УДК 004.457

СТАН СУЧАСНИХ ІНСТРУМЕНТІВ ДЛЯ ПОКРАЩЕННЯ ІЗОЛЯЦІЙНИХ ВЛАСТИВОСТЕЙ КОНТЕЙНЕРІВ

К. О. Заіграєв[1], Л. Ю. Гальчинський[1]

[1] *Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського», Фізико-технічний інститут*

Анотація

У цій науковій статті детально проаналізовано низку програмних застосунків із відкритим кодом, які можна використати для покращення ізоляційних властивостей між контейнером та хостом, у тих випадках якщо операційна система хоста не здатна надати гарні ізоляційні властивості для роботи контейнерів, наприклад якщо ядро ОС хоста має вразливість нульового дня.

Ключові слова: віртуалізація,контейнери,безпека

Вступ

На сьогоднішній час у циклі розробки програмного забезпечення досить активно використовуються контейнери - представкники віртуалізації на рівні операційної системи. Контейнер можна уявляти як спеціальну структуру, що включає в себе бінарний файл програми зі всіма її залежностями та бібліотеками або як самостійний компонент великої програми, на прикладі мікросервісу. Основними перевагами контейнерів є швикда побудова та розгортання на цільовій системі, а також кросплатформенність.

Сучасна технологія контейнеризації побудована за рахунок особливостей ядра Linux, а саме *cgroup, namespace, chroot* та інших. Саме через це контейнер з точки зору ОС виглядає як самостійний процес, який може побачити кожен користувач, просто під'єднавшись до хост системи, у якій запущені контейнери. У порівнянні із віртуальними машинами, у яких за ізоляцію відповідає гіпервізор або VMM на рівні фізичних ресурсів, контейнери зі своєю "ізоляцією"спочатку виглядали досить жалюгідно.

Однак технологія контейнеризації не стояла на місці і за довгий час існування контейнерів було розроблено низку інструментів, що дозволяють значно покращити безпеку та ізоляційні властивості контейнерів.

1. Постановка задачі

Базуючись на знаннях технології контейнеризації має бути запропоновано рішення щодо покращення безпеки контейнерів, що запускаються на хост системі із вразливостями на рівні операційної системи. Для забезпечення цієї цілі було проаналізовано певну кількість застосунків, що мають на меті покращити ізоляційні властивості контейнерів на цільовій ОС.

Об'єктом дослідження є оцінка забезпечення безпеки програмними застосунками, що можуть вдосконалити ізоляційні властивості контейнерів на хост системі.

Предметом дослідження є аналіз досліджуваних застосунків на можливість протидіяти атакам, що використовують вразливості операційної системи, на якій запущені контейнери.

2. Аналіз застосунків

2.1. gVisor

Google gVisor розміщує контейнер у пісочницю, перехоплюючи системні виклики приблизно так само, як гіпервізор перехоплює системні виклики гостьової віртуальної машини.

Згідно з документацією gVisor [1] – є «ядром простору користувача», що реалізує системні виклики контейнерів до ядра Linux через проміжний шар за допомогою паравіртуалізації, тобто через перевстановлення інструкцій таким чином, ніби цього шару не існує.

Для цього компонент gVisor під назвою Sentry перехоплює системні виклики контейнера. Sentry сильно захищений від середовища за допомогою seccomp, так що він не може отримати доступ до ресурсів самої файлової системи. Коли йому потрібно зробити системні виклики, пов'язані з доступом до файлів, він завантажує їх до цілком окремого процесу, який називається Gofer.

Навіть ті системні виклики, які не пов'язані з доступом до файлової системи, не передаються безпосередньо до ядра хоста, а замість цього повторно реалізуються всередині Sentry. По суті, це гостьове ядро, яке працює в просторі користувача.

Проект gVisor представляє собою виконуваний файл, який називається runsc, сумісний із контейнерами формату OCI і який дуже сильно нагадує генератор OCI контейнерів runc. Запуск контейнера за допомогою runsc дозволяє легко побачити процеси

gVisor, але якщо у вас є існуючий файл config.json для runc, вам, доведеться попередньо згенерувати runsc-сумісну версію. У наступному прикладі продемонстровано створення за допомогою gVisor контейнера Alpine:

```
$ skopeo copy \
      docker://alpine:latest \
      oci:alpine:latest
$ sudo umoci unpack \
       --image \
      alpine:latest \
      alpine-bundle
$ cd alpine-bundle
$ runsc spec
$ sudo runsc run sh
```

Як можна було побачити із останньої строчки під час запуску контейнера було викликано командну оболонку sh всередині контейнера. Якби було запущено звичайний контейнер, то ми могли б побачити викликану командну оболонку у лістингу процесів хоста, однак чи виконується це ствердження для gVisor? Спробуємо дізнатись це переглянувши лістинг активних процесів зі сторони хост системи, як показано на прикладі нижче:

```
$ ps fax
# Only processes hierarchy was left
COMMAND
...
|     \_ bash
|        |    \_ sudo runsc run sh
|        |        \_ runsc run sh
|        |          \_ runsc-gofer
|        |          \_ runsc-sandbox
|        |                \_ [exe]
|        |                   \_ [exe]
|        |                   \_ [exe]
|        |                   \_ [exe]
```

Ви можете побачити процес запуску runc, який породив два процеси: один - для Gofer; інший - runc-sandbox, який в документації gVisor згадується як Sentry. Пісочниця має дочірній процес, який, у свою чергу, має ще три дочірніх процеси, однак нам невідомо, що саме за процеси запущені всередині контейнера, бо замість зрозумілих назв ми бачимо невідомі процеси під назвою [exe].

Ця нездатність бачити процеси, що виконуються всередині пісочниці gVisor, набагато більше схожа на поведінку, яку ви бачите у віртуальній машині, ніж у звичайному контейнері. І це забезпечує додатковий захист процесів, що виконуються всередині пісочниці: навіть якщо зловмисник отримує root-доступ на хості, все ще існує відносно сильна межа між хостом і запущеними процесами.

За допомогою gVisor надається можливість протидіяти великій кількості вразливостей, що пов'язані із ядром операційної системи та системними бібліотеками, наприклад Google Cloud [2], використовуючи gVisor змогла покрити вектор атаки, що пов'язаний із вразливістю ядра ОС Linux CVE-2020-1438, яка спочатку заважає вимкненню опції CAP_NET_RAW у версії ядра Linux 5.9-rc4. Однак, навіть якщо цю опцію увімкнено, уразливість не існує для gVisor: проблемний код C у Linux не використовується в мережевому стеці gVisor. Що ще важливіше, такий вид атаки - використання записів, що виходять за межі масиву - не зустрічаються у Sentry та його мережевому стеці завдяки використанню Go.

Хоча ця ізоляція виглядає дуже потужною, є три істотні обмеження:

- По-перше, не всі системні виклики Linux були реалізовані в gVisor. Якщо ваш застосунок хоче використовувати будь-який із нереалізованих системних викликів, він не може працювати в gVisor.
- По-друге - це продуктивність. У багатьох випадках продуктивність дуже близька до тієї, що досягається за допомогою runc, але якщо ваш застосунок робить багато системних викликів, це може вплинути на його ефективність.
- По-третє - на момент написання цієї статті gVisor використовує cgroup першої версії, хоча не всі генератори контейнерів ще перейшли на використання нової - другої - версії, що створює деякі незручності для прогресивного управління ресурсами у порівнянні із cgroup другої версії.

Оскільки gVisor змінює ядро, воно є великим і складним, і ця складність передбачає відносно високий шанс включити деякі власні вразливості.

Як ви бачили в цьому підрозділі, gVisor забезпечує механізм ізоляції, який більше нагадує віртуальну машину, ніж звичайний контейнер. Однак gVisor впливає лише на спосіб доступу програми до системних викликів. Простори імен, cgroups та chroot все ще використовуються для ізоляції контейнера.

2.2. Kata containers

Як було сказано раніше, під час запуску контейнера, генератор контейнерів створює новий процес у хост системі. Ідея Kata контейнерів [3] полягає в тому, щоб запускати контейнери в окремій віртуальній машині. Цей підхід надає можливість запускати контейнери із зображень у форматі OCI з усіма перевагами ізоляції віртуальної машини.

Kata використовує проксі-сервер між середовищем виконання контейнера та окремим цільовим хостом, де працює код програми. Проксі-сервер виконання створює окрему віртуальну машину за допомогою QEMU для запуску контейнера від його імені.

Основним недоліком Kata контейнерів полягає у тому, що вам потрібно почекати, поки віртуальна машина завантажиться. Однак певні групи ентузіастів змогли пришвидшити процес запуску віртуальної машини завдяки видаленню частини інтерфейсу ядра ОС, що витрачає багато часу під час запуску (наприклад лістинг підключенних девайсів), на прикладі Firecracker.

2.3. Unikernels

Операційна система, на яку спираються контейнери, працює в образі віртуальної машини, яку можна використовувати повторно для будь-якої іншої програми, наприклад генератора контейнерів podman. Зрозуміло, що програми навряд чи використовуватимуть усі функції операційної системи. І якби була можливість прибрати частини ОС, що не використовується програмою, тоді поверхня атаки була б набагато меншою у порівнянні зі звичайною ОС.

Ідея Unikernels [4] полягає у створенні спеціального образу віртуальної машини, що складається з програми та частин операційної системи, які потрібні застосунку. Це зображення машини може працювати безпосередньо на гіпервізорі, забезпечуючи той самий рівень ізоляції, що і звичайні віртуальні машини, але з невеликим часом запуску.

Кожна програма повинна бути скомпільована в образ Unikernel, укомплектована всім необхідним для роботи. Гіпервізор може завантажувати цю машину точно так само, як і звичайний образ віртуальної машини Linux.

Проект IBM Nabla використовує методи Unikernel для контейнерів. Контейнери Nabla використовують суворо обмежений набір із семи системних викликів, що забезпечується профілем seccomp. Усі інші системні виклики з програми обробляються в рамках компонента ОС бібліотеки Unikernel. Отримуючи доступ лише до невеликої частини ядра, контейнери Nabla зменшують поверхню атаки. Недоліком є те, що вам потрібно перебудувати ваш програмний застосунок у формат контейнерів Nabla.

Висновки

У даній статті було розглянуто низку інструментів, основним завданням яких є:

- покращення віртуалізації між контейнерами та хост системою, на якій працюють контейнери за рахунок використання додаткого шару віртуалізації, який у випадку вдалої компрометації хосту або контейнера не дасть можливості хакеру взаємодіяти із іншими контейнерами, які розташовані поруч;
- мінімізація ризиків та поверхні атаки не тільки на систему контейнерів, де працює програмний застосунок, а й хост систему.

В результаті проведеного дослідження, можна дійти висновку, що найкращим варіантом є використання інфраструктурного рішення, що використовує Kata контейнери, які базуються на віртуальних машинах із операційною системою, побудованою на Unikernels. Подане рішення є найбільш ефективним, тому що у порівнянні із gVisor підтримка не всіх системних викликів, а також швикдодія є доволі важливими пунктами під час розробки та поставки програмного забезпечення на одному рівні із бепезкою.

Перелік використаних джерел

1. gVisor documentation. — Access mode: `https://gvisor.dev/docs/`.
2. Eric Brewer gVisor: Protecting GKE and serverless users in the real world— 2020 — `https://cloud.google.com/blog/products/containers-kubernetes/how-gvisor-protects-google-cloud-services-from-cve-2020-1438`.
3. Lize Rice Container Security. Fundamental Technology Concepts that Protect Containerized Application. — 2019 — Access mode: `https://www.oreilly.com/library/view/container-security/9781492056690/`.
4. Ali Raza, Parul Sohal, James Cadden, Jonathan Appavoo, Ulrich Drepper, Richard Jones, Orran Krieger, Renato Mancuso, Larry Woodman Unikernels: The Next Stage of Linux's Dominance — 2019 — `https://www.cs.bu.edu/~jappavoo/Resources/Papers/unikernel-hotos19.pdf`.