JKSCI

한국컴퓨터정보학회논문지
Journal of The Korea Society of Computer and Information
Vol. 25 No. 9, pp. 21-29, September 2020
https://doi.org/10.9708/jksci.2020.25.09.021

Security Assessment Technique of a Container Runtime Using System Call Weights

Jihyeok Yang*, Byungchul Tak**

*Student, School of Computer Science and Engineering, Kyungpook National University, Daegu, Korea
**Professor, Dept. of Computer Science and Engineering, Kyungpook National University, Daegu, Korea

[Abstract]

In this paper, we propose quantitative evaluation method that enable security comparison between Security Container Runtimes. security container runtime technologies have been developed to address security issues such as Container escape caused by containers sharing the host kernel. However, most literature provides only a analysis of the security of container technologies using rough metrics such as the number of available system calls, making it difficult to compare the secureness of container runtimes quantitatively. While the proposed model uses a new method of combining the degree of exposure of host system calls with various external vulnerability metrics. With the proposed technique, we measure and compare the security of runC (Docker default Runtime) and two representative Security Container Runtimes, gVisor, and Kata container.

▸ **Key words**: Container Security, Container Runtime, Vulnerability, System call, Exploit

[요 약]

본 연구에서는 보안 컨테이너 런타임 간의 직접적인 보안성 비교를 가능하게 하는 정량 평가 기법을 제안한다. 보안 컨테이너 런타임(Security Container Runtime) 기술들은 컨테이너가 호스트 커널을 공유하여 발생하는 컨테이너 탈출(Container escape)과 같은 보안 이슈를 해결하기 위하여 등장하였다. 하지만 대부분의 문헌들에서 컨테이너 기술들의 보안성에 대하여 사용 가능한 시스템 콜 개수와 같은 대략적인 지표를 이용한 분석만을 제공하고 있어서 각 런타임에 대한 정량적인 비교 평가가 힘든 실정이다. 반면에 제안 모델은 호스트 시스템 콜의 노출 정도를 다양한 외부 취약점 지표들과 결합하는 새로운 방식을 사용한다. 제안하는 기법으로 runC(도커 기본 런타임) 및 대표적인 보안 컨테이너 런타임인 gVisor, Kata container의 보안성을 측정하고 비교한다.

▸ **주제어**: 컨테이너 보안, 컨테이너 런타임, 취약점, 시스템 콜, 취약점 공격

• First Author: Jihyeok Yang, Corresponding Author: Byungchul Tak
*Jihyeok Yang (flash0223@knu.ac.kr), School of Computer Science and Engineering, Kyungpook National University
**Byungchul Tak (bctak@knu.ac.kr), Dept. of Computer Science and Engineering, Kyungpook National University
• Received: 2020. 08. 13, Revised: 2020. 09. 02, Accepted: 2020. 09. 03.

http://www.ksci.re.kr pISSN:1598-849X | eISSN:2383-9945

I. Introduction

컨테이너는 현재 클라우드 컴퓨팅 분야에서 가장 핵심적인 가상화 기술이다. 컨테이너는 호스트 커널을 공유하기 때문에 패키징 및 배포가 쉽고 빠르다는 이점을 가져 IT 산업에서 전반적으로 이용되고 있다. 하지만 호스트 커널을 공유하는 것으로 인하여, 컨테이너에 접근하여 호스트로 빠져나가는 컨테이너 탈출(container escape)[1]와 같은 보안 취약점[2]을 가진다. 이러한 보안상 취약점을 해결하기 위하여, 보안 컨테이너 런타임 (security container runtime) 기술이 등장하였다. 보안 컨테이너 런타임은 호스트 커널과 컨테이너 런타임 사이에 커널 기능을 수행하는 컴포넌트를 추가하여 호스트 커널의 사용을 최소화하여 보안성을 높이고자 하는 시도이며, 대표적인 보안 컨테이너 런타임으로 구글의 gVisor[3], Kata container[4], IBM의 nabla container[5]등이 있다.

다양한 보안 컨테이너 런타임 기술의 등장으로 인하여 기술들 간의 장단점을 이해하여 최적의 기술을 채택하여야 하는 문제점이 발생하였다. 또한 동일한 기술이라 하더라도 설정된 옵션, 지속적인 업데이트로 인하여 보안성이 변하기도 하므로 보안 컨테이너 런타임 기술의 선택이 더욱 어렵고 중요한 문제가 된다.

하지만 현재까지는 보안 컨테이너 런타임의 보안성을 비교하는 잘 알려진 방법은 없는 실정이다. 기존 도커 컨테이너 런타임인 runC나 보안 컨테이너 런타임 간의 비교 및 분석 자료의 주된 내용은 런타임의 성능에 대한 비교 및 분석을 하는 자료들이다. 이러한 자료들에서 컨테이너 런타임의 보안성을 측정하는 것은 대략적인 지표를 이용한 간단한 비교 정도이다.

현재의 보안 컨테이너 런타임 비교 분석 연구에서 보안성의 지표로 사용하는 것을 살펴보면, 컨테이너 실행에 허용되는 호스트 시스템 콜 개수 혹은 사용되는 호스트 커널 함수 개수 및 코드 양이 있다. gVisor와 runC(Docker의 기본 런타임)에 대한 비교 연구[6]는 성능 비교를 다루며, 오직 gVisor의 사용에 허용되는 호스트 시스템 콜 개수를 분석한다. 그리고 서로 다른 보안 컨테이너 런타임에 대한 비교 분석 연구[7, 8] 또한 성능 비교를 다루며, 보안성 관점에서는 컨테이너 실행에 허용되는 호스트 시스템 콜 개수, 사용되는 호스트 커널 코드 양[7] 혹은 사용되는 호스트 커널 함수 개수 [8]에 대하여 비교한다. 하지만 이러한 지표들은 각 함수나 코드 라인 마다 위험도가 다르기 때문에 보안성의 지표로 사용하기는 어렵다.

본 연구에서는 보안 컨테이너 런타임의 보안성을 하나의 컨테이너 런타임의 위험도 수치로 정량화 하여 비교를 가능하게 하는 기법을 제안한다. 컨테이너 런타임의 위험도는 각 시스템 콜 별 호스트 시스템 콜의 사용률과 리스크(Risk)정보를 바탕으로 측정된다. 시스템 콜의 위험도는 리눅스 커널과 연관된 CVE의 공격 코드를 파싱하여 사용되는 라이브러리 함수로부터 호출되는 시스템 콜을 추출하는 것을 바탕으로 한다. 추출된 시스템 콜 정보를 통하여 CVE[9]의 CVSS 점수를 통해 해당 시스템 콜이 사용되는 CVE의 위험도, 해당 시스템 콜이 사용되는 CVE 취약점을 이용한 공격 코드에서 해당 시스템 콜의 중요도를 바탕으로 측정되며 수치화 된다. 최종적으로 수치화 된 각 시스템 콜의 위험도와 사용률을 곱한 값의 합을 컨테이너의 위험도를 나타내는 수치로 사용한다. 본 연구의 실험에서는 제안하는 기법을 이용하여 수집한 공격 코드를 바탕으로 생성된 시스템 콜 위험도 정보에 대하여 분석한 자료를 제시한다. 그리고 생성된 시스템 콜 위험도 정보를 이용하여 도커의 기본 런타임인 runC와 두 가지 보안 컨테이너 런타임 Kata container, gVisor의 위험도를 측정하였다. 그 중 gVisor는 버전 별, 옵션 별 보안 컨테이너 런타임의 위험도 변화를 측정하기 위하여, 6개월 간격의 두 가지 버전과 호스트 네트워크 스택을 사용하는 옵션, 사용하지 않는 옵션 두 가지 네트워크 옵션에 대한 추가 실험을 실행하였다. 그 결과 gVisor는 runC 보다 위험도 점수가 낮고, Kata container 보다 높았다. 그리고 두 버전의 gVisor는 seccomp filter에 의해 허용되는 호스트 시스템 콜의 개수가 차이가 나지만 동일한 위험도를 가졌고, 호스트 네트워크 스택을 사용하는 경우, 위험도가 더 증가하였다. 본 논문에서는 이에 대한 자세한 분석을 제시한다.

본 논문의 구성은 다음과 같은 순서를 가진다. 제 2장에서는 보안 컨테이너 런타임에 대한 배경 지식 및 관련 연구를 소개한다. 제 3장에서는 제안하는 보안 컨테이너 런타임 보안성 측정 기법에 대하여 소개 하며, 제 4장에서는 제안하는 기법을 이용한 보안 컨테이너 런타임 보안성 측정을 하며, 측정 결과에 대한 분석을 제시한다. 마지막으로 제 5장에서 결론 및 향후 계획을 기술한다.

II. Preliminaries

1. Security Container Runtime

기존의 도커의 기본 컨테이너 런타임인 runC는 호스트 커널을 공유하며, 호스트와 격리된 환경을 제공하기 위하여 seccomp, namespace, cgroups등 리눅스 커널의 보안 기능을 사용하였다. 하지만 호스트 커널을 공유한다는

근본적인 문제 때문에, 컨테이너 탈출과 같은 보안적 이슈가 존재한다. 보안 컨테이너 런타임 (Security Container Runtime)기술은 호스트 커널의 사용을 최소화하는 것에 초점을 맞춘 기술이며, 현재까지 등장한 보안 컨테이너 런타임 기술들은 서로 다른 방식으로 호스트 커널과 컨테이너 사이에 커널 기능을 가지는 컴포넌트를 추가하여 호스트 커널의 사용을 최소화 하였다. 대표적인 보안 컨테이너 런타임 몇 가지를 살펴보면, 먼저 gVisor[3]는 실행되는 시스템 콜의 호출을 가로채어 커널 기능을 수행하는 프로세스인 Sentry에서 호스트 커널 대신 수행한다. nabla container[5]의 경우 단일주소 공간, 단일 프로세스만을 가지는 Library OS를 탑재한 유니 커널 가상머신을 하나의 프로세스로 동작하도록 만들어[10] 프로세스의 시스템 콜 호출을 Library OS 내에서 처리하여 호스트 커널의 사용을 감소시켰다. 그리고 Kata container[4]는 런타임이 경량화된 가상머신을 생성하고, 그 안에 컨테이너를 생성하여 가상머신 속의 컨테이너를 기존 컨테이너 에코 시스템에서 제어 할 수 있도록 만든 컨테이너 런타임 기술이다. 따라서 Kata container 내에서 실행되는 프로세스는 가상머신의 커널(게스트 커널)을 이용하게 되고, 호스트 커널의 사용량은 가상머신과 동일한 수준을 가진다.

2. Related works

2.1 Kernel Security Metric

기존에 리눅스 커널 자체의 보안성을 비교하기 위한 지표를 제시하는 연구들이 존재한다. 공격 표면 평가 기법[11]은 커널의 공격 표면의 위험도를 평가하기 위하여 사용되는 커널 코드 라인의 수, 해당 커널 코드 파트를 이용하는 CVE의 개수, 순환 복잡성[17] 총 독립된 세 가지의 지표를 제시한다. 세 지표에서 코드 라인 각각의 위험도를 고려하지 않는 라인의 개수, CVE의 위험도를 고려하지 않는 해당 커널 코드 파트를 이용하는 CVE의 개수이기 때문에 지표에서 위험도 부분을 고려하는 것이 필요하다. 그리고 Lock-in-Pop[12]은 인기 있는 어플리케이션들에서 자주 사용되는 커널 코드가 자주 사용되지 않는 코드들 보다 취약점이 적다는 것을 증명하며, 커널 코드의 이용 정도를 커널 보안성의 지표로 사용하였다. 하지만 이 지표에서 증명된 자주 사용되는 커널 코드에 포함되는 취약점은 레이스 컨디션, 커널 내부 데이터 구조체 결함 등 측정 되지 못한 취약점들이 있어 정확하다고 할 수 없다. 마지막으로 컨테이너의 보안성 강화를 위한 로우 레벨 인터페이스 연구[13]는 기존의 인터페이스와의 비교를 위하여 인터페이스에서 사용하는 커널 코드 사용량을 대략적으로 나타낼 수 있는 사용되는 커널 코드 양이라는 지표를 제안하였다. 이는 대략적인 보안성 측정 지표이며, 커널 코드의 위험도를 고려하는 것이 추가적으로 필요하다.

2.2 Security Container Runtime Comparison

현재 보안 컨테이너 런타임에 대하여 비교 분석을 한 연구는 다양하지 않으며, 주로 성능에 대한 비교 분석을 다루고 있다. gVisor와 runC에 대한 비교 연구[6]의 주된 비교는 시스템 콜 호출 오버헤드와 네트워크, 파일 입출력 등의 다양한 성능적인 측면에서의 실험이며, 보안성과 관련된 분석은 오직 gVisor에서 사용 가능한 호스트 시스템 콜 개수 뿐이다. gVisor와 아마존에서 개발 중인 보안 컨테이너 런타임인 firecracker[14]의 비교 분석 연구[7]는 두 런타임과 runC에 대하여 사용에 허용되는 호스트 시스템 콜 개수와 사용되는 호스트 커널 코드 양을 지표로 보안성을 분석하였다. 그리고 nabla container와 다른 보안 컨테이너 런타임의 공격 표면 측정[8]은 호스트 커널 함수 개수라는 지표를 통하여 보안성을 비교하였다. 기존의 보안 컨테이너 런타임 분석 연구에서는 보안 컨테이너 런타임의 보안성 비교를 위한 지표로 시스템 콜 혹은 함수의 개수, 코드의 양과 같은 단순한 지표를 사용한다. 하지만 이러한 지표들은 함수, 코드 라인 각각의 위험도가 다르기 때문에, 보안성 분석을 위한 지표로 사용하기 힘들다.

III. The Proposed Scheme

1. Proposed System Overview

보안 컨테이너 런타임 위험도 측정은 Fig. 1과 같이 i) CVE 정보를 이용하여 각 시스템 콜의 위험도를 수치화 시키는 것, ii) 타겟 컨테이너 런타임을 이용한 컨테이너에서 시스템 콜이 여러 파라미터 조합으로 호출 되었을 때 호스트 시스템 콜의 사용률을 측정하는 것, 두 단계를 통하여 얻은 정보를 이용하여 컨테이너 런타임의 위험도를 측정하게 된다.

2. Measuring System Call Risk

시스템 콜의 위험도는 시스템 콜 마다 다르게 측정될 수 있으며, 여러 가지 요소에 영향을 받는다. 예를 들어, 공격 코드에서 핵심적인 역할을 하는 시스템 콜은 해당 공격 코드가 이용하는 CVE의 다른 공격 코드에서도 이용되기 때문에, 다른 시스템 콜 보다 위험도가 높다. 반면에 등록된지 오래된 CVE에서 사용되는 시스템 콜들은 현재 대부분

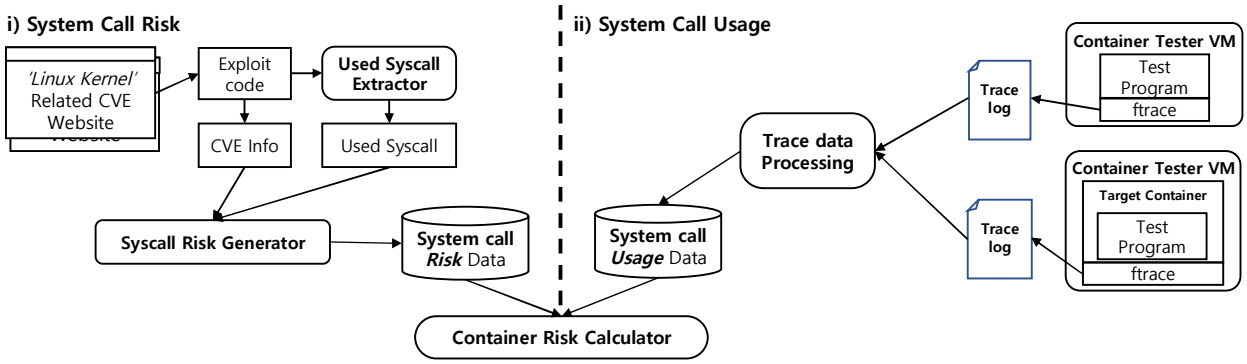

Fig. 1. Security Container Runtime Risk Measurement System Architecture

CVE에 대한 패치가 이미 완료된 커널 버전을 사용할 것이기 때문에, 위험도가 떨어진다. 이러한 시스템 콜의 위험도에 영향을 미치는 정보들을 반영하기 위하여 리눅스 커널과 연관된 공격 코드 및 CVE 정보를 이용하여 시스템 콜 위험도를 측정하였다. 측정 방법은 취약점에 대한 공격 코드를 제공하는 사이트인 exploitDB[15]에서 리눅스 커널과 연관된 공격 코드를 수집한다. 그리고 수집한 공격 코드들에서 시스템 콜 정보를 추출(Used Syscall Extractor)하고, CVE정보와 사용되는 시스템 콜 정보를 이용하여 시스템 콜 위험도 정보를 생성(Syscall Risk Generator)하는 것으로 이루어진다.

2.1 Used Syscall Extractor

Table 1. Exploit Code Count by Language

Language	Count
C	266 (93.33%)
Python	8 (2.81%)
Ruby	7 (2.46%)
Perl	2 (0.70%)
Assembly	2 (0.70%)
Total	**285**

Used Syscall Extractor의 역할은 수집된 리눅스 커널과 관련된 공격 코드들에서 사용하는 시스템 콜을 추출하는 것이다. 하지만 공격 코드들은 모두 라이브러리 함수로 이루어져 라이브러리 함수 중 시스템 콜과 이름이 같은 wrapper function을 제외하면, 라이브러리 함수에서 어떤 시스템 콜을 사용하는지 알 수 없다. 이러한 문제점을 해결하기 위하여 리눅스에서 사용하는 libc 오브젝트 파일을 역어셈블러[18]를 이용하여 어셈블리 코드로 만든 후, 어셈블리 코드에서 라이브러리 함수가 호출하는 다른 라이브러리 함수, 시스템 콜에 대한 매핑 테이블을 제작한다. 그 후, 수집한 공격 코드들을 파싱하여 코드에서 사용하는 라이브러리 함수를 추출하고, 추출된 라이브러리 함수를 매핑 테이블을 이용하여 코드에서 사용되는 시스템 콜 정보로 변환한다. 하지만 수집된 공격 코드들을 파싱하여 코드에서 사용하는 라이브러리 함수를 추출하여 libc 함수와 매핑하는 것은 간단하지 않다. C언어의 라이브러리 함수는 libc 함수의 이름을 그대로 사용하지만, 몇 가지 예외 케이스가 존재한다. 그리고 공격 코드는 C언어로만 구성되는 것이 아니며, Table. 1과 같이 소수의 C언어 이외의 언어인 파이썬, 루비 그리고 셸 스크립트로 구성된 코드가 존재한다. 이러한 C언어 이외의 언어들은 최종적으로 libc를 사용하긴 하지만 libc의 함수 이름을 그대로 사용하지 않는 예외 케이스가 C언어 보다 더 다양하게 존재한다. 본 연구에서는 리눅스 커널과 연관된 공격 코드 구성 중 대부분을 차지하는 C언어로 제작된 공격코드를 대상으로 파싱 작업을 진행하였다.

2.2 Syscall Risk Generator

Syscall Risk Generator는 수집된 공격 코드들을 CVE별로 분류하며, 각 CVE에 대한 CVSS[16] 정보를 수집한다. 그리고 각 CVE에서 등장하는 시스템 콜이 얼마나 핵심적인 역할을 하여 위험도가 높은 지 나타내기 위하여 tf-idf를 적용한 식 (1)를 정의한다. 식 (1)에서 d는 하나의 CVE, t는 시스템 콜로 둔다. 하나의 CVE에서 시스템 콜 빈도는 시스템 콜이 등장한 CVE와 연관된 공격 코드의 수이며, $DTM(d,t)$은 CVE-시스템 콜의 빈도를 나타내는 행렬이다. $idf(d,t)$은 기존의 tf-idf 기법에서 역 문서 빈도 수에 해당하는 식이며, 특정 시스템 콜 t가 전체 리눅스 커널과 연관된 CVE에서 얼마나 공통적으로 사용되는지 나타내는 값이다. 따라서 M은 $DTM(d,t)$과 $idf(d,t)$를 곱한 값으로, tf-idf 가중치가 고려된 CVE-시스템 콜 빈도 행렬이다.

$$M = DTM(d,t) \times idf(d,t) \quad (1)$$

그리고 각 CVE의 현재 시점에서의 위험도를 계산하기 위하여, 가중치 W를 식 (2)로 둔다. 식 (2)에서 $CVSS(d)$는 CVE d의 CVSS 점수를 나타낸다. $TW(d)$는 CVE d가 출시된 후 현재까지의 개월 수를 나타내는 시간 가중치이다. 따라서 가중치 W는 CVSS의 값이 높을수록 증가하지만, 해당 CVE가 오래될수록 가중치가 낮아지도록 정의된다.

$$W = \frac{CVSS(d)}{TW(d)} \quad (2)$$

마지막으로 행렬 M에서 j번째 열에 해당하는 시스템 콜의 위험도 v_j는 j번째 열에 해당하는 시스템 콜을 사용하는 CVE집합 S_j를 대상으로 $i \in S_j$ 수식이 성립하는 CVE i에 대하여, tf-idf가 고려된 CVE-시스템 콜 빈도 수 값 M_{ij}을 CVE 가중치 W_i를 곱하는 것을 모든 CVE에 대하여 수행한 후, 그 값들을 더해 평균을 구하여 시스템 콜의 위험도로 사용한다. 이는 식 (3)으로 정의된다.

$$v_j = Mean(\sum_i W_i \times M_{ij}) \quad (3)$$

3. Measuring Host System Call Usage Rate

보안 컨테이너 런타임 기술은 컨테이너에서의 사용 가능한 호스트 시스템 콜의 개수를 줄여 공격 표면을 감소시켜서 위험도를 줄인다. 즉, 사용 가능한 호스트 시스템 콜의 개수는 각 보안 컨테이너 런타임의 위험도 비교를 위한 중요한 평가 기준이 된다. 하지만 하나의 시스템 콜은 시스템 콜의 파라미터 조합에 따라 컨테이너 내에서 호스트 시스템 콜의 호출 가능 여부가 결정되기 때문에, 하나의 시스템 콜에 대하여 단순히 허용 여부 정보를 이용하는 것이 아닌 파라미터 조합을 고려하는 것이 좋다. 하지만 모든 시스템 콜의 모든 파라미터 조합 테스트는 불가능에 가깝다. 이러한 특성을 고려하여 보안 컨테이너 런타임의 사용 가능한 호스트 시스템 콜을 비교하기 위하여, 리눅스 커널을 테스트하기 위한 LTP[19] 프로젝트의 시스템 콜 테스트 프로그램을 사용하였다. LTP의 시스템 콜 테스트 프로그램은 모든 시스템 콜에 대한 다양한 테스트 프로그램을 가지고 있으며 각 시스템 콜 별 다양한 파라미터 조합을 가지고 시스템 콜을 테스트할 수 있기 때문에, LTP의 시스템 콜 테스트 프로그램은 보안 컨테이너 런타임의 호스트 시스템 콜 사용 정보를 측정하기에 적합하다. LTP를 이용한 컨테이너 런타임의 사용되는 호스트 시스템 콜 측정은 컨테이너의 시작과 종료까지 컨테이너 런타임 동작을 위한 프로세스에 대하여 측정되며, 호스트 시스템 콜 사용률 계산은 식 (4)와 같은 시스템 콜 별 호스트 시스템 콜 사용률 측정 식을 사용하였다. 식 (4)는 호스트에서 실행되는 테스트 프로그램에서 시스템 콜 a를 호출한 횟수 H_a, 컨테이너 런타임에서 똑같은 테스트 프로그램을 실행하였을 때 컨테이너 런타임에서 시스템 콜 a에 대하여 호출되는 호스트 시스템 콜의 개수를 S_a로 두어 하나의 시스템 콜에 대한 호스트 시스템 콜 사용률 U_a를 측정할 수 있다. 식 (4)는 리눅스의 전체 시스템 콜에 대하여 각 시스템 콜 별로 측정되며, 사용되는 호스트 시스템 콜을 추적(tracing)하기 위하여 적은 오버헤드 생성 및 특정 프로세스의 자식 프로세스를 모두 추적 할 수 있는 추적 도구인 ftrace[20]를 이용하였다.

$$U_a = Min(\frac{S_a}{H_a}, 1) \quad (4)$$

IV. Experiment

1. Container Runtime Risk Measurement

Table 2. Experiment Environment

	Version
OS	18.04.2
Kernel	4.18.0
Docker	19.03.6
gVisor	v20191104, v20200323
Kata container	1.11.0

실험의 구성은 먼저 제안하는 기법을 이용한 보안 컨테이너 런타임 위험도 비교 분석을 위하여 수집한 공격 코드를 바탕으로 생성된 시스템 콜 위험도 정보에 대하여 분석한다. 그리고 runC, gVisor, Kata container을 대상으로 컨테이너 런타임의 위험도를 측정한다. 보안 컨테이너 런타임은 같은 보안 컨테이너 런타임이라고 할지라도 버전별, 옵션 별로 런타임 위험도가 달라질 수 있다. 이를 확인하기 위하여, 세 런타임 중 gVisor를 이용하여 옵션 별, 버전별 보안성의 차이를 관찰한다. 옵션 별 보안성 차이를 관찰하기 위하여, gVisor 내부에 구현된 네트워크 스택을 사용하는 default 옵션과 호스트 네트워크 스택을 사용하는 hostnet 옵션 두 가지 네트워크 옵션 케이스에 대하여 위험도를 비교 분석한다. 그리고 버전 별 보안성 차이를 관찰하기 위하여, 2019년 버전 하나와 2020년 버전 하나를 대상으로 위험도를 측정한다. 자세한 컨테이너 런타임 별 실험 환경 정보는 Table. 2와 같다.

Table 3. Container Runtime Risk Measurement Result

	runC	gVisor-v20191104		gVisor-v20200323		Kata container
Network Option	-	default	hostnet	default	hostnet	-
The Number of Available Host Syscall by Seccomp	except 51	52	63	56	67	-
The Number of Syscall Used in Container Runtime	237	116	112	114	110	107
Container Risk Score - Frequency	0.938	0.875	0.865	0.875	0.865	0.787
Container Risk Score - with CVE Info	9.629	2.305	2.301	2.294	2.273	1.678

* The number of system call used by LTP test program is **282**

Table 4. The Rank of Unimportant Syscalls by Syscall Risk Measurement Method

	Frequency	With CVE Info
rt_sigprocmask	Rank 1	Rank 104
close	Rank 2	Rank 103
fcntl	Rank 3	Rank 101
dup	Rank 4	Rank 99
execve	Rank 5	Rank 72

Table 5. The Rank of Important Syscalls by Syscall Risk Measurement Method

	Frequency	With CVE Info
chown	Rank 57	Rank 1
prctl	Rank 48	Rank 2
syslog	Rank 46	Rank 3
mount	Rank 86	Rank 4
timerfd_create	Rank 102	Rank 5

2. System Call Risk Measurement Analysis

Table 4와 Table 5는 두 가지 방법에 의해 시스템 콜 위험도를 측정한 뒤 Table 4는 비교적 공격 코드에서 핵심이 되지 않는 시스템 콜의 측정 방법별 순위를 나타내고, Table 5는 공격 코드에서 핵심적인 시스템 콜의 측정 방법별 순위를 나타낸 표이다. 두 방법에 대하여 살펴보면, 첫 번째 방법은 단순히 시스템 콜이 등장한 공격 코드의 개수로 위험도를 측정하여 위험도 값은 해당 시스템 콜이 등장한 공격 코드의 개수를 전체 공격 코드의 개수로 나눈 값을 가지며, 측정한 결과는 Frequency열과 같다. 두 번째 방법은 본 연구에서 제안하는 CVE 정보를 포함한 시스템 콜 위험도 측정 방법이며, 측정 방법의 결과는 With CVE Info 열과 같다. 먼저 공격 코드에서 비교적 핵심적인 역할을 하지 않는 시스템 콜에 대하여 Table 4 측정된 결과를 살펴보면, 각 쓰레드의 동작에 관여하여 공격 코드의 핵심적인 역할을 하지 않는 rt_sigprocmask 시스템 콜의 경우, frequency열에서는 1위지만 With CVE Info열에서는 104위의 매우 낮은 순위를 가지고 있다. 그리고 파일을 단순히 닫는 역할을 하여 공격 코드의 동작에 핵심적인 역할을 하지 않는다고 판단되는 close 시스템 콜도 Frequency열에서 2위지만, With CVE Info열에서는 103위에 위치한다. 그 외 파일 특성을 변경하는 fcntl, 파이프를 생성하는 dup, 현재 프로세스에서 프로그램을 로드하여 실행하는 execve 역시 공격 코드에서 취약점 공격의 핵심이 아닌 시스템 콜들이다. 이들 또한 Frequency 열에서 상위권을 나타내지만, With CVE Info 열에서는 하위권에 위치한다. 이처럼 단순히 시스템 콜이 등장하는 공격 코드 개수를 시스템 콜의 위험도로 두었을 때, 공격 코드에서 핵심적인 역할을 하는 시스템 콜이 아닌 주변적인 시스템 콜이지만 많은 공격 코드에 등장하는 시스템 콜이 높은 위험도를 가지게 된다. 반면에 본 연구의 시스템 콜 위험도 측정 방법을 사용하면, 해당 시스템 콜들에 대하여 위험도가 낮게 측정된다.

Table 5를 살펴보면, chown은 CVE-2016-8655 취약점을 이용하는 공격 코드에서만 이용하는 시스템 콜이지만 직접적으로 영향을 끼치는 시스템 콜은 아니다. 하지만 prctl은 CVE-2006-2451 취약점에서 권한 상승을 유발하는 시스템 콜이다. 그리고 syslog 시스템 콜은 CVE-2017-7308 취약점을 이용하는 공격 코드에서 리눅스 커널 보안 메커니즘인 KASLR을 우회하는 작업에서 사용된다. mount 시스템 콜 또한, CVE-2015-8660의 overlayfs의 취약점을 이용하기 위해서 사용되는 핵심 시스템 콜이다. 마지막으로 timerfd_create 시스템 콜은 CVE-2017-10661에서 소개된 취약점인 timerfd를 생성하는 시스템 콜로, 해당 CVE를 이용하는 공격 코드에서 매우 핵심적인 시스템 콜이다. 이러한 시스템 콜들에 대하여, Frequency 열의 순위 값들은 모두 낮은 순위를 가지며, With CVE Info열의 순위 값들은 모두 상위권의 순위 값을 가진다. 이처럼 본 연구의 시스템 콜 위험도 측정 방

식으로 시스템 콜의 위험도를 측정하였을 때, 공격 코드에서 핵심적인 시스템 콜이 높은 위험도를 가져 시스템 콜의 위험도를 비교적 타당하게 측정하는 방법인 것을 알 수 있다. 하지만 chown 시스템 콜과 같이, 특정 취약점을 사용하는 공격 코드에서만 사용되지만 핵심적인 시스템 콜은 아닌 시스템 콜이 높은 위험도를 가지는 한계점을 가지는 것을 알 수 있다.

3 Container Runtime Risk Analysis

컨테이너 런타임 위험도 측정 결과는 Table 3과 같다. *'The Number of Available Host Syscall by Seccomp'* 열은 컨테이너 런타임이 seccomp에 의하여 정의된 사용할 수 있는 시스템 콜의 개수를 나타낸다. 이는 컨테이너 런타임 내에서 호출 할 수 있는 시스템 콜 개수이며, 기존의 컨테이너 런타임의 보안성을 나타내는 지표로 활용된 것이다. 그리고 *'The Number of Syscall Used Container Runtime'* 열은 본 연구 기법의 시스템 콜 사용률 측정 파트에서 컨테이너 런타임이 시작되고 컨테이너 런타임이 호스트와 정보를 주고 받으며, 런타임 내부에서 시스템 콜 테스트 프로그램을 동작시키는 모든 과정에서 추적된 사용하는 호스트 시스템 콜 개수이다. 마지막으로 *'Container Risk Score - Frequency'* 열과 *'Container Risk Score - with CVE Info'* 열은 시스템 콜 위험도 측정 기법을 다르게 하여 본 연구의 컨테이너 런타임 위험도 측정 기법을 통해 생성된 컨테이너 런타임 위험도 점수이며, 각각 시스템 콜이 등장한 공격 코드의 개수로 시스템 콜의 위험도를 측정한 것(Frequency)와 본 연구에서 제안하는 CVE 정보를 포함한 시스템 콜 위험도 측정 방법(with CVE Info)를 사용하였다.

3.1절 에서는 시스템 콜 위험도 측정 방식에 따라 전반적으로 컨테이너 위험도 점수 값이 어떻게 변화하는지 비교하고, 3.2절 ~ 3.4절에서는 CVE 정보를 포함한 시스템 콜 위험도를 이용한 컨테이너 위험도 점수 값만을 이용하여 해당 컨테이너 런타임의 위험도를 분석한다.

3.1 Score Comparison by Syscall Measurement Method

두 가지 시스템 콜 위험도 측정 방법에 따른 컨테이너 런타임 위험도 측정 점수를 살펴보면, 시스템 콜이 등장한 공격 코드의 개수로 시스템 콜의 위험도를 측정하였을 때, 점수의 차이가 존재하지만 점수 차이 폭이 크지 않다. 반면에 본 연구에서 제안하는 CVE 정보를 포함한 시스템 콜 위험도 측정 방법을 사용하였을 때에는 컨테이너 런타임 별로 런타임 위험도 점수가 큰 폭의 차이를 가졌다. 이는 보안 컨테이너 런타임에서 Table 4의 시스템 콜과 같이 주변적이면서 여러 곳에서 자주 쓰이는 시스템 콜의 경우 컨테이너 런타임을 구성하는 과정에서 쓰이기 때문에 대부분 컨테이너 런타임에서 호출 될 것이며, 이로 인하여 높은 위험도를 가지는 시스템 콜의 사용률이 비슷하고, 전반적으로 비슷한 점수를 가진다. 반면에 본 연구에서 제시한 CVE 정보를 포함한 시스템 콜 위험도는 Table 4의 시스템 콜들에 대하여 시스템 콜 위험도가 적기 때문에, 해당 시스템 콜에 의하여 전반적인 컨테이너 런타임 위험도 점수가 결정되지 않는다.

3.2 runC

runC 런타임은 Seccomp를 이용하여 51 개의 시스템 콜을 제외한 모든 시스템 콜에 대하여 컨테이너 내에서 호출이 가능하도록 설정[21]되어 있다. 본 연구 기법에서 사용한 시스템 콜 테스트 프로그램은 총 282개의 시스템 콜을 호출한다. 하지만 runC를 실행하여 시스템 콜 테스트 프로그램을 동작시킨 후, 컨테이너를 종료하는 것 까지 사용하는 시스템 콜 개수는 237개이다. 이 중 몇몇 시스템 콜들은 시스템 콜 테스트 프로그램의 호출 횟수 보다 더 많이 호출되었다. 이는 getcwd, getdents64, mkdirat와 같은 파일 시스템 관련 시스템 콜이 컨테이너 이미지를 이용하여 컨테이너의 파일 시스템을 생성하는 과정에서 사용되고, 그 외에 컨테이너를 관리하는 containerd-shim 프로세스의 동작에서 몇 가지 시스템 콜들이 사용되기 때문이다. 이와 같이 runC는 시스템 콜 테스트 프로그램에서 호출하는 282개의 시스템 콜 중 대부분의 시스템 콜에 대하여 호출 가능하기 때문에, 측정된 컨테이너 런타임 위험도가 보안 컨테이너보다 4배 이상 높게 측정된다.

3.3 gVisor

gVisor의 seccomp 규칙 파일[22]을 분석해보면, 버전 v20191104보다 v20200323에서 허용되는 호스트 시스템 콜 개수가 default 네트워크 옵션과 hostnet 네트워크 옵션을 사용할 때 모두 4개 더 많다. 이는 preadv, pwritev 과 같은 추가적인 파일 시스템 지원을 위한 시스템 콜과 gVisor의 리눅스 커널 5.2~5.4버전에서의 버그를 대응하기 위한 시스템 콜을 포함하여 총 6개가 추가되고, 버전이 업그레이드되면서 dup2, prctl 함수가 제거가 되었기 때문이다. 이와 달리 실제 컨테이너 런타임 동작에서 필요한 시스템 콜은 오히려 v20191104 버전 보다 v20200323에서 두 개가 줄었다. 줄어든 시스템 콜은 dup3와 statfs이며, 이 두 시스템 콜은 컨테이너 런타임 내부의 사용자 프

로세스에서 호출 된 것이 아닌 gVisor의 파일 시스템 역할을 하는 컴포넌트인 gofer에서 호출하는 함수이며, 버전이 업그레이드되면서 사용하지 않는 것으로 보인다. 이와 같이 실제 컨테이너 런타임 동작에서 사용되는 시스템 콜의 개수는 줄어들고, 몇 가지 호출 횟수가 감소한 시스템 콜들로 인하여 측정된 컨테이너 런타임 위험도 점수 또한 약간 감소하였다.

네트워크 옵션을 기준으로 gVisor의 seccomp 규칙 파일을 보면, 버전에 관계 없이 default 옵션보다 hostnet 옵션을 사용하는 경우, 11개의 bind, connect와 같은 네트워크와 관련된 시스템 콜을 더 사용할 수 있도록 규칙을 정의하였다. 하지만 실제 컨테이너 런타임 동작에서 필요한 시스템 콜은 오히려 default보다 hostnet에서 4개 감소하였다. 감소한 시스템 콜은 recvmmsg, setgid, setns, setuid에 해당한다. 해당 함수들은 gVisor 내부의 네트워크 스택인 netstack을 구현하기 위하여 사용되는 함수들로 분석되며, hostnet이 seccomp에서 허용하는 시스템 콜은 증가하였지만 실제 사용하는 시스템 콜 개수가 줄어든 이유는 컨테이너 내부 프로세스에 추가적으로 사용 가능한 네트워크 관련 시스템 콜이 컨테이너를 생성하고 컨테이너를 관리하는 과정에서 default 옵션에서도 기본적으로 네트워크 관련 시스템 콜들이 사용되기 때문이다. 이러한 이유로 줄어든 시스템 콜 개수로 인하여 컨테이너 런타임 위험도 또한 default에 비해 hostnet에서 더 감소하였다.

3.4 Kata Container

Kata container는 가상 머신을 사용하기 때문에, 컨테이너 내부의 프로세스가 호출하는 시스템 콜은 모두 가상 머신의 게스트 커널에서 처리하기 때문에, 컨테이너 내부에서 호출 할 수 있는 호스트 시스템 콜의 개수는 0개이다. 하지만 Kata container를 구동하여 테스트 프로그램을 실행하였을 때, 총 107개의 호스트 시스템 콜이 사용되었다. 이는 QEMU 가상 머신 모니터(VMM)과 같은 가상 머신 구동을 위한 컴포넌트가 사용하는 시스템 콜과 가상 머신의 표준 입출력을 호스트에 제공[23]하는 것에 있어서 사용되는 통신 관련 시스템 콜들이다. 이처럼 Kata container는 동작을 위하여 107개의 시스템 콜을 호출하지만, 컨테이너 내부 프로세스의 시스템 콜이 측정되지 않아 비교적 적은 호출 횟수를 가지기 때문에 시스템 콜 별 사용률이 적어 제일 낮은 런타임 위험도 점수를 가진다.

V. Conclusions

본 연구에서는 보안 컨테이너 런타임의 보안성을 컨테이너 런타임의 위험도로 정량화하는 방법을 제안하였다. 컨테이너 런타임의 위험도는 리눅스 커널과 연관된 공격 코드에서 이용되는 시스템 콜 위험도 정보와 보안 컨테이너 런타임에서 시스템 콜 테스트 프로그램을 구동하였을 때, 호출되는 호스트 시스템 콜 사용 정보를 바탕으로 측정된다. 그리고 runC, gVisor, Kata container에 대하여 컨테이너 런타임 위험도를 측정하였다. 그 결과 위험도는 runC가 제일 높았고, Kata container가 가상 머신을 사용하여 컨테이너 내부에서 동작하는 프로세스가 호스트 커널을 사용하지 않기 때문에 위험도가 제일 낮았다. gVisor는 Kata container에 비해 약간 높은 위험도를 가지고 있었고, 버전이나 네트워크 옵션 차이에 따른 위험도 차이는 거의 없었다.

향후 연구 진행 방향은 시스템 콜 위험도를 측정할 때, 특정 취약점과 연관된 공격 코드에서만 사용되지만 비핵심적인 시스템 콜이 높은 시스템 콜 위험도를 가지는 한계점을 해결하는 것이 있다. 그리고 현재 제안하는 기법이 시스템 콜만 지표로 사용하기 때문에, 그 외의 호스트 커널 사용의 위험도 측정을 할 수 있도록 다른 지표를 추가하는 것과 CVE에 대한 정확한 패치 정보와 같은 추가 정보들을 수집하도록 웹 크롤링 부분을 강화하는 것이 있다.

ACKNOWLEDGEMENT

This work was supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIT) (No. NRF-2019 R1C1C1006990)

REFERENCES

[1] Z. Jian, L. Chen, A Defense Method against Docker Escape Attack, In Proceedings of the 2017 International Conference on Cryptography, Security and Privacy (ICCSP'17), pp.142-146, Wuhan, China, March 2017. DOI: 10.1145/3058060

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

[3] GVisor, https://gvisor.dev

[4] Kata container, https://katacontainers.io
[5] Nabla container, https://nabla-containers.github.io/
[6] Ethan G. Young, et al., The True Cost of Containing: A gVisor Case Study., In Proceedings of the 11th USENIX Conference on Hot Topics in Cloud Computing(HotCloud'19), p. 16, Renton WA, USA, July 2019. 10.5555/3357034.3357054
[7] Anjali, Tyler Caraza-Harter, Michael M.Swift., Blending containers and virtual machines: a study of firecracker and gVisor., Proceedings of the 16th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE'20), pp. 101-113, Lausanne, Switzerland, March 2020. 10.1145/3381052.3381315
[8] Measuring the Horizontal Attack Profile of Nabla Containers, https://blog.hansenpartnership.com/measur ing-the-horizontal-attack-profile-of-nabla-containers/
[9] CVE, https://cve.mitre.org/
[10] D. Williams, R. Koller, M. Lucina, and N. Prakash. Unikernels As Processes. In Proceedings of the ACM Symposium on Cloud Computing, SoCC '18, pp. 199–211, New York, NY, USA, October 2018. 10.1145/3267809.3267845
[11] A. Kurmus, R. Tartler, D. Dorneanu, B. Heinloth, V. Rothberg, A. Ruprecht, W. Schroder-Preikschat, D. Lohmann, and R. Kapitza, Attack Surface Metrics and Automated Compile-Time OS Kernel Tailoring, in Proceedings of the 20th Network and Distributed System Security Symposium(NDSS'13), San Diego, CA, Feburary 2013.
[12] Y. Li, B. Dolan-Gavitt, S. Weber, and J. Cappos, Lock-in-Pop: Securing Privileged Operating System Kernels by Keeping on the Beaten Path. In Proceedings of In Annual Technical Conference USENIX ATC'17, pp. 1–13, SANTA CLARA, CA, July 2017. 10.5555/3154690.3154692
[13] D. Williams, R. Koller, and B. Lum. Say goodbye to virtualization for a safer cloud. In Proc. of USENIX HotCloud, p. 20, Boston, MA, July 2018. 10.5555/3277180.3277200
[14] A. Agache, M. Brooker, A. Iordache, A. Liguori, R. Neugebauer, P. Piwonka, and D.-M. Popa. Firecracker: Lightweight virtualization for serverless applications, In 17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20), pp.419-434, Santa Clara, USA, Feburary 2020.
[15] ExploitDB, https://www.exploit-db.com/
[16] CVSS v2 Calculator, https://nvd.nist.gov/vulnmetrics/cvss/v2-calculator
[17] T.J. McCabe. "A Complexity Measure". In: Software Engineering, IEEE Transactions on SE-2.4 (1976), pages 308–320. ISSN: 0098-5589. DOI: 10 . 1109 / TSE . 1976. 233837
[18] Objdump man page, https://linux.die.net/man/1/objdump
[19] LTP Project, https://github.com/linux-test-project/ltp
[20] Ftrace man page, https://linux.die.net/man/1/ftrace
[21] Docker Seccomp Profile, https://docs.docker.com/engine/security/seccomp/
[22] GVisor Seccomp Rule, https://github.com/google/gvisor/blob/master/runsc/boot/filter/config.go
[23] A. Randazzo, I. Tinnirello, Kata Containers: An Emerging Architecture for Enabling MEC Services in Fast and Secure Way, In Proceedings of the 2019 Sixth International Conference on Internet of Things: Systems, Management and Security (IOTSMS 2019), pp. 209-214, Granada, Spain, October 2019, DOI: 10.1109/IOTSMS48152.2019.8939164

Authors

Jihyeok Yang received B.S. degree in department of information and communication engineering from Yeungnam University, Gyeongsan, Korea, in 2019. Jihyeok Yang is currently taking M.S. course in the School of Computer Science and Engineering, Kyungpook National University. He is interested in cloud computing, operating system and security, and distribution systems.

Byungchul Tak received his B.S. degree from Yonsei University in 2000, M.S. from KAIST in 2003 and Ph.D. degrees in Computer Science and Engineering from the Pennsylvania State University at University Park in 2012. Dr. Tak joined the faculty of the Department of Computer Science at Kyungpook National University, Dague, Korea, in 2017. He is currently an Assistant Professor in the Department of Computer Science. His research interests are in cloud computing, distributed systems, operating system and big data analytics.