*Journal of The Korea Institute of Information Security & Cryptology* ISSN 1598-3986(Print)
VOL.33, NO.1, Feb. 2023 ISSN 2288-2715(Online)
https://doi.org/10.13089/JKIISC.2023.33.1.87

# 컨테이너 환경에서의 호스트 자원 고갈 공격 영향 분석*

이 준 희,[1†] 남 재 현,[2] 김 진 우[3‡]
[1,3]광운대학교 (학생, 교수), [2]단국대학교 (교수)

# Analysis of the Impact of Host Resource Exhaustion Attacks in a Container Environment*

Jun-hee Lee,[1†] Jae-hyun Nam,[2] Jin-woo Kim[3‡]
[1,3]Kwangwoon University (Student, Professor), [2]Dankook University (Professor)


## 요 약

컨테이너는 최근 주목받고 있는 서버 가상화 기술로, 기존 가상머신과 달리 더 가볍고 빠르게 독립 환경의 구축을 가능하게 한다. 이러한 장점으로 많은 기업들이 컨테이너를 활용하여 다양한 서비스들을 구축 및 배포하기 시작하였다. 하지만, 컨테이너가 도입 될수록 새로운 문제점 또한 노출하고 있는데, 특히 컨테이너 간 같은 커널을 공유하는 구조 때문에 발생하는 보안 취약점들이 지속적으로 발견되고 있다. 본 논문에서는 공격자가 컨테이너 환경의 구조적 취약점을 악용하여 할 수 있는 위협 중 호스트의 자원을 고갈시키는 공격, 이른바 호스트 자원 고갈 공격의 영향을 분석해보고자 한다. 특히, 가장 널리 사용되는 컨테이너 플랫폼인 도커를 이용해 구축한 컨테이너 환경에서 공격자가 CPU, 메모리, 디스크 공간, 프로세스 ID, 소켓 등의 주요 호스트 자원을 고갈 시켰을 때 발생하는 영향에 대해 분석하였다. 총 5가지 종류의 자원 고갈 공격 시나리오를 서로 다른 호스트 환경과 컨테이너 이미지에서 재현하였으며, 결과적으로 그 중 3가지의 공격이 효과적으로 다른 컨테이너를 서비스 불능을 만드는 것을 보였다.

## ABSTRACT

Containers are an emerging virtualization technology that can build an isolated environment more lightweight and faster than existing virtual machines. For that reason, many organizations have recently adopted them for their services. Yet, the container architecture has also exposed many security problems since all containers share the same OS kernel. In this work, we focus on the fact that an attacker can abuse host resources to make them unavailable to benign containers—also known as host resource exhaustion attacks. Then, we analyze the impact of host resource exhaustion attacks through real attack scenarios exhausting critical host resources, such as CPU, memory, disk space, process ID, and sockets in Docker, the most popular container platform. We propose five attack scenarios performed in several different host environments and container images. The result shows that three of them put other containers in denial of service.


**Keywords:** Cloud computing security, Container, Docker, Linux system security

## I. 서 론

컨테이너(Container)는 운영체제 (Operating System, OS) 수준의 서버 가상화 기술로 기존의 가상머신을 대체할 수 있는 차세대 가상 환경으로 많은 각광을 받고 있다. 특히, 데이터 센터와 같은 클라우드 환경에서 다양한 서비스들을 구축 및 배포하는데 널리 사용되고 있는데 이는 컨테이너의 많은

Received(11. 03. 2022), Modified(12. 12. 2022), Accepted(12. 24. 2022)
* 본 연구는 과학기술정보통신부 및 정보통신기획평가원의 SW 중심대학지원사업의 연구결과로 수행되었음(2017-0-00096).
† 주저자, hunroung@kw.ac.kr
‡ 교신저자, jinwookim@kw.ac.kr(Corresponding author)

장점 때문이다. 기존 하이퍼바이저 기반 가상화 기술의 경우 서버 자체를 가상화해야 하기 때문에 별도의 게스트 OS를 필요로 하였으며, 이로 인하여 가상머신 자체도 무거워졌다. 또한, 게스트 OS와 호스트 OS를 거쳐 CPU나 메모리와 같은 자원에 접근해야 하기 때문에 성능이 느려진다는 단점도 존재하였다. 하지만, 컨테이너의 경우 리눅스에서 제공하는 네임스페이스, 제어 그룹 등의 커널 기능 자체를 활용하여 별도의 OS 없이 가볍고 빠르게 애플리케이션들을 위한 독립 환경을 만들어준다. 그리고 이러한 장점에 주목하여 도커(Docker), 쿠버네티스 등 다양한 컨테이너 런타임 및 오케스트레이션 플랫폼이 등장하게 되었으며, 2022년 까지 75%의 글로벌 기업이 컨테이너 기반 클라우드 네이티브 시스템을 구축할 것으로 예측되고 있다[1].

하지만, 컨테이너라는 새로운 가상환경 구조는 전에 없던 또 다른 보안 취약점들을 노출시키게 되었다. 예를 들어, 공격자가 컨테이너에서 관리자 권한의 쉘을 탈취하는 권한 상승 공격을 성공했을 경우 해당 컨테이너를 실행하는 호스트 환경의 쉘 역시 제어할 수 있게 된다[2]. 기존 가상머신 환경에서는 독립된 OS 환경을 가지기 때문에 공격에 성공하여도 호스트 OS에 영향을 미치지 못했던 것과 대조적인 부분으로, 근본적으로 모든 컨테이너가 호스트 OS와 커널을 공유하는 점 때문에 이러한 공격이 가능하게 된 것이다. 그리고, 이러한 컨테이너 환경에서의 구조적 취약점은 실제 환경에서 공격자로 부터 악용되었을 때 컨테이너 환경 전체에 심각한 위협을 초래할 수 있다.

최근, 보안 학계에서는 이와 같은 심각성을 인지하여 컨테이너 환경의 취약점을 분석하고 잠재적인 공격 시나리오를 도출하고자 하였다. 예를 들어, 컨테이너 생성에 사용되는 제어 그룹, 커널 파라미터 등의 취약점을 악용하여 공격자의 컨테이너(악성 컨테이너)로 부터 호스트 자원을 고갈시키는 공격이 제안되었다 [5, 6]. 본 논문에서는 이러한 공격을 *호스트 자원 고갈 공격* 이라고 정의한다.

본 논문에서는 컨테이너 가상화에 사용되는 커널 기능들 외에 컨테이너와 호스트가 공유하는 자원들을 알아보고, 호스트 자원 고갈 공격이 미치는 영향을 분석하고자 한다. 그리고 이를 위해 기존 연구에서 제시된 호스트 자원 고갈 공격 시나리오들을 바탕으로 새로운 공격 시나리오 구성하고 실제 컨테이너 환경에서 재현해 봄으로써, 다양한 호스트 자원 고갈 공격이 일반 컨테이너나 호스트에 미치는 영향을 분석 하고자 한다.

## II. 배경지식

본 장에서는 본 논문을 이해하기 위해 필요한 컨테이너에 대한 기본적인 배경 지식을 설명한다.

### 2.1 컨테이너와 네임스페이스

컨테이너[1]는 기존의 하이퍼바이저 기반의 서버 가상화 기술을 대체하기 위한 새로운 운영체제 수준의 가상화 기술이다. 컨테이너는 기본적으로 리눅스에서 제공하는 네임스페이스, 제어 그룹 등의 커널 기능들을 이용해서 애플리케이션을 위한 독립 환경을 구축하는데, 이는 Fig. 1에서 볼 수 있듯이, 하이퍼바이저가 독립된 게스트 OS를 가지는 가상 머신을 생성하는 방식과는 대조적인 부분이다. 결과적으로 컨테이너는 하이퍼이저 기반의 가상 머신 환경에 비해 더 가볍고 빠르게 가상 환경을 구축하게 해준다. 그리고 대표적인 컨테이너 엔진으로 도커가 있다.

컨테이너는 앞서 언급하였듯이 리눅스 네임스페이스(namespace)를 이용해 가상 환경을 구성한다. 리눅스 네임스페이스는 프로세스 별로 프로세스 ID, 사용자 ID, 파일 시스템, 네트워크, 제어 그룹 등의 자원을 분할하는데 사용된다. 컨테이너 엔진은 사용자의 요구사항에 맞게 네임스페이스를 자동으로 생성, 컨테이너를 쉽게 구성하도록 한다.

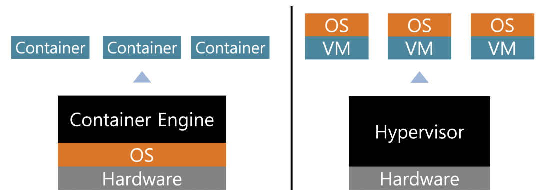


Fig. 1. The comparison of the container architecture (left) and the hypervisor architecture (right).

1) 컨테이너는 LXC (Linux Containers), 솔라리스 컨테이너, 도커(Docker) 등에서 사용하는 이름으로 오늘날 OS 레벨 가상화 기술을 칭하는 대표적인 명칭이 되었다.

### 2.2 제어 그룹

제어 그룹(Control Group, cgroup)은 리눅스 커널 기능으로 컨테이너의 CPU, 메모리, I/O 등 시스템 자원 이용을 제어하는데 사용된다. cgroup은 자원 종류에 따라 여러 개의 서브시스템으로 나뉘어져 있으며, 계층적인 형태(트리)로 구성된다. 그리고 사용자의 요구사항에 따라 여러 트리 형태의 cgroup을 구성할 수 있으며, 이를 컨테이너에 적용함으로써 컨테이너가 사용할 수 있는 자원을 제어하게 된다. 리눅스에서는 기본적으로 부모 프로세스의 cgroup이 자식 프로세스로 상속된다. 즉, 사용자가 컨테이너 엔진으로 별도의 설정을 전달하지 않는다면, 컨테이너가 생성될 때 해당 컨테이너는 호스트의 cgroup을 상속받게 된다.

### 2.3 커널 파라미터

커널 파라미터(kernel parameter)는 리눅스 커널의 시스템 변수들을 지칭한다. 리눅스 시스템의 최적화 또는 설정을 위해 사용되며, 최대 열수 있는 파일 개수 또는 TCP 소켓 개수 등을 예로 들 수 있다. 커널 파라미터는 커맨드 라인 명령어나 sysctl 툴, 또는 환경 설정 파일을 통해 지정할 수 있다. 도커에서는 컨테이너 생성시 --sysctl 명령어를 통해 컨테이너 별로 특정 커널 파라미터를 다르게 설정할 수도 있다. 제어 그룹처럼 사용자로부터의 별다른 설정이 없다면, 기본적으로 호스트의 커널 파라미터를 상속받는다 [9].

### 2.4 위협 모델 및 가정

본 논문에서는 공격자가 컨테이너를 사용하는 멀티 테넌트 클라우드 환경을 목표로 한다고 가정한다. 최근 많은 클라우드 관리자들은 테넌트들에게 가상 환경 제공을 위해 기존의 가상 머신처럼 컨테이너를 임대하는 방식을 채택하고 있다. 따라서 공격자가 이러한 클라우드 환경에서 소수의 컨테이너를 임대하고 이들을 악성 컨테이너로 활용하여 자원 고갈 공격을 수행한다고 가정한다. 이를 위해 공격자는 악성 컨테이너에서 공격 스크립트나 프로그램을 배포 및 실행할 수 있어야 하는데, 따라서 Ubuntu, CentOS 등의 범용 OS 컨테이너 이미지를 목표 컨테이너 환경에서 실행한다고 가정한다. 다만, 공격자가 다른 컨테이너 또는 호스트 OS에 접근하거나 추가적인 취약점을 알아내는 공격 시나리오는 본 논문의 목표에서 제외한다. 또한, 본 논문에서는 클라우드 관리자들은 컨테이너 플랫폼의 기본 자원 설정을 사용한다고 가정한다.

## III. 호스트 자원 고갈 공격

본 장에서는 호스트 자원 고갈 공격을 재현하기 위한 실험 환경과 각 케이스 별 공격 시나리오 및 결과를 보이도록 한다.

### 3.1 실험 환경

본 논문에서 제시하는 공격 시나리오들을 검증하기 위한 실험 환경으로 3가지 머신을 사용하였다.

- **Machine A** (Intel(R) Core(TM) i7-10700F 2.90GHz CPU, 16GB RAM, 512GB HDD)
- **Machine B** (Intel(R) Core(TM) i7-11800H 4.6GHz CPU, 32GB RAM, 2TB HDD)
- **Machine C** (Apple M2 CPU, 8GB RAM, 256GB SSD)

각 머신의 호스트 OS는 Ubuntu 20.04를 사용하였으며 리눅스 커널 버전은 5.13.0-52-generic x86_64이다. 컨테이너 플랫폼으로는 도커 20.10.17으로, 컨테이너에서 실행되는 이미지는 Ubuntu를 사용하였다. 이미지 버전별로 취약점이 상이한지를 파악하기 위해 Ubuntu 14.04 LTS, 16.04 LTS, 18.04 LTS, 20.04 LTS에서 공격 시나리오를 테스트하였다.

총 5가지 종류의 자원 고갈 공격 시나리오를 테스트 하였으며, 각각을 RA (Resource Attack)-* 기호로 표기하도록 한다.

### 3.2 CPU 및 메모리 고갈 공격(RA-1)

호스트 환경에서 컨테이너는 CPU, 메모리 등 자원을 공유하게 된다. 이때 도커의 기본 설정을 사용하게 되면 컨테이너가 호스트 자원에 대해 제한 없이 사용할 수 있게 된다. 이를 이용하여 하나의 컨테이너에서의 자원 과소비가 다른 컨테이너에 영향을 미치는지 알아보는 실험을 진행하였다. Fig. 2는 이러한 CPU 및 메모리 고갈 공격에 대한 시나리오이다.

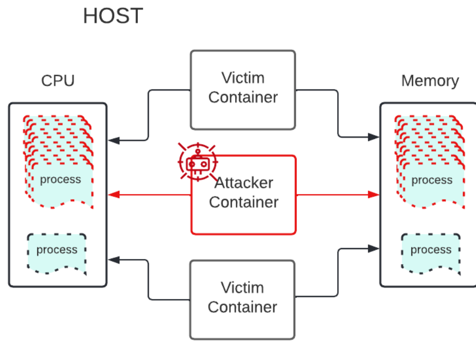


Fig. 2. The scenario of the CPU and memory exhaustion attack.

공격에 따른 영향을 정량적으로 평가하기 위해 성능 감소율(performance reduction rate)을 측정하였다. 이를 위해 먼저 피해자 컨테이너에서 메모리 동적 할당과 간단한 수학 연산을 진행한 후, 종료되는 프로세스를 5초 간격으로 실행하였다. 이때 각 컨테이너의 성능을 100%로하고 공격 시 연산 처리율이 얼마나 감소하는지를 측정, 성능 감소율을 모니터링 하였다. 공격자 컨테이너에서는 리눅스의 stress 툴을 사용하여 CPU, 메모리 사용률을 90% 이상을 유지하였다.

Fig. 3은 공격전과, 공격후의 연산량이 달라진 것을 보여주는 그림이다. 공격자 컨테이너가 간단한 연산을 사용했음에도 불구하고 성능이 크게 감소하였다. 따라서 실제 환경에서 이보다 더 복잡한 작업을 하는 경우에는 더 큰 오버헤드를 발생시킬 것으로 예상된다. Table 1은 CPU 및 메모리 자원 고갈 공격에 대한 실험 결과이며, 모든 도커 이미지와 머신에서 약 30%의 성능 감소율을 보이는 것을 관찰할 수 있었다.

Table 1. The performance reduction rate measured from different test machines and container images.

| Test Machine | Container Ubuntu Version | Performance Reduction Rate (%) |
|---|---|---|
| Machine A (Intel® i7 10800F) | 14.04 LTS | 30% |
| | 16.04 LTS | 30% |
| | 18.04 LTS | 30% |
| | 20.04 LTS | 30% |
| Machine B (Intel® i7 11800H) | 14.04 LTS | 30% |
| | 16.04 LTS | 30% |
| | 18.04 LTS | 30% |
| | 20.04 LTS | 30% |
| Machine C (Apple M2) | 14.04 LTS | 30% |
| | 16.04 LTS | 30% |
| | 18.04 LTS | 30% |
| | 20.04 LTS | 30% |

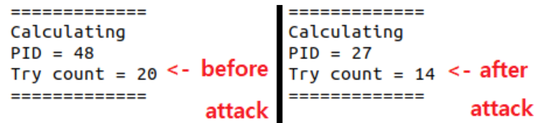


Fig. 3. The result of the CPU/memory exhaustion attack. Before (left) and after (right).

### 3.3 파일 시스템 자원 고갈 공격(RA-2)

일반적으로 컨테이너 실행 시 도커에서 제공하는 기본 설정을 사용하면 컨테이너가 cgroup의 CPU, 메모리, 저장공간 등에 자원에 대해 제한 없이 사용할 수 있게 된다. 이를 이용하여 만약 공격자 컨테이너에서 저장 공간을 과도하게 소비하게 되면 다른 컨테이너에 어떤 영향을 미치는지 알아보는 실험을 진행하였다. Fig. 4는 이러한 파일 시스템 자원 고갈 공격에 대한 시나리오다.

실험을 위해 공격자 컨테이너에서 truncate, fallocate, dd, head 등의 명령어를 이용하여 더미 데이터를 생성하였다. 이를 통해 공격자 컨테이너가 호스트의 저장 공간을 모두 사용 후, 피해자 컨테이너에서 간단한 프로세스 실행, 패키지 업데이트 및 파일 저장 등을 시도하여 정상적으로 동작하는지를 점검하였다.

Table 2는 실험에 대한 결과를 요약한 것이다.

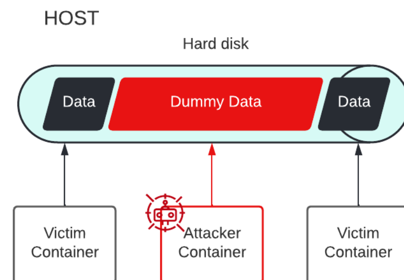


Fig. 4. The scenario of the file system resource exhaustion attack.

Table 2. The disk usage rate measured from different test machines.

| Test Machine | Total Disk Size | Disk Usage Rate Before Attack (%) | Disk Usage Rate After Attack (%) |
|---|---|---|---|
| Machine A (Intel® i7 10800F) | 512GB | 100% | 0% |
| Machine B (Intel® i7 11800H) | 2TB | 100% | 0% |
| Machine C (Apple M2) | 256GB | 100% | 0% |

공격 전후로 디스크 사용률(disk usage rate)을 측정하였는데, 호스트의 디스크 사용률이 0%가 될 때까지 공격자 컨테이너가 아무 제한 없이 더미 데이터를 생성할 수 있었다. 이로 인해 피해자 컨테이너에서는 패키지의 업데이트를 진행할 수 없을 뿐 아니라, 파일을 생성하는 그 어떤 행위도 할 수 없었다. 다만 파일 시스템을 사용하지 않는 프로세스의 경우 문제없이 사용가능하였는데, 파일의 생성 및 저장 시에는 오류를 반환하였다. Fig. 5는 파일 시스템 자원 고갈 공격 후 호스트 환경에서의 저장공간 상태를 보여준다.

```
new@new-Inspiron-3881:~$ df -h
Filesystem      Size  Used Avail Use% Mounted on
udev            7.7G     0  7.7G   0% /dev
tmpfs           1.6G  2.4M  1.6G   1% /run
/dev/nvme0n1p2  468G  468G     0 100% /
tmpfs           7.7G     0  7.7G   0% /dev/shm
tmpfs           5.0M  4.0K  5.0M   1% /run/lock
tmpfs           7.7G     0  7.7G   0% /sys/fs/cgroup
```

Fig. 5. The snapshot of the target file system after performing the attack.

### 3.4 프로세스 자원 고갈 공격(RA-3)

컨테이너는 운영체제를 이미지화 시킨 파일을 통해 설정 값들을 불러온다. 이때 커널 파라미터를 따로 설정해두지 않는다면 일부 설정에 대해 호스트의 커널 파라미터 값을 그대로 상속받게 된다. 이 중에는 실행 가능한 프로세스 양의 최대값을 제한하는 pid_max 값이 있다. 호스트가 여러 컨테이너를 소유하고 있는 경우 호스트의 pid_max 값을 컨테이너들이 똑같이 상속 받게 되므로 컨테이너는 호스트의 프로세스 자원을 제한 없이 이용할 수 있는 것과 마찬가지이다. 이러한 특성을 악용하여 공격자 컨테이너가 더미 프로세스를 과하게 생성하였을 때 피해자 컨테이너에 어떤 영향을 미치는지 실험하였다. Fig. 6는 이러한 프로세스 자원 고갈 공격에 대한 시나리오이다.

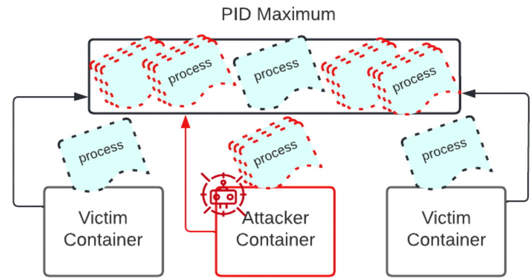


Fig. 6. The scenario of the process exhaustion attack.

실험 과정은 공격자 컨테이너에서 fork() 함수를 이용하여 프로세스를 빠르게 복제한다. 이와 같은 공격은 포크 폭탄(fork bomb)으로도 알려져 있다. 이때 피해자 컨테이너는 5초의 간격을 두고 프로세스의 PID 값을 출력 및 종료를 반복하여, 프로세스 생성 작업에 문제가 없는지를 모니터링 한다.

공격자 컨테이너에서 포크 폭탄을 이용하여 만든 공격을 진행하여 빠르게 프로세스를 만들어냈다. 이에 두 개의 피해자 컨테이너는 프로세스를 10회 반복하다 "Resource Temporarily Unavailable" 오류와 함께 프로세스를 실행시키지 못했다. 이는 호스트내의 여러 컨테이너를 운영 중이라면 매우 치명적인 문제로 작용할 것으로 예상된다. Fig. 7은 공격 후의 피해자 컨테이너에서 프로세스 자원이 부족하여 프로세스가 작동하지 않는 것을 보여준다.

```
===========
Calculating. . .
PID = 4889
Calculate complete
===========
Try count
2
===========
Calculating. . .
PID = 4890
Calculate complete
===========
Try count
3
./UIDSH.sh: fork: retry: Resource temporarily unavailable
./UIDSH.sh: fork: retry: Resource temporarily unavailable
./UIDSH.sh: fork: retry: Resource temporarily unavailable
```

Fig. 7. The result of the process exhaustion attack.

### 3.5 소켓 자원 고갈 공격(RA-4)

컨테이너가 리눅스 호스트로부터 상속하는 커널 파라미터에는 file_max 라는 것이 있다. 이는 open 가능한 파일의 최대 수를 제한하는 파라미터이다. 소켓도 파일의 한 종류이므로 file_max 값에 영향을 받으며 소켓에 대한 제한을 따로 설정할 수 있다. (ulimit -a로 확인 가능) 즉, 컨테이너들이 소켓 제한량을 상속받게 된다면, 이는 호스트의 소켓 자원을 제한 없이 이용할 수 있는 것과 마찬가지이다.

이러한 특성을 악용하여, 만약 공격자 컨테이너가 소켓을 과하게 열고 닫지 않는다면 피해자 컨테이너에 어떤 영향을 미치는지 실험하였다. Fig. 8은 소켓 자원 고갈 공격에 대한 시나리오이다. 실험 과정은 열려있는 TCP 서버에 클라이언트가 접속하고 소켓을 닫지 않는 프로세스 구현하였다. 이후 공격자 컨테이너에서 반복적으로 실행하여 소켓을 소모시키도록 하였다. 이때 실제로 사용되는 서버 프로세스에서는 오랫동안 소켓을 열어둘 이유가 없기 때문에, 클라이언트에서만 소켓을 닫지 않는 방법으로 실험을 진행하였다. 피해자 컨테이너는 정상적인 통신을 외부와 지속한다.

실험 결과로 공격자 컨테이너에는 다량의 소켓이 열리면서 통신오류가 발생하였지만 다른 피해자 컨테이너에서는 정상적인 통신이 가능하였다. 이에 대한 원인을 소켓 통신 순서도(Fig. 9)와 함께 분석하자면 다음과 같다. 먼저 클라이언트 프로세스는 SYN received 이후로 대기하지만 서버 프로세스는 established 이후 대기 상태로 들어간 후 일정 시간이 지나면 소켓을 닫게 된다. 즉, 실제로 서버 프로세스가 소켓 파일을 열어둔 채로 유지된 것이 아닌 통신이 된 후에 타임아웃으로 종료된 것이다. 따라서 호스트의 가용 소켓은 일정하게 유지되므로 다른 컨테이너의 소켓 통신에 영향을 주지 못한 것이다. 만약 소켓 개수를 소진시키고자 한다면 SYN flooding 공격을 변형하여 진행하면 가능할 것으로 보인다.

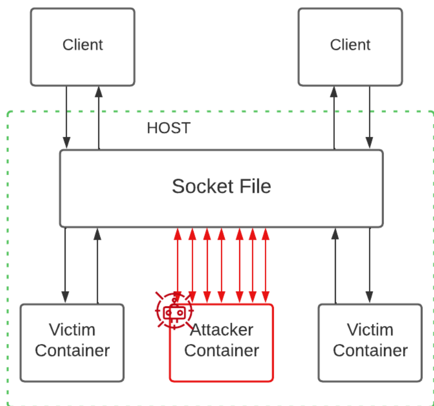


Fig. 8. The scenario for socker resource exhaustion attack.

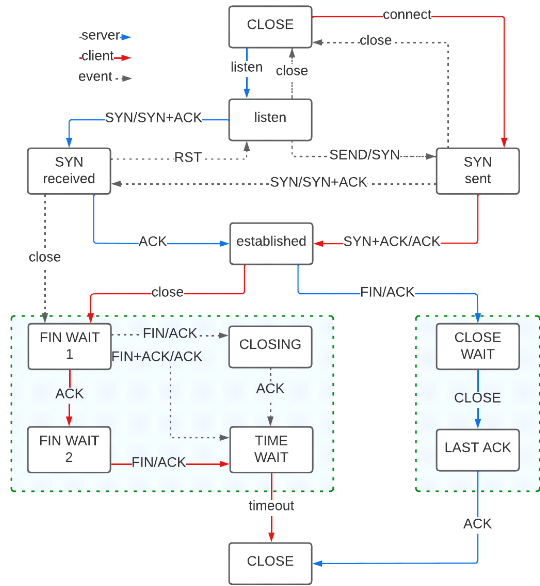


Fig. 9. Flow chart for socket communication.

### 3.6 유사 난수 발생기 고갈 공격(RA-5)

/dev/random은 유사 난수 발생기 중에서도 높은 보안성을 지닌 난수를 제공하는 특수 파일이다. /dev/random 난수 생성도(Fig. 10)와 같이 여러 요인들을 통해 불확실성을 수집하고 이를 통해 난수를 만들어낸다. 이렇게 높은 보안성을 지닌 난수 발생기의 특성상 2016년도부터 꾸준히 난수 생성 속도에 대한 문제점이 제시되었다.

도커 환경에서 /dev/random을 호스트에서 제공하며, 모든 컨테이너가 공유하는 방식이라 가정한다, 이 경우, 하나의 컨테이너에서 난수를 모두 소진시켜 난수를 이용해야하는 다른 컨테이너에서는 난수를 불러오지 못 할 것이다. Fig. 11은 이러한 유사 난수 발생기 고갈 공격의 시나리오이다.

실험 과정은 피해자 컨테이너에서 5초의 간격을 두고 /dev/random 으로부터 숫자 하나를 최대 10

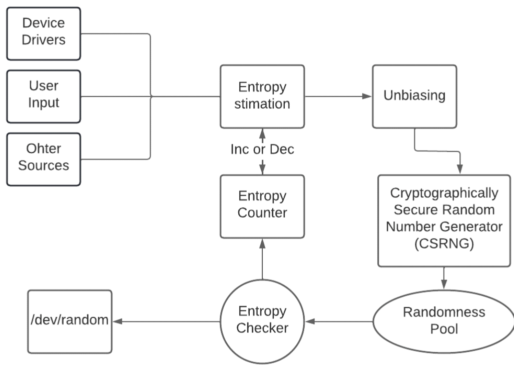


Fig. 10. Flow chart for the /dev/random procedure.

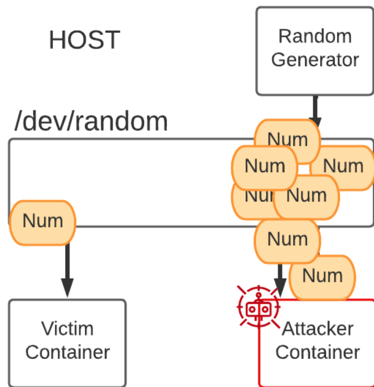


Fig. 11. The scenario 1 for pseudo-random number generator exhaustion attack.

개까지 받아 배열을 만들어 출력한다. 이때 공격자 컨테이너는 무한대로 /dev/random을 오픈하여 난수를 소모시킨다.

실험 결과로 공격자 컨테이너에서는 일정 파일 수를 넘은 후로 /dev/random이 파일 오픈량 제한에 의해 오류를 보였으나, 피해자 컨테이너는 문제없이 난수를 불러올 수 있었다. 이를 두고 두가지 경우로 나누어 예상하였는데, 첫 번째로 /dev/random은 컨테이너별로 독립되어 있다. 두 번째로 난수 소모 속도보다 난수 생성 속도가 월등히 빠르다. Fig. 12은 해당 실험 결과로 피해자 컨테이너는 문제없이 작동하나, 공격자 컨테이너는 오류가 발생한 것을 볼 수 있다.

추가 실험으로 여러 대의 컨테이너를 사용하여 24시간 동안 /dev/random을 지속적으로 소모하는 실험을 진행하였으나, 결과는 달라지지 않았다. 자료

```
Attack in progress. . .                              Attacker Container
open count : 56668
/dev/random can not access ( Maximum open exceeded )
Attack in progress. . .
open count : 56669
/dev/random can not access ( Maximum open exceeded )
Attack in progress. . .
open count : 56670
/dev/random can not access ( Maximum open exceeded )
Attack in progress. . .
open count : 56671
/dev/random can not access ( Maximum open exceeded )
=============
Calculating. . .                                     Victim Container
Random value : 3619724129
Random value : 3475649046
Random value : 3291877177
Random value : 3620926988
Random value : 2218920211
Random value : 3960195355
Random value : 243462692
Random value : 2867707157
```

Fig. 12. The result of pseudo-random number generator exhaustion attack (top: attacker container, bottom: attacker container).

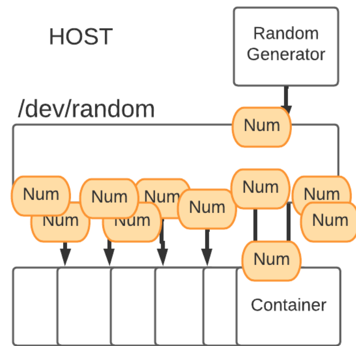


Fig. 13. The scenario 2 for pseudo-random number generator exhaustion attack.

를 찾아본 결과 2019년도 이후로 /dev/random의 난수 부족 현상에 대한 언급이 없는 것으로 보아 일반적인 방법으로는 난수를 모두 소모하기 어려울 것이라고 추측된다. Fig. 13는 추가 실험에 대한 시나리오이다.

### 3.7 결과 요약 및 총평

Table 3은 버전별 자원 고갈 공격 결과를 나타내는 표이다. 유사 난수 발생기 고갈 공격과 소켓 자원 고갈 공격을 제외하고 모두 성공한 것을 볼 수 있다.

CPU 및 메모리 고갈 공격, 파일 시스템 고갈 공격, 프로세스 자원 고갈 공격이 이번 실험에서 성공한 자원 고갈 공격들이다. 위 공격들의 특징은 컨테

이너 설정을 통해 예방할 수 있다는 점이다. CPU와 메모리의 경우 컨테이너 실행 시에 옵션으로 주기만 하면 제어가 가능하다. 파일 시스템 고갈 공격과 프로세스 자원 고갈 공격 역시 예방이 가능하다. 하지만 이 두 공격의 경우 초기 설정이 복잡하다. 저장공간 제한의 경우 특정 파일 시스템을 사용중이여야 한다. 그 후에 도커 설정 값을 바꾸고 이미지를 다시 세팅해야한다. 이 외에도 실행 시에 옵션으로 주는 방법이 있다. 프로세스 자원 고갈 공격의 경우 커널 파라미터를 수정하여 예방할 수 있으나, 이를 위해서는 도커 파일을 통해 설정하거나 privileged 모드를 이용해야한다. 이때 privileged 모드는 또 다른 취약점이 될 수 있기 때문에 위험하다.

소켓 자원 고갈 공격의 경우 실험을 진행하면서 컨테이너 간 통신, 외부와 컨테이너의 통신 등에 추가적인 취약점을 확인할 수 있었다. 본 논문에서는 호스트의 자원을 고갈 시키는 공격을 위주로 다루기 때문에 리눅스의 소켓이라는 파일 자원을 고갈 시키는 방법으로 고안한 실험이었으나, 실제로는 소켓 통신에서 다루어져야 하는 실험이기 때문에 추가적인 연구를 통해 다룰 예정이다.

유사 난수 발생기 고갈 공격의 경우 실제로 비슷한 원리로 공격을 진행한 연구가 있었다[6]. 해당 연구에서는 유사 난수 발생기의 난수 생성 속도보다

Table 3. Summary of experimental results of attack scenarios according to test machines and container image versions.

| Test Machine | Container Ubuntu Version | RA-1 | RA-2 | RA-3 | RA-4 | RA-5 |
|---|---|---|---|---|---|---|
| Machine A (Intel® i7 10800F) | 14.04 LTS | O | O | O | X | X |
| | 16.04 LTS | O | O | O | X | X |
| | 18.04 LTS | O | O | O | X | X |
| | 20.04 LTS | O | O | O | X | X |
| Machine B (Intel® i7 11800H) | 14.04 LTS | O | O | O | X | X |
| | 16.04 LTS | O | O | O | X | X |
| | 18.04 LTS | O | O | O | X | X |
| | 20.04 LTS | O | O | O | X | X |
| Machine C (Apple M2) | 14.04 LTS | O | O | O | X | X |
| | 16.04 LTS | O | O | O | X | X |
| | 18.04 LTS | O | O | O | X | X |
| | 20.04 LTS | O | O | O | X | X |

빠르게 소모시킬 수 있었으나, 해당 문제점이 지속적으로 업데이트 되어 해결이 된 것으로 보인다. 본 실험에 대해서는 추후 추가적인 연구가 필요할 것으로 여겨지며, 가능한 시나리오로 여러 대의 머신을 이용하여 난수를 소모시킬 수 있는지, 리눅스 커널의 파라미터 중 open 파일 최대 개수를 조작하여 난수 생성을 제한 할 수 있는지 등이 있다.

## IV. 논 의

본 장에서는 제시한 자원 고갈 공격의 시사점 및 원인과 이에 대한 대응방법에 대해 논의하도록 한다.

### 4.1 공개 컨테이너 이미지에 대한 보안 인식 미비

현재 컨테이너 분야에서는 도커 허브[10]와 같이 사전에 제작된 컨테이너 이미지를 공개적으로 배포하는 저장소가 보편화되어 있다. 특히, 도커 허브에는 도커가 공식적으로 제공하거나, 도커의 확인을 받은 개발자가 제작한 소프트웨어를 제외하고도 많은 일반 사용자들이 제작한 컨테이너 이미지 파일들이 존재한다. 보안에 대해 민감한 기업이나 기관은 이런 이미지들을 사용할 때, 여러 가지의 취약점에 대해 고려하고 도커의 자원 제한 옵션들을 사용하여 컨테이너를 관리할 것이다. 그러나 보안 지식이 없는 개인 사용자, 소규모 그룹 사용자들은 이러한 도커의 취약점에 대해 잘 인지하지 못할 것이다. 따라서 만약 자원 고갈 공격을 수행하는 컨테이너 이미지를 설치하게 될 시 전체 컨테이너 운용에 악영향을 끼칠 수 있다.

### 4.2 컨테이너 이미지의 자원 제한 가이드라인 부재

제시된 자원 고갈 공격을 방지하기 위해서는 컨테이너 이미지 배포자가 적절한 정보를 제공하는 것 또한 요구된다. 특히 해당 컨테이너가 어떤 자원을 소모하는지와 어떻게 자원 제한을 걸 수 있는지 등에 관한 정보를 제공해야한다. 예를 들어 데이터베이스와 같이 저장 공간에 크게 영향을 주는 이미지들의 경우 컨테이너의 저장 공간, 파일 제한량 등을 설정하는 방법에 대해 제공하여야 한다. 이러한 사실을 모르고 데이터베이스와 웹 서버 컨테이너를 동시에 운영 시, 데이터베이스 컨테이너가 저장 공간을 모두 사용하여 웹 서버 컨테이너에 영향이 갈 수가 있다. 현재 도커 이미지를 배포하고 있는 도커 허브의 공식

배포 이미지조차 컨테이너가 사용하는 자원에 대한 제어 방법을 제공하고 있지 않다.

### 4.3 컨테이너 플랫폼의 기본 설정 보완 필요

도커와 같은 컨테이너 플랫폼에서 자원 제한에 관한 기본 설정을 살펴보면, 컨테이너가 호스트의 가용 자원량과 동일하도록 커널 파라미터와 cgroup을 상속받는데[8], 이는 자원 사용 제한이 없는 것과 마찬가지이다. 따라서 기본 설정으로는 컨테이너가 모든 종류의 자원 고갈 공격에 노출될 수 있다. 이를 보완하기 위해서 컨테이너 플랫폼의 기본 자원 제한 옵션을 정의할 필요가 있다. 예를 들어서, 컨테이너 플랫폼 설치 시 'CPU 1개 및 50% 제한' 등과 같은 기본 cgroup을 몇 가지 정의하여 사용자에게 선택하도록 할 수 있다.

### 4.4 컨테이너의 사용 자원 사전 파악 필요

자원 고갈 공격에 대한 대응책으로 컨테이너를 배포 전 테스트를 통해 사용하는 자원의 최대치를 파악한 후 이에 맞게 제한을 줄 수 있다. 이를 위해 오프라인 테스트를 통하여 컨테이너의 자원 최대 사용량을 파악한 뒤 알맞은 cgroup이나 커널 파라미터를 컨테이너별로 할당해야 한다. 이 때 컨테이너 플랫폼의 컨테이너 런타임 옵션을 활용할 수 있다. 예를 들어, 도커의 경우 docker run 명령어 실행 시 --cpus, --memory 등으로 CPU, 메모리 cgroup 서브시스템을 설정할 수 있으며, 커널 파라미터의 경우 --ulimit, --sysctl 등을 통해 조정 할 수 있다[8]. 다만 오프라인 테스팅 시 컨테이너의 자원이 최대로 활용되는 적절한 워크로드를 찾는 것이 필요할 것으로 예상된다[7].

### 4.5 호스트 자원 실시간 모니터링 및 제어 필요

컨테이너들의 사용 자원이 호스트의 최대 자원을 초과하는지를 수시로 모니터링하는 것도 대응 방법이 될 수 있다. 현재 cgroup과 같은 메커니즘으로 개별 컨테이너의 자원 최대치는 제한할 수 있지만, 호스트의 총 소요 자원을 모니터링하지는 않고 있다. 따라서 이른바 자원에 대한 글로벌 리미트(global limit)를 명시하고 컨테이너의 소모 자원 총합이 이를 초과하는지를 수시로 모니터링하여 제어를 해야할 것이다.

## V. 관련 연구

본 장에서는 컨테이너 보안과 관련된 연구를 살펴보도록 한다.

최근 컨테이너 서비스가 실제로 사용하는 시스템 콜 집합을 분석하여 컨테이너에 과도한 시스템 콜 권한을 부여하지 않도록 하는 연구들이 제안되었다[3, 4]. Ghavamnia 등은 정적 프로그램 분석 기법을 사용하여 컨테이너에서 실행되는 서비스가 필요로 하는 최소한의 시스템 콜 집합을 파악하는 시스템인 Confine을 제안하였다[3]. Lei 등은 동적 프로그램 분석 기법을 이용하여 컨테이너 서비스가 실제로 사용하는 시스템 콜 집합을 학습하고 이를 런타임에 적용하는 시스템인 SPEAKER를 제안하였다[4]. 그러나, 이들 모두는 공격자가 시스템 콜을 악용하여 컨테이너에 침입한다는 가정하에 이를 방지하고자 제안된 시스템이며 자원 고갈 공격을 방어하지는 못한다는 한계점이 존재한다.

한편으로는 본 논문에서와 같이 컨테이너가 호스트 및 다른 컨테이너와 동일한 커널을 공유한다는 점을 악용한 자원 고갈 공격이 제안되었다[5, 6]. Gao 등은 cgroup으로 컨테이너에 자원을 제한하더라도 cgroup 제약 범위 밖의 커널 스레드를 이용해 자원을 고갈 시키는 공격을 제안하였다[5]. Yang 등은 커널 파라미터나 데이터 구조와 같은 추상 자원들을 고갈시킴으로써 다른 컨테이너의 정상적인 작업을 방해하는 공격을 제시하였다[6]. 이와 같이 기존에 제시된 자원 고갈 공격들은 본 연구와 그 목적이 유사하다. 그러나 이들은 모두 리눅스 커널의 특징을 예상치 못한 방향으로 악용해 자원 제한 사항을 우회하는 것을 목적으로 한다. 따라서 공격자가 커널 스레드, cgorup, 커널 파라미터 구현체 등 리눅스 커널 내부 사항에 대한 지식을 가져야한다. 그러나 본 논문의 경우, 공격자가 컨테이너 플랫폼의 부실한 기본 cgroup과 커널 파라미터 제어 정책을 노리기 때문에, 상세한 구현 사항을 알 필요 없다는 점에서 차이가 있다.

## VI. 결론 및 향후 연구

컨테이너는 오늘날 멀티테넌트 클라우드 환경에서 널리 도입되고 있는 호스트 가상화 기술이다. 컨테이

너 환경을 쉽게 구성할 수 있게 해주는 쿠버네티스나 도커와 같은 플랫폼의 등장에 힘입어 컨테이너는 다양한 분야에서 적극적으로 도입되고 있다. 반면, 그동안 컨테이너 환경에서 발생할 수 있는 잠재적인 취약점은 충분히 검토되지 않았는데, 특히 자원 고갈 공격에 관련된 보안 문제가 충분히 다뤄지지 않았다. 본 논문에서 제시한 자원 고갈 공격 시나리오들은 충분히 예방 가능한 공격들임에도 불구하고, 일반 컨테이너와 호스트가 쉽게 영향을 받는 것을 보였다. 실험 결과로 현재 도커 허브와 같이 공개 컨테이너 저장소에서 배포되는 컨테이너 이미지가 자원 고갈 공격에 취약함을 보였다. 향후 연구로, 실제 클라우드 환경에서 배포되는 컨테이너 이미지를 대상으로 자원 고갈 공격을 재현해 보고자한다. 또한, 컨테이너 이미지가 자원 고갈 공격에 취약한지 여부를 자동으로 찾아주는 툴을 개발하여 안전한 컨테이너 환경을 만드는 데 더욱 기여하고자 한다.

## References


〔1〕 Gartner Research, *"Best Practices for Running Containers and Kubernetes in Production"* https://www.gartner.com/en/documents/3902966, Feb. 2019.

〔2〕 Aqua Sec, "CVE-2022-0185 in Linux Kernel Can Allow Container Escape in Kubernetes," https://blog.aquasec.com/cve-2022-0185-linux-kernel-container-escape-in-kubernetes, Jan. 2022.

〔3〕 S. Ghavamnia, T. Palit, A. Benameur, and M. Polychronakis, "Confine: Automated system call policy generation for container attack surface reduction," 23rd International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2020), pp. 443-458, Oct. 2020.

〔4〕 L. Lei, J. Sun, and K. Sun "SPEAKER: Split-phase execution of application containers," International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment. Springer, Cham, pp. 230-251, Jun. 2017.

〔5〕 X. Gao, Z. Gu, and Z. Li, H. Jamjoom, and C. Wang, "Houdini's escape: Breaking the resource rein of linux control groups," Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, pp. 1073–1086, Nov. 2019.

〔6〕 N. Yang, W. Shen, J. Li, Y. Yang, K. Lu, J. Xiao, T. Zhou, and C. Qin, "Demons in the shared kernel: Abstract resource attacks against os-level virtualization," Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security, pp. 764-778, Nov. 2021.

〔7〕 K. McDonough, X. Gao, and S. Wang "TORPEDO: A Fuzzing Framework for Discovering Adversarial Container Workloads," 2022 52nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), pp. 402-414, Jun. 2022.

〔8〕 Docker Docs, "Runtime options with Memory, CPUs, and GPUs" https://docs.docker.com/config/containers/resource_constraints/, Jul. 2022.

〔9〕 Docker Docs, "Configure namespaced kernel parameters (sysctls) at runtime" https://docs.docker.com/engine/reference/commandline/run/#configure-namespaced-kernel-parameters-sysctls-at-runtime/, Jul. 2022.

〔10〕 Docker Hub, "Docker hub" https://hub.docker.com/, Jul. 2022.

## 〈저자소개〉

이 준 희 (Jun-hee Lee) 학생회원
2019년 2월~현재: 광운대학교 소프트웨어학부 학사과정
〈관심분야〉 컨테이너, 보안, 클라우드 컴퓨팅

남 재 현 (Jae-hyun Nam) 종신회원
2013년 2월: 서강대학교 컴퓨터공학과 학사
2015년 2월: 한국과학기술원 정보보호대학원 석사
2020년 2월: 한국과학기술원 전산학부(정보보호대학원) 박사
2022년 9월~현재: 단국대학교 컴퓨터공학과 조교수
〈관심분야〉 클라우드 컴퓨팅, SDN/NFV, 컨테이너, 시스템 보안, 네트워크 보안

김 진 우 (Jin-woo Kim) 정회원
2015년 2월: 충남대학교 컴퓨터공학부 학사
2017년 2월: 한국과학기술원 전산학부(정보보호대학원) 석사
2022년 2월: 한국과학기술원 전기및전자공학부 박사
2022년 3월~현재: 광운대학교 소프트웨어학부 조교수
〈관심분야〉 네트워크 보안, 클라우드 보안, SDN