Bachelor Thesis

Bachelor's Programme in IT Forensics and Information Security, 180 credits

Docker forensics: Investigation and data recovery on containers

IT forensics, 15 credits

Halmstad 2020-06-03

Niklas Englund, Pontus Davidsson

HALMSTAD UNIVERSITY

Abstract

Container technology continuously grows in popularity, and the forensic area is less explored than other areas of research concerning containers. The aim of this thesis is, therefore, to explore Docker containers in a forensic investigation to test whether data can be recovered from deleted containers and how malicious processes can be detected in active containers. The results of the experiments show that, depending on which container is used, and how it is configured, data sometimes persists after the container is removed. Furthermore, file carving is tested and evaluated as a useful method of recovering lost files from deleted containers, should data not persist. Lastly, tests reveal that malicious processes running inside an active container can be detected by inspection from the host machine.

Glossary

- **OS**: Short for operating system, acts as an interface between the computer hardware and user.
- **Virtualization**: The process of creating a virtual version of something, such as virtual applications or servers.
- **VM**: Short for virtual machine, uses virtualization technology to create an emulation of a computer system.
- **File carving**: The process of reassembling computer files from data streams in the absence of metadata.
- **API**: Short for application programming interface, defines interactions between multiple software intermediaries.
- **Httpd**: The Apache HyperText Transfer Protocol server program.
- **MySQL**: Open-source relational database management system.
- **PID**: Short for process identifier, is a unique number that identifies a running process.

List of Figures

List of Tables

1. Introduction

The last few decades have seen a remarkable change in how data and applications are managed. Originally, applications and processes were divided each on its own physical server which in turn created a number of challenges. Because each running application had to be accommodated by a physical server, its resources were often inefficiently utilized. This led to problems such as high cost and poor scalability as each server was both expensive and took up space in the datacenter. [1]

With these challenges in consideration, a new technology emerged called virtualization, which allowed the creation of multiple virtual machines on each physical server running their own OS. This eliminates the issue of one application per server and, as a result, higher utilization of hardware resources can be achieved as well as fewer physical servers needed. In addition, deployment of a virtual machine is faster than conventional methods of installing a new server. [2]

The virtualization technology sees advances in resource utilization, cost, and scalability, however, it could be improved upon further, hence the arrival of containers. Containers are lightweight and they utilize resources more efficiently than virtual machines, allowing more applications to be run on a single physical server. This technology allows a single OS on a machine to host multiple isolated environments. These environments are tailored for a specific purpose rather than a full-fledged OS that uses up unnecessary resources, therefore they contain only the necessary software required for its purpose. [3]

With the increased popularity of containers [3], it becomes necessary for companies and other users to be able to perform forensic investigations on containers, should an incident happen or a crime is committed, that could potentially cause damage. The focus of this thesis is put on one of the most commonly used container software, called Docker, to increase the understanding of containers and how they can be investigated.

1.1 Purpose

Insufficient research has been done on container forensics. In this thesis, therefore, we aim to investigate how containers work and, *by using forensic tools*, examine different methods of retrieving data during forensic investigations on containers. Thus, forensic capabilities on containers are tested and evaluated. As containers rise in popularity and demand, the need for container forensics is increasing. The purpose is, therefore, to inform relevant users on how a select set of forensic methods can be used to extract valuable data from containers. Other related research discussed recovery methods of deleted files from active containers and container images. The same research also emphasizes locating the origin, i.e container, or image, of the files. This thesis, however, tests whether files can be recovered from deleted containers and evaluates how well the method used works. Furthermore, some focus is put on examining active containers to detect simulated malicious processes.

1.2 Related work

Research has been done on the security aspects of containers. The authors of [3] explore different security requirements in-depth to give a clear picture of different attack possibilities towards containers.

Container forensics, however, is less researched. The authors of [4] aim to explore in detail how Docker images work and show how data can be extracted from the images themselves. The author also writes about file carving on containers, while focusing on deleted files from active containers. Furthermore, the paper explains the difficulties that come with file recovery, for instance, the lack of metadata that stores information such as file location. This makes it hard to identify which containers the files belonged to.

The paper [5] analyzes different ways of network forensic investigations in Docker-based container environments. This is done by capturing network traffic in different container environment setups. Their evaluation concluded that a correct capture process depends on the type of infrastructure, as the packet capturing is easier implemented on single docker containers, whereas the complexity grows with the number of containers.

The authors of [6] explain Docker service principles and features while analyzing forensic methods and models in related cloud environments. The authors also propose a Docker API-based forensic solution that can extract data from running containers and send raw evidence data to a forensic data center.

2. Problem Statement

Containers are a relatively new technology that is addressing pre-existing issues such as resource utilization in servers. This performance increase makes containers a great alternative to VMs and thus sees a continuous increase in usage. Being able to understand how to perform container forensics and the types of data that can be extracted, both from live and postmortem investigations is important. This thesis, therefore, aims to examine containers and their underlying structure to determine how valuable information can be extracted. As a result, the following questions should be answered:

- Does data persist when a container is deleted?
    - If so, can this data be used to restore the container?
- If a container is deleted, can its files be recovered via file carving?
- Can malicious activity in a container be detected through examination of running processes from the host computer?

2.1 Problematization

Because containers are becoming more frequently used amongst people working in IT, several different container software have been developed. Thus, the scope of this thesis is limited to one of the most prominently used container software called Docker. As the focus is put on a single container software, the result should not necessarily reflect how all container software functions.

3. Methodology

To answer the thesis' main questions, necessary information about the topic is first gathered and experiments are then conducted to provide practical knowledge. This method of first gathering relevant information, then using that knowledge to design experiments in order to test and explore containers should yield sufficient data to achieve results that can be analyzed. To the best of our knowledge, no scientific research has specifically answered this thesis' questions, therefore performing experiments is a necessary way to gather data in order to answer them.

Three experiments will be conducted using Docker container software running in an Ubuntu Linux environment. The Linux terminal is used to navigate the underlying structure of the containers in order to map where different types of data are stored. The first experiment is designed to test whether Docker persists data that can be used to restore a container, once a container has been removed. In this experiment, two containers are created. The first being an Ubuntu container and the second a MySQL container. Data will be stored in each container before their removal and Docker's ability to persist data will then be examined.

The second experiment builds on the first by examining deleted containers and performing file recovery. In this experiment, an Ubuntu and MySQL container are used to store a random set of files. Several tests will then be performed in which the containers, and all their associated files, including their volumes, will be deleted. File carving is then used in an attempt to recover the deleted data stored in the containers. Before each test, a new container is created and given new data.

The third experiment is designed to analyze two active containers: MySQL and httpd, from the host machine in order to detect simulated malicious activity. As containers are created to perform one specific task only, care should be taken to ensure only intended processes are running.

3.1 Problematization

The method of gathering information prior to designing the experiments, i.e determining what information is relevant for the topic, proves a challenge due to the low amount of research done in the field. This information is, however, important in order to provide sufficient knowledge to understand the experiments and the results. The challenge of providing sufficient relevant information may lead to some useful information not being covered.

While docker can be configured to run on Windows or Mac OS, all experiments will be performed on a Linux distribution. The experiments could, therefore, differ in terms of execution and results if conducted on a different OS. Furthermore, containers can run different types of applications, however, only a few select applications are used in the experiments. The results may, therefore, not reflect how other applications work.

While the second experiment requires a higher amount of data for a more accurate evaluation, due to time constraints, the amount of testing will be limited to five tests per container. This limitation is mainly due to the relatively long test durations, as each test requires the creation of a forensic image which is then used for file carving. Although this experiment requires the most amount of data, the results should be sufficient to evaluate and draw conclusions from.

4. Theory

To understand the results and discussion of this thesis, important background information about Docker containers is necessary. This chapter aims to provide knowledge of what Docker containers are, how they function, and related tools that will be used to examine them.

4.1 Docker image

A docker image is a file that contains data and information required to create a group of processes with defined properties. A docker image is composed of read-only layers. Each layer corresponds to a command executed during the creation of an image. This makes the creation of an image highly customizable as pre-existing images can be adapted or built upon to serve a slightly different purpose. The Information an image contains ranges from which network ports should be available to which programs should run during the execution of the docker image. When executed, a well-isolated environment called a container is created based on the specifications of the image file. [7]

The docker images are easily shareable which makes it possible to create identical environments on different OSs. The benefit of being able to run a given container on any OS is that applications on different systems will not encounter any compatibility issues since they are running inside the isolated environment created from the docker image. [7]

4.2 Docker containers

Containers share many similarities with VMs, such as allowing different types of software to run in isolated environments, however they function differently. [8] A container is an isolated environment, more specifically, a read and writable layer created on top of the read-only layers of the image. This layer is often called the container layer and is located in the */var/lib/docker/overlay2* directory. All changes made to the running container, such as creating or deleting files, are written in the container layer. If the container is deleted, however, the writable layer is also removed while the underlying image remains unchanged. [9]

As containers involve bundling an application with all related files, libraries, and required dependencies. This makes containers lightweight as only the necessary files are added to the container, unlike a VM where an entire OS is deployed which is one of the biggest differences between containers and VMs. As seen in *figure 1*, several VMs can run on the same host with hypervisor technology. Containers, on the other hand, communicate directly with the host OS, which is both advantageous and disadvantageous for different reasons. The advantages are lower storage space usage and performance since the container does not have to load an entire OS. One disadvantage, however, is that sharing an OS with the host is a concern from a security perspective as this makes containers less isolated than a VM. [7, 9]

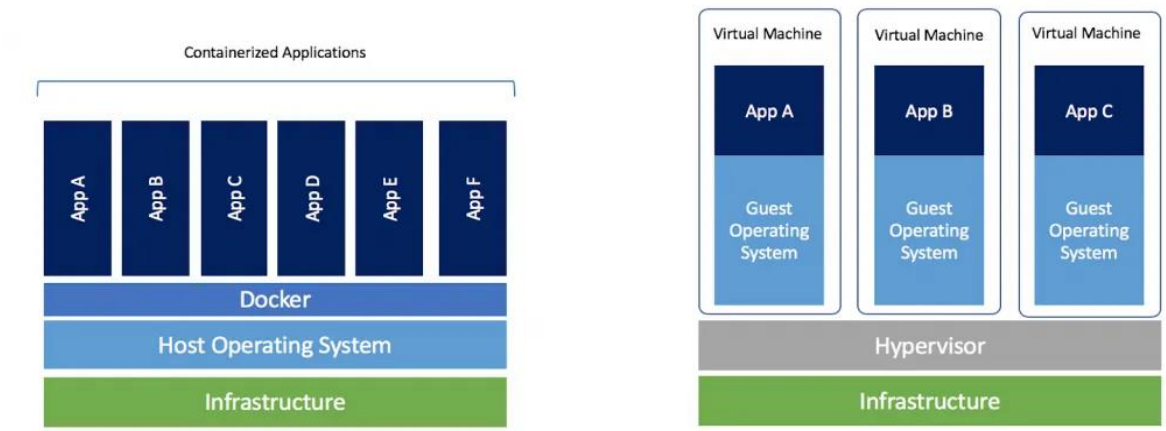

Figure 1: Comparison between container and virtual machine structure. [8]

4.3 Docker Volumes

The way that a container works by default is that all changes done in an active container are stored in the writable layer. As such, when a container is deleted this layer is deleted as well. This results in that the data contained in the writable layer does not persist when the container is removed. There are, *however*, ways to make the data persist even if the container is deleted. Two methods of achieving this are through bind mounts and volumes. [10]

Bind mounts work by mounting a file or directory on the host directly to a location inside the container. Even though bind mounts have limited functionality compared to volumes, there are several benefits of using them. File sharing, for example, between the host machine and container is relatively simple through the use of bind mounts. [10]

Volumes, *on the other hand*, are the preferred way to persist data from containers. Volumes are managed and created with the Docker software and are stored within a directory on the Docker host, which is the directory that is mounted into the container. Volumes are similar to bind mounts, the difference is that volumes are managed by Docker and are isolated from other core functionalities on the host machine. A volume can be mounted into several containers simultaneously, which makes sharing data between containers simple. Deleting a container will not erase the mounted volume as it needs to be specifically removed. Although great for persisting data, it could lead to unused volumes taking up storage space on the drive. [11]

Additional advantages volumes have over bind mounts are that they are easier to back up and migrate, as they work in both Windows and Linux containers. Volume drivers exist that enable storing volumes on remote hosts, for example in the cloud. It also allows encryption of the contents in a volume. Another advantage is that volumes do not increase the size of the container using it. [11]

4.4 Namespaces

A main purpose of containers is that they should be isolated from both each other and the host. This ensures it is possible to run a range of applications on a single host without them interfering with each other. To accomplish this, without resorting to using virtual machines, it is necessary to isolate the processes. Linux namespaces are therefore one of the most important parts that make the container technology work, as they allow for isolation of global system resources between processes. Without namespaces, a process running in one container could potentially interfere with a process running in another container. With namespaces, however, the processes in different containers are no longer aware of each other and thus are no longer able to disrupt each other. [7,12]

Most containers make use of seven namespaces, as seen in *table 1*, in order to provide the necessary isolation between containers. [13,14]

Namespace	**Description**
Mount	Isolates the filesystems mount points
UTS	Isolates hostnames and domain names
IPC	Isolates interprocess communication resources
PID	Isolates the PID number space
Network	Isolates network interface
User	Isolates UID/GID number spaces
Cgroup	Isolates croup root directory

Table 1: The seven namespaces used in Docker containers.

4.5 File carving

When a file is being removed, only metadata such as the indicator of where the file is located on the drive is being deleted. [15] This states that the storage space where the file is located is ready for new data to be stored. Thus, any deleted files can be recovered so long as the file's storage location has not been overwritten with new data. To recover deleted files, certain forensic techniques can be used. [16]

File carvers are software tools designed to restore data that was lost due to being removed or damaged. File carving tools analyze a drive for byte patterns in a data stream that matches the headers and footers of the selected file. If a match is found, all data in between the header and footer is considered part of the file. The file will then be written to a new location containing this data. This method does, however, require the header and footer to be clear, the file to not be fragmented, and the file to not be encrypted. [15] Files on a media are either stored in a contiguous (one piece) or fragmented way. If the files are fragmented, the task becomes more difficult as the files are split into multiple fragments, not necessarily located in the correct order. Fragmentation occurs when the filesystem must split a file into parts because there is not enough contiguous free space available on the media. It can also occur in other scenarios, for example, when several files are written to the media simultaneously. To carve fragmented files, additional, more complex steps, are required. [17]

While there are variations in ways to perform file carving, they all require the ability to identify the file header. File carving starts with scanning the media and creating a list of headers, indicating the beginning of the files. Different file carving options may be chosen depending on the file type. Shown below are some methods in how to carve for contiguous files. [17]

Header and fixed-size carving use a fixed size of data after the header. This means that the file size can be specified to ensure that the targeted files will be captured. This likely means, however, that excess data not belonging to the original file will be captured as well. This method, therefore, only works with file formats that tolerate extra data after the end of the file. [17]

Header and size carving use a similar approach where the size is derived from the file header instead of a fixed size. This, however, only works for file formats that contain the file size within the file header. Because the file size is determined by the file header, the produced file from this method will not contain any extra data at the end. [17]

Header and footer carving use the footer to determine the end of the file. The capture size will, therefore, be limited to the nearest footer after the header. This method also does not produce excess data at the end of the file. [17]

5. Experiment

5.1 Experiment 1

This experiment will be testing whether data persists once a container is removed and if this potential data can be used to restore the container. The individual parts of the experiment will be testing different containers and configurations. The first part of this experiment will be using an Ubuntu container with a jpg and pdf file stored in its home directory. Part two of this experiment is designed similarly to the first. An Ubuntu container and the same file types will be used, this time, however, a volume will be created together with the container. The third part of the experiment will be using a MySQL container consisting of a database. To see how the containers are created, see Appendix.

Part 1: Ubuntu

First, an Ubuntu container is created (see Appendix). The Ubuntu container is then removed with the command:

```
docker container rm "Container name"
```

Lastly, the Docker directories are examined to discern whether any data has persisted.

Part 2: Ubuntu with volume

First, a volume is created manually. Secondly, the Ubuntu container is created and the volume is tied to its */home* directory (see Appendix). The Ubuntu container is then removed with the command:

```
docker container rm "Container name"
```

To restore the data, the volume is mounted in a new container. This container should be created from the same image from which the original container was created. During the creation of the container, the volume is specified as the mount source, and the mount destination is selected as the mount target with the command:

```
docker run --name "Container name" -d -e MYSQL_ROOT_PASSWORD=123 --mount source="Volume name",target="mount destination" ubuntu
```

Part 3: MySQL

A MySQL container is created along with a database (see Appendix). Before removing the container, the corresponding volume and mount destination needs to be located. This is done with the following commands:

`docker inspect "Container name" | grep volume` *to find the volume*

```
"Type": "volume",
"Source": "/var/lib/docker/volumes/4f45aa0ed7d507a44d39ce6ee03149e502320a2c856d6e8d89219edd0cde1feb/_data",
```

Figure 2: Command output displaying volume tied to container.

`docker inspect "Container name" | grep Destination` *to locate the default mount destination*

```
"Destination": "/var/lib/mysql",
```

Figure 3: Command output displaying mount destination.

`docker container rm "Container name"` *to remove the container*

To restore the data, the volume is mounted in a new container. This container should be created from the same image from which the original container was created. During the creation of the container, the volume is specified as the mount source, and the mount destination is selected as the mount target with the command:

`docker run --name "Container name" -d -e MYSQL_ROOT_PASSWORD=123 --mount source="Volume name",target="Mount destination" mysql`

5.2 Experiment 2

This experiment will utilize file carving in an attempt to recover lost files from deleted containers. In contrast to the first experiment, the examination of the removed containers will take a more in-depth approach. The experiment will be divided into two parts, consisting of five tests each. The first part will be conducted on an Ubuntu container. During each test, an arbitrary set of files is stored on the container. Afterward, the containers are removed, and a forensic image of the host machine is captured, allowing file carving of the deleted files. The second part of the experiment will be conducted similarly, the difference being that a MySQL container is used instead.

Part 1: Ubuntu

The files used in the tests are displayed in the table below. To see how the container is created, and how the files are copied to the container, see Appendix.

Test	Files Used
1	Jpg, Pdf
2	Jpg, Png
3	Png, Pdf
4	Pdf
5	Jpg, Pdf, Html

Table 2: The files used in experiment 2, part 1.

After the container is created and set up, remove the container and its associated files with the command:

`docker container rm -v "Container name"`

Create a forensic image of the host machine with the command:

`dcfldd if="Partition" of="Output location of forensic image"`

Lastly, file carving is performed using a forensic software tool called Scalpel. The *scalpel.conf* file is modified by specifying what type of file it should locate. An example output of the *scalpel.conf* file can be seen in *figure 4*, which demonstrates how the configuration file can be modified to carve for a specific file type, in this case, a jpg file.

```
#               case    size    header                  footer
#extension   sensitive
#
#---------------------------------------------------------------------
# EXAMPLE WITH NO SUFFIX
#---------------------------------------------------------------------
#
# Here is an example of how to use the no extension option. Any files
# beginning with the string "FOREMOST" are carved and no file extensions
# are used. No footer is defined and the max carve size is 1000 bytes.
#
#      NONE     y      1000     FOREMOST
#
#---------------------------------------------------------------------
# GRAPHICS FILES
#---------------------------------------------------------------------
#       ibd     y       114688  \x00\x01\x38\x93\x00\x00\x00\x01
#
# AOL ART files
#       art     y       150000  \x4a\x47\x04\x0e        \xcf\xc7\xcb
#       art     y       150000  \x4a\x47\x03\x0e        \xd0\xcb\x00\x00
#
# GIF and JPG files (very common)
#       gif     y       5000000         \x47\x49\x46\x38\x37\x61        \x00\x3b
#       gif     y       5000000         \x47\x49\x46\x38\x39\x61        \x00\x3b
        jpg     y       5242880         \xff\xd8\xff???Exif             \xff\xd9        REVERSE
        jpg     y       5242880         \xff\xd8\xff???JFIF             \xff\xd9        REVERSE
```

Figure 4: File carving settings example.

Scalpel is then executed with the command:

`scalpel -o "Path to Output" "Path to forensic image"`

Part 2: MySQL

The files used in the tests are displayed in the table below. To see how the container is created, and how the files are copied to the container, see Appendix.

Test	Files Used
1	Ibd
2	Ibd
3	Png, Pdf
4	Pdf, jpg
5	Jpg, Pdf, Html, Wav

Table 3: The files used in experiment 2, part 2

After the container is created and set up, remove the container and its associated files with the command:

```
docker container rm -v "Container name"
```

Create a forensic image of the host machine with the command:

```
dcfldd if="Partition" of="Output location of forensic image"
```

Just as the first part, the *scalpel.conf* file is modified to carve for the specified test files. This time, however, Ibd files are used, which are not originally configured in the *scalpel.conf* file and must be added manually. As the different Ibd files may have variations in header values, it has to be specified for each Ibd file.

Scalpel is then executed with the command:

```
scalpel -o "Path to Output" "Path to forensic image"
```

5.3 Experiment 3

This experiment will test whether malicious activity in a container can be detected through an examination of running processes. This experiment will be conducted on two different containers, each incorporating three tests. During each test, a legitimate process related to the container will run, and an arbitrary set of processes is used to simulate malicious activity. The tests will first be conducted on a MySQL container, and afterward on an httpd container. To see the creation of the containers, see Appendix.

Test	**Malicious process**
1	Nmap
2	Bash script
3	Netstat

Table 4: The simulated malicious processes used in experiment 3.

First, start the malicious process. Continue by inspecting the container from the host and search for which PID is used in the container with the command:

```
docker inspect "Container name" | grep Pid
```

Find the parent PID, which is the main process for the container, with the command:

```
pstree -aps "PID"
```

Display all processes running inside the container with the command:

```
pstree -aps "Parent PID"
```

6. Results

The results of the experiments are presented individually for each part. The results from the first experiment are detailed in text format, contrary to the first experiment, however, the results from the second and third experiments are presented in tables, displaying the success rate of the tests carried out.

6.1 Experiment 1

The first experiment was designed to test whether data persists once a container is deleted. The experiment is divided into three parts, and yielded the following results.

Part 1: Ubuntu

When the Ubuntu container was deleted, the writeable layer, which records all changes made during an active container, was removed with it. During the examination of the Docker directories, no other information about the container was found, except the Ubuntu image itself, therefore, any files added to the active container did not persist in any way.

Part 2: Ubuntu with volume

In contrast to the first part where no volume existed, data persisted inside the manually created volume. As the volume was linked to the container's */home* directory, all files stored in this location were, therefore, preserved in the volume. The volume, when mounted with the new container, could be used to restore the container, along with its original files. However, the files could simply be extracted from the volume as well.

Part 3: MySQL

During the deletion of the MySQL container, not all data belonging to the container was deleted. In this case, when creating the container which encompasses the MySQL database, a volume that holds the container data was created automatically along with it. This volume remained intact and could, therefore, be used to create a new container, incorporating the same data. By inspecting the new container using a bash shell, it could be confirmed that the MySQL database survived the removal of the original container.

6.2 Experiment 2

The second experiment was divided into two parts, in which five tests were carried out in each. This experiment aimed to test if, and how well, file carving can be used to recover data from deleted containers.

Part 1

Table 5 displays the results from the tests carried out on the Ubuntu container; seven out of ten files were successfully recovered. The recovered files are displayed in *figure 5*.

Test	Recovered	Not recovered
1	Jpg, Pdf	-
2	-	Jpg, Png
3	Pdf	Png
4	Pdf	-
5	Jpg, Pdf, Html	-

Table 5: Results of experiment 2, part 1.

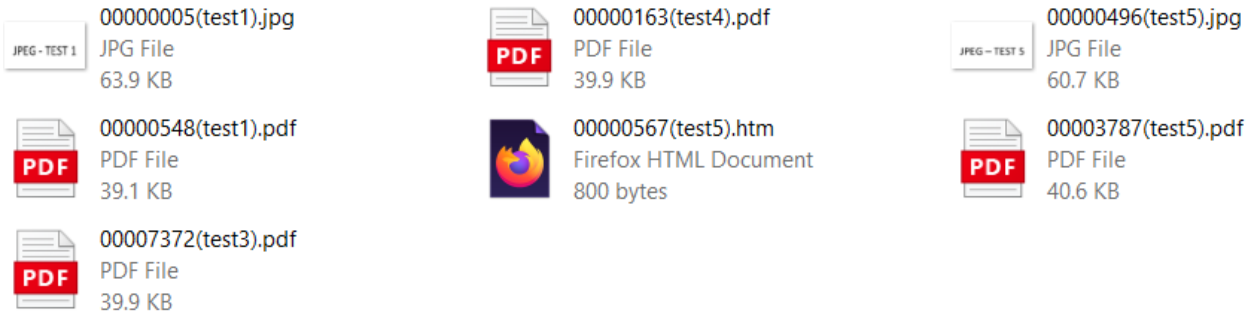

Figure 5: The files recovered in experiment 2, part 1.

Part 2

Table 6 displays the results from the tests carried out on the MySQL container. The recovered files are displayed in *figure 6*.

Test	Recovered	Partially recovered	Not recovered
1	Ibd	-	-
2	Ibd	-	-
3	Pdf	-	Png
4	Pdf, Jpg	-	-
5	Jpg, Pdf, Html	Wav	-

Table 6: Results of experiment 2, part 2.

In these five tests, eight out of ten files were successfully recovered, while one file (wav) was partially recovered. The carved wav file was smaller in size in comparison to the original Wav file, even though its full size was specified in the *scalpel.conf* file. While most of the file was recovered, some audio in the end was missing.

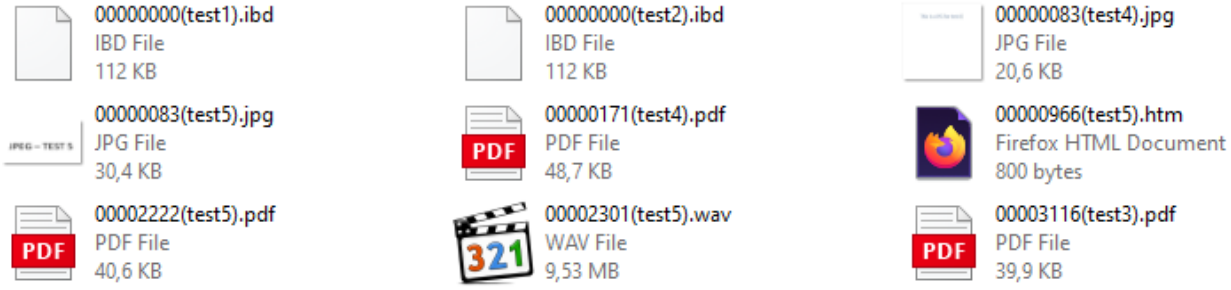

Figure 6: The files recovered in experiment 2, part 2.

Regarding the Ibd files, certain data related to the database could be extracted such as database and table name. However, the Ibd files were not enough to restore the database. Content of one of the Ibd files is displayed in *figure 7.*

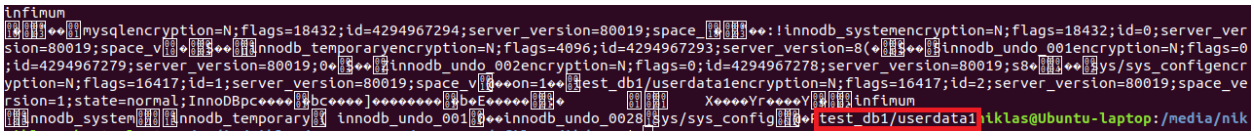

Figure 7: Database related information contained in an Ibd file

To summarize the results from both parts, the tests yielded varied results. The majority of the files, i.e 75% (15/20 files), however, were successfully recovered via file carving. This was confirmed by verifying that the file headers matched between the original and recovered files. Furthermore, both the contents and sizes of the recovered files were verified to be identical to the original files. However, metadata such as filenames and timestamps were not recovered.

6.3 Experiment 3

The third experiment was designed to test whether malicious activity in a container can be detected by examining the running processes on the host machine. Six tests were done, three on each container.

Table 7 displays the results from the tests carried out on both containers.

Test	Malicious process	Result
1	Nmap	Detected
2	Bash script	Detected
3	Netstat	Detected

Table 7: Results of experiment 3.

By examining the output from the *pstree* command during the first test, the malicious process *nmap* can be seen amongst the legitimate processes running in the two containers, as displayed in *figure 8* and *figure 9*.

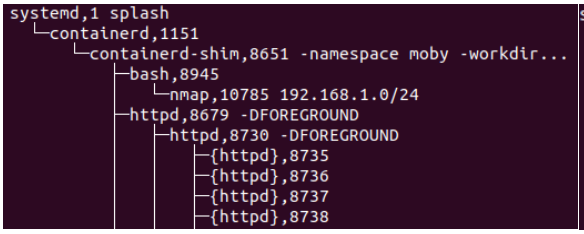

Figure 8: The malicious process Nmap detected in the httpd container.

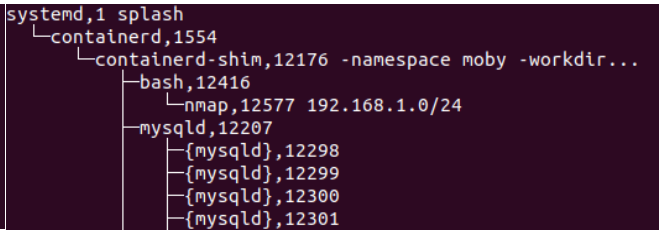

Figure 9: The malicious process Nmap detected in the MySQL container.

The bash script can be seen amongst the legitimate processes running in the two containers, as displayed in *figure 10* and *figure 11*.

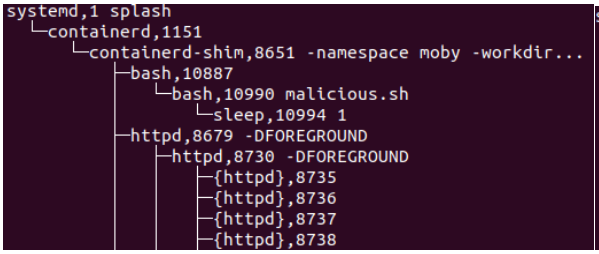

Figure 10: The malicious process running a bash script detected in the httpd container.

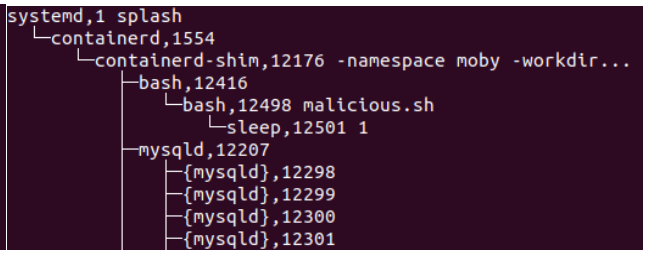

Figure 11: The malicious process running a bash script detected in the MySQL container.

The malicious process *netstat* can be seen amongst the legitimate processes running in the two containers, as displayed in *figure 12* and *figure 13*

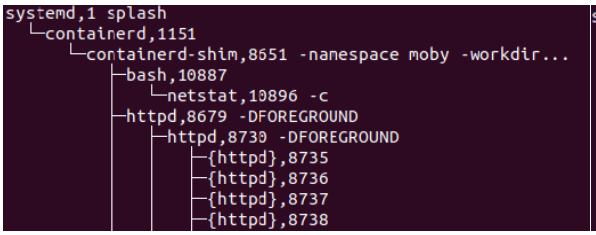

Figure 12: The malicious process Netstat detected in the httpd container.

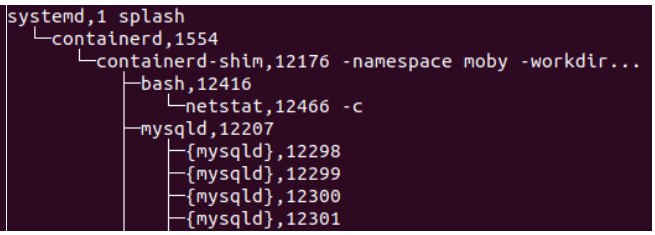

Figure 13: The malicious process Netstat detected in the MySQL container.

7. Discussion

The purpose of this thesis is to examine how, and evaluate how well, information can be retrieved from containers. The examinations encompass both active and removed containers to get a wider understanding of container forensics. The methods used, i.e gathering relevant information and performing experiments, were chosen to acquire both theoretical and practical knowledge about the topic. Possessing theoretical knowledge was important, as it laid the foundation of our thesis and aided in the design of the experiments. The practical knowledge, however, gained from examining and exploring containers and their underlying structure, facilitated the performing of the experiments, which in turn yielded sufficient results that could be evaluated in order to answer the thesis' questions.

The OS used whilst performing the experiments was the Linux distribution Ubuntu. While Windows was an optional OS, as it can run Docker software as well, Ubuntu was preferred because of its ability to run Docker software natively. It proved to be a well suited OS, as it is relatively user friendly while still encompassing the necessary features to successfully carry out the experiments. The Ubuntu OS was installed on a 25GB partition. The partition size was chosen to keep the image capture time feasible, while still having enough free space to not significantly increase the risk of file fragmentation.

Docker was chosen as the container software for this thesis, mainly because it is one of the most prominently used container software and, thus, is of comparatively high relevance amongst the different container software available. The open-source forensic tool Scalpel was used for file carving in our experiments. Scalpel was chosen because it is relatively well documented online and proved easy to use for our cause. Another user-friendly feature of Scalpel is that it comes with a predefined configuration file, containing carving settings for the most commonly used file types. This means that, for most of the chosen files for the experiments, Scalpel was easily configured to carve those file types.

To determine whether data persists through the deletion of Docker containers, the first experiment was designed to have different containers, with and without volumes, removed and examined. Performing this test, both on containers with volumes and containers without, then comparing the Docker directories for persisted data, showed the difference a volume has on container data persistence. Testing the different containers, it showed that one of the containers automatically created its volume, and thus, data persisted without manual creation of a volume. The tests also revealed that containers must have their volumes specifically removed, together with the container, to completely remove their data. In the case where a container creates its volume automatically, the user may be unaware of the volume, therefore, the volume may still be intact after the container has been removed. Furthermore, the test results showed that having a volume intact means that the container can be completely restored.

While the authors of [4] discussed file carving on deleted files in active containers, the second experiment in this thesis tests and evaluates file carving performance on deleted containers. The tests in the experiment were conducted on two different containers, with five tests per container. The higher amount of testing, in addition to several different file types being used for this experiment, was conducted to gather a higher quantity of data. This allows for a more accurate evaluation of whether file carving is possible, and how well it performs, on deleted containers. The experiment was designed to also test whether conducting file carving on different containers would have any effects on file carving performance. As the number of successful file recoveries was relatively even amongst the two containers, the results did not show any indication that container type should affect file carving performance. There was, however, a pattern in the unsuccessfully recovered file types. The Png file type was unable to be recovered in any of the attempts. There could be several different factors for this, for example, there could be a header or footer mismatch between the file and the Scalpel configuration file. Another potential factor is that the files were fragmented, i.e parts of the file stored in different locations, which has implications on file carving and requires more advanced steps to be able to, if at all possible, recover the files. Confirming that the recovered files were identical to the original files, except for metadata, was done by verifying that the header values of the recovered files matched the original files. Moreover, the file sizes were verified to be unchanged as well.

In contrast to the earlier experiments, the third experiment focuses on active containers. The experiment is designed to test whether malicious processes in containers can be detected by examining running processes on the host machine. The experiment was conducted on two different containers, with three tests per container. Each test incorporates a legitimate and simulated malicious process. The chosen containers were selected to recreate a scenario that resembles reality relatively well. Therefore, the MySQL and httpd containers were chosen because they run commonly used processes, i.e MySQL, and apache (httpd) processes. The gathered results showed that, during each test, the malicious processes were detected. While this method of detecting malicious processes is relatively simple, the experiment yielded valuable results. Because container processes are isolated from the host, if container processes are altered, for example by changing the process name, they should theoretically still be shown the same way on the host. Therefore, access to the host should be required to hide malicious activity from inspection by the host.

Concerning the limitations of the conducted experiments, a total of ten tests were performed on the second experiment. While this amount of testing yielded sufficient data to answer the thesis' question regarding file recovery, further testing could potentially have provided a more accurate evaluation. The amount of testing, however, was limited due to time constraints as each test took a relatively long time, even though small-sized partitions were used for the experiments. The relatively long test durations were mainly factored by the creation of forensic images, and the use of file carving techniques on said images, for every test. Regarding the first and third experiments, the lower amount of testing was chosen because the gathered data was enough to draw conclusions and answer the corresponding questions of the thesis.

8. Conclusion

Performing the experiments on different containers provided insight into their differences and similarities. An observed difference is automatic volume creation, as some container images, such as the MySQL image, are configured to create a volume along with the container. Other containers, for example the Ubuntu container, require manual creation of the volume.
Similarities include file carving performance on the different containers, where the results show a relatively equal success rate. Another similarity among the containers is that the simulated malicious processes were displayed in the same way on the host, regardless of the container used.

In comparison, a container forensic investigation does not differ much from an ordinary forensic investigation on a host machine. The same tools can be used to successfully extract data from containers. Knowledge about containers and their volumes, however, is necessary to successfully restore data from removed containers. As the results indicate, volumes can be used to fully restore a deleted container. Existing volumes should, therefore, be considered as a first step in recovering deleted container data in, for example, forensic investigations or due to unintentional removal of containers. However, as container volumes are not always automatically created, file carving may be necessary to recover deleted container data, should no volume exist. While volumes act as a backup, containing existing files, file carving attempts to recover deleted files. This, however, does not always succeed and is dependent on multiple factors. The relatively high success rate of the file carving performed in the experiments, however, indicates that it is a viable, and relatively effective, method of extracting data from removed containers.

The third experiment distances itself slightly from the other two experiments by investigating active containers in an attempt to detect simulated malicious activity, rather than examining deleted containers. All malicious processes were detected using process identifying commands on the host machine, indicating that this is a strong method that can be used to monitor active containers for malicious activity.

While the method used for examining malicious processes in active containers proved effective, the identified malicious processes were unaltered, making them relatively easy to detect. Further research in this area could, therefore, be to manipulate the process running in the container, testing whether this affects the detectability of those processes. The manipulation of the processes could potentially consist of a process name change, PID change, or process hiding. These examinations would then test whether the malicious process, inspected by the host, would be displayed differently, thus potentially making it harder to detect.

9. References

[1] S. Baca, 2017, “Virtualization For Newbies”, [Online]. Available: https://www.globalknowledge.com/us-en/resources/resource-library/articles/virtualization-for-newbies/ [Accessed on 2020-05-11]

[2] A. Agarwal, R. Luniya, M. Bhatnagar, M. Gaikwad and V. Inamdar, "Reviewing the World of Virtualization," 2012 Third International Conference on Intelligent Systems Modelling and Simulation, Kota Kinabalu, 2012, pp. 554-557, doi: 10.1109/ISMS.2012.44.

[3] S. Sultan, I. Ahmad and T. Dimitriou, "Container Security: Issues, Challenges, and the Road Ahead," in IEEE Access, vol. 7, pp. 52976-52996, 2019, doi: 10.1109/ACCESS.2019.2911732.

[4] A. Dewald, M. Luft and J. Suleder. 2018. “INCIDENT ANALYSIS AND FORENSICS IN DOCKER ENVIRONMENTS”. ERNW White Paper 64. Available: https://static.ernw.de/whitepaper/ERNW_Whitepaper64_IncidentForensicDocker_signed.pdf

[5] Daniel Spiekermann, Tobias Eggendorfer, and Jörg Keller. 2019. “A Study of Network Forensic Investigation in Docker Environments”. In Proceedings of the 14th International Conference on Availability, Reliability and Security (ARES ’19). Association for Computing Machinery, New York, NY, USA, Article 44, 1–7. DOI:https://doi.org/10.1145/3339252.3340505

[6] Jie Xiang and Long Chen. 2018. “A Method of Docker Container Forensics Based on API”. In Proceedings of the 2nd International Conference on Cryptography, Security and Privacy (ICCSP 2018). Association for Computing Machinery, New York, NY, USA, 159–164. DOI:https://doi.org/10.1145/3199478.3199506

[7] “Docker overview”, [Online]. Available: https://docs.docker.com/engine/docker-overview/ [Accessed on 2020-05-11]

[8] Jenny Fong, 2018, “Are Containers Replacing Virtual Machines?”, [Online], Available: https://www.docker.com/blog/containers-replacing-virtual-machines/ [Accessed on 2020-05-11]

[9] Bowen, 2017, “Docker Images, Containers, and Storage Drivers”. [Online]. Available: https://bowenli86.github.io/2017/01/10/tools/docker/Docker-Images-Containers-and-Storage-Drivers/ [Accessed on 2020-05-11]

[10] “Manage data in Docker”, [Online]. Available: https://docs.docker.com/storage/ [Accessed on 2020-05-11]

[11] “Use Volumes”, [Online]. Available: https://docs.docker.com/storage/volumes/ [Accessed on 2020-05-11]

[12] M. Ridwan, “Separation Anxiety: A Tutorial for Isolating Your System with Linux Namespaces:”, [Online]. Available: https://www.toptal.com/linux/separation-anxiety-isolating-your-system-with-linux-namespaces. [Accessed on 2020-05-11]

[13] E. King, 2016 “Linux Namespaces”, [Online]. Available: https://medium.com/@teddyking/linux-namespaces-850489d3ccf, [Accessed on 2020-05-11]

[14] Linux Programmer’s Manual, [Online]. Available: http://man7.org/linux/man-pages/man7/namespaces.7.html [Accessed on 2020-05-11]

[15] F. Benthin, “Recovering Deleted Files with Scalpel”, [Online]. Available: https://www.linux-magazine.com/Online/Features/Recovering-Deleted-Files-with-Scalpel
[Accessed on 2020-05-11]

[16] “Data Recovery Techniques On Linux”, [Online]. Available: https://www.howtoforge.com/data-recovery-techniques-on-linux [Accessed on 2020-05-11]

[17] Alexey V. Gubin, “File carving methods in data recovery”, [Online]. Available: https://www.klennet.com/carver/carving-methods.aspx [Accessed on 2020-05-11]

10. Appendix

Experiment 1

Create and set up Ubuntu container

- Download an Ubuntu image from Dockerhub with the command:
  `docker pull ubuntu`
- Create container with the command:
  `docker run --name "Container name" -dit ubuntu`
- Open a bash shell in the container with the command:
  `docker exec -it "Container name" bash`
- Copy files to the container with the command:
  `docker cp "File name" "Container ID":"New file location"`

Create and set up Ubuntu container with volume

- Create volume with the command:
  `docker volume create --name "Volume name"`
- Run container with the created volume using the command:
  `docker run --name "Container name" -dit -v "Volume name":/home ubuntu`
- Open a bash shell in the container with the command:
  `docker exec -it "Container name" bash`
- Copy files to the container with the command:
  `docker cp "File name" "Container ID":"New file location"`

Create and set up MySQL container

- Download a MySQL image from Dockerhub with the command:
  `docker pull mysql`

- Create container with the command:
  `docker run --name "Container name" -d -e MYSQL_ROOT_PASSWORD=123 mysql`

- Open a bash shell in the container with the command:
  `docker exec -it "Container name" bash`

- Log in to MySQL inside the container with the command:
  `mysql -uroot -p123`

- Create database with the command:
  `CREATE DATABASE "Database name";`

- Select the created database with the command:
  `use "Database name";`

- Create table with the command:

```
CREATE TABLE "Table name" (
user_id INT AUTO_INCREMENT PRIMARY KEY,
username VARCHAR(255) NOT NULL,
password VARCHAR(255) NOT NULL,
created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP);
```

- Insert data with the command:

```
INSERT INTO "Table name" (username, password) VALUES ('testuser',
'testpassword');
```

Experiment 2

Create and set up Ubuntu container

- Create container with the command:
  docker run --name *"Container name"* -dit ubuntu

- Open a bash shell in the container with the command:
  docker exec -it *"Container name"* bash

- Copy files to the container with the command:
  docker cp *"File name" "Container ID"*:"New file location"

Create and set up MySQL container

- Create container with the command:
  docker run --name *"Container name"* -d -e MYSQL_ROOT_PASSWORD=123 mysql

- Open a bash shell in the container with the command:
  docker exec -it *"Container name"* bash

- Log in to MySQL inside the container with the command:
  mysql -uroot -p123

- Create database with the command:
  CREATE DATABASE "Database name";

- Select the created database with the command:
  use "Database name";

- Create table with the command:
  ```
  CREATE TABLE "Table name" (
  user_id INT AUTO_INCREMENT PRIMARY KEY,
  username VARCHAR(255) NOT NULL,
  password VARCHAR(255) NOT NULL,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP );
  ```

- Insert data with the command:
  ```
  INSERT INTO "Table name" (username, password) VALUES ('testuser',
  'testpassword');
  ```

Experiment 3

Create and set up httpd container

- Download the container with the command:
  `docker pull httpd`
- Create container with the command:
  `docker run -dit --name "Container name" -p 8080:80 httpd`
- Open a bash shell in the container with the command:
  `docker exec -it "Container name" bash`
- Install the tools used to simulate malicious activity:
  `apt install nmap && apt install net-tools`

Create and set up MySQL container

- Create container with the command :
  `docker run --name "Container name" -d -e MYSQL_ROOT_PASSWORD=123 mysql`
- Open a bash shell in the container with the command:
  `docker exec -it "Container name" bash`
- Install the tools used to simulate malicious activity:
  `apt install nmap && apt install net-tools`

Pontus Davidsson

Niklas Englund

PO Box 823, SE-301 18 Halmstad
Phone: +35 46 16 71 00
E-mail: registrator@hh.se
www.hh.se