-Server auf dem Host (s. Abschnitt 3.6.4.1). Dieser Server verteilt gemäß der aufgeführten Netzwerkspezifikation die IPv4-Adressen 10.100.100.1 bis 10.100.100.254. Ferner ermöglicht der Host die IPv6-Autokonfiguration mit dem Netzwerkpräfix 2001:198:5a1:100.

Auf beiden Hosts werden jeweils zwei Gäste gestartet und mit dem Netzwerk verbunden. Die Konfiguration ist in Abb. 43, die entsprechende logische Netzwerktopologie in Abb. 44 dargestellt. Hierbei entspricht der virtuelle Switch der Linux-Bridge. Die Kommunikation mit den externen Netzwerken wird über einfache Filterregeln beschränkt (s. Abschnitt 3.6.4.1), die hier durch die Paketfilter dargestellt werden. Die Vermittlung zwischen Gästen und externen Netzwerken erfolgt über die Routing-Funktion des Linux-Kernels, hier als virtueller Router dargestellt. Bei dem realen Switch handelt es sich um den (in Abb. 43 nicht eingezeichneten) Switch, an den die physischen Netzwerkschnittstellen der Hosts angeschlossen sind. Der Router entspricht dem Router in Abb. 43 und dient zum Zugang zu externen Netzwerken.

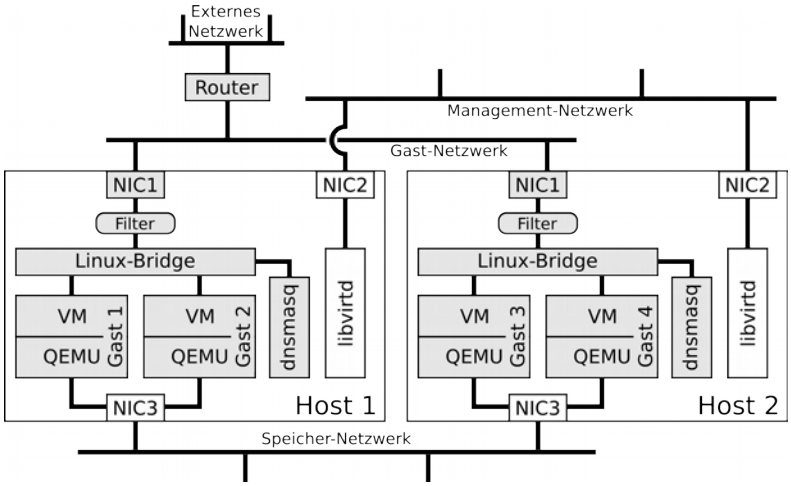

Abbildung 43: Konfiguration der Testumgebung mit einem libvirt-Netzwerk im Modus „routed“

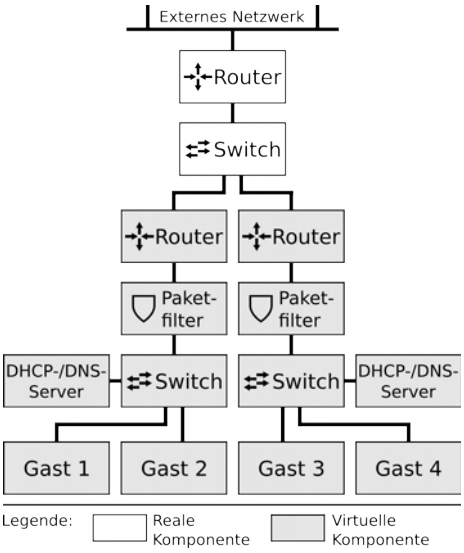

Abbildung 44: Logische Netzwerktopologie von libvirt-Netzwerken im Modus „routed“

8.4.4.2.2 Ergebnisse

Die Testdurchführung, deren Ergebnisse in Tabelle 40 zusammengefasst sind, zeigt, dass die Konfiguration eine Kommunikation mittels IPv4 und IPv6 grundsätzlich ermöglicht. Es besteht allerdings weder eine Trennung zwischen den Gästen und dem Management- oder Storage-Netzwerk noch zwischen den Gästen und dem Host. Entsprechend sind die Testfälle 3 bis 5 nicht bestanden. Der Grund hierfür ist, dass die Realisierung des virtuellen Netzwerks im Modus routed weitestgehend der Realisierung im Modus isolated

entspricht. Wesentliche Unterschiede sind die automatische Aktivierung von IP-Forwarding durch libvirt und das Fehlen einer Filterregel, die das Weiterleiten von Paketen untersagt. Somit ist es den Gästen grundsätzlich möglich, mit allen am Host angeschlossenen Netzwerken zu kommunizieren.

Eingeschränkt wird diese Kommunikation allerdings durch die in Abschnitt 3.6.4.1 beschriebenen Regeln zur Verhinderung von netzwerkbasierten Angriffen, wie z. B. IP-Spoofing. Ob Antwortpakete der Systeme, die der Gast kontaktiert hat, an diesen zurückgeleitet werden, hängt von der Konfiguration der realen Netzwerkkomponenten ab. Verhindern diese eine Rückleitung, so ist die Kommunikation lediglich unidirektional. Es lässt sich daher festhalten, dass libvirt den Zugang der Gäste nicht auf das Gast-Netzwerk beschränkt. Empfangene Pakete können an jeder Netzwerkschnittstelle ausgeleitet werden. Ob die Antwort eines kontaktierten Systems des Management- oder Storage-Netzwerks an den Gast zurückgeleitet wird, hängt jedoch typischerweise von den externen Netzwerkkomponenten ab.[106]

Des Weiteren gelten die Ergebnisse des Modus isolated bezüglich des Zugriffs auf Netzwerkdienste des Hosts analog. Eine strikte Trennung der Dienste des Hosts vom Gast existiert folglich nicht.

Ferner ist zu erwähnen, dass auch der Zugriff auf das externe Gast-Netzwerk geroutet und nicht geswitcht ist. Eine Kommunikation auf Layer 2 kann somit lediglich zwischen den Gästen eines Hosts, nicht jedoch zu dem externen Gast-Netzwerk erfolgen.

#	Bedrohung	Ergebnis	Bemerkung
1	Vorhandene Konnektivität mittels IPv4	bestanden	
2	Vorhandene Konnektivität mittels IPv6	bestanden	
3	Fehlende Trennung vom Management-Netzwerk	nicht bestanden	Je nach Konfiguration der realen Netzwerkkomponenten ist nur eine unidirektionale Kommunikation möglich.
4	Fehlende Trennung vom Storage-Netzwerk	nicht bestanden	Je nach Konfiguration der realen Netzwerkkomponenten ist nur eine unidirektionale Kommunikation möglich.
5	Erreichbarkeit von Diensten des Hosts durch den Gast	nicht bestanden	Der Netzwerkdienst SSH des Hosts ist vom Gast aus erreichbar.
6	Kommunikation mittels beliebiger Protokolle über Ethernet	nicht bestanden	
7	MAC-Address-Spoofing	nicht bestanden	
8	IP-Address-Spoofing	nicht bestanden	
9	ARP-Spoofing	nicht bestanden	
10	ARP-Flooding	nicht bestanden	
11	NDP-Spoofing/ RA-Spoofing	nicht bestanden	
12	DHCP-Spoofing	nicht bestanden	

Tabelle 40: Testergebnisse des libvirt-Netzwerks im Modus „routed“

106 Hier besteht die Gefahr, dass durch einen einfachen Verbindungstest der Anschein erweckt wird, dass eine Kommunikation nicht möglich ist, obwohl eine unidirektionale Datenübermittlung sehr wohl durchgeführt werden kann.

8.4.4.2.3 Härtung

Zur Härtung der Konfiguration wird analog zu der Härtung einer Linux-Bridge, die in Abschnitt 8.4.1.3 beschrieben wird, der Filter clean-traffic aktiviert. Darüber hinaus wird die Beschreibung des virtuellen Netzwerks angepasst. Diese wird um die Angabe einer Netzwerkschnittstelle erweitert, über die der Netzwerkverkehr exklusiv ausgeleitet werden soll. Die modifizierte Beschreibung ist im Folgenden aufgeführt. Die physische Netzwerkschnittstelle des externen Gast-Netzwerks wird hierbei als eth0 bezeichnet.

```
<network>
        <name>routenet</name>
        <bridge name="virbr0" />
        <forward mode="route" dev="eth0"/>
        <ip address="10.100.100.1" netmask="255.255.255.0">
                <dhcp>
                        <range start="10.100.100.1" end="10.100.100.254" />
                </dhcp>
        </ip>
        <ip family="ipv6" address="2a01:198:5a1:100::1" prefix="64"/>
</network>
```

Die Ergebnisse der Härtungsmaßnahmen sind in Tabelle 41 zusammengefasst und entsprechen denen des Modus isolated (s. Abschnitt 8.4.4.1.3).

#	Bedrohung	Ergebnis	Bemerkung
1	Vorhandene Konnektivität mittels IPv4	bestanden	
2	Vorhandene Konnektivität mittels IPv6	nicht bestanden	IPv6-Verkehr wird durch den Filter clean-traffic verworfen.
3	Fehlende Trennung vom Management-Netzwerk	bestanden	Die Gäste können jedoch die IP-Adresse des Hosts im Management-Netzwerk ermitteln (s. Abschnitt 8.2).
4	Fehlende Trennung vom Storage-Netzwerk	bestanden	Die Gäste können jedoch die IP-Adresse des Hosts im Storage-Netzwerk ermitteln (s. Abschnitt 8.2).
5	Erreichbarkeit von Diensten des Hosts durch den Gast	nicht bestanden	
6	Kommunikation mittels beliebiger Protokolle über Ethernet	bestanden	Nur IPv4, ARP und RARP sind erlaubt.
7	MAC-Address-Spoofing	bestanden	
8	IP-Address-Spoofing	bestanden	
9	ARP-Spoofing	bestanden	
10	ARP-Flooding	bestanden	
11	NDP-Spoofing/	bestanden	IPv6-Verkehr wird durch den Filter clean-traffic

#	Bedrohung	Ergebnis	Bemerkung
	RA-Spoofing		verworfen.
12	DHCP-Spoofing	nicht bestanden	

Tabelle 41: Testergebnisse des libvirt-Netzwerks im Modus „routed“ nach Härtung

8.4.4.3 Modus nat

8.4.4.3.1 Konfiguration

Auf beiden Hosts der Testumgebung (s. Kapitel 4) wird ein libvirt-Netzwerk im Modus nat erstellt. Hierbei ist die physische Schnittstelle des externen Gast-Netzwerks aktiv und mit einer IPv4- und IPv6-Adresse des Netzwerks konfiguriert.

```
<network>
        <name>natnet</name>
        <bridge name="virbr0" />
        <forward mode="nat" />
                <ip address="192.168.122.1" netmask="255.255.255.0">
                        <dhcp>
                                <range start="192.168.122.2"
end="192.168.122.254" />
                        </dhcp>
                </ip>
</network>
```

Auf beiden Hosts werden jeweils zwei Gäste gestartet und mit dem Netzwerk verbunden. Die Konfiguration unterscheidet sich von der in Abschnitt 8.4.4.2 beschriebenen nur durch die Tatsache, dass für die ausgehenden IPv4-Pakete eine Network Address Translation (NAT) stattfindet (vgl. Abschnitt 3.6.4.1).

8.4.4.3.2 Ergebnisse

Die Testdurchführung, deren Ergebnisse in Tabelle 42 zusammengefasst sind, zeigt, dass die Konfiguration eine Kommunikation mittels IPv4 und IPv6 grundsätzlich ermöglicht. Es besteht weder eine Trennung zwischen den Gästen und dem Management- oder Storage-Netzwerk noch zwischen den Gästen und dem Host. Entsprechend sind die Testfälle 3 bis 5 nicht bestanden.

Die Ergebnisse entsprechen denen des Modus routed (s. Abschnitt 8.4.4.2.2). Der Grund hierfür ist, dass die Konfiguration im Modus nat weitestgehend der im Modus routed entspricht (s. Abschnitt 3.6.4.1). Die Konfiguration unterscheiden sich jedoch darin, dass für alle aus dem Host ausgeleiteten Netzwerkpakete NAT stattfindet. Hierdurch ist nach der Ausleitung der Host als Absender im IP-Paket eingetragen.

Somit treten dieselben Risiken wie bei der Verwendung des Modus routed auf. Diese werden jedoch durch das NAT verschärft. So ist es einem externen System nicht möglich, anhand der IP-Absenderadresse zu ermitteln, ob das Paket vom Host erzeugt oder von einem Gast produziert und durch den Host lediglich weitergeleitet wurde. Auch führt das NAT dazu, dass Antworten der durch den Gast kontaktierten Systeme in aller Regel an den Gast zurückgeleitet werden.

#	Bedrohung	Ergebnis	Bemerkung
1	Vorhandene Konnektivität mittels IPv4	bestanden	IPv4-Verkehr wird geNATet.
2	Vorhandene Konnektivität mittels IPv6	bestanden	IPv6-Verkehr wird geroutet.
3	Fehlende Trennung vom Management-Netzwerk	nicht bestanden	
4	Fehlende Trennung vom Storage-Netzwerk	nicht bestanden	
5	Erreichbarkeit von Diensten des Hosts durch den Gast	nicht bestanden	
6	Kommunikation mittels beliebiger Protokolle über Ethernet	nicht bestanden	
7	MAC-Address-Spoofing	nicht bestanden	
8	IP-Address-Spoofing	nicht bestanden	
9	ARP-Spoofing	nicht bestanden	
10	ARP-Flooding	nicht bestanden	
11	NDP-Spoofing/RA-Spoofing	nicht bestanden	
12	DHCP-Spoofing	nicht bestanden	

Tabelle 42: Testergebnisse des libvirt-Netzwerks im Modus „nat“

8.4.4.3.3 Härtung

Die durchgeführten Härtungsmaßnahmen entsprechen denen des Modus routed. Insbesondere wird, wie im Folgenden aufgeführt, die Netzwerkschnittstelle spezifiziert, mit der Netzwerkverkehr exklusiv ausgeleitet werden soll. Hierbei wird die physische Netzwerkschnittstelle des externen Gast-Netzwerks als eth0 bezeichnet.

```
<network>
        <name>natnet</name>
        <bridge name="virbr0" />
        <forward mode="nat" dev="eth0"/>
                <ip address="192.168.122.1" netmask="255.255.255.0">
                        <dhcp>
                                <range start="192.168.122.2" \
        end="192.168.122.254" />
                        </dhcp>
                </ip>
</network>
```

Die Ergebnisse der Härtungsmaßnahmen sind in Tabelle 43 zusammengefasst. Sie entsprechen denen des Modus routed (s. Abschnitt 8.4.4.2.3).

#	Bedrohung	Ergebnis	Bemerkung
1	Vorhandene Konnektivität mittels IPv4	bestanden	
2	Vorhandene Konnektivität mittels IPv6	nicht bestanden	IPv6-Verkehr wird durch den Filter clean-traffic verworfen.
3	Fehlende Trennung vom Management-Netzwerk	bestanden	
4	Fehlende Trennung vom Storage-Netzwerk	bestanden	
5	Erreichbarkeit von Diensten des Hosts durch den Gast	bestanden	
6	Kommunikation mittels beliebiger Protokolle über Ethernet	bestanden	Nur IPv4, ARP und RARP sind erlaubt.
7	MAC-Address-Spoofing	bestanden	
8	IP-Address-Spoofing	bestanden	
9	ARP-Spoofing	bestanden	
10	ARP-Flooding	bestanden	
11	NDP-Spoofing/RA-Spoofing	bestanden	IPv6-Verkehr wird durch den Filter clean-traffic verworfen.
12	DHCP-Spoofing	nicht bestanden	

Tabelle 43: Testergebnisse des libvirt-Netzwerks im Modus „nat“ nach Härtung

8.4.5 Zusammenfassung

Auch wenn das Ziel aller beschriebenen Methoden die Anbindung der Gäste an ein physisches Gast-Netzwerk ist, so unterscheiden sich die sicherheitstechnischen Eigenschaften dieser Methoden doch erheblich.

Fast alle untersuchten Konfigurationen stellen die Trennung der Gäste vom Management- und Storage-Netzwerk auch ohne Härtungsmaßnahmen sicher. Lediglich für virtuelle libvirt-Netzwerke in den Modi routed und nat ist hierzu ein entsprechender Parameter in der Netzwerkbeschreibung notwendig. Besondere Vorsicht ist jedoch geboten, sofern auf dem Host IP-Adressen an Netzwerkschnittstellen gebunden werden, an die der Gast angeschlossen ist. Außer bei virtuellen libvirt-Netzwerken, die eine solche Adressvergabe benötigen und selbsttätig durchführen, führt dies in allen getesteten Konfigurationen zu einer zumindest teilweisen Aufhebung der Netzwerktrennung.

Ebenso verhindern die meisten der untersuchten Konfigurationen auch ohne Härtungsmaßnahmen, dass Gäste auf Netzwerkdienste des Host zugreifen. libvirt-Netzwerke stellen hier jedoch in allen drei Modi eine bedeutende Ausnahme dar. Tatsächlich werden entsprechende Beschränkungen nicht durch libvirt veranlasst. Vielmehr wird auf eine sichere Konfiguration des Betriebssystems vertraut. Eine fehlende

Absicherung durch das Betriebssystem führt dazu, dass Gäste prinzipiell auf die Managementdienste des Hosts zugreifen können.

Darüber hinaus schützt keine der untersuchten Konfigurationen vor typischen netzwerkbasierten Angriffen, wie z. B. ARP- oder IP-Spoofing. Zur Abwehr solcher Angriffe stehen je nach Anbindungsmethode zwei Ansätze zur Verfügung:

- Zum einen kann eine entsprechende Filterung des Netzwerkverkehrs auf dem Host vorgenommen werden. libvirt ermöglicht dies bei Verwendung virtueller Netzwerke oder einer vorkonfigurierten Linux-Bridge. Hierzu bietet libvirt eine Schnittstelle, mit der flexibel umfangreiche Filterregelwerke erstellt werden können. Es zeigt sich, dass dieser Mechanismus in der Lage ist, die analysierten Angriffsarten abzuwehren. Hierbei hat sich insbesondere der vordefinierte Filter clean-traffic als wirkungsvoll herausgestellt, auch wenn dieser nicht alle Angriffe erfolgreich abwehrt und eine Kommunikation mittels IPv6 gänzlich unterbindet. Darüber hinaus kann Open vSwitch derart konfiguriert werden, dass die untersuchten Angriffe durch den Host unterbunden werden. Dies muss jedoch von der eingesetzten Control-Plane umgesetzt werden.
- Zum anderen besteht die Möglichkeit, die Vermittlung und Filterung der Kommunikation auf externe Netzwerkkomponenten auszulagern. Hierzu eignet sich insbesondere macvtap im Modus vepa, da dieser Modus zur Ausleitung jeglicher Kommunikation aus dem Host führt. Dies ermöglicht eine strikte Trennung zwischen der eigentlichen Virtualisierung und der Verarbeitung der Kommunikation und kann somit zu einem erhöhten Sicherheitsniveau beitragen. Hierzu wird jedoch ein Switch benötigt, der den Hairpin-Modus unterstützt. Die weiteren Modi von macvtap sind jedoch von einem sicherheitstechnischen Standpunkt aus betrachtet nur bedingt empfehlenswert. Insbesondere sollte hierbei der Hairpin-Modus, sofern er unterstützt wird, stets deaktiviert sein. Der Modus bridge ermöglicht weder eine host-interne noch eine host-externe Filterung der Kommunikation zwischen den Gästen eines Hosts. Der Modus private stellt eine Erweiterung des Modus vepa dar und soll eine Kommunikation der Gäste untereinander gänzlich unterbinden. Es zeigt sich jedoch, dass sich diese Schutzfunktion unter Umständen umgehen lässt. Der Modus passthough birgt, neben der fehlenden Möglichkeit einer Filterung durch den Host, das Problem, dass lediglich eine exklusive Nutzung der physischen Netzwerkschnittstelle möglich ist.

9 Sicherheitsanalyse der Speicheranbindung

9.1 Zielsetzung

Eine wesentliche Voraussetzung für den sicheren Betrieb einer Virtualisierungsumgebung ist der Schutz der Festplattenabbilder. Es wird daher untersucht, auf welchem Wege ein solcher Schutz – basierend auf der in Kapitel 4 beschriebenen Testumgebung – realisiert werden kann. Darüber hinaus wird analysiert, wie dies bei Nutzung der netzwerkbasierten Speicherlösungen Ceph und GlusterFS möglich ist.

Zentrale Fragestellungen sind daher:

- Wie kann der Host, insbesondere QEMU, die Vertraulichkeit und Integrität der Festplattenabbilder schützen?
- Stellen Ceph und GlusterFS die Verfügbarkeit der Abbilder sowohl im Allgemeinen als auch während eines Angriffs sicher?
- Stellen Ceph und GlusterFS die Vertraulichkeit und die Integrität der gespeicherten Abbilder sicher?
- Gewährleisten Ceph und GlusterFS die Vertraulichkeit und die Integrität der Abbilder während der Datenübermittlung?
- In welcher Form ermöglichen Ceph und GlusterFS eine sichere gegenseitige Authentifizierung zwischen allen Kommunikationspartnern, die über ein Netzwerk interagieren?
- In welcher Form können mit Ceph und GlusterFS Zugriffe auf Festplattenabbilder autorisiert werden?

Schutzfunktionen der Softwarekomponenten werden entsprechend untersucht und auf ihre prinzipielle Tauglichkeit hin überprüft. Auf eine vollständige Schwachstellenanalyse einschließlich einer systematischen Quelltextanalyse wird jedoch verzichtet.

Die gesamte Untersuchung wird im Kontext des in Abschnitt 2.1 beschriebenen Bedrohungsszenarios durchgeführt. Es wird daher betrachtet, ob der Schutz in dem Fall gewährleistet ist, dass ein Angreifer gezielt auf das Netzwerk einwirkt. Eine Analyse darüber hinausgehender Bedrohungen, z. B. durch technische oder organisatorische Mängel, wird unterlassen. Ebenso wird eine Betrachtung der Performance, wie z. B. des Datendurchsatzes, nicht durchgeführt.

Für die Untersuchung wird Ceph in Version 10.2.2 und GlusterFS in Version 3.8.5 genutzt. Die Installation erfolgt auf den CentOS-7-Systemen, die in Kapitel 4 beschrieben wurden.

9.2 Allgemein

9.2.1 Verfügbarkeit

Eine wesentliche Voraussetzung für den sicheren Betrieb einer Virtualisierungsumgebung ist die Verfügbarkeit der Festplattenabbilder der Gäste. Ist ein Zugriff auf die virtuellen Festplatten gänzlich oder partiell nicht möglich, so führt dies zwangsweise zu einer Einschränkung der Verfügbarkeit des Gastes und im Extremfall zu dessen Totalausfall.

Die Bereitstellung der Festplattenabbilder kann entweder durch lokale Speicherung auf dem Host oder über ein Netzwerk erfolgen. Die Ausfallsicherheit bei lokaler Speicherung auf dem Host lässt sich durch gängige Maßnahmen erhöhen, wie z. B. durch redundante Speicherung auf mehreren Festplatten. Da diese Maßnahmen virtualisierungsunabhängig sind, wird an dieser Stelle nicht weiter auf sie eingegangen. Zu beachten ist jedoch, dass ein Komplettausfall des Hosts typischerweise zu einem langfristigen Ausfall des Gastes führt. Eine zeitnahe Reaktivierung des Gastes auf einem anderen Hosts ist in diesem Fall oftmals nicht möglich. Darüber hinaus kann QEMU keine Live-Migration durchführen, da hierzu das

Festplattenabbild den beiden an der Migration beteiligten Hosts zur Verfügung stehen muss. Bei Nutzung lokal gespeicherter Festplattenabbilder ist aufgrund der fehlenden Migrationsmöglichkeit ein Update zentraler Softwarekomponenten daher regelmäßig schwierig. Die Entscheidung für lokal gespeicherte Festplattenabbilder hat damit auch weitreichende Konsequenzen für das Patch-Management.

Die Verfügbarkeit der Festplattenabbilder bei netzwerkbasierter Speicherung ist hingegen in ganz erheblichem Maße von dem hierzu eingesetzten Produkt abhängig. Eine entsprechende Betrachtung für Ceph und GlusterFS wird in den Abschnitten 9.3.2 bzw. 9.4.2 durchgeführt.

9.2.2 Vertraulichkeit und Integrität

9.2.2.1 Strikte Zugriffsbeschränkung

Neben der Sicherstellung der Verfügbarkeit ist die Wahrung der Vertraulichkeit und Integrität der Festplattenabbilder von besonderer Bedeutung. Eine Standardmaßnahme, um dies sicherzustellen, besteht darin, sowohl den physischen als auch den logischen Zugriff auf Daten und datenverarbeitende Systeme zu beschränken und den Zugriff Unbefugter somit zu unterbinden.

Bei Verwendung eines netzwerkbasierten Speicher-Backends muss sich die Zugriffsbeschränkung auch auf das hierzu verwendete Netzwerk erstrecken. Insbesondere bei einer großen Anzahl von Speicherknoten und einer ausgedehnten räumlichen Verteilung kann dies einen erheblichen technischen wie organisatorischen Aufwand bedeuten. Eine wesentliche technische Maßnahme ist hierbei, die Anzahl an Systemen mit Zugriff auf das Speicher-Netzwerk zu minimieren. In komplexen Virtualisierungsumgebungen wird dies jedoch durch die große Anzahl an Hosts erschwert, die zur Ausführung der Gäste zwingend über einen Zugriff auf das Speicher-Backend verfügen müssen. Besondere Bedeutung gewinnen daher technische Maßnahmen zur Authentifizierung und Autorisierung der beteiligten Netzwerkdienste.

Sofern die Vertraulichkeit und die Integrität der Daten ausschließlich durch eine Zugriffsbeschränkung – also ohne Verschlüsselungs- und Signierungsmaßnahmen – gewährleistet werden soll, muss sich diese Maßnahme in aller Regel auf das gesamte System beziehen, das aus dem Host, dem Speicher-Netzwerk und dem Speicher-Backend besteht.

Ausnahmen können existieren, sofern anderweitige Maßnahmen ergriffen werden, wie z. B. eine Verschlüsselung und Signierung (s. Abschnitt 9.2.2.2).

9.2.2.2 Verschlüsselung und Signierung

9.2.2.2.1 Allgemein

Ein weiterer Weg, um die Vertraulichkeit und Integrität des Festplattenabbildes zu schützen, ist dessen Verschlüsselung und Signierung mit kryptografisch ausreichend starken Verfahren. Systeme, die ausschließlich auf derart geschützte Daten, nicht jedoch auf das verwendete Schlüsselmaterial Zugriff haben, brauchen zur Wahrung von Vertraulichkeit und Integrität regelmäßig keine weiteren Schutzmaßnahmen. Insbesondere bei Nutzung komplexer, verteilter Speicher-Backends kann dies den organisatorischen und technischen Aufwand, verglichen mit der Realisierung einer strikten Zugriffsbeschränkung, reduzieren.

Zu beachten ist jedoch, dass das Zugriffsverhalten des Gastes auf das Festplattenabbild prinzipiell Rückschlüsse auf die Datenverarbeitung des Gastes und somit auf die Daten selbst erlauben kann. Ist es einem Angreifer möglich, die Zugriffe auf ein verschlüsseltes Festplattenabbild zu beobachten, so kann im Allgemeinen nicht gänzlich ausgeschlossen werden, dass er hierdurch Klartextdaten rekonstruieren kann.

Für besonders sensible Daten können daher trotz Verschlüsselung und Signierung weitergehende Schutzmaßnahmen erforderlich sein.

9.2.2.2.2 Gastbasierte Umsetzung

Die Verschlüsselung und Signierung seiner Daten kann vom Gast selbst durchgeführt werden. Der Vorteil hiervon ist, dass die eingesetzten Verfahren unabhängig von den Möglichkeiten der Virtualisierungsumgebung umgesetzt werden können. Somit können die Maßnahmen an den jeweiligen Schutzbedarf des Gastes angepasst werden. Nachteilig ist jedoch, dass die Sicherheit der persistenten Daten somit wesentlich von den Gästen abhängt und dass deren Konfiguration individuell durchgeführt werden muss. Auch ist zu beachten, dass ein Zugriff des Hosts auf die entschlüsselten Daten und auf das verwendete Schlüsselmaterial nicht ausgeschlossen werden kann. Auch wenn die Entschlüsselung innerhalb des Gastes durchgeführt wird, so ist es dem Host prinzipiell möglich, auf die Daten des Hosts zuzugreifen. Die Verschlüsselung und Signierung durch den Gast schützt somit bei einer Kompromittierung des Speicher-Backends und des Netzwerks. Bei einer Kompromittierung des Hosts kann die Vertraulichkeit und Integrität der Daten auf diesem Wege jedoch nicht gewährleistet werden.

Ein weiterer Nachteil ist, dass eine lückenlose Integritätssicherung des Festplattenabbilds auf diesem Wege schwierig ist. Typischerweise werden Bestandteile des Gastes, die sehr früh während dessen Startvorgangs geladen werden (wie z. B. der Bootloader), nicht verifiziert. Solche Lücken in der Integritätssicherung lassen unter Umständen eine Kompromittierung des Festplattenabbildes und damit des Gastes zu. Darüber hinaus muss das Schlüsselmaterial – typischerweise während des Bootvorgangs – in den Gast eingebracht werden. Grund ist, dass eine ungeschützte Speicherung zusammen mit den verschlüsselten Daten ist aus naheliegenden Gründen nicht sinnvoll ist. Die manuelle Eingabe des Schlüsselmaterials beim Starten des Gastes ist jedoch aufwendig.

Überdies ähnelt der Vorgang der Verschlüsselung und Signierung innerhalb des Gastes prinzipiell dem Vorgang auf einem nicht virtualisierten System. Zu berücksichtigen ist jedoch, dass entsprechende Software typischerweise entworfen wurde, um Daten auf lokalen Datenträgern zu schützen. Hierbei wird gewöhnlich ausgeschlossen, dass das Zugriffsverhalten auf den Datenträger von außerhalb des Systems beobachtet werden kann. Sollte dies aufgrund der Verwendung eines netzwerkbasierten Speicher-Backends jedoch möglich sein, so kann hieraus eine Einschränkung der Vertraulichkeit resultieren (s. Abschnitt 9.2.2.2.1). Davon abgesehen, kann entsprechende Software aber weitgehend analog zu nicht virtualisierten Systemen eingesetzt werden. Aus diesem Grund unterbleibt eine weitergehende Betrachtung im Rahmen dieser Studie.

9.2.2.2.3 Hostbasierte Umsetzung

Prinzipiell kann die Verschlüsselung und Signierung auch durch den Host – und auf diesem Wege transparent für den Gast – durchgeführt werden. Vorteilhaft ist hierbei, dass ein identisches Schutzniveau bezüglich der Vertraulichkeit und Integrität der Festplattenabbilder für alle Gäste des Hosts erreicht werden kann. Bei Nutzung eines netzwerkbasierten Speicher-Backends werden die Daten hierdurch auch vor Einsichtnahme und Verfälschung während der Speicherung und der Übertragung geschützt. Separate Schutzmaßnahmen des Speicher-Backends zur Sicherung der Vertraulichkeit und Integrität sind daher nicht unbedingt notwendig.[107]

Da der Host grundsätzlich Einsicht in die Datenverarbeitung des Gastes nehmen kann, führt eine Verschlüsselung und Signierung durch den Host in aller Regel zu keinem geringeren Schutzniveau, als wenn Verschlüsselung und Signierung durch den Gast durchgeführt werden (s. Abschnitt 9.2.2.2.2).

QEMU ist imstande, das Festplattenabbild mit AES bei einer Schlüssellänge von 128 Bit zu verschlüsseln, sofern das Abbild im Format qcow2 gespeichert wird. Die Implementierung weist jedoch eine Reihe

107 Zu berücksichtigen ist jedoch eine mögliche Gefährdung der Vertraulichkeit durch die Analyse des Zugriffsverhaltens (s. Abschnitt 9.2.2.2.1).

gravierender Mängel auf. Hier ist vor allem die Nutzung der Sektorennummer als Initialisierungsvektor bei der Verschlüsselung von Sektoren zu nennen. Die Folge des gewählten Verfahrens ist eine prinzipielle Anfälligkeit für statistische Analysen. Hierdurch werden insbesondere Choosen-Plaintext-Angriffe ermöglicht (s. [QCOWE]). Darüber hinaus wird ein Wechsel des Schlüsselmaterials während der Ausführung des Gastes nicht unterstützt. Zudem wird keine Signierung ermöglicht. Im Allgemeinen ist eine Verfälschung des Abbildes daher nicht erkennbar. Dies ist insbesondere problematisch, da die mit AES verschlüsselten Chiffretext-Blöcke mithilfe des Modus Cipher Block Chaining (CBC) miteinander verknüpft werden. Wird der Chiffretext eines Sektors an eine andere Position innerhalb des Festplattenabbildes kopiert, so bleibt der Klartext (mit Ausnahme der ersten 16 Byte) unverändert. Auch ist es möglich, durch Änderung des Chiffretextes eines Blocks gezielt beliebige Bits im Klartext des nachfolgenden Blocks zu invertieren.[108] Aufgrund dieser Mängel sollte davon abgesehen werden, die Verschlüsselungsfunktion des Formates qcow2 zu nutzen. Eine Entfernung der Funktion aus QEMU ist von den Entwicklern geplant und kann als wahrscheinlich angesehen werden. Darüber hinausgehend bietet QEMU keine Funktionen zur Sicherung der Vertraulichkeit und Integrität von Festplattenabbildern.

Um dennoch einen entsprechenden Schutz durch den Host zu realisieren, können entsprechende Funktionen des Betriebssystems genutzt werden. Bedeutend ist hier dm-crypt im Zusammenhang mit Linux Unified Key Setup (LUKS). Diese Funktionen ermöglichen es, fast beliebige Blockgeräte zu verschlüsseln. Das Blockgerät kann hierbei sowohl ein lokaler Datenträger als auch ein netzwerkbasiertes Storage-Backend sein, wie z. B. Ceph oder GlusterFS. Das konkret zu verwendende Verschlüsselungsverfahren kann weitgehend frei konfiguriert werden. Um neben der Verschlüsselung auch einen Integritätsschutz zu realisieren, kann dm-integrity genutzt werden. Im Vergleich zu dm-crypt ist dm-integrity jedoch weit weniger verbreitet und erprobt. Eine weitergehende Analyse von LUKS, dm-crypt und dm-integrity wird im Rahmen dieser Studie nicht durchgeführt. Für weitere Informationen sei auf die Dokumentation [DMCRYPT] bzw. [DMINT] verwiesen. Erwähnenswert ist jedoch, dass libvirt die genannten Werkzeuge nicht unterstützt. Der Einsatz der genannten Werkzeuge erfordert daher regelmäßig einen nicht unerheblichen zusätzlichen administrativen Aufwand.

9.2.2.2.4 Backend-basierte Umsetzung

Darüber hinaus besteht die Möglichkeit, Festplattenabbilder auf dem Storage-Backend zu verschlüsseln und zu signieren. Das erzielte Schutzniveau ist jedoch geringer, als wenn die Verschlüsselung und Signierung durch den Gast oder den Host durchgeführt wird. Der Grund hierfür ist, dass einem Angreifer durch eine Kompromittierung eines Speicherknotens typischerweise nicht nur ein Zugriff auf die geschützten Daten, sondern auch auf das Schlüsselmaterial möglich ist.[109]

Sinnvoll können entsprechende Maßnahmen dagegen bei der Übermittlung über ein nicht vertrauenswürdiges Netzwerk sein. Dies kann verhindern, dass ein Angreifer mit Zugriff auf das Netzwerk die übertragenen Daten einsieht oder verfälscht. Eine entsprechende Betrachtung für Ceph und GlusterFS wird in Abschnitt 9.3 bzw. 9.4 vorgenommen.

9.3 Ceph

9.3.1 Funktionsweise

Ceph ist eine Softwarelösung zur hoch verfügbaren Speicherung großer Datenmengen. Die Daten werden hierbei auf einer variablen Anzahl von Speicherknoten redundant vorgehalten. Die Speicherknoten können logisch über mehrere Netzwerke und physisch über mehrere Standorte verteilt werden. Ermöglicht wird

108 Der Klartext des Blocks, dessen Chiffretext manipuliert wird, wird hierdurch jedoch unvorhersehbar verändert.
109 Ein entsprechender Schutz kann jedoch die Entsorgung von defekten Datenträgern vereinfachen, sofern durch Vernichtung des Schlüsselmaterials eine Rekonstruktion der gespeicherten Daten verhindert wird.

dies durch den Algorithmus CRUSH, der in [CRUSH] spezifiziert wird. Er bestimmt, auf welchen Speicherknoten Daten abzulegen bzw. aufzufinden sind. Die Designziele dieses Verfahrens sind vor allem die Sicherstellung einer hohen Verfügbarkeit der Daten und die Skalierbarkeit des Systems zur Speicherung großer Datenmengen[110]. Der Algorithmus ist insbesondere dafür ausgelegt, die Datenverfügbarkeit nicht nur bei einzelnen Ausfällen, wie z. B. bei einem Festplattendefekt, sondern auch bei gehäuften Schadensfällen, wie dem Ausfall eines Rechenzentrums, sicherzustellen. Die Umstände, unter denen eine Verfügbarkeit der Daten garantiert werden kann, hängen hierbei sowohl von der eingesetzten Hardware (insbesondere von der Anzahl der Speicherknoten) als auch von der Softwarekonfiguration ab. Weiterführende Informationen dazu sind der Dokumentation von Ceph ([CDOC]) zu entnehmen. Relevant ist jedoch, dass [CRUSH] stets von einem unbeabsichtigten Schadensfall ausgeht, nicht jedoch von einem beabsichtigten Angriff.

Ceph umfasst mehrere Netzwerkdienste, die den Algorithmus CRUSH nutzen. Die wesentlichen Dienste sind zum einen der Ceph Object Storage Daemon (OSD) und zum anderen der Ceph Monitor Daemon (Monitor). Ersterer dient zur Speicherung der Daten. Letzterer hat die primäre Aufgabe, die Verfügbarkeit der Daten sicherzustellen. Er leitet bei einem Ausfall eine Umverteilung der Daten ein, um den konfigurierten Redundanzgrad der Datenspeicherung aufrechtzuerhalten. Um einen Single-Point-of-Failure zu vermeiden, werden hierbei sowohl OSDs als auch Monitore mehrfach vorgehalten. Aus dem gleichen Grund greift auch der Client (d. h. der vom Host ausgeführte QEMU-Prozess) mithilfe des Algorithmus CRUSH direkt auf die jeweiligen OSDs zu. Letzteres erfordert, dass dem Host der Netzwerkzugriff auf alle OSDs ermöglicht wird. QEMU speichert die Festplattenabbilder hierbei als RADOS Block Device (RBD). Der eigentliche Zugriff erfolgt mit dem Netzwerkprotokoll RADOS. Alternative Speicherverfahren, wie z. B. das Netzwerkdateisystem CephFS und der hiermit verbundene Ceph Metadata Server Daemon (MDS), sind bei Nutzung von QEMU nicht relevant und werden daher im Rahmen dieser Studie nicht betrachtet.

9.3.2 Verfügbarkeit

Ceph wurde von Grund auf mit dem Ziel entworfen, Ausfälle kompensieren zu können. Hierzu lässt sich Ceph, ebenso wie der zugrunde liegende Algorithmus CRUSH, umfangreich konfigurieren (s. [CDOC] und [CRUSH]). Dies umfasst insbesondere das Verfahren zur Wahl der Speicherknoten, auf denen ein Datum abgelegt werden soll. Beispielsweise kann festgelegt werden, dass dies stets auf Speicherknoten vorgehalten werden soll, die über unabhängige Netzwerke angebunden oder an unterschiedlichen Standorten betrieben werden.

Zu fragen ist hingegen, ob ein gezieltes Einwirken auf das verteilte System dessen Verfügbarkeit stören kann. Sofern es einem Angreifer möglich ist, die Netzwerkkommunikation des Hosts mit den Speicherknoten gänzlich zu unterbinden, kann die Verfügbarkeit der Gäste offensichtlich nicht gewährleistet werden. Ein Angreifer, der, wie im Bedrohungsszenario aus Abschnitt 2.1 beschrieben, uneingeschränkte Kontrolle über das Netzwerk hat, kann somit die Ausführung der Gäste unterbinden.

Ein Angreifer, der nur eingeschränkt oder nur auf einen Teil des Netzwerks einwirken kann, kann die Verfügbarkeit ebenfalls stören, sofern auf eine Authentifizierung (s. Abschnitt 9.3.3) verzichtet wird. In einem solchen Fall ist es dem Angreifer zum einen möglich, auf die gespeicherten Daten zuzugreifen und diese zu manipulieren undzu löschen. Zum anderen besteht die Gefahr, dass er in die Interaktion zwischen den OSDs und den Monitoren eingreift. Hierdurch könnte die automatische Umverteilung von Daten gestört und ein Datenverlust herbeigeführt werden.

110 [CPB] zeigt beispielsweise, dass eine hoch verfügbare Datenspeicherung im mehrstelligen Petabyte-Bereich realisierbar ist.

9.3.3 Authentifizierung

Um die Gefahr durch einen Angreifer der Zugriff auf das Netzwerk hat zu reduzieren, bietet Ceph eine gegenseitige Authentifizierung der Kommunikationspartner. Realisiert wird sie mit dem Protokoll CephX, dessen Kern ein ticketbasiertes Authentifizierungsverfahren ist.[111] CephX erfordert die Erstellung eines individuellen kryptografischen Schlüssels für jedes Subjekt, d. h. für OSD, Monitor und Client. Diese Schlüssel werden nicht nur dem jeweiligen Subjekt, sondern auch den Monitor-Systemen bekannt gemacht. Die Monitor-Systeme stellen auf Anfrage Tickets aus, die das anfragende Subjekt gegenüber genau einem anderen Subjekt ausweisen. Tickets werden mit dem Advanced Encryption Standard (AES) und dem Schlüssel des anfragenden Subjekts verschlüsselt übertragen. Damit wird sichergestellt, dass nur das anfragende Subjekt das Ticket entschlüsseln und verwenden kann. Sollte ein Angreifer dennoch Zugriff auf ein entschlüsseltes Ticket bekommen, so kann er sich hiermit lediglich gegenüber dem im Ticket angegebenen Subjekt ausweisen. Wird beispielsweise ein OSD so kompromittiert, dass es einem Angreifer möglich ist, die an diesen OSD übersandten Tickets einzusehen, so kann er diese Tickets nicht nutzen, um sich gegenüber einem anderen OSD auszuweisen.

Besonders sicherheitskritisch sind in diesem Verfahren die Monitore, da jeder von ihnen sämtliche kryptografischen Schlüssel vorhält. Dies ist erforderlich, damit jeder Monitor Tickets ausstellen kann und somit ein Single-Point-of-Failure im Zusammenhang mit der Authentifizierung vermieden werden kann. Die Kompromittierung eines einzelnen Monitors hat allerdings typischerweise die Kompromittierung des gesamten Speicher-Backends und aller gespeicherten Daten zur Folge.

Mit CephX steht somit ein hoch verfügbares Verfahren zur Verfügung, mit dem eine prinzipiell sichere gegenseitige Authentifizierung der beteiligten Subjekte möglich ist. Eine vollständige Schwachstellenanalyse des Designs und der Implementierung dieses Verfahrens wird im Rahmen dieser Studie jedoch nicht durchgeführt. An dieser Stelle wird daher darauf verzichtet, die Sicherheit der Authentifizierung mittels CephX abschließend zu bewerten.

9.3.4 Autorisierung

Auf Basis der gegenseitigen Authentifizierung, die sich mit CephX realisieren lässt, wird eine Autorisierung des jeweiligen Kommunikationspartners durchgeführt. Bei einem Zugriff auf ein Festplattenabbild wird überprüft, ob dem Client ein entsprechendes Zugriffsrecht auf den Storage-Pool eingeräumt wurde, der das Abbild enthält. Die Autorisierung erfolgt damit nicht unmittelbar bezogen auf das Festplattenabbild, sondern mittelbar auf den entsprechenden Storage-Pool. Dies hat zur Folge, dass Abbilder innerhalb des gleichen Storage-Pools effektiv identische Zugriffsberechtigungen aufweisen.

Dieses Verfahren kann durch sogenannte Namespaces ergänzt werden. Ein Namespace dient hierbei zur Gruppierung der in einem Storage-Pool gespeicherten Festplattenabbilder. In jedem Namespace können individuelle Autorisierungsinformationen hinterlegt werden. Somit ist es möglich, den Zugriff auf Abbilder des gleichen Storage-Pools unterschiedlich zu autorisieren. Prinzipiell ist es auf diesem Wege möglich, für jedes Abbild einen eigenen Namespace zu erzeugen und somit individuelle Zugriffsrechte zu vergeben. Dies ist jedoch wenig praktikabel. Der primäre Zweck von Namespaces ist vielmehr die Nutzung von Storage-Pools durch verschiedene Mandanten.

Die Autorisierung des Zugriffs auf die Festplattenabbilder ist somit in der Praxis recht grobgranular. Dies führt dazu, dass den Hosts typischerweise Zugriff auf eine große Anzahl oder gar alle gespeicherten Festplattenabbilder gewährt wird. Es ist somit nur sehr bedingt möglich, den Host derart zu beschränken, dass er nur auf Abbilder zugreifen kann, die er zur Ausführung seiner Gäste tatsächlich benötigt.[112]

111 Eine rudimentäre Dokumentation dieses Verfahrens findet sich in der Quelltextdatei *src/auth/cephx/CephxProtocol.h*. Einige Details dieser Dokumentation sind jedoch anscheinend veraltet und nicht mehr zutreffend.

112 Feingranularere Zugriffsbeschränkungen sind jedoch mit CephFS möglich, das ein POSIX-konformes verteiltes Dateisystem realisiert. CephFS ist jedoch zur Bereitstellung von Festplattenabbildern (insbesondere in

9.3.5 Vertraulichkeit

Ceph stellt die Vertraulichkeit der Daten ausschließlich über eine logische Zugangsbeschränkung mithilfe von Authentifizierung (s. Abschnitt 9.3.3) und Autorisierung (s. Abschnitt 9.3.4) sicher. Eine Verschlüsselung der Daten im Rahmen der Speicherung oder der Übermittlung wird nicht unterstützt. Die Daten werden auch dann unverschlüsselt übersendet, wenn das Protokoll CephX (s. Abschnitt 9.3.2) genutzt wird.

Eine Verschlüsselung des Datenträgers kann jedoch durch das Betriebssystem erfolgen. Die Umsetzung kann hierbei analog zu dem in Abschnitt 9.2.2.2.3 erläuterten Vorgehen erfolgen.

9.3.6 Integrität

Um die Integrität der auf mehreren OSDs redundant vorgehaltenen Daten zu schützen, werden diese regelmäßig auf Übereinstimmung hin überprüft. Dies geschieht auf zwei Arten:

- Zum einen durch den verhältnismäßig ressourcenschonenden Abgleich von Metadaten, wie der Größe und des Zeitpunkts der letzten Modifikation. Standardmäßig erfolgt dies täglich.
- Darüber hinaus werden die tatsächlich gespeicherten Daten abgeglichen. Standardmäßig erfolgt dies wöchentlich. Auf diesem Wege können Verfälschungen der Daten erkannt und durch die redundante Datenspeicherung korrigiert werden. Das Ziel ist hierbei, die Auswirkungen eines unbeabsichtigten Fehlers, insbesondere eines Hardwaredefekts, zu minimieren.

Das Erkennen gezielter Verfälschungen der Daten durch einen Angreifer liegt hingegen nicht im Fokus. Zwar kann solchen Verfälschungen mittels Authentifizierung (s. Abschnitt 9.3.3) und Autorisierung (s. Abschnitt 9.3.4) vorgebeugt werden. Gelingt es einem Angreifer aber dennoch, schreibenden Zugriff auf die Daten zu erlangen, so kann die Erkennung entsprechender Manipulationen durch Ceph nicht garantiert werden. Dies ist praktisch jedoch wenig relevant, da die Integrität von Daten, die von einem kompromittierten Speicherknoten stammen, ohnehin nicht ohne Weiteres angenommen werden darf.

Praxisrelevanter ist die Frage nach einer Integritätssicherung während der Datenübermittlung über das Netzwerk. Ceph ermöglicht hierzu die Signierung der Nachrichten, sofern das Protokoll CephX (s. Abschnitt 9.3.2) verwendet wird.[113] Dies wird umgesetzt, indem eine Prüfsumme über das zu versendende Paket gebildet[114] und in verschlüsselter Form[115] an die Nachricht angehängt wird. Das für die Signierung verwendete Schlüsselmaterial stammt hierbei aus dem Authentifizierungsverfahren, das in Abschnitt 9.3.2 beschrieben wurde. Problematisch ist hierbei insbesondere, dass die Prüfsumme mithilfe des Algorithmus CRC32 gebildet wird. Aufgrund der mangelhaften Kollisionsresistenz von CRC32 ist daher zu mutmaßen, dass Nachrichten unter Beibehaltung ihrer Prüfsumme verfälscht werden können. Hierbei würde die Signatur ihre Gültigkeit nicht verlieren, und dem empfangenden System wäre es im Allgemeinen nicht möglich, die Verfälschung zu erkennen. Der von Ceph umgesetzte Integritätsschutz der Netzwerkkommunikation ist daher als schwach zu bewerten.

Zusammenhang mit QEMU) wenig gebräuchlich und nur bedingt praktikabel. Aus diesem Grund wird auf eine entsprechende Betrachtung verzichtet (s. auch Abschnitt 9.3.1).

113 Die Signierung kann durch Konfiguration deaktiviert werden, ist standardmäßig jedoch aktiviert.

114 Präziser gesagt wird nicht eine einzige, sondern jeweils eine Prüfsumme für drei verschiedene Bestandteile der Nachricht berechnet.

115 Konkret wird AES genutzt.

9.4 GlusterFS

9.4.1 Funktionsweise

GlusterFS ist eine Softwarelösung, um große Datenmengen netzwerkbasiert hoch verfügbar zu speichern. Realisiert wird dies, indem Daten redundant auf mehreren Speicherknoten vorgehalten werden. Der Aufwand für Einrichtung und Betrieb ist hierbei verhältnismäßig gering. Die Konfigurationsmöglichkeiten haben, insbesondere im Vergleich zu Ceph (s. Abschnitt 9.3), einen verhältnismäßig geringen Umfang. Von besonderer Bedeutung ist der Betriebsmodus der Storage-Pools. Im Wesentlichen stehen hier die Betriebsmodi Distributed, Striped und Replicated zur Verfügung. Die ersten beiden Modi dienen dazu, die Speicherkapazität mehrerer Speicherknoten zusammenzufassen und somit die Gesamtspeicherkapazität zu maximieren. Der Betriebsmodus Replicated bewirkt hingegen eine redundante Datenspeicherung. Ferner existieren Mischformen der aufgeführten Betriebsmodi, die deren Funktionsprinzipien verbinden.

Ein zentraler Prozess zur Steuerung der Datenverteilung existiert nicht. Die Auswahl der für ein Datum zu verwendenden Speicherknoten erfolgt stattdessen durch den Client, beispielsweise QEMU anhand eines Hash-Verfahrens.

Beim Ausfall eines Speicherknotens muss dieser ersetzt und die Umverteilung der Daten manuell initialisiert werden. Eine automatische Umverteilung der Daten zur Wiederherstellung der redundanten Datenspeicherung erfolgt nicht. Infolgedessen ist der administrative Aufwand im Fehlerfall verhältnismäßig hoch. GlusterFS ist somit weniger gut geeignet, um sehr große Datenmengen hoch verfügbar zu speichern. Hierfür sind automatisch umverteilende Systeme, wie z. B. Ceph (s. Abschnitt 9.3), besser geeignet.

Der Zugriff durch QEMU erfolgt typischerweise mithilfe der Softwarebibliothek libgfapi. Alternativ können QEMU die Abbilder zur Verfügung gestellt werden, indem der Storage-Pool über das Betriebssystem des Hosts mittels Filesystem in Userspace (FUSE) in den Verzeichnisbaum eingebunden wird. Darüber hinaus existiert eine Reihe weiterer Netzwerkprotokolle, mit denen ein Zugriff möglich ist. Diese sind jedoch im Rahmen der Virtualisierung mittels QEMU nicht relevant und werden daher nicht betrachtet. Die Kommunikation sowohl zwischen den Speicherknoten als auch zwischen den Clients und den Speicherknoten kann mit Transport Layer Security (TLS) abgesichert werden. TLS wird jedoch nicht von der Softwarebibliothek libgfapi unterstützt.

9.4.2 Verfügbarkeit

Sofern GlusterFS in einem nicht redundant speichernden Modus, wie z. B. Distributed oder Striped betrieben wird, führt ein Ausfall eines Speicherknotens zu einem Datenverlust. Bei Verwendung des Modus Replicated kann ein solcher Ausfall kompensiert werden. Ob eine Verfügbarkeit auch bei einem mehrfachen Versagen gewährleistet werden kann, hängt entscheidend von der Anzahl der Speicherknoten ab, auf denen ein Datum redundant gespeichert wird. Dieser Redundanzgrad kann weitgehend frei gewählt werden. Das Verfahren zur Wahl der jeweiligen Speicherknoten ist hingegen nur recht grob konfigurierbar. Dies erschwert die Kompensation mehrfacher, miteinander zusammenhängender Ausfälle.

Neben der in Echtzeit durchgeführten redundanten Speicherung unterstützt GlusterFS die regelmäßige Übertragung der Daten auf alternative Speicherknoten. Diese Geo-Replication genannte Funktion dient primär dazu, eine Sicherungskopie an einem alternativen Standort anzulegen. Ob die Nutzung eines möglicherweise veralteten Festplattenabbilds nach einem katastrophalen Versagen des Speicher-Backends sinnvoll ist, hängt hierbei vor allem von der Funktion des Gastes ab.

Nach einem Ausfall eines Speicherknotens muss dieser ersetzt und die Umverteilung der Daten manuell initialisiert werden. Im Fehlerfall ist daher mit einer längerfristigeren Einschränkung des Redundanzgrades zu rechnen, als dies bei einer automatischen Umverteilung der Fall wäre.

Sofern es einem Angreifer möglich ist, die Netzwerkkommunikation des Hosts mit den Speicherknoten gänzlich zu unterbinden, kann die Verfügbarkeit der Gäste offensichtlicherweise nicht gewährleistet werden. Ein Angreifer, der, wie im Bedrohungsszenario aus Abschnitt 2.1 beschrieben, uneingeschränkte Kontrolle über das Netzwerk hat, kann somit die Ausführung der Gäste unterbinden.

Ein Angreifer, der nur eingeschränkt auf das Netzwerk einwirken kann, kann die Verfügbarkeit regelmäßig stören, sofern auf eine Authentifizierung (s. Abschnitt 9.4.3) verzichtet wird. In einem solchen Fall ist es ihm insbesondere möglich, Festplattenabbilder zu löschen.

9.4.3 Authentifizierung

GlusterFS ermöglicht eine gegenseitige Authentifizierung der Kommunikationspartner, sofern TLS genutzt wird. Hierzu werden Zertifikate zum Identitätsnachweis ausgetauscht. Die Echtheit des Zertifikats kann auf zweierlei Wegen überprüft werden:

- Zum einen kann das Zertifikat unmittelbar mit einer Liste von lokal gespeicherten vertrauenswürdigen Zertifikaten abgeglichen werden.
- Zum anderen kann die Prüfung auf Basis eines hinterlegten Stammzertifikats einer vertrauenswürdigen Zertifikatsautorität durchgeführt werden, die das zu prüfende Zertifikat signiert hat.

Auch wenn keine detaillierte Quellcodeanalyse des Verfahrens im Rahmen dieser Studie durchgeführt wird, so kann es zumindest prinzipiell als sicher angesehen werden. Zu beachten ist jedoch, dass das tatsächliche Schutzniveau erheblich von der jeweiligen Konfiguration von TLS – insbesondere von den verwendeten Algorithmen – abhängt (s. [TLSCS]).

9.4.4 Autorisierung

Auf Basis der Authentifizierung mithilfe von Zertifikaten (s. Abschnitt 9.4.3) ermöglicht GlusterFS eine rudimentäre Autorisierung von Clients, die auf Storage-Pools zugreifen wollen. Hierzu kann jedem Storage-Pool unabhängig voneinander eine Namensliste von zugriffsberechtigten Kommunikationspartnern hinterlegt werden. Im Rahmen der Autorisierung wird überprüft, ob der in dem Zertifikat aufgeführte Common Name (CN) des jeweiligen Kommunikationspartners in der Liste aufgeführt ist. Ist dies nicht der Fall, so wird der Zugriff verweigert. Da das Verfahren weitestgehend auf TLS basiert, kann es zumindest prinzipiell als zuverlässig angesehen werden (s. Abschnitt 9.4.3).

Darüber hinaus ist es möglich, den Zugriff auf Basis von IP-Adressen zu beschränken. Da das Fälschen von IP-Adressen im Allgemeinen einfach möglich ist, bietet diese Art der Autorisierung nur einen geringen Schutz. Durch zusätzliche Schutzmaßnahmen kann das Fälschen von IP-Adressen aber erschwert und das Schutzniveau einer IP-basierten Zugriffskontrolle erhöht werden.

Sowohl bei Nutzung von Zertifikaten als auch von IP-Adressen erfolgt die Autorisierung auf Basis des Storage-Pools, in dem das jeweilige Festplattenabbild gespeichert ist. Die Autorisierung des Zugriffs auf die Festplattenabbilder ist somit grundsätzlich recht grobgranular. Dies führt dazu, dass den Hosts typischerweise Zugriff auf eine große Anzahl oder sogar auf alle gespeicherten Festplattenabbilder gewährt wird. Den Host derart zu beschränken, dass er nur auf Abbilder zugreifen kann, die er zur Ausführung seiner Gäste tatsächlich benötigt, ist somit nur sehr bedingt umsetzbar.

9.4.5 Vertraulichkeit

GlusterFS stellt die Vertraulichkeit der Daten primär über eine logische Zugangsbeschränkung mittels Authentifizierung (s. Abschnitt 9.4.3) und Autorisierung (s. Abschnitt 9.4.4) sicher. Eine Speicherung der Daten in verschlüsselter Form ist nicht vorgesehen. Eine Verschlüsselung des Datenträgers kann jedoch

durch das Betriebssystem erfolgen. Die Umsetzung kann hierbei analog zu dem Vorgehen erfolgen, das in Abschnitt 9.2.2.2.3 erläutert wurde.

Im Rahmen der Datenübermittlung kann eine wirksame Verschlüsselung mittels TLS durchgeführt werden. Zu beachten ist jedoch, dass das tatsächliche Schutzniveau erheblich von der jeweiligen Konfiguration von TLS – insbesondere von den verwendeten Algorithmen – abhängt (s. [TLSCS]).

9.4.6 Integrität

Um die Integrität der auf mehreren Speicherknoten redundant vorgehaltenen Daten zu schützen, können diese regelmäßig auf Übereinstimmung hin überprüft werden. Standardmäßig ist diese Funktion jedoch deaktiviert. Eine Veränderung der Daten wird standardmäßig daher nicht festgestellt. Wird die bitrot genannte Funktion aktiviert, so geschieht ein Abgleich aller Daten standardmäßig alle 14 Tage. Das Ziel ist hierbei, die Auswirkungen eines unbeabsichtigten Fehlers, insbesondere eines Hardwaredefekts, zu minimieren.

Das Erkennen gezielter Verfälschungen der Daten durch einen Angreifer liegt hingegen nicht im Fokus. Zwar kann solchen Verfälschungen mithilfe von Authentifizierung (s. Abschnitt 9.3.3) und Autorisierung (s. Abschnitt 9.3.4) vorgebeugt werden. Gelingt es einem Angreifer aber dennoch, schreibenden Zugriff auf die Daten zu erlangen, so kann nicht garantiert werden, dass GlusterFS entsprechende Manipulationen erkennt. Dies ist praktisch jedoch wenig relevant, da die Integrität von Daten, die von einem kompromittierten Speicherknoten stammen, ohnehin nicht ohne Weiteres angenommen werden darf.

Im Rahmen der Datenübermittlung kann eine wirksame Integritätssicherung mittels TLS durchgeführt werden. Zu beachten ist jedoch, dass das tatsächliche Schutzniveau erheblich von der jeweiligen Konfiguration von TLS – insbesondere von den verwendeten Algorithmen – abhängt (s. [TLSCS]).

9.5 Zusammenfassung

Es lässt sich festhalten, dass QEMU selbst keine effektiven Maßnahmen zur Verfügung stellt, um die Vertraulichkeit und Integrität der Festplattenabbilder zu gewährleisten. Die hierzu angebotene Funktion der Verschlüsselung der Abbilder weist gravierende Mängel auf. Von einer Verwendung wird daher abgeraten.

Prinzipiell können aber allgemeine Funktionen des Betriebssystems, insbesondere zur Verschlüsselung und Signierung, genutzt werden. Hierzu stehen die Werkzeuge dm-crypt, dm-integrity und LUKS zur Verfügung. Mit diesen Werkzeugen kann der Host einen effektiven Schutz der Vertraulichkeit und Integrität herstellen. Zu beachten ist jedoch, dass das entsprechende Vorgehen nicht direkt von libvirt unterstützt wird. Ein entsprechender hostbasierter Schutz erfordert daher einen nicht unerheblichen erhöhten administrativen Aufwand.

Im Hinblick auf den Einsatz der netzwerkbasierten Speicherlösungen Ceph und GlusterFS ist festzustellen, dass beide prinzipiell geeignet sind, um Festplattenabbilder hochverfügbar zu speichern. Dies wird jedoch eingeschränkt, sobald ein Angreifer über das Netzwerk gezielt auf die Systeme einwirken kann. Offensichtlich ist eine Verfügbarkeit nicht zu gewährleisten, wenn der Angreifer eine uneingeschränkte Kontrolle über das Netzwerk erlangen kann. Auch wenn der Angreifer nur eingeschränkt auf das Netzwerk einwirken kann (z. B. durch einen einfachen logischen Netzwerkzugang), besteht dennoch ein erhebliches Gefährdungspotenzial. Dies gilt insbesondere, wenn auf eine Authentifizierung und eine Autorisierung durch das Speicher-Backend verzichtet wird. Ebenso birgt eine ungesicherte Klartextkommunikation in einem solchen Fall regelmäßig Risiken.

Eine gegenseitige Authentifizierung kann Ceph mit dem proprietären Protokoll CephX durchführen. Den Kern bilden hierbei Tickets, die ein Subjekt gegenüber einem anderen Subjekt ausweisen. Der Schutz der Tickets basiert auf einer symmetrischen Verschlüsselung. Die jeweiligen Schlüssel sind den Monitor-Systemen bekannt, die die Tickets ausstellen und die deshalb besonders schützenswert sind. Das Verfahren

ist prinzipiell geeignet, um die Kommunikationspartner gegenseitig zu authentifizieren. Eine vollständige Schwachstellenanalyse wurde im Rahmen dieser Studie jedoch nicht durchgeführt.

GlusterFS nutzt sowohl zur Sicherung der Netzwerkkommunikation als auch zur Authentifizierung und Autorisierung TLS. Die gegenseitige Authentifizierung erfolgt hierbei über die im Rahmen von TLS genutzten Zertifikate. TLS wird ferner verwendet, um die Netzwerkkommunikation sowohl zu verschlüsseln als auch zu signieren. Auch wenn das genaue Schutzniveau des Verfahrens von einer Reihe von Details (wie z. B. den verwendeten Algorithmen) abhängt, so kann es doch als prinzipiell sicher beurteilt werden.

Hingegen erfolgt die Kommunikation eines auf Ceph basierenden Speicher-Backends grundsätzlich unverschlüsselt. Eine Signierung wird bei Verwendung von CephX durchgeführt. Diese ist jedoch als kryptografisch schwach zu bewerten.

Sowohl Ceph als auch GlusterFS ermöglichen es, den Zugriff auf die gespeicherten Daten zu beschränken. Die zugrunde liegenden Autorisierungsverfahren sind jedoch nur recht grob konfigurierbar. Typischerweise wird ein Host autorisiert, auf eine große Anzahl von Festplattenabbildern zuzugreifen. Den Host derart zu beschränken, dass er nur auf Abbilder zugreifen kann, die er zur Ausführung seiner Gäste tatsächlich benötigt, ist sowohl mit Ceph als auch mit GlusterFS nur sehr bedingt umsetzbar.

Weder Ceph noch GlusterFS unterstützen ferner eine Verschlüsselung der gespeicherten Daten. Prinzipiell kann diese aber durch das Betriebssystem der Speicherknoten durchgeführt werden.

Die Integrität der gespeicherten Daten kann durch regelmäßigen Vergleich der redundanten Kopien überprüft werden. Ceph führt dies standardmäßig durch. Für GlusterFS ist die Funktion standardmäßig deaktiviert. Ein effektiver Schutz vor gezielten Verfälschungen wird durch die jeweiligen Verfahren im Allgemeinen nicht erzielt.

Zwar stellen Ceph und GlusterFS mehrere Schutzfunktionen bereit. Es kann dennoch resümiert werden, dass die Vertraulichkeit, Integrität und Verfügbarkeit der Festplattenabbilder üblicherweise primär durch eine strikte physische sowie logische Zugangsbeschränkung sicherzustellen sind. Die aufgeführten Schutzmaßnahmen können das Schutzniveau jedoch anheben. In dem Fall, dass ein Angreifer die Zugangsbeschränkung umgeht (z. B. durch Kompromittierung eines Hosts), stellen sie potenziell effektive Schutzmaßnahmen dar. Dies gilt umso mehr, wenn eine große Anzahl an Hosts und Speicherknoten betrieben wird und diese räumlich stark verteilt sind.

10 Sicherheitsanalyse von OpenStack

10.1 Zielsetzung

Wenngleich QEMU und libvirt die Basis für die in dieser Studie untersuchte Virtualisierung bilden, so sind zum Betrieb komplexer Cloud-Umgebungen doch weitere Softwarekomponenten nötig. Ein Projekt, das komplexe Cloud-Umgebungen ermöglicht, ist OpenStack. Es umfasst eine Reihe von netzwerkfähigen Anwendungen, die die Verwaltung einer komplexen Cloud-Infrastruktur ermöglichen.

Daher wird eine Betrachtung von OpenStack durchgeführt, und zwar in der zum Zeitpunkt der Erstellung dieser Studie aktuellen Version 8.0.0 (Mitaka). Eine erschöpfende Analyse von OpenStack wird jedoch nicht durchgeführt. Diese würde nicht nur aufgrund der Komplexität von OpenStack erschwert werden: Da OpenStack einer raschen Weiterentwicklung unterliegt, würden die damit verbundenen Änderungen der Anwendungen eine allgemeine Analyse schnell obsolet machen. Nicht zuletzt sind komplexe Cloud-Umgebungen in der Praxis oftmals hoch individuell. OpenStack unterstützt solche Individualisierungen – insbesondere durch die lose Kopplung der Anwendungen und aufgrund seines quelloffenen Charakters – auf vielfältige Weise.

Eine allgemeingültige Bewertung der Sicherheit von Virtualisierungsumgebungen, die auf OpenStack basieren, ist daher nur sehr bedingt möglich. Vielmehr müssen die konkreten Sicherheitsanforderungen im Zusammenhang mit der jeweiligen Umsetzung und den konzeptionellen und architektonischen Besonderheiten der spezifischen Installation betrachtet werden, um eine belastbare Sicherheitseinschätzung einer konkreten Umgebung zu erlangen.

Der in dieser Untersuchung gewählte Ansatz ist daher nicht die erschöpfende Untersuchung der Sicherheit von OpenStack in jeder möglichen Konfiguration. Vielmehr besteht das Ziel darin, die Interaktion einzelner Netzwerkdienste auf konzeptionelle Schwächen hin zu prüfen. Zentraler Untersuchungsgegenstand ist daher die Netzwerkkommunikation der Anwendungen. Hierbei wird insbesondere überprüft, ob eine umfassende Verschlüsselung und Signierung durchführbar ist. Ferner wird die Authentifizierung sowohl der Benutzer gegenüber den Diensten als auch der Dienste untereinander analysiert. Darüber hinaus wird untersucht, auf welche Art OpenStack die Gäste typischerweise an das Gast-Netzwerk anbindet.

10.2 Architektur

OpenStack ist eine Sammlung mehrerer quelloffener Managementwerkzeuge, mit denen eine große Anzahl von Hosts und Gästen verwaltet werden kann. Hierbei wird weitgehend von der realen Host- und Netzwerkhardware abstrahiert. Mit OpenStack können komplexe Cloud-Umgebungen zur Bereitstellung virtueller Infrastrukturen (Infrastucture as a Service, IaaS) erstellt und verwaltet werden. Anders als mit virsh und virt-manager wird also nicht ein einzelner Host, sondern eine Menge von Hosts und deren Gästen verwaltet. Die Verteilung der Gäste auf Hosts ist hierbei sehr flexibel, weitgehend automatisiert und transparent für den Nutzer der Cloud. Ferner ist OpenStack mandantenfähig. Das heißt, Nutzer können unabhängig voneinander Gäste verwalten, ohne auf Gäste anderer Nutzer Zugriff zu haben.

Um eine solch weitgehende Abstraktion von realer Hardware zu realisieren, ist eine komplexe Managementumgebung erforderlich. OpenStack ergänzt daher den Kern der Virtualisierungsumgebung, der aus KVM, QEMU und libvirt besteht, um eine Reihe von Managementfunktionen. Eine solche Managementumgebung besteht dabei aus mehreren Anwendungen, die im Sinne einer serviceorientierten Architektur verschiedene Dienste zur Verfügung stellen. Diese Dienste werden in der Regel verteilt über mehrere Server bereitgestellt.

Eine dieser Funktionen ist die Verwaltung komplexer Overlay-Netzwerke, die von der tatsächlich vorhandenen Netzwerktopologie abstrahieren. Die Fähigkeit, mithilfe von Software Netzwerke (Software Defined Network, SDN) zu erstellen, ist eine wesentliche Voraussetzung dafür, Gäste unabhängig vom ausführenden Host miteinander vernetzen zu können.

Darüber hinaus bietet OpenStack ein Benutzer- und Rollenmanagement, um die Einhaltung der unterschiedlichen Privilegien der Cloud-Nutzer sicherstellen zu können.

Auch wenn die Dienste, die zum OpenStack-Projekt gehören, die Grundlage einer auf OpenStack basierenden Cloud sind, so lassen sich diese doch sehr unterschiedlich konfigurieren und erlauben somit eine flexible Anpassung der Cloud an die jeweiligen Anforderungen. Auch ist nicht jeder Dienst des OpenStack-Projekts zwingend zur Realisierung einer Cloud erforderlich. Ferner sind die Schnittstellen der zur OpenStack gehörenden Dienste offen spezifiziert. Dies ermöglicht es, Software von Drittanbietern oder Eigenentwicklungen in die Cloud-Managementumgebung einzubinden. Die Architektur und Funktionsweise einer auf OpenStack basierenden Cloud richten sich somit in nicht unerheblichem Maße nach den jeweiligen Anforderungen des Betreibers. Verschiedene auf OpenStack basierende Clouds sind daher nicht direkt vergleichbar.

Tabelle 44 führt die zu OpenStack gehörenden Komponenten inklusive einer kurzen Funktionsbeschreibung auf. Der Zweck der Dienste wird in Abb. 45 skizziert. Komponenten, die als veraltet gekennzeichnet oder noch nicht für den produktiven Betrieb freigegeben sind, werden nicht betrachtet.

Name	Funktion
Horizon	Webbasierte Administrationsoberfläche
Nova	Verwaltung der virtualisierenden Hosts („Compute Nodes“)
Neutron	Netzwerkverwaltung
Glance	Speicher von Festplattenabbildern zur Erzeugung virtueller Maschinen
Cinder	Speicher für Blockgeräte, insbesondere Speicher von virtuellen Maschinen
Swift	Speicher für unstrukturierte Daten
Keystone	Authentifikations- und Autorisierungsdienste
Ceilometer	Überwachung und Messung der Cloud-Dienste
Heat	Orchestrierung der Cloud

Tabelle 44: Liste der OpenStack-Komponenten

Die Komponenten bestehen jeweils aus einem oder mehreren Dienstprogrammen, die die entsprechende Funktion realisieren, sowie aus kommandozeilenbasierten Werkzeugen zur Administration. Die Dienstprogramme unterschiedlicher Komponenten können auf unterschiedlichen Hosts eingerichtet werden. Entsprechendes gilt auch für die meisten Komponenten, die mehrere Dienstprogramme zur Verfügung stellen. So ist es beispielsweise möglich, den Dienst nova, der unter anderem aus den Dienstprogrammen nova-api, nova-compute und nova-scheduler besteht, auf unterschiedlichen Servern ausführen zu lassen. Die Möglichkeit der verteilten Ausführung kommt insbesondere der Lastverteilung zugute und bildet die Grundlage für die Skalierbarkeit von OpenStack. Darüber hinaus ist es möglich, einzelne Dienstanwendungen redundant zu betreiben, um die Ausfallsicherheit zu erhöhen.

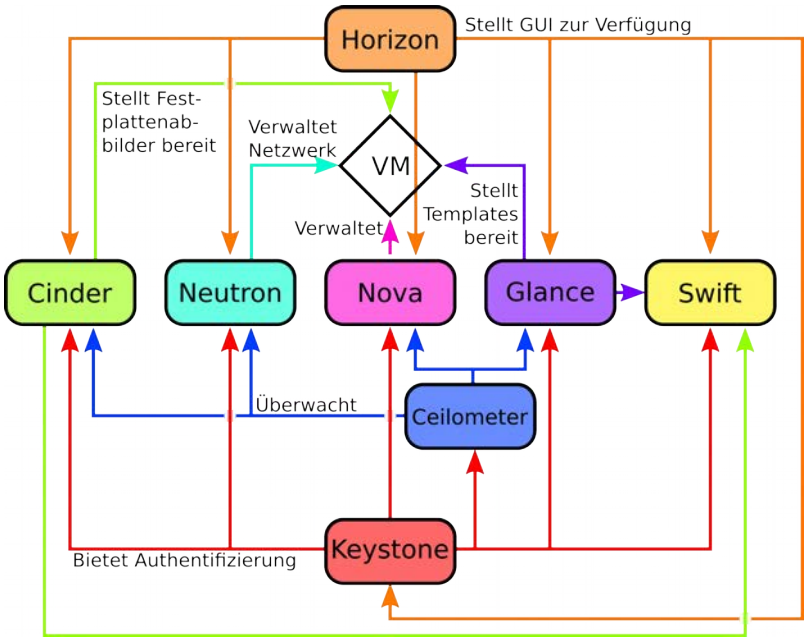

Abbildung 45: Architektur von OpenStack

Damit geht jedoch eine unter Umständen nicht unwesentliche Erhöhung der Komplexität des Gesamtsystems einher. Ebenso erhöht sich der Aufwand zu dessen Absicherung. Insbesondere der umfassende Informationsaustausch über ein Netzwerk, der für serviceorientierte Architekturen typisch ist, bietet potenzielle Angriffsflächen. Neben der offensichtlichen Gefahr, dass ein Angreifer die Verfügbarkeit der Dienste behindert, indem er die Netzwerkkommunikation stört, bestehen auch erhebliche Gefahren bezüglich der Schutzziele Vertraulichkeit und Integrität. So werden im Rahmen der Verwaltung eine Reihe sensibler Daten (z. B. Kennwörter) ausgetauscht. Auch kann die gezielte Einbringung fehlerhafter Befehle durch einen Angreifer fatale Folgen für die Ausführung des Gastes haben. Die Funktion der einzelnen Komponenten wird in den folgenden Abschnitten erläutert.

10.3 Komponenten

10.3.1 Benutzerschnittstelle

Die Komponente Horizon stellt eine webbasierte Administrationsoberfläche zur Verfügung. Sie ermöglicht es den Nutzern der Cloud, Gäste und Gast-Netzwerke zu verwalten. Durch die Mandantenfähigkeit von Horizon sind Nutzer hierbei auf die eigenen Gäste und Netzwerke beschränkt. Ferner kann der Betreiber einer Cloud Horizon zur mandantenübergreifenden Administration nutzen.

10.3.2 Virtualisierung

Die Komponente Nova bindet die virtualisierenden Hosts (Compute Nodes) in die OpenStack-Infrastruktur ein. Die wesentlichen Anwendungen sind nova-api, nova-scheduler und nova-compute. Die Anwendung nova-api bietet eine netzwerkfähige Softwareschnittstelle, um Befehle entgegenzunehmen und Informationen über Host und Gäste bereitzustellen. Diese Schnittstelle wird insbesondere von der Komponente Horizon genutzt, um den Zustand von Gästen anzuzeigen und zu ändern. So kann Horizon z. B. die Gäste starten oder stoppen oder sie auf einen anderen Host migrieren. Die Software nova-scheduler dient dazu, Gäste gemäß ihrem Ressourcenbedarf möglichst optimal auf die vorhandenen Hosts zu

verteilen. Die eigentliche Verwaltung wird durch nova-compute durchgeführt. Diese Anwendung ist auf allen Hosts installiert und steuert mithilfe von libvirt die Gäste des jeweiligen Hosts.

10.3.3 Netzwerkverwaltung

Die Komponente Neutron dient zur Erzeugung und Administration eines Overlay-Netzwerks auf Basis von Open vSwitch. Sie dient dazu, die Kommunikation sowohl zwischen Gästen untereinander als auch zwischen Gästen und der Außenwelt zu ermöglichen. Sie ist somit auch dafür zuständig, eine unerwünschte Kommunikation zwischen Gästen unterschiedlicher Mandanten zu unterbinden.

Die Verwaltung entsprechender Regeln ist – insbesondere in komplexen Virtualisierungsumgebungen – nicht trivial. So kann es beispielsweise erforderlich sein, dass zwei Gäste desselben Kunden, die in unterschiedlichen Rechenzentren betrieben werden, miteinander kommunizieren können. Hingegen soll eine virtuelle Maschine eines dritten Kunden nicht mit diesen Gästen kommunizieren oder deren Datenaustausch mitlesen können, auch wenn die virtuellen Maschinen auf demselben Host betrieben werden. Ebenso verlangt die Migration laufender Gäste von einem Host zu einem anderen Host unter Umständen eine entsprechende Anpassung des Overlay-Netzwerks.

Die Netzwerkanbindung mithilfe von Neutron wird daher in Abschnitt 10.7 untersucht.

10.3.4 Speicheranbindung

Die Speicheranbindung kann durch die drei Dienste Glance, Cinder und Swift umgesetzt werden. Glance dient hierbei zur Bereitstellung von vorbereiteten Festplattenabbildern bei der Erstellung neuer virtueller Maschinen. Cinder wird genutzt, um die individuellen Festplattenabbilder der Gäste zu speichern und diesen zur Verfügung zu stellen.

Zur Speicherung von Daten können sowohl Glance als auch Cinder die Komponente Swift nutzen. Swift übernimmt hierbei die Verwaltung der Datenträger sowie die Replikation der Daten. Swift kann hierzu sowohl Ceph (s. Abschnitt 9.3) als auch GlusterFS (s. Abschnitt 9.4) nutzen.

10.3.5 Authentifizierung

Um die Kooperation der verschiedenen Komponenten abzusichern, stellt die Komponente Keystone einen zentralen Dienst zur Authentifizierung bereit. Er erlaubt es anderen Diensten, Zugriffe der weiteren Komponenten der OpenStack-Infrastruktur untereinander auf ihre Legitimität hin zu überprüfen. Keystone verifiziert insbesondere Kennworteingaben der Nutzer. Die Grundlage für diese Überprüfung ist ein auf Token basierendes Authentifizierungssystem, das in Abschnitt 10.6 untersucht wird.

10.3.6 Überwachung

Mit Ceilometer werden die Komponenten der Virtualisierungsumgebung überwacht. Ceilometer erlaubt die kontinuierliche Protokollierung von Leistungsmerkmalen. Um dies zu ermöglichen, sind umfassende Privilegien zum Zugriff auf entsprechende Informationen nötig. Auch wenn der Einsatz von Ceilometer keine zwingende Anforderung von OpenStack ist, so stellt es doch – vor allem in komplexen Umgebungen – wichtige Dienste zur Verfügung.

10.3.7 Datenbank

Zur persistenten Speicherung von Informationen der einzelnen Dienste wird eine relationale Datenbank, wie z. B. MySQL, benötigt. Hierbei kann ein einzelner, zentraler Datenbankserver die Informationen aller

Dienste verwalten. Alternativ können für die jeweiligen Dienste unabhängig voneinander Datenbanken betrieben werden. Auch wenn die Datenbanken nur Hilfsdienste für die Kernkomponenten der Management-Umgebung sind, so ist der Schutz der gespeicherten Daten doch von zentraler Bedeutung für die Sicherheit des Gesamtsystems. Neben einem direkten Angriff auf die Dienste besteht die Möglichkeit, diese indirekt durch Kompromittierung der jeweiligen Datenbank zu manipulieren. Es ist damit zu rechnen, dass ein solcher indirekter Angriff regelmäßig ähnlich weitreichende Folgen für die jeweilige Komponente hat wie die direkte Kompromittierung des Dienstes. Erschwerend kommt hinzu, dass es gerade in kleineren Cloud-Umgebungen organisatorisch sinnvoll sein kann, die Datenspeicherung aller Komponenten mithilfe eines einzigen Datenbanksystems zu realisieren. Eine Kompromittierung eines solchen zentralen Datenbanksystems wäre fatal für die Sicherheit der Virtualisierungsumgebung.

10.3.8 Message-Broker

Neben einer Datenbank ist ein Message-Broker (typischerweise RabbitMQ) erforderlich, um den zuverlässigen Nachrichtenaustausch zwischen den Komponenten zu gewährleisten. Die Mehrheit der in einer OpenStack-Umgebung vorhandenen Systeme kommuniziert über diesen Message-Broker mittelbar untereinander (s. Abschnitt 10.4). Ein Ausfall hat weitreichende Konsequenzen für die Verfügbarkeit der entsprechenden Komponenten. Darüber hinaus gefährdet eine Kompromittierung potenziell die Vertraulichkeit und die Integrität der am Nachrichtenaustausch beteiligten Komponenten (s. Abschnitt 10.5).

10.4 Kommunikationsbeziehungen

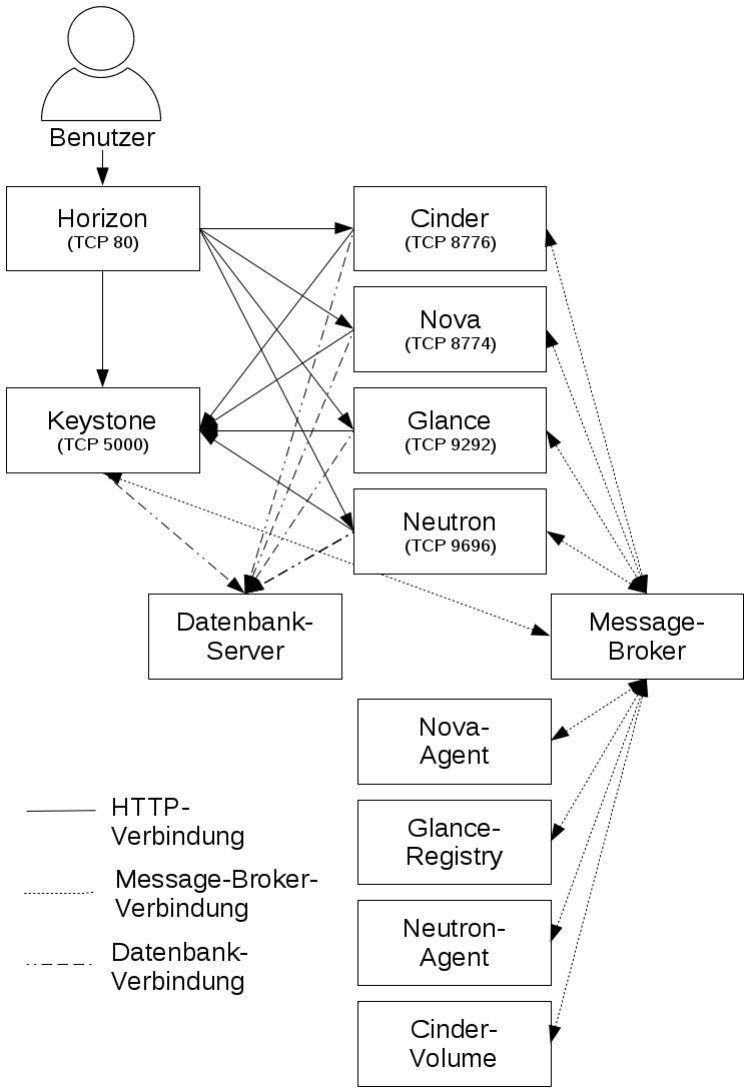

Abbildung 46: Kommunikationsverbindungen der OpenStack-Umgebung

Die meisten der zu OpenStack gehörenden Dienste werden über HTTP gesteuert. Anweisungen, die der Benutzer mit der HTTP-basierten Benutzerschnittstelle (Horizon) gibt, werden von dieser an die zuständigen Komponenten der Cloud-Umgebung weitergeleitet. Die Authentifizierung der Kommunikationsteilnehmer (d. h. sowohl des Benutzers als auch der Netzwerkdienste) erfolgt hierbei über Token, die in die HTTP-Anfragen eingebettet sind (s. Abschnitt 10.6). Die Weiterleitung der Anweisung an den letztendlich ausführenden Netzwerkdienst geschieht über einen Message-Broker (s. Abschnitt 10.3.8). Soll beispielsweise eine virtuelle Maschine neu gestartet werden, wird nach der Anfrage an die HTTP-Schnittstelle eine Nachricht mithilfe des Message-Brokers versendet. Ein Agent auf dem Host, der die zu startende virtuelle Maschine bereitstellt, empfängt diese Nachricht und setzt die Aktion um.

Darüber hinaus wird der Message-Broker verwendet, um Informationen asynchron in der OpenStack-Umgebung zu verteilen.

Ein Datenbank-Server wird von den verschiedenen Diensten genutzt, um Daten persistent zu speichern (s. Abschnitt 10.3.7). Abbildung 46 zeigt eine schematische Übersicht der wesentlichen Kommunikationsverbindungen in einer OpenStack-Umgebung. Hierbei ist zu beachten, dass die Datenspeicherung nicht zwingend in einer einzelnen zentralen Datenbank stattfinden muss.

10.5 Schutz der Kommunikation

Wichtig für die Sicherheit der Management-Umgebung ist, dass die Vertraulichkeit und Integrität der Netzwerkkommunikation der Dienste sichergestellt wird.

Die Kommunikation mittels HTTP kann hierzu vollständig durch Verwendung von TLS gesichert werden. Hierzu ist es notwendig, Zertifikate für die jeweiligen Netzwerkdienste auszustellen und diese auf den jeweiligen Systemen zu hinterlegen. Je nach eingesetzter Datenbanksoftware kann dies analog auch für die Interaktion mit der Datenbank umgesetzt werden. Dies trifft insbesondere für die Verwendung von MySQL und PostgreSQL zu. Ebenso ist dies für den von OpenStack bevorzugten Message-Broker RabbitMQ möglich.

Prinzipiell ermöglicht es OpenStack somit, die Vertraulichkeit und Integrität der Netzwerkkommunikation durch Verschlüsselung und Signierung sicherzustellen.

10.6 Authentifizierung

10.6.1 Allgemein

Die Authentifizierung wird in einer OpenStack-Umgebung mit dem Dienst Keystone realisiert. Das zentrale Element in der Authentifizierung sind Token. Nachdem ein Subjekt (d. h. ein Benutzer oder ein Dienst) eine valide Kombination aus Benutzername und Passwort an Keystone übermittelt hat, erstellt Keystone ein Token und übersendet es dem Subjekt. Dieses kann sich fortan mithilfe des Tokens gegenüber Diensten der OpenStack-Umgebung ausweisen.

Keystone kann auf vielfältige Weise konfiguriert werden, um die Benutzereingaben zu validieren. Somit können auf verhältnismäßig einfachem Wege verschiedene Systeme zur Verwaltung der Anmeldedaten in die Cloud-Umgebung eingebunden werden. Die Authentifizierung gegenüber anderen Diensten erfolgt jedoch unabhängig hiervon über die erläuterten Tokens. Eine Änderung des letztgenannten Authentifizierungsvorgangs würde daher eine Modifikation sämtlicher Dienste erfordern und ist daher weitaus schwieriger umzusetzen. Konkret existieren drei verschiedene Authentifizierungsverfahren, die auf unterschiedlich geformten Tokens basieren.

10.6.2 UUID-Token

Das UUID-Verfahren ist das simpelste von Keystone angebotene Authentifizierungsverfahren. Nachdem ein Subjekt sich erfolgreich gegenüber Keystone authentifiziert hat, generiert Keystone eine zufällige UUID gemäß [RFC4122]. Diese bildet das Token. Zusätzlich speichert Keystone die Zugehörigkeit zwischen Token und Subjekt.

Möchte der Besitzer des Tokens sich gegenüber einem Dienst authentifizieren, so überprüft der Dienst die Gültigkeit des Tokens und dessen Zugehörigkeit zum jeweiligen Subjekt, indem er eine Anfrage an Keystone stellt.

Bei dem in [RFC4122] beschriebenen Verfahren zu Generierung von zufälligen UUIDs handelt es sich um ein Verfahren, mit dem Entitäten eindeutig identifiziert werden können. UUIDs wurden nicht entwickelt, um eine Authentifizierung zu realisieren. [RFC4122] weist auf diesen Sachverhalt in Abschnitt 6 explizit hin. Die Sicherheit des Authentifizierungsverfahrens basiert in ganz erheblichem Maße darauf, dass die UUID nicht vorhersagbar ist.

Zur Implementierung nutzt Keystone das Modul uuid der Programmiersprache Python, um die UUIDs zu erzeugen. Die folgende Auflistung zeigt den wesentlichen Teil der Implementierung des Moduls uuid:

```
try:
    import os
    return UUID(bytes=os.urandom(16), version=4)
except:
    import random
    bytes = bytes_(random.randrange(256) for i in range(16))
    return UUID(bytes=bytes, version=4)
```

Sofern das Modul os importiert und die Funktion os.urandom ohne Fehler ausgeführt werden kann, darf davon ausgegangen werden, dass eine kryptografisch sichere UUID generiert wird. Im Fehlerfall wird jedoch auf die kryptografisch unsichere Funktion random.randrange zurückgegriffen. Auch wenn somit in aller Regel eine nicht vorhersagbare, zufällige UUID generiert wird, so kann Gegenteiliges nicht gänzlich ausgeschlossen werden.

Ferner ist anzumerken, dass die erstellten Tokens allesamt in der von Keystone verwendeten Datenbank gespeichert werden. Gelingt es einem Angreifer, Einsicht in diese Datenbank zu erhalten, so kann er hierdurch gültige Token zur Authentifizierung erlangen.

10.6.3 Fernet-Token

Bei dem Fernet-Verfahren wird ein Token erzeugt, das den Benutzernamen und eine Reihe weiterer Informationen enthält, wie z. B. einen Gültigkeitszeitraum. Diese Token werden mit einem symmetrischen Verschlüsselungsverfahren verschlüsselt und mit einem HMAC vor Veränderungen geschützt. Der hierzu verwendete symmetrische Schlüssel ist üblicherweise nur Keystone bekannt.

Möchte sich der Besitzer des Tokens gegenüber einem Dienst authentifizieren, so sendet der Dienst das Token des Anfragenden an Keystone. Keystone entschlüsselt das Token und überprüft anhand der HMAC dessen Gültigkeit. Das Ergebnis dieser Überprüfung wird dann dem Dienst mitgeteilt.

10.6.4 PKI-Token

Bei dem PKI-Verfahren wird ein Token erzeugt, das den Benutzernamen und eine Reihe weiterer Informationen enthält, wie z. B. einen Gültigkeitszeitraum. Keystone signiert dieses Token durch asymmetrische Verschlüsselung.

Möchte sich der Besitzer des Tokens gegenüber einem Dienst authentifizieren, so überprüft der Dienst die Gültigkeit des Tokens anhand der Signatur.

Der Vorteil dieses Verfahrens ist, dass Keystone nicht benötigt wird, um sich gegenüber einem konkreten Dienst zu authentifizieren. Der damit verbundene Arbeitsaufwand entfällt somit.

10.6.5 Bewertung

Zwar kann mit einem Token das Subjekt, das den Dienst in Anspruch nimmt, eindeutig identifiziert werden. Eine Beschränkung der Gültigkeit der Token auf einen einzelnen Dienst erfolgt jedoch nicht. Gelingt es somit einem Angreifer, ein Token einzusehen, so ist es ihm möglich, sich gegenüber allen Systemen als der Besitzer des Tokens auszugeben. Insbesondere dann, wenn ein System kompromittiert wurde, an das Token gesendet werden, muss regelmäßig davon ausgegangen werden, dass der Angreifer das Token einsehen kann. Es ist daher realistisch anzunehmen, dass sich die Kompromittierung über das einzelne System hinaus ausweiten wird. Problematisch ist dies insbesondere beim Einsatz einer großer Anzahl von

Systemen, aufgrund der potenziell erhöhten Wahrscheinlichkeit einer Kompromittierung einzelner Systeme.

Zum Schutz der Token ist es sinnvoll, die Netzwerkkommunikation der OpenStack-Umgebung zu sichern (s. Abschnitt 10.5).

Darüber hinaus ergeben sich weitere Risiken durch den indirekten Nachrichtenversand über den Message-Broker. Gelingt es einem Angreifer, sich am Message-Broker anzumelden, so kann er sich unter Umständen für den Empfang von Nachrichten eintragen. Dies würde dazu führen, das der Message-Broker entsprechende Nachrichten an den Angreifer zustellt. Sofern der Sender einer Nachricht authentifiziert werden soll, enthält die Nachricht eines der genannten Token. Infolgedessen könnte der Angreifer das Token einsehen und weiterverwenden.

10.7 Netzwerkanbindung

10.7.1 Konfiguration

Um die Netzwerkanbindung der Gäste an das Netzwerk zu untersuchen, wird die im Folgenden beschriebene Testumgebung genutzt. Die Installation erfolgt hierbei mit dem Werkzeug Packstack[116].

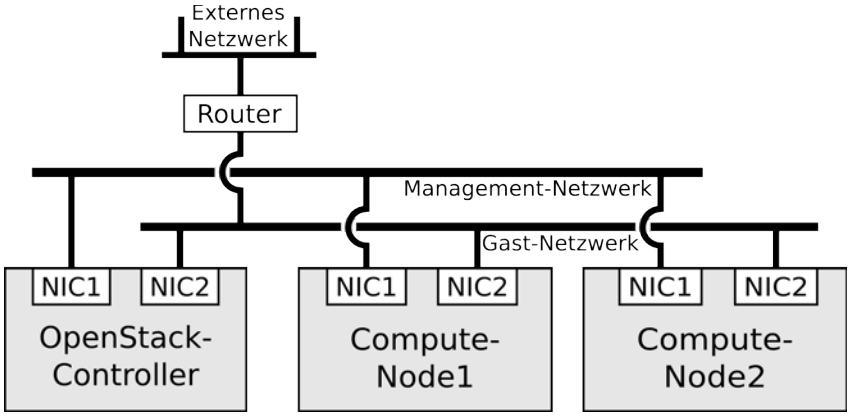

Abbildung 47: Infrastruktur der OpenStack-Testumgebung

Die erzeugte Testumgebung ist schematisch in Abb. 47 dargestellt. Sie besteht aus einem OpenStack-Controller, der die verschiedenen Dienste ausführt, und zwei Compute-Nodes, auf denen die Gäste mit KVM, QEMU und libvirt virtualisiert werden.

Alle drei Hosts sind über ein Management-Netzwerk verbunden. Dieses Netzwerk wird zur Kommunikation der Dienstanwendungen mit den jeweiligen Agenten auf den Compute-Nodes verwendet. Außerdem wird die Webanwendung Horizon über diese Schnittstelle einem Administrator zugänglich gemacht.

116https://www.rdoproject.org

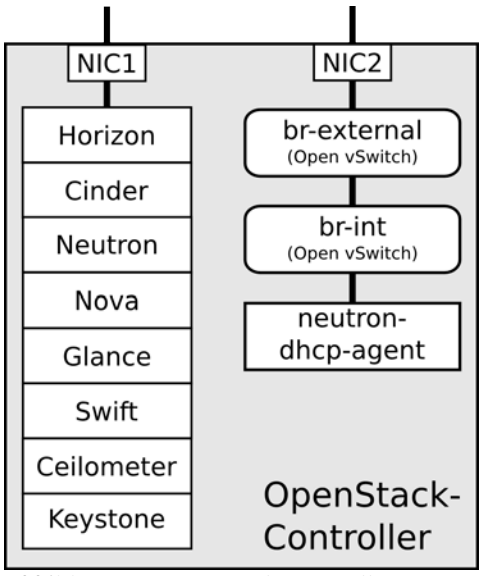

Abbildung 49: OpenStack-Controller

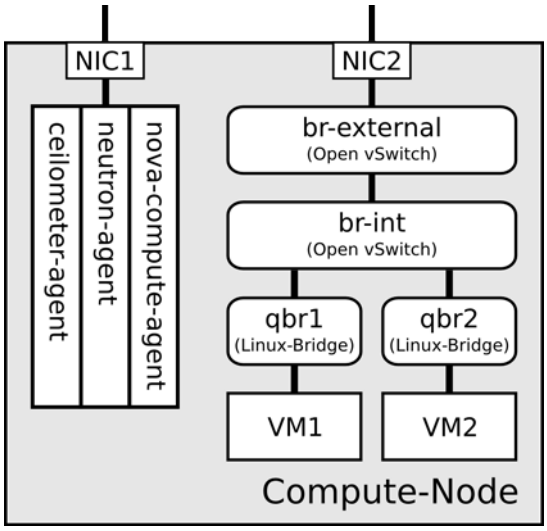

Abbildung 48: OpenStack-Compute-Nodes

Eine weitere Schnittstelle wird für den Netzwerkverkehr der Gäste genutzt. Über diese Schnittstelle ist die Kommunikation zwischen den Gästen über die Grenzen eines Compute-Nodes hinaus möglich. Eine Verbindung zu einem externen Netzwerk wird über einen Router realisiert, der mit dem Gast-Netzwerk verbunden ist.

Der OpenStack-Controller ist in Abb. 49 dargestellt. Die verschiedenen Dienstanwendungen werden zentral auf diesem Host ausgeführt. Er verwaltet die Compute-Nodes, führt selbst aber keine virtuellen Maschinen aus. Der Controller ist ebenfalls mit dem Gast-Netzwerk verbunden, um einen DHCP-Service für das Gast-Netzwerk bereitzustellen.

Der Aufbau der Compute-Nodes ist in Abb. 48 dargestellt. Über das Management-Netzwerk stellen die OpenStack-Agenten eine Verbindung mit dem Controller her. Die zweite Netzwerkkarte verbindet die Gäste mit einem physischen Netzwerk. OpenStack erzeugt für die Anbindung der virtuellen Maschinen eine Reihe von Netzwerkgeräten. Die reale Netzwerkkarte wird mit einem Open vSwitch verbunden (br-external). Die Zuordnung zwischen verschiedenen virtuellen Netzwerken und dem physikalischen Netzwerk erfolgt über diese Bridge. Die externe Netzwerkbrücke wird ebenfalls mittels Open vSwitch verbunden (br-int). Dies dient der Integration aller vorhandenen virtuellen Netzwerkschnittstellen auf dem Compute-Node. Die Gäste werden über ein Tap-Device mit der zentralen Bridge (br-int) verbunden. Mithilfe von Neutron wird ein externes Netzwerk vom Typ flat konfiguriert.

Außerdem werden zwei Benutzer eingerichtet und unterschiedlichen Projekten zugeordnet. Die Benutzer sind in der Lage, in ihrem Projekt virtuelle Maschinen zu starten und mit dem externen Netzwerk zu verbinden. Für diesen Test werden für jeden Benutzer zwei virtuelle Maschinen gestartet.

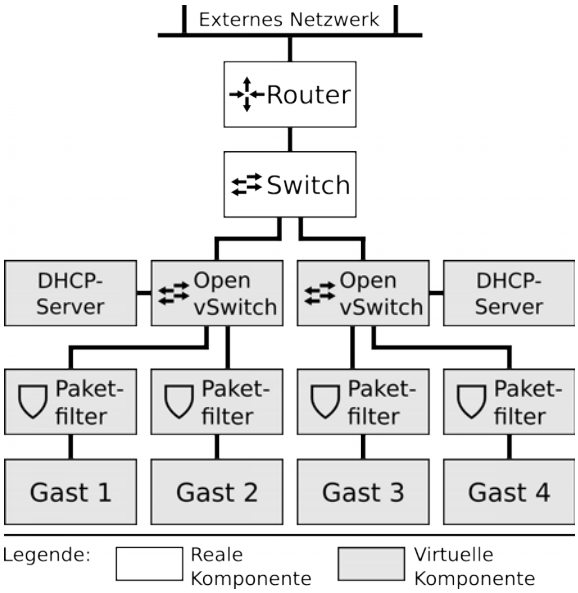

Abbildung 50: Logische Netzwerktopologie der Testumgebung OpenStack

Darüber hinaus wird die von Packstack erstellte Konfiguration angepasst, da Packstack einige Neuerungen von OpenStack nicht berücksichtigt. Dies betrifft insbesondere die Filterung des Netzwerkverkehrs, die unwirksam wird. In aktuellen Versionen von OpenStack (aber nicht bei Packstack) basiert die Filterung nicht mehr auf iptables, sondern auf Open vSwitch. Infolgedessen hat sich die Anbindung der Gast-Netzwerkschnittstellen an die zentrale Open vSwitch-Netzwerkbrücke (br-int, s. Abbildung 48) geändert. In dem iptables-basierten Konzept war jedem Gast eine separate Linux-Bridge vorgeschaltet. Auf dieser Linux-Bridge wurden die spezifischen Filter-Regeln für die Gäste angewandt. Mit dem Open-vSwitch-basierten Konzept ist die zusätzliche Linux-Bridge entfallen und die Gäste werden direkt mit der zentralen Open-vSwitch-Netzwerkbrücke verbunden.

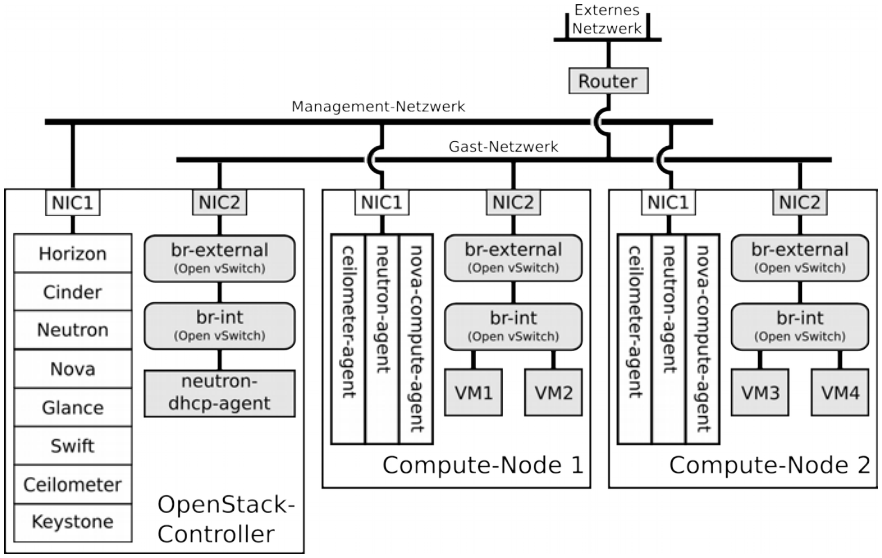

Abbildung 51: Aufbau der Testumgebung OpenStack

Die Installation mittels Packstack berücksichtigt die geänderte Situation jedoch nicht. Es wird weiterhin das iptables-basierte Firewall-Konzept verwendet. Aus diesem Grund wird die Konfiguration des Testsystems

angepasst, indem der Parameter firewall_driver in der Datei openvswitch_agent.ini manuell auf den Wert openvswitch geändert wird. Dadurch wird der Open-vSwitch-basierte Ansatz gewählt. Die hierdurch entstehende logische Netzwerktopologie wird in Abb. 50 dargestellt.

Außerdem ist für das Open-vSwitch-basierte Konzept die Version Open vSwitch 2.5.0 erforderlich. Mit CentOS 7 wird die Version 2.4.0 ausgeliefert.

Ferner wird auf die Anbindung eines netzwerkbasierten Speicher-Backends verzichtet. In Abb. 51 ist die Testumgebung skizziert.

Außerdem wird sichergestellt, dass der Host keine IP-Adresse an die Netzwerkschnittstelle des Gast-Netzwerks bindet (s. Abschnitt 8.2).

10.7.2 Ergebnisse

Die Durchführung des Tests der Netzwerkanbindung geschieht entsprechend dem in Abschnitt 8.3 spezifizierten Verfahren. Die Ergebnisse werden in Tabelle 45 dargestellt.

Lediglich Testfall 11 wird nicht bestanden. Einen Schutz vor NDP- und RA-Spoofing bietet OpenStack in Version 8.0.0 nicht. Entsprechende Filterregeln können aber durch allgemeine Funktionen des Host-Betriebssystems realisiert werden.

#	Bedrohung	Ergebnis	Bemerkung
1	Vorhandene Konnektivität mittels IPv4	bestanden	
2	Vorhandene Konnektivität mittels IPv6	bestanden	
3	Fehlende Trennung vom Management-Netzwerk	bestanden	
4	Fehlende Trennung vom Storage-Netzwerk		Es ist kein netzwerkbasiertes Speicher-Backend angebunden. Der Test ist daher nicht anwendbar.
5	Erreichbarkeit von Diensten des Hosts durch den Gast	bestanden	
6	Kommunikation mittels beliebiger Protokolle über Ethernet	bestanden	
7	MAC-Address-Spoofing	bestanden	
8	IP-Address-Spoofing	bestanden	
9	ARP-Spoofing	bestanden	
10	ARP-Flooding	bestanden	
11	NDP-Spoofing/ RA-Spoofing	nicht bestanden	
12	DHCP-Spoofing	bestanden	

Tabelle 45: Testergebnisse von OpenStack

10.8 Zusammenfassung

Zusammenfassend ist festzuhalten, dass die Netzwerkkommunikation der zu OpenStack gehörenden Dienste vollständig mittels TLS abgesichert werden kann. Beim Einsatz der üblichen, nicht zu OpenStack gehörenden Dienste (wie z. B. der Datenbank MySQL oder des Message-Brokers RabbitMQ) ist dies ebenfalls möglich. Ausnahmen können sich jedoch ergeben, falls weitere Software, die eine entsprechende Absicherung nicht unterstützt, in die OpenStack-Umgebung eingebunden wird.

Darüber hinaus existiert mit Keystone ein zentraler Authentifikationsdienst, der – umfangreich konfigurierbar – Anmeldedaten überprüft. Die von Keystone nach erfolgreicher Authentifizierung ausgestellten Token werden genutzt, um sich gegenüber den verschiedenen Diensten der OpenStack-Umgebung auszuweisen. Hierzu existieren drei verschiedene Token-Formen mit jeweils unterschiedlichen Validierungsverfahren. Ungünstig ist, dass jedes dieser Verfahren es ermöglicht, sich mit nur einem einzigen Token gegenüber allen Diensten zu authentifizieren. Wird dieses Token von einem Angreifer eingesehen, so kann dieser die Identität des Token-Besitzers gegenüber allen Diensten der OpenStack-Umgebung annehmen. Dies ist umso problematischer, da sich ein Benutzer im Rahmen der üblichen Interaktion mit OpenStack automatisch an einer Reihe von Diensten anmeldet. Ist einer dieser Dienste kompromittiert, so ist davon auszugehen, dass der Angreifer die an den Dienst gesendeten Tokens mitlesen kann. Durch die damit verbundene Übernahme der Benutzeridentität ist regelmäßig eine signifikante Rechteausweitung des Angreifers zu erwarten.

Die darüber hinausgehende Analyse der Netzwerkanbindung des Gastes an das Gast-Netzwerk zeigt keine wesentlichen Auffälligkeiten. Lediglich der Schutz vor IP-Spoofing-Angriffen in IPv6-Gast-Netzwerken wird durch OpenStack nicht nativ unterstützt. Eine Abwehr lässt sich jedoch durch allgemeine Funktionen des Host-Betriebssystems realisieren.

11 Handlungsempfehlungen

11.1 Zielsetzung

Im Folgenden werden Handlungsempfehlungen aufgeführt, die die Sicherheit einer Virtualisierungsumgebung erhöhen, die auf QEMU, KVM und libvirt basiert. Diese Empfehlungen unterstützen die Administratoren solcher Umgebungen einerseits dabei, die wahrscheinlichsten und sicherheitstechnisch folgenschwersten Fehlkonfigurationen zu vermeiden. Andererseits werden die Administratoren in die Lage versetzt, grundlegende Härtungsmaßnahmen durchzuführen. Zu diesem Zweck werden sowohl allgemeine Maßnahmen als auch Maßnahmen zur sicheren Netzwerkanbindung aufgeführt.

Die aufgeführten Handlungsempfehlungen stellen jedoch keine vollständige Anleitung zur sicheren Konfiguration einer solchen Umgebung dar. Insbesondere bleiben virtualisierungsunabhängige Härtungsmaßnahmen unberücksichtigt, wie sie z. B. die IT-Grundschutz-Kataloge ([ITGS]) aufführen. Auch ersetzen die Handlungsempfehlungen keine individuelle Sicherheitsanalyse einer konkreten Virtualisierungsumgebung.

Vor der Umsetzung der aufgeführten Maßnahmen sind diese stets daraufhin zu überprüfen, ob sie auf das jeweilige System angewandt werden können. Insbesondere dürfen Konfigurationsänderungen erst dann durchgeführt werden, nachdem geprüft wurde, dass durch sie keine Inkompatibilitäten mit der sonstigen Konfiguration des Systems entstehen.

11.2 Allgemeine Empfehlungen

11.2.1 Aktivierung von SELinux

libvirt nutzt SELinux, um die Berechtigungen des QEMU-Prozesses stark einzuschränken. Die Wahrscheinlichkeit eines erfolgreichen Ausbruchs aus dem Gast wird auf diesem Wege signifikant reduziert (s. Abschnitt 7.5). SELinux sollte daher aktiviert sein, sofern das Betriebssystem dies unterstützt.

Dies kann mit der folgenden Anweisung überprüft werden:

```
# sestatus

SELinux status:                 enabled
SELinuxfs mount:                /sys/fs/selinux
SELinux root directory:         /etc/selinux
Loaded policy name:             targeted
Current mode:                   enforcing
Mode from config file:          enforcing
Policy MLS status:              enabled
Policy deny_unknown status:     allowed
Max kernel policy version:      28
```

Die aufgeführten Ausgaben stammen von einem System mit aktiviertem SELinux. Die hervorgehobenen Zeilen sind wesentlich für die zweckmäßige Funktionstüchtigkeit von SELinux. Weichen diese auf dem zu überprüfenden System ab, so ist SELinux deaktiviert oder nicht vollständig funktionstüchtig. In diesem Fall sollte SELinux, wie in der Dokumentation des jeweiligen Betriebssystems beschrieben, aktiviert bzw. die Konfiguration angepasst werden.

Sollte SELinux vom Betriebssystem nicht unterstützt werden, AppArmor hingegen schon, so empfiehlt sich der Einsatz von AppArmor. Die Nutzung von AppArmor durch libvirt erfolgt dabei über sVirt, ist jedoch nicht Teil dieser Studie. Diesbezügliche Informationen sind der Dokumentation des jeweiligen Betriebssystems zu entnehmen.

11.2.2 Deaktivierung von Kernel Samepage Merging

Ist KSM aktiviert, so besteht die Gefahr von Seitenkanalangriffen (s. Abschnitt 6.8). Ferner ist der durch KSM gewonnene Arbeitsspeicher in der Praxis meist gering. KSM sollte daher nicht genutzt werden.

Die folgende Auflistung zeigt, wie festgestellt werden kann, ob KSM aktiv ist. Die aufgeführten Ausgaben stammen von einem System mit deaktiviertem KSM.

```
# cat /sys/kernel/mm/ksm/run
0

# cat /sys/kernel/mm/ksm/pages_shared
0

# pgrep ksmtuned
```

Sollten die in den ersten beiden Anweisungen ausgegebenen Dateien nicht existieren, so ist KSM nicht aktiv. Anderenfalls ist zu überprüfen, ob die erste Anweisung den Wert 0 oder 2, die zweite Anweisung den Wert 0 und die dritte Anweisung nichts ausgibt. Trifft mindestens einer dieser Sachverhalte nicht zu, so ist KSM aktiv. In diesem Fall sollte KSM so deaktiviert werden, wie es in der Dokumentation des jeweiligen Betriebssystems beschrieben wird.

Für CentOS 7 kann KSM durch folgende Anweisungen dauerhaft deaktiviert werden:

```
# systemctl stop ksmtuned
Stopping ksmtuned:                         [  OK  ]

# systemctl stop ksm
Stopping ksm:                              [  OK  ]

# systemctl disable ksm

# systemctl disable ksmtuned

# echo 2 > /sys/kernel/mm/ksm/run
```

11.3 Empfehlungen bezüglich der Netzwerkanbindung

11.3.1 Die IP-Adresse des Gast-Netzwerks entfernen

Wenn eine Linux-Bridge, macvtap oder Open vSwitch zur Anbindung der Gäste an ein Netzwerk verwendet wird, ist es regelmäßig nicht notwendig, IP-Adressen an die Host-Netzwerkschnittstelle des Gast-Netzwerks zu vergeben. Wird dies dennoch getan, so kann dies unter Umständen dazu führen, dass die Isolation des Gast-Netzwerks von anderen Netzwerken aufgehoben wird. Auch besteht in diesem Fall die Gefahr, dass die Gäste auf Netzwerkdienste des Hosts zugreifen (s. Abschnitt 8.2).

Aus diesem Grund sollte die Bindung von IP-Adressen an entsprechende Netzwerkschnittstellen in aller Regel unterbleiben, sofern eine der aufgeführten Anbindungsarten verwendet wird. Hierzu zählt sowohl die statische als auch die dynamische Vergabe von IPv4- und IPv6-Adressen. Letztere werden regelmäßig im Rahmen der IPv6-Autokonfiguration automatisch bezogen. Somit besteht insbesondere die Gefahr einer

unbeabsichtigten Vergabe. Aus diesem Grund sollte die Unterstützung für IPv6 für entsprechende Netzwerkschnittstellen gänzlich deaktiviert werden.

Bei Verwendung einer Linux-Bridge sollte ferner die Handlungsempfehlung sowohl für die Bridge selbst durchgeführt werden als auch für die Netzwerkschnittstelle, mit der die Bridge an das physische Netzwerk angebunden wird.

Ob die Schnittstelle des Hosts, an die das Gast-Netzwerk angeschlossen ist, eine IP-Konfiguration besitzt, kann mit dem folgenden Befehl überprüft werden. Hierbei wird angenommen, dass der Name der Schnittstelle eth0 ist. Trifft dies nicht zu, so muss der Name in der Anweisung entsprechend angepasst werden.

```
# ip addr show dev eth0|grep inet
```

Gibt diese Anweisung nichts aus, so verfügt die Schnittstelle über keine IP-Konfiguration. Andernfalls sollte die IP-Konfiguration angepasst und sollten die IP-Adressen entfernt werden. Wie dies umgesetzt werden kann, kann der Dokumentation des jeweiligen Betriebssystems entnommen werden.

Für CentOS 7 kann dies wie folgt durchgeführt werden: Um die IPv4-Adresse zu entfernen, wird die Konfigurationsdatei der jeweiligen Schnittstelle angepasst. Ist der Name der Schnittstelle eth0, so ist dies die Datei /etc/sysconfig/network-scripts/ifcfg-eth0. Wesentlich ist, dass die Parameter IPADDR und NETMASK nicht aufgeführt sind. Ferner sollte der Parameter BOOTPROTO auf den Wert none gesetzt sein. Die Konfigurationsdatei sollte daher etwa den folgenden Inhalt haben:

```
DEVICE=eth0
TYPE=Ethernet
BOOTPROTO=none
IPV6INIT=no
ONBOOT=yes
```

Darüber hinaus kann die IPv6-Konfiguration der Netzwerkschnittstelle durch folgende Anweisungen dauerhaft entfernt werden. Sofern die jeweilige Netzwerkschnittstelle nicht den Namen eth0 trägt, ist die erste Anweisung entsprechend anzupassen.

```
# echo "net.ipv6.conf.eth0.disable_ipv6=1" >> \
> /etc/sysctl.d/60-disable-ipv6-for-guests.conf

# sysctl --system
...
```

11.3.2 Die Weiterleitung von IP-Paketen deaktivieren

Wenn eine Linux-Bridge, macvtap oder Open vSwitch zur Anbindung der Gäste an ein Netzwerk verwendet wird, erfolgt die Weiterleitung von Paketen auf Layer 2. Daher ist es in aller Regel nicht notwendig, dass der Host IP-Pakete auf Layer 3 weiterleitet. Wird die Weiterleitung dennoch durch den Host durchgeführt, so kann dies zu einer Aufhebung der Isolation von Gast-Netzwerk und anderen Netzwerken führen. Dies gilt eingeschränkt auch dann, wenn der Host keine IP-Adressen an die Schnittstelle des Gast-Netzwerks vergibt (s. Abschnitt 8.2).

Aus diesem Grund sollte die Weiterleitung sowohl von IPv4- als auch von IPv6-Paketen bei Verwendung der aufgeführten Anbindungsarten unterbunden werden. Dies kann mit der folgenden Anweisung überprüft werden. Die aufgeführten Ausgaben stammen hierbei von einem Host, der eine entsprechende Weiterleitung nicht durchführt.

```
# sysctl net.ipv4.ip_forward
0
```

```
# sysctl net.ipv6.conf.all.forwarding
0
```

Ist die Ausgabe der Anweisung nicht in beiden Fällen 0, so ist die Weiterleitung auf dem Host typischerweise aktiviert und sollte deaktiviert werden. Wie dies umgesetzt werden kann, kann der Dokumentation des jeweiligen Betriebssystems entnommen werden. Zu beachten ist, dass libvirt die Weiterleitung auch unabhängig von der Betriebssystemkonfiguration aktivieren kann (s. Abschnitt 11.3.3).

Für CentOS 7 kann die Weiterleitung durch Ausführung der folgenden Anweisungen unterbunden werden:

```
# cat <<EOF > /etc/sysctl.d/60-disable-ip-forwarding.conf
> net.ipv4.ip_forward=0
> net.ipv6.conf.all.forwarding=0
> EOF

# sysctl --system
...
```

11.3.3 Das libvirt-Standardnetzwerk entfernen

Im Rahmen der Installation von libvirt wird typischerweise automatisch ein virtuelles Netzwerk eingerichtet und gestartet. Dieses Netzwerk trägt den Namen default und ist vom Typ nat (s. 8.4.4.3). Problematisch ist hierbei, dass durch das Starten dieses Netzwerks die Weiterleitung von IP-Paketen aktiviert wird (s. Abschnitt 8.2).

Das virtuelle Netzwerk sollte daher – sofern es nicht verwendet wird – entfernt werden. Dies kann mit den folgenden Anweisungen geschehen:

```
# virsh net-destroy default
Network default destroyed

# virsh net-undefine default
Network default has been undefined
```

Hierbei ist jedoch zu beachten, dass die Weiterleitung von IP-Paketen nicht automatisch deaktiviert wird. Sie kann durch folgende Anweisungen manuell unterbunden werden:

```
# sysctl -w net.ipv4.ip_forward=0

# sysctl -w net.ipv6.conf.all.forwarding=0
```

Sofern die Betriebssystemkonfiguration eine Weiterleitung vorsieht, ist diese Einstellung jedoch nicht persistent. Diesbezüglich ist Abschnitt 11.3.2 zu beachten.

11.3.4 Den Reverse-Path-Filter aktivieren

Es ist sinnvoll, den Reverse-Path-Filter (s. Abschnitt 8.2) als zusätzlichen Schutz zu verwenden, um die Isolation zwischen den Netzwerken zu wahren.

Mit der folgenden Anweisung kann überprüft werden, ob dieser Filter aktiv ist:

```
# sysctl net.ipv4.conf.all.rp_filter
net.ipv4.conf.all.rp_filter = 1
```

Wird der Wert der Konfigurationsvariablen, wie oben dargestellt, mit 1 angegeben, so ist der Filter aktiviert. Anderenfalls sollte er aktiviert werden. Wie dies umgesetzt werden kann, kann der Dokumentation des jeweiligen Betriebssystems entnommen werden.

Für CentOS 7 ist der Reverse-Path-Filter standardmäßig aktiviert.

12 Fazit

Zusammenfassend lässt sich festhalten, dass ein Großteil der virtualisierungsspezifischen Funktionen durch QEMU bereitgestellt wird. Der umfangreiche Quelltext ist durch einen modularen Aufbau sinnvoll strukturiert. Einzelne Bestandteile von QEMU sind konsequent in Form von Komponenten realisiert, die zumeist lediglich lose miteinander gekoppelt sind. Dies trifft insbesondere auf die Emulatoren zu, mit denen dem Gast die virtuelle Hardware zur Verfügung gestellt wird. Das modulare Design ermöglicht es, einzelne Komponenten gegen alternative Komponenten auszutauschen oder gänzlich auf deren Verwendung zu verzichten. Hierdurch kann die Angriffsfläche minimiert werden.

Darüber hinaus trägt die gute Strukturierung des Quelltextes zu einer einfachen Les- und Wartbarkeit von QEMU bei. Die Codequalität ist weitestgehend nicht zu beanstanden. Unzulänglichkeiten sind verhältnismäßig selten und basieren vor allem auf Konvertierungsvorgängen zwischen vorzeichenlosen und vorzeichenbehafteten Datentypen. Diese können grundsätzlich ausnutzbar sein, sind im Allgemeinen aber meist nicht sicherheitsrelevant. Eine partiell durchgeführte Überprüfung dieser Befunde zeigt keine ausnutzbaren Schwachstellen.

Der Gefährdung durch Programmierfehler, die aufgrund des großen Quelltextumfangs nicht ausgeschlossen werden kann, wird durch Härtungsmaßnahmen entgegengewirkt. Diese Maßnahmen werden im Rahmen von CentOS 7 sowohl zur Übersetzungs- als auch zur Laufzeit ergriffen. Insbesondere die Nutzung von Stack Protection, des NX-Bits und von ASLR sowohl für QEMU selbst als auch mit wenigen Ausnahmen für die eingebundenen Softwarebibliotheken bieten einen soliden Schutz. Das Ausnutzen eines Programmfehlers, um aus dem Gast auszubrechen, wird somit für eine Reihe von Angriffsarten verhindert oder zumindest erheblich erschwert.

Die Steuerung von QEMU erfolgt in aller Regel über die netzwerkfähige Verwaltungssoftware libvirt, die ihrerseits weitere Härtungsmaßnahmen implementiert. Dies sind vor allem die Nutzung von cgroups und SELinux, mit denen libvirt die Zugriffsmöglichkeiten von QEMU ganz erheblich einschränkt. Dies erfolgt standardmäßig, sofern das Betriebssystem, wie z. B. CentOS 7, dies unterstützt. libvirt realisiert somit zusätzlich zu den in QEMU integrierten Härtungsmaßnahmen eine weitere Schutzschicht. Hierdurch wird die Wahrscheinlichkeit eines erfolgreichen Ausbruchs aus dem Gast weiter signifikant reduziert.

libvirt ermöglicht ferner eine Absicherung der netzwerkbasierten Verwaltungsschnittstelle mithilfe verschiedener Protokolle. Insbesondere die Nutzung von SSH oder TLS ermöglicht hierbei einen umfassenden und effektiven Schutz der Vertraulichkeit und Integrität der Netzwerkkommunikation. Auch bei einer Kompromittierung des Netzwerks wird ein Einwirken des Angreifers auf die Gäste daher weitestgehend unterbunden. libvirt schützt den Host somit nicht nur effektiv vor Angriffen von innen, d. h. aus dem Gast heraus, sondern auch vor Angriffen von außen, d. h. über das Netzwerk.

Die Steuerung der Hardware, insbesondere der CPU, wird im Kontext der Virtualisierung vom Kernelmodul KVM nach Anweisung von QEMU durchgeführt. KVM verwehrt dem Gast hierbei den Zugriff auf sicherheitskritische Register der CPU oder schränkt diesen sinnvoll ein. Konfigurationsänderungen der CPU, die die Funktionstüchtigkeit des Hosts negativ beeinflussen oder gar einen Ausbruch aus dem Gast ermöglichen würden, werden somit effektiv unterbunden. Ungünstig ist jedoch, dass KVM dem Gast einige Funktionen zur Verfügung stellt, die in der Regel für die Virtualisierung nicht benötigt werden. Hierzu gehört insbesondere die Emulation der Hyper-V-Schnittstelle und der Performance Management Unit. Auch wenn diesbezüglich keine sicherheitskritischen Schwachstellen festgestellt wurden, so resultiert hieraus doch unnötigerweise eine erheblich vergrößerte Angriffsfläche. Wünschenswert wäre, dass sich die entsprechenden Funktionen administrativ abschalten ließen, um eine potenzielle Gefährdung zu unterbinden. Darüber hinaus ist der Quelltext von KVM nicht zu beanstanden.

Die Anbindung des Gastes an ein Netzwerk wird auf verschiedenen Wegen ermöglicht. Alle untersuchten und für die Virtualisierung von Serversystemen relevanten Anbindungsmethoden können eine strikte Isolation des Gast-Netzwerks von den weiteren Netzwerken realisieren. Auch Dienste, die auf dem Host ausgeführt werden, können für die Gäste unzugänglich gemacht werden. Bei einigen Anbindungsarten wird

dies allerdings nicht automatisch durch libvirt umgesetzt. Weiterhin existieren einige übliche Konfigurationen, die eine entsprechende Isolation aufheben, ohne dass dies für einen Administrator offensichtlich ist. Die Anbindung an das Netzwerk ist daher in der Praxis wohl die primäre Quelle sicherheitskritischer Konfigurationsfehler. Eine individuelle Überprüfung der Isolation des Gastes von den weiteren Netzwerken und Diensten des Hosts ist daher empfehlenswert.

Darüber hinaus schützt keine der untersuchten Konfigurationen vor typischen netzwerkbasierten Angriffen, wie z. B. ARP- oder IP-Spoofing. Zur Abwehr solcher Angriffe stehen je nach Anbindungsmethode zwei Ansätze zur Verfügung. Zum einen können mit libvirt entsprechende Filterregeln erstellt werden. Auf diesem Wege ist eine effektive Abwehr einer großen Anzahl an netzwerkbasierten Angriffen möglich. Zum anderen unterstützen einige der Anbindungsmethoden eine vollständige Ausleitung der gesamten Kommunikation und somit eine externe Filterung.

Überdies ermöglicht QEMU neben der Speicherung von Festplattenabbildern im lokalen Dateisystem die Anbindung an unterschiedliche netzwerkbasierte Speicher-Backends. Eine effektive hostbasierte Verschlüsselung und Signierung dieser Abbilder wird von QEMU jedoch nicht unterstützt. Die diesbezüglich einzige von QEMU angebotene Funktion ist die Verschlüsselung bei Nutzung des Formats qcow2. Von dieser Möglichkeit wird jedoch wegen gravierender Sicherheitsmängel abgeraten. Unabhängig von der Virtualisierung können allgemeine Funktionen des Betriebssystems zur Verschlüsselung und Signierung, genutzt werden. Dies bedingt aber meistens einen deutlich erhöhten administrativen Aufwand.

Im Hinblick auf den Einsatz der netzwerkbasierten Speicherlösungen Ceph und GlusterFS ist festzustellen, dass beide geeignet sind, Festplattenabbilder hochverfügbar zu speichern. Zur Absicherung sollte jedoch eine gegenseitige Authentifizierung durch das Speicher-Backend eingerichtet werden. Eine solche gegenseitige Authentifizierung wird von beiden Systemen optional angeboten. GlusterFS nutzt hierzu TLS und Zertifikate. Auf diesem Wege wird nicht nur eine sichere Authentifizierung, sondern auch eine Verschlüsselung und Signierung der Netzwerkkommunikation erreicht. Ceph greift zur Authentifizierung auf ein eigens entwickeltes Verfahren zurück. Auch wenn dieses Verfahren nicht im Detail analysiert wurde, kann davon ausgegangen werden, dass es eine zumindest grundsätzlich sichere Authentifizierung ermöglicht. Eine Verschlüsselung der Netzwerkkommunikation wird von Ceph jedoch nicht unterstützt und eine Signierung nur auf kryptografisch schwache Weise.

Sowohl Ceph als auch GlusterFS ermöglichen es, den Zugriff auf die gespeicherten Daten zu beschränken. Die zugrunde liegenden Autorisierungsverfahren sind jedoch nur recht grob konfigurierbar. Typischerweise wird ein Host autorisiert, auf eine große Anzahl von Festplattenabbilder zuzugreifen. Sowohl mit Ceph als auch mit GlusterFS ist es nur sehr bedingt möglich, den Host derart einzuschränken, dass er nur auf solche Abbilder zugreifen kann, die er zur Ausführung seiner Gäste tatsächlich benötigt.

Die wichtigste Schutzmaßnahme ist somit in aller Regel eine strikte physische wie logische Zugangsbeschränkung zu Netzwerk und beteiligten Komponenten des Speichernetzwerkes. Diese gewährleistet zum einen die Verfügbarkeit der Festplattenabbilder. Zum anderen ist sie erforderlich, weil eine Verschlüsselung und Signierung der Festplattenabbilder während der Speicherung und der Übermittlung nur bedingt praktikabel umgesetzt werden kann.

Das auf den beschriebenen Komponenten aufbauende OpenStack dient dazu, komplexe Cloud-Umgebungen zu realisieren. Es wird im Rahmen dieser Studie jedoch nur ausschnittsweise betrachtet, mit dem Fokus auf der Netzwerkkommunikation und der Authentifizierung. Bezüglich der Netzwerkkommunikation ist festzuhalten, dass eine Absicherung mittels TLS für alle zu OpenStack gehörenden Dienste möglich ist. Darüber hinaus bietet OpenStack einen zentralen Authentifizierungsdienst. Die unterstützten Verfahren sind grundlegend tauglich, um eine sichere Authentifizierung durchzuführen. Problematisch ist, dass die zum Identitätsnachweis verwendeten Token dienstunabhängig sind. Gelingt es einem Angreifer, einen solchen Token einzusehen, so kann er die Identität des Token-Besitzers gegenüber allen Diensten der Cloud einnehmen. Dies kann insbesondere zu einer Rechteausweitung des Angreifers nach der Kompromittierung eines einzelnen Dienstes führen.

Abschließend zusammengefasst sind die untersuchten Komponenten – allen voran KVM, QEMU und libvirt – dazu geeignet, eine technisch ausgereifte und sichere Virtualisierungsumgebung zu realisieren.

13 Literaturverzeichnis

CVE-2010-0306: 2010, https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2010-0306
CVE-2010-0298: 2010, https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2010-0298
PCI: Ravi Budruk, Don Anderson, Tom Shanley, PCI Express System Architecture, 2004
CVE-2014-3610: 2014, https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2014-3610
CVE-2013-1797: 2013, https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2013-1797
CVE-2013-1796: 2013, https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2013-1796
CVE-2009-3722: 2009, https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2009-3722
VMWare: , Performance Evaluation of Intel EPT Hardware Assist,
IOCTL: , Linux Programmer's Manual, 2013
UNIX-PROG: Jürgen Wolf, Linux-Unix-Programmierung, 2006
OAS: Rusty Russell, Michael S. Tsirkin, Cornelia Huck, Pawel Moll, Virtual I/O Device (VIRTIO) Version 1.0 , 2015, http://docs.oasis-open.org/virtio/virtio/v1.0/cs03/virtio-v1.0-cs03.html
NBD: Wouter Verhelst, The NBD protocol, 2015, http://sourceforge.net/p/nbd/code/ci/master/tree/doc/proto.md
BER: Daniel Berrange, QEMU QCow2 built-in encryption: just say no. Deprecated now, to be deleted soon, 2015, https://www.berrange.com/posts/2015/03/17/qemu-qcow2-built-in-encryption-just-say-no-deprecated-now-to-be-deleted-soon/
CVE-2013-0311: 2013, https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2013-0311
LIBVIRT: 2016, https://libvirt.org/docs.html
CVE-2011-1751: 2011, https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2011-1751
QEMUCTR: , Contribute/SubmitAPatch, , http://wiki.qemu.org/Contribute/SubmitAPatch
STYLE: , http://git.qemu-project.org/?p=qemu.git;a=blob_plain;f=CODING_STYLE;hb=a8c40fa2d667e585382080db36ac44e216b37a1c
CODING: , http://git.qemu-project.org/?p=qemu.git;a=blob_plain;f=HACKING;hb=a8c40fa2d667e585382080db36ac44e216b37a1c
SSP: Tobias Klein, Stack Smashing Protector, 2003
SECCOMP: , SECure COMPuting with filters,
GCC: , GCC 6.2 manual,
ARTSSA: Dowd, McDonald, Schuh, The Art of Software Security Assessment, 2007
FLAWF: , flawfinder manual,
KVMHV: Vadim Rozenfeld, KVM as a Microsoft-compatible hypervisor, 2012
INTEL2: , Intel 64 and IA-32 Architectures Software Developer's Manual, 2016
KVMMSR: Glauber Costa, KVM-specific MSRs, 2010
INTEL3A: , Intel 64 and IA-32 Architectures Software Developer's Manual, 2016
AMD2: , AMD64 Architecture Programmer's Manual, 2016
INTEL3C: , Intel 64 and IA-32 Architectures Software Developer's Manual, 2016
VMRNG: , Analysis of Random Number Generation in Virtual Environments, 2016
INTEL1: , Intel 64 and IA-32 Architectures Software Developer's Manual, 2016
INTEL3B: , Intel 64 and IA-32 Architectures Software Developer's Manual, 2016
X2APIC: , Intel 64 Architecture x2APIC Specification, 2010
AMD3: , AMD64 Architecture Programmer's Manual, 2015
AMD4: , AMD64 Architecture Programmer's Manual, 2015
VT-D: , Intel® Virtualization Technology for Directed I/O - Architecture Specification, 2016
AMD-IO: , AMD I/O Virtualization Technology (IOMMU) Specification, 2015
HASE: Rafal Wojtczuk, Joanna Rutkowska, Following the White Rabbit: Software attacks against Intel VT-d technology, 2011
PLS: Gabor Pek, Andrea Lanzi, Abhinav Srivastava, On the Feasibility of Software Attacks on Commodity Virtual Machine Monitors via Direct Device Assignment, 2014
FIT: Joe Fitzpatrick, Stupid PCIe Tricks, 2014

HHU: W-H. Hu, Lattice scheduling and covert channels, 1992
RIS: Thomas Ristenpart, Eran Tromer, Hovav Shacham, Stefan Savage, Hey, You, Get Off of My Cloud: Exploring Information Leakage in Third Party Compute Clouds, 2009
ZWU: Zhenyu Wu, Zhang Xu, Haining Wang, Whispers in the Hyper-space: High-speed Covert Channels Attacks in the Cloud, 2015
PER: Colin Percival, Cache Missing For Fun And Profit, 2005
LLC: Fangfei Liu, Yuval Yarom, Qian Ge, Gernot Heiser, Ruby B. Lee, Last-Level Cache Side-Channel Attacks are Practical, 2015
GR1: J. W. Gray III., On introducing noise into the bus-contention channel, 1993
GR2: J. W. Gray III., Countermeasures and tradeoffs for a class of covert timing channels, 1994
WHU: W. Hu, Reducing timing charmers with fuzzing time, 1991
SIG: Jens-Rainer Ohm, Hans Dieter Lüke, Signalübertragung - Grundlagen der digitalen und analogen Nachrichtenübertragungssysteme,
WAM: Gorka Irazoqui, Mehmet Sinan Inci, Thomas Eisenbarth, Berk Sunar, Wait a minute! A fast, Cross-VM attack on AES, 2014
KVMS: , KVM security, 2012
IP-SYSCTL: , https://www.kernel.org/doc/Documentation/networking/ip-sysctl.txt
MACVTAP: 2010, http://virt.kernelnewbies.org/MacVTap
QCOWE: Daniel Berrange, QEMU QCow2 built-in encryption: just say no. Deprecated now, to be deleted soon, 2015
DMCRYPT: , dm-crypt,
DMINT: , dm-integrity,
CPB: Dan van der Ster, Herve Rousseau, Ceph ~ 30PBTestReport,
CRUSH: Sage A. Weil, Scott A. Brandt, Ethan L. Miller, Carlos Maltzahn, Controlled, Scalable, Decentralized Placement of Replicated Data, 2006
CDOC: , Ceph Documentation,
TLSCS: , Transport Layer Protection Cheat Sheet, 2016
RFC4122: P. Leach, M. Mealling, R. Salz, A Universally Unique IDentifier (UUID) URN Namespace, 2005
ITGS: , IT-Grundschutz-Kataloge, 2016

Anhang A

```
#!/bin/bash

ovs-ofctl del-flows ovs-bridge

ovs-ofctl add-flows ovs-bridge - <<EOF


# Der Host (Port 1) darf frei kommunizieren
add priority=1000 in_port=1 action=normal

# UDP-Verkehr auf Port 67 wird verworfen (DHCP-Snooping)
add priority=900 udp udp_src=67 action=drop


# ================= GAST 1 ======================
# Port: 2
# MAC: 52:54:00:74:f7:15
# IPv4: 10.100.4.92
# IPv6: 2a01:198:5a1:4:5054:ff:fe74:f715
# ================================================

# DHCP-Anfragen werden ausschließlich auf Port 1 ausgeleitet
add priority=600 in_port=2 udp dl_src=52:54:00:74:f7:15 \
   udp_dst=67 action=output:1

# ARP-Antworten, die IP-Adresse 10.100.4.92 zur MAC-Adresse
# 52:54:00:74:f7:15 auflösen (oder umgekehrt) sind erlaubt.
add priority=500 in_port=2 arp dl_src=52:54:00:74:f7:15 \
   nw_src=10.100.4.92 action=normal

add priority=500 in_port=2 rarp dl_src=52:54:00:74:f7:15 \
   nw_src=10.100.4.92 action=normal

# Der Gast an Port 2 darf die IP-Adresse 10.100.4.92
# mit der MAC-Adresse 52:54:00:74:f7:15 nutzen
add priority=500 in_port=2 ip dl_src=52:54:00:74:f7:15 \
   nw_src=10.100.4.92 action=normal

# Ein Neighbor Advertisement mit der Zuordnung der IP-Adresse
# 2a01:198:5a1:4:5054:ff:fe74:f715 zur MAC-Adresse
# 52:54:00:74:f7:15 ist erlaubt.
# Andere Neighbor Advertisement oder Router Advertisement
# werden verworfen.
add priority=700 in_port=2 icmp6 icmp_type=136 \
   nd_tll=52:54:00:74:f7:15 \
   nd_target=2a01:198:5a1:4:5054:ff:fe74:f715 action=normal
add priority=600 in_port=2 icmp6 icmp_type=136 action=drop
add priority=600 in_port=2 icmp6 icmp_type=134 action=drop

# Der Gast an Port 2 darf die IP-Adresse
# 2a01:198:5a1:4:5054:ff:fe74:f715 mit der MAC-Adresse
# 52:54:00:74:f7:15 nutzen.
add priority=500 in_port=2 ipv6 dl_src=52:54:00:74:f7:15 \
   ipv6_src=2a01:198:5a1:4:5054:ff:fe74:f715 action=normal
```

```
# ================= GAST 2 =======================
# Port: 3
# MAC: 52:54:00:c7:54:24
# IPv4: 10.100.4.93
# IPv6: 2a01:198:5a1:4:5054:ff:fec7:5424
# ================================================

# DHCP-Anfragen werden ausschließlich auf Port 1 ausgeleitet
add priority=600 in_port=3 udp dl_src=52:54:00:c7:54:24 \
   udp_dst=67 action=output:1

# ARP-Antworten, die IP-Adresse 10.100.4.93 zur MAC-Adresse
# 52:54:00:c7:54:24 auflösen (oder umgekehrt) sind erlaubt.
add priority=500 in_port=3 arp dl_src=52:54:00:c7:54:24 \
   nw_src=10.100.4.93 action=normal
add priority=500 in_port=3 rarp dl_src=52:54:00:c7:54:24 \
   nw_src=10.100.4.93 action=normal

# Der Gast an Port 3 darf die IP-Adresse 10.100.4.93 mit der
# MAC-Adresse 52:54:00:c7:54:24 nutzen
add priority=500 in_port=3 ip dl_src=52:54:00:c7:54:24 \
   nw_src=10.100.4.93 action=normal

# Ein Neighbor Advertisement mit der Zuordnung der IP-Adresse
# 2a01:198:5a1:4:5054:ff:fec7:5424 zur MAC-Adresse
# 52:54:00:c7:54:24 ist erlaubt.
# Andere Neighbor Advertisement oder Router Advertisement
# werden verworfen
add priority=700 in_port=3 icmp6 icmp_type=136 \
   nd_tll=52:54:00:c7:54:24 \
   nd_target=2a01:198:5a1:4:5054:ff:fec7:5424 action=normal
add priority=600 in_port=3 icmp6 icmp_type=136 action=drop
add priority=600 in_port=3 icmp6 icmp_type=134 action=drop

# Der Gast an Port 3 darf die IP-Adresse
# 2a01:198:5a1:4:5054:ff:fec7:5424 mit der MAC-Adresse
# 52:54:00:c7:54:24 nutzen.
add priority=500 in_port=3 ipv6 dl_src=52:54:00:c7:54:24 \
   ipv6_src=2a01:198:5a1:4:5054:ff:fec7:5424 action=normal

EOF
```

Anhang B

```
./configure --target-list=x86_64-softmmu --disable-werror --cc=/usr/local/bin/gcc --cxx=/usr/local/bin/gcc '--extra-cflags=-Wextra -Wformat=2 -Wnonnull -Wnull-dereference -Winit-self -Wimplicit-int -Wimplicit-function-declaration -Wignored-attributes -Wmisleading-indentation -Wmissing-braces -Wparentheses -Wsequence-point -Wreturn-type -Wshift-count-negative -Wshift-count-overflow -Wshift-negative-value -Wswitch -Wswitch-bool -Wsync-nand -Wunused -Wuninitialized -Winvalid-memory-model -Wmaybe-uninitialized -Wunknown-pragmas -Wstrict-overflow -Warray-bounds=2 -Wbool-compare -Wtautological-compare -Wtrampolines -Wfloat-equal -Wdeclaration-after-statement -Wundef -Wplacement-new=2 -Wpointer-arith -Wtype-limits -Wbad-function-cast -Wcast-align -Wwrite-strings -Wconversion -Wdate-time -Wempty-body -Wenum-compare -Wjump-misses-init -Wsign-compare -Wsign-conversion -Wfloat-conversion -Wsizeof-pointer-memaccess -Wsizeof-array-argument -Wmemset-transposed-args -Waddress -Wlogical-op -Wlogical-not-parentheses -Waggregate-return -Wstrict-prototypes -Wold-style-declaration -Wold-style-definition -Wmissing-parameter-type -Wmissing-prototypes -Wmissing-declarations -Wmissing-field-initializers -Wpadded -Wredundant-decls -Wvolatile-register-var -Wstack-protector -Wno-unused-parameter -Wno-padded -Wno-declaration-after-statement -Wno-aggregate-return'
Disabling libtool due to broken toolchain support
Install prefix    /usr/local
BIOS directory    /usr/local/share/qemu
binary directory  /usr/local/bin
library directory /usr/local/lib
module directory  /usr/local/lib/qemu
libexec directory /usr/local/libexec
include directory /usr/local/include
config directory  /usr/local/etc
local state directory   /usr/local/var
Manual directory  /usr/local/share/man
ELF interp prefix /usr/gnemul/qemu-%M
Source path       /root/qemu-gcc
C compiler        /usr/local/bin/gcc
Host C compiler   cc
C++ compiler      /usr/local/bin/gcc
Objective-C compiler /usr/local/bin/gcc
ARFLAGS           rv
CFLAGS            -O2 -U_FORTIFY_SOURCE -D_FORTIFY_SOURCE=2 -pthread -I/usr/include/glib-2.0 -I/usr/lib/x86_64-linux-gnu/glib-2.0/include  -g
QEMU_CFLAGS       -I/usr/include/pixman-1   -fPIE -DPIE -m64 -D_GNU_SOURCE -D_FILE_OFFSET_BITS=64 -D_LARGEFILE_SOURCE -Wstrict-prototypes -Wredundant-decls -Wall -Wundef -Wwrite-strings -Wmissing-prototypes -fno-strict-aliasing -fno-common  -Wextra -Wformat=2 -Wnonnull -Wnull-dereference -Winit-self -Wimplicit-int -Wimplicit-function-declaration -Wignored-attributes -Wmisleading-indentation -Wmissing-braces -Wparentheses -Wsequence-point -Wreturn-type -Wshift-count-negative -Wshift-count-overflow -Wshift-negative-value -Wswitch -Wswitch-bool -Wsync-nand -Wunused -Wuninitialized -Winvalid-memory-model -Wmaybe-uninitialized -Wunknown-pragmas -Wstrict-overflow -Warray-bounds=2 -Wbool-compare -Wtautological-compare -Wtrampolines -Wfloat-equal -Wdeclaration-after-statement -Wundef -Wplacement-new=2 -Wpointer-arith -Wtype-limits -Wbad-function-cast -Wcast-align -Wwrite-strings -Wconversion -Wdate-time -Wempty-body -Wenum-compare -Wjump-misses-init -Wsign-compare -Wsign-conversion -Wfloat-conversion -Wsizeof-pointer-memaccess -Wsizeof-array-argument -Wmemset-transposed-args -Waddress -Wlogical-op -Wlogical-not-parentheses -Waggregate-return -Wstrict-prototypes -Wold-style-declaration -Wold-style-definition -Wmissing-parameter-type -Wmissing-prototypes -Wmissing-declarations -Wmissing-field-initializers -Wpadded -Wredundant-decls -Wvolatile-register-var -Wstack-protector -Wno-unused-parameter -Wno-padded -Wno-declaration-after-statement -Wno-aggregate-return -Wendif-labels -Wmissing-include-dirs -Wempty-body -Wnested-externs -Wformat-security -Wformat-y2k -Winit-self -Wignored-qualifiers -Wold-style-declaration -Wold-style-definition -Wtype-limits -fstack-protector-strong -I/usr/include/libpng12  -I/usr/include/spice-server -I/usr/include/glib-2.0 -I/usr/lib/x86_64-linux-gnu/glib-2.0/include -I/usr/include/pixman-1 -I/usr/include/spice-1  -I/usr/include/libusb-1.0
LDFLAGS           -Wl,--warn-common -Wl,-z,relro -Wl,-z,now -pie -m64 -g
make              make
install           install
python            python -B
smbd              /usr/sbin/smbd
module support    no
host CPU          x86_64
host big endian   no
target list       x86_64-softmmu
tcg debug enabled no
```

```
199 gprof enabled     no
200 sparse enabled    no
201 strip binaries    yes
202 profiler          no
203 static build      no
204 pixman            system
205 SDL support       yes
206 GTK support       yes
207 GTK GL support    no
208 GNUTLS support    no
209 GNUTLS hash       no
210 libgcrypt         yes
211 nettle            no ()
212 libtasn1          no
213 VTE support       no
214 curses support    yes
215 virgl support     no
216 curl support      yes
217 mingw32 support   no
218 Audio drivers     oss
219 Block whitelist (rw)
220 Block whitelist (ro)
221 VirtFS support    yes
222 VNC support       yes
223 VNC SASL support  yes
224 VNC JPEG support  yes
225 VNC PNG support   yes
226 xen support       yes
227 xen ctrl version  420
228 brlapi support    yes
229 bluez  support    yes
230 Documentation     yes
231 PIE               yes
232 vde support       yes
233 netmap support    no
234 Linux AIO support yes
235 ATTR/XATTR support yes
236 Install blobs     yes
237 KVM support       yes
238 RDMA support      no
239 TCG interpreter   no
240 fdt support       yes
241 preadv support    yes
242 fdatasync         yes
243 madvise           yes
244 posix_madvise     yes
245 sigev_thread_id   yes
246 uuid support      yes
247 libcap-ng support yes
248 vhost-net support yes
249 vhost-scsi support yes
250 Trace backends    nop
251 spice support     yes (0.12.7/0.12.5)
252 rbd support       yes
253 xfsctl support    yes
254 smartcard support no
255 libusb            yes
256 usb net redir     yes
257 OpenGL support    no
258 libiscsi support  yes
259 libnfs support    yes
260 build guest agent yes
```

```
261 QGA VSS support   no
262 QGA w32 disk info no
263 QGA MSI support   no
264 seccomp support   yes
265 coroutine backend ucontext
266 coroutine pool    yes
267 GlusterFS support yes
268 Archipelago support no
269 gcov              gcov
270 gcov enabled      no
271 TPM support       yes
272 libssh2 support   yes
273 TPM passthrough   yes
274 QOM debugging     yes
275 vhdx              yes
276 lzo support       yes
277 snappy support    yes
278 bzip2 support     yes
279 NUMA host support yes
280 tcmalloc support  no
281 jemalloc support  no
```

Anhang C

Bibliothek	Nutzt Stack Protection?	Nutzt NX-Bit?
ld-2.17.so	nein	ja
libacl.so.1.1.0	ja	ja
libaio.so.1.0.1	nein	ja
libanonymous.so.3.0.0	ja	ja
libasound.so.2.0.0	ja	ja
libasyncns.so.0.3.1	ja	ja
libattr.so.1.1.0	ja	ja
libboost_system-mt.so.1.53.0	ja	ja
libboost_thread-mt.so.1.53.0	ja	ja
libbz2.so.1.0.6	ja	ja
libc-2.17.so	ja	ja
libcap.so.2.22	ja	ja
libcelt051.so.0.0.0	ja	ja
libcom_err.so.2.1	ja	ja
libcrammd5.so.3.0.0	ja	ja
libcrypt-2.17.so	nein	ja
libcrypto.so.1.0.1e	ja	ja
libcurl.so.4.3.0	ja	ja
libdb-5.3.so	ja	ja
libdbus-1.so.3.7.4	ja	ja
libdigestmd5.so.3.0.0	ja	ja
libdl-2.17.so	nein	ja
libdw-0.163.so	ja	ja
libelf-0.163.so	ja	ja
libffi.so.6.0.1	ja	ja
libFLAC.so.8.3.0	ja	ja
libfreebl3.so	ja	ja
libgcc_s-4.8.5-20150702.so.1	nein	ja
libgcrypt.so.11.8.2	ja	ja
libgfapi.so.0.0.0	ja	ja
libgfrpc.so.0.0.1	ja	ja
libgfxdr.so.0.0.1	ja	ja
libglib-2.0.so.0.4200.2	ja	ja

Bibliothek	Nutzt Stack Protection?	Nutzt NX-Bit?
libglusterfs.so.0.0.1	ja	ja
libgmp.so.10.2.0	ja	ja
libgnutls.so.28.41.0	ja	ja
libgpg-error.so.0.10.0	ja	ja
libgsm.so.1.0.12	ja	ja
libgssapi_krb5.so.2.2	ja	ja
libgthread-2.0.so.0.4200.2	nein	ja
libhogweed.so.2.5	ja	ja
libibverbs.so.1.0.0	ja	ja
libICE.so.6.3.0	ja	ja
libidn.so.11.6.11	ja	ja
libiscsi.so.2.0.10900	ja	ja
libjpeg.so.62.1.0	ja	ja
libjson-c.so.2.0.1	ja	ja
libk5crypto.so.3.1	ja	ja
libkeyutils.so.1.5	ja	ja
libkrb5.so.3.3	ja	ja
libkrb5support.so.0.1	ja	ja
liblber-2.4.so.2.10.3	ja	ja
libldap-2.4.so.2.10.3	ja	ja
liblzma.so.5.0.99	ja	ja
liblzo2.so.2.0.0	ja	ja
libm-2.17.so	nein	ja
libnettle.so.4.7	ja	ja
libnl-3.so.200.16.1	ja	ja
libnl-route-3.so.200.16.1	ja	ja
libnsl-2.17.so	nein	ja
libnspr4.so	ja	ja
libnss3.so	ja	ja
libnss_files-2.17.so	nein	ja
libnssutil3.so	ja	ja
libogg.so.0.8.0	ja	ja
libp11-kit.so.0.0.0	ja	ja
libpcre.so.1.2.0	ja	ja
libpixman-1.so.0.32.6	ja	ja
libplc4.so	nein	ja

Bibliothek	Nutzt Stack Protection?	Nutzt NX-Bit?
libplds4.so	ja	ja
libpng15.so.15.13.0	ja	ja
libpthread-2.17.so	nein	ja
libpulsecommon-6.0.so	ja	ja
libpulse.so.0.18.0	ja	ja
librados.so.2.0.0	ja	ja
librbd.so.1.0.0	ja	ja
librdmacm.so.1.0.0	ja	ja
libresolv-2.17.so	ja	ja
librt-2.17.so	nein	ja
libsasl2.so.3.0.0	ja	ja
libsasldb.so.3.0.0	ja	ja
libseccomp.so.2.2.1	ja	ja
libselinux.so.1	ja	ja
libsmime3.so	ja	ja
libSM.so.6.0.1	ja	ja
libsnappy.so.1.1.4	ja	ja
libsndfile.so.1.0.25	ja	ja
libspice-server.so.1.8.0	ja	ja
libssh2.so.1.0.1	ja	ja
libssl3.so	ja	ja
libssl.so.1.0.1e	ja	ja
libstdc++.so.6.0.19	ja	ja
libsystemd.so.0.6.0	nein	ja
libtasn1.so.6.2.3	ja	ja
libtcmalloc.so.4.2.6	ja	ja
libtspi.so.1.2.0	ja	ja
libunwind.so.8.0.1	ja	ja
libusb-1.0.so.0.1.0	ja	ja
libusbredirparser.so.1.0.0	ja	ja
libutil-2.17.so	nein	ja
libuuid.so.1.3.0	ja	ja
libvorbisenc.so.2.0.9	ja	ja
libvorbis.so.0.4.6	ja	ja
libwrap.so.0.7.6	ja	ja
libX11.so.6.3.0	ja	ja

Bibliothek	Nutzt Stack Protection?	Nutzt NX-Bit?
libX11-xcb.so.1.0.0	nein	ja
libXau.so.6.0.0	ja	ja
libxcb.so.1.1.0	ja	ja
libXext.so.6.4.0	ja	ja
libXi.so.6.1.0	ja	ja
libXtst.so.6.1.0	ja	ja
libz.so.1.2.7	ja	ja

Anhang D

Ausschnitte aus der Datei virt.te

```
# svirt local policy
#

# it was a part of auth_use_nsswitch
allow svirt_t self:netlink_route_socket r_netlink_socket_perms;

corenet_udp_sendrecv_generic_if(svirt_t)
corenet_udp_sendrecv_generic_node(svirt_t)
corenet_udp_sendrecv_all_ports(svirt_t)
corenet_udp_bind_generic_node(svirt_t)
corenet_udp_bind_all_ports(svirt_t)
corenet_tcp_bind_all_ports(svirt_t)
corenet_tcp_connect_all_ports(svirt_t)

init_dontaudit_read_state(svirt_t)

...gekürzt...

# virtual domains common policy
#
allow virt_domain self:capability2 compromise_kernel;
allow virt_domain self:process { setrlimit signal_perms getsched setsched };
allow virt_domain self:fifo_file rw_fifo_file_perms;
allow virt_domain self:shm create_shm_perms;
allow virt_domain self:unix_stream_socket { connectto create_stream_socket_perms };
allow virt_domain self:unix_dgram_socket { create_socket_perms sendto };
allow virt_domain self:tcp_socket create_stream_socket_perms;
allow virt_domain self:udp_socket create_socket_perms;
allow virt_domain self:netlink_kobject_uevent_socket create_socket_perms;

list_dirs_pattern(virt_domain, virt_content_t, virt_content_t)
read_files_pattern(virt_domain, virt_content_t, virt_content_t)
dontaudit virt_domain virt_content_t:file write_file_perms;
```

```
dontaudit virt_domain virt_content_t:dir write;

kernel_read_net_sysctls(virt_domain)
kernel_read_network_state(virt_domain)
userdom_search_user_home_content(virt_domain)
userdom_read_user_home_content_symlinks(virt_domain)
userdom_read_all_users_state(virt_domain)
append_files_pattern(virt_domain, virt_home_t, virt_home_t)
manage_dirs_pattern(virt_domain, svirt_home_t, svirt_home_t)
manage_files_pattern(virt_domain, svirt_home_t, svirt_home_t)
manage_sock_files_pattern(virt_domain, svirt_home_t, svirt_home_t)
filetrans_pattern(virt_domain, virt_home_t, svirt_home_t, { dir sock_file file })
stream_connect_pattern(virt_domain, svirt_home_t, svirt_home_t, virtd_t)

manage_dirs_pattern(virt_domain, virt_cache_t, virt_cache_t)
manage_files_pattern(virt_domain, virt_cache_t, virt_cache_t)
files_var_filetrans(virt_domain, virt_cache_t, { file dir })

read_lnk_files_pattern(virt_domain, virt_image_t, virt_image_t)

manage_dirs_pattern(virt_domain, svirt_image_t, svirt_image_t)
manage_files_pattern(virt_domain, svirt_image_t, svirt_image_t)
manage_sock_files_pattern(virt_domain, svirt_image_t, svirt_image_t)
manage_fifo_files_pattern(virt_domain, svirt_image_t, svirt_image_t)
read_lnk_files_pattern(virt_domain, svirt_image_t, svirt_image_t)
rw_chr_files_pattern(virt_domain, svirt_image_t, svirt_image_t)
rw_blk_files_pattern(virt_domain, svirt_image_t, svirt_image_t)
fs_hugetlbfs_filetrans(virt_domain, svirt_image_t, file)

manage_dirs_pattern(virt_domain, svirt_tmp_t, svirt_tmp_t)
manage_files_pattern(virt_domain, svirt_tmp_t, svirt_tmp_t)
manage_lnk_files_pattern(virt_domain, svirt_tmp_t, svirt_tmp_t)
files_tmp_filetrans(virt_domain, svirt_tmp_t, { file dir lnk_file })
userdom_user_tmp_filetrans(virt_domain, svirt_tmp_t, { dir file lnk_file })

manage_dirs_pattern(virt_domain, svirt_tmpfs_t, svirt_tmpfs_t)
```

```
manage_files_pattern(virt_domain, svirt_tmpfs_t, svirt_tmpfs_t)
manage_lnk_files_pattern(virt_domain, svirt_tmpfs_t, svirt_tmpfs_t)
fs_tmpfs_filetrans(virt_domain, svirt_tmpfs_t, { dir file lnk_file })

manage_dirs_pattern(virt_domain, qemu_var_run_t, qemu_var_run_t)
manage_files_pattern(virt_domain, qemu_var_run_t, qemu_var_run_t)
manage_sock_files_pattern(virt_domain, qemu_var_run_t, qemu_var_run_t)
manage_lnk_files_pattern(virt_domain, qemu_var_run_t, qemu_var_run_t)
files_pid_filetrans(virt_domain, qemu_var_run_t, { dir file })
stream_connect_pattern(virt_domain, qemu_var_run_t, qemu_var_run_t, virtd_t)

dontaudit virtd_t virt_domain:process  { siginh noatsecure rlimitinh };

dontaudit virt_domain virt_tmpfs_type:file { read write };
append_files_pattern(virt_domain, virt_log_t, virt_log_t)

append_files_pattern(virt_domain, virt_var_lib_t, virt_var_lib_t)

corecmd_exec_bin(virt_domain)
corecmd_exec_shell(virt_domain)

corenet_tcp_sendrecv_generic_if(virt_domain)
corenet_tcp_sendrecv_generic_node(virt_domain)
corenet_tcp_sendrecv_all_ports(virt_domain)
corenet_tcp_bind_generic_node(virt_domain)
corenet_tcp_bind_vnc_port(virt_domain)
corenet_tcp_bind_virt_migration_port(virt_domain)
corenet_tcp_connect_virt_migration_port(virt_domain)
corenet_rw_inherited_tun_tap_dev(virt_domain)

dev_list_sysfs(virt_domain)
dev_getattr_fs(virt_domain)
dev_dontaudit_getattr_all(virt_domain)
dev_read_generic_symlinks(virt_domain)
dev_read_rand(virt_domain)
dev_read_sound(virt_domain)
```

```
dev_read_urand(virt_domain)
dev_write_sound(virt_domain)
dev_rw_ksm(virt_domain)
dev_rw_vfio_dev(virt_domain)
dev_rw_kvm(virt_domain)
dev_rw_qemu(virt_domain)
dev_rw_inherited_vhost(virt_domain)
dev_rw_infiniband_dev(virt_domain)

domain_use_interactive_fds(virt_domain)

files_read_mnt_symlinks(virt_domain)
files_read_var_files(virt_domain)
files_search_all(virt_domain)

fs_getattr_xattr_fs(virt_domain)
fs_getattr_tmpfs(virt_domain)
fs_rw_anon_inodefs_files(virt_domain)
fs_rw_inherited_tmpfs_files(virt_domain)
fs_getattr_hugetlbfs(virt_domain)
fs_rw_inherited_nfs_files(virt_domain)
fs_rw_inherited_cifs_files(virt_domain)
fs_rw_inherited_noxattr_fs_files(virt_domain)

# I think we need these for now.
miscfiles_read_public_files(virt_domain)
miscfiles_read_generic_certs(virt_domain)

storage_raw_read_removable_device(virt_domain)

sysnet_read_config(virt_domain)

term_use_all_inherited_terms(virt_domain)
term_getattr_pty_fs(virt_domain)
term_use_generic_ptys(virt_domain)
term_use_ptmx(virt_domain)
```

```
tunable_policy(`virt_use_execmem',`
	allow virt_domain self:process { execmem execstack };
')

optional_policy(`
	alsa_read_rw_config(virt_domain)
')

optional_policy(`
	nscd_dontaudit_write_sock_file(virt_domain)
')

optional_policy(`
	nscd_dontaudit_read_pid(virt_domain)
')

optional_policy(`
	ptchown_domtrans(virt_domain)
')
optional_policy(`
	pulseaudio_dontaudit_exec(virt_domain)
')

optional_policy(`
	sssd_dontaudit_stream_connect(virt_domain)
	sssd_dontaudit_read_lib(virt_domain)
	sssd_dontaudit_read_public_files(virt_domain)
')

optional_policy(`
	virt_read_config(virt_domain)
	virt_read_lib_files(virt_domain)
	virt_read_content(virt_domain)
	virt_stream_connect(virt_domain)
	virt_read_pid_symlinks(virt_domain)
	virt_domtrans_bridgehelper(virt_domain)
```

```
')
optional_policy(`
	xserver_rw_shm(virt_domain)
')

tunable_policy(`virt_use_comm',`
	term_use_unallocated_ttys(virt_domain)
	dev_rw_printer(virt_domain)
')

tunable_policy(`virt_use_fusefs',`
	fs_manage_fusefs_dirs(virt_domain)
	fs_manage_fusefs_files(virt_domain)
	fs_read_fusefs_symlinks(virt_domain)
	fs_getattr_fusefs(virt_domain)
')
tunable_policy(`virt_use_nfs',`
	fs_manage_nfs_dirs(virt_domain)
	fs_manage_nfs_files(virt_domain)
	fs_manage_nfs_named_sockets(virt_domain)
	fs_read_nfs_symlinks(virt_domain)
	fs_getattr_nfs(virt_domain)
')

tunable_policy(`virt_use_samba',`
	fs_manage_cifs_dirs(virt_domain)
	fs_manage_cifs_files(virt_domain)
	fs_manage_cifs_named_sockets(virt_domain)
	fs_read_cifs_symlinks(virt_domain)
	fs_getattr_cifs(virt_domain)
')

tunable_policy(`virt_use_usb',`
	dev_rw_usbfs(virt_domain)
	dev_read_sysfs(virt_domain)
	fs_getattr_dos_fs(virt_domain)
```

```
        fs_manage_dos_dirs(virt_domain)
        fs_manage_dos_files(virt_domain)
')

optional_policy(`
  tunable_policy(`virt_use_sanlock',`
    sanlock_stream_connect(virt_domain)
  ')
')

tunable_policy(`virt_use_rawip',`
        allow virt_domain self:rawip_socket create_socket_perms;
')

optional_policy(`
        tunable_policy(`virt_use_xserver',`
                xserver_stream_connect(virt_domain)
        ')
')
```