der Bundeswehr
Universität München
FAKULTÄT FÜR INFORMATIK

Virtualisierung: Techniken und sicherheitsorientierte Anwendungen

Dr. Udo Helmbrecht
Prof. Dr. Gunnar Teege
Björn Stelte
(Hrsg.)

Institut für Technische Informatik

Bericht 2010-04
August 2010

Werner-Heisenberg-Weg 39 • 85577 Neubiberg • Germany

Zum Inhalt

Das Seminar „Virtualisierung: Techniken und sicherheitsorientierte Anwendungen“ setzt die erfolgreiche Reihe von Seminaren zum Themenkreis IT-Sicherheit am Institut für Technische Informatik an der Universität der Bundeswehr 2010 fort.

Mit Virtualisierung bezeichnen wir Methoden und Verfahren mit denen wir Ressourcen eines Computers oder einer Computer-Infrastruktur (beispielsweise im Server-Bereich) zusammenfassen oder aufteilen koennen. Mittels Virtualisierung wollen wir dem Benutzer eine Abstraktionsschicht zur Verfügung stellen, die die eigentlichen Hardware, Rechenleistung und Speicherplatz isoliert. IBM fuehrte schon 1970 mit dem erfolgreichen Grossrechner- System IBM/370 Virtualisierung ein.

Heute gibt es viele unterschiedliche am Markt erfolgreich implementierte Loesungen. Benjamin Hoffmann setzt sich in seiner Arbeit mit der Hostvirtualisierung auseinander und vergleicht Konzepte und Produkte. Die IT-Sicherheitsaspekte hat Ingo Schwarz im Kapitel 2 untersucht, indem er Angriffstechniken auf Server-Virtualisierung untersucht und deren Gefahrenpotential darstellt. Beide Arbeiten kommen zu dem Schluss, dass die eingesetzte Virtualiserungstechnik als auch die sicherheitstechnischen Stärken und Schwächen von den Anforderungen des Nutzers und dem Einsatzszenario abhängen.

Udo Helmbrecht
August 2010

Inhaltsverzeichnis

Kapitel 1

Hostvirtualisierung - Vergleich der Konzepte und Produkte

Benjamin Hoffmann

Jeder Anwender eines Computers ist bereits mit Virtualisierung in Kontakt gekommen und hat die Techniken genutzt, meist ohne dass es ihm bewusst war. Frühere Techniken der Virtualisierung wie das Multitasking, bei dem mehrere Anwendungen scheinbar zur selben Zeit auf einen einzigen vorhandenen Prozessor zugreifen, sind heute Standard und völlig selbstverständlich. Die heutigen Virtualisierungslösungen beschäftigen sich dabei nicht mehr nur mit einer Komponente, sondern wollen gleich den ganzen Rechner virtualisieren, um ihn in ein größeres virtuelles System zu integrieren oder ihn in viele kleinere aufzuteilen.

Virtualisierung ist dabei nicht gleich Virtualisierung. Vielmehr existiert inzwischen eine Vielzahl an verschiedenen Arten und Konzepten. Entsprechend vielfältig ist die Auswahl an Software-Produkten, die je nach gewünschter Virtualisierungstechnologie passende Möglichkeiten bereithält, wobei hier bereits in den letzten Jahren eine Konsolidierung stattgefunden hat. Des Weiteren beherrschen viele Produkte mehrere Möglichkeiten der Virtualisierung, sodass es nicht immer ganz einfach ist, das richtige Produkt überhaupt zu finden. Aufgrund der vielen Möglichkeiten, die sich mit oder in einer virtualisierten Umgebung ergeben, kann das Thema Virtualisierung nicht nur für Rechenzentren oder größere Unternehmen von Interesse sein, sondern auch für kleinere Firmen oder sogar Privatanwender. Diese Möglichkeiten sollen aufgezeigt werden.

Dieses Kapitel bietet daher einen Einstieg in die Grundbegriffe und Konzepte auf dem Gebiet der Virtualisierung. Dabei werden deren Funktionsmerkmale beschrieben und an jeweils einem Software-Beispiel deutlich gemacht, sowie deren Vor- und Nachteile hervorgehoben. Zusammen soll das einen möglichst guten Überblick über die wichtigsten Konzepte und Produkte der Virtualisierung geben.

Inhaltsverzeichnis

1.1 Einführung

Die Anfänge der Virtualisierung reichen bis in die 1960er Jahre zurück. Die x86-Architektur (von der Firma Intel mit dem 8086-Prozessor 1978 eingeführt) galt allerdings lange als nicht- bzw. nicht-performant virtualisierbar. Erst 1998 entwickelte die Firma VMware erste Technologien, um sie effizient virtualisieren zu können und präsentierte kurz darauf das erste entsprechende Produkt VMware Workstation. Von diesem Zeitpunkt an entwickeln sich bis heute verschiedene Konzepte und viele weitere Produkte, die im Folgenden detailliert vorgestellt werden.
Zunächst werden Grundlagen erklärt, die für den weiteren Verlauf des Kapitels wichtig sind. Der Virtual Machine Monitor als Virtualisierungsschicht wird vorgestellt. Zudem wird grob das Prinzip von Privilegien im Prozessor erklärt. Im Folgenden werden dann die verschiedenen Konzepte der Virtualisierung dargestellt und dabei deren Wirkungsprinzip und Merkmale beschrieben. Ein Vergleich aller Techniken miteinander birgt Schwierigkeiten, da sie mitunter grundverschieden sind und daher maximal in Einzelmerkmalen verglichen werden können. Dennoch wird in Kapitel 1.8 versucht, einen zusammenfassenden Überblick zu geben. Zu jeder der hier vorgestellten Virtualisierungstechniken wird zudem mindestens ein entsprechendes Produkt dargestellt. Hier sei angemerkt, dass es oftmals nicht ganz richtig ist, eine Software nur einer Kategorie zuzuordnen, da sie meist mehr als nur eine der Techniken beherrschen. Abschließend werden die wichtigsten Aspekte des Kapitels in einer Zusammenfassung noch einmal hervorgehoben.

1.2 Grundlagen

Virtualisierung ist zunächst einmal die Abstraktion physikalisch vorhandener Hardware. Die abstrahierte Hardware kann im Folgenden neu aufgeteilt werden und zu sogenannten virtuellen Maschinen (VM) zusammengefasst werden, die wie ein eigenständiges System angesprochen werden können (Partitionierung). Möglich ist auch der Schritt in die andere Richtung, um so mehrere Rechner zu einem großen leistungsfähigeren virtuellen System zusammenzufassen (Aggregation). Letzteres wird nicht Thema dieses Kapitels sein. Im allgemeinen Sprachgebrauch bedeutet Virtualisierung die parallele Ausführung von mehreren Betriebssystemen in je einer virtuellen Maschine auf einem einzigen Rechner.

1.2.1 Virtual Machine Monitor / Hypervisor

Den Virtual Machine Monitor (VMM) kann man als diejenige Zwischenschicht betrachten, die zwischen (Gast-)Betriebssystem und Hardware sitzt, um eine Virtualisierung überhaupt zu erreichen. Er verwaltet im Allgemeinen die Systemressourcen und teilt sie den virtuellen Maschinen zu. Je nach Virtualisierungslösung simuliert er abgefangene Befehle durch eine Interpreterroutine. Man unterscheidet zwischen zwei verschiedenen Arten des VMM, Typ 1 und Typ 2 [3, Seite 22]. Ein Typ-1-VMM wird auch als „bare metal“ bezeichnet, da er direkt auf der Hardware läuft. Ein Typ-2-VMM dagegen benötigt ein Host-Betriebssystem, auf welchem er ausgeführt wird. Dieser Typ wird daher auch als „hosted“ bezeichnet.

- **Beispiele für Typ 1:** Xen, VMware ESX, Microsoft Hyper-V
- **Beispiele für Typ 2:** VMware Server/Workstation, Microsoft Virtual PC, Sun VirtualBox

Hier sei angemerkt, dass der Begriff Virtual Machine Monitor oft äquivalent mit dem Begriff Hypervisor verwendet wird. Manchmal ist allerdings mit einem Hypervisor nur ein Typ-1-VMM gemeint, es wird dann also zwischen Hypervisor (Typ-1-VMM) und „hosted“ (Typ-2-VMM) unterschieden [19]. In diesem Text wird kein Unterschied zwischen den Begriffen gemacht.

1.2.2 Die Ringe

Jeder Prozess wird in einer bestimmten Privilegien-Stufe ausgeführt. Diese Stufen werden zum besseren Verständnis in Ringen dargestellt, die von 0 ab durchnummeriert werden. Die x86-Architektur stellt 4 dieser Ringe bereit. Dabei ist Ring 0 der privilegierteste (auch „kernel space“), nur in ihm sind Zugriffe auf die Hardware möglich. Bei einem nichtvirtuellen normalen Betrieb läuft das Betriebssystem somit in diesem Ring, Anwendungen in Ring 3, die beiden anderen Ringe werden nur bei OS/2 genutzt, von anderen Betriebssystemen werden sie meist nicht verwendet (Grund dafür ist eine bessere Portabilität von Anwendungen auf eine andere Architektur, die mitunter nur 2 Ringe zur Verfügung stellen [2, Seite 225 ff.]). So wird sichergestellt, dass Anwendungen nicht direkt mit der Hardware arbeiten können, sondern derartige Anfragen an das Betriebssystem weiterreichen müssen. Um ein Betriebssystem von der Hardware zu lösen, muss es also in einen anderen Ring verschoben werden. Bei der x86-Architektur macht man es sich meist zu Nutze, dass manche Ringe ohnehin leer sind und verschiebt die Gast-Systeme in der Regel in Ring 1. In Ring 0 sitzt fortan der VMM (Kapitel 1.2.1), wie es die Abbildung 1.1 zeigt. Daraus ergeben sich einige Schwierigkeiten, da jedes Betriebssystem in der Regel so konzipiert ist, dass es nur in Ring 0 ausgeführt werden kann. Mit diesen Schwierigkeiten umzugehen ist Aufgabe der verschiedenen Konzepte der Virtualisierung.

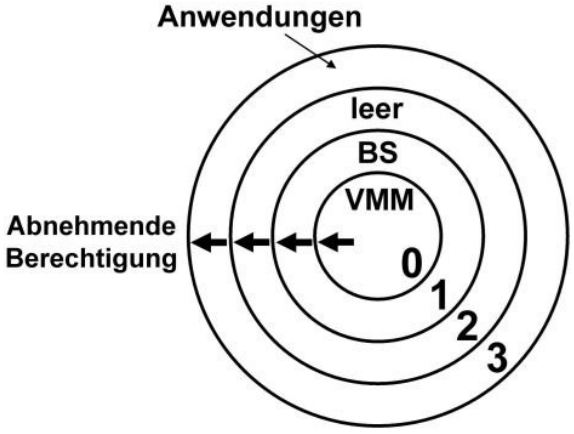

Abbildung 1.1: Ringsystem mit Virtualisierung.

AMD hat bei der Entwicklung der 64-Bit-Erweiterung für die x86-Architektur die Ringe 1 und 2 weggelassen, eben aus dem Grund, dass die meisten Betriebssysteme diese ohnehin nicht nutzen. Dies hatte allerdings Auswirkungen auf einige Virtualisierungsprodukte, die die Gäste bisher in den Ring 1 ausgelagert hatten. Diese mussten nun so modifiziert werden, dass dafür fortan auch Ring 3 verwendet werden konnte, um auch für die erweiterte Architektur Unterstützung zu bieten.

1.3 Vollständige Virtualisierung

1.3.1 Funktionsweise

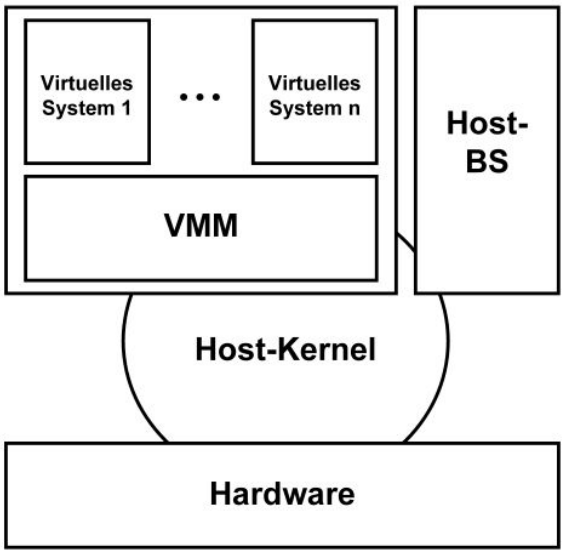

Abbildung 1.2: Prinzip der vollständigen Virtualisierung.

Bei der vollständigen Virtualisierung (in der Literatur oft auch als echte Virtualisierung oder Vollvirtualisierung bezeichnet) läuft das Gast-System komplett auf einer abstrakten Zwischenschicht und hat keinen direkten Kontakt zur physisch vorhandenen Hardware, wie es auch in Abbildung 1.2 zu sehen ist. Die Virtualisierung wird also durch Nachbildung der gesamten Hardwareumgebung erreicht. Der VMM teilt den Gästen Anteile der Hardware als virtuelle Hardware zu. Die virtuellen Systeme haben sogar ein eigenes BIOS [2], sodass sie eine komplette Nachbildung eines echten Rechners vorfinden, genau so wie sie ihn erwarten. Daher müssen Gast-Betriebssysteme bei einer vollständigen Virtualisierung nicht modifiziert werden. Jedes Betriebssystem, das auf einem nicht-virtuellen System läuft, läuft ebenso auf dessen virtuellem Gegenstück. Das bzw. die Gast-Betriebssysteme laufen in einem höheren Ring als Ring 0, der VMM in Ring 0. Dies führt dazu, dass bei jeder Operation eines Gast-OS, die einen privilegierten Status benötigt, eine Exception ausgelöst wird, die das laufende Gast-System unterbricht und zum VMM weiterleitet. Dieser muss nun entsprechende Behandlungen bereitstellen, um das Ereignis geeignet zu übersetzen und virtualisiert an das Gast-OS zurückreichen zu können [7]. Dieses Verfahren wird „Trap and Emulate“ genannt. Hier zeigt sich direkt ein großer Nachteil der vollständigen Virtualisierung. Der ständige Wechsel zwischen Gast-OS und VMM ist mit Performance-Einbußen verbunden, man spricht von bis zu 25 Prozent [2, Seite 129]. Zu diesem Wert kommt es auch, eben weil jede physikalische Komponente vom VMM virtualisiert und gesteuert werden muss.
Bei „hosted“-Lösungen läuft ein Hostbetriebssystem in Ring 0, auf dem wiederum der VMM läuft. Um hier die Exceptions überhaupt abfangen zu können, wird im Host-OS ein Kernel-Modul geladen, welches die Exceptions zum eigentlichen VMM zur Bearbeitung weiterleitet.

Binary Translation

Bei der x86-Architektur reicht dies aber noch nicht aus. So ist es auch in unprivilegierten Ringen möglich, Informationen über das Gesamtsystem auszulesen, was bei vollständiger Virtualisierung natürlich vermieden werden muss. Außerdem verhalten sich einige

Instruktionen in höheren Ringen anders, als wenn sie in Ring 0 ausgeführt werden und entsprechen damit nicht mehr dem erwarteten Verhalten des Gast-Betriebssystems. Diese Befehle müssen zusätzlich gefunden und virtualisiert werden. Da sie keine Exceptions auslösen, ist dies nur zur Laufzeit durch Untersuchen des Gast-Programmcodes kurz vor der Ausführung möglich. Wurde ein kritischer Befehl gefunden, muss er umgeschrieben werden und darf erst danach zur Ausführung gebracht werden. Dieses Umschreiben zur Laufzeit wird als Binary Translation bezeichnet [7, Seite 28].
Es existieren zwei Strategien bei der Virtualisierung von Befehlen bei Binary Translation. Zum einen kann ein solcher Befehl einfach derart verändert werden, dass er eine Exception auslöst und somit nach einem Kontext-Wechsel wieder vom VMM bearbeitet wird. Zum anderen kann der Befehl direkt in eine virtualisierte Version übersetzt und in den Code zurückgeschrieben werden. Letzteres erfordert einen höheren Implementationsaufwand, ist aber mit deutlich weniger Leistungseinbußen verbunden, da eine zusätzliche Unterbrechung und ein Wechsel zum VMM vermieden wird.

Vorteile der vollständigen Virtualisierung

- Das Gast-Betriebssystem muss nicht angepasst werden und kann unverändert in der virtuellen Maschine laufen. Somit können auch proprietäre Betriebssystem wie Microsoft Windows in einem virtuellen System laufen.
- Durch die vollständige Nachbildung der Hardware ist eine vielseitige und flexible Gast-Hardware möglich.

Nachteile der vollständigen Virtualisierung

- Die gesamte Hardware muss virtualisiert werden, was deutliche Leistungseinbußen gegenüber anderen Konzepten bedeutet.
- Bei „hosted“-Lösungen ist man auf die Kooperation des Host-Betriebssystems angewiesen, um ein entsprechendes Kernel-Modul nachinstallieren zu können.
- Die Implementation von Binary Translation ist komplex.
- Pro Gast sind hohe Systemressourcen nötig.

1.3.2 Beispiel VMware Server

Wie schon in der Einführung (siehe Kapitel 1.1) erwähnt, war VMware das erste Software-Haus, das eine Lösung zur Virtualisierung von x86-Architekturen entwickelte und vorstellte. Ihrem ersten Produkt VMware Workstation folgten zahlreiche weitere, darunter VMware ESX Server, VMware GSX Server, VMware Player und Programme zur Administration von Servern und deren virtuellen Maschinen. VMware nutzt als Virtualisierungstechnologie vor allem Binary Translation (Kapitel 1.3.1), wobei der prozessorunterstützten Virtualisierung eine immer größere Bedeutung zukommt [1] und VMware in neueren Produkten eine entsprechende Unterstützung bietet.

Da Software von VMware proprietär ist, ist über die Implementation der Technologie nicht viel bekannt. Allerdings lassen sich zum Beispiel aus einem Patentantrag von VMware einige Informationen hierzu ableiten [31]. Unter anderem geht daraus hervor, dass der VMM parallel zur Ausführung der virtuellen Maschine dessen Code liest bevor dieser ausgeführt wird und darin Befehle findet, die den Zustand der Maschine ändern würden (vgl. Kapitel 1.3.1).

Funktionen

Die Software VMware Server ist das Nachfolge-Produkt von GSX Server. Es ist nach Registrierung kostenlos für Windows und für Linux erhältlich. Die Funktionen ähneln denen der Workstation, die aber kostenpflichtig ist. Es unterstützt viele verschiedene Gast-Betriebssysteme, seit der letzten Version 2.0 auch 64-Bit-Systeme, allerdings dann nur mit speziellen Prozessor-Erweiterungen (siehe Kapitel 1.6). Es bietet den Gästen seit 2.0 unter anderem 10 verschieden konfigurierbare Netzwerkadapter, USB-2.0-Unterstützung mit der Möglichkeit bestimmte Geräte vom Host direkt an den Gast durch reichen zu lassen und bis zu 8 Gigabyte Hauptspeicher pro virtuelle Maschine (bei Version 1.0.x noch 3.6 Gigabyte). Diese und weitere Funktionen sind unter [12] nachzulesen.
Unterschiede zwischen der Workstation- und der Server-Version sind die netzwerkbasierte Auslegung von VMware Server. So ist diese Version eher dazu geeignet, die virtuellen Maschinen auf einem einzelnen leistungsstarken Server zu verwalten und betreiben und mit ggf. leistungsschwächeren Rechnern über ein Netzwerk darauf zuzugreifen. Des Weiteren sind die virtuellen Maschinen bei der Server-Version von Benutzeranmeldungen bzw. -abmeldungen unabhängig, können also bereits bei Systemstart automatisiert gestartet werden, ohne dass sich ein Benutzer anmeldet bzw. müssen nicht beendet werden, wenn sich ein Benutzer abmeldet. Deutlicher Nachteil der Server-Version gegenüber der Workstation-Version ist die Beschränkung der Snapshot-Funktion auf einen einzigen Snapshot, mit dem man zu einem früheren Zustand der virtuellen Maschine zurückkehren kann. Außerdem ist die Einrichtung von sogenannten Shared Folders, sowie Drag&Drop zwischen Host und Gast nicht möglich, mit denen Dateien auf einfache Weise ausgetauscht werden könnten.

Verwaltung

Die virtuellen Maschinen werden seit Version 2.0 über eine browserbasierte Oberfläche verwaltet [12]. Die Oberfläche nennt sich VI Web Access. Mit ihr ist es möglich entweder direkt am realen Server oder von einem entfernten Rechner Informationen über das Host-System und die virtuellen Maschinen einzuholen, virtuelle Maschinen einzurichten, zu administrieren und auszuführen bzw. zu beenden. Eine direkte Steuerung der virtuellen Maschine beziehungsweise des darauf laufenden Betriebssystems ist mit der VMware Remote Console möglich. Auch sie ist direkt auf dem Host der virtuellen Maschinen oder aber auf einem entfernten Rechner ausführbar. Die Installation geschieht über VI Web Access, danach läuft sie unabhängig davon, als Browser-Addon.

Einsatzgebiete

VMware Server ist vor allem für kleinere Produktivumgebungen interessant. Außerdem ist es nützlich für Testumgebungen auf einem zentralen Performance-starken Host, der somit Arbeitsplatz-Rechner entlasten kann [13]. Die Software eignet sich zudem zur Erstellung von virtuellen Maschinen, die danach mit dem ebenfalls kostenlosen VMware Player genutzt werden oder auch als kostenloser Einstieg in die Virtualisierung von Systemen, da die Übernahme von virtuellen Maschinen zum kostenpflichtigen Produkt VMware ESX Server möglich ist.

1.4 Paravirtualisierung

1.4.1 Funktionsweise

Die Idee der Paravirtualisierung ist es, die Gäste derart anzupassen, dass sie von sich aus befähigt sind, mit einer Virtualisierungsschicht zu arbeiten. Dazu wird der Befehlssatz der Gäste um sogenannte Hypercalls erweitert, die es ermöglichen sie direkt anzusprechen. Diese Schicht wird vom Hypervisor durch angepasste Schnittstellen bereitgestellt (siehe auch Abbildung 1.3). Die Hardware muss also nicht mehr vollständig virtualisiert werden und der Hypervisor verhält sich eher passiv [7, S. 31], da der Gast-Programmcode nicht zur Laufzeit untersucht werden muss. Da bei dieser Methode die ständigen Exceptions wegfallen, ist sie deutlich schneller als die vollständige Virtualisierung. Auf der anderen Seite können hierbei nur quelloffene Betriebssysteme als Gäste dienen, da ihr Kernel verändert bzw. erweitert werden muss. Der Open-Source-Hypervisor Xen ist wohl der bekannteste Vertreter der Paravirtualisierung und wird nachfolgend im Abschnitt 1.4.2 vorgestellt.

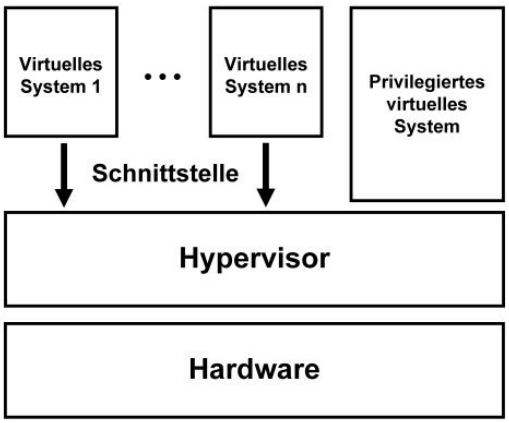

Abbildung 1.3: Prinzip der Paravirtualisierung

Die meisten Produkte dieser Art stellen eine besondere virtuelle Maschine bereit. Sie dient der Verwaltung des VMM, sodass sie Einfluss auf die anderen normalen virtuellen Maschinen nehmen kann. Sie kann direkt auf die Hardware zugreifen und initialisiert Geräte mittels passender Treiber. Außerdem macht sie die Hardware und die Geräte den anderen virtuellen Maschinen über den Hypervisor zugänglich. Der Hypervisor selbst verwaltet damit nur noch einen kleinen Teil der Hardware selbst und muss auch nur diesen Teil unterstützen.

Vorteile der Paravirtualisierung

- Deutlicher Geschwindigkeitsvorteil durch Wegfall der zahlreichen Exceptions.
- Die Hardware muss nur einmal durch den Hypervisor virtualisiert werden (Schnittstellen-Bereitstellung) und nicht für jede virtuelle Maschine.
- Den Kern eines Betriebssystems anzupassen ist weniger aufwendig als Binary Translation für die vollständige Virtualisierung zu realisieren [1].

Nachteile der Paravirtualisierung

- Der Kern des Gast-Betriebssystems muss angepasst werden. Das Betriebssystem muss dazu quelloffen sein und es muss ein entsprechender Patch vorhanden sein.
- Pro Gast sind hohe Systemressourcen nötig.

1.4.2 Beispiel Xen

Xen ist wohl das berühmteste Beispiel für ein Produkt der Paravirtualisierungs-Technik. Es existiert seit 2001, damals als Teil eines Projektes zu verteiltem Rechnen an der Universität Cambridge. Seit 2003 wird es unabhängig von diesem Projekt weiterentwickelt. In einem Dokument von 2003 wird Xen erstmals beschrieben [16]. Die Software war so erfolgreich, dass sich eine Firma namens XenSource bildete, die Xen-Produkte vertreibt und Support anbietet.
Xen bot zu Anfang nur Unterstützung für paravirtualisierte Gäste, seit der Version 3.0 kann Xen auch mit Hilfe der neueren Prozessor-Technologie (Kapitel 1.6) virtualisieren. Auf die entsprechenden Änderungen bei Xen, bei Verwendung dieser Technik, wird in Abschnitt 1.6.4 eingegangen.
Wie schon in Kapitel 1.4.1 erwähnt, müssen die für die virtuellen Maschinen bestimmten Betriebssysteme angepasst werden. Es muss also eine Art Patch für das jeweilige System geben, um die Arbeit mit Xen zu ermöglichen. Bei den verbreiteten Linux-Distributionen ist das in der Regel der Fall, weiterhin arbeiten auch NetBSD, FreeBSD und Solaris unter Xen. Windows-basierte Gäste können nicht paravirtualisiert betrieben werden, sondern nur unmodifiziert mit Hardwareunterstützung virtualisiert werden.
In der Xen-Terminologie wird eine virtuelle Maschine als Domäne bezeichnet. Xen besteht im wesentlichen aus drei Komponenten, dem Hypervisor selbst, einer privilegierten Domäne (Domain 0; Dom0) und den Gast-Domänen (Domain U; DomU).

Xen Hypervisor

Der Xen Hypervisor läuft als Typ-1-Hypervisor direkt auf der Hardware ohne Host-Betriebssystem. Er stellt die Schnittstelle zwischen der echten Hardware und den Gästen dar, indem er eine Virtualisierungsschicht bereitstellt, die der echten Hardware zwar ähnelt, sich aber doch davon unterscheidet. Er teilt weiterhin den Domains Teile des

Hauptspeichers zu. Dabei können Gäste bevorzugt werden, er garantiert jedoch gleichzeitig jedem Gast eine gewisse Mindestmenge. Ebenso teilt er die CPU den Gästen zu, sodass diese effizient genutzt wird. Die Hardwareunterstützung findet nicht durch den Hypervisor statt, sondern ist Aufgabe der Domain 0 [14].

Domain 0

Die sogenannte Domain 0 ist eine spezielle virtuelle Maschine mit privilegierten Rechten zur Verwaltung des Systems. Sie hat die Möglichkeit andere Domains zu erstellen und deren virtuelle Geräte zu verwalten. Sie übt des Weiteren administrative Aufgaben aus, wie zum Beispiel das Fortsetzen oder Migrieren einer Domain. Die Hardwareunterstützung wird durch die Treiber in der Domain 0 sichergestellt. Dies hat den Vorteil, dass der Hypervisor zum einen schlanker ist und zum anderen direkt von der Hardwareunterstützung des jeweilig verwendeten Kernels profitiert [2].
Innerhalb der Domain 0 läuft der Daemon xend. Er dient der Verwaltung der virtuellen Maschinen, ist zentrale Anlaufstelle für virtualisierte Ressourcen und gewährleistet Zugriff auf deren Konsole. Xend protokolliert alle Ereignisse in einer Datei, die somit gut zur Fehlersuche und -behebung dienen kann. Er erhält die Befehle über eine HTTP-Schnittstelle von dem Kommandozeilen-Werkzeug xm.
In der Domain 0 sind zwei Treiber enthalten, Network Backend Driver und Block Backend Driver, um Netzwerk- und Festplattenanfragen der Gäste zu unterstützen, die direkt mit der Netzwerk-Hardware bzw. den Festplatten kommunizieren, um die Anfragen abzuwickeln [14].

Domain U

Eine Domain U beherbergt die Gäste eines Xen-Systems und wird daher auch als unprivilegiert bezeichnet. Eine solche Domain hat keinen direkten Kontakt zur Hardware und kommuniziert nur über die Schnittstelle des Hypervisors. Die Gäste werden für den Betrieb in einer Domain U angepasst, damit sie mit dieser Schnittstelle arbeiten können. Ihre Anwendungen verwaltet sie selber, was auch das CPU-Scheduling innerhalb der virtuellen Maschine betrifft. Das bedeutet, wenn eine Domain den Prozessor vom Hypervisor zugeteilt bekommen hat, entscheidet sie selbst, für welche Prozesse sie ihn benutzt. Eine paravirtualisierte Maschine wird auch Domain U PV genannt. Eine Domain U PV enthält die zwei Treiber PV Network Driver und PV Block Driver für den Netzwerk- bzw. Festplattenzugriff [14].

1.5 Betriebssystem-Virtualisierung

1.5.1 Funktionsweise

Die Virtualisierung auf Betriebssystemebene geht einen etwas anderen Weg als die beiden zuvor vorgestellten Methoden. Hierbei werden keine vollständigen virtuellen Maschinen verwendet, sondern voneinander getrennte Laufzeitumgebungen zur Verfügung gestellt. Diese werden meist Container genannt, manchmal auch Jails oder Systempartitionen, wobei letztgenannter Begriff nicht mit der Partitionierung eines Datenträgers zu verwechseln ist. In den Containern können keine separaten Kernel gestartet werden. Sie dienen der Trennung der verschiedenen Umgebungen voneinander [9]. Auf dem physischen Rechner läuft der Host-Kernel als einziger, seine Bibliotheken werden von den Gästen mit genutzt. Die Virtualisierungslösung isoliert die Container voneinander, sodass Prozesse eines Containers auch nur in diesem agieren können. Somit muss sie in der Lage sein, Befehle aus einem Container zu beschränken und muss gleichzeitig den virtualisierten Prozessen alle normalen Schnittstellen des Host-Betriebssystems bereitstellen.

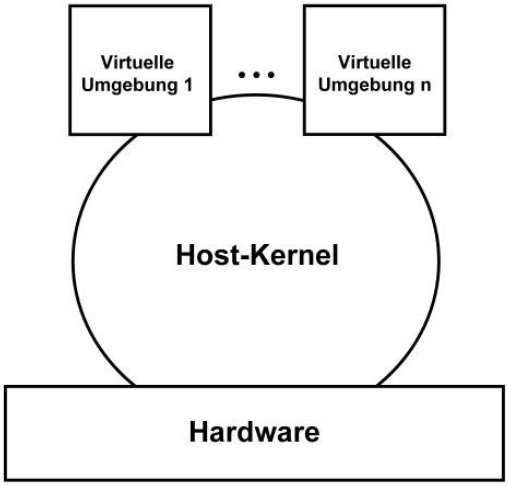

Abbildung 1.4: Prinzip der Betriebssystem-Virtualisierung.

Da auf dem gesamten System nur ein Kernel läuft, ist man bei den virtualisierten Anwendungen auf dasselbe Betriebssystem angewiesen. Allerdings wird meist innerhalb der Container ein gesamtes Dateisystem abgebildet, sodass es möglich ist, verschiedene Distributionen desselben Betriebssystems zu verwenden. Nutzen beispielsweise zwei verschiedene Linux-Distributionen denselben Kernel, können sie bei der Betriebssystem-Virtualisierung parallel in je einem Container installiert werden.
Die Betriebssystem-Virtualisierung kann sehr nützlich sein, wenn man nicht darauf angewiesen ist, zwei oder mehrere verschiedene komplette Systeme zu nutzen, sondern nur Anwendungen oder Dienste voreinander schützen möchte; zum Beispiel können ein Webserver und ein Mailserver sehr gut auf einem System laufen. Bei der Kompromittierung einer der beiden ist der andere allerdings ebenfalls in Gefahr. Laufen die beiden Server aber in je einem Container, so besteht diese Gefahr nicht.
Diese Methode der Virtualisierung hat noch weitere Vorteile. Zum einen wird der Overhead gering gehalten, da nicht jedes virtuelle System einen eigenen Kernel hat, woraus sich ein Geschwindigkeitsvorteil ergibt. Man spricht von einem Virtualisierungsschwund von nur einem bis drei Prozent gegenüber dem Host. Des Weiteren ist diese Art recht flexibel im Ressourcen-Management, da alle Prozesse innerhalb desselben Kernels laufen und es demzufolge nicht notwendig ist Ressourcen fest zu binden, sondern sie je nach Auslastung zu verteilen.
Beispiele für Software-Produkte, die eine Betriebssystem-Virtualisierung ermöglichen sind OpenVZ, Parallels Virtuozzo Container, BSD-Jails oder Solaris Zones. OpenVZ und Parallels Virtuozzo Container werden in den Abschnitten 1.5.2 und 1.5.3 vorgestellt.

Vorteile der Betriebssystem-Virtualisierung

- Overhead der vollständigen Virtualisierung wird vermieden, so ergibt sich eine hohe Geschwindigkeit des Gastes.
- Es sind nur geringe Systemkapazitäten für jeden Gast notwendig, so sind bei Bedarf viele Container nebeneinander möglich.
- Hohe Flexibilität bei der Verteilung von Systemressourcen an die Container.

Nachteile der Betriebssystem-Virtualisierung

- Keine Möglichkeit andere Betriebssysteme als das des Hosts zu verwenden.
- Änderungen im Host-Betriebssystem betreffen ebenso den Gast.

1.5.2 Beispiel OpenVZ

OpenVZ ist eine freie Software, die auf einem Linux-System mehrere voneinander getrennte Umgebungen erstellt und verwaltet. Bei OpenVZ nennen sich diese Container auch Virtual Environments, oder kurz VE. OpenVZ ist im Kern die Basis von Virtuozzo Containers (Kapitel 1.5.3), welches kommerziell von der Firma Parallels vertrieben wird. Drei Komponenten bilden den Kern von OpenVZ: ein für OpenVZ angepasster Linux-Kernel, einige Programme und Werkzeuge für OpenVZ und schließlich die Installations-Templates zur Erstellung von neuen Containern.

Kernel

Das Host-System muss mit einem Kernel mit OpenVZ-Unterstützung gestartet werden. OpenVZ stellt komplette OpenVZ-kompatible Kernel, aber auch Kernel-Patches zum Anpassen des vorhandenen eigenen Kernels, zum Download zur Verfügung. Die Erweiterung ermöglicht mehrere virtuelle Umgebungen in einem einzigen Kernel, ein Ressourcen-Managements für Container und das sogenannte Checkpointing, eine Funktion, um einen Container im Betrieb zu pausieren, dessen Zustand zu speichern und später fortzusetzen. Um die Isolation der Container zu gewährleisten, hat jede Umgebung ihre eigenen Ressourcen, die durch den Kernel zugeteilt werden, sowie ein eigenes Datei-System, eigene Benutzer- und Gruppenverwaltung, eigene Prozesse (nur Prozesse der VE sind sichtbar) und eine eigene Netzwerkverwaltung.
Das Ressourcen-Management besteht wiederum aus drei Komponenten:

- Einer Festplattenspeicher-Verwaltung in zwei Stufen: In der ersten Stufe kann der OpenVZ-Server-Administrator Speicherobergrenzen für jeden Container festlegen. Die zweite Stufe ist die Limitierung des Speichers für Benutzer und Gruppen, die der normalen Quota eines Linux-Systems entspricht.

- Ein ebenfalls zweistufiger CPU-Scheduler, der in der ersten Stufe entscheidet, welcher Container den Prozessor zu einem bestimmten Zeitpunkt zugeteilt bekommt und in der zweiten Stufe, welcher Prozess innerhalb des Containers ausgeführt wird.
- Des Weiteren die sogenannten User Beancounters, welche zum einen Obergrenzen, aber auch Garantien für die Nutzung von Systemressourcen darstellen, sodass ein einzelner Container nicht eine Ressource alleine blockieren kann, ihm auf der anderen Seite aber auch eine gewisse Zeit der Nutzung zugesichert wird. Außerdem lässt sich die Auslastung der jeweiligen Ressource auslesen, um die Counter ggf. anzupassen. Zu diesen Ressourcen zählt zum Beispiel der Arbeitsspeicher.

Mit dem Checkpointing ist eine Migration eines im Betrieb befindlichen Containers zu einem beispielsweise anderen Server möglich. Allerdings befindet sich das System dabei in einem pausierten Zustand und ist komplett in einer Datei gespeichert. Die Datei wird zum Zielsystem überführt, wo es dann aus dem pausierten Zustand heraus fortgesetzt werden kann. Dieser Ablauf kann innerhalb von wenigen Sekunden abgeschlossen sein, sodass für einen eventuell existierenden Client nur eine minimale Verzögerung spürbar ist.

Werkzeuge und Templates

OpenVZ bringt ein Werkzeug namens vzctl mit sich, ein Kommandozeilen-Programm zur Erstellung und Verwaltung der Container. Bei der Erstellung sind die Templates eine große Hilfe. Sie bezeichnen ein Set von Paketen; ein Template-Cache meint ein gepacktes Archiv einer chroot-Umgebung, also im Grunde eine frische Installation eines Containers nach Vorgabe eines spezifischen Templates. Mithilfe des Kommandos `vzctl create` wird das Archiv entpackt und ein neuer Container steht zur Verfügung. Aufgrund dieser Art des Cachens ist die Installation eines neuen Containers in sehr kurzer Zeit abgewickelt. Es existieren weitere Werkzeuge, zum Beispiel zur Erstellung solcher Template-Caches.

1.5.3 Beispiel Parallels Virtuozzo Containers

Virtuozzo Containers enthält als Basis die Kernkomponenten von OpenVZ, was darauf zurückzuführen ist, dass Virtuozzo den Linux-Kernel modifiziert, welcher unter der GNU Generel Public License steht, sodass die Änderungen wiederum unter selbiger Lizenz veröffentlicht werden müssen. Das ist durch OpenVZ (auch OpenVirtuozzo; siehe Kapitel 1.5.2), das diese Komponenten enthält, geschehen. Im Unterschied zu OpenVZ wird Virtuozzo von der Firma Parallels kommerziell angeboten, zum einen in der besagten Linux-Version, zudem aber auch in einer Version für Windows. Diese Software ist damit die einzige, die diese Art der Virtualisierung in Windows ermöglicht [7].
Die Anwendungsgebiete ähneln zunächst einmal denen von OpenVZ. Allerdings ist das Produkt für größere Virtualisierungsumgebungen gedacht, was sich in Anzahl und Art der Verwaltungsmöglichkeiten äußert. Container werden bei Virtuozzo auch virtuelle Server genannt; die physikalischen Server, auf denen virtuelle Server laufen, nennen sich Hardware-Nodes. Auf jedem vorhandenen Hardware-Node wird von Virtuozzo zunächst ein sogenannter Service-Container eingerichtet, ein spezieller virtueller Server, auf dem

einige Werkzeuge zur Administration der normalen virtuellen Server zu finden sind. Außerdem bietet der Service-Container Schnittstellen, die anderen Anwendungen Zugriff auf ihn und damit auf die Container des Systems erlauben. Diese Schnittstellen sind unter Linux und Windows gleich, sodass dieselben Programme darauf zugreifen können.

Werkzeuge

Zu den Verwaltungsprogrammen im Service-Container zählt Parallels Virtual Automation (früher Infrastructure Manager (PIM)), eine Web-basierte zentralisierte Verwaltung eines oder mehrerer Container. Es erstellt Übersichten zur Ressourcen-Überwachung, bietet Echtzeit-Statistiken und kann zum Beispiel Container mit hoher Ressourcen-Auslastung herausfiltern, um darauf reagieren zu können. Es ist ebenfalls möglich einen Container auf einen anderen Virtuozzo-Server zu migrieren. Der Service-Container bietet ein weiteres Web-Interface mit dem Namen Parallels Power Panel (PPP). Mit ihm kann der Administrator des Virtuozzo-Servers den Zugang zu einzelnen Containern für bestimmte Nutzer gewähren. So können Container-Administratoren ihre virtuellen Server mit allen Rechten verwalten, ohne Gefahr zu laufen, die anderen virtuellen Server zu beschädigen oder zu beeinflussen. Mit Power Panel kann ein Container gestartet oder gestoppt werden - eine Sicherung, Wiederherstellung, Neuinstallation oder Reparatur im Reparatur-Modus ist möglich und es kann Einfluss auf die Dienste und Prozesse genommen werden. Außerdem bietet das Panel natürlich eine Ressourcenüberwachung.
Auf jedem Hardware-Node liegt eine lokale Virtuozzo-Installation vor, welche die Daten verwaltet und die Container zur Ausführung bringt. In einer Virtuozzo-Umgebung mit mehreren Hardware-Nodes ist es sinnvoll dedizierte Backup-Nodes zu bestimmen, auf den nur Sicherungen von Containern liegen. Werden keine Container ausgeführt, ist für diesen Server auch keine Lizenz für Virtuozzo nötig. Natürlich ist es auch auf einem Hardware-Node möglich, Backups zu lagern [7]. Die gesamte Virtuozzo-Umgebung wird mit dem Verwaltungswerkzeug Parallels Management Console (PMC) verwaltet. Das Programm ist grafikbasiert und kann auch auf einem normalen Arbeitsplatz-Rechner installiert sein. Es verbindet sich anhand der jeweiligen Service-Container-Schnittstellen mit jedem Virtuozzo-Server und hat damit Zugriff auf die gesamte Umgebung. Es bietet diverse Assistenten zur Erstellung von neuen Containern, die Konfiguration von einzelnen, mehreren oder allen Containern und die Überwachung des Systems anhand von Systeminformationen, Ereignisprotokollen oder automatisierten Warnungen bei Überschreitung von Grenzwerten. Es kann ebenfalls Container zwischen Systemen migrieren und es ist möglich die Firewall des Hardware-Nodes oder der Container zu konfigurieren.

Templates

Virtuozzo-Templates enthalten fertige Software- oder ganze Systeminstallationen, die auf jedem Hardware-Node einzeln verwaltet werden können. Soll ein Template als Vorlage einer neuen Container-Installation dienen, so werden einfach nur entsprechende Verknüpfungen von den Template- auf die Container-Daten gesetzt. Für den Container sind dies ganz normal existente Dateien. Auf dem gesamten System liegen diese Daten somit nur einmal vor, was Speicherplatz spart. Außerdem können gemeinsam genutzte Daten vom

Host-System auch zusammen gecacht werden, was wiederum einen Geschwindigkeitsvorteil bringt. Möchte ein Container Änderungen an solchen verknüpften Daten vornehmen, ersetzt Virtuozzo die Verknüpfung in der virtuellen Umgebung durch eine tatsächlich vorhandene Datei. Änderungen werden erst dann vorgenommen. Das Template selbst und die damit verknüpften Container bleiben somit unabhängig voneinander. Durch dieses Template-System können Anwendungen oder aber auch ganze Container-Installationen in kurzer Zeit bereitgestellt werden, da die eigentliche Installation bereits auf dem System vorhanden ist und nur die entsprechenden Verknüpfungen gesetzt werden müssen.

1.6 Prozessorunterstützte Virtualisierung

1.6.1 Funktionsweise

Eine weitere Methode der Virtualisierung hat sich in den letzten Jahren mit der Einführung einer neuen Erweiterung für Prozessoren entwickelt. Bei der prozessorunterstützten Virtualisierung wird ein Teil der Aufgaben des Hypervisors in Hardware implementiert und in den Prozessor integriert. Intel stellte diese Erweiterung 2003 erstmals vor [23] und integrierte sie seit 2005 in viele ihrer Prozessoren. Bei AMD ist diese Technologie seit 2006 Bestandteil fast aller ihrer CPUs.

Angemerkt sei bereits hier, dass sich die Technologien der beiden Firmen zwar ähneln, aber nicht kompatibel sind. Details zu den Unterschieden sind im folgenden Abschnitt (1.6.2) zu finden.
Hauptmerkmal der Prozessoren mit den Erweiterungen IVT (Intel Virtualization Technology) bzw. AMD-V (AMD Virtualization) ist, dass hier Betriebssysteme unmodifiziert virtualisiert werden können. Beide Firmen setzen auf eine Erweiterung der Ringe (siehe Abschnitt 1.2.2), sodass die Gäste in ihrer gewohnten Umgebung, nämlich Ring 0, laufen können [2]. Binary Translation ist nicht mehr notwendig und die Implementierung des VMM damit weniger komplex.

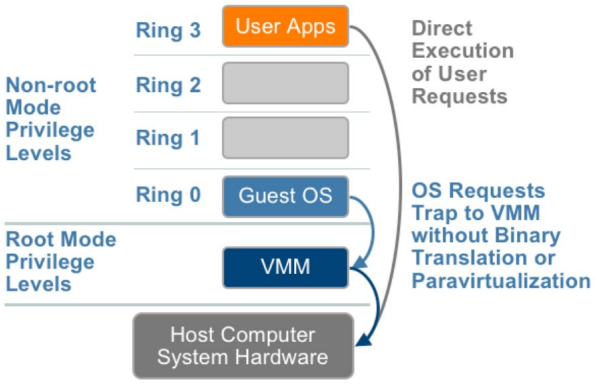

Abbildung 1.5: Prinzip der hardwarebasierten Virtualisierung [1].

Vorteile der prozessorunterstützten Virtualisierung

- Anpassung der Gast-Betriebssysteme durch Patches o.ä. ist nicht mehr notwendig, sodass auch auf proprietäre Systeme (bspw. Microsoft Windows) zurückgegriffen werden kann.
- Architektur der Virtualisierungslösung ist schlanker, da wesentliche Aufgaben direkt von der Hardware übernommen werden.

- Geringerer Overhead als bei der vollständigen Virtualisierung. Man spricht von Leistungseinbußen zwischen 10-15 Prozent [2, Seite 150].

Nachteile der prozessorunterstützten Virtualisierung

- Unterstützung durch die Hardware ist Voraussetzung. Gegebenenfalls muss neue Hardware erworben werden.
- Virtualisierungslösungen müssen sowohl die Intel- als auch die AMD-Lösung unterstützen.

Es sei noch angemerkt, dass wenn von einer hardwarebasierten Virtualisierung gesprochen wird, oft die prozessorgestützte gemeint ist. Es lassen sich jedoch auch andere Teile der Hardware virtualisieren, sodass die Begriffe durchaus differenziert verwendet werden sollten.

1.6.2 Aktuelle Hardwaretechnologien

Sowohl bei der Intel VT- also auch der AMD-V-Erweiterung kann einer virtuellen Maschine die CPU direkt zugeteilt werden, was als VM Entry (Intel) bzw. VMRUN (AMD) bezeichnet wird. Ein Entzug der CPU wird entsprechend VM Exit bzw. #VMEXIT genannt [2].

Intel

Intel VT ist in zwei Varianten aufgeteilt, VT-i, welches die Erweiterung für die 64-Bit-Architektur Intel Itanium (IA-64) meint, und VT-x, welches die Erweiterung für die x86-Architektur bezeichnet. Oftmals wird für Intel VT auch der Entwicklungsname Vanderpool verwendet.
Die Intel-Technologie bringt einen erweiterten Befehlssatz für den Prozessor mit sich, der als Virtual Machine Extensions (VMX) bezeichnet wird. Dabei gibt es zwei Modi, den Root- und den Non-Root-Modus. Im Root-Modus arbeitet der Hypervisor. Dieser Modus entspricht im Grunde der Arbeitsweise eines Prozessors ohne VMX-Unterstützung. Die virtuellen Maschinen laufen im Non-Root-Modus. Das Gast-Betriebssystem hat keine Möglichkeit den Non-Root-Modus zu erkennen.
Die erwähnten VM Entries und VM Exits werden bei Intel von einer Datenstruktur namens virtual-machine control structure (VMCS) verwaltet. Diese Struktur beinhaltet unter anderem die Gruppen „guest-state area“ und „host-state area“, welche jeweils in mehrere Felder unterteilt sind. Bei VM Entries wird der Prozessor-Zustand aus der guest-state area geladen. Bei einem VM Exit wird der Prozessor-Zustand zunächst in der guest-state area gespeichert und dann der in der host-state-area gespeicherte Zustand in den Prozessor geladen. In den Feldern einer weiteren Gruppe, VM-Execution Control Fields, können Befehle definiert werden, die zu einem VM Exit führen sollen. Es existieren weitere Felder, um beispielsweise den Grund für einen VM Exit zu speichern. Das Format und Layout

der VMCS im Speicher ist durch die Architektur aber nicht in allen Details festgelegt, sodass spezifische Anpassungen zur Optimierung einzelner Virtualisierungslösungen möglich sind.
Weitere Details der VMCS und Intel VT können in [5] nachgelesen werden.

AMD

Die entsprechende Erweiterung bei AMD nennt sich einfach AMD Virtualization oder kurz AMD-V. Geläufig ist auch der Name Pacifica, der AMD-V während der Entwicklung bezeichnete, der aber eigentlich seit 2006 abgelegt wurde.
Der erweiternde Befehlssatz besteht hier aus den SVM-Operationen, wobei SVM für Secure Virtual Machine steht. Der Hypervisor läuft im normalen Modus, virtuelle Maschinen in einem neu eingeführten Gast-Modus. Auch hier können die Gast-Systeme diesen Modus nicht erkennen.
Das Pendant zur VMCS bei Intel ist bei AMD der Virtual Machine Control Block (VMCB). In ihm ist eine Liste der Instruktionen enthalten, die zu einem VM Exit führen. Des Weiteren enthält er verschiedene Kontroll-Bits für die Gast-Umgebung, sowie einen Bereich für den Zustand des Prozessors bei Ausführung eines Gastes.
Unterschiede zwischen der Intel- und der AMD-Lösung werden bei der Speicherverwaltung deutlich. AMDs Prozessoren besitzen einen integrierten Memory-Controller, der ebenfalls virtualisiert wird. Während Vanderpool-CPUs die Adressübersetzung des Speichers der virtuellen Maschine in physikalische Adressen softwareseitig durchführen müssen, stellt der sogenannte Nested-Paging-Modus von AMD-V jeder virtuellen Maschine ein eigenes virtualisiertes Register zur Verfügung, das die Adresse der Seitentabelle enthält. Damit sind die Adressübersetzungen Aufgaben der Hardware und somit deutlich effizienter; ein zusätzlicher Vorteil besteht darin, dass einige vorher nötigen Kontextwechsel entfallen.
Ein weiterer Unterschied zu Intel ist, dass AMD-V den Real-Mode virtualisieren kann. Sollen entsprechende Anwendungen in der virtuellen Maschine laufen, muss der Modus nicht emuliert werden, wie es bei Intel der Fall ist.
Weitere Details zu AMD-V sind zum Beispiel in [6] zu finden.

1.6.3 Beispiel Kernel-based Virtual Machine und QEMU

Die Kernel-based Virtual Machine (KVM) gibt es seit 2006 und ist seit der Linux-Kernel-Version 2.6.20 in ihn integriert. Sie wird von dem israelischen Unternehmen Qumranet entwickelt, das von der Firma Red Hat übernommen wurde. Seit der Veröffentlichung wächst ihre Verbreitung stetig, was auf die konsequente Anwendung der hardwarebasierten Virtualisierungstechnik und die Unterstützung von libvirt zurückzuführen ist [17]. Libvirt ist eine Bibliothek zur Steuerung mehrerer heterogener Virtualisierungslösungen. Neben der Unterstützung für Intels und AMDs Prozessor-Technologie werden auch Nested Paging [18] und I/O Memory Mapping Units unterstützt. Es gibt zwar auch einen Patch mit dem man mit KVM paravirtualisierte Gäste beherbergen kann, dieser ist allerdings nicht in den Kernel integriert. KVM besteht aus den Kernel-Modulen kvm.ko und kvm-intel.ko bzw.

kvm-amd.ko, sowie der Gerätedatei /dev/kvm. Nach Laden der Module ist der Linux-Kernel selbst der Hypervisor. Ein weiteres privilegiertes System wie bei Xen (Abschnitt 1.4.2 bzw. 1.6.4) gibt es so nicht.

QEMU

KVM virtualisiert keine Hardware. Diese Aufgabe übernimmt QEMU. Ohne Erweiterung ist QEMU ein Emulator für diverse Architekturen, darunter die x86-, x86-64- oder PowerPC-Architektur [2]. Mit einem Zusatzmodul namens KQEMU beherrscht es auch die vollständige Virtualisierung, ist dann aber auf die x86-Architektur angewiesen. Die Entwicklung von KQEMU wurde jedoch eingestellt [17], um sich auf KVM zu konzentrieren. KVM und QEMU werden inzwischen zusammen entwickelt. Viele Linux-Distributionen bieten oft nur noch Installationspakete für beide gemeinsam an.

Funktionen

KVM bietet eine Live-Migration, um Maschinen auf einen anderen Server zu verschieben. Die Maschinen können dabei in Betrieb sein. Das Programm bietet außerdem die sogenannte „Nested Virtualization“, mit der man innerhalb einer virtuellen Maschine wiederum virtualisieren kann. So kann eine KVM in einer bereits durch KVM virtualisierten Umgebung gestartet werden. Mit dem KSM-Verfahren (Kernel Shared Memory) findet KVM Speicherbereiche verschiedener virtueller Maschinen mit identischem Inhalt und fasst diese zusammen. So kann den virtuellen Maschinen mehr virtueller Speicher angeboten werden, als physikalisch vorhanden.
Mit QEMU bringt ein Werkzeug namens qemu-img mit sich. Mit ihm lassen sich virtuelle Speicher als Image-Datei erstellen. Auch das Vergrößern, Verschlüsseln und Komprimieren von Images lässt sich mit qemu-img bewerkstelligen. Snapshots werden ebenfalls mit dem Kommandozeilen-Programm gemacht. Teilweise können QEMU-Images mittels qemu-img auch in die Formate anderer Virtualisierungslösungen konvertiert oder zumindest dafür vorbereitet werden. Eine Kernel-based Virtual Machine kann nach etwas Konvertierungsarbeit zum Beispiel unter VirtualBox, Xen oder dem VMware Player genutzt werden. Andersherum unterstützt KVM die Formate anderer Lösungen oft direkt, zum Beispiel vdi-Images von VirtualBox oder vmdk-Images von VMware [17].

1.6.4 Beispiel Xen

Die Software Xen wurde bereits in Abschnitt 1.4.2 in Hinblick auf Paravirtualisierung vorgestellt. Daher wird hier nur kurz auf die Unterschiede eingegangen, die bei einem hardwareunterstützten Virtualisierungsbetrieb auftreten.
Ein Nachteil der verschiedenen Wege von Intel und AMD ist es, dass entsprechende Software beide Technologien unterstützen muss. Bei Xen nennt sich diese Unterstützung Hardware Virtual Machine (HVM). Die Domänen heißen hier entsprechend Domain U HVM. Sie haben keine Kenntnis darüber, dass sie überhaupt virtualisiert laufen und sich noch

andere Gäste auf dem System befinden (können). Bei der Erstellung von HVM-Gästen müssen Änderungen an Konfigurationsdateien vorgenommen werden und diese als Parameter an das entsprechende xm-Kommando zur Einrichtung einer HVM übergeben werden [15].
Des Weiteren enthält ein HVM-Gast die erwähnten PV Treiber (vgl. Kapitel 1.4.2) nicht. Stattdessen läuft für jede einzelne Domain U HVM in der Domain 0 ein Daemon namens Qemu-dm. Dieser wickelt dann alle Anfragen an Netzwerk oder Datenspeicher ab.
Das Gast-Betriebssystem erwartet einen normalen Start der Maschine, der aber mangels echter Hardware in dieser Form nicht stattfindet. Somit wird in die Gäste eine weitere Software integriert, Xen Virtual Firmware, die das BIOS einer normalen Maschine simuliert. Sie stellt sicher, dass der Gast alle nötigen Start-Instruktionen erhält, die er erwartet [14].

1.7 Weitere Verfahren

1.7.1 Emulation

Die Emulation unterscheidet sich von den bisher vorgestellten Methoden vor allem in der Hinsicht, dass sie vollkommen unabhängig von der Hardware ist. Alle Komponenten der virtuellen Maschine werden durch Software dargestellt. Für CPU-Befehle bedeutet das, das der Emulator sie interpretiert, sie nach seiner Logik ausführt und das Ergebnis zurück gibt. Dieses muss demjenigen Ergebnis entsprechen, das bei einer Ausführung auf einer realen CPU ermittelt worden wäre.
Die Vor- und Nachteile liegen auf der Hand. Bei den anderen Verfahren werden viele Operationen direkt an die reale Hardware weitergegeben, sodass Host- und Gastarchitektur identisch sein müssen. Aufgrund der vollkommenen Unabhängigkeit von der Hardware bei der Emulation, lassen sich auch andere Architekturen durch den Emulator darstellen, als die physisch vorhandene. Emulatoren werden zum Beispiel eingesetzt, wenn das Zielsystem nicht (mehr) verfügbar ist, weil es zum Beispiel noch nicht entwickelt ist oder nicht mehr entwickelt wird, oder weil die Beschaffung zu teuer wäre.
Da die Befehle, wie oben bereits erwähnt, nicht auf der realen CPU ausgeführt werden, ergibt sich dadurch auf der anderen Seite ein mitunter erheblicher Geschwindigkeitsverlust der virtuellen Maschine. Der Overhead entsteht dadurch, dass das Entgegennehmen, Interpretieren, Ausführen und Zurückgeben der Instruktionen durch den Emulator komplett softwareseitig durchgeführt wird.

Bochs

Bochs ist ein Open-Source-Emulator für die x86-Architektur, der seit 1994 entwickelt wird. Zunächst kostenpflichtig, wurde Bochs im Jahr 2000 von MandrakeSoft gekauft, welche den Quellcode unter Open-Source-Lizenz veröffentlichte.
Die Software simuliert einen kompletten x86-Rechner vom Booten des Systems bis zum Herunterfahren, wobei jeder einzelne Befehl softwareseitig simuliert wird. So ist es möglich

eine große Anzahl verschiedener Betriebssysteme ohne Manipulation mit Bochs zu betreiben. Die Software ist für die meisten populären Betriebssysteme verfügbar, ebenfalls für die anderer Architekturen. So ist es mit Hilfe von Bochs beispielsweise möglich auf einer PowerPC-Architektur mit Mac OS eine Windows-Maschine zu emulieren.
Neben dem Ausführen von x86-Anwendungen auf Nicht-x86-Rechnern gibt es eine weitere oft genutzte Einsatzmöglichkeit für Bochs. Jeder Befehl, der über Bochs ausgeführt wird, ist einsehbar. So kann man in Bochs ausgeführte Betriebssysteme und Anwendungen sehr bequem debuggen. Die Simulation kann dabei in verschiedenen Geschwindigkeiten ausgeführt oder gar pausiert werden, um zum Beispiel die aktuellen Inhalte der CPU-Register einzusehen.

1.7.2 API-Emulation

Grundsätzlich können verschiedene Betriebssysteme auf derselben Architektur lauffähig sein und damit natürlich auch deren Anwendungen. Der Grund dafür, dass die Anwendungen eines speziellen Betriebssystems nicht auf einem anderen Betriebssystem laufen sind die spezifischen Schnittstellen der Systeme. Als Beispiel sei genannt, dass man durchaus auf einem Rechner ein Windows-System oder aber ein Linux-System einrichten könnte, aber keine Windows-Anwendungen auf einem Linux-System installieren kann und andersrum.
In solch einem Fall muss es nicht notwendig sein eine komplette virtuelle Maschine zu emulieren, da die Anwendung auf der Architektur selbst prinzipiell laufen kann und da man diese Anwendung ins bestehende System integrieren will und eine Isolation durch eine VM somit nicht nötig oder gar gewünscht ist. Es reicht also, die betriebssystemspezifischen Schnittstellen zu emulieren. Dies ist die Aufgabe von sogenannten API-Emulatoren. Wine ist eines der bekanntesten Produkte dieser Art und wird im Folgenden vorgestellt.

Wine

Wine ist wohl der bekannteste API-Emulator. 1993 wurde das Projekt ins Leben gerufen. Sun Microsystems hatte zuvor die Software Wabi vorgestellt, welche es ermöglichte, Windows-Programme auf Solaris laufen zu lassen. Die Chancen für eine Portierung auf Linux waren aber gering, sodass sich eine Gruppe zusammen fand, Wine zu entwickeln [27]. Die aktuelle stabile Version ist 1.0.1. Dass Wine keine Emulator-Schicht bereitstellt und damit kein komplettes Windows-System emuliert, drückt sich bereits im Namen aus: Wine steht für Wine Is Not an Emulator. Vielmehr dient es dazu, Windows-Programme unter POSIX-kompatiblen [24] Betriebssystemen lauffähig zu machen. Dazu stellt es die Windows-spezifischen Schnittstellen zur Verfügung. Dabei übersetzt es Instruktionen so, dass das aktuelle Betriebssystem sie normal verarbeiten kann [25].
Die nächste Version von Wine wird die Version 1.2 in den nächsten Monaten sein. Das Hauptziel soll die Unterstützung von 64-Bit-Programmen sein. Unter [26] lassen sich weitere Ziele nachlesen.

1.8 Vergleich der Verfahren

Wie bereits in der Einführung erwähnt , gestaltet sich ein direkter Vergleich der Verfahren schwierig. Jedes Konzept hat entsprechend der zugehörigen Vor- und Nachteile sein eigenes Anwendungsgebiet. In der Tabelle 1.1 wird die Performanz bewertet, sowie die eventuelle Notwendigkeiten zur Modifikation der Gast-Betriebssysteme und die Abhängigkeit von spezieller Hardware.

Konzept	**Performanz**	**Gast-Modifikation**	**spezielle Hardware**	**Besonderheiten**
vollständige Virtualisierung	gering	nein	nein	erste Virtualisierungsmöglichkeit für x86
Para-virtualisierung	gut	notwendig	nein	Zusammenwirken von Gast mit Virtualisierungslösung
Betriebssystem-Virtualisierung	sehr gut	nein	nein	Isolation von Umgebungen, ohne eigenes Betriebssystem
Prozessor-unterstützte Virtualisierung	mittel	nein	notwendig	Teile der Hardware übernehmen Aufgaben der Virtualisierung
Emulation	schlecht	nein	nein	Unabhängigkeit von Hardware(-Architektur)
Konzept	**Performanz**	**Gast-Modifikation**	**spezielle Hardware**	**Besonderheiten**

Tabelle 1.1: Vergleich der verschiedenen Konzepte der Virtualisierung

1.9 Vergleich der Produkte

Neben einem Vergleich der einzelnen Konzepte in Kapitel 1.8 folgt hier nun eine knappe Übersicht der populärsten Software-Produkte für Virtualisierung. Dabei sind auch Produkte erwähnt, die im bisherigen Teil noch keine Erwähnung gefunden haben. Doch selbst diese Auswahl sollte nicht als vollständig betrachtet werden, da es noch eine Vielzahl weiterer Virtualisierungsprodukte gibt. Auch die Informationen sind mit Vorsicht zu genießen, da bei den meisten Produkten die Entwicklung stetig voranschreitet. Auf eine Angabe der jeweilig aktuellen Version und deren spezifischen Funktionen wurde aus diesem Grund bewusst verzichtet.

Produkt	Technik	Architektur	Lizenz	Host-BS	Gast-BS
VMware Server, Workstation und Player	vollst. Virtualisierung (mit BT) / HW-basierte Virtualisierung	x86, x86_64	proprietär	Linux, Windows	Linux, Windows, BSD und weitere
Xen	Paravirtualisierung / HW-basierte Virtualisierung	x86, x86_64, IA64, PPC, ARM	GPL	Linux, NetBSD	Linux, Windows, BSD, Solaris
OpenVZ	BS-Virtualisierung	x86, x86_64, IA64, PPC, SPARC, ARM	GPL	Linux	wie Host
Virtuozzo	BS-Virtualisierung	x86, x86_64, IA64	proprietär	Linux, Windows	wie Host
Bochs	Emulation	diverse	GPL	Linux, Windows, BSD, OS/2, BeOS	Linux, DOS, Windows und weitere
VirtualBox	vollst. Virtualisierung / HW-basierte Virtualisierung	x86, x86_64	proprietär, Teile GPL	Linux, Windows, Macintosh, OpenSolaris	Linux, Windows, Solaris, Mac OS X Server, BSD (teilweise), OS/2
KVM	HW-basierte Virtualisierung / Paravirtualisierung (mit Patch)	x86, x86_64 und Portierungen	GPL	Linux	Linux, Windows, BSD, Solaris, FreeDOS, ReactOS
Windows Virtual PC	HW-basierte Virtualisierung	x86, x86_64	proprietär	Windows 7	Windows XP, Vista, 7
Produkt	**Technik**	**Architektur**	**Lizenz**	**Host-BS**	**Gast-BS**

Tabelle 1.2: Vergleich verschiedener Software-Produkte der Virtualisierung

1.10 Zusammenfassung und Ausblick

Wie aus den einzelnen Abschnitten dieses Kapitels hervorgeht, haben alle der vorgestellten Virtualisierungslösungen Stärken und Schwächen. Die Vor- und Nachteile liegen dabei

teilweise in unterschiedlichen Richtungen, sodass sich jeder Nutzer individuell seine Lösung zusammenstellen muss, eine Patentlösung existiert dabei nicht. Dabei ist aber zu sagen, dass im Bereich der x86-Architektur, auf die sich das Feld der Virtualisierung konzentriert, die hardwarebasierte Virtualisierung an Bedeutung gewinnt. Die Nachteile des Performance-Schwunds werden von leistungsstarken Prozessoren aufgefangen und die Abhängigkeit von spezieller Hardware spielt eine immer kleinere Rolle, da die Technologie immer größere Verbreitung findet. Dafür gewinnt man eine sehr viel größere Freiheit bei der Auswahl des Gast-Betriebssystems. Außerdem wird diese Technik durch weitere Technologien, wie dem Nested Paging, immer besser unterstützt werden. Verdrängen wird sie die anderen Konzepte aber nicht. Möchte man ohnehin ein Open-Source-Betriebssystem als Gast einsetzen kann die Paravirtualisierung aufgrund des Geschwindigkeitsvorteils eventuell die bessere Wahl sein. Möchte man nur Programme oder Dienste auf dem selben Betriebssystem voneinander abschotten, so wäre das Erstellen einer ganzen virtuellen Maschine je Dienst völlig überdimensioniert. Möchte man gar Betriebssystem oder Programme einer anderen Architektur verwenden, führt kein Weg an einer Emulation vorbei. Einzig die vollständige Virtualisierung wird wohl an Bedeutung verlieren, da ihre Vorteile auch bei anderen Lösungen zu finden sind.
Nach der Entscheidung für ein passendes Konzept, folgt die Auswahl der Software. Auch bei dieser ist die Wahl mitunter nicht leicht, da sogar innerhalb eines Softwarehauses verschiedene Software-Produkte mit ähnlichen Funktionen angeboten werden. Hier spielen Merkmale wie Stabilität, Zuverlässigkeit oder Sicherheit eine Rolle, aber auch Faktoren unabhängig vom Produkt selbst, wie zum Beispiel die Verfügbarkeit und Bedienerfreundlichkeit von geeigneten Management-Werkzeugen zur Verwaltung von größeren Virtualisierungsprojekten können wichtig sein.
Letztendlich muss jeder Anwender der Virtualisierung seinen eigenen individuellen Entscheidungsprozess durchlaufen, um das für sich optimale Ergebnis mit passenden Konzepten und Produkten erreichen zu können.

Literaturverzeichnis

[1] VMware, Inc. *Understanding Full Virtualization, Paravirtualization, and Hardware Assist*, Palo Alto 2007.

[2] Fischer, Markus. *Xen - Das umfassende Handbuch*, Galileo Press, Bonn 2009.

[3] Goldberg, Robert P.. *Architectural Principles for Virtual Computer Systems*, Harvard University, 1973.

[4] VMware, Inc *Software and Hardware Techniques for x86 Virtualization*, Palo Alto 2009.

[5] Neiger, Gil; Santoni, Amy; Leung, Felix; Rodgers, Dion; Uhlig, Rich. *Intel Virtualization Technology: Hardware Support for Efficient Processor Virtualization*, Intel Corporation, 2006.

[6] Advanced Micro Devices, Inc. *AMD64 Virtualization Codenamed Pacifica Technology - Secure Virtual Machine Architecture Reference Manual*, 2005.

[7] Thorns, Fabian (Hrsg.). *Das Virtualisierungs-Buch*, Computer & Literatur Verlag GmbH, Böblingen 2008.

[8] Adams, Keith; Agesen, Ole. *A Comparison of Software and Hardware Techniques for x86 Virtualization*, San Jose, 2006.

[9] Kolyshkin, Kirill. *Virtualization in Linux*, http://download.openvz.org/doc/openvz-intro.pdf, 2006.

[10] Parallels. *Eine Einführung in Virtualisierung auf Betriebssystemebene und Parallels Virtuozzo Containers - White Paper Version 2.0*

[11] Parallels Holdings, Ltd.. *Parallels Virtuozzo Containers - Evaluation Guide*, Sunrise Valley Drive, 1999-2009.

[12] VMware, Inc.. *VMware Server User's Guide - VMware Server 2.0*, Palo Alto, 2008.

[13] Ahnert, Sven. *Virtuelle Maschinen mit VMware und Microsoft*, Addison-Wesley, München, 2009.

[14] Citrix Systems. *How Does Xen Work, Version 1.0*, http://www.xen.org/files/Marketing/HowDoesXenWork.pdf, December 2009.

[15] Clark, Christopher. *Xen - User's Manual*, http://bits.xensource.com/Xen/docs/user.pdf, 2002.

[16] Barham, Paul; Dragovic, Boris; Fraser, Keir; Hand, Steven; Harris, Tim; Ho, Alex; Neugebauer, Rolf; Pratt, Ian; Warfield, Andrew. *Xen and the Art of Virtualization*, University of Cambridge Computer Laboratory, Cambridge 2003.

[17] Warnke, Robert; Ritzau, Thomas. *qemu-kvm & libvirt*, Books on Demand GmbH, Norderstedt, 2010.

[18] Advanced Micro Devices, Inc. *AMD-V Nested Paging*, Juli 2008.

[19] Fruth, Florian E.J. *Grundbegriffe der Virtualisierung*, freies magazin 10/2009, Rutesheim, 2009.

[20] Sha, Amit. *Deep Virtue Kernel-based virtualization with KVM*, Januar 2008.

[21] Intel *http://www.intel.com/de_de/business/itcenter/topics/virtualization/index.htm*, Zugriff am 21.05.2010.

[22] AMD *http://sites.amd.com/us/business/it-solutions/virtualization/Pages/amd-v.aspx*, Zugriff am 21.05.2010.

[23] Otellini, Paul. *Transcript of the keynote speech in Intel Developer Forum from Fall 2003*, http://www.intel.com/pressroom/archive/speeches/otellini20030916.htm, Zugriff 30.05.2010.

[24] IEEE & Open Group. *POSIX.1-2008 bzw. IEEE Std 1003.1-2008*, http://www.opengroup.org/onlinepubs/9699919799/, Zugriff 31.05.20010.

[25] Wine Developers. *Wine User Guide*, http://ftp.winehq.org/pub/wine/docs/en/wineusr-guide.pdf.

[26] ScottRitchie. *Wine Release Plan*, http://wiki.winehq.org/WineReleaseCriteria, 2010.

[27] Wine Developers. *Wine History*, http://www.winehq.org/site/history, Zugriff 31.05.2010.

[28] Bochs Developers. *bochs: The Open Source IA-32 Emulation Project (Home Page)*, http://bochs.sourceforge.net, Zugriff 02.06.2010.

[29] Microsoft. *Windows Virtual PC*, http://www.microsoft.com/downloads/details.aspx?FamilyID=2b6d5c18-1441-47ea-8309-2545b08e11dd&DisplayLang=de, Zugriff 03.06.2010.

[30] Bochs Developers. *Introduction to Bochs*, http://bochs.sourceforge.net/doc/docbook/user/introduction.html, Zugriff 02.06.2010.

[31] VMware, Inc. *Virtualization system including a virtual machine monitor for a computer with a segmented architecture - Patent No.: 6397242*, 2002.

[32] Bochs Developers. *Who uses Bochs?*, http://bochs.sourceforge.net/doc/docbook/user/who-uses-bochs.html, Zugriff 02.06.2010.

[33] KVM Developers. *Guest Support Status*, http://www.linux-kvm.org/page/Guest_Support_Status, Zugriff 12.06.2010.

Kapitel 2

Angriffstechniken auf Server-Virtualisierung und deren Gefahrenpotential

Ingo Schwarz

Inhaltsverzeichnis

2.1 Einleitung

Virtualisierung spielt eine immer bedeutendere Rolle in der Informationstechnik. Dazu ist zu sagen, dass das Konzept nicht neu ist, denn schon in den 50er Jahren gab es beispielsweise virtuelle Speicher. Durch die mit Hilfe von Virtualisierung erreichte Steigerung von Effizienz, Flexibilität und Verlässlichkeit wird die Auslastung von IT-Ressourcen erhöht, die Qualität verbessert und das Management der Systeme vereinfacht. [1]

In der vorangegangenen Seminararbeit von Thomes Schletze wurde bereits der Sinn und Zweck von Virtualisierung erwähnt. In erster Linie werden im Zuge der Hardwarekonsolidierung Kosten für das jeweilige Unternehmen gesenkt, denn dank Virtualisierung von Servern können zahlreiche virtuelle Server auf wenigen physikalischen Rechnern installiert werden. Des Weiteren ist der Betreiber einer solchen virtuellen Serverfarm flexibler, wenn es darum geht neue Server aufzusetzen bzw. welche abzuschalten, da nun nicht mehr der lange Weg über eventuelle Neuanschaffung seitens der Hardware gegangen werden muss. Durch die Pflege von nun nur noch wenigen physikalischen Systemen und durch Vereinheitlichung der Virtualisierungssoftware wird der Wartungsaufwand verringert und die Verlässlichkeit und Verfügbarkeit erhöht.

Allerdings birgt die zunehmende Virtualisierung neue Sicherheitsrisiken in sich. So ist für eine virtuelle Umgebung ein maßgeschneidertes Sicherheitskonzept nötig. [2] Der Schutzbedarf konzentriert sich dabei, wie bei allen IT-Systemen, auf Vertraulichkeit, Integrität und Verfügbarkeit. In unserem Fall gibt es noch weitere Einflussfaktoren für die Sicherheit. Die Sicherheitskonzeption wird oft nicht in die wirtschaftliche Betrachtung einbezogen, weshalb sie in der Kostenbilanz nicht erfasst und ihr nicht viel Aufmerksamkeit geschenkt wird. Neben dem Risikofaktor Mensch gibt es noch zahlreiche technisch bedingte Schwachstellen. Zum Beispiel gilt es seitens der Hersteller von Virtualisierungssoftware ein intensives Patch-Management durchzuführen, da schnell auf aufgedeckte Sicherheitslücken reagiert werden muss. Schließlich kann eine Kompromittierung eines virtuellen Servers die Übernahme sämtlicher Server auf der betroffenen Maschine nach sich ziehen. Welche Angriffsszenarien es gibt und welche Sicherheitslücken bestehender Virtualisierungskonzepte ausgenutzt werden, soll in dieser Seminararbeit erörtert werden. Nach einer Einführung in verschiedene Formen der Virtualisierung sowie in Sicherheitsarchitekturen von Intel und Advanced Micro Devices (AMD) sollen mehrere Angriffstechniken detailliert dargelegt werden. Dabei soll unter anderem auf das Maß der Gefährdung sowie auf Aufklärungsmöglichkeiten und geeignete Gegenmaßnahmen eingegangen werden.

Zuerst gilt es jedoch ein paar Einsatzszenarien von Virtualisierung in der IT-Sicherheit vorzustellen. Laut Joanna Rutkowska gibt es drei grundlegende Ansätze, die Sicherheit von IT-Systemen zu steigern:[3]

security-by-correctness: Hinter diesem Ansatz steht die Annahme, dass man Software produzieren kann, welche weder Fehler noch Sicherheitslücken aufweist. In dem Fall gäbe es keine Sicherheitsprobleme, da jeder Code korrekt im Sinne der Implementierung, des

[1] http://www.contentmanager.de/magazin/artikel_1190_virtualisierung_bitkom.html

[2] http://www.zdnet.de/sicherheits_analysen_truegerische_sicherheit_virtualisierung_genuegt_nicht-_story-39001544-39159043-4.htm

[3] http://theinvisiblethings.blogspot.com/2008/09/three-approaches-to-computer-security.html

Designs und des Verhaltens zur Laufzeit wäre. Microsoft versucht mit dem "Secure Development Life-Cycle" security-by-correctness umzusetzen, indem man eine Vielzahl von Sicherheitsüberprüfungen parallel zur Softwareentwicklung durchführt. [4] Problematisch bei der Entwicklung korrekter Software ist, dass es an sicheren Programmiersprachen und deren Verbreitung und zuverlässigen Code-Verifizierern, sowie an Algorithmen mangelt, welche entscheiden, ob sich ein bestimmter Code in der späteren Einsatzumgebung richtig verhalten wird.

security-by-isolation: Wegen der Probleme, die beim erstgenannten Ansatz entstanden sind, ist die Idee aufgekommen, einen Computer in mehrere kleinere Teile aufzuteilen. Somit würde eine Kompromittierung eines Teilsystems die Funktionsfähigkeit des Gesamtsystems nicht beeinflussen. Ein Anwendungsbeispiel für diesen Ansatz der Schaffung von Sicherheit ist das Sandboxing-Verfahren. Diese wird im Abschnitt "Begriffe" näher erklärt.

security-by-obscurity: Dieser dritte Ansatz zielt auf Maßnahmen ab, die es einem Angreifer möglichst schwer machen, seine schadende Wirkung zu erreichen. Selbst wenn Sicherheitslücken vorhanden sind, ist es für den Eindringling sehr schwierig einen Server zu kompromittieren wenn dieser eine untypische Adressierung oder Pointer-Verschlüsselung benutzt. Allerdings wird so nicht verhindert, dass Fehler im System für Angriffe ausgenutzt werden, sondern der Angriff wird lediglich erschwert. Des Weiteren stellt sich die Frage, ob eine Verdunklung (obscurity) systemressourceneffizient funktioniert und ob nicht zu viel Systemleistung für die ständige Verschlüsselung aufgewendet wird. Letztlich kann man sich sicher sein, dass wenn man sein System dermaßen unkenntlich macht, die Herstellerunterstützung im Falle eines Crashs nutzlos wird.

2.2 Virtualisierung - Eine Abstraktion

Was genau ist Virtualisierung und welche Varianten gibt es? Diese, und die Frage, mit welchen Technologien die Chiphersteller Intel und AMD das Konzept der Virtualisierung unter dem Aspekt Sicherheit implementieren, soll im Folgenden erläutert werden.

Virtualisierung ist eine logische Abstraktion von physikalischer Hardware. Allgemein gesprochen, handelt es sich bei Virtualisierung um eine Trennung zwischen einer Dienstanfrage von der darunterliegenden physikalischen Vermittlung dieser Anfrage. Hierbei besteht eine gängige Realisierung darin, zwischen Anwendungsprogramm respektive Betriebssystem und Hardware eine Zwischenschicht, die sogenannte Virtualisierungsschicht (auch Hypervisor), einzufügen.[18] Das Resultat ist, dass die Ressourcen des Hostsystems zwischen mehreren Gastsystemen aufgeteilt werden können und somit eine simultane Nutzung dieser ermöglicht wird.[3]

[4] http://www.microsoft.com/security/sdl/about/process.aspx

2.2.1 Virtualisierungstechniken

Es existieren drei grobe Ansätze für die Virtualisierung von Servern, die sogenannte Hostvirtualisierung. Bei der Mehrzahl dieser Ansätze wird ein Virtual Machine Monitor (VMM) (auch Hypervisor) als Zwischenschicht eingeführt, wie oben erwähnt. Dieser verwaltet die Menge an instantiierten Betriebssystemen und vermittelt zwischen diesen und der Hardware.[8] Dazu wird setzt jedes Betriebsystem auf einer Virtuelle Maschine (VM) auf.

Die vollständige Virtualisierung ermöglicht es dem Gastsystem unverändert eingesetzt zu werden, da die gesamte Hardwareumgebung virtualisiert nachgebildet wird. Der dadurch entstehende negative Effekt des Einbußens von Systemleistung wird mit der Paravirtualisierung vermieden. Hier gilt es die Gastsysteme so anzupassen, dass eine Anbindung an einen einheitlichen Hypervisor möglich ist. Diese Anpassungen beschränken sich in der Regel auf das Hinzufügen sogenannter Hypercalls in den Befehlssatz der Gastsysteme, mit welchen der Hypervisor aufgerufen werden kann. Der dritte Ansatz für die Hostvirtualisierung heißt Betriebssystem-Virtualisierung. Im Gegensatz zu den anderen Konzepten übernimmt das Hostsystem die Rolle des Vermittlers. Die Gastbetriebssysteme laufen in voneinander isolierten Containern, welche die verschiedenen Laufzeitumgebungen darstellen.[8]

2.2.2 Intel und AMD und Virtualisierung

Wie bereits erwähnt, wurde Virtualisierung in der Vergangenheit auf spezialisierten high-end Servern und Mainframecomputern verwendet. Durch die Entwicklung immer leistungsfähigerer Prozessoren der beiden bekannten Chiphersteller Intel und AMD wurde nun die Prozessor-unterstützte Virtualisierung möglich.[12] Welche Konzepte hier umgesetzt wurden und inwiefern diese sich in der Vergangenheit als sicher erwiesen haben, soll im Folgenden geschildert werden.

Intel und AMD implementieren nach eigenen Angaben eine Sicherheitsarchitektur, die auf einem 2-bit Rechte System beruht. Dabei ist 0 (binär 00) der Privilegierteste und 3 (binär 11), der sogenannte Ring mit den wenigsten Rechten. Der Ring gibt an, ob privilegierte Befehle, welche elementare CPU Funktionalitäten steuern, von einem Programm ausgeführt werden dürfen.[12] In der Praxis werden jedoch fast ausschließlich die Ringe 0 und 3 genutzt.[5]

Virtualisierungstechnologien von Intel

Intel hat in den vergangenen Jahren seine IA-32 (Intel Architecture, x86, 32 Bit) und Itanium Prozessoren so modizifiziert, dass diese Virtualisierung nun direkt unterstützen. Dabei ist die Endung VT-x repräsentativ für die IA-32 Architektur und VT-i die Abkürzung für "Virtualization Technology für the Itanium architecture"(64 Bit).

[5] http://www.tecchannel.de/server/virtualisierung/432777/amd_pacifica_virtualisierung_von_cpu_speicher/index4.html

Der wesentliche Unterschied zwischen **VT-x** und der ursprünglichen Intel Architecture, x86, 32 Bit (IA-32) ist ein erweiterter Befehlssatz, die sogenannte Virtual Machine Extension (VMX).[6] Intel führt mit dieser Erweiterung *VMX root operations* und *VMX non-root operations* ein. Der Modus *root operations* ist als Laufzeitumgebung für die VMM angedacht, weshalb hier alle Befehlsprivilegien freigegeben sind. Zweitere stellen der VMM eine alternative/virtuelle Umgebung zur Verfügung, welche identisch zu den root operations Anweisungen aus allen vier Ringen zulässt. In diesem Modus sollen die Gast-Betriebssysteme und Anwendungen ausgeführt werden.[17] Ergo gibt es Übergänge vom root- zum non-root-Modus und umgekehrt. Diese nennt Intel *VM entry* für den Übergang hin zur *VMX non-root operation* und *VM exit* für die andere Richtung. Die eben beschriebenen Übergänge werden von der Virtual Machine Control Structure (VMCS) gesteuert. Die VMCS speichert weiterhin den Zustand von Host- und Gastsystem(en) um die *VM entries* und *exits* sinnvoll zu verwalten.[5]

Die Sicherheitsarchitektur der Intel VT-x Prozessoren wird in erster Linie durch die Fähigkeit der VMCS, bestimmte Befehle von Gast-Betriebssystemen zu blockieren, definiert. Dafür beinhaltet die VMCS eine Mehrzahl an Variablen, welche die *VMX non-root operations* kontrollieren und steuern. Beispielsweise gibt es ein Feld, welches bei Aktivierung externe Interrupts unterbindet, indem sofort ein *VM exit* erzwungen wird.[12]

Für die **VT-i** Architektur hat Intel den gewöhnlichen Itanium-Prozessor um eine sogenannte Processor Abstraction Layer (PAL) in Form einer Firmware erweitert. In dieser Schicht gibt es das Pprocess status register (PSR), welches angibt, ob der VMM (für PSR.vm = 0) oder das Gast-Betriebssystem (für PSR.vm = 1) zum aktuellen Zeitpunkt den Prozessor nutzen darf. Dabei ist es unerheblich, auf welchem Ring der Gast ausgeführt wird, was eine gewisse Flexibilität gegenüber Gastsystemen darstellt.[7] Das PSR.vm bit steuert das Verhalten aller privilegierter und auch einiger nicht-privilegierter Befehle. Versucht ein Gast also einen dieser Befehle auszuführen, welcher vielleicht nicht-privilegiert aber für die VMM reserviert ist, wird ihm die Kontrolle mittels Zurücksetzen des PSR.vm auf 0 entzogen.[12]

Zusammenfassend ist zu sagen, dass der Intel VT-i mit der PAL ein gewisses Maß an Sicherheit vorweisen kann, indem mittels einer globalen Variable die Umgebung für eine virtuelle Maschine einfach ein- und ausgeschaltet werden kann. Weitere, hier nicht weiter erwähnte Konfigurationsoptionen legen außerdem fest, unter welchen Umständen Virtualisierung unterbunden wird, was wiederum einen Gewinn an Sicherheit darstellt.

Neben der prozessor-unterstützten Virtualisierung hat Intel auch Anstrengungen bezüglich Virtualisierung auf chipset-Ebene getätigt. Die Intel **Virtualization Technology for Directed I/O (VT-d)** assistiert dem Prozessor bei der Virtualisierung der Ein-/Ausgabegeräte.[4] Laut Hersteller soll VT-d sowohl durch Isolation von Ein-/Ausgabegeräten zur Sicherheit und Verlässlichkeit beitragen als auch die Leistungsfähigkeit der virtualisierten Hardware steigern. Das Prinzip besteht darin, dass mittels Emulation oder Paravirtualisierung (Software) nur solche Gast-Betriebssysteme auf ein Gerät Zugriff haben,

[6] http://www.tecchannel.de/server/virtualisierung/402566/intels_vanderpool_virtualisiert_cpus-/index4.html

[7] http://software.intel.com/en-us/articles/how-to-incorporate-intel-virtualization-technology-into-an-overview-of-itaniumr-architecture/

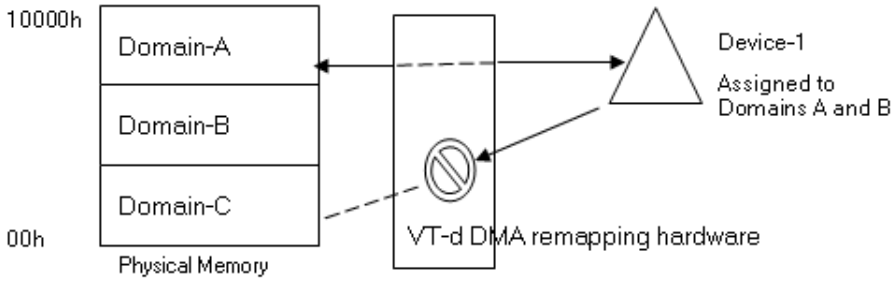

Abbildung 2.1: DMA-remapping Schicht des Intel VT-d, [4]

welches ihnen zugeteilt ist. Dies geschieht per Einschränkung des direct memory access (DMA) auf vorher bestimmte Domänen oder Bereiche im Hauptspeicher. Diese sogenannten Schutzdomänen sind voneinander isoliert. Tritt nun das Szenario ein, dass ein solcher Bereich (z.B. ein Gast-Betriebssystem) auf ein Ein-/Ausgabegerät zugreifen möchte, prüft die installierte DMA-remapping Hardware, ob der besagte Bereich authorisiert ist seinen Zugriff durchzuführen.

Die Abbildung 2.1 zeigt, dass hier die Domäne C keinen Zugriff auf das Gerät hat, welches explizit den Domänen A und B zugewiesen ist.

Der Nutzen von VT-d besteht darin, dass delinquenter DMA unterbunden wird was die Funktionsfähigkeit des Gesamtsystems einschränken könnte und dass die Bildung der Schutzdomänen in beliebiger Granularität vorgenommen werden kann. Somit kann man selbst innerhalb einer Virtuellen Maschine mehrere Domänen erstellen. Einerseits kann so Redundanz als proaktive Maßnahme gegen Fehlfunktionen geschaffen werden und andererseits können infizierte oder defekte Domänen einfach ignoriert oder gelöscht werden.[4]

Virtualisierungstechnologien von AMD

AMD Secure Virtual Machine Architecture (SVM), auch *Pacifica* genannt, ist das Pendant zum Intel VT, wobei zu erwähnen ist, dass die beiden Entwicklungen unabhängig voneinander abgelaufen und die Prozessoren somit auch nicht kompatibel sind. Allgemein gesprochen garantieren AMD-V Prozessoren Unterstützung für Virtualisierung mittels schnellem Umschalten zwischen VMM und Gast, Zugriffsschutz für externen Speicher und einem effizienten Interrupt-Management. Letzteres wird durch den *Virtual Memory Control block (VMCB)* sichergestellt, indem ein externes Interrupt den momentan aktiven Gast zum *#VMEXIT* (vergleiche *VM exit* bei Intel) zwingt. Analog zu Intel, ermöglicht der SVM-Befehl *#VMRUN* (vergleiche *VM entry*) dem Prozessor im Gast-Modus zu arbeiten.[1]

Ähnlich wie beim Intel VT-d werden Ein-/Ausgabegeräte explizit einem oder mehreren Gast-Systemen zugewiesen und nicht Allokierten der Zugriff verweigert. Dies ist ein nennenswerter Unterschied zu den x86 Prozessoren von Intel, welche keinen integrierten Memory-Controller besitzen. Im Gegensatz zu Intel hat AMD eine Hardwarelösung gefunden, welche den aufwendigeren Ansatz der Emulation umgeht und somit weniger

zeitaufwendig ist.[8] Jeder virtuellen Maschine wird mit einem "nested-paging-modus" ein eigenes virtuelles Register zur Verfügung gestellt, welches die Adresse der Seitentabelle enthält. Auch wenn hiermit eine weitere Übersetzungsschicht eingeführt wird, erfolgen die Vorgänge auf Hardwareebene und somit effizienter als die Lösung von Intel. Des Weiteren sorgt ein in den *Pacifica* integrierter Device Exclusion Vector (DEV) dafür, dass DMA-fähige Geräte direkt und ohne Hilfe des Prozessors auf den Speicher des Systems zugreifen können. Der DEV ist ein Array von bits im physikalischen Speicher, wobei jedes Bit einem 4Kbyte Bereich im Hauptspeicher entspricht.[1] Es ist jede Schutzdomäne mit einem DEV gekoppelt, sodass bei jedem DMA-Zugriff dessen Gültigkeit überprüft werden kann.

2.3 Angriffsszenarien

Bisher wurde erläutert, worum es sich bei Virtualisierung handelt und welche Bedeutung sie hinsichtlich der Sicherheit in der Informationstechnik hat. Man kann jedoch mithilfe von Virtualisierung auch Angriffe auf Server starten, wobei man entweder Fehler in deren Architektur/Implementierung ausnutzt oder indem man die Virtualisierung direkt dafür verwendet, die Kontrolle über ein System zu erlangen. Da Virtualisierung zunehmend von Unternehmen benutzt wird, verwundert es nicht, warum Angreifer nicht auf die eigene Nutzung dieser Technologie verzichten. Dazu gilt es jedoch erst einmal zwei wichtige Begriffe zu erläutern: die Sandbox und das Rootkit.

2.3.1 Grundlegende Begriffe

Eine **Sandbox** ist, wie der Name schon sagt, eine Testumgebung. Hier wird die Realität zu Zwecken der Demonstration und Forschung nachgestellt/virtualisiert. In der Informationstechnik ist eine Sandbox eine abgeschottete Laufzeitumgebung für Programmcode. Dies kann ein komplettes Betriebssystem oder ein Browser sein. [9] Der Unterschied zur virtuellen Maschine besteht darin, dass bei der Sandbox kein virtueller Computer respektive Server nachgebildet wird. Vielmehr wird hier auf ein bestehendes Betriebssystem aufgesetzt.

Sandboxes stellen eine sichere Umgebung zur Verfügung, in welcher Programmcode nur beschränkt ausgeführt werden kann. Dabei unterscheidet man zwischen Kernel-level- und User-level-Sandbox. Erstere ist effizienter, da hier die Funktionen des Betriebssystems direkt genutzt werden, jedoch muss der Hersteller für dieses auch die Spezifikation freigeben. Der zweite Sandboxtyp nutzt Mechanismen, die verhindern, dass der Benutzer (Gast) aus seiner Sandbox "ausbricht".[9] Das bedeutet, dass Prozesse innerhalb der abgeschotteten Umgebung versuchen auf das Hostsystem (hier der Hypervisor) beziehungsweise die physikalische Hardware zuzugreifen. Dieser Vorgang wird auch als *VM escape* bezeichnet.[10]

[8] http://www.tecchannel.de/server/virtualisierung/432777/amd_pacifica_virtualisierung_von_cpu-_speicher/index9.html

[9] http://www.pcwelt.de/start/sicherheit/firewall/praxis/2101527/was_ist_eine_sandbox/

[10] http://lonesysadmin.net/2007/09/22/what-is-vm-escape/

Schadsoftware hat oft die Eigenschaft die Umgebung, von der sie analysiert wird, zu beschädigen um eben das Analysieren zu unterbinden. Da Sandboxes jedoch einfach neu zu bauen sind, sind sie besonders gut dafür geeignet, Schadsoftware zu entdecken und deren Verhalten zu beobachten.[11] Schlussfolgernd ist klar, dass das Sandboxing-Verfahren in erster Linie von Antiviren-Herstellern für die Suche nach neuen Schadprogrammen genutzt wird.

Ein **Rootkit** ist Schadsoftware, welche die Kontrolle über ein Computersystem erlangt, ohne sich vorher zu authorisieren. Das Ziel ist das Erlangen und Erhalten dieser Kontrolle. Dies wird in der Regel dadurch erreicht, indem in das zu infizierenden System eine "Hintertür" installiert wird, welche vom Angreifer später genutzt wird um das System zu sabotieren.[6] Bei der Übernahme des Zielsystems wird in der Regel auf bestehende Sicherheitslücken im Betriebssystem oder der Hardware zurückgegriffen bzw. auf die Naivität des Benutzers gesetzt, welcher seinen Computer unbeabsichtigt mit wenigen Mausklicks mit einem Virus oder Trojaner infizieren kann. Diese ungewollt aus dem Netzwerk (z.B. das Internet) heruntergeladenen Programme können verschiedene Ebenen der Interaktion durchführen, wie aus den Grundlagen der IT-Sicherheit bekannt ist. Die für den befallenen Rechner ungefährlichsten Schadprogramme sind solche, die mit dem Zielsystem garnicht interagieren, sondern höchstens denial-of-service Angriffe auf andere Server starten oder Phishing durchführen. Es folgen Programme, die Informationen über das Zielsystem beobachten und weitermelden. Hierzu gehört das Abhören von Tastatureingaben sowie das Mitlesen vom Netzwerkverkehr. Schließlich gibt es solche Schadsoftware, die die Manipulation des Zielsystems zum Zweck hat. Das schließt das Kontrollieren vom Netzwerkverkehr, den Versand von Emails und das Ausführen und Beenden von Anwendungen ein.[10]

Schadprogramme, auch oft als Malware bezeichnet, benutzen verschiedene Wege, ein System zu infiltrieren und dessen Datenstrukturen und Programme zu manipulieren. Joanna Rutkowska hat auch hierfür eine Klassifikation vorgenommen, welche hier aus Sicht der System-Kompromittierung zu sehen ist. Rutkowska hat Malware in vier elementare Typen unterteilt.

Type-0-malware hat eigene Dateien oder modifiziert existierende Dateien so, dass die Schadsoftware von dort aus weiter agieren kann. Dies ist der am meisten verbreitete Typ unter den User-mode rootkits. Es werden in der Regel Prozesse gestartet und Dateien versteckt oder verändert, sodass dem Administrator eine falsche Sicht auf sein System vorgegaukelt wird.[6]

Die Erkennung einer solchen Kompromittierung ist sehr leicht zu entdecken. Mittels Signatur scans und Integrität checks kann die Authentizität der Daten auf dem Rechner überprüft werden.

Type-1-malware modifiziert ausschließlich read-only Daten. Dazu gehören Code und statische Variablen innerhalb laufender Anwendungen oder Kerne. Hier soll der Kontrollfluss zum Schadcode direkt aus Anwendungen in der Ausführung heraus umgeleitet werden. Es wird die Technik des "Inline Function Hooking" eingesetzt.[6]

Eine Erkennung eines solchen Befalls ist aus dem Gast-Modus heraus unmöglich, da in der Regel nur ein Administrator die Berechtigung hat Integrität scans über signierte ausführbare Dateien durchzuführen - was hier notwendig ist. Beispielsweise könne man Code und

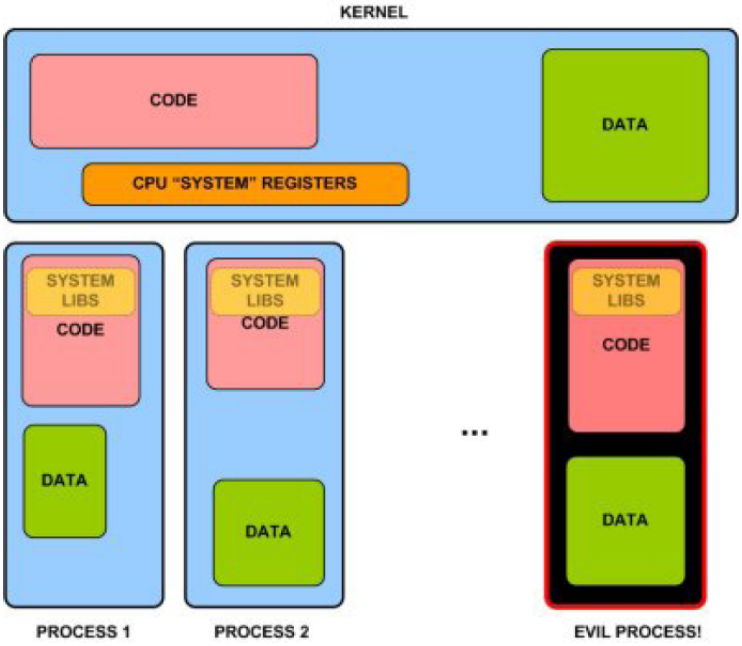

Abbildung 2.2: Type-0-malware[15]

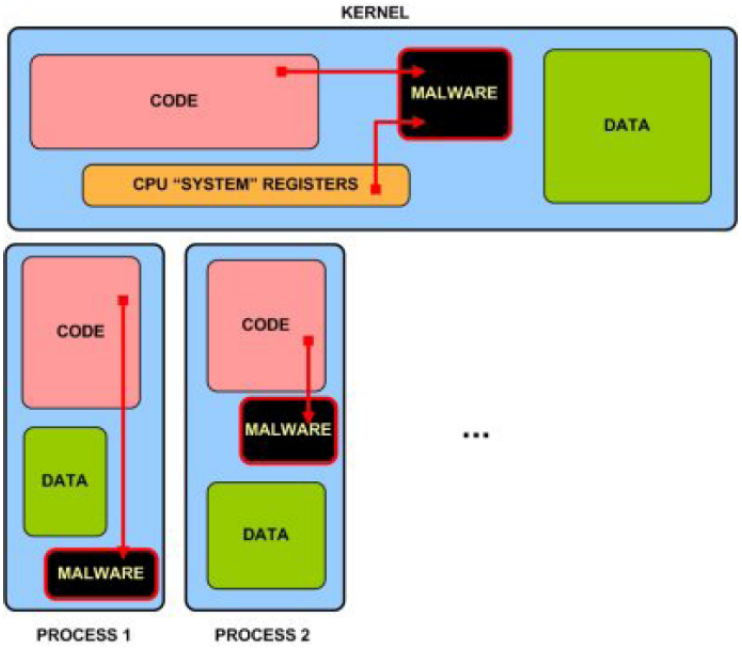

Abbildung 2.3: Type-1-malware[15]

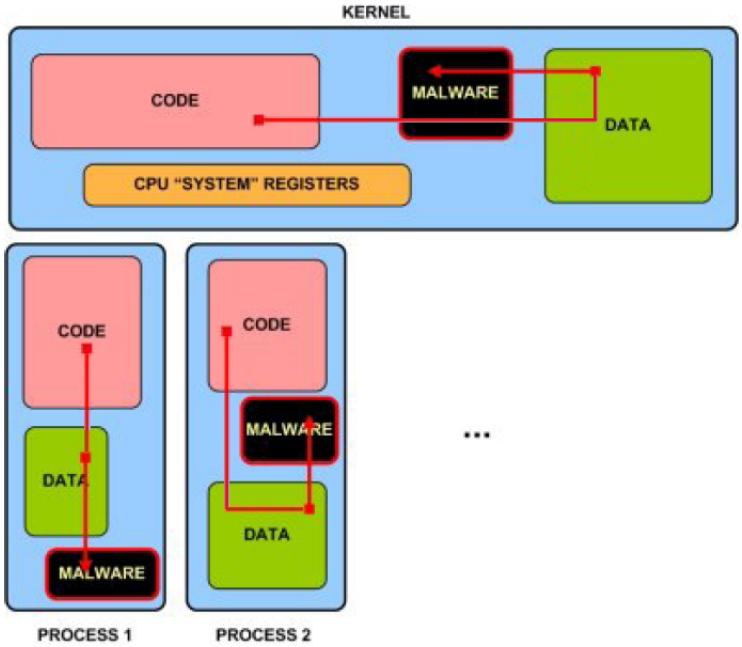

Abbildung 2.4: Type-2-malware[15]

statische Daten mit Prüfsummen verifizieren um den Zustand bezüglich der Korrektheit des Programms festzustellen.

Type-2-malware verändert nun nur noch Anwendungsdaten, welche von den auf dem Rechner installierten Programmen ohnehin manipuliert werden sollen. Es werden also ausschließlich dynamische Ressourcen verändert um zum Beispiel in Datenbanken abgelegte Pointer zum Schadprogramm zu führen und nicht zu der Anwendung, welche sie in erster Linie anstoßen sollten. Greift nun ein "gutes" Programm auf einen solchen Verweis zu, bewirkt der modifizierte Eintrag in der Datenbank eine solche Umleitung.[15]

Es wird klar, dass Signatur scans oder Ähnliches hier unwirksam sind, da die veränderten Daten wissentlich dynamischer Natur sind. Es ist theoretisch möglich ein Suchprogramm zu implementieren, welches alle Einhakstellen kennt und die korrekte Belegung dieser prüft. Ein solcher *Memory-integrity-scanner* existiert noch nicht, jedoch müsste dieser mindestens den Bereich im Hauptspeicher untersuchen, in dem Variablen des Betriebssystems abgelegt sind.[15]

Type-3-malware benötigt keine Einhakpunkte im Zielsystem selbst. Stattdessen wird dieses von außerhalb kontrolliert. Da ein solcher Befall nicht vom infizierten System aus erkannt werden kann, ist diese Art der Schadsoftware absolut unsichtbar, was ihr wiederum den Namen "stealth malware" bescherte.[7]

In Abbildung 2.5 kann man bereits erkennen, dass hier Virtualisierung ein potentes Mittel zur Durchführung eines solchen Angriffs ist. So könne man das Zielsystem samt Kernel und Prozessen auf eine andere Ebene der Privilegien verschieben. Damit unterbricht man die direkte Anbindung des Betriebssystems zur Hardwareebene.[15]

Dieser Typ 3 der Schadsoftware ist das, was wir unter einem Rootkit verstehen, da hier die vollständige Kontrolle über ein System erlangt wird ohne das System selbst zu ver-

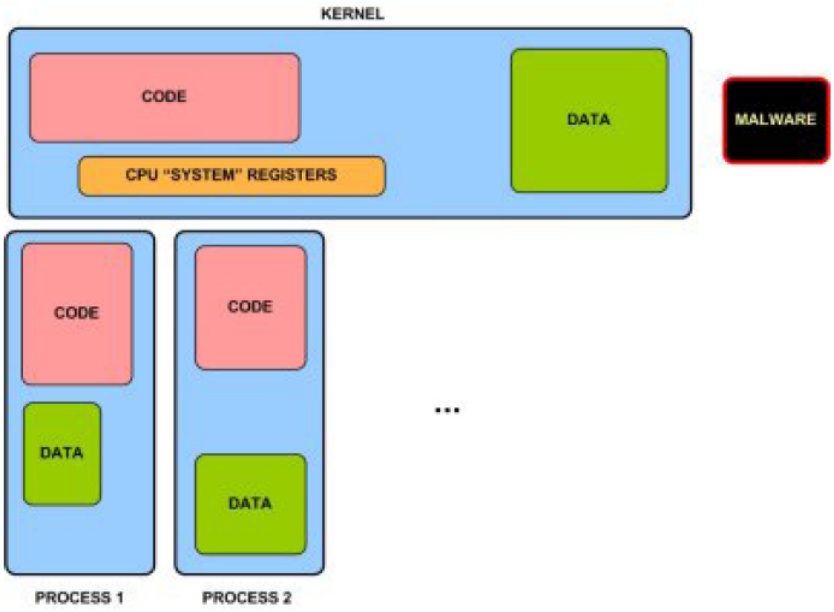

Abbildung 2.5: Type-3-malware[15]

ändern. In den folgenden Abschnitten werde ich mich ausschließlich auf diesen Typ der Schadprogramme fokussieren.

2.3.2 Zwei Arten von Rootkits

Grundsätzlich wird zwischen zwei Arten von Rootkits, die Virtualisierung als Mittel zur Kompromittierung verwenden, unterschieden. Ein Virtual Machine based Rootkit (VMBR) dringt in ein Zielsystem ein und bewirkt, dass beim nächsten Start des Rechners ein VMM vor dem eigentlichen Betriebssystem gestartet wird. Dieser betreibt das Zielsystem in einer VM. Die andere Art der Hardware Virtualization based Rootkits (HVBR)s verschiebt das Zielsystem zur Laufzeit in eine virtuelle Umgebung. In diesem Abschnitt sollen bestimmte Angriffsszenarien vorgestellt werden, wobei es sich lediglich um proof-of-concept handelt, es also nur eine Implementierung eines solchen Rootkits mit den wichtigsten Kernfunktionalitäten gibt.

Der Weg, auf dem eines der oben genannten Programme auf den Rechner gelangt, ist bei allen identisch. Die primäre Schwachstelle eines jeden IT-Systems ist der Mensch, denn er installiert, kontrolliert und verwaltet das System. Somit ist es keine Überraschung, dass Trojaner und Viren in einen Rechner eindringen wenn der Benutzer meist auch Administrator ist. Dies erleichtert es dem Angreifer das Zielsystem zu kompromittieren.

2.3.3 SubVirt - Ein VMBR

Die VMBRs werden auch kurz mit SubVirt bezeichnet. Das Konzept wurde von Microsoft erarbeitet und präsentiert.[14] Das Rootkit installiert einen “thin Hypervisor” unter das

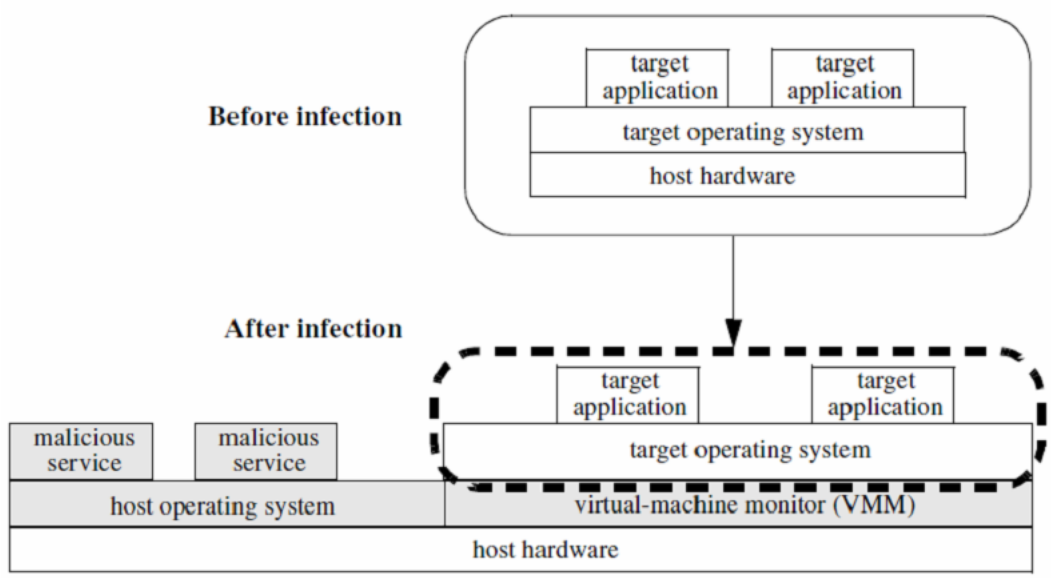

Abbildung 2.6: SubVirt: Vorher-Nachher[10]

aktuelle Betriebssystem und betreibt dieses in einer virtuellen Maschine. Die Schadprozesse selbst, welche nach Installation der besagten Hintertür in das Gesamtsystem eindringen, werden oft innerhalb einer weiteren VM installiert. Dies macht es noch unwahrscheinlicher, dass diese Prozesse vom Benutzer erkannt werden, da sich parallele (virtuelle) Umgebungen keinesfalls untereinander erkennen.

Abbildung 2.6 zeigt, wie das System vor der Übernahme aussieht und wie es danach aussehen soll. Die grau hinterlegten Flächen gehören zum VMBR und das eigentliche Betriebssystem ist nach dem Befall nur noch ein Gastsystem unter der Obhut des VMM.

Nachdem SubVirt in das Zielsystem eingedrungen ist, versucht es die boot-Sequenz so zu verändern, dass zuerst das VMBR und aus diesem heraus das Betriebssystem gestartet wird. Dazu wird der Startzustand des VMBR an der Stelle auf einem persistentem Speichermedium abgelegt, welche vorher das Hochfahren des Betriebssystems anstieß. Das Resultat ist eine manipulierte oder mittels Virtualisierung verschobene boot-Sektion, welche beim erneuten Starten des Rechners in einer virtuellen Maschine gestartet wird.

Diese Art von Rootkits ist deshalb gefährlich, weil es sich um user-mode malware handelt, welche einfacher zu implementieren ist und in einer beliebigen Programmiersprachen verfasst werden kann. Das VMBR erlangt mittels Virtualisierung Zugriff auf alle Bibliotheken und Ressourcen des Betriebssystems. Allerdings können diese Manipulationen durch das Überprüfen von etwaigen Signaturen im boot-Sektor erkannt werden. Hat der VMM des Angreifers erst einmal die Kontrolle übernommen, fällt es ihm relativ leicht diese auch zu behalten, schließlich arbeitet der Benutzer nicht mehr auf dem physikalischen, sondern dem virtuellen Speicher seiner VM.[10] Grundsätzlich ist zu sagen, dass sämtliche Zugriffe des Zielsystems auf die Hardware vom VMM abgefangen werden können, woraufhin dieser gegebenenfalls ein gefälschtes Resultat für die Anfrage des Benutzers zurückliefert.

Die wohl gravierendste Schwachstelle der VMBR besteht darin, dass ein Neustart des Systems zwingend notwendig ist damit das Rootkit zur Wirkung kommt, denn für ge-

wöhnlich verbringen Server den größten Teil ihres Daseins im laufenden Betrieb. Des Weiteren kann der Zeitraum zwischen Einschalten der Stromversorgung und Initiierung des VMMs effektiv zur Verteidigung genutzt werden. Somit kann auf Ebene des BIOS dafür gesorgt werden, dass wenn ein anderes Programm als das Betriebssystem des Nutzers gestartet wird, die boot-Sequenz mit einer Fehlermeldung abgebrochen wird.[10] Außerdem wird durch das Beenden und Starten des VMMs die Dauer eines Neustarts merkbar verlängert.[7] Dieses Argument kann allerdings relativ einfach entwertet werden. Der VMM kann den Neustart simulieren. Dabei wird der Rechner in einen sogenannten low-power-mode versetzt. Dazu werden die Festplatten heruntergefahren, Lüfter ausgeschaltet und möglichst alle LEDs stromlos geschaltet. Der Hauptspeicher ist weiterhin mit Strom versorgt wodurch beim erneuten Einschalten lediglich der low-power-mode verlassen wird. Das VMBR hat in diesem Szenario zu keinem Zeitpunkt die Kontrolle verloren. Ein Haken dieser Strategie des vorgekaukelten Herunterfahrens ist, dass die Ausstattung von Rechnern mit LEDs sehr unterschiedlich ausfallen kann. Hier kann ein aufmerksamer Benutzer merken, dass mit seinem System etwas nicht stimmt.

Neben der Problematik des Neustarts ist es auch möglich, dass der Benutzer auf eine eventuelle Infiltration seines System aufmerksam wird, indem er bemerkt, dass sich seine Hardware anscheinend geändert hat. Das liegt daran, dass es oft schwer fällt hochspezialisierte Geräte wie Grafik- oder Netzwerkkarten detailgetreu virtualisiert nachzubilden.[7] Eine weitere Methode um herauszufinden, ob man sich in einer virtualisierten Umgebung befindet, ist bei x86 Prozessoren die Prüfung des sogenannten IDT Registers. Dieses verweist auf einen bestimmten Bereich im Hauptspeicher wenn Virtualisierung verwendet wird und auf einen anderen Bereich wenn dies nicht der Fall ist. Somit kann durch Abfrage des Wertes des IDT Registers einfach festgestellt werden, ob momentan seitens des Prozessors Virtualisierung verwendet wird und man womöglich Opfer dieses Angriffs ist.[7]

Es ist unschwer zu erkennen, dass VMBR zwar mächtig sind, aber sich die Detektion relativ trivial gestaltet. Es gibt zwei Lösungen für Gegenmaßnahmen. Zum Einen kann man wie angesprochen eine low-level Erkennungssoftware im BIOS installieren um die physikalischen Speicher auf Schadprogramme zu untersuchen. Die andere Variante ist ein sicheres Hochfahren des Rechners mittels CD-ROM oder ähnlicher externer Medien. Es ist also fraglich, ob sich der relativ große Aufwand für die Implementierung und Einschleusung eines VMBR lohnt, um die gewünschte Menge an Schäden anzurichten.[10]

2.3.4 Vitriol - Das HVBR für Intel

Vitriol ist ein HVBR und dessen Konzept wurde von dem Sicherheitsspezialisten Dino Dai Zovi entwickelt.[14] Es nutzt Schwachstellen der Intel VT-x Prozessoren aus. Dazu zählen beispielsweise die Tatsachen, dass VT-x virtuellen Maschinen aus Effizienzgründen direkten Zugriff auf Hauptspeicher und andere Hardware gewährt und dass die Verhinderung des Erkennens, ob sich eine Software in einer virtuellen Maschine befindet, ein Designziel war.[5] Ein HVBR missbraucht die vom Hersteller des Prozessors gelieferten erweiterten Befehlssätze, um eine Zwischenschicht zwischen Hardware und Betriebssystem zu installieren; so auch Vitriol.

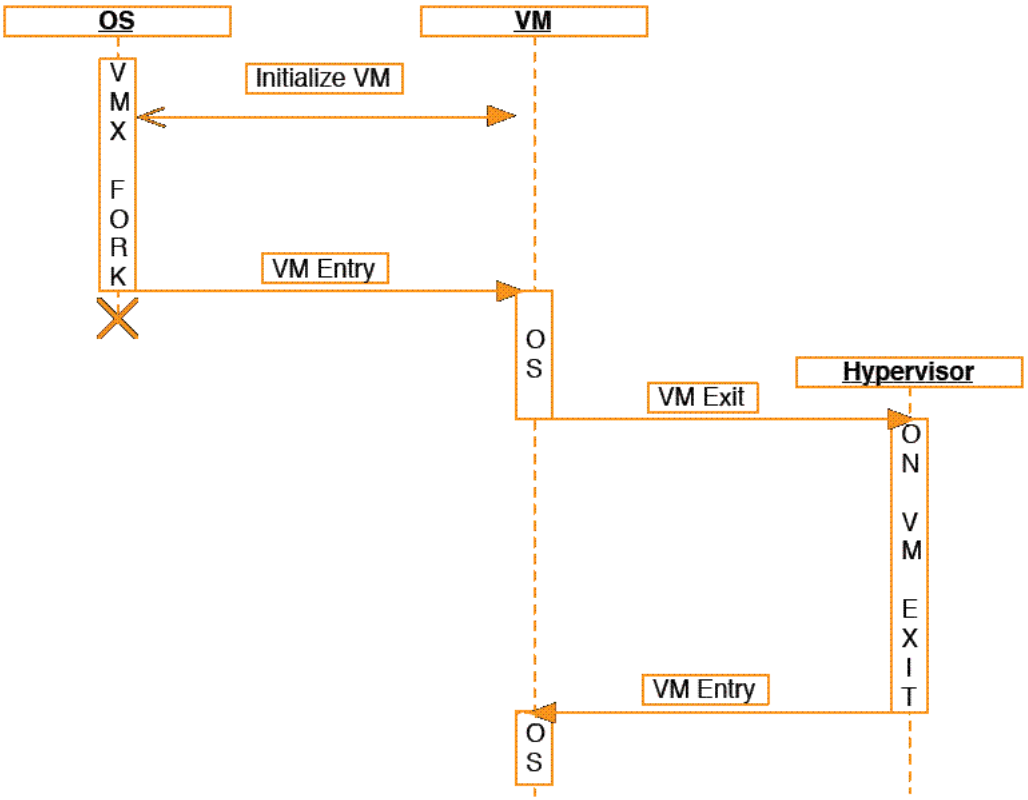

Abbildung 2.7: Vitriol Übernahme[5]

Nachdem sich Vitriol zur Laufzeit auf einem Rechner installiert hat, führt es seine 3 Hauptfunktionen aus.

Der Befehl *VMX_init()* prüft, ob es sich bei der Zielhardware um einen Computer handelt, der prozessor-unterstützte Virtualisierung unterstützt. Danach wird die VMX aktiviert und der VMCS ein Bereich im Hauptspeicher zugewiesen. Dies sind also vollkommen normale Schritte, wenn man die Virtualisierungsunterstützung seitens des Prozessors einschalten möchte.[5]

Es folgt des Befehl *VMX_fork()*, welcher dafür sorgt, dass das laufende Betriebssystem in eine virtuelle Maschine verschoben und ein neuer Hypervisor erschaffen wird. Ab diesem Zeitpunkt ist das Zielsystem nur noch ein Gast, welcher vom eben genannten Hypervisor kontrolliert wird. Dabei ist es wichtig, dass der momentane Zustand des Zielsystems erfasst und als sichtbarer Zustand der virtuellen Maschine gesetzt wird, um zu vermeiden, dass der Benutzer den Übergang bemerkt.[5]

On_vm_exit() verwaltet nun die *VM exit* events da hier normalerweise eine Möglichkeit für den Benutzer bestünde die VM, in der er sich mit dem Betriebssystem befindet, zu bemerken. Also müssen sämtliche Registerabfragen, privilegierte Befehle und Zugriffe auf die Hardware streng kontrolliert und manipuliert werden. Des Weiteren bietet sich an dieser Stelle die Chance, die besagten Hintertüren zu installieren, durch welche der Angreifer später auf das Zielsystem zugreifen kann.[5]

Abbildung 2.7 zeigt wie diese drei Befehle hintereinander ausgeführt werden und welche Konsequenzen dies für das Zielsystem hat. Ist dieses erst einmal auf die virtuelle Maschine

migriert, gibt es keine Chance mehr, die VM zu verlassen. Der Hypervisor fängt sämtliche Befehle ab und liefert für ihn günstige Ergebnisse zurück. Somit hat der Benutzer nur sehr wenige und unzuverlässige Mittel um die Kompromittierung zu erkennen, denn die Übernahme findet zur Laufzeit statt.

Vitriol ist wegen seiner unsichtbaren Natur und der kaum bemerkbaren Infiltration sehr gefährlich. Auch wenn es ähnlich wie bei SubVirt die Möglichkeit gibt, Veränderungen an der Sicht auf die Hardware festzustellen, gibt es diesmal kein Hardwarebit oder Register, das anzeigt, ob sich der Prozessor im non-root modus befindet. Dies ist, wie bereits angemerkt, vom Hersteller so gewollt. Selbst wenn es eine Abfragemöglichkeit an die Hardware gäbe, wäre es ein Leichtes für den Hypervisor diese zu manipulieren.

Es gibt jedoch Wege, sich vor einem Angriff durch Vitriol zu schützen beziehungsweise zu bemerken, dass man sich in einer virtualisierten Umgebung befindet. Der wohl einfachste Schutzmechanismus besteht darin, Virtualisierung bereits im Einsatz zu haben. HVBRs unterstützen keine nested Virtualization; also eine Virtualisierung innerhalb einer Virtualisierung. Diesen Makel kann man auch zur Detektion verwenden, indem man aus dem kompromittierten System heraus einen VMM zu starten versucht. Dabei erstellt man einen einfachen arithmetischen Algorithmus und schreibt dessen Ergebnis in ein Register. Im Fall eines Befalls durch ein HVBR tritt an dieser Stelle ein Fehler auf, weil das Rootkit natürlich verhindern wird, dass eine weitere virtuelle Umgebung erschaffen wird. Dieser Fehlschlag kann als Indiz dafür dienen, dass sich das Betriebssystem des Benutzers in einer VM befindet.

Auch wenn der Hypervisor die Frequenz der auftretenden VM exits so gering wie möglich zu halten versucht, indem Exceptions oder unnötiger I/O-Zugriff ignoriert werden, kann der Benutzer theoretisch feststellen, dass es zu kleineren Verzögerungen im Arbeitsfluss kommt. Da ein Counter wieder vom Hypervisor beeinflusst werden kann, bietet sich hier das Benutzen einer externen Uhr an. Schließlich hat Intel die Zeitdauer, wie lange ein *VM exit* oder *VM entry* dauert, veröffentlicht.

Zusammenfassend ist klar, dass Vitriol, wenn es denn in einem realen Rootkit implementiert wird, eine Gefahr für Server darstellt, die die Virtualisierungstechnologie von Intel benutzen. Eine ähnliche Gefahr besteht für Rechner mit einem AMD-V Prozessor, was durch das folgende Beispiel beleuchtet werden soll.

2.3.5 Blue Pill - Das HVBR für AMD

Analog zu Vitriol nutzt Blue Pill Prozessorvirtualisierung aus um eine zusätzliche Hypervisor-Zwischenschicht einzuführen. Auch hier wird das Zielsystem im laufenden Betrieb übernommen. Somit zählt das von Joanna Rutkowska entwickelte proof-of-concept Rootkit zu den HVBR.[11] Dabei ist Frau Rutkowska besonders bemüht auf dem Gebiet der "stealth malware" Fortschritte zu erzielen, was ihr mit diesem Rootkit angesichts des Aufruhrs in der IT-Sicherheit klar gelungen ist.

[11] http://theinvisiblethings.blogspot.com/search?q=blue+pill&updated-max=2006-06-22T13%3A05%3A00%2B02%3A00&max-results=20

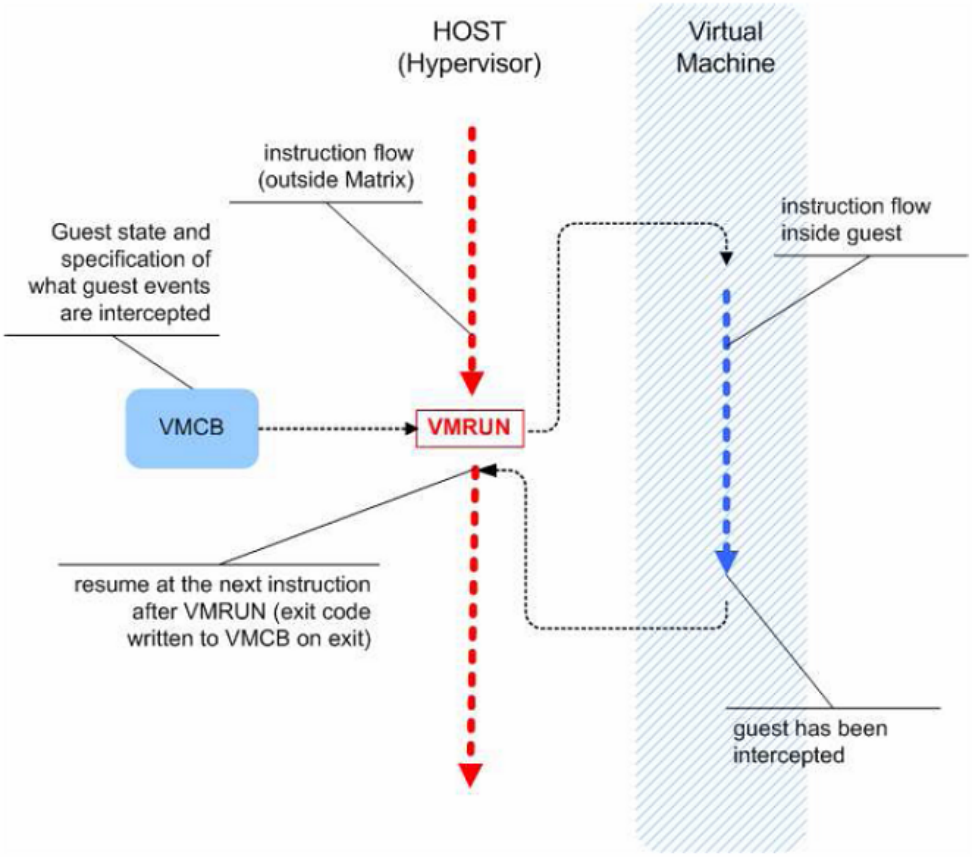

Abbildung 2.8: Die “gesunde” SVM[16]

Bevor die Funktionsweise von Blue Pill näher erläutert wird, soll Abbildung 2.8 verdeutlichen, wie der gewöhnliche Sprung im Kontrollfluss zwischen Host und virtueller Maschine abläuft. Gibt der VMCB den Befehl *#VMRUN*, werden Operationen innerhalb des Gast ausgeführt. Das Verlassen der virtuellen Maschine kann entweder durch Anfragen bezüglich Hardwarezugriff oder durch Entzug des Prozessors weil sich der Gast nicht daran hält, nur für ihn vorgesehene Befehle auszuführen, bewirkt werden. In diesem Fall wird der Kontrollfluss nach Eintreten in die VM fortgeführt.

Wie erwähnt baut der Angriff auf der secure virtual machine Architektur von AMD auf, da diese genutzt wird, das Betriebssystem des Benutzers in eine virtuelle Maschine zu verschieben. Das Erste, was Blue Pill macht, wenn es das Zielsystem infiziert hat, ist das zwölfte Bit im MSR Extended Feature Register (EFER) mit 1 zu belegen um die Ausführung jeglicher SVM Befehle zu ermöglichen.[2] Nun kann analog zu Vitriol der VMCB vorbereitet werden und mit *#VMRUN* eine virtuelle Maschine erstellt werden, welche für das Zielsystem vorgesehen ist.

Abbildung 2.9 verdeutlicht den Ablauf der Übernahme vereinfacht. Hier wird klar, dass der Benutzer nichts von der Kompromittierung seines Systems mitbekommt, da das native Betriebssystem scheinbar während der kompletten Zeit ganz normal ausgeführt wird. Dadurch, dass keine Modifikationen im BIOS, boot-Sektor oder Dateisystem des Betriebssystems selbst nötig sind, wird die Selbstinstallation von Blue Pill nicht bemerkt.[16]

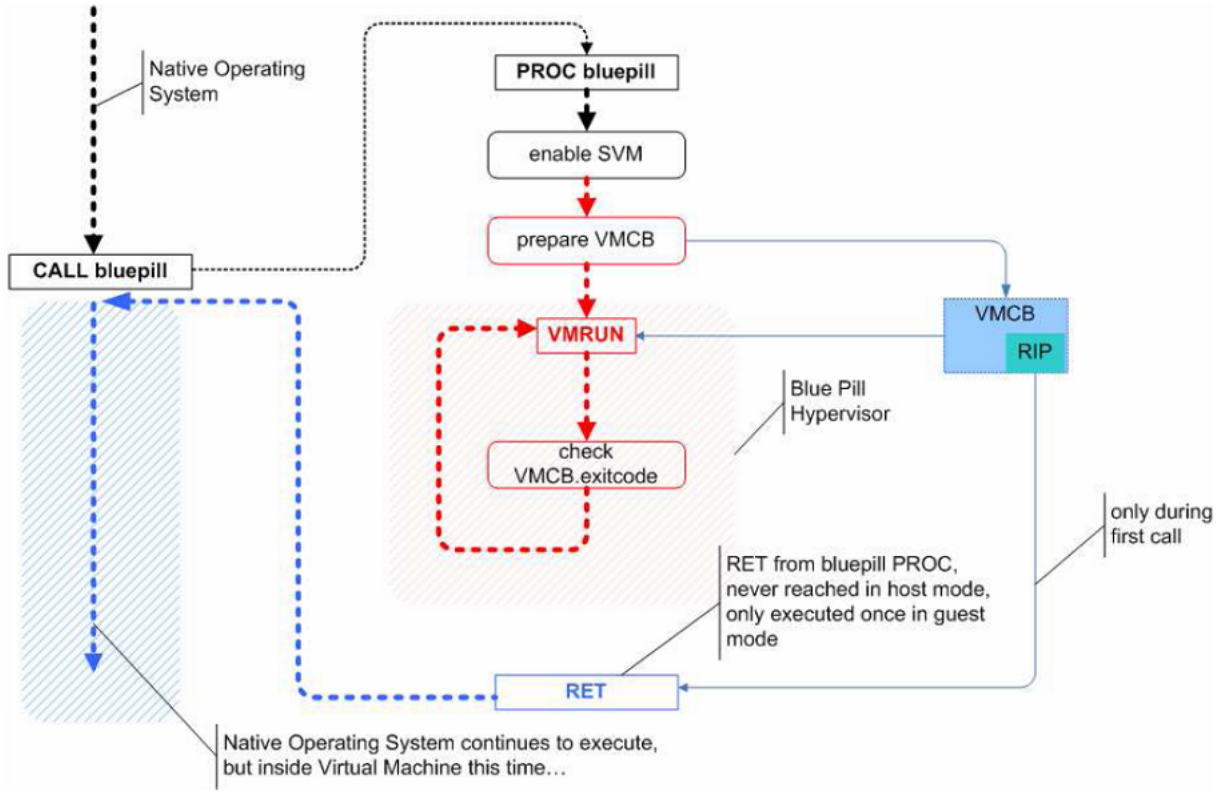

Abbildung 2.9: Blue Pill Übernahme[16]

Dennoch gibt es theoretisch Wege und Mittel einen Befall durch Blue Pill zu bemerken. Zum Einen kann man das oben angesprochene MSR EFER prüfen um festzustellen ob die SVM überhaupt aktiviert ist. Allerdings ist es ein Einfaches für den Hypervisor, eine solche Abfrage zu erkennen und gefälscht zu beantworten. Des Weiteren sind durch Einsatz des VMM die Zugriffszeiten auf bestimmte Register verlängert, was wiederum nur mit einer externen Uhr sicher festgestellt werden kann.[16]

Wie alle HVBRs überlebt auch Blue Pill einen Neustart des Systems nicht. Einerseits lässt sich darüber streiten, ob ein Neustart einen Servers sonderlich oft vorkommt, andererseits könne man wie im Abschnitt zu SubVirt einen Neustart simulieren ohne dass der Rechner wirklich heruntergefahren wird. Dennoch kann dies geschehen, was bedeutet, dass sich Blue Pill beispielsweise im Netzwerkpuffer der Netzwerkkarte ablegt, sodass selbst nach dem Neustart die Kompromittierung nicht aufgehoben ist. Diese Strategie wird in der Erweiterung “delusion backdoor” von Blue Pill implementiert und hat als Folge, dass das Rootkit das Zielsystem erneut übernehmen kann.[16]

Letztendlich sind HVBRs äußerst schwer zu erkennen da die vorgestellten Methoden nicht solche sind, die man für gewöhnlich jeden Tag durchführt. Allein Zeitmessungen mithilfe externer Uhren stellen Unternehmen mit einer Vielzahl an Rechnern, die Virtualisierung unterstützen, vor eine große Herausforderung. Nach [7] gibt es eine effektive präventive Maßnahme, die eine Übernahme verhindert:

Wird die Virtualisierungsfunktion nicht genutzt, sollte sie im BIOS deaktiviert werden. Wenn sie andererseits potenziell genutzt wird, sollte dies auch getan werden, da HVBRs

keine nested virtualization unterstützen. Durch diese simple Maßnahme kann der Einsatz der geschilderten Rootkits unterbunden werden.

2.4 Schluss

Es stellt sich nun die Frage, ob Virtualisierung mehr einen Sicherheitsgewinn oder -verlust darstellt. Angesichts der klaren Steigerung der Ressourceneffizienz von Rechnern, auf denen Virtualisierung eingesetzt wird (im Vergleich zu jenen wo das nicht der Fall ist) und in Anbetracht der Tatsache, dass die hier vorgestellten Rootkits lediglich proof-of-concept sind und zur Zeit keine reale Gefahr darstellen, kann man nicht von einem Verlust reden. Da die Spezifikationen von Vitriol und Blue Pill jedoch veröffentlicht und für Forschungszwecke verfügbar sind, ist es nur eine Frage der Zeit bis eine konkrete Schadsoftware das Konzept des HVBR aufgreift und es erfolgreich umsetzt. In dem Fall wird dem besonderen Schutzbedarf, bestehend aus Vertraulichkeit, Integrität und Verfügbarkeit nicht mehr genügend Zuwendung geschenkt. Also muss vor Allem in der Zukunft im Hinblick auf Virtualisierung ein stark erhöhtes Maß an Sorgfalt dem Sicherheitsaspekt gewidmet werden, da diese Technologie sich immer größerer Popularität erfreut.[13]

Obwohl es noch keine zuverlässige Methode gibt eine Kompromittierung eines Server festzustellen, gibt es zahlreiche proaktive Ansätze die Sicherheitsstandards deutlich zu erhöhen. Dazu zählen beispielsweise Betriebssysteme, Netzwerke und Kernels mit sicheren virtuellen Maschinen auszustatten, welche keine öffentlich zugängliche application programming interface (API) haben.[13] Außerdem gibt es den Ansatz der Trusted Execution Technology (TXT), was bedeutet, dass nach jedem Schritt der Ausführung eines Prozesses geprüft wird, ob sich das Gesamtsystem noch in einem sicheren Zustand befindet. Intel hat diese Technologie entworfen um launch-time protection (Schutz zum Zeitpunkt des Start) zu gewährleisten, nicht jedoch runtime protection (Schutz zur Laufzeit). Somit können VMMs, Betriebssystemkerne oder andere wichtige Software zwar sicher gestartet werden, eine Sicherheitsgarantie zur Laufzeit gibt es jedoch nicht.[19]

Es bleibt übrig festzustellen, dass von verschiedenen Rootkitarten ein jeweils leicht abweichendes Risikopotential ausgeht. Ohne dass jedoch solide Gegenmaßnahmen gefunden werden, kann man die beschriebenen Rootkits durchaus als “Heiliger Gral” der unerkennbaren Angriffstechniken titulieren.[7]

Literaturverzeichnis

[1] Advanced Micro Devices. *AMD64 Virtualization Codenamed "Pacifica" Technology - Secure Virtual Machine Architecture Reference Manual*, Publ.No. 33047, Rev. 3.01, AMD 2005.

[2] Advanced Micro Devices. *AMD64 Technology '-' AMD64 Architecture Programmer's Manual Volume 2: System Programming*, Publ.No. 24593, Rev. 3.14, AMD 2007.

[3] Paul Barham., Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, Andrew Warfield. *Xen and the Art of Virtualization*, University of Cambridge, Cambridge 2003.

[4] Thomas Wolfgang Burger. *Intel® Virtualization Technology for Directed I/O (VT-d): Enhancing Intel platforms for efficient virtualization of I/O devices*, Intel, 2009, siehe http://software.intel.com/en-us/articles/intel-virtualization-technology-for-directed-io-vt-d-enhancing-intel-platforms-for-efficient-virtualization-of-io-devices/, 30.06.2010 14:40

[5] Dino A. Dai Zovi. *Hardware Virtualization Rootkits*, Project Chinashop siehe http://www.theta44.org/software/HVM_Rootkits_ddz_bh-usa-06.pdf, 29.06.2010 18:07

[6] Hagen Fritsch. *Analysis and detection of virtualization-based rootkits*, Bachelorarbeit, Technische Universität München, München 2008.

[7] Stefan Gora. *Stealth Malware Virtualisierungs-Rootkits*, hakin9, Nr 7/2007

[8] Benjamin Hoffmann. *Hostvirtualisierung, Vergleich der Konzepte und Produkte*, Seminararbeit, Universität der Bundeswehr, Neubiberg 2010.

[9] Katarzyna Keahey, Karl Doering, Ian Foster. *From Sandbox to Playground: Dynamic Virtual Environments in the Grid*, Paper von "Proceedings of the Fifth IEEE/ACM International Workshop on Grid Computing (GRID04)", 2004.

[10] Samuel T. King, Peter M. Chen, Yi-Min Wang, Chad Verbowski, Helen J. Wang, Jacob R. Lorch. *SubVirt: Implementing malware with virtual machines*, University of Michigan, Microsoft Research, siehe http://www.eecs.umich.edu/virtual/papers/king06.pdf, 29.06.2010 22:31

[11] Shinsuke Miwa, Toshiyuki Miyachi, Masashi Eto, Masashi Yoshizumi, Yoichi Shinoda. *Design Issues of an Isolated Sandbox Used to Analyze Malwares*, National Institute of Information and Communications Technology, Tokio 2007.

[12] Gil Neiger, Amy Santoni, Felix Leung, Dion Rodgers, Rich Uhlig. *Intel Virtualization Technology: Hardware Support for Efficient Processor Virtualization*, Intel Technology Journal, Vol.10 Issue 03, ISSN 1535-864X, Intel 2006.

[13] Edward Ray, Eugene Schultz. *Virtualization Security*, NetSec Design & Consulting, Inc., Emagined Security, Orange, California 2008.

[14] Rösner-IT. *Sicherheitsaspekte der Virtualisierung*, IT-Security, Consulting & Service, siehe http://www.roesner-it.com/Download/Sicherheitsaspekte_der_Virtualisierung.pdf, 30.06.2010 09:35

[15] Joanna Rutkowska. *Introducing Stealth Malware Taxonomy*, COSEINC Advanced Malware Labs, Version 1.01, 2006.

[16] Joanna Rutkowska. *Subverting Vista Kernel For Fun And Profit*, COSEINC Advanced Malware Labs, SysScan06, Singapore 2006.

[17] Narendar B. Sahgal, Dion Rodgers. *Understanding Intel Virtualization Technology (VT)*, Digital Enterprise Group, Intel 2006. siehe www.sdml.info/collard/se/notes/VM.ppt, 29.06.2010 18:05

[18] Vmware, Inc.. *Understanding Full Virtualization, Paravirtualization, and Hardware Assist*, White Paper, Palo Alto 2007.

[19] Rafal Wojtczuk, Joanna Rutkowska. *Attacking Intel Trusted Execution Technology*, Invisible Things Lab, siehe http://www.invisiblethingslab.com/resources/bh09dc/Attacking%20Intel%20TXT%20-%20paper.pdf, 29.06.2010 22:45