Hochschule Darmstadt

- Fachbereich Informatik -

Konzeption und Realisierung eines modularen Sicherheits-Scanners für Betriebssystem-Images in Cloud-Infrastrukturdiensten

Abschlussarbeit zur Erlangung des akademischen Grades
Master of Science (M.Sc.)

Florian Friedrich 705487
19. August 2012

Referent:	Prof. Dr. Michael Massoth
Korreferent:	Dr. Christian Baun

Erklärung

Ich versichere hiermit, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die im Literaturverzeichnis angegebenen Quellen benutzt habe.

Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder noch nicht veröffentlichten Quellen entnommen sind, sind als solche kenntlich gemacht.

Die Zeichnungen oder Abbildungen in dieser Arbeit sind von mir selbst erstellt worden oder mit einem entsprechenden Quellennachweis versehen.

Die Arbeit ist in gleicher oder ähnlicher Form noch bei keiner anderen Prüfungsbehörde eingereicht worden.

Groß-Zimmern, 19. August 2012 _____

Florian Friedrich

Abstract

Deutsch

Cloud Computing ist in den letzten Jahren immer interessanter geworden. Beliebt ist es auch, beispielsweise im Zusammenhang mit den Infrastrukturdiensten der Amazon Web Services, bereits vorgefertigte Betriebssystemimages zu nutzen. Dadurch können neue Systeme viel schneller aufgesetzt werden. Jedoch birgt dieses Vorgehen auch Gefahren für diejenigen, die solche Images bereit stellen und für die, welche diese Images später nutzen. So könnten auf den einen Seite noch private Daten in den Images vorhanden sein. Auf der anderen Seite können aber auch sicherheitskritische Fehlkonfigurationen vorhanden sein. Um diese Probleme zu minimieren, wird im Rahmen dieser Masterarbeit eine Scanner für Betriebssystemimages in Infrastrukturdiensten erstellt. Mit Hilfe dieses Scanners können solche Images vor der Veröffentlichung oder Nutzung auf eventuell vorhandene Probleme hin überprüft werden. Im Anschluss wird der Scanner anhand zufällig ausgewählter Images der Amazon Web Services getestet und gezeigt, dass nach wie vor private Daten in Images vergessen werden.

English

In the last few years, cloud computing has become more and more interesting. For example, it is also popular for example in relation to the infrastructure services of the Amazon web services using prebuild operating system images. As a result, it is possible to deploy new systems much faster. But this approach also involves dangers for all those who prepare such images and those who use such images later on. On the one hand, private data might be left on the images; on the other hand, there might be some security critical misconfigurations. To minimize these problems, a scanner for operating system images in infrastructure services will be created within the scope of this master thesis. With the help of this scanner, such images can be checked for these problems before they are released or used. Afterwards, the scanner is tested with randomly choosen images of the Amazon web services and it turns out, that there are still private data left in such images.

Danksagung

Es gibt so vielen Menschen zu danken, die mich unterstützt haben. Alleine dafür wäre eine eigene Arbeit nötig. Deshalb will ich mich hier auf die wichtigsten beschränken.

Als erstes will ich mich bei meiner Familie bedanken:

- Meine Mutter Angelika, die mich stets mit Rat und Tat unterstützt hat und den nervigen Part des Korrekturlesens übernommen hat.

- Mein Vater Georg, der mir während der Arbeit bei der Einrichtung meiner Wohnung geholfen hat.

- Mein Bruder Dominik, der mir zur Seite stand und mich immer wieder motiviert hat.

Ein ganz besonderer Dank geht an meine Freundin Erika, die ich über alles liebe. Ohne sie hätte ich die Arbeit wohl nicht geschafft und hätte auf halber Strecke aufgegeben.

Darüber hinaus ein großer Dank an die Mitarbeiter der Hochschule Darmstadt, die da wären:

- Prof. Dr. Michael Massoth, mein Referent, der mir immer wieder Vorschläge für weitere Ergänzungen gemacht hat.

- Dr. Christian Baun, mein Korreferent, welcher mir Unterstützung bezüglich Formatierung und Formulierung gegeben hat.

- Und zur guter Letzt noch ein ganz großer Dank an Torsten Wiens, der viel Zeit für mich geopfert hat, um mir jede Woche neue Anmerkungen und Vorschläge zukommen lassen zu können.

Dann noch ein Dank an alle meine Freunde der Deutschen Lebens Rettungs Gesellschaft in Babenhausen, die mich stets motiviert haben und mir den Rücken frei gehalten haben, wenn es zeitlich eng wurde.

Und zuletzt auch noch ein Dank an meinen besten Freund Markus und dessen Freundin Verena. Die gemixten House-Tracks haben mir die Arbeit erleichtert.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1. Einleitung

Die bislang existierenden Erkenntnisse über die Sicherheit von Cloud Computing-Diensten sind überschaubar. Einige Probleme und Herausforderungen sind auf die Images zurückzuführen, mit deren Hilfe die virtuellen Maschinen in Infrastrukturdiensten aufgesetzt werden. Diese Images können von den Benutzern selber erstellt und mit anderen Benutzern geteilt werden. Ebenso sind vorgefertigte Images verfügbar.

In diesem Zusammenhang können unter anderem Probleme mit der Datensicherheit entstehen. Denkbare Szenarien sind das versehentliche Veröffentlichen privater Daten über solche Images oder Schadsoftware in infizierten Images. Die Situation wird dadurch verschärft, dass nur wenige Scanner mit eingeschränktem Funktionsumfang existieren, um solche Sicherheitsprobleme zu entdecken.

1.1. Aufgabenstellung

Im Rahmen dieser Masterarbeit wird ein Scanner zum Überprüfen von Images in Infrastrukturdiensten entwickelt. Zunächst erfolgt eine Definition der sicherheitsrelevanten Probleme im Zusammenhang mit Images. Im Anschluss wird ein Konzept für einen entsprechenden Scanner erarbeitet. Der Scanner wird modular aufgebaut, um eine gute Erweiterbarkeit zu gewährleisten. Zudem wird der Scanner flexibel einsetzbar und anpassbar werden, was beispielsweise durch den Einsatz von Datenbanken statt fest codierten Zeichenketten erreichbar ist. Im Anschluss erfolgt die Implementierung des Scanners und das Testen. Abschließend wird die Leistungsfähigkeit des Scanners evaluiert und mit Werkzeugen, mit ähnlichen Zielsetzungen, verglichen.

1.2. Wirtschaftlicher Nutzen

Cloud Computing wurde im Laufe der letzten Jahre immer beliebter. Immer mehr Daten werden von firmeneigenen Systemen in die Cloud verschoben, um Kosten einzusparen. Viele dieser Firmen haben jedoch Bedenken, was die Sicherheit Ihrer Daten anbelangt. Es ist daher wichtig, die Sicherheit in der Cloud stetig zu verbessern. Dadurch könnten auch Firmen, welche auf Grund von Sicherheitsbedenken, noch nicht die Cloud nutzen, in Zukunft dazu angeregt werden.

1.2.1. Studie von AMD Research

Im Mai 2011 veröffentlichte AMD Research hierzu eine Studie [AMD11] über das Interesse von Unternehmen und deren Befürchtungen in Bezug auf das Cloud Computing [Pic11].

Laut dieser Studie nutzen ungefähr 70 % der befragten Unternehmen Cloud-Dienste oder evaluieren zumindest die Möglichkeiten. 63% der Unternehmen, welche Cloud-Dienste einsetzen, speichern Daten im Wert von $250.000 und mehr dort. 9% speichern Daten im Wert von über $10.000.000 in Cloud-Diensten.

Als größte Vorteile von Cloud-Diensten sehen Unternehmen die Flexibilität (50%), reduzierte Hardwarekosten (49%) und höhere Effizienz (46%).

Zahlreiche Unternehmen sehen Risiken bei der Nutzung von Cloud-Diensten. Bei diesen Risiken handelt es sich um Sicherheitsbedenken wie Viren, Hacker und Ähnliches (63%). Dahinter folgt der befürchtete Verlust von Daten (45%) und die Abhängigkeit von der Internetverbindung (34%).

Ein ähnliches Bild stellt sich auch bei den Unternehmen dar, welche nicht vor haben, Cloud-Dienste zu nutzen. So geben 20%-26% der befragten Unternehmen, die Cloud-Dienste nicht nutzen wollen an, dass sie diese Dienste auf Grund von Sicherheitsbedenken nicht nutzen.

Die Studie zeigt, dass zahlreiche Unternehmen Cloud-Dienste einsetzen und wertvolle Daten dort speichern. Zudem zeigt die Studie, dass Bedenken bezüglich der Sicherheit der Daten existieren. Darum ist es wichtig, die Sicherheit der Cloud-Dienste zu erforschen, um diese zu verbessern.

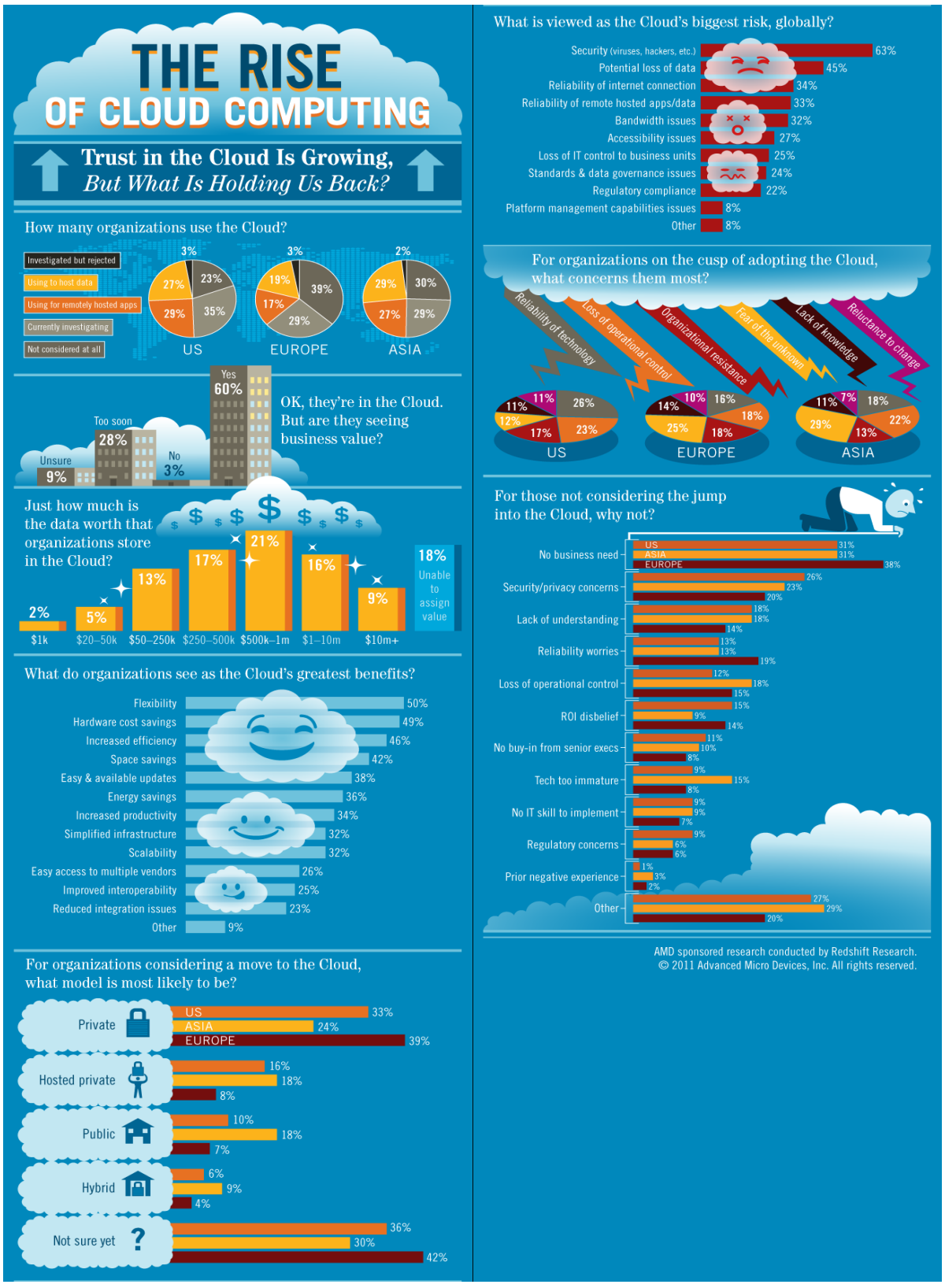

Abbildung 1.1.: AMD 2011 Global Cloud Computing Adoption, Attitudes and Approaches Study: Infographics

1.2.2. Handel mit gestohlenen Daten

In Untergrundforen wird bereits heute mit privaten Daten gehandelt. Hierbei geht es zwar hauptsächlich um Kreditkartendaten, doch auch Passwörter und andere Zugangsdaten werden gerne ausgetauscht.

Am Beispiel von Kreditkartendaten wurden im Jahre 2011 von Panda Security eine Recherche zu den Preisen solcher Daten durchgeführt [Sch11]. Hier zeit sich auch der Wert solcher Daten. Für Daten, mit denen sich mindestens 82.000 Euro erwirtschaften lassen, müssen 700 Euro gezahlt werden. Dieser Markt könnte auch in Zukunft auf wertvolle Zugangsdaten zu Cloud-Infrastrukturdiensten erweitert werden, da hier, wie die Studie von AMD Research zeigt, oft noch wertvollere Daten liegen.

1.3. Zusammenfassung

Dieses Kapitel hat gezeigt, dass die Cloud bereits heute ein wichtiger wirtschaftlicher Faktor in der IT ist. Jedoch existieren auch Risiken in Bezug auf die Datensicherheit. Dies ist insofern problematisch, da teilweise sehr wertvolle Daten in der Cloud gespeichert werden. Daher ist es das Ziel dieser Masterarbeit, solche Risiken zu minimieren.

2. Sicherheit beim Cloud Computing

„Als Cloud Computing bezeichnet man die Idee, komplette IT-Infrastrukturen abstrakt darzustellen und dynamisch an den eigenen Bedarf anpassen zu können. Hierbei gehören zur IT-Infrastruktur sowohl Rechenleistung, Datenspeicher als auch Netzwerkkapazität. Teilweise wird auch Software hinzugezählt. Der Name leitet sich von der Tatsache ab, dass die abstrahierte IT-Infrastruktur aus Sicht des Anwenders eine unscharfe Wolke darstellt.“ [MK12]

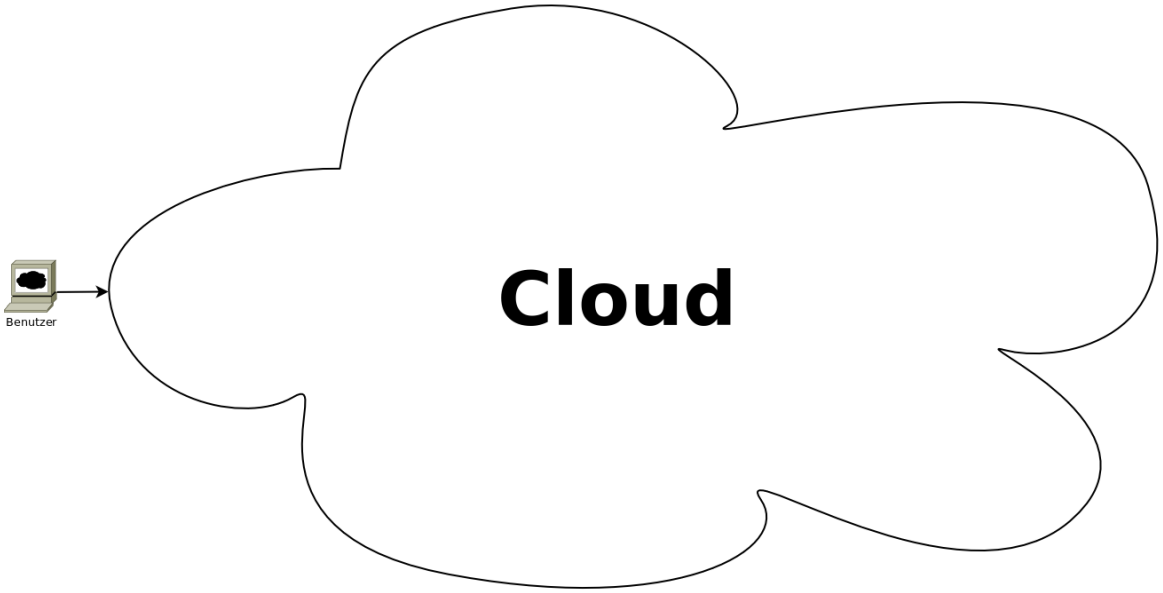

Abbildung 2.1.: Cloud-Übersicht aus Sicht des Benutzers

Im Grund handelt es sich beim Cloud Computing um Dienste, mit verschiedenen Eigenschaften:

- Ein Dienst wird nach außen hin als Einheit repräsentiert. Der Benutzer bzw. Kunde muss

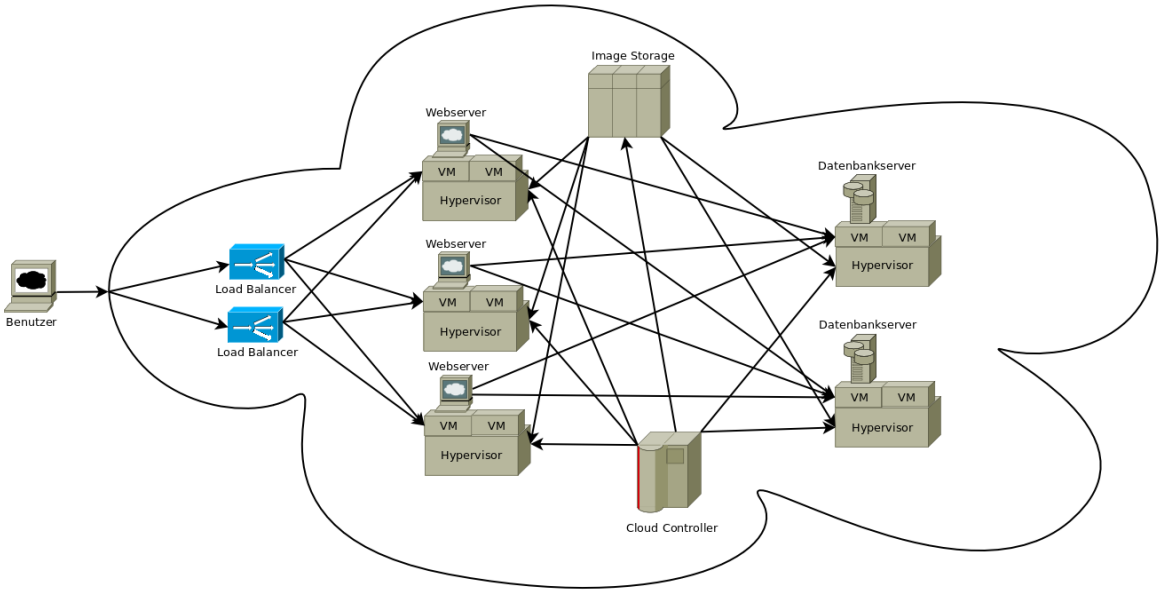

Abbildung 2.2.: Cloud-Übersicht aus Sicht des Betreibers

sich nicht mit der Komplexität an Systemen hinter dem Dienst beschäftigen.

- Intern ist der Dienst über mehrere Systeme verteilt, mit dem Ziel einer hohen Skalierbarkeit.

- Ein Dienst ist hochverfügbar und dadurch fast ausfallsicher. Dies ist laut dem Bundesamt für Sicherheit in der Informationstechnik ab einer Verfügbarkeit von mindestens 99,99% der Fall [fSidI09].

- Ein Dienst kann kurzfristig an veränderte Gegebenheiten, wie zum Beispiel Belastungsspitzen oder neue Softwareversionen, angepasst werden. Dieses wird durch die Kombination von guter Skalierbarkeit und Hochverfügbarkeit gewährleistet.

Darüber hinaus werden beim Cloud-Computing 3 verschiedene Ausprägungen unterschieden [MK12]. Diese bauen, wie in Bild 2.1 gezeigt, aufeinander auf.

- IaaS - Infrastructure as a Service - IaaS ist die niedrigste Schicht beim Cloud Computing. Hierbei werden dem Benutzer lediglich virtuelle Maschinen zur Verfügung gestellt. Dadurch ist der Benutzer frei in der Wahl des Betriebssystems und der darauf installierten Software. Beispiele für IaaS sind die Amazon Web Services [Ama12], Eucalyptus [Sys12]

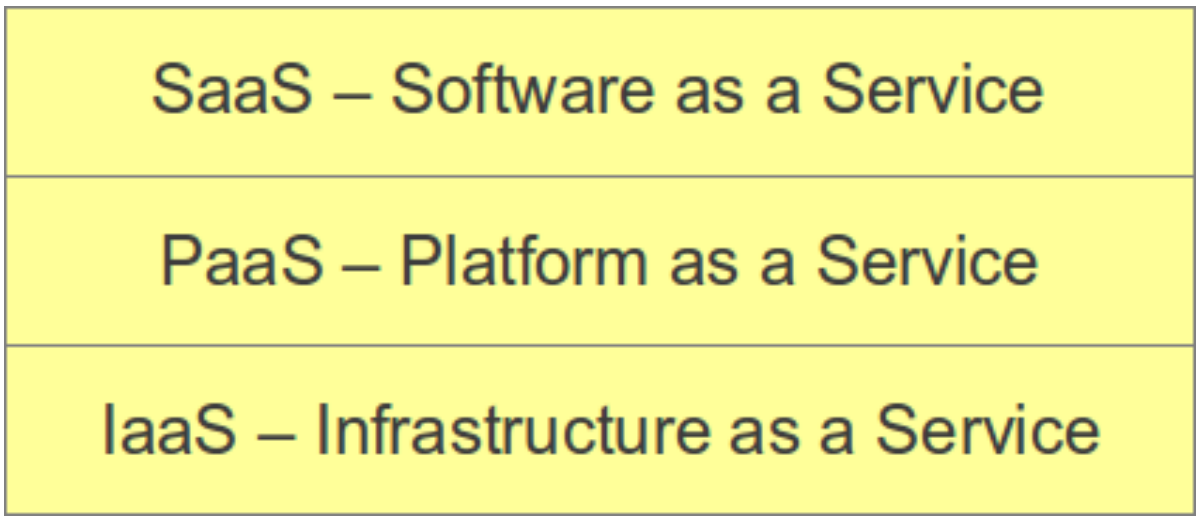

Abbildung 2.3.: Ausprägungen des Cloud-Computing

und OpenStack [Ope12]. IaaS ist im Rahmen dieser Arbeit die wichtigste Ausprägung.

- PaaS - Platform as a Service - PaaS geht noch einen Schritt weiter, als IaaS. Hierbei wird auch eine Laufzeitumgebung für die Software des Benutzers bereitgestellt. Dies kann ein Betriebssystem oder eine entsprechende Entwicklungsplattform sein. Hier kann dann die Software des Benutzers installiert bzw. entwickelt werden. Beispiele für PaaS sind die Anbieter Force.com [For12] und die Google App Engine [Goo12a].

- SaaS - Software as a Service - SaaS wird immer beliebter und stellt dem Benutzer eine Software als Dienst zur Verfügung. So muss sich der Benutzer nicht um die Konfiguration komplexer Rechner- und Betriebssystemlandschaften kümmern, welche sonst bei größeren Softwaresystemen nötig wäre. Ein Beispiel für SaaS ist Office356 von Microsoft [Mic12].

2.1. Anbieter

In den letzten Jahren haben sich verschiedene Anbieter im Bereich des Cloud Computing etabliert. Insbesondere im IaaS-Bereich gibt es mehrere große Anbieter. Die größten dieser Anbieter hat das unabhängige Informationsportal und Anbieterverzeichnis clouds.de am 16. April 2012 aufgelistet [Clo12].

2.1.1. Amazon

Amazon ist der mit Abstand größte Cloud-Provider. Der Infrastrukturdienst EC2 der Amazon Web Services [Ama12] läuft auf schätzungsweise mehr als 450.000 physikalischen Maschinen. Der Umsatz beläuft sich schätzungsweise auf über 1 Milliarde Dollar, bei einer Wachstumsrate von etwa 70%. Jedoch gibt Amazon in diesem Bereich keine genauen Umsatzzahlen bekannt. Es wird geschätzt, dass sich dieser Umsatz bis zum Jahr 2014 auf bis zu 2,5 Milliarden Dollar belaufen könnte.

2.1.2. Rackspace

Der im texanischen San Antonio beheimatete Anbieter Rackspace ist zu einer bedeuteten Größe auf dem Markt des Cloud Computing geworden. Viele Experten sehen Rackspace auf dem zweiten Platz nach Amazon. Die im Bereich Cloud Computing generierten Umsätze beliefen sich im Jahr 2011 auf 189 Millionen Dollar. Dies entspricht einem Wachstum von 89% gegenüber dem Vorjahresumsatz von 100 Millionen Dollar. Rackspace ist ein maßgeblicher Unterstützer des OpenStack-Standards und der gleichnamigen Open Source-Softwareimplementierung [Ope12].

2.1.3. Google

Nach der Anzahl der Server spielt auch Google eine große Rolle im Cloud Computing. Insbesondere durch die Infrastruktur von Google Mail und anderen Diensten. Jedoch sind viele der Nutzer dieser Dienste keine zahlenden Kunden. Die Dienste werden hauptsächlich durch Werbeeinblendungen finanziert. Es gibt zwar einen PaaS-Service namens Google App Engine, welcher jedoch nur von einer geringen Anzahl Benutzern genutzt wird. Dies kann sich jedoch in den nächsten Jahren ändern, da Google im Juni 2012 einen eigenen IaaS-Dienst namens Google Compute Engine [Goo12b] vorgestellt hat [Han12].

2.1.4. Microsoft

Microsoft bietet seit etwa 2 Jahren mit Azure [Azu12] eine eigene Cloud-Plattform an. Sie hat sich gut etabliert und ist durch die Integration von hauseigenen ERP-Lösungen attraktiv. Auch die Anbindung an das Framework Apache Hadoop [Had12], welches für die Entwicklung von skalierbarer, verteilt arbeitender Software genutzt werden kann, dürfte Azure gerade bei Providern im Big Data Segment interessant machen.

2.1.5. VMWare

VMWare [VMw12] ist in erster Linie ein Hersteller von Virtualisierungs-Software. Allerdings läuft die Software vCloud des Unternehmens bereits bei zahlreichen Drittanbietern. Darüber hinaus veröffentlichte VMware mit CloudFoundry kürzlich einen wettbewerbsfähigen PaaS-Dienst. Außerdem hält VMware die Vormachtstellung im Bereich Server Virtualisierung. Dies macht VMware zu einem wichtigen Unternehmen im Bereich Cloud Computing.

2.2. Infrastruktur

Mit Hilfe der Infrastruktur läuft die Cloud und die darauf aufbauenden Dienste. Jedoch sind auch für manche Komponenten der Infrastruktur bereits Schwachstellen bekannt.

2.2.1. Cloud Controller

Der Cloud Controller leitet bei Infrastrukturdiensten die Anweisungen zum Starten oder Stoppen der virtuellen Maschinen an die physischen Knoten weiter. Zudem verwaltet er die verfügbaren Storage-Systeme. Ein erfolgreicher Angriff auf den Cloud Controller kann daher fatale Folgen haben, bis hin zur kompletten Übernahme der Cloud-Infrastruktur durch den Angreifer. Die Dienstnutzer kommunizieren zum Beispiel bei den Amazon Web Services (AWS) [Ama12] und der freien Cloud-Lösung Eucalyptus [Sys12] via signierten SOAP-Nachrichten mit dem Cloud Controller.

XML-Wrapping

Beim XML-Wrapping werden SOAP-Nachrichten in der Art und Weise abgewandelt, dass die Signatur der Nachricht noch gültig ist, aber gleichzeitig modifizierte Befehle vom Cloud Controller ausgeführt werden. Ein solcher Angriff auf die Cloud von Amazon und die freie Cloud-Lösung Eucalyptus wurde 2011 in dem Paper „All Your Clouds are Belong to us“ [Som11] beschrieben.

2.2.2. Hypervisor

Der Hypervisor ist eine weitere kritische Komponente in einer Cloud. Er kontrolliert die Ausführung der virtuellen Maschinen auf der physischen Hardware. Verschiedene Angriffe sind auf den Hypervisor denkbar. [Fer07]

Bei einer Detektions-Attacke versucht der Angreifer heraus zu finden, ob und mittels welchem Hypervisor eine virtuelle Maschine läuft. Dieses kann erste Anhaltspunkte für weitere Attacken geben. Interessant ist diese Attacke unter anderem für Viren und Trojaner. Diese können durch eine solche Attacke eventuell erkennen, ob sie in einem Emulator ausgeführt werden und in einem solchen Fall den Start verweigern. Dadurch können diese von Virenexperten nur schwer analysiert werden, da diese für Ihre Analysen oft Emulatoren benutzen. Diese Attacke ist von Emulator zu Emulator unterschiedlich. So kann man einen QEMU-Emulator bereits anhand des Prozessornamens erkennen. Andere Emulatoren, wie beispielsweise Virtualbox benötigen etwas mehr Fachwissen um den Emulator zu erkennen. [Fer07]

Auch Denial of Service-Attacken auf den Hypervisor sind möglich. Je nach Design des Hypervisors kann diese Attacke mehr oder weniger schwer ausfallen. Eventuell wird dadurch nur die virtuelle Maschine beeinträchtigt, auf welcher die Attacke ausgeführt wird. Es ist aber auch denkbar, dass auch andere Maschinen auf dem gleichen physischen Knoten oder der gesamte physische Knoten selbst in Mitleidenschaft gezogen werden. Eine mögliche Auswirkung ist der Ausfall des kompletten Hostsystems mit allen darauf laufenden virtuellen Maschinen. Dieses war beispielsweise durch bestimmte Instruktionen bei der Virtualisierungslösung XEN bis einschließlich Version 3.3 möglich. [st11]

Ein kritisches Szenario ist es, wenn es eine virtuelle Maschine schafft, aus ihrer virtuellen Umgebung auszubrechen und Zugriff auf das darunterliegende physische System zu erhalten. Dieses

kann zum Beispiel durch Fehler in den virtualisierten Treibern geschehen. Ein solcher Fehler wurde beispielsweise 2005 im Netzwerktreiber von VMWare entdeckt. [She05]

2.2.3. Image Storage

Der Image Storage ist ein wichtiges System im Rahmen eines Cloud-Infrastrukturdienstes. Dieses System speichert Betriebssystemimages und stellt diese, bei Bedarf, zur Verfügung um neue virtuelle Maschinen aufsetzen zu können. Auf diesen Images befindet sich bereits ein Betriebssystem und teilweise auch Dienste. Dadurch kann ein neues System schnell aufgesetzt werden, da die Installation weg fällt.

Die Befehle nimmt der Image Storage vom Cloud Controller entgegen. Dieser entscheidet, ob ein Image auf dem Storage gespeichert wird oder auf welche physikalische Maschine es transferiert wird, wenn eine neue virtuelle Maschine gestartet werden soll.

Es ist darüber hinaus bei größeren Cloud-Infrastrukturdiensten, wie den Amazon Web Services, möglich, die eigenen Images anderen Nutzern zur eigenen Verwendung zur Verfügung zu stellen. Hierbei entstehen aber oft Probleme mit der Datensicherheit.

Private Daten in Cloud-Betriebssystemimages

In öffentlichen Cloud-Infrastrukturdiensten stehen für die Benutzer vorgefertigte Images bereit, um kurzfristig virtuelle Systeme starten zu können. Auch die Benutzer können zum Beispiel bei den Amazon Web Services eigene Images erstellen, in den Cloud-Dienst importieren und diese Dienste anderen Benutzern verfügbar machen. In diesen Images können sich noch private Daten befinden oder können wieder hergestellt werden.

Das Fraunhofer-Institut für Sichere Informationstechnologie (SIT) veröffentlichte zu dieser Problematik im Juni 2011 eine Pressemitteilung:

„Wissenschaftler des Darmstädter Forschungszentrums CASED haben im Juni 2011 große Sicherheitsmängel in zahlreichen virtuellen Maschinen in der Amazon-Cloud entdeckt. Von 1100 untersuchten öffentlichen Amazon Machine Images (AMIs), auf denen Cloud-Dienste basieren, waren rund 30 Prozent so verwundbar, dass Angreifer teilweise Webservices oder virtuelle

Infrastrukturen hätten manipulieren oder übernehmen können.“ [fSI11]

Hierzu wurde ein Paper mit ausführlicheren Informationen und Lösungsansätzen veröffentlicht [AMI11a].

2.3. Typische Dienste in der Cloud

Nachdem bereits die Infrastruktur der Cloud betrachtet wurde, werden nun noch typische Dienste betrachtet, welche auf der Cloud aufbauen. Oft sind dies Dienste, welche auch schon ohne Cloud genutzt wurden, jedoch durch die Cloud noch leistungsstärker werden.

2.3.1. Load Balancer

Ein typischer Dienst in der Cloud ist ein Load Balancer. Dieser dient zur Lastverteilung der eingehenden Anfragen auf mehrere Server. Daher ist die Lastverteilung eine wichtige Komponente, um die Vorteile der Cloud nutzen zu können.

Neben dem DNS-Basierten Verfahren zur Lastverteilung, bei dem im DNS-System zu einer Domain mehrere IP-Adressen hinterlegt werden, gibt es auch die Lastverteilung mit Hilfe einer Hardware- oder Softwarekomponente. Diese leitet dann die Anfragen an die entsprechenden Server weiter und kann dadurch die Last auf die einzelnen Server verteilen [Loa12].

2.3.2. Webserver

Webserver sind ein typisches Anwendungszenario für Cloud-Dienste. Je nach Last können kurzfristig zusätzliche virtuelle Systeme zur Verfügung gestellt werden. Es existieren verschiedene Softwarelösungen um Webserver zu realisieren.

Die populärsten Webserver waren, laut netcraft.com, im Januar 2012 Apache mit einer Verbreitung von 57,93%, nginx mit 12,18% und der Microsoft IIS mit 12,14%. [Net12]

Durch die Komplexität der Webserver sind Sicherheitslücken nicht unüblich. Laut der Open Source Vulnerability Database wurden im Jahr 2011 im Zusammenhang mit dem Apache Webserver 49, mit nginx eine und mit dem Microsoft IIS ebenfalls eine Sicherheitslücke veröffentlicht. [Osv12]

Es ist also empfehlenswert, die Serversoftware in regelmäßigen Abständen zu aktualisieren, um Angreifern keine bekannten Angriffsmöglichkeiten zu bieten.

2.3.3. Datenbankserver

Auch Datenbankserver sind typische Cloud-Dienste. Diese können verschiedenste Aufgaben in der Cloud übernehmen. Zum Beispiel können die Datenbankserver die Daten für die Webserver speichern. Aber auch die Verarbeitung großer Datenmengen kann in der Cloud, aufgrund größerer Ressourcen, erfolgen.

Es gibt eine ganze Reihe verschiedenster Datenbankserver.

2.3.4. Dateiserver

Auch Dateiserver sind typische Cloud-Dienste. Mit Hilfe der Dateiserver können große Datenmengen in der Cloud gespeichert werden und von überallk auf der Welt zugänglich gemacht werden. Viele Cloud-Dienste, welche heute von Privatpersonen genutzt werden, dienen dem Speichern von Daten in der Cloud. Dazu gehören beispielsweise Dropbox [Dro12] oder auch Google Drive [Goo12c].

2.4. Zusammenfassung

Dieses Kapitel hat gezeigt, dass die Cloud ein sehr komplexes System mit vielen unterschiedlichen Komponenten ist. Durch diese Komplexität können jedoch auch Risiken entstehen. Teilweise erzeugen die Vorteile der Cloud, wie die Möglichkeit, Betriebssystemimages mit anderen Entwicklern zu teilen, neue Risiken, insbesondere in Bezug auf die Datensicherheit.

3. Problemstellung

Images sind eine einfache Lösung, um schnell eine Instanz in der Cloud starten zu können. Hierbei wird ein System bereits auf einer virtuellen Festplatte vorgefertigt, um dann als Vorlage für Instanzen genutzt werden zu können. Es können sowohl eigene Images anderen Benutzern zur Verfügung gestellt werden, als auch Images, die andere Benutzer zur Verfügung stellen, genutzt werden. Hierbei können aber mehrere Probleme auftreten, die die Datensicherheit beeinflussen können.

Sollte man ein Image anderen Benutzern zur Verfügung stellen, kann es passieren, dass man vergisst, private Daten, wie private SSL-Schlüssel zu löschen. Außerdem können eventuell bereits gelöschte Daten wieder hergestellt werden, wenn sie nicht ordnungsgemäß überschrieben wurden.

Wenn man ein Image von einem Anderen, eventuell fremden, Benutzer für seine eigenen Instanzen nutzt, stellen sich andere Probleme dar. So kann es zum Beispiel sein, dass das Image mit Schadsoftware, wie Rootkits infiziert ist. Des weiteren kann es sein, dass in dem Image eventuell veraltete Software enthalten ist, welche Sicherheitslücken aufweist. Zusätzlich können einzelne Konfigurationseinstellungen schlecht gewählt sein und die Instanz dadurch angreifbar werden.

Im Folgenden soll auf die einzelnen Punkte nochmal detailliert eingegangen werden.

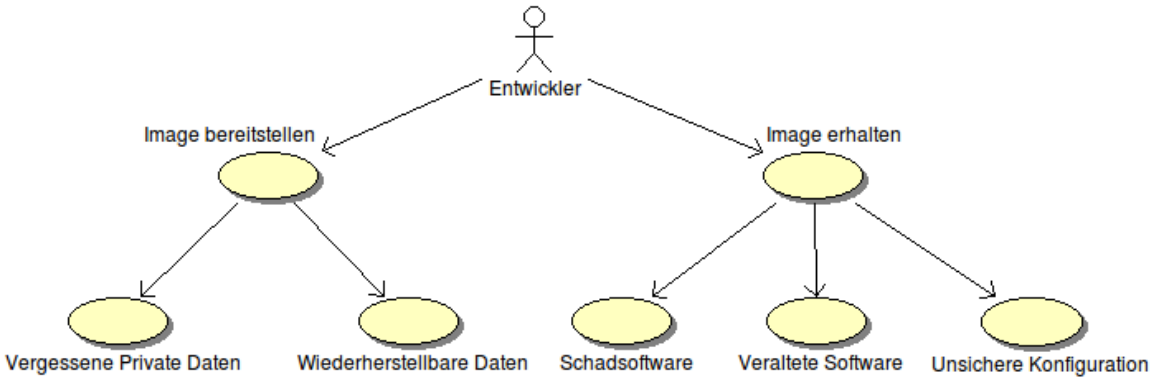

Abbildung 3.1.: Übersicht über Probleme mit Cloud-Images

3.1. Vergessene private Daten

Oft werden Images im Vorfeld vorkonfiguriert. Hierzu müssen beispielsweise temporäre Benutzer angelegt werden und zusätzliche Software installiert und konfiguriert werden. Im Anschluss wird das Image verpackt und veröffentlicht.

Immer wieder wird hierbei jedoch versäumt, private Daten zu löschen, welche für die Konfiguration genutzt wurden. Dies können private Schlüssel oder auch Passwörter sein. Diese Dateien müssen vor der Veröffentlichung sorgfältig entfernt werden, um einen Missbrauch nach der Veröffentlichung zu verhindern.

3.2. Wiederherstellbare Daten

Eine besondere Form vergessener Privater Daten stellen wiederherstellbare Daten dar. Hier wurden private Daten zwar gelöscht, aber nicht überschrieben. Somit können diese Dateien, mit mehr oder weniger Aufwand, wieder hergestellt werden. Daher sind diese Daten genau so kritisch zu betrachten, wie private Daten, die nicht gelöscht wurden. Es muss sichergestellt werden, dass private Daten nicht nur gelöscht, sondern auch überschrieben werden. Dies kann sowohl bei der Löschung selbst, als auch erst später geschehen, indem der freie Speicher überschrieben wird.

3.3. Kompromittierte Images

Mit Hilfe von Rootkits können sich Angreifer unbemerkt Zugang zu einem System verschaffen. Es wäre beispielsweise möglich, ein Rootkit in ein Image zu integrieren und anschließend zu veröffentlichen. Jeder, der das Image im Anschluss nutzt, ist von vornherein infiziert. Da das Rootkit sich selbst gegen eine Entdeckung im laufenden Betrieb schützt, ist es ratsam, das Image von außen nach Rootkits zu scannen, damit der Schutzmechanismus des Rootkits nicht aktiv werden kann.

3.4. Angreifbare Software

Beispielsweise bei den Amazon Web Services gibt es viele vorgefertigte Images, welche ein Anwender für seine Instanz nutzen kann. Jedoch existieren die meisten schon länger, so dass die Software in der Regel bereits veraltet ist. Oft lässt sich die Software über den Paketmanager aktualisieren. Jedoch kann es sein, dass dies versäumt wird oder die Distribution bereits so alt ist, dass diese keine aktuellen Updates mehr enthält. Dadurch kann das Image anfällig für Exploits gegen die darauf installierte Software sein. So wird zum Beispiel die Ubuntu-Distribution *Maverick* seit dem 10. April 2012 nicht mehr mit Aktualisierungen versorgt [Ubu12a]. Die aktuelle Ubuntu-Version *Precise* wird hingehen, aufgrund eines verlängerten Support-Zeitraums, noch bis zum 26. April 2017 Aktualisierungen erhalten [Ubu12b].

3.5. Unsichere Konfigurationseinstellungen

Wie auch bei der verwendeten Software, kann man sich auch bei der Konfiguration nicht auf den Entwickler des Images verlassen. Es können diverse Einstellungen gesetzt sein, welche die Software unsicher machen. Hierzu zählen zum Beispiel Einstellungen, welche den Mail-Server als Relay konfigurieren. Oder fehlerhafte Netzwerk- und Firewall-Einstellungen.

Eine besondere Form unsicherer Konfigurationseinstellungen sind Konfigurationseinstellungen, welche Unbefugten, ähnlich einem Rootkit, Zugriff auf das Image gewähren. Dazu zählen zum Beispiel zusätzliche Benutzeraccounts, vordefinierte Passwörter oder autorisierte SSH-Schlüssel.

Wenn man selbst ein Image erstellt, können auch Konfigurationseinstellungen zu vergessenen Privaten Daten zählen. Beispielsweise, indem vergessen wurde, Passwörter wieder aus Konfigurationsdateien zu entfernen. Oder indem eigene Hostnamen oder Benutzernamen in den Konfigurationseinstellungen genutzt wurden.

3.6. Zusammenfassung

Dieses Kapitel hat die einzelnen Risiken, im Zusammenhang mit Betriebssystemimages bei Cloud-Infrastrukturdiensten, noch einmal genauer beleuchtet. Es wurde gezeigt, dass die Nutzung dieser Images eine Reihe verschiedenster Risiken beinhaltet. Dies betrifft sowohl den Entwickler, der das Image zur Verfügung stellt, als auch den Entwickler, der dieses Image später nutzt.

4. Verwandte Arbeiten

Bereits in der Vergagenheit wurden Tools entwickelt, um die Sicherheitsprobleme im Zusammenhang mit Betriebssystem-Images zu minimieren. Eine Reihe dieser Tools werden in diesem Kapitel kurz vorgestellt.

4.1. AMI aiD

Das Tool AMI aiD, kurz AMID, wurde 2011 im Rahmen des Papers "AmazonIA: When Elasticity Snaps Back" [AMI11a] von Sven Bugiel, Stefan Nürnberger, Thomas Pöppelmann, Ahmad-Reza Sadeghi und Thomas Schneider für das Center for Advanced Security Research Darmstadt (CASED), TU Darmstadt entwickelt. [AMI11c]

Im Folgenden wird kurz wiedergegeben, wie die Funktionalität des Tools im zugehörigen Paper [AMI11a] beschrieben wird und anschließend, welche Funktionalität im veröffentlichten Code [AMI12] vorhanden ist.

4.1.1. Funktionalität laut Paper

Es nutzt Python als Programmiersprache und scannt Amazon-Images nach sicherheitsrelevanten Dateinamen. Für den Zugriff auf die Amazon Web Services kommt die Boto-Bibliothek zum Einsatz [Bot12].

Der Scanner selbst wird in einer eigenen Instanz betrieben. Soll nun ein Image gescannt werden, wird hierzu eine neue Instanz mit diesem Image erstellt und anschließend die Festplatte

der Instanz an die Scanner-Instanz angebunden. Nun kann der Scanner das Dateisystem nach Auffälligkeiten scannen.

Der Scanner scannt nach auffälligen Dateien im Dateisystem. Hierzu gehören private Schlüssel, Zertifikate, die Bash-Historie und Quelltext-Repositories. Eine weitergehende Analyse, wie beispielsweise der Suche nach Exploits oder Rootkits findet nicht statt.

4.1.2. Funktionalität laut Code

Nicht jede Funktionalität, welche im Paper beschrieben wird, ist jedoch im Tool auch wirklich vorhanden (Stand: Mai 2012).

So fehlt die komplette Funktionalität, um Images auf den Amazon Web Services automatisiert zu scannen. Dadurch ist es dem Scanner auch nicht möglich, ein Image von außen zu scannen, sondern lediglich manuell von innen.

Auch fallen bei einem Blick auf den Quelltext noch ein paar weiter Details auf. So stehen die Patterns der Dateinamen beispielsweise direkt im Code. Auch ist der Scanner nicht modular erweiterbar, da dieser lediglich aus einer einzelnen Python-Datei besteht. In dieser wurde die komplette Funktionalität des Tools implementiert.

4.2. AMI-Exposed

Das Tool AMI-Exposed wurde im Jahr 2011 auf der Sicherheitskonferenz Defcon im Rahmen des Vortrags "Get Off of My Cloud: Cloud Credential Compromise and Exposure" vorgestellt. [AMI11b]

Es verfolgt einen ähnlichen Ansatz, wie auch AMID. Jedoch ist es etwas umfangreicher und modularer. So ist es mit dem Tool auch möglich, Befehle in einer Instanz abzusetzen, um Informationen aus einer laufenden Instanz zu erhalten. Dies wird genutzt, um eine Liste aktiver Verbindungen zu erhalten.

AMI-Exposed ist in Ruby [Rub12] geschrieben. Auch AMI-Exposed kann auf die Amazon Web

Services zugreifen. Hier kommt das AWS SDK für Ruby [AWS12b] zum Einsatz.

AMI-Exposed ist darauf ausgelegt, eine große Anzahl an Images parallel zu scannen. Hierzu werden die Images als neue Instanzen bei den Amazon Web Services gestartet. Im Gegensatz zu AMID laufen aber alle Tests in der laufenden Instanz ab. Hierzu verbindet sich das Tool mit Hilfe von SSH mit der laufenden Instanz und analysiert das Image mit Hilfe von Systembefehlen, wie *find*, *grep* oder *netstat*.

Die einzelnen Tests sind in Module ausgelagert. Dadurch ist AMI-Exposed leichter erweiterbar, als beispielsweise AMID. Es werden bereits einige Tests mitgeliefert:

- Suche nach vergessenen AWS-Zertifikaten und Schlüsseln.
- Überprüfen ausgehender Verbindungen, um Backdoors aufzuspüren.
- Suche nach Bash-Historien (*.bash_history*) mit eingegebenen Befehlen.
- Suche nach Bash-Profilen (*.bash_profile*, *.bashrc* und */etc/profile.d/*) mit eventuell privaten Einstellungen
- Suche nach autorisierten SSH-Schlüsseln (*authorized_keys*), welche sich mit der Instanz verbinden können.
- Suche nach privaten RSA- und DSA-Schlüsseln.
- Überprüfung, ob der SSH-Server eine Passwort-basierte Authentifizierung zulässt.
- Suche nach Historien des Texteditors Vim (*.viminfo*; das Tool sucht jedoch fälschlicherweise nach *.vim_info*)

Eine Suche nach Exploits findet auch hier nicht statt.

Rootkits können nur über den Test für ausgehende Verbindungen aufgespürt werden. Da das Tool jedoch auf dem Image selbst läuft, ist solch einer Analyse jedoch nicht zu trauen, da das Rootkit diese Informationen manipulieren kann.

4.3. Rootkit-Scanner

Rootkit-Scanner sind kleine Tools, welche sich auf das Aufspüren von Rootkits unter Linux spezialisiert haben. Dadurch sind diese auch für Cloud-Images von Interesse, um die Gefahr einer Hintertür in dem System zu verringern.

Bekannte Rootkit-Scanner sind beispielsweise RKHunter [RKH12] und Chkrootkit [Chk12]. Diese nutzen verschiedene Ansätze um nach Rootkits zu suchen. Dazu gehören die Suche nach verdächtigen Dateien oder die Überprüfung geladener Module. RKHunter besitzt darüber hinaus die Fähigkeit, Checksummen der wichtigsten Systemdateien zu erstellen und diese damit anschließend nach Veränderungen zu überprüfen.

4.4. Secure-Delete Tools

Secure-Delete Tools sind kleine Hilfsprogramme, um Dateien sicher zu löschen. Dadurch können diese anschließend nicht wieder hergestellt werden.

Bei der Entwicklung eines Cloud-Images ist darauf zu achten, dass private Dateien anschließend sicher gelöscht werden. Daher können solche Tools die Sicherheit solcher Images für den Entwickler erhöhen, da es für spätere Nutzer nicht mehr möglich ist, bei der Entwicklung genutzte private Daten wieder herzustellen.

Es gibt auch Secure-Delete Tools, welche noch weiter gehen, als nur Dateien sicher zu löschen. So können zum Beispiel manche Tools auch den freien Speicher überschreiben. Dadurch werden bereits gelöschte Dateien endgültig gelöscht und können nicht mehr rekonstruiert werden.

Ein Beispiel für ein Secure-Delete Tool ist das Tool THC-SecureDelete [Sec12b].

4.5. Metasploit

Bei Metasploit [Met12] handelt es sich um ein Framework, mit dem man Exploits ausführen kann. Dadurch kann ein System auf Schwachstellen getestet werden.

Die Liste der verfügbaren Exploits wird regelmäßig aktualisiert. Dadurch kann man mit Hilfe von Metasploit auch Cloud-Images auf Schwachstellen hin testen. Jedoch nur auf solche, für die Exploits existieren.

Durch die Möglichkeit, mehrere IP-Adressen auf einmal anzugreifen ist hiermit auch ein Batch-Scan möglich.

Darüber hinaus bietet Metasploit jedoch keine weiteren Funktionen, um die Sicherheit von Cloud-Images zu erhöhen.

4.6. Vergleich

In diesem Abschnitt, sollen die vorgestellten Tools mit einander verglichen werden. Dadurch kann sich ein besserer Überblick verschafft werden.

4.6.1. Kriterien

In diesem Abschnitt sollen die einzelnen, verwandten Arbeiten, mit dem geplanten Scanner verglichen werden. Hierzu ist zuerst eine Analyse der Kriterien nötig, nach denen verglichen werden soll.

- Modularer Aufbau - Ist die Software modular aufgebaut? Kann die Software also später auch durch Dritte leicht erweitert werden, ohne den Hauptteil selbst ändern zu müssen?

- Datenbankanbindung - Besitzt die Software eine Datenbankanbindung, um die Patterns für die Scans zu speichern? Dadurch können auch Laien später neue Patterns einfügen ohne den Quelltext ändern zu müssen.

- Freispeicher-Überschreiber - Kann das Tool den Freien Speicher überschreiben, um bereits gelöschte Dateien endgültig zu entfernen?

- Rootkit-Scanner - Kann das Tool nach Rootkits-Scannen?

- Interner Scan - Kann das Tool ein Image von Innen, also im laufenden Betrieb, scannen?

- Externer Scan - Kann das Tool ein Image von Außen, also ohne laufende Instanz des Images, scannen? Es ist möglich, ein Cloud-Image in das vorhandene Dateisystem einzubinden und dann zu scannen. Dieses Vorgehen ist allerdings unabhängig von dem Tool. Daher wird solch ein Verfahren hier nur als die Möglichkeit eines externen Scans gewertet, wenn das Tool diese Funktionalität explizit durch entsprechende Automatismen unterstützt.

- Dateinamen-Scanner - Kann das Tool nach auffälligen Dateien, anhand Ihres Namens, scannen?

- Dateiinhalts-Scanner - Kann das Tool die Inhalte von Dateien nach Auffälligkeiten scannen?

- Batch-Scan - Kann das Tool mehrere Images hintereinander automatisiert scannen?

- Lizenz - Unter welcher Lizenz ist das Tool lizenziert? Kann der Quelltext für kommerzielle Projekte genutzt werden oder steht dieser überhaupt zur Verfügung?

4.6.2. Kandidaten

Die nachfolgenden Tools sollen verglichen werden.

- AMID - Der Schwachstellen-Scanner für Cloud-Images, welches 2011 für das CASED entwickelt wurde.

- AMI-Exposed - Ein weiterer Schwachstellen-Scanner für Cloud-Images, welcher 2011 entwickelt wurde.

- RKHunter - Ein verbreiteter Rootkit-Scanner.

- THC-SecureDelete - Eine Sammlung mehrerer Secure-Delete Tools, welche unter anderem auch den freien Speicher überschreiben können.

- Metasploit - Ein Framework, um Systeme auf Exploits zu testen.

4.6.3. Vergleich

Im Folgenden werden nun die Eigenschaften der vorgestellten verwandten Arbeiten tabellarisch aufbereitet.

Tabelle 4.1.: Vergleich verwandter Arbeiten

Kriterium	**AMID**	**AMI-Exposed**	**RKHunter**	**SecureDelete**	**Metasploit**
Modularer Aufbau	Nein	Ja	Nein	Nein	Ja
Datenbank-anbindung	Nein	Nein	Ja	Nein	Nein
Freispeicher-Überschreiber	Nein	Nein	Nein	Ja	Nein
Rootkit-Scanner	Nein	Nein	Ja	Nein	Nein
Interner Scan	Ja	Ja	Ja	Ja	Ja
Externer Scan	Nein	Nein	Nein	Nein	Ja
Dateinamen-Scanner	Ja	Ja	Ja	Nein	Nein
Dateiinhalts-Scanner	Nein	Ja	Ja	Nein	Nein
Batch-Scan	Nein	Ja	Nein	Nein	Ja
Lizenz	GPL	GPL	GPL	GPL	BSD

4.7. Zusammenfassung

Im Rahmen dieses Kapitels wurde gezeigt, dass es bereits einige Tools gibt, welche die Probleme im Zusammenhang mit Betriebssystemimages von Cloud-Infrastrukturdiensten minimieren sollen. Dies geschieht auf unterschiedlichste Art und Weise. So existieren Scanner, um die Images untersuchen zu können, spezielle Secure-Delete Tools, um Daten sicher löschen zu können und auch Exploit-Frameworks, um solche Images auf Schwachstellen hin zu untersuchen.

Jedoch hat ein Vergleich dieser Tools ergeben, dass keines dieser Tools alle Anforderungen, an einen leistungsfähigen und modularen Scanner, erfüllt.

5. Konzeption des Frameworks

Im Folgenden soll ein Framework entwickelt werden, mit dessen Hilfe Images von virtuellen Maschinen nach Schwachstellen gescannt werden können. Dieses Framework soll modular aufgebaut werden, um es später leichter erweitern zu können.

Es wird Module für den Zugriff auf das Image geben. Dadurch ist es möglich, die Images auf verschiedene Arten zu scannen.

Zum einen ist es möglich, das Image von innen zu scannen, indem das Framework in einer laufenden Instanz ausgeführt wird.

Zum anderen wird es möglich sein, das Image von außen zu scannen, indem das Image über das Modul ausgelesen wird und dadurch gescannt werden kann.

Im weiteren Verlauf sollen 2 Zugriffsmodule entwickelt werden:

- Lokales Dateisystem
- Externes Image

Außerdem wird es Module für die Scan-Routinen geben. So ist es leichter, das Framework später mit zusätzlichen Scans zu erweitern.

Im Rahmen dieser Arbeit sollen folgende Scan-Module entwickelt werden:

- Dateinamen-Scanner
- Dateiinhalts-Scanner

- Freispeicher-Überschreiber
- Rootkit-Scanner
- Exploit-Scanner
- Registry-Scanner

Wo es möglich ist, werden die Module Datenbanken nutzen, in denen die Signaturen, nach denen gescannt werden soll, stehen. Dadurch muss später das Framework selbst nicht geändert werden, um weitere Signaturen hinzuzufügen.

Zusätzlich wird das Framework selbst als Klasse mit fest definierten Schnittstellen implementiert werden. Dadurch ist es möglich, verschiedene Frontends für das Framework zu entwickeln.

Folgende Frontends sollen entwickelt werden:

- Text basiertes Menü
- Parametrisierter Befehl
- AWS-Automatik-Scanner

5.1. Genutzte Ressourcen

Bei der Entwicklung des Frameworks kommen verschiedene Ressourcen zum Einsatz.

Das Framework an sich wird in Python [Pyt12a] programmiert werden. Diese Programmiersprache ist heute schon vielerorts im Einsatz. [Pyt12b]

Darüber hinaus werden die einzelnen Module, für deren Realisierung, weitere Software enthalten.

Der externe Zugriff auf das Image der virtuellen Maschine wird über die Bibliothek Libguestfs [Lib12] realisiert werden. Diese Bibliothek kann alle geläufigen Images lesen und schreiben.

Für den Rootkit-Scanner werden die Rootkit-Scanner RKHunter und Chkrootkit integriert werden.

Der Exploit-Scanner wird die Datenbank der National Vulnerability Database nutzen um nach anfälliger Software in dem Image zu suchen.

Der Registry-Scanner wird die Hive-Bibliothek der Libguestfs-Bibliothek nutzen um auf die Registry in einem Windows-Image zugreifen zu können.

Als Datenbank, um Signaturen und Ähnliches zu Speichern, wird SQLite zum Einsatz kommen.

5.2. Framework

Das Framework selbst ist für die Ansteuerung der einzelnen Module zuständig. So lädt es erst das gewünschte Zugriffs-Modul für den Zugriff auf das Image. Als Zweites lädt es das gewünschte Scan-Modul und übergibt diesem das Zugriffs-Modul, damit es vom Scan-Modul genutzt werden kann. Zuletzt führt es das Scan-Modul aus und präsentiert die Ergebnisse.

Das Framework liefert für die einzelnen Schritte passende API-Aufrufe. Diese können dann dem Benutzer, zum Beispiel durch ein Textmenü, zugänglich gemacht werden.

Für die Zugriffs- und Scan-Module stellt das Framework passende Interfaces bereit, um die Entwicklung zu erleichtern.

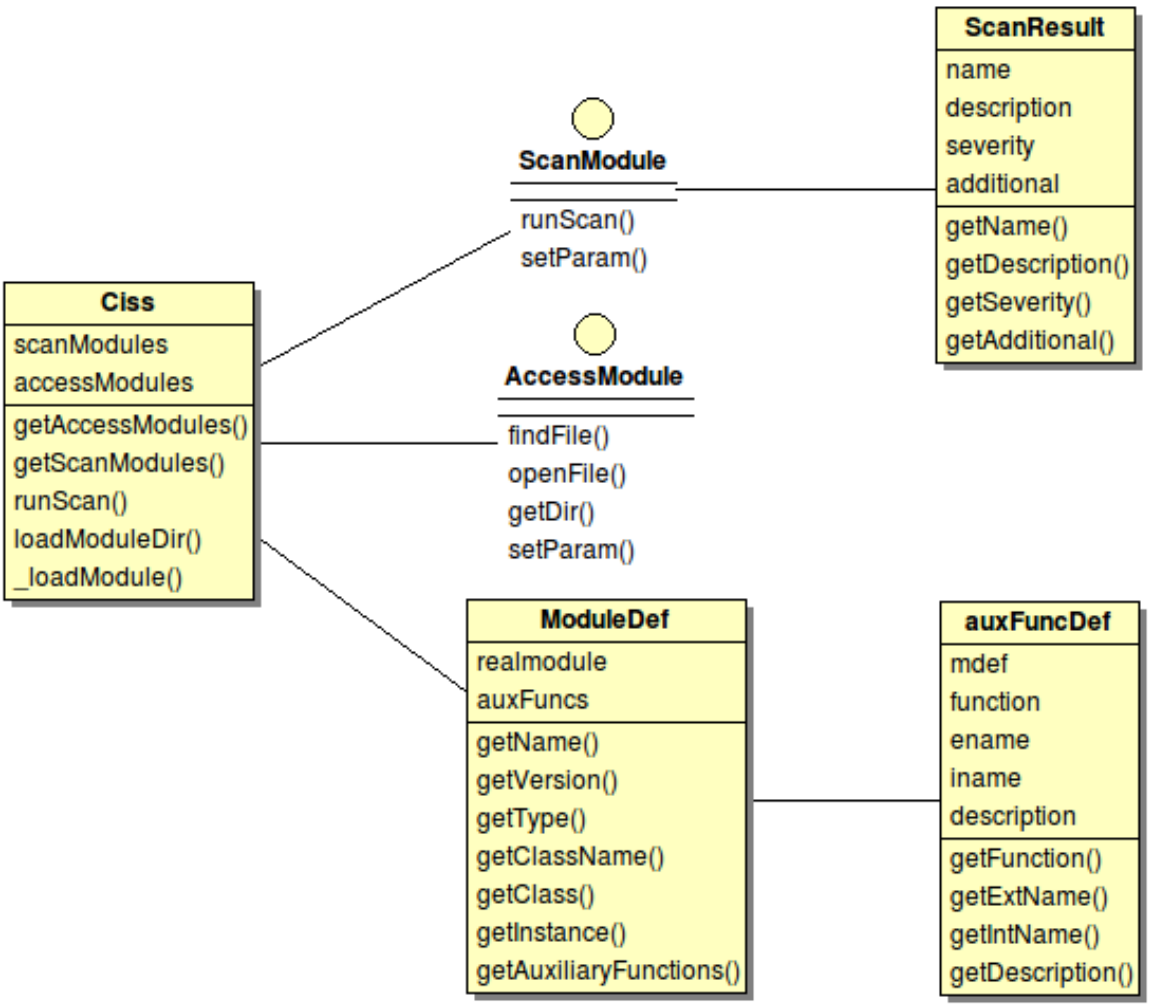

Abbildung 5.1.: Klassenübersicht des Frameworks

5.3. Zugriffs-Module

Mit Hilfe der Zugriffs-Module werden den Scan-Modulen der Zugriff auf die Daten bereitgestellt, welche gescannt werden sollen. Dadurch müssen sich die Scan-Module keine Gedanken machen, wie der Zugriff auf die Daten funktioniert.

5.3.1. Lokales Dateisystem

Das Zugriffs-Modul für das lokale Dateisystem ist ein simples Zugriffsmodul. Es kann direkt die bereitgestellten Systemfunktionen nutzen, um Dateien und Verzeichnisse auf dem lokalen Dateisystem zu lesen.

Es ist daher nicht nötig, das Image aus dem Cloud-Infrastrukturdienst herunter zu laden. Jedoch ist es anfällig für Rootkits, falls das Image lokal gescannt wird. Da das zu scannende System bereits läuft und die Systemfunktionen des eigenen Kernels genutzt werden, könnten die Ergebnisse der Systemfunktionen von einem Rootkit verändert worden sein.

Das Modul enthält einen Parameter namens *basepath* um ein Basisverzeichnis anzugeben. Hierdurch ist es möglich, ein separates Dateisystem in einem bestimmten Verzeichnis einzubinden und im Anschluss das Dateisystem so zu scannen, als wäre das Verzeichnis, in dem das Dateisystem eingebunden wurde, das Wurzelverzeichnis.

Durch diesen Parameter kann das Modul auch für Images verwendet werden, ohne dass eine virtuelle Maschine mit diesem Image gestartet werden muss. Hierzu wird das Dateisystem des Images, welches gescannt werden soll, an ein separates System angebunden und mit Hilfe des *basepath*-Parameters so gescannt, als wäre das Dateisystem des Images das Wurzeldateisystem. So können keine Rootkits aktiv werden, sofern die Scanner-Instanz selbst nicht infiziert ist.

5.3.2. Externes Image

Beim Zugriff auf ein externes Image läuft das Image noch nicht, sondern liegt nur als Festplattenabbild vor. Beim Zugriff muss das Image, je nach verwendetem Format, entsprechend

ausgelesen werden. Dies ist nötig, um die Dateien und Verzeichnisse innerhalb des Images auslesen zu können.

Das Image muss vorher aus der Cloud auf ein separates System heruntergeladen werden. Dafür können aber die Sicherheitsvorkehrungen eventueller Rootkits nicht aktiv werden, da der Kernel des Images nicht geladen ist.

5.4. Scan-Module

Die Scan-Module bewältigen im Framework die Hauptarbeit. Dies kann der Scan nach Schwachstellen sein oder auch das Überschreiben bereits gelöschter Daten.

5.4.1. Dateinamen-Scanner

Der Dateinamen-Scanner scannt das Image nach auffälligen Dateinamen. Dadurch kann er eventuell vergessene private Schlüssel oder andere private Dateien aufspüren. Hierzu bedient er sich einer Datenbank, in welcher Daten, nach denen gescannt werden soll, hinterlegt sind.

In der Datenbank existiert lediglich die Tabelle *filename*. Diese Tabelle hat mehrere Spalten:

- *ID* - Eine eindeutige ID, welche lediglich einer leichteren Indexierung der Einträge dient.
- *Pattern* - Das Pattern, in Form eines regulären Ausdrucks, nach dem gesucht werden soll.
- *Beschreibung* - Eine kurze Beschreibung, nach was das Pattern genau sucht.
- *Einstufung* - Eine Angabe, wie kritisch ein Fund mit dem Patern ist.

Es gibt hier 3 verschiedene Einstufungen:

- *Info* - Diese Datei ist für sich gesehen noch nicht sicherheitsrelevant, kann jedoch Aufschlüsse über eventuelle weitere Sicherheitslücken liefern. Diese Einstufung wird vor allem

für wichtige Auflistungen genutzt. Ein Beispiel ist hier die Auflistung der vorhandenen Home-Verzeichnisse.

- *Warnung* - Diese Datei könnte unter Umständen sicherheitsrelevant sein, muss es aber nicht. Ein Beispiel hierfür sind zum Beispiel öffentliche Schlüssel.

- *Kritisch* - Diese Dateien sind in jedem Fall sicherheitsrelevant und sollten nur vorhanden sein, wenn es wirklich so gewollt ist. Ein Beispiel hierfür sind private Schlüssel.

Der Scanner scannt nach folgenden Patterns:

Tabelle 5.1.: Dateinamen-Patterns

Pattern	**Beschreibung**	**Einstufung**
*.priv	Private Schlüssel	Kritisch
*.pub	Öffentliche Schlüssel	Warnung
*.crt	Zertifikate	Warnung
id_[dr]sa	Private DSA-/RSA-Schlüssel	Kritisch
*.gpg | *.pgp	GPG-/PGP-Dateien	Kritisch
*.jks	Java key-store	Kritisch
secret | *key* | *private*	Interessante Dateien	Warnung
.bash_history	History mit ausgeführten Shell-Befehlen	Kritisch
.svn/ | .git/ | .hg/	Quellcode-Repositories	Warnung

5.4.2. Dateiinhalts-Scanner

Ähnlich, wie der Dateinamen-Scanner, scannt der Dateiinhalts-Scanner nach Patterns, welche in einer Datenbank hinterlegt sind. Jedoch sucht dieser Scanner innerhalb von vorgegebenen Dateien nach den Patterns. Dadurch können beispielsweise unsichere Konfigurationseinstellungen, Backdoors oder vergessene Passwörter gefunden werden.

Genau, wie der Dateinamen-Scanner nutzt auch der Dateiinhalts-Scanner reguläre Ausdrücke, um sicherheitsrelevante Dateiinhalte zu finden. Hierbei werden sowohl für die Dateinamen,

als auch für die Dateiinhalte, reguläre Ausdrücke genutzt, um den Scanner noch flexibler zu gestalten.

Die genutzte Datenbank besteht aus der Tabelle *filecontent*, welche 5 Spalten besitzt:

- *ID* - Eine eindeutige ID, welche lediglich einer leichteren Indexierung der Einträge dient.
- *Datei-Pattern* - Dies ist ein Pattern, in Form eines regulären Ausdrucks. Dieses Pattern repräsentiert die Dateinamen, deren Inhalte durchsucht werden sollen.
- *Inhalts-Pattern* - Auch dies ist ein Pattern, welches durch einen regulären Ausdruck angegeben wird. Dieses Pattern gibt den Inhalt an, nach dem die gefundenen Dateien durchsucht werden sollen.
- *Beschreibung* - Eine kurze Beschreibung, nach welchen Inhalten das Pattern genau sucht.
- *Einstufung* - Eine Angabe, wie kritisch ein Fund mit dem Patern ist.

Wieder gibt es 3 verschiedene Einstufungen:

- *Info* - Ein solcher Eintrag muss noch nicht sicherheitsrelevant sein. Dennoch sollte man sich auch diese Meldungen genau ansehen, um weitere Sicherheitslücken aufzuspüren. Oft werden Auflistungen als Info eingestuft. Zum Beispiel die Auflistung der Einträge der passwd-Datei sind als Info eingestuft.
- *Warnung* - Dieser Eintrag könnte sicherheitsrelevant sein oder private Daten weitergeben, wenn er nicht absichtlich so gewählt wurde. Ein Beispiel hierzu sind SQL-Passwörter in Konfigurationsdateien.
- *Kritisch* - Solch ein Eintrag ist in jedem Fall sicherheitsrelevant und sollte genauestens überprüft werden. Ein Beispiel sind Einträge in der *authorized_keys*-Datei.

Auch hier gibt es wieder eine Reihe von Patterns nach denen gescannt werden kann:

Tabelle 5.3.: Dateiinhalt-Patterns

Datei-Pattern	Inhalts-Pattern	Beschreibung	Einstufung
/etc/proftpd.conf	*SQLConnectInfo*	SQL-Zugangsdaten für proftpd	Kritisch
*/authorized_keys	*	SSH-Schlüssel, die sich zur Instanz verbinden können	Kritisch
*.pem	*PRIVATE KEY*	Privater Schlüssel	Kritisch

5.4.3. Freispeicher-Überschreiber

Der Freispeicher-Überschreiber dient dem Zweck, den freien Speicher innerhalb eines Images zu überschreiben. Dadurch werden bereits gelöschte Daten nochmals überschrieben um eine Wiederherstellung unmöglich zu machen. So kommen andere Personen, die das Image nutzen, nicht mehr an diese Daten heran.

Es gibt verschiedene Herangehensweisen, um so einen Überschreiber zu realisieren:

Am effektivsten wäre es, das vorhandene Dateisystem zu analysieren und dadurch die freien Bereiche ausfindig zu machen. Jedoch ist das, aufgrund der verschiedensten Dateisysteme, sehr aufwändig.

Einfacher ist es daher, eine oder mehrere Dateien innerhalb des Images zu erstellen und diese immer weiter anwachsen zu lassen, bis diese den kompletten freien Speicher des Images füllen. Im Anschluss muss noch eine Synchronisation der Festplatte des Images angestoßen werden, um sicher zu gehen, dass auch alle Daten aus dem Cache auf die Festplatte geschrieben wurden. Zuletzt werden die Dateien wieder gelöscht und bleiben dadurch nur noch als Datenmüll im freien Speicher zurück. Dadurch sind alle Daten im freien Speicher überschrieben.

5.4.4. Rootkit-Scanner

Der Rootkit-Scanner scannt, mit Hilfe bereits existierender Rootkit-Scanner, das Image nach Rootkits, um die Gefahr, dass das Image bereits von Beginn an infiziert ist, zu minimieren. Jedoch kann eine Infektion nie hundertprozentig ausgeschlossen werden.

Es ist ratsam, das Image nur von außen zu scannen, da das Rootkit im laufenden Betrieb die Erkennung, durch eingebaute Sicherheitsvorkehrungen, verhindern kann.

5.4.5. Exploit-Scanner

Mit Hilfe des Exploit-Scanners kann nach veralteter und anfälliger Software innerhalb des Images gesucht werden.

Auch hierzu wird eine Datenbank genutzt. Da bereits mehrere Projekte mit Informationen zu Schwachstellen existieren, wird dieses Modul auf eine solche Quelle zurückgreifen. Im Folgenden soll kurz evaluiert werden, welche Quelle für dieses Modul am besten geeignet ist.

Als wichtigste Kriterien muss die Datenbank Informationen über die anfällige Software besitzen und frei verfügbar sein. Darüber hinaus sollte die Datenbank so umfangreich wie möglich sein und muss herunterladbar sein, um sie auch lokal nutzen zu können.

Metasploit

Metasploit [Met12] ist ein Framework um Exploits auszuführen. Es enthält eine große Anzahl an Exploits, welche stetig erweitert wird. Auch ist die Software frei verfügbar.

Allerdings enthält Metasploit keine genauen Angaben über die anfällige Software, sondern verweist hierzu auf andere Datenbanken. Dadurch ist Metasploit selbst als solche Datenbank ungeeignet.

Exploit-DB

Die Exploit-DB [Exp12] ist eine Datenbank, welche verschiedenste Exploits bereitstellt. Es enthält eine große Anzahl (aktuell über 17.000) verschiedenster Exploits. Das komplette Archiv, lässt sich auch herunterladen.

Jedoch fehlen auch hier, ähnlich wie bei Metasploit, genaue Angaben über die Software, welche

die Exploits betrifft. Lediglich die ID der Common Vulnerabilities and Exposures (siehe weiter unten) wird angegeben. Dadurch müsste man hier für diese Angaben wieder auf eine andere Datenbank zurückgreifen, wodurch auch diese Datenbank direkt genutzt werden könnte.

Open Source Vulnerability Database

Die Open Source Vulnerability Database [Osv12] oder kurz OSVDB ist eine unabhängige Schwachstellen-Datenbank, welche von einer Community gepflegt wird. Aktuell enthält diese über 77.000 Einträge und ist dadurch sehr umfangreich. Auch enthält sie Informationen über die anfällige Software und kann direkt als SQLite-Datenbank heruntergeladen werden.

Jedoch sind bei vielen Schwachstellen nur unzureichende Informationen eingetragen. So fehlt bei der Mehrheit der Schwachstellen die wichtige Angabe der anfälligen Software. Die Datenbank ist dadurch zwar einsetzbar, würde aber nicht sehr zufriedenstellend funktionieren.

Securityfocus

Auch die Webseite Securityfocus [Sec12a] stellt eine sehr umfangreiche Schwachstellen-Datenbank bereit. Diese enthält, neben den Informationen zu den Schwachstellen, auch Angaben über die dafür anfällige Software.

Dennoch ist auch diese Datenbank ungeeignet, da diese keine Möglichkeit bietet, die Datenbank für lokale Abfragen herunter zu laden.

Common Vulnerabilities and Exposures

Common Vulnerabilities and Exposures [CVE12] oder kurz CVE ist ein Verzeichnis mit einer großen Anzahl an Einträgen zu Schwachstellen. Es vergibt eindeutige Identifikationsnummern, um einen einheitlichen Austausch von Informationen zu Schwachstellen zu ermöglichen. Daher findet man diese Identifikationsnummern auch in vielen Schwachstellen-Datenbank. Darüber hinaus ist die Datenbank frei verfügbar und kann in verschiedenen Formaten herunter geladen werden.

Da jedoch Common Vulnerabilities and Exposures eher ein Verzeichnis und keine Datenbank ist, enthält es keine Detailinformationen zu den Schwachstellen. Vor Allem fehlen die benötigten Informationen über anfällige Software. Deshalb ist auch das CVE-Verzeichnis nicht als Informationsquelle für den Scanner geeignet.

National Vulnerability Database

Die National Vulnerability Database [NVD12] ist eine Schwachstellen-Datenbank, welche vom National Institute of Standards and Technology und dem United States Computer Emergency Readiness Team gepflegt wird. Die Datenbank baut hierbei direkt auf den Informationen aus dem CVE-Verzeichnis auf und erweitert diese durch zusätzliche Informationen. Dazu gehören, neben Detailinformationen zur Schwachstelle, auch die hier wichtigen Informationen über betroffene Software und den Versionen dieser Software. Darüber hinaus ist die Datenbank frei verfügbar und kann als XML-Dokument herunter geladen werden.

Die National Vulnerability Database ist sehr gut als Informationsquelle für den Exploit-Scanner geeignet. Die Informationen sind sehr vollständig und es gibt die benötigten Angaben über die betroffene Software. Außerdem kann die Datenbank für lokale Auswertungen heruntergeladen werden.

Ergebnis

Nach Prüfung verschiedener Informationsquellen über Schwachstellen ist die National Vulnerability Database als Informationsquelle die beste Wahl. Daher wird diese bei der Umsetzung als Referenz genutzt werden. Da die Datenbank jedoch nur als XML-Dokument vorhanden ist, muss diese erst in eine SQLite-Datenbank konvertiert werden, was bei der Umsetzung des Exploit-Scanners in Form eines kleinen Tools berücksichtigt werden wird.

Um die Qualität der Informationen weiter zu verbessern wäre es sinnvoll, mehrere Datenquellen zu kombinieren. Dies würde jedoch den über den Umfang dieser Arbeit hinaus gehen und ist daher nur als zukünftige Verbesserung vorgesehen.

5.4.6. Registry-Scanner

Viele Informationen, welche unter Linux über diverse Konfigurationsdateien verteilt sind, sind unter Windows in der Registry gespeichert. Hierzu gehören auch sicherheitsrelevante Informationen, wie Passwörter, automatisch startende Programme und Ähnliches. Der Registry-Scanner scannt die Registry nach auffälligen Werten, beziehungsweise gibt eine Übersicht über sicherheitsrelevante Einstellungen aus. Auch hier sind die Patterns, nach denen gescannt werden soll, wieder in einer Datenbank gespeichert.

5.5. Frontends

Mit Hilfe der Frontends wird das Framework, welches nur aus einer API besteht, lauffähig gemacht. So können Nutzer das Framework mit einem textbasierten Menu benutzen oder auch automatisch innerhalb der Amazon Web Services laufen lassen.

5.5.1. Text basiertes Menü

Das Text basierte Menü ist als direktes Frontend für einen Benutzer gedacht. Er kann dadurch sehr einfach einen Scan nach dem Anderen ausführen und bekommt übersichtlich alle Module und Resultate angezeigt. Im Hauptmenü werden, neben den einzelnen Menüpunkten, auch das gerade aktive Zugriffs-Modul und Scan-Modul angezeigt.

Folgende Menüpunkte wird es im Hauptmenü geben:

- *Zugriffs-Modul auswählen* - Dadurch kann das aktive Zugriffs-Modul geändert werden. Der Nutzer kann aus einer Liste aller geladenen Zugriffsmodule das passende per Tastendruck wählen.

- *Scan-Modul auswählen* - Analog zu dem vorhergehenden Menüpunkt kann hier das aktive Scan-Modul geändert werden. Auch hier wird dem Benutzer wieder eine Liste der geladenen Scan-Module präsentiert, aus der der Benutzer das gewünschte Modul auswählen kann.

- *Scan starten* - Hiermit wird das ausgewählte Scan-Modul mit dem ausgewählten Zugriffs-Modul gestartet.

- *Ergebnisse anzeigen* - Dadurch können alle vorhergegangenen Scan-Ergebnisse seit dem Start des Programms angezeigt werden.

- *Ergebnisse in Datei schreiben* - Hiermit werden die vorangegangenen Scan-Ergebnisse in eine Datei geschrieben.

- *Beenden* - Beendet das Programm.

Dadurch, dass das Programm sich alle Scan-Ergebnisse seit dem Start des Programms merkt, können auch mehrere Scans hintereinander ausgeführt werden und anschließend als ganzes die Ergebnisse analysiert oder gespeichert werden.

5.5.2. Parametrisierter Befehl

Mit Hilfe eines parametrisierten Befehls ist es möglich, den Scanner auch aus einem Skript heraus zu starten. Hierbei wird kein Menü angezeigt, sondern alle gewünschten Operationen werden als Parameter an den Befehl angehängt.

Folgende Parameter sind geplant:

- *-h* - Gibt eine Übersicht über die einzelnen Parameter aus.

- *-v* - Gibt die Versionsnummer aus und beendet sich wieder.

- *-a* - Auswahl des gewünschten Zugriffs-Moduls.

- *-s* - Auswahl des gewünschten Scan-Moduls. Dieser Parameter kann auch mehrmals angegeben werden. Dann werden mehrere verschiedene Scans hintereinander ausgeführt.

- *-f* - Gibt die Datei an, in welche die Resultate geschrieben werden sollen. Wenn dieser Befehl fehlt, werden die Resultate in die Standardausgabe geschrieben.

5.5.3. AWS

Der Scan eines Images auf den Amazon Web Services erfordert einige Vorarbeit. Diese soll mit Hilfe des AWS-Images abgenommen werden. Als Bibliothek für den Zugriff auf die Amazon Web Services soll Boto [Bot12] zum Einsatz kommen.

Als Voraussetzung für die Ausführung des AWS-Frontends wird eine kleine Instanz in den Amazon Web Services benötigt (im Folgenden auch Scanner-Instanz genannt). Diese Scanner-Instanz führt den Scanner aus. Da bei den Amazon Web Services nur Instanzen, aber keine Images direkt gescannt werden können, muss zuerst eine weitere Instanz mit dem Image gestartet werden, welches gescannt werden soll. Diese Instanz könnte zwar auch lokal gescannt werden, was aber eher unsicher ist, da sich Rootkits verstecken könnten. Daher wird diese Instanz sofort wieder gestoppt und, mit Ausnahme der Festplatte, wieder gelöscht. Anschließend wird diese Festplatte als zweite Festplatte der Scanner-Instanz eingebunden. Diese Festplatte enthält nun das Image, welches gescannt werden soll. Nachdem der Scanner seine Arbeit auf der zweiten Festplatte abgeschlossen hat, wird die Festplatte wieder von der Scanner-Instanz getrennt und gelöscht.

XML-Konfigurationsdatei

Das AWS-Frontend wird mit Hilfe einer XML-Datei konfiguriert. Diese trägt den Dateinamen *awsconfig.xml* und liegt im Hauptverzeichniss des Scanners. Der Aufbau dieser Datei sieht wie folgt aus:

Hierbei haben die einzelnen Tags folgende Funktionen:

- *awsid* - Hier wird die Benutzer-ID der Amazon Web Services angegeben.
- *awskey* - Dies ist der Zugriffsschlüssel für den Zugriff auf die Amazon Web Services.
- *region* - Die Region, in der der Scan stattfinden kann. Muss mit der Region übereinstimmen, in welcher auch die Scan-Instanz liegt.
- *zone* - Gibt die Zone innerhalb der Region an. Auch diese muss mit der Zone der Scanner-Instanz übereinstimmen. Diese beiden Parameter sind nötig, da es nur möglich ist, Fest-

```
<?xml version="1.0" encoding="UTF-8"?>
<aws>
   <awsid>ABGDJFHJDDO4F24GGSSQ</awsid>
   <awskey>ZieugfiukdlKlwi52IwsBXoWFqQbiwoPBM+dh</awskey>
   <region>us-east-1</region>
   <zone>us-east-1d</zone>
   <instanceType>t1.micro</instanceType>
   <scanInstanceID>i-3c354c45</scanInstanceID>
   <volumeDevice>/dev/sdf</volumeDevice>
   <mountDir>/tmp/scanDir/</mountDir>
   <saveMode>SQLite</saveMode>
   <awsimages>
      <imageid>ami-0055ad69</imageid>
      <!-- <imageid>ami-08a45f61</imageid> -->
   </awsimages>
   <scanmodules>
      <module>Filename Scanner</module>
      <module>Filecontent Scanner</module>
   </scanmodules>
</aws>
```

Abbildung 5.2.: Aufbau der *awsconfig.xml*

platten von Instanzen in der gleichen Region und Zone auszutauschen.

- *instanceType* - Gibt den Typ der Instanzen an, die aus den zu scannenden Images gestartet werden sollen.

- *scanInstanceID* - Gibt die ID der Scanner-Instanz an.

- *volumeDevice* - Gibt das Gerät an, unter dem die zu scannenden Festplatten in der Scanner-Instanz angebunden werden. Fehlt dieser Parameter, wird das Gerät unter */dev/sdf* genommen.

- *mountDir* - Dies gibt das Verzeichnis an, in dem die zu scannende Festplatte eingehängt wird. Standardmäßig wird */tmp/scanDir/* genutzt.

- *saveMode* - Gibt an, ob die Resultate in einer SQLite-Datenbank oder in XML-Dateien gespeichert werden sollen. Mögliche Werte sind *XML* und *SQLite*. Standard ist *SQLite*.

- *awsimages* - Enthält eine Liste aus *imageid*-Tags. Jedes dieser Tags gibt die ID eines Images an, welches gescannt werden soll.

- *scanmodules* - Enthält mehrere *module*-Tags. Mit diesen Tags wird jeweils der Name eines Scanner-Moduls angegeben, welches für jedes angegebene Image ausgeführt werden soll.

5.6. Zusammenfassung

In diesem Kapitel wurde das Framework, zum Scannen von Betriebssystemimages, zusammen mit diversen Modulen und Frontends, konzipiert.

Es wurden zwei Zugriffs-Module konzipiert, welche den zugriff auf die Daten der Images gewährleisten.

Außerdem wurden sechs Scan-Module konzipiert. Diese sollen später die Images nach Schwachstellen scannen.

Zuletzt wurden noch drei Frontends konzipiert. Erst durch diese Frontends wird das Framework, welches selbst nur auf einer API besteht, bedienbar.

6. Umsetzung

Im Folgenden Kapitel wird nun die Implementation des Scanners, dokumentiert. Diese besteht aus dem Framework, passenden Modulen und Frontends für den Zugriff auf das Framework.

Das Framework selbst besteht fast ausschließlich aus dem Modul-System, mit dem die einzelnen Module geladen werden. Dieses wird separat dokumentiert. Lediglich die Funktion *runScan* existiert noch zusätzlich zu den Funktionen des Modul-Systems. Diese Funktion ist für die Durchführung eines Scans mit Hilfe eines Zugriffs- und eines Scan-Moduls. Hierzu wird lediglich die Funktion *runScan* des Scan-Moduls aufgerufen und das Zugriffs-Modul als Parameter übergeben. Anschließend gibt die Funktion die Resultate des Scan-Moduls zurück.

6.1. Modul-System

Das Framework besitzt ein Modul-System, um es leichter anpassbar zu machen. So ist es später bei neuen Scannmethoden oder Zugriffsmethoden nicht nötig, das Framework selbst anzupassen. Das Modul-System ist sehr einfach gestrickt und bedient sich einiger Vorteile von Python.

Jedes Modul muss aus einer Hauptklasse bestehen. Für ein Scanner-Modul muss diese Klasse von der Klasse *ScanModule* abgeleitet sein. Bei einem Zugriffs-Modul muss die Klasse von *AccessModule* abgeleitet sein. Außerdem benötigt das Modul eine globale Variable mit dem Namen *Module*. Diese Variable muss von dem Python-Typ *dictionary* sein. Dieser Typ ist in Python ein Array mit einer Schlüssel -> Wert Zuweisung. Dadurch wird für jedes Modul einige Metadaten definiert, welche auch von Laien gelesen werden können, die Python nicht beherrschen.

Der Aufbau der Metadaten in Form der globalen Variable *Module* sieht wie folgt aus:

```
Module = {
  "name": "Example Module",
  "version": "0.1",
  "class": "TestModule",
  "auxFunctions": [["auxFunc", "auxFunc", "Example auxilliary function"]]
}
```

Abbildung 6.1.: Metadaten für Module

Die einzelnen Parameter haben folgende Bedeutung:

- Der Parameter *name* beinhaltet den Namen des Moduls.
- Mit dem Parameter *version* wird eine Version definiert.
- Mittels *class* wird der Name der Hauptklasse bestimmt. Diese Zeichenkette muss exakt mit dem Namen der Hauptklasse übereinstimmen, damit das Modul später geladen werden kann.
- Mittels *auxFunctions* wird eine Liste der Zusatzfunktionen des Moduls angegeben. Die Funktionsweise der Zusatzfunktionen wird in einem eigenen Abschnitt behandelt. Dieser Parameter ist optional.

Module werden innerhalb des Frameworks durch die Klasse *ModuleDef* repräsentiert.

Diese bietet diverse Funktionen um auf die Metadaten und andere Informationen über das Modul zugreifen zu können:

- Mit der Funktion *getName* kann man sich den Namen, welcher in den Metadaten des Moduls definiert wurde, zurückgeben lassen.
- Die Funktion *getVersion* liefert die Modulversion, welche in den Metadaten des Moduls steht.
- *getClassName* liefert den Namen der Hauptklasse des Moduls als Zeichenkette zurück, so wie diese in den Metadaten des Moduls definiert wurde.

- Die Funktion *getType* liefert den Typ des Moduls. Der Typ wird automatisch anhand der Basisklasse des Moduls erkannt und muss daher nicht extra in den Metadaten des Moduls stehen. Für Module, welche von der Klasse *ScanModule* abgeleitet wurden, liefert die Funktion die Zeichenkette „SCAN“. Für Module mit der Basisklasse *AccessModule* liefert die Funktion „ACCESS“. Sollte das Modul von keiner der beiden Klassen abgeleitet sein, wird „UNKNOWN“ zurückgegeben.

- *getClass* gibt die Hauptklasse als Klasse zurück und dient daher dem direkten Zugriff auf diese Klasse.

- *getInstance* gibt eine Instanz bzw. ein Objekt des assoziierten Moduls zurück. Diese ist in einem *ModuleDef*-Objekt immer dieselbe.

- *getAuxiliaryFunctions* liefert eine Liste der im Modul definierten Zusatzfunktionen in Form von *auxFuncDef*-Objekten.

- *__init__* ist der Konstruktor. Dieser initialisiert die einzelnen Membervariablen. Dazu erzeugt er auch eine Instanz bzw. ein Objekt des assoziierten Moduls.

Für Scanner-Module existiert eine Hilfsklasse *ScanResult*. Diese Klasse repräsentiert ein einzelnes Scan-Ergebnis.

Diese Klasse hat folgende Funktionen für den Zugriff auf die Daten des Ergebnisses:

- *getName* gibt den Namen der Ergebnisses zurück. Dies kann der Name der Datei, ein Pattern oder eine sonstige textuelle Kurzform des Ergebnisses sein.

- *getDescription* liefert eine genaue Beschreibung des Ergebnisses. Dazu gehört Beispielsweise eine Beschreibung der gefundenen Datei oder des Dateiinhalts.

- *getSeverity* gibt die Einstufung in Textform zurück. Geläufige Einstufungen sind *Kritisch*, *Warnung* oder *Information*.

- *getAdditional* liefert zusätzliche Informationen über das Ergebnis. Dieses Feld kann auch leer sein, wenn es keine zusätzlichen Informationen gibt.

- *__init__* ist der Konstruktor der Klasse. Ihm werden als Parameter die Daten für die

einzelnen Felder übergeben (*Name*, *Description*, *Severity*, *Additional*)

Für Scanner-Module existiert die Klasse *ScanModule* als Interface. Ein Scanner-Modul muss alle Funktionen dieser Klasse implementieren.

Das Scanner-Modul hat lediglich folgende Funktion:

- Mit der Funktion *runScan* wird ein Scan gestartet. Als Parameter wird der Funktion das gewünschte Zugriffs-Modul übergeben. Als Rückgabewert muss das Modul eine Liste mit *ScanResult*-Objekten zurückgeben.

Analog zu *ScanModule* gibt es für die Zugriffs-Module eine Klasse *AccessModule*, welche als Interface dient. Auch diese Klasse besitzt diverse Funktionen, welche alle implementiert werden müssen:

- Die Funktion *findFile* dient der Suche nach Dateien. Als Parameter wird zuerst ein regulärer Ausdruck der Datei übergeben. Dieser kann auch Teile des Pfades, bzw. den kompletten Pfad enthalten. Der zweite Parameter ist optional und dient der Angabe eines Basisverzeichnisses. Die Funktion sucht nur in diesem und den darunterliegenden Verzeichnissen. Wenn der Parameter nicht angegeben wird, wird im gesamten System gesucht. Auch der dritte Parameter ist optional und gibt an, ob nach dem ersten Fund die Suche abgebrochen werden soll oder ob alle Funde zurückgegeben werden sollen. Standardmäßig werden alle Funde zurückgegeben. Die Funktion gibt eine Liste mit den Pfaden der gefundenen Dateien zurück.

- Mit der Funktion *openFile* wird eine Datei geöffnet. Als Parameter wird der Dateiname mit Pfad angegeben. Die Funktion liefert ein File-Objekt zurück.

- Um eine Liste mit allen Inhalten eines Verzeichnisses (Dateien und Unterverzeichnisse) zu erhalten kann die Funktion *getDir* genutzt werden. Als Parameter wird der Pfadname zu dem Verzeichnis übergeben.

Der Hauptteil des Modul-Systems befindet sich in der Klasse *Ciss*. Diese ist für die Steuerung des Frameworks zuständig. Für das Modul-System sind folgende Attribute und Methoden dieser Klasse wichtig.

- Das Attribut *accessModules* ist eine Liste, welche alle geladenen Zugriffs-Module enthält.

Um auf diese Liste zugreifen zu können, existiert die Methode *getAccessModules*.

- Analog zu *accessModules* für Zugriffs-Module einhält das Attribut *scanModules* eine Liste mit allen geladenen Scanner-Modulen. Auf diese Liste kann über die Funktion *getScanModules* zugegriffen werden.

- *loadModuleDir* ist eine Funktion, welche das Unterverzeichnis *modules* nach geeigneten Modulen durchsucht und lädt. Dazu übergibt es die Namen aller Dateien mit den Endungen .py und .pyc nacheinander als Modulname an die Funktion *_loadModule*.

- Der größte Teil des Modul-Systems steckt in der Funktion *_loadModule*. Diese lädt ein einzelnes Modul anhand seines Namens. Dazu wird die eingebaute Python-Funktion *__import__* benutzt. Diese kann ein Python-Modul zur Laufzeit nachladen. Nach dem Import wird überprüft, ob das Modul gültig ist, also passende Metadaten definiert. Wenn das so ist, wird ein neues *ModuleDef*-Objekt für das gerade geladene Modul erzeugt und an die passende Modul-Liste angefügt.

6.1.1. Zusatzfunktionen

Mittels Zusatzfunktionen können einzelne Funktionen eines Moduls separat verfügbar gemacht werden. Dadurch können beispielsweise andere Module die Funktionen nutzen und müssen diese nicht noch einmal selbst implementieren. Ein gutes Beispiel hierfür ist der Dateiinhalts-Scanner. Dieser kann durch die Zusatzfunktionen die Funktionalität des Dateinamen-Scanners nutzen um die Dateien zu finden, welche gescannt werden sollen. Die Zusatzfunktionen werden intern als Methoden der Modul-Klasse implementiert. Dadurch können einzelne Attribute der Klasse wiederverwendet werden.

Der größte Teil der Funktionalität des Zusatzfunktionen-Systems ist in der Klasse *auxFuncDef* und im Konstruktor der *ModuleDef*-Klasse implementiert.

Im Konstruktor der *ModuleDef*-Klasse wird die Liste der Zusatzfunktionen aus den Metadaten ausgelesen und entsprechende *auxFuncDef*-Objekte erzeugt. Jeder Listeneintrag in den Metadaten hat der Reihe nach folgende Parameter:

- Der erste Parameter gibt den externen Namen an. Dieser Name kann frei gewählt werden

und dient hauptsächlich dem Auffinden der Zusatzfunktion durch andere Module.

- Anschließend folgt der interne Name. Dieser gibt den Funktionsnamen an, welche die Funktion innerhalb der Modul-Klasse besitzt.

- An dritter Position folgt eine textuelle Beschreibung der Funktion.

Die Klasse *auxFuncDef* hat verschiedene Methoden für den Zugriff auf die Zusatzfunktion und deren Parameter.

- *__init__* ist wieder der Konstruktor. Dieser initialisiert die Membervariablen. Neben den einzelnen Parametern aus den Metadaten und der Funktion selbst wird noch das *ModuleDef*-Objekt übergeben, zu dem die Zusatzfunktion gehört. Dies wird beim Zugriff auf die Funktion benötigt.

- *getExtName* liefert den externen Namen der Funktion, wie dieser in den Metadaten des Moduls angegeben wurde.

- *getIntName* liefert den internen Namen.

- *getDescription* gibt die textuelle Beschreibung aus den Metadaten zurück.

- *getFunction* dient dem Zugriff auf die Funktion. Hierbei wird der Umweg über eine lambda-Funktion gegangen. Dies ist eine Funktion, welche keine separate Definition benötigt, sondern direkt innerhalb einer anderen Funktion definiert und implementiert werden kann. Diese Funktion ruft lediglich die eigentliche Funktion auf und übergibt dieser das Objekt des Moduls. Das ist nötig, da die Funktion innerhalb der Modul-Klasse implementiert ist und daher ein Objekt dieser Klasse angegeben werden muss. Durch diesen Umweg über die lambda-Funktion muss sich ein späterer Entwickler keine Gedanken über das passende Objekt machen und dieses auch nicht übergeben.

6.2. Zugriffs-Module

In diesem Abschnitt wird die Implementation der Module für den Zugriff auf die Daten der Images dokumentiert.

6.2.1. Lokales Dateisystem

Das Zugriffs-Modul für das lokale Dateisystem kann sehr einfach implementiert werden, da hier die bereits vorhandenen Hilfsfunktionen von Python genutzt werden können. Im Folgenden wird kurz auf die Details der Implementierten Funktionen des *AccessModule*-Interfaces eingegangen.

- Für die Funktion *getDir* wird die Python-Funktion *os.listdir* genutzt, welche genau die gewünschte Funktionsweise von *getDir* liefert.

- Mit Hilfe der, in Python integrierten, Funktion *open* kann direkt die Funktion *openFile* implementiert werden.

- Die Funktion *findFile* ist ein wenig komplexer. Jedoch kann auch hier viel Arbeit von den Funktionen in Python abgenommen werden. Im Vorfeld wird der reguläre Ausdruck einmalig kompiliert, um das nicht später für jeden Vergleich erneut bewerkstelligen zu müssen. Ein manuelles durchlaufen der Verzeichnisse und Unterverzeichnisse würde sehr lange dauern. Python stellt jedoch genau für diesen Fall die Funktion *os.walk* bereit, welche genau diese Funktion performant bereitstellt. Nun wird bei jeder durchlaufenen Datei der Dateiname mit dem Pattern verglichen und bei Übereinstimmung der Dateiname mit Pfad an die Liste, welche später zurückgegeben werden soll, angehängt.

6.3. Scan-Module

Nach den Zugriffs-Modulen wird nun die Implementation der Scan-Module dokumentiert, welche für den Scan der Images zuständig sind.

6.3.1. Dateinamen-Scanner

Das Dateinamen-Scanner-Modul ist ein Modul, welches mit Hilfe von regulären Ausdrücken nach Dateinamen sucht. Es nutzt dazu eine SQLite-Datenbank, welche die einzelnen Patterns, zusammen mit einer Beschreibung enthält.

Im Folgenden wird die Implementation der einzelnen Methoden kurz beschrieben.

- In *runScan* wird nach den Dateinamen gescannt. Hierzu werden zuerst alle Einträge aus der Datenbank gelesen. Im Anschluss wird nacheinander nach allen Patterns mit Hilfe der *findFile*-Methode des *AccessModule*-Objektes gesucht. Für jedes Ergebnis dieser Suche wird ein *ScanResult*-Objekt erstellt und zu der Liste, welche am Ende zurückgegeben werden soll, hinzugefügt.

6.3.2. Dateiinhalts-Scanner

Der Dateiinhalts-Scanner ist dem Dateinamens-Scanner sehr ähnlich, geht jedoch noch etwas weiter. Nachdem eine Datei anhand eines Patterns gefunden wurde, wird anschließend noch der Inhalt der Datei, bzw. der Dateien anhand eines Patterns durchsucht. Hierzu stehen wieder die Patterns, zusammen mit einer textuellen Beschreibung des gesuchten Problems in einer SQLite-Datenbank.

Hier soll wieder kurz die Implementierung des Moduls anhand der Methoden beschrieben werden.

- Die Logik des Moduls findet sich wieder in der Methode *runScan*. Zuerst werden wieder alle Einträge aus der zugehörigen Datenbank gelesen. Anschließend beginnt der Scanner, wieder damit, die einzelnen Dateien anhand der Datei-Patterns zu suchen. Dazu werden diese an die Methode *findFile* des *AccessModule*-Objektes übergeben. Wurde eine Datei gefunden, wird diese mit der *openFile*-Methode geöffnet und der Dateiinhalt ausgelesen. Anschließend wird dieser Inhalt nach dem Inhalts-Pattern durchsucht. Hierzu wird die Funktionalität des Python-Moduls *re* genutzt, um nach den Patterns zu suchen. Wurde eine Übereinstimmung gefunden, wird ein passendes *ScanResult*-Objekt erstellt und die Liste dieser Objekte am Ende zurückgegeben.

6.3.3. Freispeicher-Überschreiber

Mit Hilfe des Freispeicher-Überschreibers wird der freie Speicher auf dem Cloud-Image überschrieben. In dem Zusammenhang werden bereits gelöschte Dateien überschrieben, damit diese

nicht wieder hergestellt werden können. Dies geschieht mit Hilfe von großen Dateien, welche mit Null-Zeichen gefüllt werden, bis kein Speicherplatz mehr verfügbar ist.

Hier sollen wieder der Ablauf des Moduls beschrieben werden:

- Die Logik des Moduls befindet sich in der Methode *runScan*. Hier wird zuerst eine Zeichenkette aus Null-Zeichen erzeugt. Diese Zeichenkette ist 32 Megabyte groß und wird später immer wieder in die Dateien geschrieben. Durch diese Zeichenkette kann die Datei später schneller beschrieben werden, da immer Blockweise auf die Festplatte geschrieben werden kann, anstatt immer nur Zeichen für Zeichen.

- Anschließend beginnt eine Endlosschleife. Hier ist auch die Behandlung der Ausnahmen wichtig, da diese ausgelöst werden, wenn die Festplatte voll ist. Dadurch wird auch die Schleife wieder verlassen. Dies ist die Einfachste Methode um sicher zu gehen, dass Daten geschrieben werden, bis die Festplatte komplett voll ist.

- Innerhalb der Schleife wird über das Zugriffs-Modul immer wieder eine neue Datei geöffnet und anschließend so lange mit der vorbereiteten Zeichenkette beschrieben, bis entweder die Festplatte voll ist und eine Ausnahme erzeugt wird oder die Datei zwei Gigabyte groß ist. Durch diese Limitierung auf zwei Gigabyte kann das Modul auch mit Dateisystemen Arbeiten, welches nur Dateien bis zu zwei Gigabyte unterstützt.

- Innerhalb der Ausnahmebehandlung wird überprüft, ob der Fehlercode 28 ausgelöst wurde. Dieser Fehlercode bedeutet, dass kein freier Speicher mehr verfügbar ist und ist der hier erwartete Wert. Sollte jedoch ein anderer Fehlercode auftreten, zum Beispiel, weil auf die Datei nicht zugegriffen werden konnte, wird dies in dem Resultat des Moduls vermerkt und die Methode verlassen.

- Wenn alles erfolgreich war, werden die erzeugten Dateien wieder gelöscht und die Methode mit einem passenden Resultat beendet.

6.3.4. Exploit-Scanner

Mit Hilfe des Exploit-Scanners kann nach angreifbarer Software innerhalb des Images gesucht werden. Dazu wird die Datenbank der National Vulnerability Database genutzt. Da diese jedoch

nur als XML-Dokument vorliegt, muss vor der Umsetzung des eigentlichen Moduls erst ein Tool für die Konvertierung umgesetzt werden.

Datenbank

Als Datenbank-Engine kommt auch hier wieder SQLite zum Einsatz. Dies hat den Vorteil, dass kein separates Programm benötigt wird.

Die Datenbank für den Exploit-Scanner hat folgendes Schema

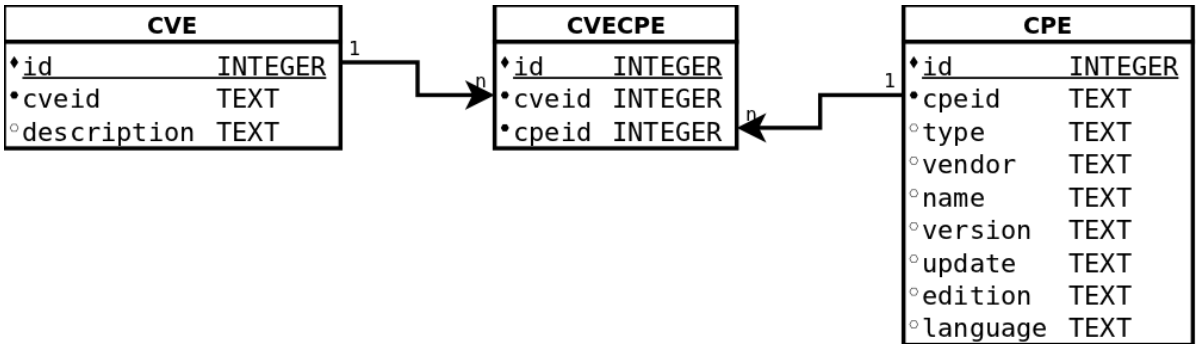

Abbildung 6.2.: Exploit-Modul - Datenbank-Schema

Nun soll kurz auf die einzelnen Tabellen eingegangen werden.

- In der Tabelle CVE werden die Informationen über die Schwachstelle selbst gespeichert. Dazu gehört die CVE-ID und die Beschreibung, nicht jedoch die betroffene Software.

- Die betroffene Software wird im Detail in de Tabelle CPE gespeichert. Hierzu werden, neben der CPE-ID selbst, auch die einzelnen Elemente in separaten Feldern gespeichert.

- Die Kombination der CVE-Tabelle und der CPE-Tabelle findet in der Tabelle CVECPE statt. Hier werden die IDs der einzelnen CVE-Einträge mit den IDs der CPE-Einträge verknüpft. An dieser Stelle werden jedoch die internen IDs der Datenbank genutzt und nicht die CVE-IDs oder CPE-IDs, da numerische Werte hier leichter zu handhaben sind.

XML Konvertierer

Mit Hilfe des Tools *nvdcveconvert.py* sollen die Daten aus den XML-Daten der National Vulnerability Database in eine SQLite-Datenbank konvertiert werden. Dadurch können die Daten leichter verarbeitet werden.

XML-Format

Im folgenden sollen die für uns wichtigen Teile des XML-Dokuments beschrieben werden:

- Nach der XML-Deklaration beginnt das Dokument mit einem *nvd*-Tag, der das gesamte Dokument umschließt.
- Innerhalb des *nvd*-Tags befinden sich mehrere *entry*-Tags. Hierbei steht jedes *entry*-Tag für einen Datensatz.
- Der *entry*-Tag besitzt ein *id*-Attribut, in welchem die ID des Eintrags steht. Dies ist in der Regel die CVE-ID.
- Unterhalb des *entry*-Tags befindet sich, unter anderem, ein Tag namens *vuln:summary*, welches die Beschreibung der Schwachstelle beinhaltet.
- Ebenfalls unterhalb des *entry*-Tags existiert ein *vuln:vulnerable-software-list-Tag*.
- Innerhalb des *vuln:vulnerable-software-list*-Tags befinden sich mehrere *vuln:product*-Tags, in denen jeweils eine betroffene Softwareversion enthalten ist.

CPE-Format

Die Angabe einer betroffenen Softwareversion ist eine zusammengesetzte Zeichenkette, welche mehrere, durch Doppelpunkt getrennte, Einträge enthält. Diese wird auch CPE genannt, was für Common Platform Enumeration steht.

Am Beispiel des Eintrags *cpe:/o:microsoft:windows_server_2008:r2:sp1:x64* sollen die einzelnen Elemente dieser Zeichenkette kurz erläutert werden:

- Der Eintrag beginnt immer mit dem Element *cpe*.

- Im Anschluss wird festgelegt, ob es sich bei dem Betroffenen System um Hardware (*/h*), ein Betriebssystem (*/o*) oder eine Applikation (*/a*) handelt.
- Dem folgt die Angabe der Herstellers, hier Microsoft.
- Anschließend folgt der Name des Produktes. In diesem Fall Windows Server 2008.
- Das nächste Element gibt die Version wieder. Im Beispiel ist die Version R2.
- An sechster Stelle steht die Angabe über die Update-Version, hier Service Pack 1.
- Nun folgt die Angabe über die Edition, also in diesem Fall x64.
- Nach der Angabe über die Edition kann noch eine Angabe über die Sprachversion (zum Beispiel *german*) stehen. Dies ist im Beispiel aber nicht der Fall.

Lediglich die ersten 3 Teile (*cpe*, Teil des Systems, Hersteller) sind zwingend notwendig. Alle anderen sind optional, wobei aber in der Regel mindestens noch die Angabe über den Namen des Produktes und der Version vorhanden sind. Leere Elemente am Ende der Zeichenkette können hierbei einfach weggelassen werden.

Da nicht genau spezifiziert ist, was eine Version ist, was eine Update-Version ist und was zum Namen des Produktes gehört kann es bei ein und dem selben Produkt auch mehrere Schreibweisen geben. In unserem Beispiel könnte die Zeichenkette daher auch folgendermaßen lauten: *cpe:/o:microsoft:windows_server_2008_r2:sp1::x64*.

Wahl des XML-Parsers

Python stellt, wie die meisten anderen Programmiersprachen auch zwei verschiedene XML-Parser zur Verfügung. Dies ist zum Einen der DOM-Parser, zum Anderen der SAX-Parser.

Der DOM-Parser parst beim Aufruf das gesamte XML-Dokument und stellt es in Form von Objekten dar. Dadurch ist er sehr flexibel und einfach einzusetzen. Die XML-Dateien der National Vulnerability Database sind jedoch teilweise sehr groß. So hat die Datei mit den Daten des Jahres 2011 eine Größe von über 80 MB. Durch diese enorme Größe stößt der DOM-Parser an seine Grenze, da der Arbeitsspeicher in der Regel nicht ausreicht, um das gesamte Dokument in Form von Objekten im Speicher abzubilden. Dadurch bleibt hier nur die Wahl des

SAX-Parsers, welcher nur wenig Arbeitsspeicher benötigt.

Der SAX-Parser parst das Dokument nur nach und nach in Einzelschritten. Hierbei kommen Events zum Einsatz. Jedes mal, wenn zum Beispiel ein Tag geöffnet oder geschlossen wird oder der textuelle Inhalt eines Elements eingelesen wird, wird eine bestimmte Funktion aufgerufen. Dadurch ist der Parser schwerer zu handhaben, da man wichtige Informationen, wie die aktuelle Ebene, selber zwischenspeichern muss. Dafür benötigt er aber nur wenig Arbeitsspeicher, da das Dokument nicht komplett im Speicher abgebildet werden muss.

Umsetzung

Um den SAX-Parser zu benutzen, muss zunächst eine neue Klasse erstellt werden, welche von der Klasse *sax.handler.ContentHandler* abgeleitet ist. Diese beinhaltet die Funktionen, welche vom SAX-Parser aufgerufen werden, wenn ein neues Event auftritt.

Diese Klasse wird hier *CVEXMLHandler* genannt und hat vier Memberfunktionen implementiert:

- *__init__* ist der Konstruktor der Klasse. Hier werden die benutzten Membervariablen initialisiert. Um den Zugriff auf Einträge in der Datenbank zu beschleunigen, werden hier einzelne Listen mit bereits in der Datenbank vorhandenen IDs erstellt. Dadurch ist es später nicht notwendig, für jede Überprüfung einer ID, eine neue Abfrage an die Datenbank zu stellen.

- *startElement* wird aufgerufen, wenn der Parser auf einen öffnenden Tag stößt. In dieser Funktion wird überprüft ob das Tag für uns interessant ist und der Name des Tags zwischengespeichert, um später überprüfen zu können, ob sich beispielsweise ein *vuln:product*-Tag unterhalb eines *vuln:vulnerable-software-list*-Tags befindet. Darüber hinaus werden einige Zähler für die spätere Import-Statistik angepasst und überprüft, ob der gerade geparste Eintrag schon in der Datenbank vorhanden ist, also übersprungen werden kann.

- *endElement* ist das Gegenstück zu *startElement* und wird aufgerufen, wenn der Parser auf einen schließenden Tag stößt. Hier wird wieder der Name des aktiven Tags angepasst. Außerdem werden die textuellen Inhalte der Tags ausgewertet. Ebenso werden in dieser Funktion die einzelnen Einträge in die Datenbank geschrieben.

- *characters* wird dann aufgerufen, wenn ein textueller Inhalt eines Elements eingelesen wird. Sofern wir uns in einem, für uns wichtigen, Tag befinden wird dieser Text zwischengespeichert, um später in der Funktion *endElement* ausgewertet zu werden.

Neben der *CVEXMLHandler*-Klasse, welche die Hauptarbeit übernimmt, sind lediglich noch der Hauptteil und zwei Funktionen von Interesse

- Im Hauptteil wird lediglich die Verbindung zur Datenbank aufgebaut und nacheinander die beiden Hilfsfunktionen *_prepare_db* und *_parse_xml* aufgerufen.

- *_prepare_db* dient hierbei der Vorbereitung der Datenbank. In dieser Funktion werden die einzelnen Tabellen in der Datenbank erzeugt, sofern diese noch nicht vorhanden sind.

- *_parse_xml* schließlich erzeugt eine Instanz der *CVEXMLHandler*-Klasse, lädt diese in den SAX-Parser und startet diesen. Anschließend gibt die Funktion noch die Statistiken aus, welche während des Parsens erzeugt wurden.

6.4. Frontends

Zuletzt wird nun noch die Implementation der Frontends dokumentiert. Diese sind für den Zugriff auf das Framework verantwortlich.

6.4.1. Frontend: Textbasiertes Menü

Mit Hilfe der textbasierten Menüs kann der Benutzer das Framework komfortabel ansteuern. Es besteht aus einer einzelnen Klasse *MenuFrontend* mit mehreren Methoden. Um die Übersichtlichkeit zu wahren sind die einzelnen Funktionalitäten des Menüs in Hilfsmethoden ausgelagert. Im Folgenden soll auf die einzelnen Methoden eingegangen werden.

- *__init__* - Dies ist der Konstruktor der Klasse. In Ihm werden die Attribute der Klasse initialisiert. Dies sind, neben den IDs des aktuell ausgewählten Scan-Moduls, beziehungsweise Zugriffs-Moduls, das *Ciss*-Objekt des Frameworks und die Liste der Resultate.

- *_cls* - Mit dieser Funktion wird die Konsole gelöscht um diese übersichtlicher machen zu können. Hierzu wird ein Systembefehl ausgeführt. Unter Windows ist das *cls*, sonst wird der Befehl *clear* benutzt

- *_runScan* - Diese Funktion führt den Scan aus. Hierzu werden zuerst, anhand der IDs, Instanzen des ausgewählten Zugriffs-Moduls und des Scan-Moduls erstellt. Anschließend wird die *runScan*-Methode des Scan-Moduls aufgerufen und dieser das Zugriffs-Modul als Parameter übergeben. Die Resultate werden an die entsprechende Liste angehängt.

- *_printResults* - Hiermit werden die Resultate auf dem Bildschirm ausgegeben. Als Trennzeichen zwischen den Feldern wird ein Tabulator benutzt. Die einzelnen Texte der Ergebnisse werden in UTF-8 umgewandelt, damit auch Umlaute korrekt angezeigt werden.

- *_writeResultsToFile* - Ähnlich der printResults-Methode werden auch hier die Ergebnisse ausgegeben. Diesesmal jedoch in die angegebene Datei. Die Datei wird mit Hilfe des *codecs*-Moduls geöffnet um auch Umlaute korrekt speichern zu können.

- *_loadModules* - Dies dient dem Laden der Module und ruft lediglich die Methode *loadModuleDir* des *Ciss*-Objektes auf. Als Verzeichnis, im welchem die Module liegen wird *modules* genutzt.

- *_printScanModuleList* - Hiermit wird eine Liste der geladenen Scan-Module ausgegeben. Hierbei wird über die Liste, welche die *getScanModules*-Methode des *Ciss*-Objektes zurück gibt, iteriert. Jedem Modul wird eine ID zugewiesen, welche gleichzeitig die Position in dieser Liste ist, beginnend bei 0.

- *_printAccessModuleList* - Analog zu *_printScanModuleList* wird hier die Liste der geladenen Zugriffs-Module ausgegeben.

- *_printActiveAccessModule* - Diese Funktion gibt die Informationen über das gerade geladene Zugriffs-Modul aus. Hierzu wird wieder die Liste der geladenen Zugriffs-Module über die Methode *getAccessModules* des *Ciss*-Objektes abgerufen. Anschließend werden die Informationen über das Modul, welches in dieser Liste an der Position der gewählten ID steht, ausgegeben.

- *_printActiveScanModule* - Hier wird analog zu *_printActiveAccessModule* das aktive Scan-Modul ausgegeben.

- *_chooseAccessModule* - Dies dient der Auswahl des aktiven Zugriffs-Moduls. Nachdem die Liste der Zugriffs-Module über *_printAccessModuleList* ausgegeben wurde, wird auf die Eingabe des Benutzers gewartet, der die ID des Zugriffs-Moduls, welches aktiviert wird, eingeben soll. Anschließend wird die Eingabe noch validiert.

- *_chooseScanModule* - Hiermit wird, analog zu *_chooseAccessModule* das aktive Scan-Modul gewählt.

- *_printMainMenu* - Diese Funktion gibt das Hauptmenü aus. Zuerst wird die Konsole gelöscht. Anschließend werden die einzelnen Menüpunkte ausgegeben, gefolgt von der Angabe des aktiven Zugriffs-Moduls und des aktiven Scan-Moduls.

- *run* - Das ist die Haupt-Funktion des Menüs und die einzige Methode, welche von Außen aufgerufen werden sollte. Diese Methode lädt zuerst, mit Hilfe der Methode *_loadModules* die Module. Anschließend zeigt es in einer Schleife immer wieder das Hauptmenü über *_printMainMenu* und wartet auf die Auswahl des Benutzers. Anschließend werden, passend zu dem gewählten Punkt, die entsprechenden Hilfsfunktionen aufgerufen. Die Schleife endet erst, wenn der Benutzer eine null eingibt, um das Programm zu beenden.

6.4.2. Frontend: AWS

Mit Hilfe des AWS-Frontends ist es möglich, Instanzen direkt innerhalb der Amazon Web Services zu scannen. Hierzu wird der Scanner auf einer eigenen kleinen Instanz installiert. Anschließend wird der Scanner mit Hilfe einer XML-Konfigurationsdatei konfiguriert und gestartet.

Die Implementation des AWS-Frontends ist um ein vielfaches komplexer, als die des Textbasierten Menüs.

Es existieren auch hier eine Reihe verschiedener Funktionen:

- *_runScan* - Auch hier führt diese Funktion den Scan aus. Als Zugriffsmodul wird automatisch das Modul für den lokalen Zugriff ausgewählt. Darüber hinaus wird der Parameter *basepath* des Zugriffsmoduls auf das Verzeichnis gesetzt, wo das Dateisystem des Images später eingebunden werden wird. Anschließend werden, anhand der Unterelemente des Parameters *scanmodules*, innerhalb der XML-Konfigurationsdatei die passenden Scanmo-

dule ausgewählt und für jedes Scanmodul der Scan gestartet.

- _ *writeSQLiteResults* - Diese Funktion speichert die Resultate in einer SQLite-Datenbank. Hierzu wird die Datenbank *results.db* geöffnet. Anschließend werden, falls noch nicht geschehen, die Tabellen erzeugt. Als nächstes werden Informationen über das aktuelle Image abgerufen und die Daten für die Datenbank vorbereitet. Nun werden die allgemeinen Daten und die Resultate in die Datenbank geschrieben.

- _ *writeSQLiteTimes* - Diese Funktion schreibt die aktuelle Zeitstatistik in die Datenbank. Es wird zuerst wieder die Datenbank geöffnet und anschließend die Daten in die Datenbank geschrieben. Diese extra Funktion ist nötig, da das schreiben der Scan-Resultate in die Datenbank noch zeitlich erfasst wird.

- _ *createTextNode* - Dies ist eine Hilfsfunktion um später die Resultate in eine XML-Datei schreiben zu können. Sie erzeugt ein einzelnes XML-Element mit textuellem Inhalt.

- _ *writeXMLResults* - Diese Funktion schreibt die aktuellen Resultate in eine XML-Datei. Hierzu wird ein neuer XML-Baum erzeugt und für jedes Resultat ein eigenes Unterelement erzeugt. Dieses enthält nun weitere Unterelemente mit den Daten des Resultats. Das XML-Dokument wird anschließend gespeichert. Dazu wird als Dateiname die ID des Images genutzt. In der aktuellen Version werden die allgemeinen Daten des Images und die Zeitdaten noch nicht im XML-Dokument gespeichert.

- _ *loadXMLConfig* - Mit Hilfe dieser Funktion werden die Konfigurationseinstellungen geladen. Hierzu wird die Datei *awsconfig.xml* geladen und die einzelnen Einträge in den passenden Variablen gespeichert.

- _ *scanImage* - Dies ist die Funktion, welche die Hauptfunktionalität des Frontends enthält. Sie wird für jedes zu scannende Image aufgerufen und führt den Scan durch. Dazu sind einige Vor- und Nacharbeiten nötig. Der genaue Ablauf dieser Funktion wird im Anschluss noch genauer behandelt.

- *run* - Dies ist die Funktion, mit der das Frontend startet. Es checkt als erstes, ob der Benutzer *root* ist, da dies nötig ist, um später die Dateisysteme einbinden zu können. Im Anschluss werden die Module geladen und eine Verbindung zu den Amazon Web Services erstellt. Nun wird für jedes, in der Konfigurationsdatei angegebene, Image die Funktion _ *scanImage* aufgerufen.

Aufgrund der Komplexität der Hauptfunktion *_scanImage* wird diese hier noch einmal detailliert beschrieben:

- Als erstes wird die Struktur für die Zeitmessung initialisiert. Es handelt sich hierbei um ein simples Array. Außerdem wird eine Variable mit der Startzeit gesetzt. Hierbei wird absichtlich, statt der für Laufzeitmessungen empfohlenen Funktion *time.clock()*, die Funktion *time.time()* genutzt, da diese unter Linux eine höhere Genauigkeit besitzt.

- Als nächstes werden die allgemeinen Informationen über das Image abgerufen und eine Instanz gestartet.

- Nun wird darauf gewartet, dass die Instanz bereit ist und die Zeit genommen.

- Anschließend wird die Instanz wieder gestoppt.

- Es wird darauf gewartet, dass sich die Instanz beendet hat und erneut die Zeit genommen.

- Im Anschluss wird der Gerätename der Hauptfestplatte anhand des Parameters *root_ device_ name* ermittelt und abgerufen.

- Die Festplatte wird nun von der Instanz getrennt und die Zeit genommen.

- Nun wird die Instanz gelöscht und darauf gewartet, dass die Festplatte verfügbar ist.

- Als nächstes wird die Festplatte an die Scan-Instanz angebunden und auf die Festplatte gewartet.

- Nun wird, sofern noch nicht vorhanden, das Verzeichnis erzeugt, in dem später das Dateisystem eingebunden werden soll und die Zeit genommen.

- Anschließend pausiert das Frontend für 30 Sekunden, um dem Betriebssystem Zeit zu geben, die neue Festplatte zu finden.

- Nun wird geprüft, ob und unter welchem Gerätenamen die Festplatte in dem Betriebssystem erschienen ist. Dies kann */dev/sd<x>* oder auch */dev/xvd<x>* sein, je nach genutzter Linux-Distribution.

- Das Dateisystem wird eingebunden und die Zeit genommen.
- Jetzt wird der eigentliche Scan gestartet.
- Nachdem den Scan abgeschlossen ist, wird das Dateisystem wieder ausgehängt und die Festplatte entfernt.
- Sobald die Festplatte wieder verfügbar ist, wird diese gelöscht und die Zeit genommen.
- Zuletzt werden die Resultate gespeichert, die letzte Zeit genommen und die Zeiten gespeichert.

6.5. Zusammenfassung

Im Rahmen dieses Kapitels wurde die Umsetzung des Scanners und diverser Module dokumentiert.

Die Funktionsweise des Modul-Systems wurde beschrieben. Darüber hinaus wurde das Konzept der Zusatzfunktionen dokumentiert.

Von den zwei konzipierten Zugriffs-Modulen wurde das Modul für den Zugriff auf das lokale Dateisystem umgesetzt. Mit Hilfe dieses Moduls ist bereits der Zugriff auf die meisten Images, teilweise mit Vorarbeiten, möglich.

Darüber hinaus wurden drei Scan-Module umgesetzt. Hiermit lässt sich bereits ein Image auf auffällige Dateien hin untersuchen. Außerdem lässt sich das Image von bereits gelöschten Daten bereinigen. Außerdem wurden Vorarbeiten für den Exploit-Scanner durchgeführt.

Zuletzt wurden noch zwei der drei Frontends umgesetzt. Mit dem textbasierten Menü lässt sich der Scanner lokal einsetzen. Das AWS-Frontend ermöglicht den automatisierten Scan von Images, welche von den Amazon Web Services bereitgestellt werden.

7. Analyse und Bewertung

Um die effektivität des entwickelten Frameworks, wird es in diesem Kapitel, anhand von 100 zufällig ausgewählten Images, getestet. Darüber hinaus werden allgemeine Statistiken zu den Amazon Web Services gesammelt, um einen besseren Überblick über die angebotenen Images zu haben.

7.1. Vorbereitung

Im die Evaluation vorbereiten zu können, werden zunächst Informationen über die Images benötigt, welche bei den Amazon Web Services zur Verfügung stehen. Dies wird mit Hilfe des Tools AWSImages realisiert.

7.1.1. AWSImages-Tool

Im folgenden wird ein Tool namens AWSImages entwickelt. Dieses ist für die Auswertung der Daten, welche die Amazon Web Services zu den einzelnen Images zur Verfügung stellt, zuständig.

Konzeption

Das Tool AWSImages dient der automatisierten Erfassung von Informationen über Images, welche bei den Amazon Web Services zur Verfügung stehen. Hierzu nutzt das Tool die boto-Bibliothek, um diese Informationen abzurufen. Anschließend sollen diese Informationen weiter

spezifiziert werden. Zuletzt sollen alle Informationen in einem XML-Dokument gespeichert werden. Dadurch sind diese später einfacher, für statistische Auswertungen, zu verarbeiten.

Amazon stellt für die einzelnen Images eine große Anzahl an Informationen bereit. Jedoch sind nicht alle Informationen brauchbar, bzw. teilweise auch unvollständig.

Im Folgenden sollen die einzelnen Informationen kurz beschrieben werden:

- *id* - Stellt eine eindeutige ID bereit, um das Image eindeutig identifizieren zu können
- *location* - Der Pfad und Name des Images. Dieser kann auch weitere Informationen, wie der Name der Distribution enthalten.
- *state* - Gibt an, ob das Image gerade verfügbar ist.
- *owner_id* - Die ID des Besitzers des Images.
- *owner_alias* - Sofern vorhanden, der Alias des Besitzers.
- *is_public* - Gibt an, ob das Image öffentlich ist.
- *architecture* - Gibt die Architektur des Images an (*i386* für 32-Bit Images oder *x86_64* für 64-Bit Images).
- *platform* - Die Plattform des Images. Enthält *windows* für Windows-Images, sonst *None*.
- *type* - Der Typ des Images (*kernel*, *ramdisk* oder *machine*).
- *kernel_id* - Die ID des Kernels, mit dem das Image gestartet werden soll.
- *ramdisk_id* - Sofern benötigt, die ID der Ramdisk, mit der das Image gestartet werden soll.
- *name* - Der Name des Images, welcher bei der Veröffentlichung festgelegt wurde.
- *description* - Die Beschreibung des Images, welche bei der Veröffentlichung festgelegt

wurde.

- *product_ codes* - Produkt-Codes, welche mit dem Image verknüpft sind (AWS Devpay-IDs oder AWS Marketplace-IDs)
- *block_ device_ mapping* - Liste der virtuellen Laufwerke, welche das Image enthält.
- *root_ device_ type* - Der Typ des Boot-Laufwerks (*instance-store* oder *ebs*).
- *root_ device_ name* - Der Gerätename des Boot-Laufwerks.
- *virtualization_ type* - Der Typ der Virtualisierung (*paravirtual* oder *hvm*).
- *hypervisor* - Der genutzte Hypervisor (*xen* oder *ovm*).
- *instance_ lifecycle* - Aktuell ungenutzter Parameter.

Das Attribut *block_ device_ mapping* besitzt darüber hinaus für jedes Laufwerk weitere Attribute. Einige dieser Attribute werden jedoch nur bei einer laufenden Instanz genutzt:

- *ephemeral_ name* - Ungenutzter Parameter
- *no_ device* - Gibt an, ob das Laufwerk ins Leere führt.
- *volume_ id* - Die AWS-ID des Laufwerks im laufenden Betrieb. Wird bei einem Image nicht genutzt.
- *snapshot_ id* - Die ID des Snapshots, von dem das Laufwerk ursprünglich erstellt wurde.
- *status* - Der Status des Laufwerks (ob es an die Instanz gerade angebunden ist oder nicht). Wird bei einem Image nicht genutzt.
- *attach_ time* - Gibt an, wann das Laufwerk an die Instanz angebunden wurde. Wird bei einem Image nicht genutzt.
- *delete_ on_ termination* - Gibt an, ob das Laufwerk gelöscht werden soll, wenn die Instanz

beendet wird.

- *size* - Liefert die Größe des Laufwerks in Gigabyte.

Umsetzung

Um überhaupt eine Verbindung zu den Amazon Web Services herstellen zu können, werden Zugangsdaten hierfür benötigt. Diese besorgt sich das Tool zunächst aus der Datei *awsconfig.xml*, welche auch von dem AWS-Frontend genutzt wird.

Anschließend stellt es eine Verbindung her und ruft mittels der Funktion *get_all_images* eine Liste aller verfügbaren Images. Dies kann, je nach Geschwindigkeit der Internetverbindung, einige Zeit in Anspruch nehmen.

Diese Funktion liefert eine Liste der Images, welche anschließend in ein XML-Dokument umgewandelt werden muss. Hierbei wird jedes Image in einem *image*-Tag gekapselt. Dieser *image*-Tag erhält noch ein Attribut namens *xmlid*. Dieses Attribut enthält eine fortlaufende Nummer, um die Datensätze später leichter verarbeiten zu können.

Bevor nun die einzelnen Attribute eines Images verarbeitet werden, wird zuerst das Attribut *platform* modifiziert. Dieses Attribut enthält bei Windows-Images den Inhalt *windows*, bei Linux jedoch *None*. Um dieses Tag für Linux-Images brauchbar zu machen, wird der Name des Images, welcher in dem Attribut *location* steht, untersucht. Dies geschieht mit Hilfe einer Liste aus regulären Ausdrücken, welche die Namen bekannter Linux-Distributionen, wie Debian, Ubuntu, Redhat, Suse, Centos oder Gentoo, repräsentieren. Sofern der Vergleich mit einem der regulären Ausdrücke erfolgreich ist, wird das Attribut *platform* durch den Namen der Distribution ersetzt. Dadurch können die Images später anhand Ihrer Distribution klassifiziert werden.

Die meisten der Attribute der einzelnen Images können direkt als untergeordnete Tags gespeichert werden. Hierzu gehören die Attribute: *id, location, state, owner_id, owner_alias, is_public, architecture, platform, type, kernel_id, ramdisk_id, name, description, root_device_type, root_device_name, virtualization_type, hypervisor* und *instance_lifecycle*.

Lediglich die Attribute *product_codes* und *block_device_mapping* benötigen weitere Aufmerksamkeit, da diese aus Listen beziehungsweise Objekten bestehen.

Hierbei ist *product_ codes* eine simple Liste. Diese kann iteriert werden. Der übergeordnete Tag hat den Namen *product_ codes*. Dieser enthält untergeordnete Tags namens *product_ code*. Jedes product_code-Tag repräsentiert hierbei einen Eintrag in der Liste.

Das Attribut *block_ device_ mapping* ist etwas komplexer zu handhaben. Es besteht aus dem Datentyp *dictionary*. Dies ist eine Schlüssel -> Wert-Zuweisung. Hierbei ist der Schlüssel ein Gerätename der Festplatte. Der Wert ist ein *BlockDeviceType*-Objekt. Dieses enthält weitere Attribute. Das übergeordnete Tag trägt den Namen *block_ device_ mapping*. Das Tool iteriert nun über die einzelnen Schlüssel -> Wert-Paare und erstellt für jedes Paar ein untergeordnetes Tag namens *block_ device*. Dieses Tag enthält ein Attribut namens *name*, welches den Gerätenamen, also den Schlüssel des Paares, enthält. Anschließend wird für jedes Attribut des *BlockDeviceType*-Objektes ein weiteres, dem Tag *block_ device* untergeordnetes, Tag erstellt.

Das erzeugte XML-Dokument wird zuletzt mittels der Funktion *writexml* gespeichert.

7.2. Allgemeine Statistiken

Zu allererst werden einige Statistiken über die Images in den Amazon Web Services [Ama12] gesammelt. Hierzu wird das bereits angesprochene AWSImages-Tool genutzt und dieses Tool um einige statistische Anzeigen erweitert. Im Folgenden werden die Ergebnisse dargestellt.

Sämtliche allgemeinen Statistiken wurden am 17.07.2012, um 7:00 Uhr aufgenommen.

7.2.1. Typen

Es gibt bei den Amazon Web Services [Ama12] 3 verschiedene Typen an Images. Maschinen-Images enthalten die Daten der virtuellen Maschinen selbst. Kernel-Images enthalten einen Kernel. Dieser wird vom Hypervisor geladen. Dadurch kann auch ein Maschinen-Image mit einem anderen Kernel gestartet werden, ohne das Maschinen-Image verändern zu müssen. Zuletzt gibt es noch Ramdisk-Images. Diese enthalten, analog zu den Kernel-Images, eine Ramdisk für die Maschinen-Images. Dies ist nur nötig, falls das Maschinen-Image keine Ramdisk mitliefert und der Kernel nicht alle Treiber mitliefert, um das Dateisystem des Maschinen-Images ansprechen zu können. Eine kurze Dokumentation zu Kernel- und Ramdisk-Images findet sich in der

offiziellen Dokumentation der Amazon Web Services [AWS12a].

Es ist zu erwarten, dass es vielmehr Maschinen-Images als Kernel- oder Ramdisk-Images gibt, da diese auch für mehrere Maschinen-Images genutzt werden können.

Tabelle 7.1.: Verteilung der Image-Typen

Typ	Anzahl	Anteil in [%]
Machine	11197	93,54
Kernel	433	3,62
Ramdisk	340	2,84
Gesamt	11970	100

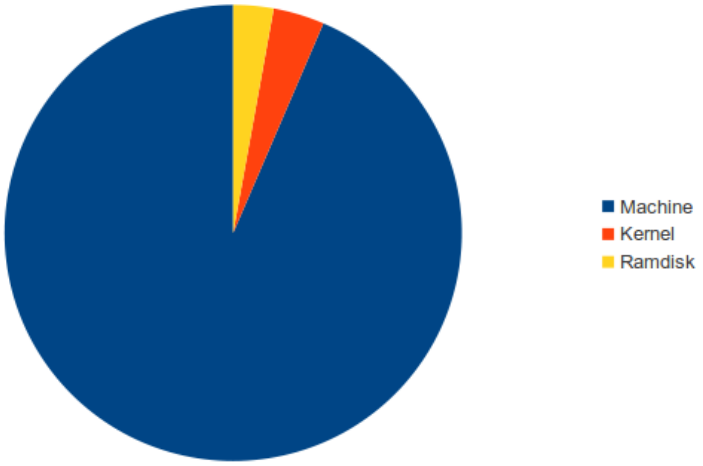

Abbildung 7.1.: Verteilung der Image-Typen

Wie erwartet, gibt es mit 11197 Maschinen-Images fast 26 mal so viele Maschinen-Images, wie Kernel-Images. Im Gegensatz zu den Ramdisk-Images sind es sogar fast 33 mal so viele Maschinen-Images. Insgesamt existieren 11970 verschiedene Images.

In den weiteren Kapiteln werden nur noch die 11197 Maschinen-Images untersucht.

7.2.2. Distributionen

Bei den Amazon Web Services [Ama12] werden Maschinen-Images mit einer ganzen Reihe verschiedenster Distributionen angeboten. Jedoch lässt sich die Info, welche Distribution auf welchem Image läuft, nicht ohne Weiteres herausfinden, da hierzu keine Informationen geliefert

werden. Lediglich Windows-Images werden eindeutig als solche ausgegeben. Daher muss hier die genutzte Distribution anhand des Image-Namens detektiert werden. Dies ist jedoch nicht bei allen Images möglich, so dass es auch Images gibt, bei denen es nicht ersichtlich ist, welche Distribution auf dem Image läuft. Die Distribution ließe sich nur durch eine eingehende Analyse des Systems im laufenden Betrieb genau herausfinden. Es kann jedoch davon ausgegangen werden, dass sich die Distributionen in den unbekannten Images ähnlich zusammensetzen, wie die Distributionen in den bekannten Images. Es lassen sich allerdings auch für unbekannte Images weitere Daten, wie die Architektur, der genutzte Hypervisor, die Festplattengröße und andere Daten, welche die API der Amazon Web Services zurückgeben, nutzen. Eine Liste dieser Daten findet sich in Kapitel 7.1.1.

Es wird erwartet, dass viele Images mit einer unbekannten Distribution laufen, da diese nicht ohne eingehendere Analyse erkannt werden kann. Es wird darüber hinaus wohl einige Windows-Images geben, da diese eindeutig als solche identifiziert werden können. Alle weiteren Distributionen werden wohl in in etwa in gleichem Maße vertreten sein.

Tabelle 7.2.: Verteilung der Distributionen

Distribution	Anzahl	Anteil in [%]
Unbekannt	5665	50,59
Suse	27	0,24
Fedora	164	1,46
Centos	426	3,80
Amazon	61	0,54
Redhat	174	1,55
FreeBSD	15	0,13
Ubuntu	3333	29,77
Debian	151	1,35
Gentoo	31	0,28
Windows	1150	10,27
Gesamt	11197	100

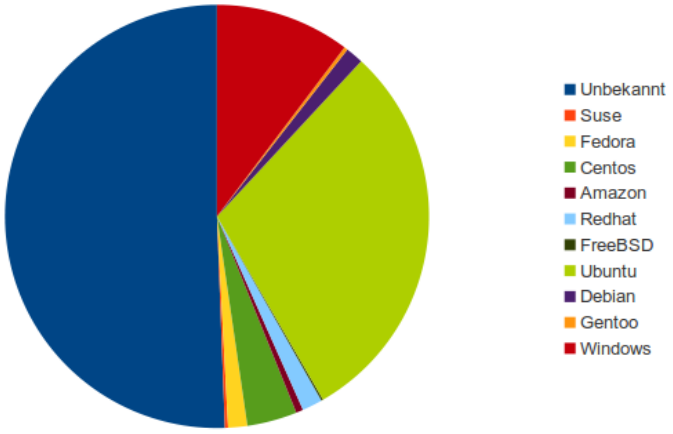

Abbildung 7.2.: Verteilung der Distributionen

Wie erwartet, konnten etwa 50% der Images nicht identifiziert werden. Auch sind mit rund 10% einige Windows-Images dabei. Überraschend ist jedoch der große Unterschied bei den Linux-Distributionen. Hier sticht eindeutig Ubuntu mit einem Anteil von fast 30% heraus, während alle anderen Distributionen zusammen nicht mal 10% ausmachen. Dies lässt sich lediglich durch die große Ubuntu-Community erklären.

7.2.3. Architekturen

In den Amazon Web Services werden 2 verschiedene Rechnerarchitekturen unterstützt: 32 Bit und 64 Bit PC-Prozessoren. Die noch relativ neue Möglichkeit, auch eine ARM-Architektur zu virtualisieren [Tec12], wird nicht unterstützt. Die Hersteller von Betriebssystemen empfehlen nach und nach, Server mit einer 64 Bit-Architektur zu betreiben [Pho11].

Aufgrund der Empfehlung der Hersteller, ist zu erwarten, dass es weitaus mehr 64 Bit, als 32 Bit Images gibt.

Tabelle 7.3.: Verteilung der Architekturen

Architektur	Anzahl	Anteil in [%]
x86_64	5397	48,20
i386	5800	51,80
Gesamt	11197	100

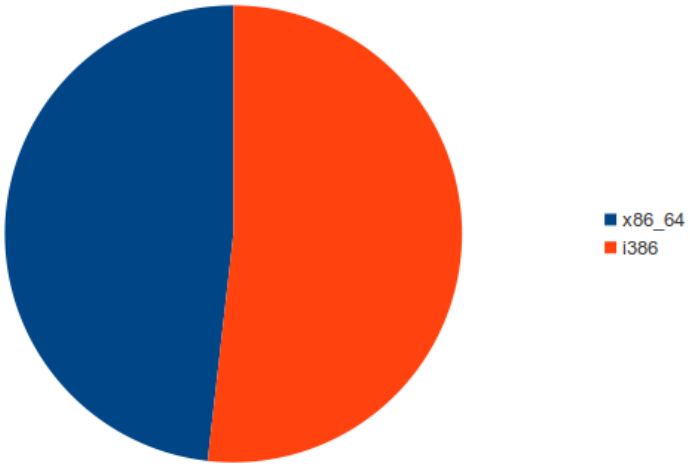

Abbildung 7.3.: Verteilung der Architekturen

Auch dieses Ergebnis überrascht. Entgegen der Erwartungen, ist die Verteilung von 32 Bit und 64 Bit Images in etwa gleich. Eine Erklärung für dieses Phänomen wäre, dass 64 Bit-Architekturen noch nicht so lange empfohlen werden. Die Anzahl der 64 Bit-Architekturen sollte aber in den nächsten Jahren weiter steigen.

7.2.4. Speicherverfahren

Um die Daten der Images zu speichern, unterstützen die Amazon Web Services 2 verschiedene Speicherverfahren: Instance Store und Electronic Block Storage (EBS) [AWS08]. Bei Instance Store sind die Daten nur vorübergehend vorhanden und verschwinden, sobald die Instanz wieder gelöscht wird. EBS hingehen arbeitet mit virtuellen Festplatten, die ähnlich, wie echte Festplatten gehandhabt werden können. Diese können an eine Instanz angehängt und wieder abgehängt werden. Die Daten auf dieser Platte sind auch noch vorhanden, wenn die Instanz gestoppt wird. Erst, wenn die Instanz gelöscht wird, kann, je nach Einstellung, auch die EBS-Platte gelöscht werden. Im Gegensatz zum Instance Store, muss der Benutzer von EBS jedoch ein zusätzliches Entgelt bezahlen. Dieses richtet sich nach der Größe der virtuellen Festplatten.

Es wird oft empfohlen, EBS zu benutzen, da es besser zu verwalten und persistent ist [Ale12]. Es ist daher zu erwarten, dass die Nutzung von EBS und Instance Store in etwa ausgeglichen ist, da Instance Store schon länger verfügbar ist, EBS aber empfohlen wird.

Tabelle 7.4.: Verteilung der Speicherverfahren

Verfahren	Anzahl	Anteil in [%]
EBS	5221	46,63
Instance-Store	5976	53,37
Gesamt	11197	100

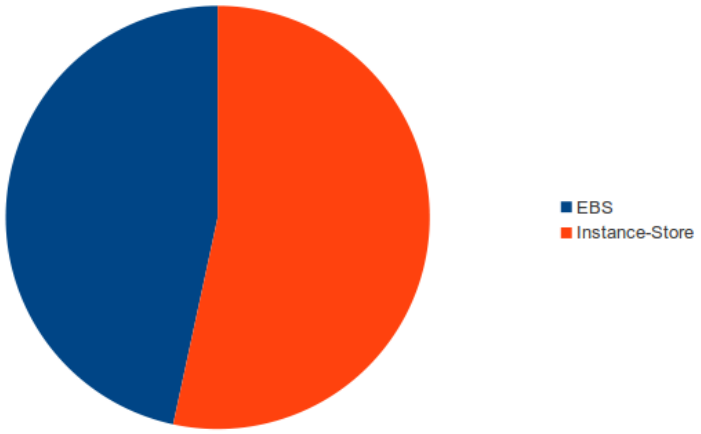

Abbildung 7.4.: Verteilung der Speicherverfahren

Wie erwartet, wird Instance Store und EBS in etwa gleich oft benutzt. Instance Store wird nur geringfügig öfter benutzt. Es ist jedoch zu erwarten, dass es in Zukunft immer mehr EBS-Images geben wird und Instance Store mit der Zeit nur noch eine untergeordnete Rolle spielen wird.

7.2.5. Hypervisors

Amazon bietet 2 verschiedene Hypervisors an. Den XEN-Hypervisor [XEN12] und die Oracle Virtual Machine (OVM) [OVM12]. Während XEN für jedes Betriebssystem nutzbar ist, wurde OVM auf einige spezielle Produkte von Oracle spezialisiert und kann auch, innerhalb der Amazon Web Services, nur für diese verwendet werden [AWS12f].

Es ist, aufgrund dieser Beschränkung zu erwarten, dass es lediglich einige wenige ganz spezielle Images gibt, die OVM nutzen. Der größte Teil der Images wird XEN benutzen.

Tabelle 7.5.: Verteilung der Hypervisors

Hypervisor	Anzahl	Anteil in [%]
XEN	11159	99,66
OVM	38	0,34
Gesamt	11197	100

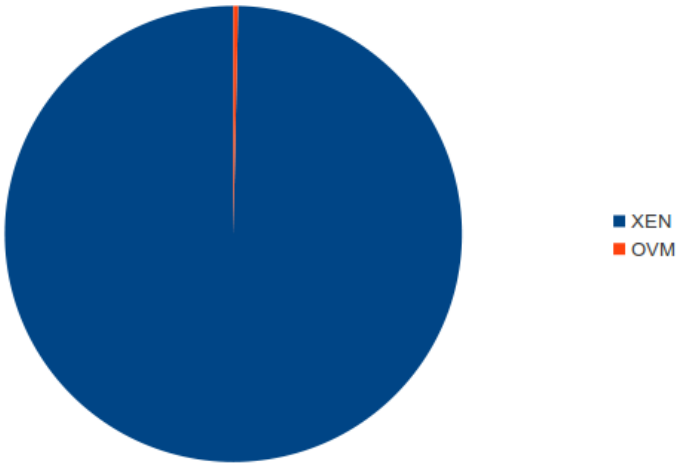

Abbildung 7.5.: Verteilung der Hypervisors

Wie erwartet existieren lediglich 38 Images, welche OVM als Hypervisor nutzen. Über 99% der Images nutzen hingegen XEN als Hypervisor.

7.2.6. Virtualisierungstypen

Es existieren für die Images 2 verschiedene Virtualisierungstypen. Diese sind die Paravirtualisierung und die Vollvirtualisierung (HVM) [XEN11]. Hierbei entspricht die Vollvirtualisierung der Bereitstellung eines virtuellen Rechners, auf dem ein Betriebssystem ohne Änderungen installiert werden kann. Das Betriebssystem muss also nicht wissen, dass es auf einer virtuellen Maschine läuft. Bei der Paravirtualisierung hingegen muss das Betriebssystem mit dem Hypervisor zusammenarbeiten. Es ist also nicht möglich, das gleiche Betriebssystem zu verwenden, als bei einer physikalischen Maschine ohne Hypervisor. Daher ist auch Paravirtualisierung nur mit Linux-Betriebssystemen möglich, da Windows keine Möglichkeit zur Paravirtualisierung besitzt.

Man könnte erwarten, dass die Vollvirtualisierung hier erheblich öfter genutzt wird, als die

Paravirtualisierung, da diese keine Modifikation des Betriebssystems benötigt. Jedoch schränkt Amazon die Verwendung der Vollvirtualisierung ein. So ist es nur möglich, die Vollvirtualisierung bei Windows-Images (da diese keine Paravirtualisierung unterstützen) und bei großen Cluster-Images [AWS12c] zu nutzen. Es ist daher eher zu erwarten, dass die meisten Linux-Images die Paravirtualisierung nutzen und es daher einen größeren Anteil paravirtualisierter Images, als vollvirtualisierter Images geben wird.

Tabelle 7.6.: Verteilung der Virtualisierungstypen

Typ	Anzahl	Anteil in [%]
HVM	1319	11,78
Paravirtual	9878	88,22
Gesamt	11197	100

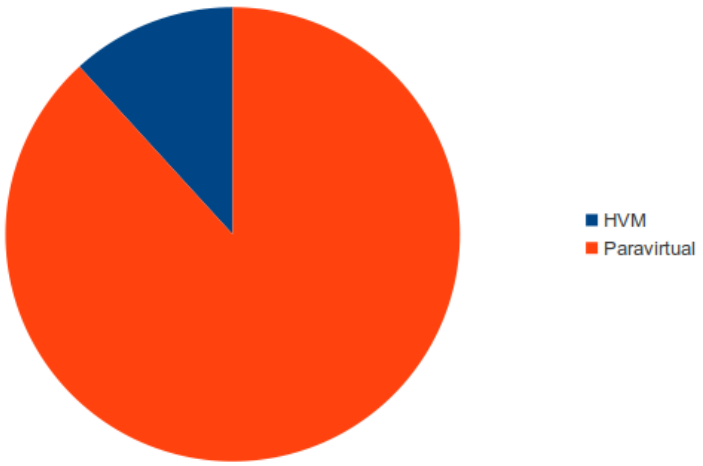

Abbildung 7.6.: Verteilung der Virtualisierungstypen

Tatsächlich machen die paravirtualisierten Images mit über 88% den Großteil der Images aus. Lediglich 1319 von 11197 Images nutzen die Vollvirtualisierung. Zieht man hiervon die 1150 Windows-Images ab, bleiben lediglich noch 169 vollvirtualisierte Linux-Images übrig, welche ausschließlich auf großen Cluster-Instanzen betrieben werden können.

7.2.7. Kernel

Der Kernel ist die zentrale Komponente eines Betriebssystems. Bei ihm laufen alle Systemzugriffe zusammen. Daher ist es besonders wichtig, den Kernel regelmäßig zu aktualisieren.

Bei den Amazon Web Services ist es daher möglich, bzw. auch nötig, den Kernel anzugeben, welcher genutzt werden soll. Dazu stellt Amazon verschiedenste Kernel in Form von Kernel-Images zur Verfügung. Daher ist es auch möglich, den Kernel zu ändern, ohne das Maschinen-Image ändern zu müssen. Der Kernel wird dann direkt vom Hypervisor beim Start einer Instanz geladen. Lediglich vollvirtualisierte Images benötigen keinen externen Kernel, da diese ein System mit Bootloader zur Verfügung gestellt bekommen, welcher den Kernel aus dem Maschinen-Image lädt.

Um bei einem paravirtualisierten Image einen eigenen, nicht externen Kernel nutzen zu können, muss als Kernel der spezielle Kernel *PV-Grub* genutzt werden [AWS12e]. Dieser stellt dann einen Bootloader bereit, welcher wieder den eigentlichen Kernel aus dem Maschinen-Image lädt.

Es ist zu erwarten, dass die meisten Images standardmäßig den PV-Grub-Kernel nutzen. Dadurch können diese dann einen eigenen Kernel auf dem Maschinen-Image selbst installieren und sind nicht auf die Kernel von Amazon angewiesen. Dazu müssten viele Images vertreten sein, die gar keinen Kernel angegeben haben, da diese Images vollvirtualisiert sind, wie beispielsweise Windows-Images. Dahinter dürften vor allem Ubuntu-Kernel kommen, da diese Distribution den größten Teil der Images ausmacht.

Tabelle 7.7.: Meist genutzte Kernel

Kernel	Name	Anzahl	Anteil in [%]
Kein Kernel	-	2774	24,77
aki-825ea7eb	pv-grub-hd0_1.02-i386	1194	10,66
aki-407d9529	pv-grub-hd0-V1.01-i386	1157	10,33
aki-427d952b	pv-grub-hd0-V1.01-x86_64	1087	9,71
aki-805ea7e9	pv-grub-hd0_1.02-x86_64	938	8,38
aki-a71cf9ce	ec2-vmlinuz-2.6.21.7-2.fc8xen.i386	762	6,81
aki-0b4aa462	ubuntu-lucid-amd64-linux-image-2.6.32	703	6,28
aki-b51cf9dc	ec2-vmlinuz-2.6.21.7-2.fc8xen.x86_64	397	3,55
aki-754aa41c	ubuntu-lucid-i386-linux-image-2.6.32	360	3,22
aki-6eaa4907	vmlinuz-2.6.21-2.fc8xen-ec2-v1.0.i386	170	1,52
Restliche	-	1655	14,78
Gesamt	-	11197	100,00

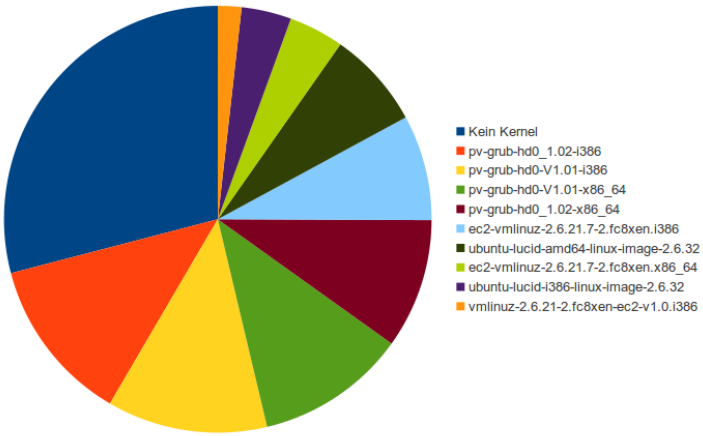

Abbildung 7.7.: Meist genutzte Kernel-Images

Wie erwartet, nutzen die meisten Images einen PV-Grub-Kernel in seinen verschiedenen Versionen. Die vier PV-Grub-Kernel-Images zusammen sind bei 4376 Images als Standardkernel angegeben.

Dahinter kommen 2774 Images, welche keinen Kernel angegeben haben. Dies überrascht jedoch, da es insgesamt nur 1319 vollvirtualisierte Images gibt. Bleiben also 1455 paravirtualisierte Images, welche keinen Standardkernel angegeben haben. Es gibt jedoch keine Informationen in der Dokumentation der Amazon Web Services, was in einem solchen Fall geschieht. Eine erste Vermutung ist, dass diese Images ohne die manuelle Angabe eines Kernels nicht starten. Ein schneller Test lässt jedoch die Vermutung zu, dass in einem solchen Fall automatisch ein passender PV-Grub-Kernel geladen wird, da Maschinen-Images, welche einen Kernel im Dateisystem installiert haben, trotzdem mit diesem starten. Images ohne einen eigenen Kernel oder einem unkonfigurierten Kernel starten hingegen ohne die Angabe eines Standardkernels nicht.

Auch werden, wie erwartet, viele Ubuntu-Kernel genutzt. Insgesamt 1063 Images nutzen einen der beiden, hier aufgeführten Ubuntu-Kernel.

Eine Überraschung ist allerdings der große Anteil an Kerneln mit der Bezeichnung *fc8xen*. Die drei hier vertretenen Kernel mit dieser Bezeichnung werden von 1329 Images genutzt. Eine kurze Recherche ergibt, dass dieser Kernel zu der Distribution Fedora in der Version 8 gehört [FC808]. Dies verwundert um so mehr, da die Distribution unter allen Maschinen-Images nur 164 mal detektiert wurde. Eine Erklärung wäre, dass viele Images mit unbekannter Distribution Fedora-Images sind. Eine kurze Analyse ergibt jedoch auch, dass dieser Kernel auch von Images mit anderen Distributionen genutzt wird. Um genau zu sein, wurden 123 Images

mit Centos-Distribution gefunden, welche diesen Kernel nutzen. Dies ist jedoch ein Nachbau von Fedora. Aber auch 292 Ubuntu-Images, 101 Debian-Images und ein Gentoo-Image nutzen diesen Kernel.

7.2.8. Ramdisks

In einer Ramdisk sind Dateien gespeichert, welche schon bei der Initialisierung des Kernels benötigt werden. Dies sind in der Regel Module für den Zugriff auf das Dateisystem, können aber auch Konfigurationsdateien oder Ähnliches sein. Es ist jedoch auch möglich, die benötigten Module fest in den Kernel einzubauen. In einem solchen Fall ist keine Ramdisk mehr nötig, um das System starten zu können.

Es ist zu erwarten, dass die meisten Images keine Ramdisk benutzen, da oft Kernel benutzt werden, welche keine separate Ramdisk benötigen. Dies gilt vor allem für den PV-Grub-Kernel, welcher in den meisten Images genutzt wird. Nach der Auswertung der Kernel ist zu erwarten, dass einige Ubuntu-Images und Fedora-Images eine Ramdisk benutzen und daher in der Statistik auftauchen werden.

Tabelle 7.8.: Meist genutzte Ramdisks

Ramdisk	Name	Anzahl	Anteil in [%]
Keine Ramdisk	-	9162	81,83
ari-a51cf9cc	ec2-initrd-2.6.21.7-2.fc8xen.i386	769	6,87
ari-b31cf9da	ec2-initrd-2.6.21.7-2.fc8xen.x86_64	400	3,57
ari-42b95a2b	initrd-2.6.21.7-2.fc8xen-ec2-v1.0.1.i386	158	1,41
ari-7cb95a15	initrd-2.6.21.7-2.fc8xen-ec2-v1.0.1.x86_64	146	1,30
ari-94ba58fd	ubuntu-hardy-i386-linux-image-2.6.24	55	0,49
ari-dbc121b2	initrd-2.6.18-xenU-ec2-v1.2.i386	34	0,30
ari-7b739e12	ubuntu-karmic-amd64-linux-image-2.6.31	33	0,29
ari-56ce2c3f	initrd-2.6.21.7-2.ec2.v1.2.fc8xen.x86_64	30	0,27
ari-25d43a4c	OEL5.5-x86_64	25	0,22
Restliche	-	385	3,44
Gesamt	-	11197	100,00

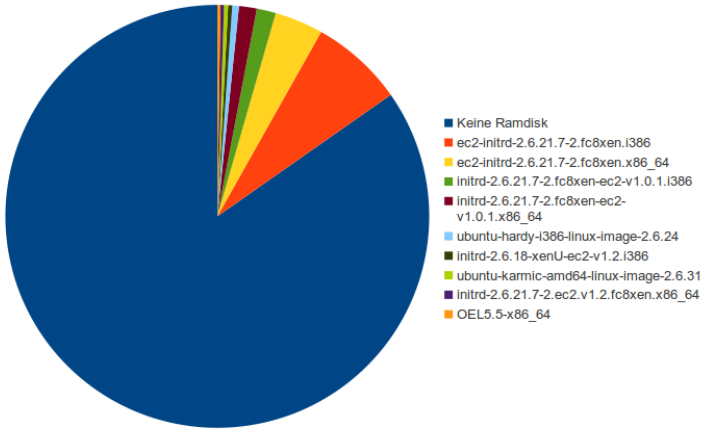

Abbildung 7.8.: Meist genutzte Ramdisk-Images

Wie bereits erwartet, nutzen die meisten Images, um genau zu sein 9162, standardmäßig keine Ramdisk.

Dahinter kommen insgesamt fünf Ramdisks, welche zu den Fedora-Kerneln gehören. Diese Ramdisks werden von insgesamt 1503 Maschinen-Images genutzt.

Ebenso wie erwartet, sind auch zwei Ubuntu-Ramdisks vertreten. Diese werden von 88 Maschinen-Images genutzt. Auffällig ist hierbei, dass es sich lediglich um Ramdisks der älteren Ubuntu-Versionen *Hardy* und *Karmic* handelt. Bei den meist genutzten Kerneln sind jedoch hauptsächlich Kernel der Ubuntu-Version *Lucid*, welche auf *Karmic* folgte, vertreten. Dies lässt die Vermutung zu, dass Ubuntu-Kernel für die Cloud ab *Lucid* keine Ramdisk mehr benötigen. Eine kurze Recherche bestätigt diese Vermutung [Mos09].

Außerdem ist noch eine Ramdisk mit dem Namen *initrd-2.6.18-xenU-ec2-v1.2.i386*, welche von 34 Images genutzt wird, vertreten. Jedoch ist dieser Name nicht sehr aussagekräftig, wodurch es nicht möglich ist, eine Distribution zuzuordnen.

Zuletzt wird noch eine Ramdisk mit dem Namen *OEL5.5-x86_64* von 25 Images genutzt. Dieses Image gehört zu Oracle Enterprise Linux in der Version 5.5, welches in Oracle Linux umbenannt wurde [Ora12].

7.3. Evaluation

Im weiteren Verlauf wird nun das Framework getestet. Hierzu werden zufällig 100 Images aus dem Vorrat der Amazon Web Services gewählt und mit Hilfe des Dateinamen-Scanners und des Dateiinhalts-Scanners gescannt. Als Frontend kommt das AWS-Frontend zum Einsatz. Bevor jedoch die Images gescannt werden, müssen noch Vorbereitungen getroffen werden.

7.3.1. Vorbereitungen

Leider können nicht alle Images aus dem Vorrat für einen Scan ausgewählt werden. Daher muss als Erstes der Vorrat nach bestimmten Kriterien eingeschränkt werden:

- Speicherverfahren - Instance Store basierte Images können, im Gegensatz zu den EBS basierten Images, nicht gescannt werden, da diese keine virtuelle Festplatte besitzen. Es wäre daher nur möglich, das Image von innen im laufenden Betrieb zu scannen, was aber aktuell nicht automatisiert geschehen kann. Daher müssen alle Images, welche Instance Store benutzen über den Parameter *root_ device_ type* herausgefiltert werden.

- Windows-Images - Es ist mit dem Tool in seiner jetzigen Funktion nicht möglich, Windows-Images zu scannen, da diese andere Filter benötigen als Linux-Images, um auffällige Dateien zu finden. Daher werden alle Windows-Images anhand des Parameters *platform* herausgefiltert.

- Produkt-Codes - Aus Kostengründen ist es nicht möglich Images zu scannen, welche Produkt Codes enthalten, da für diese Images erst eine separate Lizenz erworben werden muss. Daher werden alle Images, bei denen der Parameter *product_ codes* nicht leer ist, herausgefiltert.

- Virtualisierungstyp - Da Linux-Images, welche mittels Vollvirtualisierung betrieben werden, nur mit Hilfe teurer Cluster-Instanzen betrieben werden können, werden diese über den Parameter *virtualization_ type* herausgefiltert.

- Mehrere Partitionen - Der Scanner unterstützt aktuell nicht die Möglichkeit, ein Image zu scannen, welches über mehrere Partitionen verteilt ist. Daher werden diese Images

herausgefiltert. Jedoch ist das automatisierte erkennen eines Images mit mehreren Partitionen etwas aufwändiger, als dies bei den vorangegangenen Punkten der Fall ist. Darüber hinaus betrifft dies nur wenige Images. Daher werden diese Images von Hand herausgefiltert.

Nachdem nun die Images gefiltert wurden, bleiben noch 3717 Images übrig. In dieser Zahl sind jedoch noch die Images enthalten, welche mehr als eine Partition enthalten. Diese werden erst später, bei der Auswahl der 100 zufälligen Images berücksichtigt.

7.3.2. Testablauf

Mittels eines Zufallsgenerators werden nun 100 Images ausgewählt. Sollte hierbei ein Image ausgewählt werden, welches mehr als eine Partition enthält, wird dieses durch ein anderes Image ersetzt. Um die Images zu scannen, wird das AWS-Frontend des Frameworks genutzt. Der Scanner selbst läuft auf einer eigenen Instanz innerhalb der Amazon Web Services.

Jedoch existiert innerhalb der Amazon Web Services noch ein Bug, welcher das automatisierte Scannen der Images verhindert [AWS12d]. Es ist nicht möglich, einen Gerätenamen für eine Festplatte mehr als einmal zu benutzen, ohne das System neu starten zu müssen. Es ist also dem Scanner nicht möglich, die einzelnen Festplatten nacheinander an das gleiche Gerät der Scanner-Instanz anzuhängen. Daher wird das AWS-Frontend hier um einen Workaround ergänzt. Nach jedem Image, welches gescannt wurde, wird der Gerätename um einen Buchstaben erhöht. Dadurch ist es zumindest möglich, 20 Images am Stück zu scannen, bevor die Scanner-Instanz neu gestartet werden muss.

Nachdem dieses Problem gelöst ist, können nun nach und nach die Images in die Datei *awsconfig.xml* eingetragen und gescannt werden. Alle 20 Images wird die Scanner-Instanz einmal neu gestartet und die nächsten 20 Images können gescannt werden. Die Ergebnisse werden in einer SQLite-Datenbank gespeichert. Nachdem alle Images gescannt wurden, können die Ergebnisse ausgewertet werden.

7.3.3. Ergebnisse

Nachdem alle Images gescannt wurden, werden die Ergebnisse aufbereitet. Hierzu werden die Daten aus der SQLite-Datei *results.db*, welche vom AWS-Frontend angelegt wurde, ausgewertet. Dies umfasst als erstes die Funde an sich. Im Anschluss werden die Daten der Zeitmessung ausgewertet.

Funde

Die Funde können primär nach der Kategorie eingeteilt werden. Es gibt sowohl Funde der Kategorie *Critical*, also kritische Funde, als auch Funde der Kategorie *Warn*, welche Warnungen repräsentiert. Sekundär können noch die Funde nach dem eingesetzten Filter unterschieden werden. Dies soll auch die Grundlage der ersten Auswertungen sein.

Als erstes werden alle Funde der Kategorie *Critical* nach Filtern geordnet zusammengetragen:

Tabelle 7.9.: Funde der Kategorie *Critical*

Filter	Funde	Anteil in [%]
GPG-/PGP-files	473	65,42
Private key	106	14,66
History of shell-commands	68	9,41
Athorized keys for ssh-connection	61	8,44
Private DSA-/RSA-keys	15	2,07
Gesamt	723	100

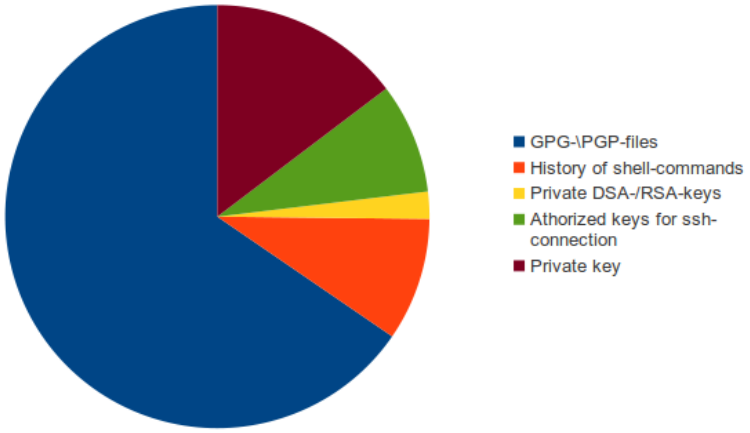

Abbildung 7.9.: Funde der Kategorie *Critical*

Wir sehen, dass der Filter *GPG-/PGP-files* mit 473 Funden heraussticht. Dies ist damit zu erklären, dass dieser Filter sehr allgemein gehalten ist und lediglich nach Dateien mit den Endungen *.gpg* und *.pgp* sucht. Dieser Filter könnte auch in die Kategorie *Warn* verschoben werden, da nicht jeder dieser Funde auch einen privaten Schlüssel repräsentieren muss. Jedoch kann es auch ein privater Schlüssel sein, weswegen dieser Filter hier in der Kategorie *Critical* eingeordnet wurde.

Auch der Filter *Private Key* sticht mit 106 Funden heraus. Jedoch handelt es sich hierbei um einen Filter des Dateiinhalts-Scanners, der extra noch einmal eine Datei nach einem privaten Schlüssel untersucht. Deshalb kann hierbei davon ausgegangen werden, dass es sich tatsächlich um private Schlüssel handelt. Jedoch kann nicht gesagt werden, ob dies wirklich produktive Schlüssel sind oder Schlüssel, die zum Testen erzeugt wurden.

An dritter Stelle folgt mit 68 Funden der Filter *History of shell commands*. Dieser sucht nach der Datei *.bash_history* in der die eingegebenen Befehle gespeichert werden. In dieser Datei könnten auch Passwörter stehen, sofern die in einem Befehl mit eingegeben wurden.

Mit 61 Funden folgt der Filter *Authorized keys for ssh-connection*. Dieser Filter überprüft, ob die Datei *.authorized_keys* existiert und Text beinhaltet. Ist dies der Fall, können sich später bei einer Instanz alle Personen verbinden, die einen passenden Schlüssel besitzen. Dies wird in der Regel vor allem derjenige sein, der dieses Image erstellt hat.

Zuletzt mit 15 Funden ist noch der Filter *Private DSA-/RSA-keys* vertreten. Dieser sucht nach privaten Schlüsseln, welche in Dateien namens *id_rsa* oder *id_dsa* stehen.

Neben den Kritischen Filtern existieren noch Filter, welche als Warnung dienen. Diese sind oft nicht so aussagekräftig, wie die kritischen Meldungen, können jedoch in manchen Fällen interessant sein:

Tabelle 7.10.: Funde der Kategorie *Warn*

Filter	Funde	Anteil in [%]
SVN-repository	23886	72,62
Interesting files	8359	25,41
Certificates	293	0,89
Public keys	191	0,58
Git-repository	162	0,49
Gesamt	32891	100

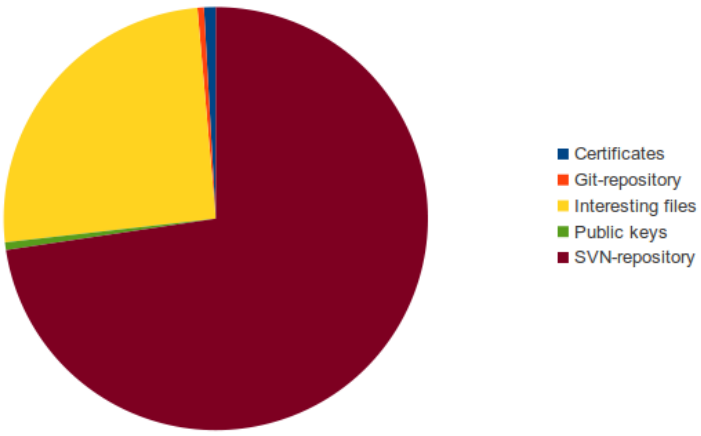

Abbildung 7.10.: Funde der Kategorie *Warn*

Hier sticht sofort der Filter *SVN-repository* mit 23886 Funden heraus. Jedoch sei hierzu angemerkt, dass dieser Filter viele Dateien eines SVN-Repositories als einzelnen Fund berücksichtigt. Dadurch kann ein einzelnes großes Repository bereits sehr viele Meldungen generieren. Hier kommen die meisten Meldungen von einem einzelnen Image mit der ID *ami-9d428cf4* und dem Namen *BUILD_LINUX_MASTER*. Alleine dieses Image erzeugt 23502 der 23886 Meldungen dieses Filters.

An zweiter Stelle folgt der Filter *Interesting Files* mit 8359 Funden. Dieser Filter ist sehr allgemein gehalten, weswegen es nicht verwundert, dass dieser Filter viele Meldungen erzeugt. Er sucht lediglich nach Dateinamen, welche die Zeichenketten *secret*, *key* oder *private* beinhalten. Deswegen werden viele dieser Meldungen wohl Falschmeldungen sein.

Mit 293 Funden an dritter Stelle folgt der Filter *Certificates*. Dieser sucht nach Zertifikaten, indem er nach Dateien mit der Endung *.crt* scannt. Diese enthalten in der Regel zwar keine privaten Schlüssel, können jedoch andere private Daten, wie Namen oder Anschrift, enthalten.

Der Filter *Public Keys* folgt mit 191 Funden. Dieser sucht nach Dateien mit der Endung *.pub*. Diese enthalten in der Regel öffentliche Schlüssel, welche ähnlich wie Zertifikate zu behandeln sind.

An letzter Stelle ist der Filter *Git-Repository* mit 162 Funden vertreten. Dieser ist ähnlich des Filters *SVN-Repository* aufgebaut und kann daher auch viele Meldungen erzeugen.

Funde nach Distribution

Nachdem die Funde allgemein betrachtet wurden, ist vor allem auch interessant, bei welchen Distributionen welche Daten gefunden wurden. Dadurch können eventuell Rückschlüsse auf die Nutzer der einzelnen Distributionen und deren Sicherheitsverständnis gezogen werden.

Vor der Auswertung der Funde muss zuerst die Verteilung der Distributionen bei den gescannten Images analysiert werden:

Tabelle 7.11.: Verteilung der Distributionen bei Evaluation

Distribution	Anzahl
Unbekannt	47
Centos	3
Fedora	2
Gentoo	1
Redhat	4
Ubuntu	43

Wie zu erwarten, sind die meisten Images unbekannte und Ubuntu-Images. Vier Redhat-, drei Centos-, zwei Fedora- und ein Gentoo-Image sind auch noch dabei.

Nachdem die Verteilung der Distributionen festgestellt wurde, können nun die Funde ausgewertet werden. Begonnen wird mit der Funden der Kategorie *Critical*.

Hierbei wird zuerst eine Auswertung der Gesamtfunde der einzelnen Filter, nach Distributionen sortiert, erstellt:

Tabelle 7.12.: Funde der Kategorie *Critical* nach Distribution

Filter	Unbekannt	Centos	Fedora	Gentoo	Redhat	Ubuntu
Athorized keys for ssh-connection	57	0	2	0	0	2
GPG-/PGP-files	179	0	0	0	0	294
History of shell-commands	58	1	3	2	0	4
Private DSA-/RSA-keys	15	0	0	0	0	0
Private key	91	0	0	0	4	11
Gesamt	400	1	5	2	4	311

Im Anschluss wird eine Auswertung über die durchschnittlichen Funde pro Image der einzelnen Filter, nach Distributionen sortiert, erstellt:

Tabelle 7.13.: Funde der Kategorie *Critical* nach Distribution pro Image

Filter	Unbekannt	Centos	Fedora	Gentoo	Redhat	Ubuntu
Athorized keys for ssh-connection	1,21	0,00	1,00	0,00	0,00	0,05
GPG-/PGP-files	3,81	0,00	0,00	0,00	0,00	6,84
History of shell-commands	1,23	0,33	1,50	2,00	0,00	0,09
Private DSA-/RSA-keys	0,32	0,00	0,00	0,00	0,00	0,00
Private key	1,94	0,00	0,00	0,00	1,00	0,26
Durchschnitt	1,70	0,07	0,50	0,40	0,20	1,45

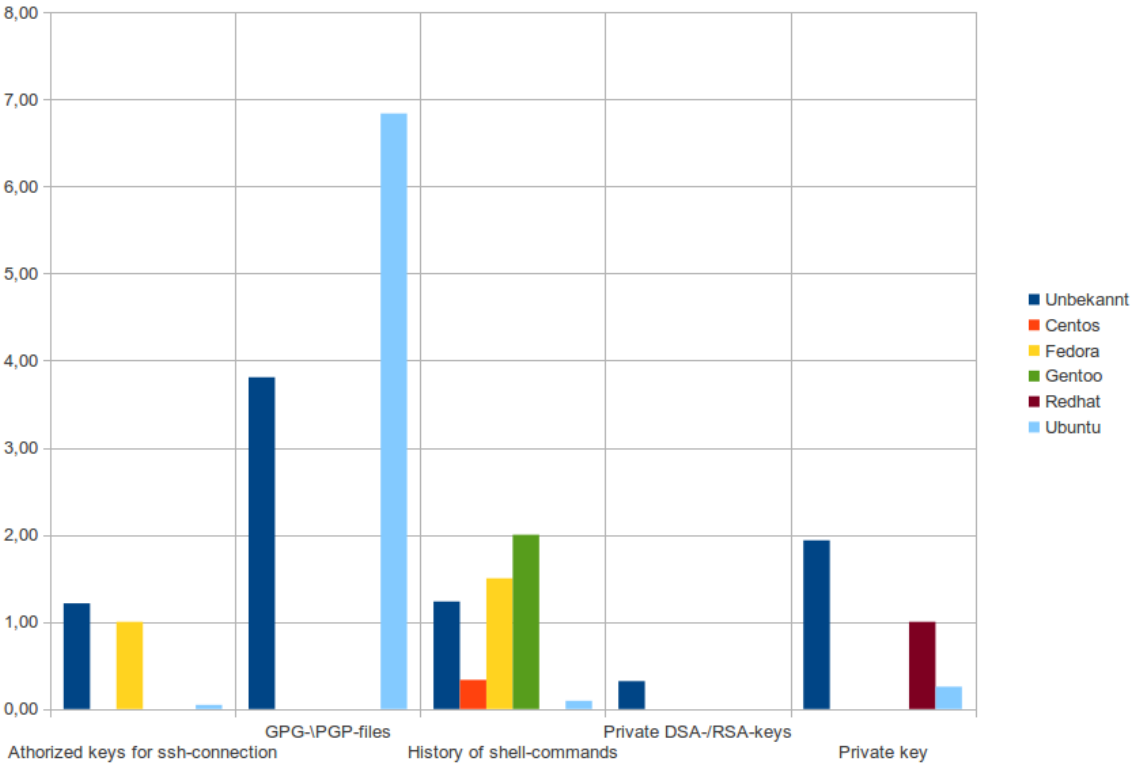

Abbildung 7.11.: Funde der Kategorie *Critical* nach Distribution

Hier sind jetzt primär die Funde pro Image von Interesse, da diese die Verteilung der Distributionen berücksichtigen. Es stechen die Images mit unbekannter Distribution und die Ubuntu-Images heraus. In beiden Kategorien wurden Durchschnittlich 1,7 bzw. 1,45 Funde pro Image registriert. Bei den restlichen Distributionen sind es maximal 0,5 Funde pro Image.

Eine eigene Interpretation für den hohen Wert bei den Images mit unbekannter Distribution ist, dass diese Images viele Images von Laien beinhalten. Diese haben Images von bekannten Distributionen abgewandelt und anschließend unter einem Namen gespeichert, welcher den Namen der Distribution nicht mehr beinhaltet. Viele Images von professionellen Entwicklern beinhalten im Namen noch die Distribution und werden daher korrekt detektiert.

Der hohe Wert bei Ubuntu-Images ließe sich auf ähnliche Art und Weise erklären. Ubuntu hat eine sehr große Community. So existieren viele Tools und Anleitungen zum Bau eigener Images auf Basis von Ubuntu. Dadurch werden auch hier wieder Laien angelockt.

Weitere Erkenntnisse zeigen sich, wenn man die einzelnen Filter betrachtet. So finden sich bei allen Distributionen, außer Redhat, Images, welche noch die Bash-Historie beinhalten. Dies zeigt, dass immer mehr Entwickler zwar darauf achten, private Daten wieder zu löschen, jedoch nach wie vor die Bash-Historie vergessen, welche eine versteckte Datei ist.

Nun können die Funde der Kategorie *Warn* ausgewertet werden.

Zuerst werden wieder die Gesamtzahl der Funde, auf Distributionen verteilt, analysiert:

Tabelle 7.14.: Funde der Kategorie *Warn* nach Distribution

Filter	Unbekannt	Centos	Fedora	Gentoo	Redhat	Ubuntu
Certificates	138	3	3	1	37	111
Git-repository	162	0	0	0	0	0
Interesting files	4467	114	66	13	211	3488
Public keys	117	11	6	3	6	48
SVN-repository	23886	0	0	0	0	0
Gesamt	28770	128	75	17	254	3647

Im Anschluss folgt wieder die Analyse der Funde pro Image:

Tabelle 7.15.: Funde der Kategorie *Warn* nach Distribution pro Image

Filter	Unbekannt	Centos	Fedora	Gentoo	Redhat	Ubuntu
Certificates	2,94	1,00	1,50	1,00	9,25	2,58
Git-repository	3,45	0,00	0,00	0,00	0,00	0,00
Interesting files	95,04	38,00	33,00	13,00	52,75	81,12
Public keys	2,49	3,67	3,00	3,00	1,50	1,12
SVN-repository	508,21	0,00	0,00	0,00	0,00	0,00
Durchschnitt	122,43	8,53	7,50	3,40	12,70	16,96

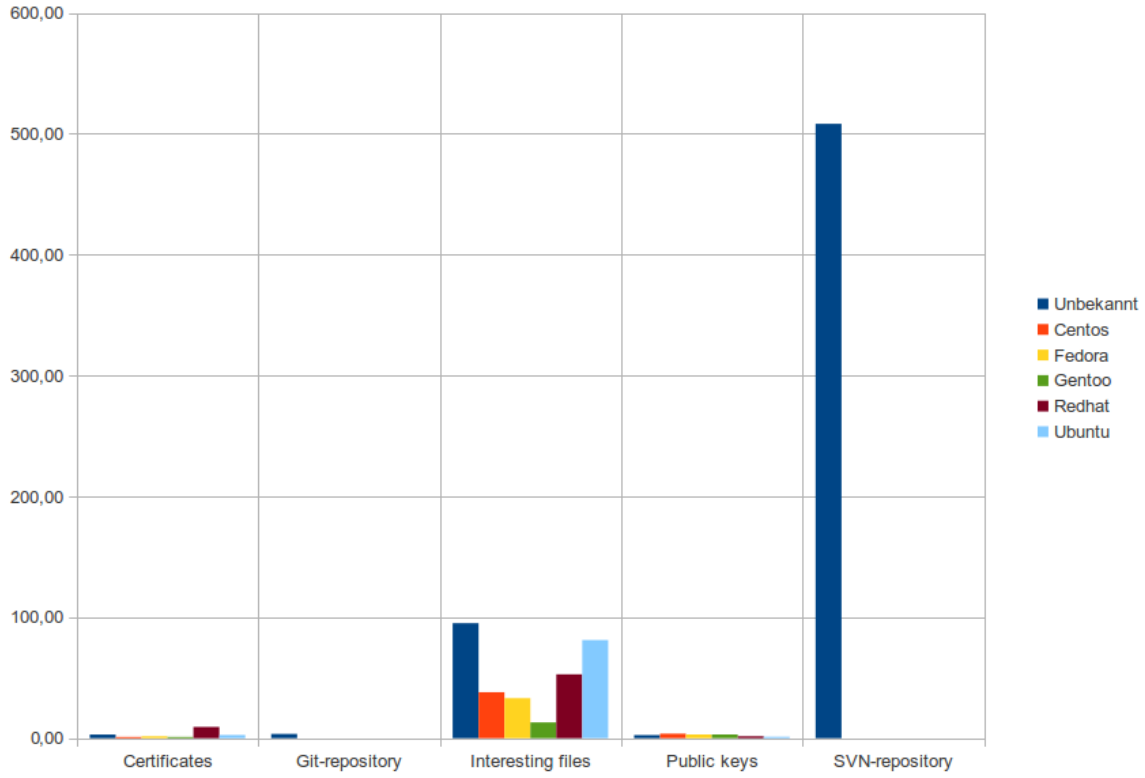

Abbildung 7.12.: Funde der Kategorie *Warn* nach Distribution

Hier sticht wieder der Filter *SVN-Repository* heraus. Dies kommt, wie bereits vorher angesprochen, von dem Image *BUILD_LINUX_MASTER*. Bei diesem Image kann, auf Grund des Namens, keine Distribution detektiert werden. Der Name lässt außerdem darauf schließen, dass auch dieses Image von einem Laien entwickelt wurde.

Von diesem Ausreißer abgesehen, zeigt sich auch hier wieder ein ähnliches Bild, wie bei den Funden der Kategorie *Critical*. Vor allem bei den Images mit unbekannter Distribution und bei Ubuntu-Images wurden viele auffällige Dateien gefunden.

Daher gelten auch hier die gleichen Vermutungen. Viele Images mit unbekannter Distribution und Ubuntu-Images werden von Laien erstellt und generieren daher mehr Funde, als professionell erstellte Images.

Funde nach Architektur

Als nächstes werden die Funde nach Architektur unterschieden. Hierbei wird jedoch nicht auf die einzelnen Filter eingegangen, sondern lediglich nach den Kategorien *Critical* und *Warn* unterschieden. Amazon bietet als Architekturen 32 Bit (*i386*) und 64 Bit (*x86_64*) an.

Tabelle 7.16.: Funde nach Architektur

Architektur	Anzahl	Funde - Critical	Funde - Warn	Funde - Critical / Architektur	Funde - Warn / Architektur
i386	54	349	5418	6,46	100,33
x86_64	46	374	27473	8,13	597,24

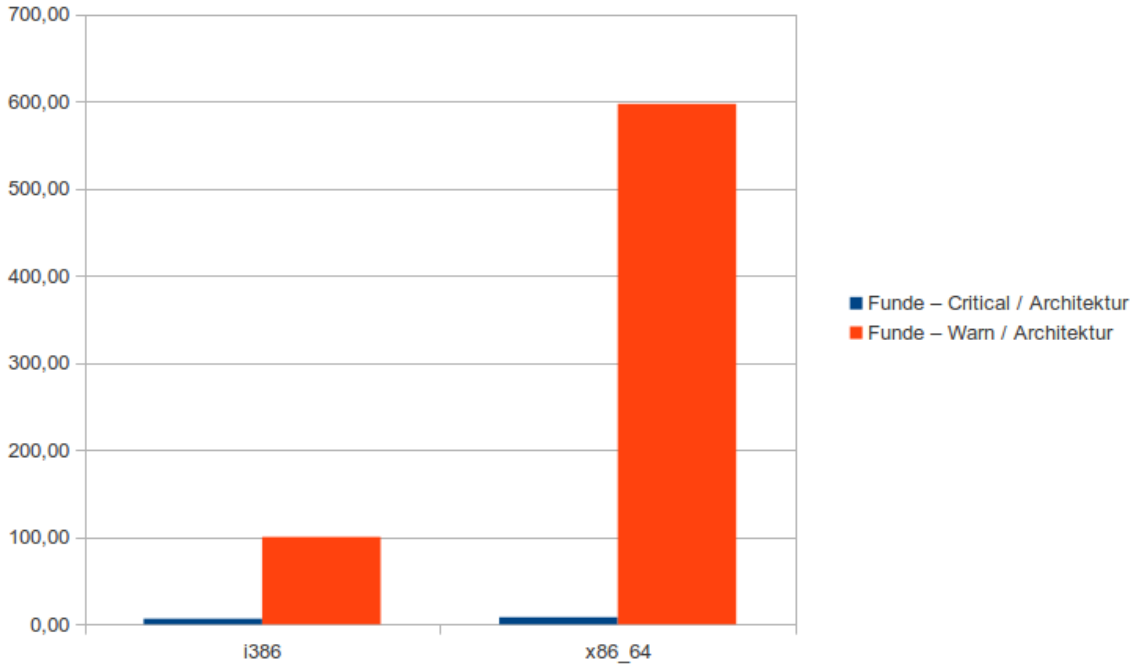

Abbildung 7.13.: Funde nach Architektur

Leider liefert die Unterscheidung nach Architektur keine neuen Erkenntnisse. Der Ausreißer bei den Warnungen ist erneut auf das Image *BUILD_LINUX_MASTER* zurückzuführen. Ansonsten existieren keine größeren Unterschiede zwischen 32 Bit und 64 Bit Systemen.

Zeitmessung

Zuletzt sollen noch die Zeiten der Scans analysiert werden. Hierzu wurde der Ablauf eines Scans in mehrere Schritte unterteilt:

- Instanz starten - Die Instanz mit dem Image wird gestartet und darauf gewartet, dass diese bereit ist.
- Instanz stoppen - Die Instanz wird wieder gestoppt und darauf gewartet, dass diese komplett beendet wurde.
- Laufwerk entfernen - Das Laufwerk wird von der ursprünglichen Instanz entfernt.
- Laufwerk anbinden - Das Laufwerk wird an die Scanner-Instanz angebunden.
- Laufwerk mounten - Das Laufwerk wird in das Dateisystem des Scanners eingebunden.
- Scannen - Der eigentliche Scan.
- Laufwerk entfernen - Das Laufwerk wird wieder von der Scanner-Instanz entfernt.
- Resultate speichern - Die Resultate werden in der SQLite-Datenbank gespeichert.

Die Rohdaten der Zetmessung mit den Daten jedes einzelnen Images findet sich in Anhang A.

Als erstes werden die Gesamtzeiten aller Images betrachtet, um eventuelle Ausreißer zu erkennen:

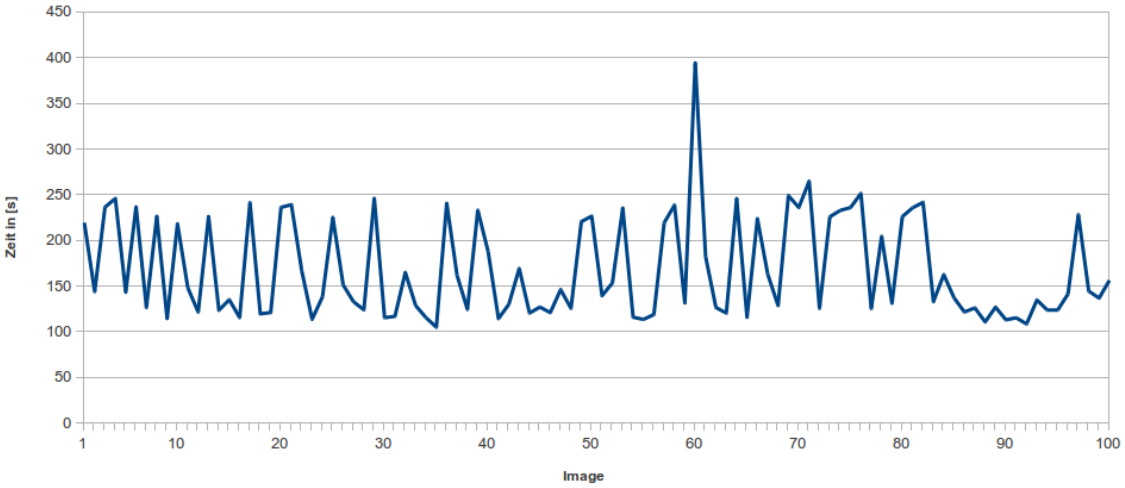

Abbildung 7.14.: Zeitmessung über alle Images

Bis auf einen Ausreißer liegen die Zeiten aller Images im Bereich von 100 bis etwa 250 Sekunden. Lediglich ein Image benötigte fast 400 Sekunden. Hierbei handelt es sich erneut um das bereits bekannte Image *BUILD_LINUX_MASTER*, bei dem der Scan, aufgrund vieler Dateien, entsprechend länger gedauert hat.

Nun werden die einzelnen Schritte analysiert. Hierbei wird zu jedem Schritt das Minimum, der Median und das Maximum der Zeit angegeben, die bis dahin benötigt wurde. Die Messung findet über alle gescannten Images statt.

Tabelle 7.17.: Zeitmessung - Gesamtzeit

Schritt	Min	Median	Max
Instanz Starten	20,623	30,817	71,018
Instanz stoppen	41,066	71,456	193,089
Laufwerk entfernen	41,293	71,725	198,301
Laufwerk anbinden	46,954	77,397	204,052
Laufwerk mounten	77,027	107,503	234,147
Scannen	90,784	128,099	377,344
Laufwerk entfernen	106,578	143,829	393,325
Resultate speichern	106,688	143,951	395,532

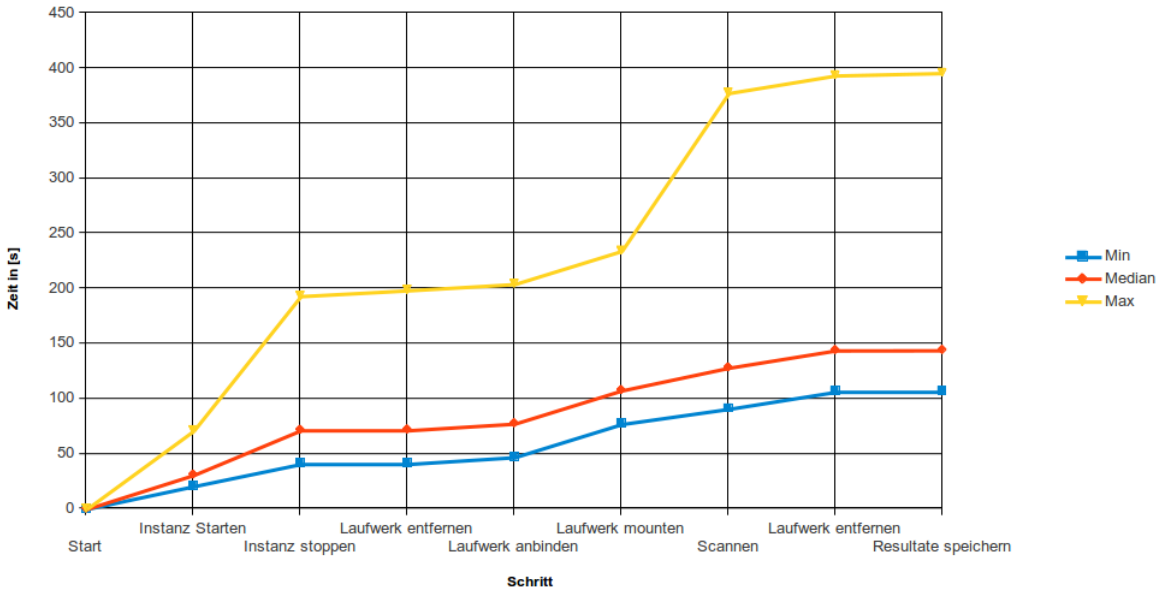

Abbildung 7.15.: Gesamtzeit nach Schritt

Anhand dieser Messung lässt sich bereits herausfinden, welche Schritte, im Vergleich zu den Restlichen, länger benötigen und daher noch weiter optimiert werden sollten. Diese sind vor allem die Schritte Instanz starten, Instanz stoppen, Laufwerk mounten, Scannen und das zweite Laufwerk entfernen.

Nachdem die Gesamtzeiten betrachtet wurden, können nun noch die Zeiten, die pro Schritt benötigt wurden, verglichen werden. Hierbei werden, neben dem Minimum, dem Maximum und dem Median, auch das erste Quantil, das dritte Quantil, der Mittelwert, die Varianz und die Standardabweichung betrachtet. Darüber hinaus wird auch ein Box Plot zur Veranschaulichung erstellt.

Tabelle 7.18.: Zeitmessung pro Schritt

Statistik	Instanz Starten	Instanz stoppen	Laufwerk entfernen	Laufwerk anbinden	Laufwerk mounten	Scannen	Laufwerk entfernen	Resultate speichern
Minimum	20,623	20,355	0,185	1,264	30,073	0,073	14,795	0,095
Maximum	71,018	162,081	5,213	6,646	65,080	269,522	46,131	2,207
1. Quartil	30,720	30,531	0,209	5,632	30,089	2,762	15,762	0,110
Median	30,817	40,598	0,223	5,658	30,107	5,998	15,801	0,118
3. Quartil	30,972	151,350	0,254	5,720	30,220	14,996	15,905	0,129
Mittelwert	30,383	71,680	0,315	5,619	30,546	14,817	16,369	0,148
Varianz (n-1)	42,581	2838,908	0,260	0,405	12,256	930,367	10,007	0,045
Standardabweichung (n-1)	6,525	53,281	0,510	0,637	3,501	30,502	3,163	0,211

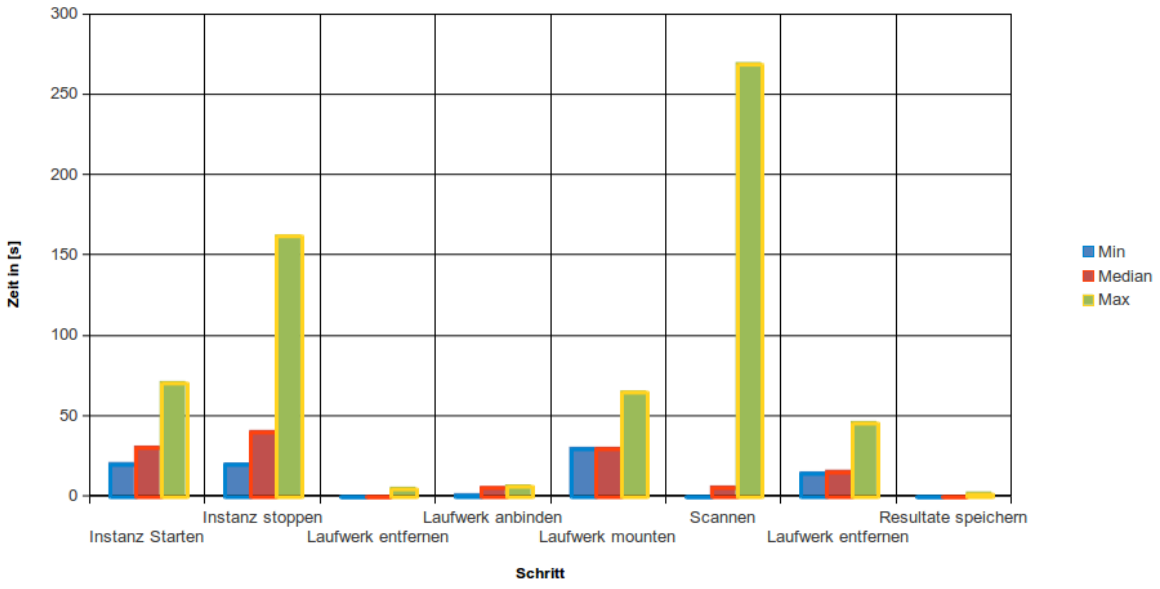

Abbildung 7.16.: Zeit pro Schritt

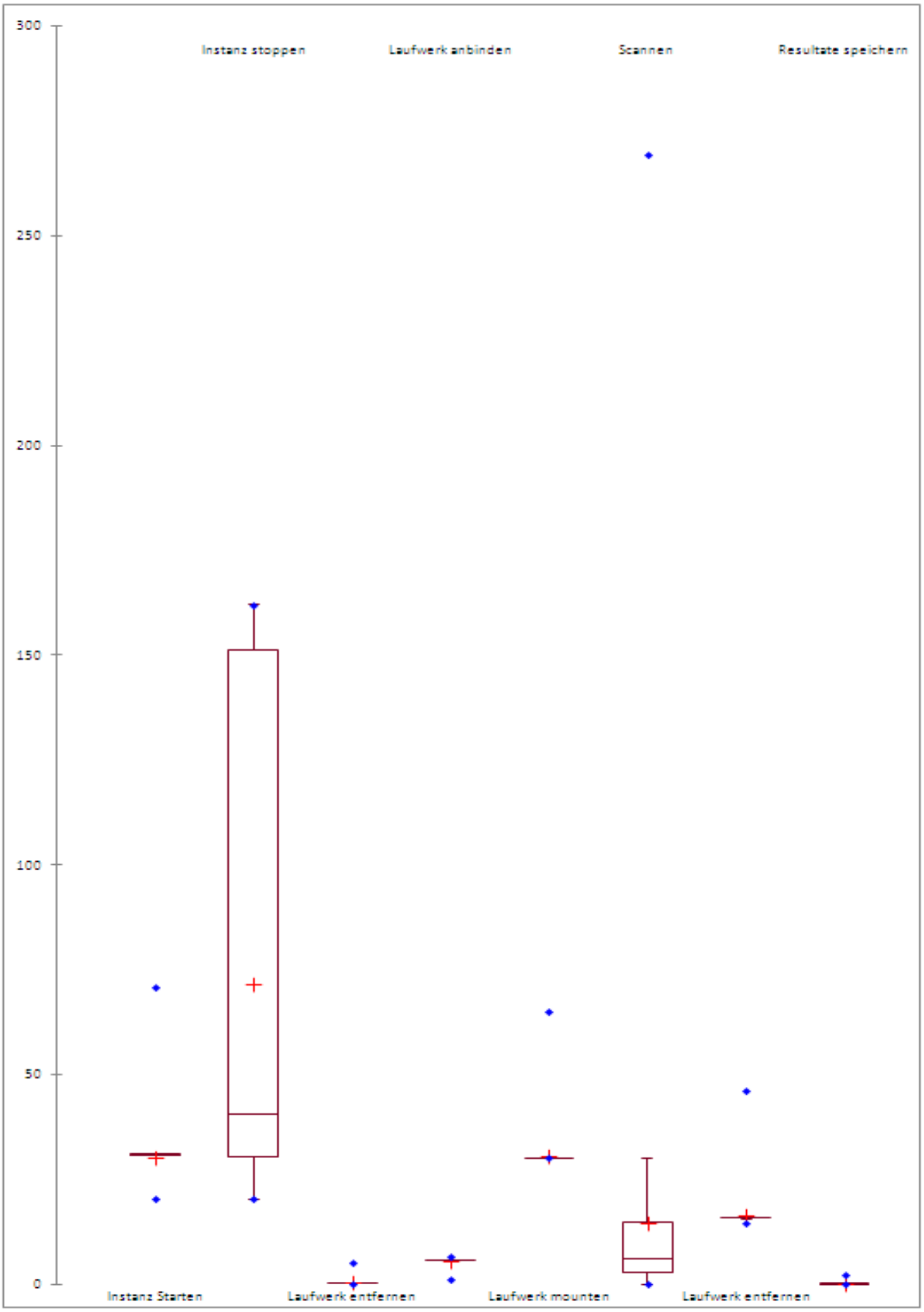

Abbildung 7.17.: Zeitmessung pro Schritt - Box Plot

Auch hier werden noch einmal die Ergebnisse der vorangegangenen Gesamtzeitmessung bestätigt. Die Schritte Laufwerk anbinden, Resultate speichern und das erste Laufwerk entfernen sind sehr performant und zeigen nur geringe Varianzen und Abweichungen. Die Schritte Instanz starten, Instanz stoppen, Laufwerk mounten, Scannen und das zweite Laufwerk entfernen hingegen zeigen eine größere Varianz und eine größere Abweichung. Hier ist also noch Verbesserungsbedarf.

Beim Box Plot zeigte sich hier ein Skalierungsproblem. So werden hier einige Schritte nur als Linien, statt als Boxen angezeigt. Dies liegt daran, dass alle Werte dieser Schritte sehr nah beieinander liegen, während die Werte bei anderen Schritten weit auseinander liegen. Dies lässt sich auch in der Tabelle ablesen. Dadurch ist es nicht möglich, den Plot zufriedenstellend zu skalieren.

7.4. Zusammenfassung und Fazit

Es wurde festgestellt, dass die Amazon Web Services eine Vielzahl unterschiedlichster Images bereitstellt. Darunter viele verschiedene Distributionen, Kernel und Ramdisks. Auch stehen verschiedene Speicherverfahren, Virtualisierungstypen, Hypervisors und Ähnliches zur Verfügung. Lediglich die Angabe über die genutzte Distribution ist ein großer Mangel.

Die Tests der 100 zufällig ausgewählten Images haben gezeigt, dass nach wie vor viele Images private Daten enthalten. Vor allem die Datei *.bash_history* wird oft noch vergessen. Aber auch andere Daten werden nach wie vor vergessen. Daher ist zu raten, sich noch stärker an die Richtlinien, die bereitgestellt werden zu halten [AWS12g]. Diese Richtlinien gehen sogar noch einmal explizit auf die Bash-Historie ein und empfiehlt diese zu löschen, da diese sogar den Schlüssel für den Zugriff auf die Amazon Web Services enthalten kann.

Die Tests haben aber auch gezeigt, dass der Scanner noch weiter verbessert werden kann. So sind, gerade in der Kategorie *Warn*, noch viele Falschmeldungen in den Ergebnissen vertreten. Darüber hinaus haben die Zeitmessungen gezeigt, dass auch die Laufzeit des Scanners noch weiter verbessert werden kann.

8. Zusammenfassung und Ausblick

Die Arbeit hat gezeigt, dass es immer noch Probleme im Zusammenhang mit der Verteilung von Betriebssystem-Images in Cloud-Infrastrukturdiensten gibt.

Um dieses Problem anzugehen, wurde ein Framework entwickelt, mit dem es möglich ist, solche Betriebssystem-Images nach Schwachstellen und privaten Daten zu scannen. Hierzu wurde auch ein Modul-System entwickelt, um das Framework später leicht erweitern zu können. Außerdem wurden SQLite-Datenbanken genutzt, um das Framework noch flexibler und leichter erweiterbar zu machen.

Im Anschluss wurde das Framework anhand von 100 zufällig ausgewählten Images getestet. Dieser Test zeigte nicht nur, dass das Framework funktioniert, sondern zeigte auch, dass nach wie vor viele Images private Daten enthalten. Vor allem die Datei *.bash_history* wurde oft in solchen Images vergessen. Außerdem zeigte sich, dass das Framework, in Bezug auf die Laufzeit und die Ergebnisse, noch verbessert werden kann.

8.1. Ausblick

Auch wenn das Framework bereits funktionsfähig ist, so kann doch noch einiges verbessert werden. Diverse Module wurden noch nicht implementiert und die Filterregeln können noch verbessert und erweitert werden, um bessere Ergebnisse zu erhalten. Außerdem können noch weitere komplett neue Module konzipiert und umgesetzt werden.

8.1.1. Implementation weiterer Module

Wenn auch bereits die wichtigsten Module umgesetzt wurden, so fehlen noch weitere Module, welche zwar bereits konzipiert, aber noch nicht umgesetzt wurden.

So fehlt die Implementation des Zugriffs-Moduls für externe Images noch komplett. Diese könnte mit Hilfe der Bibliothek Libguestfs [Lib12] geschehen.

Darüber hinaus fehlt auch noch die Umsetzung des Rootkit-Scanners. Dies könnte mit Hilfe eines bereits fertigen Rootkit-Scanners, wie Chkrootkit [Chk12] oder RKHunter [RKH12] geschehen. Jedoch müssten diese Tools vorher noch evaluiert werden. Sollte sich einer der Rootkit-Scanner als brauchbar erweisen, kann dieser durch das Modul aufgerufen werden und die Ausgaben von dem Modul entsprechend ausgewertet und passende Resultate ausgegeben werden.

Auch die endgültige Umsetzung des Exploit-Scanners fehlt noch. Es wurden bisher lediglich Tools entwickelt, um die Daten aus der National Vulnerability Database [NVD12] in eine SQLite-Datenbank zu konvertieren, damit diese später von dem Modul leichter verarbeitet werden können. Das Modul könnte später dann die Versionsnummern der Software, anhand des Paketmanagers, mit den Versionsnummern in der Datenbank vergleichen und bei Übereinstimmungen Alarm schlagen. Auch wäre es denkbar, die Versionsnummer direkt über die Ausgabe der einzelnen Programme zu erfahren, beispielsweise durch den Parameter *–version*, den die meiste Software unter Linux versteht. Dies hätte den Vorteil, dass auch dann die Versionsnummern erkannt werden können, wenn das System keinen Paketmanager besitzt oder die Software ohne Zuhilfenahme des Paketmanagers installiert wurde.

Außerdem fehlt noch der Registry-Scanner. Dieser ist für den Scan von Windows-Images sehr wichtig. Erst durch solch einen Scanner, zusammen mit passenden Regeln für den Dateinamens-Scanner und den Dateiinhalts-Scanner, können Windows-Images zufriedenstellend gescannt werden. Für den Zugriff auf die Registry könnte die hivex-Bibliothek [hiv12] des libguestfs-Projektes genutzt werden.

Zuletzt fehlt noch das Frontend als parametrisierter Befehl. Erst dieses Frontend ermöglicht es, den Scanner auch aus Skripten heraus zu nutzen. Einige Überlegungen zu den einzelnen Kommandozeilen-Parametern wurden im Rahmen der Konzeption bereits gemacht.

8.1.2. Filterregeln

Auch die Filterregeln sind noch nicht ausreichend. Bis jetzt sind lediglich ein paar geläufige Linux-Filter eingebaut. Durch die Nutzung einer Datenbank ist der Einbau neuer Filter auch möglich, ohne den Quelltext verändern zu müssen.

Darüber hinaus Fehlen auch noch Filter für Windows-Images. Diese können momentan noch gar nicht nach Schwachstellen durchsucht werden.

Auch ist die Erkennungsrate im Allgemeinen noch nicht zufriedenstellend. Insbesondere bei den Filtern der Kategorie *Warn* sind nach einem Scan noch viele falsche Meldungen vorhanden. Dies könnte beispielsweise durch ein Whitelist-System verbessert werden. Da könnten dann bekannte Dateien in den Datenbanken eingetragen werden, welche normalerweise eine Falschmeldung erzeugen würden. Diese Dateien würden dann bei der Auswertung ignoriert werden.

8.1.3. Neue Module

Zuletzt ist es auch denkbar, noch komplett neue Module zu implementieren. Dies könnten Module sein, welche mittels eines neuronalen Netz oder Heuristiken versuchen nach auffälligen Dateien zu suchen oder auch Module, welche ein System im laufenden Betrieb nach Schwachstellen, wie offene Ports oder überflüssige Prozesse, scannen.

A. Rohdaten der Zeitmessung

Tabelle A.1.: Rohdaten der Zeitmessung

ID	Instanz Starten	Instanz stoppen	Laufwerk entfernen	Laufwerk anbinden	Laufwerk mounten	Scannen	Laufwerk entfernen	Resultate speichern
1	20,692	162,056	162,259	168,736	198,994	204,979	220,764	220,926
2	30,959	91,699	91,925	97,537	127,685	129,741	145,533	145,658
3	30,750	182,501	182,728	188,387	219,274	222,019	237,759	237,922
4	30,727	182,165	182,365	187,974	218,284	231,391	247,204	247,316
5	30,854	71,993	72,200	77,864	107,950	129,173	144,869	144,995
6	20,712	172,102	172,296	177,889	208,336	216,945	237,950	238,054
7	20,841	51,374	51,881	57,524	87,777	111,713	128,104	128,317
8	21,147	173,761	173,974	179,856	210,025	211,935	227,707	227,831
9	30,744	61,275	61,464	67,128	97,219	100,286	116,071	116,188
10	30,774	152,194	152,589	158,328	188,411	203,346	219,493	219,600
11	30,840	61,316	61,558	67,290	97,383	133,481	149,399	149,739
12	32,520	62,980	63,198	68,898	98,989	107,347	123,124	123,235
13	20,749	172,170	172,391	177,992	208,281	211,211	227,541	227,636
14	30,706	71,242	71,461	77,200	107,288	109,266	125,097	125,213
15	31,095	61,540	61,894	67,676	97,759	120,085	136,593	136,773
16	30,840	61,441	61,649	67,892	97,983	101,251	117,082	117,200
17	31,611	183,029	183,323	188,992	219,520	221,775	242,661	242,771
18	31,047	61,487	61,690	67,353	97,443	105,361	121,178	121,305
19	33,047	63,586	63,951	69,686	99,845	102,622	122,560	122,656
20	30,724	182,170	182,376	188,036	218,223	221,213	237,549	237,656
21	31,437	182,974	183,180	188,788	218,884	224,694	240,499	240,609
22	30,772	61,224	61,409	67,033	97,180	152,137	168,096	168,208
23	30,865	61,365	61,903	67,576	97,666	99,499	115,194	115,324
24	40,782	81,933	82,152	87,783	117,869	123,881	139,620	139,730
25	21,286	172,812	173,032	178,649	208,898	210,811	226,562	226,665
26	30,802	61,308	61,725	67,335	97,415	135,884	152,781	152,880
27	30,848	71,416	71,623	77,274	107,396	118,625	134,395	134,495
28	30,715	71,335	71,564	77,274	107,365	109,861	125,590	125,718

29	30,894	182,571	183,128	188,785	219,006	231,623	247,386	247,502
30	30,896	61,388	61,629	67,273	97,362	101,271	117,012	117,112
31	30,819	61,315	61,534	67,171	97,250	102,284	118,400	118,521
32	30,870	61,771	61,995	67,758	132,838	150,562	166,389	166,509
33	30,728	71,327	71,567	77,193	107,285	114,369	130,135	130,254
34	30,688	61,175	61,403	67,026	97,144	101,225	117,008	117,110
35	20,764	51,268	51,469	58,115	88,207	90,784	106,578	106,688
36	30,798	182,218	182,439	188,089	218,385	226,128	241,926	242,038
37	30,960	71,527	71,758	77,391	107,479	146,711	163,138	163,258
38	30,839	72,554	72,761	78,414	108,589	110,290	126,100	126,205
39	30,843	182,292	182,512	188,164	218,441	218,515	234,363	234,537
40	30,828	71,536	71,775	77,423	107,502	173,209	189,059	189,187
41	20,754	61,271	61,488	67,242	97,412	100,181	115,967	116,083
42	30,775	71,330	71,608	77,238	107,331	115,749	131,635	131,754
43	31,282	71,867	72,502	78,148	108,234	154,861	170,750	170,866
44	30,812	61,280	61,541	67,225	97,314	106,206	122,024	122,133
45	30,893	71,498	71,771	77,403	107,504	112,408	128,584	128,705
46	30,721	61,244	61,465	67,087	97,171	106,605	122,507	122,618
47	30,699	71,247	71,455	77,105	107,251	131,865	147,636	147,747
48	30,775	71,322	71,524	77,429	107,513	111,453	127,172	127,329
49	30,693	61,146	61,351	62,747	92,881	206,014	221,892	222,126
50	20,786	172,158	172,370	178,030	208,837	212,266	227,939	228,064
51	30,757	71,370	71,578	77,219	107,304	125,160	140,945	141,075
52	42,240	92,946	93,202	98,868	129,060	139,168	154,855	154,976
53	30,794	182,359	182,616	188,214	218,468	221,026	236,743	236,850
54	30,910	61,449	61,700	67,337	97,429	101,657	117,477	117,595
55	31,314	61,815	62,031	67,756	97,942	99,288	115,048	115,183
56	31,158	61,689	61,935	67,592	97,680	104,508	120,350	120,483
57	21,029	166,429	166,664	172,402	202,721	204,970	220,761	220,870
58	22,443	174,289	174,555	180,395	210,796	224,291	240,052	240,166
59	41,035	72,010	72,282	77,958	108,217	117,279	133,055	133,254
60	31,071	71,839	72,059	77,736	107,822	377,344	393,325	395,532
61	51,067	101,778	102,070	107,802	137,895	167,974	183,833	183,951
62	30,864	61,585	61,807	67,379	97,469	112,247	128,074	128,191
63	30,754	61,244	61,462	67,123	97,317	106,201	121,965	122,102
64	31,017	182,502	182,753	188,513	218,610	231,289	247,050	247,189
65	31,432	62,156	62,355	67,889	98,075	101,620	117,424	117,578
66	20,623	171,966	172,162	177,804	208,067	209,477	225,177	225,279
67	30,677	62,427	62,629	68,286	98,453	148,421	164,436	164,580
68	40,973	71,441	71,669	77,326	107,413	114,401	130,196	130,347

69	30,816	182,347	183,355	188,967	219,245	234,425	250,395	250,514
70	31,650	183,676	183,916	189,647	219,921	221,712	237,411	237,514
71	30,671	182,159	182,353	188,024	218,237	220,039	266,171	266,275
72	31,767	62,259	62,855	68,918	99,008	111,380	127,174	127,294
73	31,706	173,941	174,153	179,813	210,033	211,529	227,266	227,367
74	21,402	172,826	173,035	178,666	208,885	218,407	234,220	234,363
75	30,868	183,165	183,442	189,103	219,284	220,603	237,359	237,463
76	31,008	193,089	198,301	204,052	234,147	237,033	252,790	252,911
77	31,513	72,096	72,505	78,485	108,574	110,301	126,998	127,116
78	71,018	152,309	152,561	158,496	188,677	191,051	205,845	205,946
79	20,894	51,366	51,589	57,244	90,065	116,588	132,703	132,809
80	30,790	172,169	172,386	178,104	208,468	211,623	227,456	227,567
81	30,775	182,253	182,462	188,100	218,372	221,272	237,135	237,243
82	30,707	182,111	182,347	187,993	218,327	227,239	243,000	243,111
83	30,739	61,362	61,564	67,202	97,367	118,845	134,610	134,714
84	30,754	81,778	82,004	87,959	118,041	147,278	164,002	164,131
85	20,761	51,159	51,412	56,956	87,119	122,741	138,504	138,627
86	30,818	61,395	61,591	67,274	97,386	102,277	123,121	123,242
87	30,806	71,437	71,649	77,339	107,413	111,823	127,683	127,797
88	20,710	41,066	41,293	46,954	77,027	96,414	112,281	112,404
89	30,750	71,392	72,062	77,943	108,022	112,773	128,479	128,597
90	30,815	61,347	61,587	67,220	97,307	98,621	114,601	114,738
91	31,553	62,054	62,365	68,013	98,102	101,098	116,823	116,960
92	30,849	61,363	61,578	62,842	93,002	94,340	110,104	110,224
93	30,760	81,486	81,709	87,384	117,474	120,575	136,273	136,388
94	30,879	71,471	71,692	77,318	107,404	110,144	125,869	125,981
95	30,842	71,439	71,636	77,209	107,300	109,321	125,028	125,141
96	30,824	71,417	71,637	77,300	107,395	127,025	142,788	142,906
97	20,828	172,451	172,681	178,395	208,649	213,226	229,317	229,626
98	30,872	71,690	71,932	77,557	107,637	130,165	146,038	146,173
99	30,922	61,416	61,627	67,374	97,464	122,504	138,279	138,401
100	41,027	91,684	92,156	97,780	127,878	142,152	157,829	157,955

Literaturverzeichnis

[Ale12] *You Should Use EBS Boot Instances on Amazon EC2.* `http://alestic.com/2012/01/ec2-ebs-boot-recommended`, Januar 2012. [Online; Stand 30. Juli 2012].

[Ama12] Amazon: *Amazon Web Services.* `http://aws.amazon.com/`, 2012. [Online; Stand 4. Februar 2012].

[AMD11] *AMD 2011 Global Cloud Computing Adoption, Attitudes and Approaches Study: Infographics.* `http://blogs.amd.com/work/amd-2011-global-cloud-computing-adoption-attitudes-and-approa` Mai 2011. [Online; Stand 28. November 2011].

[AMI11a] *AmazonIA: When Elasticity Snaps Back.* `http://www.trust.informatik.tu-darmstadt.de/fileadmin/user_upload/Group_TRUST/PubsPDF/BNPSS11.pdf`, Oktober 2011. [Online; Stand 25. April 2012].

[AMI11b] *AMI Exposed.* `http://www.secureworks.com/research/tools/amiexposed/`, Oktober 2011. [Online; Stand 28. November 2011].

[AMI11c] *AMID.* `http://trust.cased.de/AMID`, Oktober 2011. [Online; Stand 28. November 2011].

[AMI12] *AMI aiD (AMID) - Scanning a system for security or privacy critical data before publishing or when started as Amazon Machine Image (AMI).* `http://code.google.com/p/amid/`, 2012. [Online; Stand 25. April 2012].

[AWS08] *Amazon EBS (Elastic Block Store) - Bring Us Your Data.* `http://aws.typepad.com/aws/2008/08/amazon-elastic.html`, August 2008. [Online; Stand 30. Juli 2012].

[AWS12a] *AWS Documentation - Kernels and RAM Disk FAQ.* `http://docs.amazonwebservices.com/AWSEC2/latest/UserGuide/FAQs_Kernel.html`, 2012. [Online; Stand 26. Juli 2012].

[AWS12b] *AWS SDK for ruby.* `http://aws.amazon.com/de/sdkforruby/`, 2012. [Online; Stand 05. April 2012].

[AWS12c] *Cluster Instances.* `http://docs.amazonwebservices.com/AWSEC2/latest/UserGuide/using_cluster_computing.html`, 2012. [Online; Stand 30. Juli 2012].

[AWS12d] *Drives cannot be attached, detached, and then reattached.* `https://forums.aws.amazon.com/thread.jspa?messageID=337378`, April 2012. [Online; Stand 30. Juli 2012].

[AWS12e] *Enabling Your Own Linux Kernels.* `http://docs.amazonwebservices.com/AWSEC2/latest/UserGuide/UserProvidedkernels.html`, 2012. [Online; Stand 10. August 2012].

[AWS12f] *Oracle and AWS FAQs.* `http://aws.amazon.com/de/solutions/global-solution-providers/oracle/faqs/`, 2012. [Online; Stand 30. Juli 2012].

[AWS12g] *Sharing AMIs Safely.* `http://docs.amazonwebservices.com/AWSEC2/latest/UserGuide/AESDG-chapter-sharingamis.html`, 2012. [Online; Stand 14. August 2012].

[Azu12] *Windows Azure.* `http://www.windowsazure.com`, 2012. [Online; Stand 4. Juli 2012].

[Boc11] Bochum, Ruhr-Universität: *Cloud Computing: „Wolke“ mit Lücken.* `http://aktuell.ruhr-uni-bochum.de/pm2011/pm00336.html.de`, Oktober 2011. [Online; Stand 28. November 2011].

[Bot12] *Python interface to Amazon Web Services.* https://github.com/boto/boto, 2012. [Online; Stand 02. März 2012].

[Bug11] Bugiel, Sven: *AmazonIA: When Elasticity Snaps Back.* http://www.trust.informatik.tu-darmstadt.de/fileadmin/user_upload/Group_TRUST/PubsPDF/BNPSS11.pdf, 2011. [Online; Stand 28. November 2011].

[Chk12] *Chkrootkit.* http://www.chkrootkit.org/, 2012. [Online; Stand 21. März 2012].

[Clo12] Clouds.de: *Die erfolgreichsten Cloud Provider nach Amazon.* http://www.clouds.de/blog/295-erfolgreichte-cloud-anbieter, April 2012. [Online; Stand 3. Juli 2012].

[CVE12] *Common Vulnerabilities and Exposures.* http://cve.mitre.org/, 2012. [Online; Stand 28. November 2011].

[Dro12] *Dropbox.* https://www.dropbox.com/, 2012. [Online; Stand 19. August 2012].

[Exp12] *Exploits-Database by Offensive Security.* http://www.exploit-db.com/, 2012. [Online; Stand 28. November 2011].

[FC808] *AMI: 32bit Fedora 8 Xenified kernel - 2.6.21.7-2.fc8xen.* https://aws.amazon.com/amis/32bit-fedora-8-xenified-kernel-2-6-21-7-2-fc8xen, März 2008. [Online; Stand 10. August 2012].

[Fer07] Ferrie, Peter: *Attacks on More Virtual Machine Emulators.* http://ivanlef0u.fr/repo/windoz/attacks2.pdf, April 2007. [Online; Stand 28. November 2011].

[For12] *Social & Mobile Application Development Platform - Force.com.* http://www.force.com/, 2012. [Online; Stand 1. Juli 2012].

[fSI11] Sichere Informationstechnologie, Fraunhofer-Institut für: *Erhebliche Sicherheitsbedrohung durch sorglose Cloud-Nutzung.* http://www.sit.fraunhofer.de/de/medien-publikationen/

pressemitteilungen/2011/20110620-sicherheitsbedrohung-durch-cloud-nutzung.html, Juni 2011. [Online; Stand 28. November 2011].

[fSidI09] Informationstechnik, Bundesamt für Sicherheit in der: *1.2 Definitionen und Metriken für die Hochverfügbarkeit.* https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Hochverfuegbarkeit/1_2_Definitionen_pdf.pdf?__blob=publicationFile, 2009. [Online; Stand 11. Juni 2012].

[Goo12a] *Google App Engine.* https://developers.google.com/appengine/, 2012. [Online; Stand 1. Juli 2012].

[Goo12b] *Google Compute Engine.* http://cloud.google.com/products/compute-engine.html, 2012. [Online; Stand 3. Juli 2012].

[Goo12c] *Google Drive. Speichern und Zusammenarbeiten leicht gemacht.* https://drive.google.com, 2012. [Online; Stand 19. August 2012].

[Had12] *Apache Hadoop.* http://hadoop.apache.org, 2012. [Online; Stand 4. Juli 2012].

[Han12] Handelsblatt: *Google fordert Amazon beim Cloud-Computing heraus.* http://www.handelsblatt.com/technologie/it-tk/it-internet/cloud-dienste-google-fordert-amazon-beim-cloud-computing-heraus/6812308.html, Juni 2012. [Online; Stand 3. Juli 2012].

[Hei09] *Sicheres Löschen: Einmal überschreiben genügt.* http://heise.de/-198816, Januar 2009. [Online; Stand 21. März 2012].

[hiv12] *hivex - Windows Registry "hive" extraction library.* http://libguestfs.org/hivex.3.html, 2012. [Online; Stand 15. Juli 2011].

[Kae06] Kaeo, Merike: *IPv6 Security Technology Paper.* http://en.scientificcommons.org/23518655, Juli 2006. [Online; Stand 28. November 2011].

[Lib12] *libguestfs, library for accessing and modifying VM disk images.* http://libguestfs.org//, 2012. [Online; Stand 26. Juli 2012].

[Loa12] *Load Balancing.* http://www.elektronik-kompendium.de/sites/net/0906201.htm, 2012. [Online; Stand 19. August 2012].

[Met12] *Metasploit.* http://metasploit.org/, 2012. [Online; Stand 28. November 2011].

[Mic12] Microsoft: *Office Online Services - Hosted in the Cloud - Microsoft Office 365.* http://office365.microsoft.com, 2012. [Online; Stand 1. Juli 2012].

[MK12] Michael Kofler, Ralf Spenneberg: *KVM für die Server-Virtualisierung*, Mai 2012.

[Mos09] Moser, Scott: *ServerLucidCloudKernelRamdisk.* https://wiki.ubuntu.com/ServerLucidCloudKernelRamdisk, November 2009. [Online; Stand 11. August 2012].

[Net12] *January 2012 Web Server Survey.* http://news.netcraft.com/archives/2012/01/03/january-2012-web-server-survey.html#more-5297, Januar 2012. [Online; Stand 28. November 2011].

[NVD12] *National Vulnerability Database.* http://nvd.nist.gov/, 2012. [Online; Stand 28. November 2011].

[Ope12] *OpenStack Open Source Cloud Computing Software.* http://openstack.org/, 2012. [Online; Stand 1. Juli 2012].

[Ora12] *Oracle Linux.* http://www.oracle.com/de/technologies/linux/index.html, 2012. [Online; Stand 11. August 2012].

[Osv12] *The Open Source Vulnerability Database.* http://osvdb.org/, 2012. [Online; Stand 28. November 2011].

[OVM12] *Oracle VM.*
`http://www.oracle.com/us/technologies/virtualization/oraclevm/index.html`, 2012. [Online; Stand 30. Juli 2012].

[Pho11] *Finally! Ubuntu 12.04 LTS Will Recommend 64-bit.*
`http://www.phoronix.com/scan.php?page=news_item&px=MTAxMTQ`, November 2011. [Online; Stand 30. Juli 2012].

[Pic11] *AMD 2011 Global Cloud Computing Adoption, Attitudes and Approaches Study: Infographics.*
`http://blogs.amd.com/work/amd-2011-global-cloud-computing-adoption-attitudes-a`
Mai 2011. [Online; Stand 28. November 2011].

[Pyt12a] *Python Programming Language – Official Website.*
`http://python.org/`, 2012. [Online; Stand 28. November 2011].

[Pyt12b] *Python Success Stories.*
`http://www.python.org/about/success/#software-development`, 2012. [Online; Stand 28. November 2011].

[Rac12] *Rackspace.*
`http://www.rackspace.com`, 2012. [Online; Stand 4. Juli 2012].

[RKH12] *Rootkit Hunter.*
`http://rkhunter.sourceforge.net/`, 2012. [Online; Stand 21. März 2012].

[Rub12] *Ruby - A programmer's best friend.*
`http://www.ruby-lang.org`, 2012. [Online; Stand 05. April 2012].

[Rut08] Rutkowska, Joanna: *Bluepilling the Xen Hypervisor.*
`http://www.invisiblethingslab.com/resources/bh08/part3.pdf`, August 2008. [Online; Stand 28. November 2011].

[Sch11] *Schwarzmarktpreise für gestohlene Online-Banking-Daten ermittelt.*
`http://heise.de/-1183524`, Februar 2011. [Online; Stand 12. August 2012].

[Sec12a] *Securityfocus.*
http://www.securityfocus.com/, 2012. [Online; Stand 28. November 2011].

[Sec12b] *THC-SecureDelete.*
http://www.thc.org/releases.php, 2012. [Online; Stand 14. April 2012].

[She05] Shelton, Tim: *ACS Security Assessment Advisory - Remote Heap Overflow.*
http://lists.grok.org.uk/pipermail/full-disclosure/2005-December/040442.html, Dezember 2005. [Online; Stand 28. November 2011].

[Som11] Somorovsky, Juraj: *All Your Clouds are Belong to us – Security Analysis of Cloud Management Interfaces.*
http://www.computerworld.com.pt/media/2011/10/AmazonSignatureWrapping.pdf, Oktober 2011. [Online; Stand 28. November 2011].

[Spa08] Sparks, R.: *RFC5393: Addressing an Amplification Vulnerability in Session Initiation Protocol (SIP) Forking Proxies.*
http://www.hjp.at/doc/rfc/rfc5393.html, Dezember 2008. [Online; Stand 28. November 2011].

[Spe04] Specht, Stephen M.: *Distributed Denial of Service: Taxonomies of Attacks, Tools and Countermeasures.*
http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.133.4566&rep=rep1&type=pdf, September 2004. [Online; Stand 28. November 2011].

[st11] team, Xen.org security: *Xen Security Advisory 4 (CVE-2011-2901) - Xen 3.3 vaddr validation.*
http://xen.1045712.n5.nabble.com/Xen-Security-Advisory-4-CVE-2011-2901-Xen-3-3-vaddr-va
html, September 2011. [Online; Stand 11. Juni 2012].

[Sys12] Systems, Eucalyptus: *Eucalyptus.*
http://www.eucalyptus.com/, 2012. [Online; Stand 4. Februar 2012].

[Tec12] *The ARM-Powered Cloud Comes To OpenStack.*
http://techcrunch.com/2012/07/18/the-arm-powered-cloud-comes-to-openstack/,

Juli 2012. [Online; Stand 30. Juli 2012].

[Ubu12a] *Maverick Meerkat.*
`http://wiki.ubuntuusers.de/Maverick_Meerkat`, 2012. [Online; Stand 16. August 2012].

[Ubu12b] *Precise Pangolin.*
`http://wiki.ubuntuusers.de/Precise_Pangolin`, 2012. [Online; Stand 16. August 2012].

[VMw12] *VMware.*
`http://www.vmware.com/`, 2012. [Online; Stand 4. Juli 2012].

[Wri08] WRIGHT: *Overwriting Hard Drive Data: The Great Wiping Controversy.*
`http://www.springerlink.com/content/408263ql11460147/`, 2008. [Online; Stand 12. März 2011].

[XEN11] *Xen Wiki - XenOverview.*
`http://wiki.xensource.com/xenwiki/XenOverview`, März 2011. [Online; Stand 30. Juli 2012].

[XEN12] *Xen.org.*
`http://xen.org/`, 2012. [Online; Stand 30. Juli 2012].