A=SIT Technology
E-Government Innovationszentrum

8. Mai 2019 Studie · Digitale Identität · Privatsphäre & Krypto · 15 Min. Lesezeit

Mögliche Seitenkanäle durch Deduplikation

Archiviert. Dieser Inhalt wird nicht mehr aktualisiert. Er bleibt zur Information verfügbar.

Stand 2019-03-20
Quelle https://technology2.a-sit.at/veroeffentlichungen/moegliche-seitenkanaele-durch-deduplikation/

Moderne Dateisysteme wie ZFS bieten durch die Unterstützung von Copy on Write (CoW) erweiterte Funktionen wie Deduplikation an. Durch Deduplikation können mehrfach vorhandene Dateien zusammengeführt werden. Dadurch kann wertvoller Speicherplatz gespart werden. Durch den Einsatz von CoW ändern sich jedoch die Ausführungszeiten bestimmter Operationen. So konnten wir nachweisen, dass der Schreibvorgang beim Schreiben bereits vorhandener Daten weniger Zeit benötigt, als bei noch nicht vorhandenen Daten. Dadurch kann ein Angreifer mit Systemzugriff Rückschlüsse auf andere auf dem System gespeicherte Daten ziehen. Besonders im Umfeld von Mehrbenutzersystemen und Virtual Private Server (VPS)-Systeme sollte deshalb der Einsatz von Deduplikation überdacht werden.

1. Einleitung

Der Cloud-Markt wächst stetig [1]. Im Cloud-Bereich werden, besonders für weniger rechenintensive Einsatzgebiete, sogenannte Virtual Private Server (VPS) verwendet. Hierbei nutzen mehrere Teilnehmer dieselbe Hardware. Dadurch können Ressourcen wie CPU-Zeit geteilt werden. Infolgedessen kann die Hardware besser ausgelastet und günstiger angeboten werden. Um die Sicherheit der einzelnen Teilnehmer zu gewährleisten werden in diesem Umfeld virtuelle Maschinen verwendet. Dadurch wird verhindert, dass einzelne Teilnehmer auf die Daten anderer Teilnehmer zugreifen können. Diese virtuellen Maschinen werden als Abbild gespeichert. Die Abbilder benötigen viel Speicherplatz, sie haben jedoch oft überlappende Bereiche durch den Einsatz ähnlicher Anwendungssoftware und Betriebssysteme. Damit die gleichen Daten nicht mehrmals abgespeichert werden müssen, wurden verschiedene Ansätze basierend auf Deduplikation entwickelt, die bei der Speicherung von Abbildern virtueller Maschinen eingesetzt werden können

[2] [3] [4]. So kann effektiv Speicherplatz gespart werden. Ähnlich wie bei der Speicherung der Abbilder kann Deduplikation auch für den Inhalt des Hauptspeichers verwendet werden. Hierbei wurde jedoch gezeigt, dass die Deduplikation des Hauptspeichers von geteilten Servern ein Sicherheitsrisiko darstellen kann [5]. Im Hinblick auf diese Erkenntnis untersucht dieser Bericht, ob die Deduplikation der Abbilder auch Informationen über andere Teilnehmer offenbart.

2. Hintergrund

Diese Sektion gibt Hintergrundwissen über den Funktionsumfang moderner Dateisysteme und beschreibt unterschiedliche Arten von Deduplikation.

2.1. Unterstützung für Deduplikation

Moderne Dateisysteme wie ZFS [6] und btrfs [7] bieten im Vergleich zu herkömmlichen Dateisystemen einen erweiterten Funktionsumfang. So unterstützen diese Dateisysteme neben einer Vielzahl neuer Funktionen auch Copy on Write (CoW). Durch CoW können diese Dateisysteme einzelne Blöcke mehrfach referenzieren. Dadurch kann Speicherplatz gespart werden. Dank der Unterstützung für CoW erlauben es diese Dateisysteme, Dateien mit (teilweise) gleichem Inhalt zu deduplizieren. Dadurch werden mehrfach vorhandene Dateien bzw. Teile davon nur einmal auf dem Speichermedium abgelegt. Weitere Vorteile sind z.B. die Unterstützung für Snapshots, die es dem Nutzer ermöglichen, unmittelbar Abbilder von ganzen Partitionen anzulegen, ohne dabei den Speicherplatzbedarf zu verdoppeln. Auch Entwickler traditionellerer Dateisysteme haben den Vorteil von CoW erkannt – so erhielt beispielsweise das Dateisystem XFS mit dem Linux Kernel 4.9 die Unterstützung für CoW und damit auch für Deduplikation [8].

2.2. In-Line- und Post-Process-Deduplikation

Bei der Unterstützung von Deduplikation gibt es zwei verschiedene Ansätze, die sich im Zeitpunkt der Deduplikation unterscheiden. Bei der sogenannten In-Line- oder Inband-Deduplikation wird die Deduplikation direkt beim Schreiben der Daten durchgeführt. Dadurch werden Daten, die bereits in dieser Form auf dem Datenträger vorhanden sind, sofort auf den bestehenden Datenblock referenziert und müssen nicht erst auf das Medium geschrieben werden. Um den Schreibvorgang nicht unnötig zu verlangsamen, werden bei der Online-Deduplikation Hashwerte der bereits auf dem Datenträger vorhandenen Blöcke im Hauptspeicher gehalten. Dadurch steigt der Speicherverbrauch mit steigender Anzahl an auf dem Datenträger vorhandenen Daten. Bei der Post-Process-Deduplikation hingegen werden die Daten wie bei herkömmlichen Dateisystemen zuerst vollständig auf den Datenträger geschrieben. Erst zu einem späteren Zeitpunkt werden die Daten verglichen und Duplikate entfernt. Dieser Vorgang kann in der Regel durch den Nutzer manuell gestartet oder automatisch in regelmäßigen Abständen erfolgen. Das Dateisystem ZFS unterstützt die In-Line-Deduplikation, während btrfs und XFS auf die Post-Process-Deduplikation setzen.

2.3. Das ZFS Dateisystem

ZFS arbeitet mit Speicherpools, die im Allgemeinen als *zpools* bezeichnet werden. In diesen Speicherpools können beliebig viele Datensätze erstellt werden. Für diese Datensätze können Optionen wie Deduplikation oder Komprimierung individuell konfiguriert werden. Auch wenn zwei identische Dateien auf unterschiedlichen Datensätzen gespeichert sind, werden die Daten über diese verschiedenen Datensätze hinweg dedupliziert, solange sich diese Datensätze auf demselben zpool befinden und für beide die Deduplikation aktiviert wurde. ZFS dedupliziert auf Blockebene, d.h. es werden immer einzelne Blöcke miteinander verglichen. Diese Blöcke werden als *records* bezeichnet. Die maximale Größe dieser records kann über den Parameter *recordsize* konfiguriert werden. Diese records haben standardmäßig eine Größe von 128 KiB. Diese Maximalgröße kann an den jeweiligen Einsatzzweck angepasst werden. Eine Besonderheit von ZFS ist, dass records auch kleiner als die definierte recordsize sein können. Diese

Eigenschaft wird verwendet, um kleine Dateien effizienter zu speichern. Da wir uns in diesem Bericht jedoch mit großen Abbildern virtueller Maschinen beschäftigen, beeinflusst diese Eigenschaft die folgenden Experimente nicht.

Beim Schreiben von Daten berechnet ZFS einen Hashwert der geschriebenen Datensätze und speichert diesen zusammen mit anderen Metadaten auf Medium. Dieser Hashwert wird unabhängig von der Verwendung der Deduplikation generiert, da er beispielsweise auch zur Überprüfung der Integrität gespeicherter Datensätze verwendet wird. Wenn die Deduplikation aktiviert ist, werden die Hashes und andere Metadaten, die zum Erkennen replizierter Datensätze erforderlich sind, in der DeDuplicationTable (DDT) gespeichert, der im Adaptivte Replacement Cache (ARC) von ZFS gespeichert ist. Der ARC befindet sich im Allgemeinen im Hauptspeicher des Systems, der zusätzlich durch separate Cache-Geräte ergänzt werden kann. Wenn die Deduplikation aktiviert ist, vergleicht ZFS den berechneten Hashwert mit den Hashwerten im DDT. Wenn der Hash in der DDT vorhanden ist, wird der Referenzzähler für diesen Datensatz erhöht und der Schreibvorgang abgeschlossen. Daher muss der Datensatz nicht auf das Medium geschrieben werden. Wenn der Hash dagegen nicht im DDT gefunden werden kann, muss der Datensatz an das zugrundeliegende Speichermedium übergeben werden. Daher kann davon ausgegangen werden, dass der Schreibvorgang früher abgeschlossen wird, wenn die Datei bereits auf der Festplatte vorhanden ist. Diese Annahme wird im folgenden Abschnitt überprüft.

3. Resultate

Die folgenden Unterabschnitte beschreiben den Versuchsaufbau, die durchgeführten Experimente und die erzielten Resultate.

3.1. Versuchsaufbau

In unserer Evaluierung konzentrieren wir uns auf das Dateisystem ZFS. Durch die sofortige Deduplikation der Inhalte dieses In-Line-Deduplikation verwendenden Dateisystem können direkt beim Schreiben von Daten die Messungen durchgeführt werden. Im Vergleich zu Post-Process-Deduplikation, wie sie in btrfs verwendet wird, muss also nicht darauf gewartet werden, bis der Deduplikationprozess durchgeführt wurde.

Die Experimente wurden auf einem Rechner mit Intel® Core™ i5-6200U CPU und 12GB Arbeitsspeicher durchgeführt. Beim Hostbetriebssystem handelt es sich um Ubuntu 18.04 LTS mit Linux Kernel 4.15 und ZFS on Linux 0.7.5. Während ZFS ursprünglich für das Betriebssystem Solaris entwickelt wurde, unterstützt auch Ubuntu seit der Version 16.04 LTS dieses Dateisystem offiziell [9]. Die virtuellen Maschinen laufen im Experiment auf QEMU/KVM. Die Abbilder der virtuellen Maschinen werden auf einem gemeinsamen ZFS-Pool gespeichert, welcher eine Recordgröße von 64 KiB verwendet. Die angegriffene virtuelle Maschine läuft während den Experimenten im Hintergrund.

Das Angriffsprogramm wurde auf einer virtuellen Maschine mit Ubuntu 18.10 ausgeführt. Dazu wurden die zu entdeckende Datei in die VM geladen. Die Datei wurde dabei in ein *tmpfs* gespeichert, d.h., sie wurden nicht in das VM-Image, sondern in den Hauptspeicher geladen. Um generell falsche Resultate zu vermeiden, sollten diese Dateien möglichst wenig Überlappung mit den vom Angreifer gespeicherten Daten haben.

Während der Auswertungsphase wird die zu erkennende Datei direkt in den Speicher gemappt. Danach schreibt das Messprogramm die Datei auf das Dateisystem der virtuellen Maschine und misst die dafür benötigte Zeit. Um die Ausführungszeit korrekt messen zu können, wird im Messprogramm beim Öffnen der zu schreibenden Datei der Dateideskriptor mit der Option `O_SYNC` geöffnet. Dadurch wartet das Betriebssystem, bis alle Daten geschrieben wurden und setzt im Vergleich zum Standardverhalten den Programmfluss nicht fort. Zusätzlich wurde die Ausführungszeit der Schreiboperationen auf Basis von Intels® Leitlinie zum Benchmarken von Code

[10] gemessen. Dadurch wird verhindert, dass die Messergebnisse durch die Out-of-order-Ausführung moderner Prozessoren gegebenenfalls verfälscht werden.

Durch zeitgleich ablaufende Vorgänge auf dem Datenträger bzw. dem System kann es zu Unterschieden bei der Ausführungszeit kommen. Um solche Unterschiede auszugleichen, können die Messungen mehrmals ausgeführt und danach statistisch analysiert werden. Um nach einer durchgeführten Messung den Ausgangspunkt wiederherzustellen, wird die veränderte Datei nach jedem Messvorgang mit Zufallsdaten überschrieben und gelöscht.

3.2. Erkennung eines möglichen Offsets

Wie im vorherigen Abschnitt erwähnt, arbeitet unser ZFS-Pool mit Records, die eine Größe von 64 KiB haben. Wir haben jedoch festgestellt, dass die meisten Dateisysteme, die während der Installation eines Betriebssystems erstellt wurden, eine Blockgröße von 4 KiB verwenden. Replizierte Dateien werden daher möglicherweise nicht auf die Recordgröße ausgerichtet und dadurch gegebenenfalls nicht dedupliziert. Während wir beobachtet haben, dass einige Betriebssysteme (z.B. Ubuntu 18.10) (große) Dateien auf 64 KiB-Blöcke ausrichten, ist dies bei anderen Betriebssystemen wie Windows 10 nicht der Fall. Um einen möglichen Diskrepanz der Ausrichtung auf realen Systemen zu berücksichtigen, können die Messungen mit gepaddeten Dateien wiederholt aufgerufen werden. Im Folgenden gehen wir davon aus, dass die virtuelle Maschine eine Blockgröße von 4 KiB verwendet. Daher werden wir die Datei am Anfang in Schritten von 4 KiB auffüllen. Um zu vermeiden, dass die Messungen durch das Schreiben von Paddings unterschiedlicher Größe verzerrt werden, wird die geschriebene Datei immer mit 64 KiB aufgefüllt. Der Teil des Pads, der nicht am Anfang der Datei verwendet wird, wird an das Ende der Datei angehängt. Abbildung 1 zeigt die Ergebnisse der Erkennung einer nicht ausgerichteten Datei in einer virtuellen Maschine mit Windows 10. Wie in dieser Abbildung ersichtlich, benötigt das Schreiben der Datei mit einem Padding von 36 KiB (Messungen 73-79) weniger Zeit als das Schreiben der Datei mit einem anderen Offset. Dadurch können wir darauf schließen, dass die Datei an diesem Offset in einer anderen virtuellen Maschine vorhanden ist. In den folgenden Unterabschnitten gehen wir davon aus, dass die Dateien an der Datensatzgröße des ZFS-Pools ausgerichtet sind.

Abbildung 1 Erkennung des Offsets innerhalb eines ZFS Records.

3.3. Erkennung benachbarter Betriebssysteme

In der ersten Fallstudie identifizieren wir das Betriebssystem, das auf einer benachbarten virtuellen Maschine verwendet wird. Um das Betriebssystem zu identifizieren, wurde eine für das Betriebssystem charakteristische Datei ausgewählt. In diesem Experiment wird zuerst festgestellt, ob Ubuntu auf einer benachbarten virtuellen Maschine installiert ist. Die Datei *gnome-3-26- 1604_70.snap* wurde zur Identifizierung des Betriebssystems ausgewählt, da es standardmäßig mit dem Betriebssystem ausgeliefert

wird, selbst wenn während der Installation ausgewählt wird, dass nur eine minimale Anzahl an Applikationen installiert werden soll. Da die zur Durchführung der Messungen verwendete virtuelle Maschine auch Ubuntu 18.10 verwendet, wurde die entsprechende Datei vor den Messläufen von diesem System entfernt, um eine Verfälschung der Ergebnisse zu vermeiden. Die Messungen sind in Abbildung 2 zu finden. Die roten Balken zeigen die Experimente, bei denen Ubuntu nicht auf demselben Medium installiert war, während die blauen Balken die Experimente zeigen, bei denen Ubuntu installiert war. Die Abbildung zeigt 25 verschiedene Messungen für jeden der beiden Fälle, von insgesamt 200 (100 je Fall) durchgeführten Messungen. Wie in der Abbildung gezeigt wird, ist die Zeit, die zum Schreiben der Datei benötigt wird, im Allgemeinen viel höher, wenn die Datei nicht vorhanden ist, was unsere frühere Annahme bestätigt. Aufgrund einiger Ausreißer ist es ratsam, mehrere Messungen durchzuführen, bevor eine Schlussfolgerung über das verwendete Betriebssystem gezogen wird. Aber auch unter Berücksichtigung der Ausreißer konnten wir mit Hilfe von k-means 81% der Messungen korrekt gruppieren. Da jede Messung nur wenige Sekunden dauert, ist es möglich, das benachbarte Betriebssystem in kurzer Zeit zu erkennen, selbst wenn mehrere Messungen durchgeführt werden.

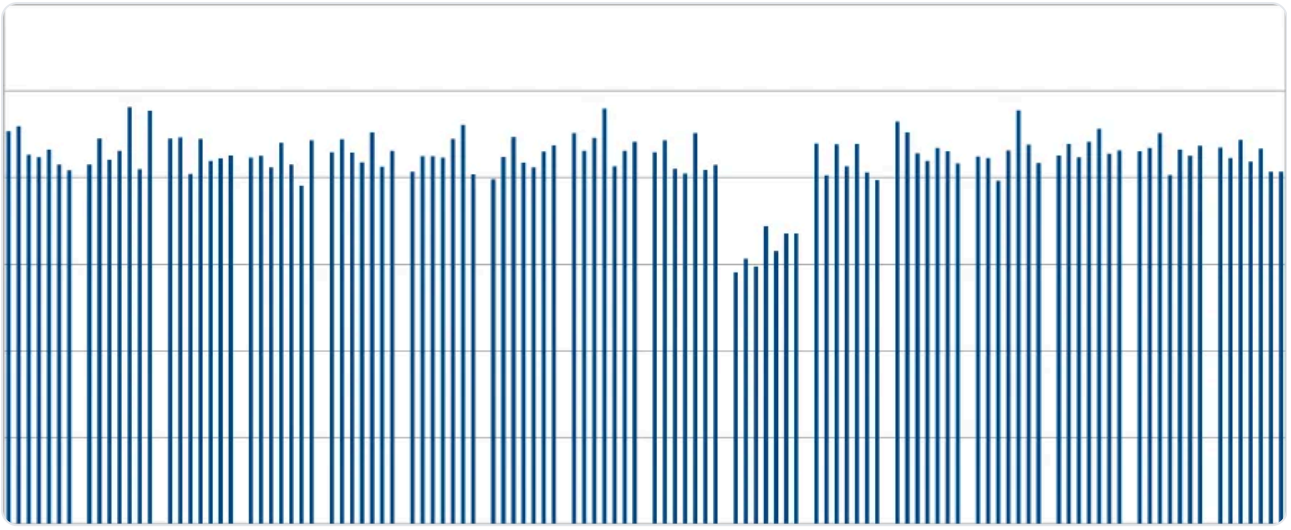


Abbildung 2: Ausführungszeit des Schreibprozesses der Datei gnome-3-26-160470.snap. Der Vollständigkeit halber identifizieren wir auch eine benachbarte VM mit Windows 10. Die Messungen sind in Abbildung 3 dargestellt. Ähnlich wie in der vorherigen Abbildung zeigen die roten Balken die Experimente, bei denen Windows 10 nicht installiert wurde, während die blauen Balken die Experimente zeigen, bei denen Windows 10 installiert wurde. In diesem Fall wurde die Bibliothekdatei Microsoft.Photos.dll zur Identifizierung des Betriebssystems ausgewählt, da diese nur in Windows 10 vorhanden ist. Wie im vorherigen Experiment kann wieder unterschieden werden, ob das Betriebssystem in einer benachbarten VM eingesetzt wird oder nicht. Zusammenfassend ist es also möglich, sowohl Linux- als auch Windows-basierte virtuelle Maschinen zu erkennen.

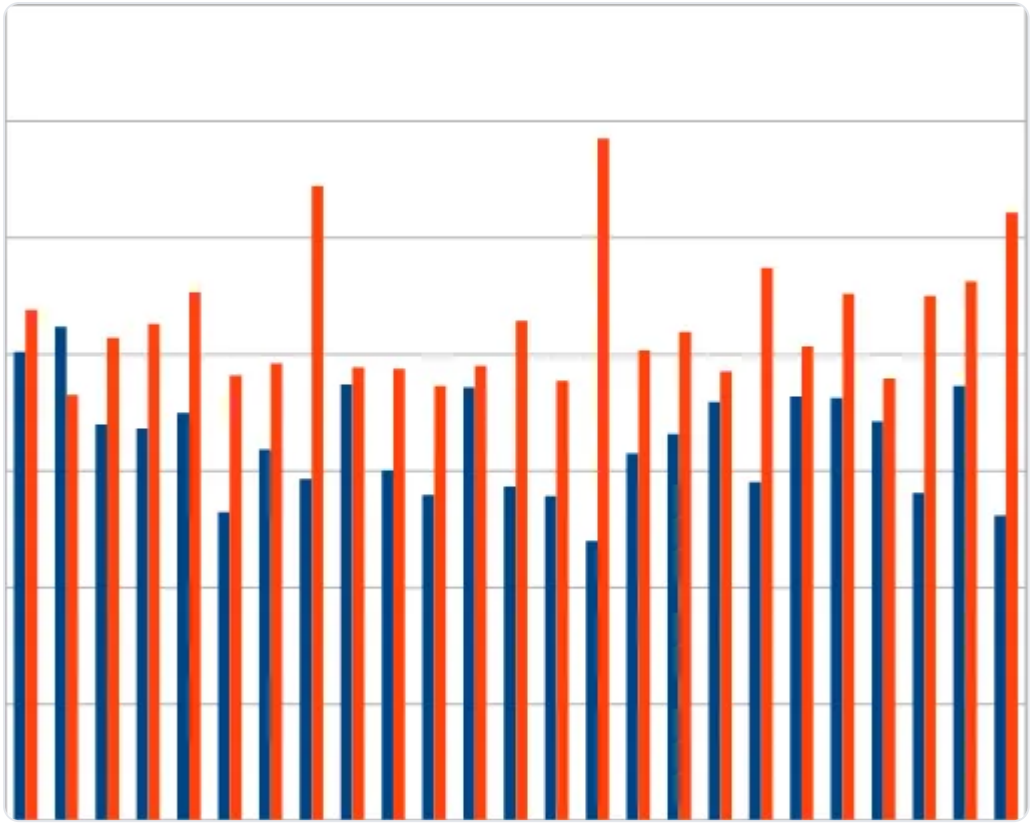
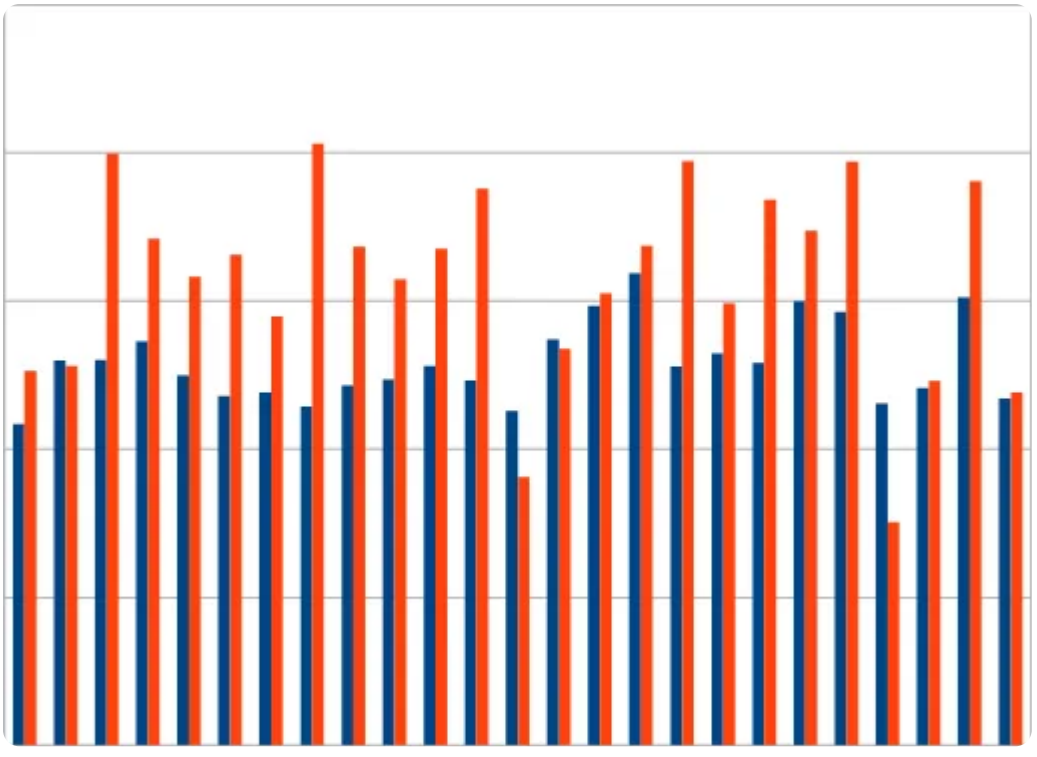


Abbildung 3: Ausführungszeit des Schreibprozesses der Datei Microsoft.Photos.dll.



3.4. Erkennung von Applikationen

Die folgende Fallstudie zeigt, dass es auch möglich ist, einzelne Anwendungen in einer parallel laufenden virtuellen Maschine zu erkennen. Auf der benachbarten virtuellen Maschine wird Ubuntu 18.10 ausgeführt. Die zu erkennende Anwendung ist der Firefox-Webbrowser. Die gesammelten Messergebnisse für dieses

Experiment sind in Abbildung 4 gezeigt. Ähnlich wie in der vorherigen Fallstudie sind 25 Messungen dargestellt, wobei die roten Balken den Fall veranschaulichen, in dem Firefox nicht installiert ist, während die blauen Balken die Experimente zeigen, in denen Firefox in einer benachbarten virtuellen Maschine installiert wurde. In den Testfällen, in denen Firefox installiert ist, wurde es aus den offiziellen Paketquellen von Ubuntu installiert. Wir identifizieren Firefox durch Erkennen der gemeinsam genutzten Datei *libxul.so*, die ein integraler Bestandteil des Webbrowsers ist. Insgesamt haben wir für dieses Experiment 100 Messläufe (jeweils 50 pro Fall) durchgeführt. Durch den Einsatz von k-means war es möglich, 83% der Messungen korrekt zu gruppieren, was zeigt, dass es tatsächlich möglich ist, das Vorhandensein einer Anwendung in einer benachbarten virtuellen Maschine zu erkennen.

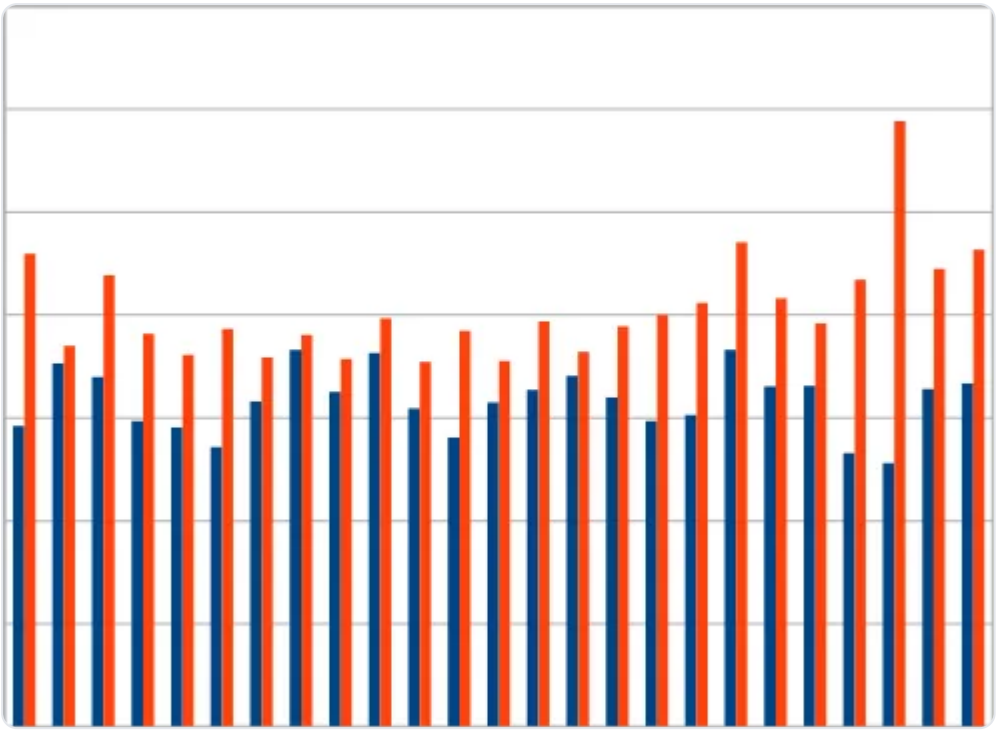


Abbildung 4: Ausführungszeit des Schreibprozesses der Datei libxul.so.

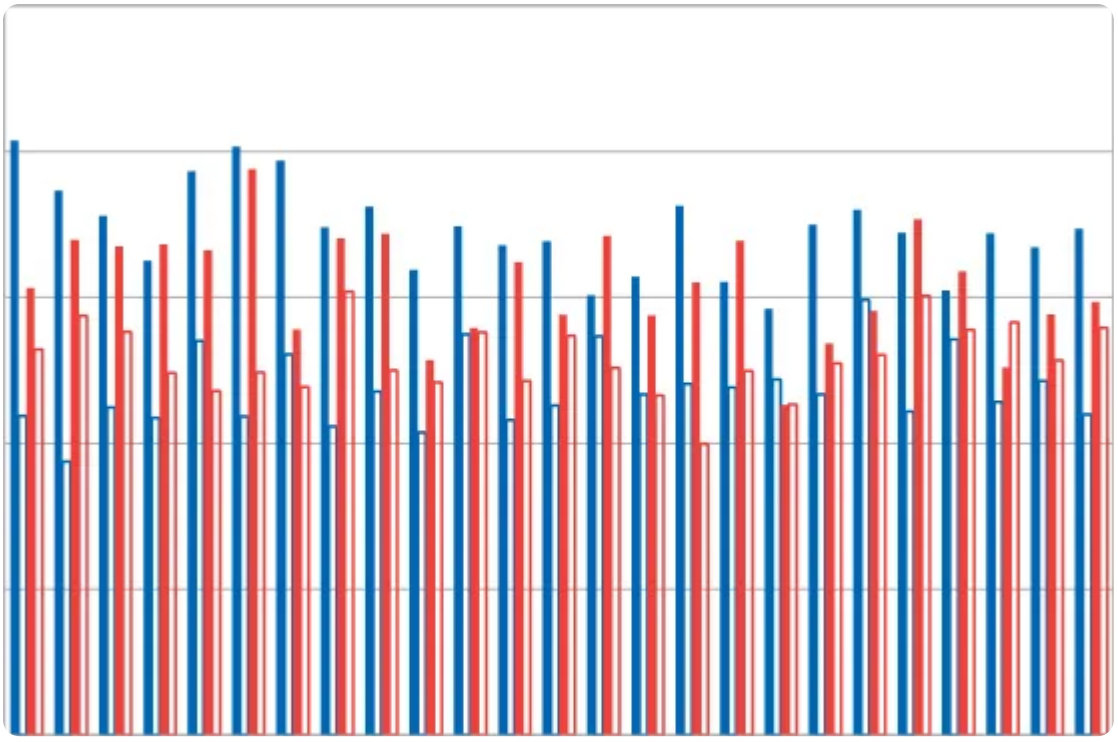
3.5. Erkennung (veralteter) Programmversionen

Noch interessanter als das bloße Vorhandensein einer Anwendung ist der Patch-Level eines Programms, da begleitende Changelogs Informationen über potenzielle Schwachstellen liefern. So zeigen wir im nächsten Experiment, dass es möglich ist, die genaue Versionsnummer von Firefox zu ermitteln. Abbildung 5 zeigt diese Ergebnisse. Im Experiment unterscheiden wir vier verschiedene Fälle:

1. Firefox 65.0 ist installiert, es wird jedoch nach Firefox 65.0.1 gesucht
2. Firefox 65.0.1 ist installiert, Firefox 65.0.1 wird gesucht
3. Firefox 65.0.1 ist installiert, aber es wird nach Firefox 65.0 gesucht
4. Firefox 65.0 ist installiert, Firefox 65.0 wird gesucht Wie in den vorherigen Experimenten zeigt das Diagramm für jeden Fall 25 Messungen. Dieses Experiment zeigt, dass es möglich ist, die genaue Version einer Anwendung zu erkennen, indem man die Ausführungszeit des Schreibens der entsprechenden Datei misst. Folglich ist es also möglich, Anwendungen mit bekannten Schwachstellen in einer benachbarten virtuellen Maschine zu erkennen. Zum Beispiel wurden in Firefox 65.0.1 vier verschiedene Sicherheitslücken geschlossen (siehe https://www.mozilla.org/en-

US/security/advisories/mfsa2019-05/), von denen einige offen dokumentiert sind. Das Update enthielt keine weiteren Änderungen. Ein Angreifer kann also möglicherweise die bekannten Sicherheitslücken verwenden, um eine benachbarte virtuelle Maschine anzugreifen.

Ähnlich wie bei den vorherigen Experimenten verwendeten wir k-means, um alle 200 durchgeführten Messläufe (100 für jeden Fall) in zwei Cluster zu gruppieren. Mit k-means konnten wir 91,5% der Messungen korrekt zwischen Fall 1 und 2 unterteilen. Bei der Anwendung von k-means auf Fall 3 und 4 können 93,5% der Schreibmessungen korrekt partitioniert werden. Dieses Experiment zeigt, dass es möglich ist, die installierte Version einer Anwendung mit hoher Genauigkeit zu ermitteln, selbst wenn nur eine einzige Messung ausgewertet wird.



1 2 3 4 5 6 7 8 9 10111213141516171819202122232425 65.0 installiert, teste auf 65.0.1 65.0.1 installiert, teste auf 65.0.1 65.0.1 installiert, teste auf 65.0 65.0 installiert, teste auf 65.0 *Abbildung 5 Ausführungszeit des Schreibprozesses verschiedener Versionen der Datei libxul.so.*

4. Diskussion

Dieser Abschnitt behandelt Einschränkungen des Angriffsvektors und mögliche Gegenmaßnahmen auf Seiten von Betreibern und Nutzern der virtuellen Maschinen.

4.1. Einschränkungen

Durch die Attacke kann nicht herausgefunden werden, ob ein Programm oder eine virtuelle Maschine zum aktuellen Zeitpunkt läuft oder sich nur auf dem Datenträger befindet. Beim Vorhandensein mehrerer benachbarter virtuellen Maschinen kann ebenso nicht unterschieden werden, auf welcher virtuellen Maschine sich ein Programm befindet. Damit ist diese Attacke vergleichbar mit Attacken auf die Seitendeduplikation im Arbeitsspeicher wie [5]. Da auch nicht ausgeführte Programmdateien im Arbeitsspeicher liegen können, kann auch hier keine definitive Aussage über aktuell laufende Programme

gemacht werden. Ebenso ist es nicht möglich zu erkennen, in welcher benachbarten virtuellen Maschine sich ein Programm befindet. Dennoch kann die Möglichkeit, Dateien in anderen virtuellen Maschinen zu erkennen, die Sicherheits- und Datenschutzanforderungen von VM-Nutzern unterminieren.

4.2. Gegenmaßnahmen

Um diese Sicherheitslücke zu vermeiden, können Abbilder verschiedener virtueller Maschinen auf eigenen ZFS-Pools bzw. Partitionen gespeichert werden, die unabhängig voneinander dedupliziert werden. Dadurch kann können Dateien in anderen virtuellen Maschinen nicht mehr erkannt werden. Durch den Einsatz dieser Gegenmaßnahme können jedoch nur mehr die Dateien innerhalb eines Abbilds dedupliziert werden. Da die Speichervorteile vor allem daraus resultieren, dass sich die Betriebssystem- und Programmdateien verschiedener Systeme gleichen, wird so der Nutzen der Deduplikation stark verringert. Alternativ zur Deduplikation kann auch Komprimierung eingesetzt werden. Dadurch werden zwar doppelt auf dem Datenträger vorhandene Dateien nicht verhindert, es kann aber dennoch Speicherplatz gespart werden. Zusätzlich benötigt die Komprimierung weniger Arbeitsspeicher im Vergleich zur In-Line-Deduplikation, da Informationen, die zur Deduplikation benötigt werden, hierbei nicht anfallen. Vor allem bei vielen duplizierten Dateien dürfte die Speicherplatzreduktion durch Komprimierung alleine jedoch geringer ausfallen.

Auf Seiten der Nutzer virtueller Maschinen kann Verschlüsselung eingesetzt werden. Durch die Verschlüsselung wird auch eine Deduplikation der Daten verhindert. Dadurch kann wirksam davor vorgebeugt werden, dass Applikationen oder in der virtuellen Maschine gespeicherte Daten erkannt werden. Hierbei ist jedoch darauf zu achten, dass auch bei einer vollständigen Festplattenverschlüsselung Daten die für eine Identifizierung ausreichen unverschlüsselt im Abbild gespeichert werden können [11]. Deswegen sollten auch Dateien wie der Linux-Kernel mitverschlüsselt werden [12], die bei einer standardmäßigen Installation oft unverschlüsselt gespeichert werden.

5. Fazit

Durch den Einsatz von Deduplikation kann Speicherplatz gespart werden. Vor allem im VM-Umfeld, wo oft ähnliche Basisabbilder eingesetzt werden, kann die Reduktion des benötigten Speicherplatzes substantiell sein. Die Verwendung von Deduplikation kann jedoch Informationen preisgeben, die die Privatsphäre und Sicherheit anderer Nutzer beeinträchtigt. So konnten wir benachbarte Betriebssysteme, installierte Programme und sogar deren Patchlevel erkennen. Durch diese angeführten Nebeneffekte und daraus resultierenden Sicherheitsbedenken sollte, trotz der Speicherplatzvorteile, der Einsatz von Deduplikation in gewissen Bereichen, wo sich mehrere Nutzer denselben darunterliegenden Speicher teilen, überdacht werden.

Referenzen

[1] Statista, „Cloud services market revenue in Austria from 2016 to 2021 (in million U.S. dollars),“

2016, 2017.

[2] C.-H. Ng, M. Ma, T.-Y. Wong, P. P. C. Lee und J. C. S. Lui, „Live Deduplication Storage of Virtual Machine Images in an Open-Source Cloud,“ in *Middleware*, Lissabon, Portugal, 2011.

[3] K. Jin und E. L. Miller, „The Effectiveness of Deduplication on Virtual Machine Disk Images," in *SYSTOR*, Haifa, Israel, 2009.

[4] X. Zhao, Y. Zhang, Y. Wu, K. Chen, J. Jiang und K. Li, „Liquid: A Scalable Deduplication File System for Virtual Machine Images," in *IEEE Transactions on Parallel and Distributed Systems*, 2014.

[5] K. Suzaki, K. Iijima, T. Yagi und C. Artho, „Memory deduplication as a threat to the guest OS,"

in *EUROSEC*, Salzburg, Österreich, 2011.

[6] Oracle, „What Is ZFS?," [Online]. Available: https://docs.oracle.com/cd/E19253-01/819 5461/zfsover-2/index.html. [Zugriff am 15 02 2019].

[7] „btrfs Wiki," [Online]. Available: https://btrfs.wiki.kernel.org/index.php/Main%5FPage. [Zugriff am 15 02 2019].

[8] Linux kernel source tree, „XFS has gained super CoW powers!," 13 10 2016. [Online].

Available: https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/commit/?id=35a891be96f1f8e12 27e6ad3ca827b8a08ce47ea. [Zugriff am 25 02 2019].

[9] A. Cunningham, „ZFS filesystem will be built into Ubuntu 16.04 LTS by default," arsTechnica, 18 2 2016. [Online]. Available: https://arstechnica.com/gadgets/2016/02/zfs-filesystem-will-bebuilt-into-ubuntu-16-04-lts-by-default. [Zugriff am 25 02 2018].

[1 G. Paoloni, „How to benchmark code execution times on Intel® IA-32 and IA-64 instruction set 0] architectures," 2010.

[1 P. Landau, „Full-system encryption needs to be supported out-of-the-box including /boot and 1] should not delete other installed systems," Launchpad, 25 5 2018. [Online]. Available: https://bugs.launchpad.net/ubuntu/+source/ubiquity/+bug/1773457. [Zugriff am 27 02 2019].

[1 P. Kogan, „Full disk encryption with LUKS (including /boot)," 23 5 2014. [Online]. Available: 2] https://www.pavelkogan.com/2014/05/23/luks-full-disk-encryption/. [Zugriff am 27 02 2019].