Docker API 를 이용한 컨테이너 아티팩트 자동 수집

방재훈*
(Ajou University, Department of Cyber Security)

Automated Collection of Container Artifacts Using Docker API

Jaehun Bang
(Ajou University, Department of Cyber Security)

요 약

가상화 기술은 여러 개의 가상 컴퓨터가 동일한 하드웨어를 공유하면서도 독립된 컴퓨터 시스템으로 동작할 수 있게 하는 기술이다. 이 기술은 하드웨어 가상화에서 서비스 가상화를 포함하는 컨테이너 기술까지 발전해왔다. 이러한 발전은 동일한 호스트를 여러 사용자가 컨테이너 기술을 통해 공유하는 클라우드 환경의 발전을 촉진하였다.

컨테이너 기술의 중요한 점은 하나의 호스트를 사용하더라도 각 컨테이너에 대한 소유권과 프라이버시가 다르다는 것이다. 이는 개인 PC 에서 특정 정보만을 압수하는 선별 압수를 지향하는 현재의 디지털 포렌식 방향성과 맞물려, 클라우드에서 아티팩트(artifact) 수집에 많은 어려움을 초래한다. 또한, PaaS 형태로 컨테이너를 제공하는 Docker 의 경우, 컨테이너가 동작 중일 때 관리자 규칙에 따라 컨테이너를 삭제하고 새로 만드는 경우가 많아 신속한 데이터 수집이 중요하다.

이러한 문제점을 해결하기 위해 본 연구에서는 원격에서 Docker API 를 통해 Docker 의 컨테이너, 이미지, 네트워크, 시스템 정보 등의 아티팩트를 자동으로 수집하는 프로그램을 개발하는 프로젝트를 진행하였다.

1. 서 론

클라우드는 동일한 호스트에서 가상화 기술을 통해 복수의 가상 컴퓨터(VM)를 구성한다. 가상화 기술은 물리적 하드웨어를 완전히 에뮬레이트하는 전가상화에서 특정 하드웨어를 에뮬레이트하는 반가상화를 거쳐, 서비스 운영에 필요한 환경만을 격리하여 운영하는 컨테이너 기술로 발전해왔다.

컨테이너 기술이 안정적으로 개발되면서 다양한 기업에서 이를 이용한 클라우드 플랫폼을 제공하고 있다. 클라우드 서비스는 컴퓨팅 자원을 제공하는 IaaS (Infrastructure as a Service) 예를 들어 Amazon 의 AWS, Naver 의 Naver Cloud Platform 과 같은 형태가 있다. 반면, 애플리케이션 및 서비스 운영에 필요한 자원을 제공하는 PaaS(Platform as a Service) 형태의 Docker 도 존재한다. 그 외에 서비스 이용에 필요한 자원을 제공하는 SaaS (Software as a Service)가 있다. 각 아키텍처는 제공 대상과 제공 범위에 따라 구분된다.

다양한 컨테이너 아키텍처의 등장으로 여러 상품이 판매되거나 제공되고 있으며, 이와 관련된 다양한 사고도 발생하고 있다. 컨테이너 기술은 낮은 비용과 높은 효율성으로 사용자에게 편리하고 신뢰할 수 있는 서비스를 제공하지만, 이를 불법적인 활동에 악용할 수도 있다. 혹은 정상적으로 제공되는 서비스에 침투하여 악성 행위를 할 수도 있다. 이러한 악성 행위를 규제하기 위해 클라우드 환경에서의 디지털 포렌식 작업이 중요하다.

많은 사용자에게 익숙한 Docker 에 대해 디지털 포렌식을 진행할 때 다양한 문제가 존재한다:

1. Docker 는 컨테이너 기술을 통해 가상화된 서비스를 제공한다. 즉, 하나의 호스트에서 다양한 컨테이너가 동작하는데, 증거를 수집하기 위해서는 용의 컨테이너를 특정해야 한다.
2. Docker 는 복수의 컨테이너에 대해 다양한 프로세스를 제공하며, Kubernetes(k8s)로 관리되는 경우가 많다. k8s 는 컨테이너 서비스에서 문제가 발생하거나 서비스를 업데이트하는 등의 이벤트가 발생하면 컨테이너를 삭제하고 재생성하는 경우가 많다. 즉, 사건 당시의 컨테이너와 수집 당시의 컨테이너가 동일하지 않을 수 있다.
3. 클라우드 서비스는 가용성이 꾸준히 유지되어야 한다. 따라서 서비스 운영 중 방해되지 않는 선에서 원시 데이터를 수집해야 하며, 이는 무결성 유지에 어려움을 야기한다.

따라서 Docker 에서 디지털 포렌식을 진행하기 위해서는 신속하고 정확하게 대상 컨테이너와 관련된 정보만을 수집할 수 있어야 한다. 이러한 요구사항을 충족하는 디지털 포렌식 도구를 개발하고자 본 연구를 진행하였다.

2. 컨테이너

컨테이너 기술이 가상화 기술의 최신 형태라는 점은 앞서 언급한 바 있다. 본 연구를 이해하기 위해서는 컨테이너의 정의, 가상화 대상, 그리고 제공하는 기능을 명확히 이해해야 한다.

컨테이너는 2000 년대 중반 리눅스의 LXC(Linux Container) 기술로부터 도입된 개념이다. LXC 는 단일 호스트에서 여러 독립적인 리눅스 커널을 컨테이너 형식으로 실행하기 위해 운영체제를 가상화하는 기술로 시작되었다.

과거에는 네트워크, 저장소, 운영체제 등 사용자마다 환경과 정책이 달라 동일한 프로그램에서도 서로 다른 오류가 발생하는 경우가 많았다. 이를 해결하기 위해 개발자들은 각 환경에 맞게 프로그램을 수정하는데 많은 노력을 기울여야 했다. 이러한 문제를 해결하고 프로그램의 이식성을 높이며 안정성을 보장하기 위해 도입된 개념이 바로 컨테이너이다.

컨테이너는 애플리케이션과 그 종속성을 함께 패키징하여 일관된 실행 환경을 제공한다. 이는 개발자들이 다양한 환경에서 일관된 방식으로 애플리케이션을 실행할 수 있게 하여, 개발 및 배포의 효율성을 크게 향상시킨다. 컨테이너는 또한 자원을 효율적으로 사용하며, 필요에 따라 신속하게 확장하거나 축소할 수 있는 유연성을 제공한다.

2-1. 컨테이너 특징

컨테이너는 애플리케이션과 서비스 동작에 필요한 환경 및 자원을 모듈화되고 격리된 컴퓨팅 환경으로 제공한다.

전가상화와 반가상화에서는 하이퍼바이저(Hypervisor)를 통해 게스트 운영체제(OS) 정보와 애플리케이션을 패키징하여 하드웨어 레벨의 가상화를 제공한다. 반면, 컨테이너는 실행에 게스트 OS 나 하이퍼바이저가 필요없는 OS 레벨의 가상화 구조를 갖추고 있어 기존 기술에 비해 훨씬 가볍다. 이는 단순히 가벼운 것 이상의 장점을 제공한다.

기존 가상화(VM)는 여러 서비스를 제공하기 위해 각기 다른 게스트 OS 를 실행하며, 하드웨어 가상화 등을 모두 구현해야 하기 때문에 오류가 많고 자원을 많이 소모한다. 반면, 컨테이너는 게스트 OS 와 하이퍼바이저가 없어 각 서비스별로 필요한 자원(라이브러리, 구성 파일 등)만을 포함하기에 가볍고 오류가 적으며, 자원 소모가 적다.

컨테이너는 서비스를 제공하거나 개발할 수 있는 환경을 구성하는데, 이를 사용하면 서비스나 애플리케이션에 대한 마이크로서비스 구현이 가능해진다.

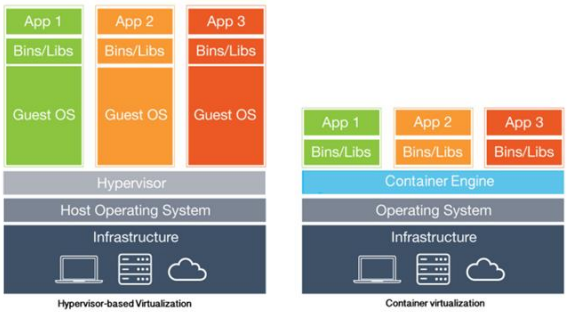

Figure 1. Hypervisor / Containers

3. Docker

Docker 사는 Docker 를 개발자가 앱 개발의 복잡성을 극복하고 아이디어를 실현할 수 있도록 돕는 PaaS(Platform as a Service) 서비스의 일종으로 설명한다. 즉, Docker 는 개발자가 개발에 필요한 환경 및 자원의 관리를 대신하여, 개발자가 오직 개발에만 집중할 수 있게 한다.

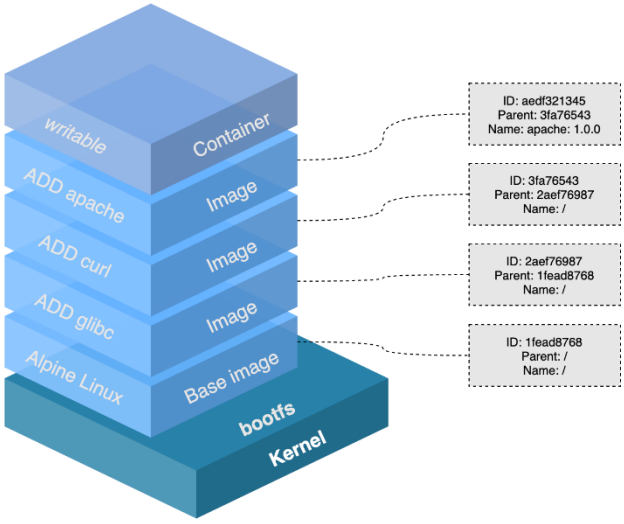

Figure 2. Docker Architecture

3-1. Docker Images

Docker 이미지는 애플리케이션 실행에 필요한 환경을 포함한 파일로서, 일종의 템플릿 역할을 한다. 기본적으로 실행에 필요한 다양한 자원 및 파일을 포함하며, 읽기 전용으로 변경이 불가능하다. 이는 가상머신에서의 스냅샷과 유사한 개념이다.

Docker 이미지는 Docker Hub 를 통해 공유되는 이미지

를 사용할 수도 있고, 개인 사용자가 만든 커스텀 이미지를 사용할 수도 있다. Docker Hub 에서는 이미지를 업로드하거나 다운로드할 수 있으며, GitHub 와 같이 이미지를 관리하고 공유할 수 있는 기능을 제공한다. 또한, 이미지에 대한 버전 관리도 지원된다.

Docker 이미지가 불변성을 갖는 이유는 이미지의 일관성을 유지하여 개발자가 동일한 조건에서 서비스를 테스트하고 개발할 수 있도록 하기 위함이다. 이는 문서 양식과 같이 템플릿으로서의 역할을 제공한다고 이해할 수 있다.

이러한 이미지를 기반으로 개발 환경을 구축하기 위해서는 Docker 컨테이너를 생성해야 한다.

3-2. Docker Containers

Docker 는 자원 분할 및 스케줄링의 기본 단위로 컨테이너를 사용한다. 컨테이너는 전체 소프트웨어 실행 환경을 패키징하여 분산 애플리케이션을 플랫폼에서 구축, 게시 및 실행할 수 있도록 설계되었다.

Docker 컨테이너는 애플리케이션에 필요한 자원과 환경을 포함하고 있어 가볍고 이식성이 뛰어나다. 이를 통해 애플리케이션을 빠르게 배포할 수 있다. 각 컨테이너는 OS 커널의 네임스페이스(Namespace)와 cgroups 등의 기술을 통해 자원이 격리되고 보호된다. 또한, 도커 데몬이 호스트 OS 와 통신하여 자원을 할당해준다. 그리고 컨테이너 자체를 메인 OS 와 다른 컨테이너 사이를 격리한다.

컨테이너는 이미지를 기반으로 생성된다. 이미지에는 특정 목적에 맞는 파일이 들어있는 파일 시스템과 격리된 시스템 자원 및 네트워크를 사용할 수 있는 독립된 공간이 포함된다. 컨테이너는 이미지를 기반으로 생성되는 과정에서 이미지를 변조하지 않는데, 이는 컨테이너가 레이어(Layer) 형태로 저장되기 때문이다.

Figure 2 를 보면, 호스트 커널 위에 여러 이미지가 존재하며, 그 위에 컨테이너가 존재한다. 여기서 쓰기 가능한(Writable) 대상은 컨테이너뿐이며, 컨테이너는 이미지와 달리 변경된 부분을 별도로 저장하여 이미지 자체에는 변화를 초래하지 않는다. 다만, 이미지가 여러 개 존재하는 것을 확인할 수 있는데, 컨테이너에 모든 변화 정보를 저장하고 있다면, 컨테이너에 문제가 발생하거나 패치 후 컨테이너를 삭제 및 재생성해야 할 때 문제가 생길 수 있다. 이를 해결하기 위해 베이스 이미지(base image)에서 변경된 내용을 일종의 스냅샷 형식으로 커밋(commit)할 수 있다. 저장된 이미지는 베이스 이미지에서 분기된 하나의 브랜치로 이해할 수 있다.

패치 등의 이유로 컨테이너를 삭제 후 재생성할 때, 가장 최신 이미지를 기반으로 생성함으로써 손실을 최소화하고, 배포를 위한 버전 관리도 수행할 수 있다.

3-3. Docker API

Docker 는 개발자의 자유도를 높이기 위해 API 를 공개하고 있다. 이를 통해 애플리케이션 관리의 자동화를 개발하거나, Docker 백엔드, 라이브러리 및 Docker Hub 와 직접 통신할 수 있다.

Docker 호스트 서비스는 Docker 클라이언트와 Docker 백엔드로 구성된다. Docker 서비스는 백그라운드에서 제공되며, 소켓 방식을 통해 외부와 통신이 가능하다. 기본적으로 보안을 위해 로컬 통신만을 수신하지만, 로컬 클라이언트는 백그라운드 소켓과 직접 통신할 수 있다. 관련 API 를 통해 제어하거나 확인할 수 있는 정보의 종류는 아래 표와 같다.

함수	정보
컨테이너 목록	컨테이너 (비)활성 목록
컨테이너 정보	컨테이너 세부 정보
컨테이너 추출	컨테이너를 .tar 파일로 추출
컨테이너 프로세스 정보	컨테이너 내부에서 동작 중인 프로세스 정보
컨테이너 로그	컨테이너 동작 로그

Table 1. Containers API

함수	정보
이미지 목록	이미지 목록
이미지 히스토리	이미지 관련 로그
이미지 정보	이미지 세부 정보
이미지 추출	이미지를 .tar 파일로 추출

Table 2. Images API

함수	정보
네트워크 목록	네트워크 목록
네트워크 정보	네트워크 세부 정보

Table 3. Networks API

함수	정보
볼륨 목록	볼륨 목록
볼륨 정보	볼륨 세부 정보

Table 4. Volumes API

함수	정보
플러그인 목록	플러그인 목록
플러그인 정보	플러그인 세부 정보

Table 5. Plugins API

함수	정보
도커 데몬 정보	도커 데몬 환경 정보
도커 버전	컴포넌트, 도커 등의 버전 정보

도커 디스크 정보	이미지, 컨테이너, 볼륨 등의 사용 디스크 정보
도커 이벤트	실시간 도커 이벤트 정보

Table 6. System Info API

3-4. Docker Forensics

Docker를 사용하는 조직에서는 Docker를 관리하기 위해 Kubernetes(k8s)를 사용한다. 이를 통해 컨테이너에서 문제가 발생하거나 패치를 진행할 때, 컨테이너를 종료하고 최신 이미지로 재생성하는 작업을 수행한다. 그러나 관리자는 언제 이러한 리빌딩이 진행될지 파악하기 어려운 경우가 많다. 따라서 Docker 에서 아티팩트를 수집하기 위해서는 신속하게 수집하는 것이 중요하다.

문제는 컨테이너나 이미지 등 문제의 Docker 에 대해 아티팩트를 수집하는 과정에서 변조가 발생하면 일부 아티팩트가 지워질 수 있다는 점이다. 따라서 본 연구의 목적은 Docker를 운영 중인 조직에서 수사관이 접근할 수 있는 포트를 열어 원격으로 접근함으로써 신속한 아티팩트 수집이 가능하도록 하는 것이다.

4. 개발

개발 언어	버전
Python	3.11.4
Docker-py	7.1.0

Table 7. Develop Environment

Docker API 수집 프로그램(Docker API Collecting Program, 이하 DAC)은 Python 코드로 제작되었다. DAC 에서 가장 중요한 점은 데이터의 무결성이다. 원격으로 아티팩트를 가져오기 때문에 데이터가 중간에 변조될 가능성이 존재한다. 이는 추후 컨테이너 및 이미지 추출 과정에서 자세히 다룰 것이다.

DAC 는 수집 대상 Docker 에서 제공한 IP 와 포트 정보를 입력으로 사용한다. 입력된 주소를 바탕으로 dockerClient 를 생성하여 해당 Docker 에 대한 부분적 제어권을 확보한다.

4-1. 컨테이너

Docker 는 Client.Containers.list() 명령어를 통해 GET 형식으로 Docker 에 존재하는 모든 컨테이너 정보를 반환한다. 옵션을 통해 비활성 컨테이너의 정보도 수집할 수 있다.

수집된 컨테이너 목록의 각 요소는 컨테이너에 대한 객체로 동작한다. 해당 객체를 통해 각 컨테이너에 대한 세부 정보나 로그 등을 확인할 수 있다. 컨테이너 객체는 Docker 에 생성된 각 컨테이너별로 매칭되며, Client.Containers.get()을 통해 특정 컨테이너를 지정하여 호출할 수 있다. 컨테이너는 .logs() 명령어를 통해 생성 후 내부 로그를 확인할 수 있다.

```
[7m80;root@c239990d0e2f: ~[mroot@c239990d0e2f:~# vim A.txt
[m>0;root@c239990d0e2f: ~[mroot@c239990d0e2f:~# clear
0;root@c239990d0e2f: ~[mroot@c239990d0e2f:~# exit
exit
0;root@c239990d0e2f: /[mroot@c239990d0e2f:/# ls
bin   dev  home  lib64  mnt  proc  run   srv  tmp  var
boot  etc  lib   media  opt  root  sbin  sys  usr
0;root@c239990d0e2f: /[mroot@c239990d0e2f:/# exit
exit
0;root@c239990d0e2f: /[mroot@c239990d0e2f:/# ls
bin   dev  home  lib64  mnt  proc  run   srv  tmp  var
boot  etc  lib   media  opt  root  sbin  sys  usr
0;root@c239990d0e2f: /[mroot@c239990d0e2f:/# ps
   PID TTY          TIME CMD
     1 pts/0    00:00:00 bash
    17 pts/0    00:00:00 ps
0;root@c239990d0e2f: /[mroot@c239990d0e2f:/# clear
0;root@c239990d0e2f: /[mroot@c239990d0e2f:/# exit
exit
```

Figure 3. Container Logs

Figure 3 을 확인하면, 컨테이너 내부에서 실행한 명령어와 실행 결과가 로그로 저장되어 있음을 알 수 있다. 이를 통해 컨테이너 내에서 사용자의 행위를 분석할 수 있다. 또한, 컨테이너의 생성 시간, 사용 이미지, 컨테이너 이름, 사용 OS 등의 세부 정보를 확인할 수 있어 컨테이너에 대한 메타데이터를 얻을 수 있다.

```
Id: c239990d0e2f53f0a006fcafb0a883634c23577ce4740a98a549d2aac6580314
Created: 2024-06-13T09:59:06.571662698Z
Path: bash
Args: []
State: {'Status': 'exited', 'Running': False, 'Paused': False, 'Restarti
Image: sha256:52969f533a19705d218d1963d58a593beb844699b3464b5e1b5523ab4b
ResolvConfPath: /var/lib/docker/containers/c239990d0e2f53f0a006fcafb0a88
HostnamePath: /var/lib/docker/containers/c239990d0e2f53f0a006fcafb0a8836
HostsPath: /var/lib/docker/containers/c239990d0e2f53f0a006fcafb0a883634c
LogPath: /var/lib/docker/containers/c239990d0e2f53f0a006fcafb0a883634c23
Name: /adoring_clarke
RestartCount: 0
```

Figure 4. Container Info

컨테이너가 활성 상태라면 .top() 명령어를 통해 내부에서 동작 중인 프로세스 정보를 확인할 수 있다. 해당 프로세스 정보는 컨테이너가 비활성 상태가 되면 조회할 수 없으며, 컨테이너가 다시 활성 상태로 돌아가도 휘발되어 사라진다.

export() 명령어를 통해 특정 컨테이너를 tar 파일 형태로 추출할 수 있다. 이는 증거 수집에 중요한 부분이다. Docker 로컬 환경에서 c239990d0e2f 컨테이너를 export 한 후, DAC 를 통해 export 한 파일 간의 SHA-256 해시를 비교하면 동일한 것을 확인할 수 있다. 이를 통해 컨테이너의 무결성을 유지하면서 증거 수집이 가능함을 확인할 수 있다.

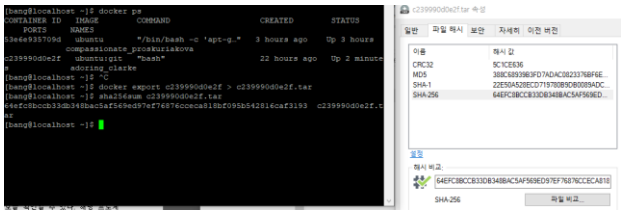

Figure 5. Export Hash

4-2. 이미지

Docker 클라이언트는 Images.list() 명령어를 통해 이미지 목록을 반환한다. 특정 이미지를 지정하려면 반환된 리스트에서 선택하거나, 특정 이미지 이름을 사용해 Images.get() 명령어로 이미지 객체를 생성할 수 있다.

생성된 이미지 객체는 .history() 명령어를 통해 이미지 사용 히스토리를 조회하거나, .attrs 속성을 통해 세부 정보를 조회할 수 있다. 이를 통해 이미지 생성 시간, 위치, 호스트명, OS 정보 등의 세부 정보를 수집할 수 있다.

```
Id : sha256:52969f533a19705d218d1963d58a593beb844699b3464b5e1b5523ab4bf0b787
RepoTags : ['ubuntu:git']
RepoDigests : []
Parent : sha256:f9a80a55f492e823bf5d51f1bd5f87ea3eed1cb31788686aa99a2fb61a27af6a
Comment :
Created : 2024-06-13T04:41:05.153988217Z
DockerVersion : 26.1.4
Author :
Config : {'Hostname': 'd7b53138c72d', 'Domainname': '', 'User': '', 'AttachStdin'
Architecture : amd64
```

Figure 6. Images Info

이미지를 확보하는 방법으로 save() 명령어를 사용하여 tar 파일 형태로 추출할 수 있다. 하지만, 이렇게 추출한 이미지 파일은 로컬에서 추출한 tar 이미지 파일과 서로 다른 SHA-256 해시값을 가지게 된다. 이는 무결성이 훼손된다는 것을 의미한다.

이미지 파일 자체에 손상이 있는 것은 아니지만, 수신되는 과정에서 수신 PC 인 Windows 의 파일명 정책에 의해 이름이 ubuntu:git 에서 ubuntu-git 으로 변경되면서 메타데이터의 변화가 발생한 것으로 예상된다.

4-3. 네트워크 & Etc…

Docker 에 연결된 네트워크 인터페이스 목록은 .networks.list() 명령어를 통해 수집할 수 있다. 수집된 인터페이스 객체를 통해 사용 드라이버, 생성 시간, 서브넷 등의 다양한 정보를 얻을 수 있다.

또한, Docker 볼륨과 플러그인 정보도 동일한 방식으로 수집할 수 있다. 사용 명령어에서 networks 부분을 plugins 또는 volumes 로 수정하여 해당 정보를 수집할 수 있다.

4-4. 시스템

Docker 에서는 다양한 시스템 정보를 수집할 수 있다. Client.info() 명령어를 통해 연결된 Docker Daemon 에 대한 정보를 수집할 수 있으며, 동작 중인 컨테이너 수, 이미지 수, 설정, OS 정보, Docker 루트 디렉토리 정보 등을 포함한다. 이를 통해 hostconfig.json 과 같은 API 로 수집하지 못한 아티팩트의 위치를 특정할 수 있다.

```
ID : 387d9d08-8b21-42c2-adcb-610803f1b5f4
Containers : 2
ContainersRunning : 2
ContainersPaused : 0
ContainersStopped : 0
Images : 4
Driver : overlay2
DriverStatus : [['Backing Filesystem', 'xfs'], ['
Plugins : {'Volume': ['local'], 'Network': ['brid
```

그림 7. Daemon system info

Client.version() 명령어를 통해 Docker 에서 사용하는 플랫폼, 컴포넌트, Docker 버전, 커널 버전 등의 정보를 수집할 수 있다.

```
Platform :
    Name : Docker Engine - Community

Components :
    Name : Engine
    Version : 26.1.4
    Details :
        ApiVersion : 1.45
        Arch : amd64
        BuildTime : 2024-06-05T11:31:02.000000000+00:00
        Experimental : false
        GitCommit : de5c9cf
        GoVersion : go1.21.11
        KernelVersion : 3.10.0-1160.119.1.el7.x86_64
        MinAPIVersion : 1.24
        Os : linux

    Name : containerd
    Version : 1.6.33
    Details :
        GitCommit : d2d58213f83a351ca8f528a95fbd145f5654e957
```

Figure 8. Version info

Client.df() 명령어는 현재 Docker 에서 사용 중인 디스크에 대한 정보를 반환한다. 단순히 사용량만을 반환하는 것이 아니라, 사용 중인 컨테이너, 이미지, 볼륨의 세부 정보도 출력하여 디스크 점유 상태를 확인할 수 있다.

마지막으로, Docker 에서는 이벤트가 실시간으로 생성되는데, 이를 Client.events() 명령어를 통해 실시간으로

수집할 수 있다. 이벤트 감시를 통해 Docker 포렌식 중에 발생하는 이벤트를 실시간으로 수집하여 위변조 시도를 감시할 수 있으며, 이벤트의 이상 패턴을 확인할 수 있다.

V. 결론

Docker 와 같은 컨테이너 기술은 현대 개발 및 배포 환경에서 필수불가결한 존재로 자리매김하고 있다. 이에 따라 컨테이너에 대한 디지털 포렌식의 중요성이 증가하고 있다. 본 논문에서는 Docker API 를 활용하여 Docker 의 아티팩트 자동 수집 방법론을 제시하고, 이를 통해 원격으로 컨테이너 기반 환경에서 증거를 수집하는 방법의 중요성과 한계점을 논의하였다. 이를 통해 다음과 같은 결론을 도출할 수 있었다.

1. 신속성

   컨테이너 기술의 특성상 컨테이너는 빠르게 생성되고 삭제될 수 있다. 이로 인해 증거의 변조나 손실 가능성이 높으며, Docker API 는 이러한 상황에서 즉각적인 수집이 가능하다는 강점을 갖는다.

2. 무결성

   원격으로 아티팩트를 수집하는 과정에서 아티팩트의 무결성은 무엇보다 중요하다. 컨테이너의 경우 다양한 테스트에서 무결성이 보존되는 것을 확인했지만, 이미지의 경우 무결성이 훼손되어 수집되는 사례가 있어 부분적인 성공으로 평가할 수 있다. 또한, 프로세스와 같은 휘발성 정보는 컨테이너가 정지 상태일 때 지워질 수 있으므로, 이러한 휘발성 정보를 우선적으로 확보해야 함을 확인할 수 있었다.

3. 수집 범위

   Docker API 는 Docker 에서 제공하는 공식 API 로, Docker 에서 IP 와 포트만 열어주면 상당한 범위의 정보를 수집할 수 있다. 이를 통해 컨테이너 환경에서 발생하는 보안 사건을 효과적으로 분석할 수 있다.

4. 가용성 보호

   - Docker 는 조직에서 서비스 배포 및 운영을 위해 사용된다. 특정 서비스에 대해 압수하거나 이미징을 진행하면 가용성이 일부 훼손될 수 있다. 반면 Docker API 는 컨테이너가 동작 중이더라도 해당 컨테이너와 관련된 정보를 빠르고 영향 없이 수집할 수 있다.

본 연구는 Docker 환경에서의 디지털 포렌식을 위한 기초를 제시하며, 이를 통해 향후 컨테이너 기반 클라우드 환경에서의 포렌식 연구에 기여하고자 한다. 향후 연구에서는 Docker 를 관리하는 Kubernetes(k8s)에서 API 나 로컬을 통해 아티팩트를 수집하고 이를 Docker 아티팩트와 함께 분석하여 자동화로 인한 증거 훼손을 파악하고, 이러한 패턴을 분석하여 수집 방법론에 대해 연구하고자 한다.

참고문헌

[1] Jie Xiang, Long Chen (2018), A Method of Docker Container Forensics Based on API, ICCSP

[2] Docker.docs, Develop with Docker Engine API, https://docs.docker.com/engine/api/