Incident-Analyse und Forensik in Docker-Umgebungen

Andreas Dewald · Matthias Luft

ERNW Research GmbH

Zusammenfassung

In diesem Artikel beschreiben wir die Auswirkungen des vermehrten Einsatzes von *Docker* im Unternehmensumfeld auf forensische Untersuchungen und Analysen von Vorfällen. Auch wenn Docker einen immer höheren Verbreitungsgrad findet [CD16], wurden die Implikationen der veränderten Laufzeitumgebungen für forensische Prozesse und Werkzeuge noch kaum betrachtet. Wir beschreiben technologische Grundlagen von Docker-Containern und stellen basierend auf diesen dar, welche Änderungen sich in Bezug digitale Spuren und bisherige Methoden zu deren Erhebung ergeben. Konkret betrachten wir, welche Spuren innerhalb eines Docker Containers verglichen mit einer klassischen Virtuellen Maschine verloren gehen, bzw. in anderer Form gesichert werden müssen und, welche neuen Spuren und Möglichkeiten durch Docker selbst entstehen.

1 Einführung

Docker[1] ist eine noch immer vergleichsweise neue Technologie die im Bereich der Agilen Softwareentwicklung aufgrund des flexiblen und schnellen Deployment-Modells sowie der Möglichkeit zur schnellen Entwicklung basierend auf einem umfangreichen Ökosystem bereits einen hohen Verbreitungsgrad gefunden hat [Inc16]. Dieser hohe Verbreitungsgrad in Kombination mit einer hohen Adaptionsrate führt dazu, dass bereits jetzt viele Server als Docker-Hosts betrieben werden, auf denen eine Vielzahl von (Micro-) Services in sogenannten *Containern* laufen und damit die IT-Landschaft maßgeblich beeinflussen. Diese Beeinflussung wurde bereits aus verschiedenen Blickwinkeln der IT-Sicherheitssicht beleuchtet [GDT15, Doc16, CMDP16, CCK16]. Allerdings wurde bisher noch keine Betrachtung im Kontext von Incident-Analysen und forensischen Untersuchungen durchgeführt, die aufgrund der stetig wachsenden Anzahl an Containern zunehmend an Relevanz gewinnen.

1.1 Motivation

Dieser Artikel befasst sich mit der Fragestellung, was sich für forensische Analysen und Incident-Analysen durch den Einsatz von Docker-Containern verändert. Diese Fragestellung wird folgendermaßen unterteilt:

1. Was ändert sich in Bezug auf bekannte Methoden und Spuren?
2. Welche neuen Spuren kommen durch Docker selbst hinzu?

[1] Der Begriff *Docker* bezieht sich in diesem Artikel auf die Komponenten Docker Engine und containerd in Kombination. Auch wenn *Docker* korrekterweise für eine Summe an Open-Source-Projekten steht, entspricht die angesprochene Nutzung der allgemein gängigeren Terminologie.

P. Schartner · A. Baumann (Hrsg.) · D•A•CH Security 2017 · syssec (2017) 58-69.

Bisherige forensische Methoden fokussieren sich überwiegend auf physische oder virtuelle Maschinen, die sich in den eigentlichen Analyse-Schritten nicht grundlegend unterscheiden. Bisher wurde jedoch noch nicht betrachtet, wie typische Abläufe verändern werden müssen, um dem Einsatz von Docker-Containern Rechnung zu tragen. Dieser Artikel versucht dazu beizutragen, diese Lücke zu schließen.

1.2 Verwandte Arbeiten

Das Docker-Ökosystem und dessen Auswirkungen wurde bereits aus verschiedenen Blickwinkeln der IT-Sicherheit untersucht. So haben Gummarajul et al. [GDT15] öffentlich verfügbare Docker-Images auf bekannte Schwachstellen untersucht und relevante Schwachstellen in der überwiegenden Anzahl der Images festgestellt. Das Docker Security Team [Doc16] hat sich damit befasst, wie solche Schwachstellen zukünftig vermieden und entsprechende Maßnahmen in eine Enterprise Supply Chain integriert werden können. Verschiedene Arbeiten haben bereits die Fähigkeiten und Limitierungen von Docker zur Isolation von Prozessen diskutiert und bewertet [CMDP16, GG16]. Aus forensischer Perspektive wurde die Thematik bislang nur wenig [DW16] beleuchtet.

1.3 Übersicht

Der Rest dieses Artikels ist wie folgt strukturiert: In Abschnitt 2 erläutern wir die grundlegenden Techniken von Docker, die für die weitere Arbeit verwendet werden. Abschnitt 3 behandelt dann relevante Aspekte von Docker im Kontext forensischer Untersuchungen. Abschnitt 4 fasst den Inhalt dieser Arbeit zusammen, beschreibt ein Fazit und gibt einen Ausblick auf offene Aspekte und Problemstellungen für zukünftige Arbeiten.

2 Technologische Grundlagen von Docker

Docker ist eine Open-Source-Software für die Verwaltung und das Deployment von Software-Containern. Der Begriff Software-Container ist dabei nicht eindeutig definiert, es existiert jedoch ein gemeinsames Verständnis des Begriffs über Betriebssystemgrenzen hinweg. Dieses Verständnis definiert Software-Container als eine Laufzeit-Umgebung innerhalb eines Betriebssystems die Prozesse/Prozessgruppen bei Nutzung eines einzelnen gemeinsamen Kernels voneinander isoliert. Diese Isolation bezieht sich unter anderem auf die Trennung von Prozessräumen, Mechanismen zur Interprozesskommunikation, Netzwerknutzung, Dateisystemzugriff und Ressourcennutzung und -vergabe. Software-Container unter Linux, der ursprünglichen und noch immer am weitesten verbreiteten Plattform für Docker, basieren zur Umsetzung der Isolation auf den folgenden Features des Linux Kernels: cgroups, namespaces und Dateisystem-Treiber, die verschiedene *Layer* unterstützen. Docker implementiert dabei eine offene Spezifikation für Container der Open Container Initiative [Ope].

2.1 Container und Images

Im konkreten Docker Sprachgebrauch bezieht sich der Begriff *Container* dabei auf die Laufzeit-Instanziierung eines *Images*. Ein Docker Image enthält alle notwendigen Daten und Informationen um daraus eine Gruppe von Prozessen mit bestimmten Eigenschaften zu erstellen. So enthält das Image beispielsweise Informationen darüber welche Netzwerkports durch die enthaltenen Applikationen angeboten werden und welches Programm bei der Instanziierung ausgeführt werden muss. Das Image ist dabei – abgesehen vom Kernel-syscall-Interface und

Kernel-Devices – komplett unabhängig vom Host-System, auf dem das Image instanziiert wird. Sämtliche Entitäten in der Docker-Umgebung (wie Container und Images, aber auch Netzwerksegmente) werden durch einen eindeutigen Identifier referenziert (im Folgenden: ContainerID/ImageID). Die Erstellung dieses Identifiers findet dabei allerdings auf unterschiedlichen Wegen statt: Im Falle von Images stellt der Identifier einen Hash über bestimmte Teile des Images dar während ContainerIDs zufällig generiert werden.

2.2 cgroups

cgroups (kurz für control groups) stellen einen Mechanismus des Linux-Kernels zur Limitierung und Messung von Ressourcennutzung von Prozessgruppen dar. cgroups können über verschiedene Wege (wie bspw. direkten Zugriff auf ein Overlay-Filesystem in /proc oder abstrahierende Nutzerinterfaces dafür) gesteuert werden. Für den Kontext forensischer Analysen ist dabei lediglich relevant, dass Container einer oder mehrere cgroups zugeordnet sein können.

2.3 Namespaces

Der Linux-Kernel unterstützt aktuell die folgenden Namespaces: Mount, Process ID, Network, Interprocess Communication, UTS und User ID. Der Linux- Kernel erlaubt die Erstellung unterschiedlicher Namespaces, die in der Virtualisierung der jeweiligen Ressourcen resultieren. Beispielsweise können in unterschiedlichen Namespaces die gleichen Mount-Punkte für Dateisysteme verwendet werden: So kann sowohl innerhalb aller Container als auch auf dem Host-System der Mount-Punk */mnt* verwendet werden; analoge Verhaltensweisen ergeben sich für die anderen Namespaces.

2.4 Layered Filesystem

Ein *Layered Filesystem* baut auf einem Dateisystem-Treiber auf, der die Möglichkeit bietet, ein einzelnes Dateisystem aus verschiedenen Schichten aufzubauen und gegenüber einem Prozess einheitlich und abstrahiert darzustellen. Dieses Konzept ist in Abbildung 1 illustriert.

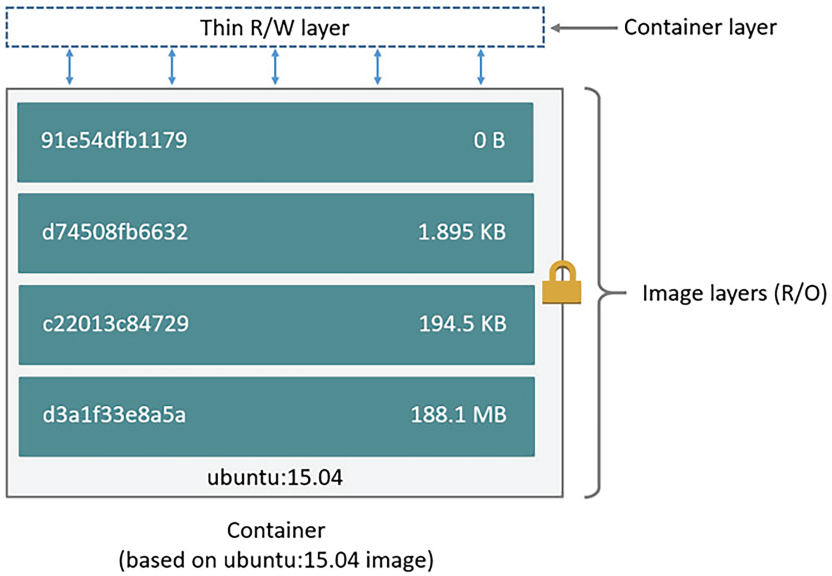

Abb. 1: Docker Illustration des Layered Filesystem Modells [Incd]

Ein Docker Image besteht aus einem oder mehreren Layern; einzelne Layer können aus unterschiedlichen Quellen stammen und von unterschiedlichen Personen oder Parteien erstellt/-bereitgestellt werden. Bei der Instanziierung eines Containers wird eine R/W-Schicht erstellt, die als oberstes Layer eingezogen wird. Sämtliche Schreibzugriffe innerhalb eines Containers werden nur in dieser Schicht ausgeführt; darunterliegende Schichten bleiben unverändert.

3 Forensische Analysen auf Docker-Hosts

Prinzipiell beginnt eine forensische Analyse auf einem Docker-Host genauso wie auf einem regulären System: Es wird ein Abbild der Festplatte und idealerweise des Hauptspeichers erstellt. Die Analyse der Abbilder kann allerdings unvollständige Ergebnisse liefern, wenn dabei nicht auf die Spezifika von (Docker-)Containern eingegangen wird. So würde eine typische Analyse von Festplatten- und Speicher-Abbild weiterhin eine Liste an Dateien und Prozessen ergeben, eine Zuordnung zu Containern oder gar die Information, ob bestimmte Dateien für eine Re-Konstruktion der tatsächlichen Dateisystem-Sicht des Live-Systems überhaupt relevant sind, würde allerdings nicht einfließen. Die folgenden Unterabschnitte stellen dabei verschiedene Aspekte dar, die bei der Analyse von Docker-Hosts beachtet werden müssen um eine umfassende forensische Analyse erstellen zu können.

3.1 Dateiwiederherstellung und Zurechenbarkeit

Docker-Container greifen über spezifische Dateisystemtreiber [Incc] auf Dateien zu. Das Dateisystem wird dabei nicht wie in traditionellen (virtualisierten) Betriebssystem-Umgebungen auf Blockdevices abgebildet, sondern basiert auf geschichteten Dateisystem-basierenden Strukturen. Diese Art des Dateisystem-Zugriffs ist in Abbildung 2 dargestellt und resultiert in verschiedenen Eigenschaften, die bei der forensischen Analyse beachtet werden müssen.

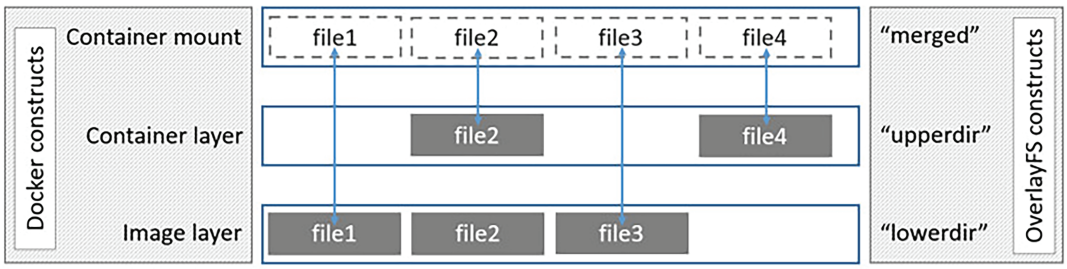

Abb. 2: Geschichtete Dateisysteme [Incb]

So wird eine Suche nach Dateien auf einem Festplatten-Abbild des Host-Systems auch Dateien aus Docker Images finden. Allerdings müssen auf einem Docker-Host die folgenden zusätzlichen Fragen beantwortet werden:

1. Durch welches Image wurde die Datei bereitgestellt?
2. Von welchen Containern wurde die gefundene Datei genutzt?
3. War die Datei auf Container-Ebene gelöscht? (Dieser Vorgang unterscheidet sich potentiell von Löschvorgängen auf regulären Dateisystemen)

Für die Beantwortung dieser Fragen ist dabei relevant, dass ein Image durch die Verwendung der R/W-Layer durch eine beliebige Anzahl Container genutzt werden kann. Eine entsprechen-

de Zuordnung ist über Laufzeit-Informationen sowie, falls vorhanden, bestimmte Konfigurationsdateien möglich.

Zur Laufzeit gibt der Befehl `docker ps` eine Liste aller gestarteten Container aus, `docker ps -a` eine Liste aller noch nicht terminierten Container. Die erste Zeile der Liste enthält den Beginn der ContainerID (im Folgenden ContainerIDShort), die komplette ContainerID kann über den Befehl `docker inspect ContainerID` ermittelt werden. Ist eine Laufzeit-Analyse nicht möglich, enthalten verschiedene Konfigurationsdateien Informationen über Container auf dem System. Das Standard-Verzeichnis für die Docker-Konfiguration ist `/var/lib/docker`. Die Container-Konfiguration ist in `/var/lib/docker/container/ContainerID` abgelegt. Container, die aktuell ausgeführt werden, lassen sich dabei über zwei Merkmale identifizieren: Zum einen enthält die Container-Konfiguration `config.v2.json` das Attribut `RRunning":true` und zum anderen sind die Linux-Dateisystemberechtigungen des Container-Unterverzeichnisses `shm` auf `1777` gesetzt. Weiterhin existiert ein dediziertes Verzeichnis für den R/W-Layer, das anzeigt ob ein Container in der Vergangenheit gestartet wurde (vgl. Abschnitt 3.2.2).

Weiterhin muss identifiziert werden, ob die Datei aus einem Container (d.h. dem R/W-Layer) oder einem Image stammt. Diese Information ist relevant für die Analyse der Sichtbarkeit der Datei zur Laufzeit. Falls eine Datei innerhalb eines Containers gelöscht wird, existieren zwei Möglichkeiten für die Löschung:

1. Die Datei stammt aus dem R/W-Layer: In diesem Fall wird die Datei mittels normaler Betriebssystem-Mechanismen gelöscht.
2. Die Datei stammt aus einem Image-Layer: In diesem Fall wird im R/W-Layer ein Löschungsverweis hinterlegt, die Datei bleibt im Image-Layer allerdings erhalten.

Dementsprechend müssen zur Wiederherstellung gelöschter Dateien unterschiedliche Methoden angewandt werden, welche wir in den folgenden Abschnitten jeweils im Detail erläutern.

3.2 Datei-Wiederherstellung aus dem R/W-Layer

Wird in einem Container eine Datei gelöscht, welche im R/W Layer des Containers gespeichert war, so wird diese Datei (zur Erinnerung: welche als reguläre Datei im Host-Dateisystem gespeichert ist) im Dateisystem des Hosts gelöscht. Welche Metadaten hierbei im Dateisystem erhalten bleiben und Möglichkeiten zur Wiederherstellung der Datei bieten, hängt vom konkreten Host-Dateisystem (wie zum Beispiel ext3, ext4, zfs, . . .) ab. Auf die jeweiligen Besonderheiten einzugehen würde den Rahmen dieses Artikels sprengen, jedoch sind dies auch keine Besonderheiten bezüglich Docker, sondern bilden die normalen Rahmenbedingungen für die forensische Dateisystem-Analyse und sind gut bekannt und dokumentiert [Car05].

Das bedeutet, dass die zur Verfügung stehenden Methoden zur Wiederherstellung einer gelöschten Datei aus dem R/W-Layer denen entsprechend, welche bei der forensischen Analyse einer physischen Festplatte, oder der virtuellen Festplatten-Datei (zum Beispiel im vmdk- oder vdi-Format) einer klassischen Virtuellen Maschine (VM) gleichen. Es bleibt lediglich anzumerken, dass bei einer klassischen VM die Möglichkeit besteht auch aus der VM heraus mit root-Rechten direkt das Festplatten-Gerät (wie z.B. `/dev/sda`) mit forensischen Software-Werkzeugen (Tools) zu analysieren. Diese Möglichkeit besteht aus einem Docker-Container heraus selbst mit root-Rechten nicht, da das Gerät nicht geöffnet werden kann (sofern der Container entgegen dem Standard-Verhalten mit erhöhten Privilegien gestartet wurde).

Grundsätzlich stehen zwei gängige Methoden zur Datei-Wiederherstellung zur Verfügung:

1. File Carving
2. Filesystem Analysis

Auf die Besonderheiten bei der Interpretation der Ergebnisse dieser beiden bereits bekannten Methoden im Docker-Kontext, gehen wir in den folgenden Abschnitten näher ein.

3.2.1 File Carving

Unter dem Begriff File Carving [PM09] versteht man das lineare Durchsuchen eines Datenträgers, eines Datenträgerabbilds oder einer Datei nach typischen Mustern (Magic Bytes) für den Beginn und/oder das Ende einer Datei, um hierdurch Dateien zu identifizieren. Da das Dateisystem hierfür nicht in Betracht gezogen wird, können mit diesem Ansatz sowohl allokierte als auch gelöschte noch nicht überschriebene Dateien wiederhergestellt werden. Allerdings können fragmentierte Dateien (bis auf wenige Ausnahmen von spezialisierten Carvern für bestimmte einzelne Dateitypen) nur unvollständig rekonstruiert werden. Weiterhin fehlen jegliche Meta-Informationen zu den Dateien, wie ihr Dateiname, Pfad, Zeitstempel oder ähnliches.

Aufgrund dieser Einschränkung kommt es bei der Anwendung in Bezug auf forensische Untersuchungen im Docker-Umfeld zu besonderen Problemen: Eine Zuordnung einer durch File Carving wiederhergestellten zuvor gelöschten Datei zu einem bestimmten Container oder auch nur die Unterscheidung, ob die Datei zu einem Docker-Container oder dem Hostsystem gehörte, ist nur über Kontextinformation möglich. Bei Dateien die durch Carving wiederhergestellt wurden fehlen jedoch, wie zuvor erörtert, solche Information in der Regel, so dass eine Zuordnung nicht mehr zuverlässig möglich ist. Lediglich, wenn der Inhalt der Datei selbst Aufschluss über deren Kontext gibt (beispielsweise, wenn sie eine ContainerID enthält), kann noch ein Bezug hergestellt werden.

Ein möglicher Ansatz, um zumindest groben Aufschluss über die Herkunft einer solchen Datei zu erhalten, ist es, alle durch das File Carving wiederhergestellten Dateien auszuschließen, die inhalts-identisch noch in allokierter Form im Dateisystem vorliegen oder durch die im nächsten Abschnitt erläuterte Methode unter Einbezug von Dateisysteminformationen wiederhergestellt werden können. Weiterhin kann dann für die verbleibenden bislang unbekannten gecarvten Dateien überprüft werden, zu welcher Blockgruppe die einzelne Datenblöcke in denen die Datei gefunden wurde gehören. Da zumindest Ext-Dateisysteme neue Dateien, wenn möglich, in derselben Blockgruppe allokieren in der sich das Elternverzeichnis befindet, kann diese Information einen groben Kontext liefern. Jedoch ist diese Information nur sehr grobgranular und zudem sehr vage, so dass sie allenfalls als hilfreich im Falle von Incident-Analysen eingestuft werden kann, jedoch für eine Verwendung in forensischen Untersuchungen zu vage sind.

3.2.2 Filesystem-Analyse

Im Gegensatz zum File Carving bedient sich die Filesystem-Analyse der im Dateisystem gespeicherten Verwaltungsstrukturen wie beispielsweise die MFT (Master File Table) im Falle von NTFS oder den Inode-Tabellen der Gruppendeskriptoren im Falle von ext-Dateisystemen. Je nach Dateisystem und Umstand lassen sich auch auf Basis dieser Informationen gelöschte Dateien wiederherstellen, wie von Brian Carrier [Car05] ausführlich beschrieben und in der Toolsammlung des Sleuthkit [Car07] implementiert. Auch wenn eine gelöschte Datei nicht immer mittels dieser Methode wiederhergestellt werden kann, bietet sie jedoch sofern sie im konkreten

Fall anwendbar ist doch einige Vorteile. Zum einen stellt eine Fragmentierung der gelöschten Datei kein Problem dar (im Gegensatz zum File Carving) und zum anderen lassen sich auch Metadaten wie Dateinamen, ganze Pfade und Zeitstempel wieder rekonstruieren. Durch diese Informationen (insbesondere Dateinamen und Pfade) ist es möglich, auch die Zugehörigkeit einer solchen Datei zum Host-System beziehungsweise zu einem Container (und zu welchem Container) feststellen. Diese Zuordnung erfolgt im Wesentlichen über Container/Image-IDs welche sich im Pfad einer zu einem Container gehörigen Datei finden. Hier unterscheiden wir zwischen den beiden von Docker vorwiegend eingesetzten Layer-Dateisystemen, dem älteren *AUFS* und dem in der aktuellen Version von Docker gängigen *Overlay2*, bei denen die Zuordnung leicht unterschiedlich erfolgt und auf die wir nun jeweils separat eingehen.

AUFS

In Abschnitt 3.1 haben wir bereits beschrieben, wie die ContainerID vorhandener Docker-Container ermittelt werden kann. Nun gehen wir von dem Fall aus, dass wir mittels der Dateisystem-Analyse eine gelöschte Datei eines Containers wiederherstellen konnten. Hierdurch ist auch der vollständige ursprüngliche Pfad der Datei bekannt. Wir erläutern nun, wie sich aus diesem Pfad schließen lässt, welchem Container die Datei zugeordnet war. Im Falle von AUFS befinden sich Dateien eines Containers unterhalb des Verzeichnisses `/var/lib/docker/aufs/`.

Nun kann sich die wiederhergestellte Datei in unterschiedlichen Layern befunden haben. Die einzelnen Layer finden sich in jeweils einzelnen Verzeichnissen unter ihrer AufsID: `/var/lib/docker/aufs/layers/$AufsID/`. Zunächst muss also aus dem Pfad der wiederhergestellten Datei auf die AufsID geschlossen werden. Diese kann einfach aus dem Pfad ausgelesen werden. Um von dieser AufsID auf die Zugehörigkeit zu einem Container zu schließen, muss für alle existierenden Container wie zuvor beschrieben die ContainerID und für jeden Container die Datei `/var/lib/docker/image/aufs/layerdb/mounts/$ContainerID/mount-id` ausgelesen werden. Die dort gespeicherte ID kann dann mit der AufsID verglichen werden. Bei einer Übereinstimmung hat man den Container der die Datei ursprünglich beinhaltete erfolgreich identifiziert.

Overlay2

Im Vergleich zu AUFS verwendet Overlay2 eine weitere ID, die sogenannte MountID. Die MountID eines Containers ist in der Container-Konfiguration `/var/lib/docker/containers/ContainerID/config.v2.json` enthalten und identifiziert den R/W-Layer des Containers, der in folgendem Ordner abgelegt ist: `/var/lib/docker/overlay2/MountID`. Der Ordner `/var/lib/docker/overlay2/MountID-init` wird dabei beim initialen Start des Containers angelegt und erlaubt dadurch eine schnelle Identifizierung aller R/W-Layer innerhalb von `/var/lib/docker/overlay2/`. Die tiefer liegenden Layer sind in der Datei `/var/lib/docker/overlay2/MountID/lower` in einer abgekürzten Form (der LayerIDShort) in absteigend Sortierung gespeichert. Die LayerIDShort wird als Name eines Symlinks im Ordner `/var/lib/docker/overlay2/l/` verwendet wobei der Symlink auf den Ordner `/var/lib/docker/overlay2/LayerID` zeigt. Weiterhin relevant für die forensische Analyse sind die Unterordner `diff` und `merged` innerhalb von `/var/lib/docker/containers/ContainerID/`. `diff` enthält dabei alle Dateien die im R/W-Layer angelegt, aber noch nicht gelöscht, wurden, `merged` enthält die Sicht auf das gesamte Dateisystem über alle Layer hinweg, die von Overlay2 bereitgestellt wird. Wird eine Datei im R/W-Layer gelöscht, die

durch einen tieferliegenden Layer bereitgestellt wird, wird ein inode allokiert, der per Flag als Linux-Character-Device markiert wird, was Overlay2 anweist, diese Datei für die Gesamtsicht zu ignorieren. Dateien die im R/W-Layer erstellt wurden und dann wieder gelöscht werden, werden mit normalen Betriebssystemmitteln gelöscht.

Abschnitt A enthält ein Skript, das alle relevanten IDs zu einer ContainerIDShort bei Nutzung von Overlay2 ausgibt.

3.3 Datei-Wiederherstellung aus einem Image-Layer

In diesem Fall wird im R/W-Layer wie in Abbildung 3 ein Löschungsverweis hinterlegt, die Datei bleibt im Image-Layer allerdings erhalten.

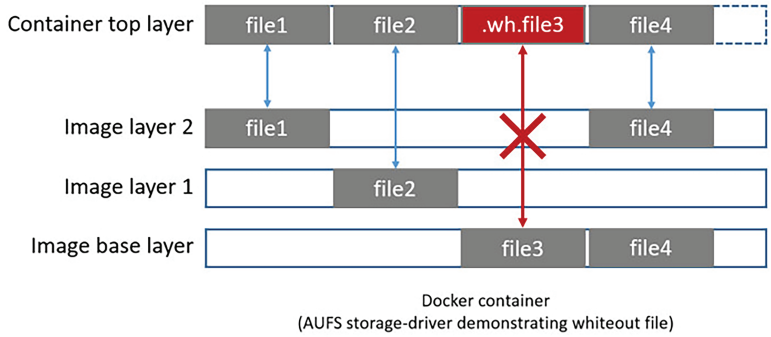

Abb. 3: Löschvorgang in LayerFS [Inca]

Overlay2 allokiert dabei einen inode im R/W-Layer, der den Namen der gelöschten Datei trägt und per Dateisystem-Flag als Character-Device markiert wird. Somit lassen sich alle Dateien die in einem tieferliegenden Layer gelöscht wurden durch folgenden Befehl identifizieren: `find /var/lib/docker/overlay2/ContainerID/diff -type c`. Die Datei-Wiederherstellung ist dann mittels einer Iteration durch die tieferliegenden Layer sowie Prüfung, ob die entsprechende Datei vorhanden ist, möglich.

Wir möchten anmerken, dass das hier beschriebene Vorgehen analog bei der post-mortem-Analyse des gesamten Host-Systems funktioniert. Auch hier können entsprechende inodes identifiziert werden und ggf. die zugehörigen Dateien aus dem Verzeichnis der die Originaldatei beinhaltenden darunterliegenden Schicht extrahiert werden.

3.4 Namespaces

Linux Namespaces resultieren in verschiedenen Auswirkungen auf die forensische Analyse. Der UTS-Namespace erlaubt die Konfiguration unterschiedlicher Zeitzonen pro Container. Die potentiellen Zeitdifferenzen müssen dabei bei der Live-Analyse von Containern beachtet werden, wirken sich allerdings nicht auf eine post-mortem-Analyse aus. Overlay2 verwendet bei der Veränderung von Dateien immer die Zeit des Host-Systems und passt die entsprechenden Zeitstempel für den jeweiligen Container dynamisch zur Laufzeit an.

Der PID-Namespace kann dazu führen, dass ein Prozess auf dem Host und ein Prozess in einem (oder mehreren) Containern die selbe Prozess-ID (PID) erhalten können. Dabei sind die PIDs auf dem Host immer eindeutig, lediglich die PID innerhalb des Containers kann identisch zu einer PID auf dem Host angezeigt werden. Diese Tatsache wird dann relevant, wenn Log-Dateien

PIDs enthalten und zur post-mortem-Analyse herangezogen werden sollen. Eine Übersetzung von Container-PID zu Host-PID ist ohne Laufzeitinformation nicht möglich.

User-Namespaces weisen ein ähnliches Verhalten zu PID-Namespaces auf und erlauben es, eine Nutzer-IDs (UIDs) oder Gruppen-IDs (GIDs) aus einem Container auf eine andere UID bzw. GID auf dem Host abzubilden. Beispielsweise kann ein Prozess innerhalb eines Containers mit der UID 0 ausgeführt werden, der entsprechende Prozess auf dem Host läuft aber unter UID 65000. Analog zu PID-Namespaces führt dies zu Problemen bei der Analyse, wenn Log-Dateien UIDs enthalten, in diesem Fall ist aber eine Zuordnung von Container-UID/GID zu Host-UID/GID über die Dateien `/etc/subuid` und `/etc/subgid` möglich.

3.5 cgroups

cgroups haben den geringsten Einfluss auf die forensische Analyse und sind nicht ausschließlich relevant für die Analyse von Docker-Hosts. cgroups im Allgemeinen können sprechende Namen besitzen die als zusätzliche Spuren (wie z.B. für auf dem Host laufende Prozesse und deren Einsatzzweck) verwendet werden können. Dabei können cgroups dynamisch zur Laufzeit oder durch Hintergrunddienste und basierend auf Konfigurationsdateien erstellt werden. Zur Laufzeit können existierende cgroups über das virtuelle Kernel-Dateisystem sysfs in `/sys/fs/cgroup` identifiziert werden. Eine persistente Konfiguration von cgroups ist über die Datei `/etc/cgconfig.conf` möglich; die Datei enthält die cgroup-Namen und kann potentiell weitere Indizien im Rahmen einer Analyse liefern.

3.6 Container-Management

Die Verwaltungskomponenten von Docker (der sogenannte `containerd`) bieten die Möglichkeit des Netzwerkzugriffs an. Dabei exponiert `containerd` einen Netzwerkport, der Zugriff auf die gesamte Verwaltungsfunktionalität von Docker bietet. Der Zugriff auf diese Netzwerkschnittstelle erfordert dabei standardmäßig keine Authentifizierung und führt daher zu der Möglichkeit unautorisiert Container zu starten. Entsprechend sind in diesem Fall auch keine Aussagen darüber mehr möglich, welcher Nutzer potentiell für den Start eines bestimmten Containers verantwortlich war.

Die Netzwerk-Exponiertheit von `containerd` kann zur Laufzeit über den Kommandozeilenaufruf bestimmt werden, der den Parameter `-l` oder `--listen` enthält. Eine persistente Konfiguration ist abhängig vom Betriebssystem; typische Ort für die entsprechenden Konfigurationsdateien sind `/etc/default/docker` und `/etc/docker/daemon.json`.

4 Zusammenfassung und Ausblick

Dieser Artikel beschreibt die größten Auswirkungen der Nutzung von Docker-Container auf Incident-Analyse und forensische Prozesse. Im folgenden fassen wir die Arbeit kurz zusammen, benennen Einschränkungen, geben einen Ausblick auf zukünftige Arbeiten und schließen den Artikel mit einem Fazit.

4.1 Zusammenfassung

In Abschnitt 2 haben wir die grundlegenden Techniken von Docker erläutert bevor wir uns dann in Abschnitt 3 den Besonderheiten von Docker im Kontext forensischer Untersuchungen gewidmet haben. Konkret haben wir in Abschnitt 3.1 die Möglichkeiten, Schwierigkeiten

und Lösungsansätze zur Rekonstruktion gelöschter Dateien und ihre Zuordnung zu Docker-Containern betrachtet und sind dabei insbesondere auf die Unterschiede bezüglich der Schicht aus der eine Datei gelöscht wurde (Abschnitt 3.2 und 3.3), sowie den Unterschieden bei der Zuordnung zwischen AUFS und Overlay2 eingegangen. Auch Möglichkeiten und Grenzen der Dateiwiederherstellung durch File Carving haben wir diskutiert. Abschnitt 3.4 befasste sich mit aus forensischer Sicht relevanten Aspekten von Docker *Namespaces* und Abschnitt 3.5 behandelt analog die sogenannten *cgroups* und das Thema Container-Management (Abschnitt 3.6).

4.2 Einschränkungen und Zukünftige Arbeiten

Diese Arbeit versteht sich als Einstieg in das Thema Docker-Forensik und zeigt Besonderheiten bei der Forensik oder Incident-Analysen einer Docker-Umgebung im Vergleich mit physischen Hosts und klassischen Virtualisierungstechniken auf. Die Betrachtung von Artefakten, welche eigens durch Docker hervorgerufen werden, ist noch nicht erschöpfend und muss künftig im Detail weitergeführt werden. Bislang ebenfalls nicht betrachtet wurde das Thema Live-Forensik eines Docker-Containers und eines Docker-Hosts, ebenso wie Besonderheiten und notwendige Anpassungen bei der Hauptspeicheranalyse eines Docker-Hosts. Ein weiteres noch unbearbeitetes Thema ist die Untersuchung von Möglichkeiten der Rekonstruktion und Zuordnung von Artefakten aus vollständig gelöschten Containern, anstatt bislang einzelner gelöschter Dateien eines an sich noch bestehenden Containers. Besonderheiten von Docker Swarms müssen ebenfalls in Zukunft noch Beachtung finden. Für all diese Aspekte sind gegebenenfalls auch Anpassungen vorhandener forensischer Werkzeuge oder die Entwicklung spezialisierter Plugins für den Docker-Kontext beispielsweise für Hauptspeicheranalyse-Tools wie Volatility und Rekall notwendig.

4.3 Fazit

Mit diesem Artikel haben wir einen Überblick über Besonderheiten von Docker bei forensischen Untersuchungen beziehungsweise Vorfallsuntersuchungen gegeben und erörtert, wie sich insbesondere das Löschen von Dateien in Docker-Containern auf deren Rekonstruierbarkeit auswirkt. Wir haben hierbei die unterschiedlichen Fälle wie die Löschung einer Datei im R/W-Layer vs. einem darunterliegenden Layer, der Verwendung des älteren AUFS vs. dem aktuell von Docker standardmäßig eingesetzten Overlay2, sowie die Rekonstruktion von Daten durch File Carving vs. Filesystem-Analyse unterschieden und erläutert. Weiterhin wurden Probleme und Grenzen dieser Methoden diskutiert sowie Einschränkungen dieser Arbeit und ein Ausblick auf weitere zukünftig zu betrachtende Themen in diesem Themengebiet aufgezeigt.

A Skript Container-Information

Das folgende Listing enthält ein Skript das alle relevanten IDs eines Containers in Overlay2 ausgibt:

```
1 #!/usr/bin/env bash
2 set -e
3 if [ -e $1 ];
4 then
5       echo "Please provide container ID as argument."
6       exit
7 fi
8
```

```
short_id="$1"
docker_lib="/var/lib/docker"
docker_containers="$docker_lib/containers"
docker_overlay="$docker_lib/overlay2"

tmp_dir=$(echo $docker_containers/$short_id*)

if [ ! -d $tmp_dir ];
then
      echo "No container matched the provided container ID."
      exit
fi

long_id=$(basename $tmp_dir)

image_id=$(grep -Po 'Image":.*?[^\\]",' \
  $docker_containers/$long_id/config.v2.json | \
  grep sha256 | cut -d ":" -f "3" | cut -d '"' -f 1)
image=$(grep -Po 'Image":.*?[^\\]",' \
  $docker_containers/$long_id/config.v2.json | \
  grep -v sha256 | cut -d '"' -f "3")

mount_id=$(cat $docker_lib/image/overlay2/layerdb/mounts/$long_id/mount-
    id)
path_to_rw_layer="$docker_overlay/$mount_id/diff"
path_to_live_mount="$docker_overlay/$mount_id/merge"
# list is ordered from highest layer to lowest layer
layer_list=$(cat $docker_overlay/$mount_id/lower)

IFS=':' read -r -a layer_array <<< "$layer_list"
echo "========== Container $long_id ======================="
echo "|"
echo "| Image:      $image"
echo "| ImageID:    $image_id"
echo "| MountID:    $mount_id"
echo "| Container Config: $docker_containers/$long_id"
echo "|"
echo "|================================================="
echo "| Layers:"

for index in "${!layer_array[@]}"; do
      if [ $index -eq 0 ];
      then
            continue
      fi
      ll=$(readlink $docker_overlay/${layer_array[$index]})
      layer=$(echo $ll | cut -d '/' -f 2)
      echo "| $docker_overlay/$layer"
      echo "|-------------------------------------------------"
done
```

Literatur

[Car05] Brian Carrier. *File System Forensic Analysis*. Pearson Education, 2005.

[Car07] Brian Carrier. The sleuth kit. http://www.sleuthkit.org/sleuthkit/desc.php, 2007.

[CCK16] Jeeva Chelladhurai, Pethuru Raj Chelliah, and Sathish Alampalayam Kumar. Securing docker containers from denial of service (dos) attacks. In *Services Computing (SCC), 2016 IEEE International Conference on*, pages 856–859. IEEE, 2016.

[CD16] ClusterHQ and DevOps.com. Container market adotpion survey 2016. https://clusterhq.com/assets/pdfs/state-of-container-usage-june-2016.pdf, 2016. Accessed: 14.03.2017.

[CMDP16] Theo Combe, Antony Martin, and Roberto Di Pietro. To docker or not to docker: A security perspective. *IEEE Cloud Computing*, 3(5):54–62, 2016.

[Doc16] Docker Security Team. Securing the enterprise software supply chain using docker. https://blog.docker.com/2016/08/securing-enterprise-software-supply-chain-using-docker/, 2016. Accessed: 14.03.2017.

[DW16] Jiang Du and Sheng Wu. Dcff: a container forensics framework based on docker. In *2016 3rd International Conference on Materials Engineering, Manufacturing Technology and Control*. Atlantis Press, 2016.

[GDT15] Jayanth Gummarajul, Tarun Desikan, and Yoshio Turner. Over 30security vulnerabilities. https://www.banyanops.com/pdf/BanyanOps-AnalyzingDockerHub-WhitePaper.pdf, 2015. Accessed: 02.04.2017.

[GG16] Aaron Grattafiori and Aaron Grattafiori. Understanding and hardening linux containers. *Whitepaper, NCC Group*, 2016.

[Inca] Docker Inc. Docker and aufs in practice. https://docs.docker.com/engine/userguide/storagedriver/aufs-driver/#container-reads-and-writes-with-aufs. Date accessed: 02.04.2017.

[Incb] Docker Inc. Docker and overlayfs in practice. https://docs.docker.com/engine/userguide/storagedriver/overlayfs-driver/. Date accessed: 02.04.2017.

[Incc] Docker Inc. Docker storage drivers. https://docs.docker.com/engine/userguide/storagedriver/. Date accessed: 02.04.2017.

[Incd] Docker Inc. Understand images, containers, and storage drivers. https://docs.docker.com/engine/userguide/storagedriver/imagesandcontainers/#images-and-layers. Date accessed: 02.04.2017.

[Inc16] Datadog Inc. 8 surprising facts about real docker adoption. https://www.datadoghq.com/docker-adoption/, 2016. Accessed: 14.03.2017.

[Ope] Open Container Initiative. Open container initiative (oci). https://www.opencontainers.org/about. Date accessed: 02.04.2017.

[PM09] Anandabrata Pal and Nasir Memon. The evolution of file carving. *IEEE Signal Processing Magazine*, 26(2):59–71, 2009.