УДК 004.42
DOI: 10.31673/2412-9070.2021.052933

О. Б. ПРИДИБАЙЛО, здобувач,
І. В. ЗАМРІЙ, канд. фіз.-мат. наук, доцент;
А. Г. ЗАХАРЖЕВСЬКИЙ, канд. техн. наук;
К. В. ПОЛОНСЬКИЙ, аспірант,
Державний університет телекомунікацій, Київ

АНАЛІЗ БЕЗПЕКИ СЕРВІСНО-ОРІЄНТОВАНОГО ПРОГРАМУВАННЯ

За останні кілька років стрімко зросло використання технологій віртуалізації. Тому потреба в ефективних і безпечних вирішеннях для віртуалізації стає дедалі очевиднішою. Віртуалізація на основі контейнерів та віртуалізація на основі гіпервізора — це два головних типи технологій віртуалізації, що з'явилися на ринку. З цих двох класів віртуалізація на основі контейнерів може забезпечити більш легке та ефективне віртуальне середовище, але не без проблеми безпеки. У статті проаналізовано рівень безпеки сервісно-орієнтованого програмування, яке базується на контейнеризації застосунків. Розглянуто дві сфери: внутрішню безпеку сервісно-орієнтованого програмування та його взаємодію з функціями безпеки ядра Linux, зокрема SELinux та AppArmor, для посилення захисту хост-системи.

Аналіз показав, що сервісно-орієнтоване програмування забезпечує високий рівень ізоляції та обмеження ресурсів для своїх контейнерів за допомогою простору імен, контрольних груп та файлової системи копіювання під час запису, навіть якщо конфігурація за замовчуванням. Воно також підтримує кілька функцій безпеки ядра, які уможливлюють посилення безпеки хоста. Єдина проблема, яку виявлено в сервіс-орієнтованому програмуванні, була пов'язана з його мережною моделлю за замовчуванням. Віртуальний ethernet-міст, який сервісно-орієнтоване програмування використовує як свою мережну модель за замовчуванням, уразливий до ARP-спуфінгу та атак заповнення MAC, оскільки він не забезпечує жодного фільтра мережного трафіку, що проходить через міст. Однак цю проблему можна вирішити, якщо адміністратор вручну додасть до мосту фільтрацію, наприклад ebtables, або змінить мережне підімкнення на більш безпечне, скажімо віртуальну мережу.

Також варто зауважити, що якщо оператор запускає контейнер як «привілейований», СОП надає повний доступ до контейнера, який майже такий самий, як і для процесів, запущених на хості. Тому безпечніше експлуатувати контейнери як «непривілейовані». Крім того, незважаючи на те, що контейнери можуть забезпечити більшу щільність віртуальних середовищ і кращу продуктивність, вони мають більшу поверхню атаки, ніж віртуальні машини, оскільки контейнери можуть безпосередньо спілкуватися з ядром хоста. Проте можна зменшити поверхню атаки, зберігаючи ці переваги. Наприклад, цього можна досягти, розмістивши контейнери всередині віртуальних машин.

Ключові слова: сервісно-орієнтоване програмування; контейнеризація; віртуалізовані застосунки; віртуальна машина; сервер; контейнер; простір імен; контрольні групи.

Вступ

Останнім десятиліттям спостерігається бурхливе зростання технологій віртуалізації, які дають можливість розділити комп'ютерну систему на кілька ізольованих віртуальних середовищ. Ці технології пропонують істотні переваги, що посприяли їхньому швидкому розвитку. Одна з найчастіших причин упровадження технологій віртуалізації — віртуалізація серверів у центрах оброблення даних. За допомогою віртуалізації серверів адміністратор може створити один або кілька екземплярів віртуальної системи на одному сервері. Ці віртуальні системи працюють як реальні фізичні сервери та можуть бути здані в оренду за підпискою. Amazon EC2, Rackspace та DreamHost — це лише деякі популярні приклади таких постачальників послуг для центрів оброблення даних. Інше поширене використання — віртуалізація робочих столів, коли на одному комп'ютері може працювати кілька екземплярів операційної системи. Віртуалізація робочого стола забезпечує підтримання застосунків, які можуть працювати лише в певній операційній системі.

Зростання використання технологій віртуалізації підвищує попит на вирішення віртуалізації, які можуть забезпечити щільні, масштабовані та безпечні для користувача середовища.

Основна частина

Сьогодні на ринку з'явилося багато вирішень для віртуалізації. Більшість технологій віртуалізації можна поділити на два основних підходи: віртуалізацію на основі контейнерів та віртуалізацію з урахуванням гіпервізора. Перший забезпечує віртуалізацію лише на рівні операційної системи, а другий — віртуалізацію лише на рівні устаткування. Віртуалізація на основі контейнерів може задовольнити більш легке та ефективне віртуальне середовище. Це дає змогу в десять разів більше віртуальних середовищ працювати на фізичному сервері, порівняно з віртуалізацією на основі гіпервізора [1]. Кожен із підходів має свої переваги та недоліки, які будуть розглянуто далі.

Віртуалізація на основі контейнерів — це спрощений підхід до віртуалізації, який використовує ядро хоста для запуску кількох віртуальних середовищ. Ці віртуальні середовища часто називають контейнерами. Linux-VServer, OpenVZ та Linux Container (LXC) — три основні представники цього підходу. Загальну архітектуру вирішення віртуалізації з урахуванням контейнерів зображено на рис. 1.

Рис. 1. Архітектура контейнерної віртуалізації

Віртуалізація на ***основі контейнерів*** здійснюється на робочому системному рівні, що дає змогу працювати з кількома застосунками без надлишкового запуску інших ядер операційних систем на хості. Його контейнери зовні мають вигляд звичайних процесів, що працюють поверх ядра, спільно із застосовуваним хостом комп'ютера. Вони надають ізольовані середовища з необхідними ресурсами для виконуваних застосунків. Ці ресурси можуть бути або спільно використовуваними з хостом, або встановлюватися окремо всередині контейнера.

Вирішення віртуалізації на ***основі гіпервізора*** забезпечують віртуалізацію на апаратному рівні. На противагу віртуалізації на основі контейнерів гіпервізор установлює повні віртуальні машини (ВМ) поверх операційної системи хоста (рис. 2).

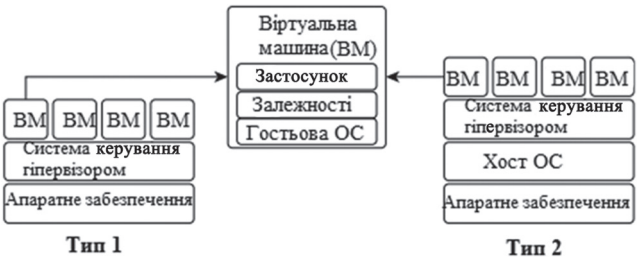

Рис. 2. Архітектура віртуалізації на основі гіпервізора

Кожна віртуальна машина складається не тільки з програми та її залежностей, а й з гостьової ОС повністю з окремим ядром. Існує два класи гіпервізорів: гіпервізор типу 1, також відомий як гіпервізор без оболонки, який працює безпосередньо поверх обладнання, що лежить в основі хоста, та гіпервізор типу 2, також відомий як розміщений гіпервізор, який працює поверх операційної системи хоста. Xen є прикладом першого типу гіпервізора, а KVM — другого. Оскільки гіпервізор 1-го типу не містить додаткового рівня ОС хоста, він забезпечує кращу продуктивність, ніж гіпервізор 2-го типу.

Відмінності в архітектурі надає деякі переваги віртуалізація на основі контейнерів порівняно з віртуалізацією на основі гіпервізора. По-перше, віртуалізація на основі контейнерів може забезпечити більшу щільність віртуальних середовищ. Оскільки контейнер не охоплює всю ОС, розмір і потрібні ресурси для запуску програми в контейнері менші за саму віртуальну машину, на якій запущено ту саму програму. У результаті на віртуальних машинах можна розгорнути більшу кількість контейнерів. По-друге, віртуалізація на основі контейнерів також пропонує найкращу продуктивність. Це було продемонстровано експериментами в деяких дослідженнях, які показали, що здебільшого продуктивність віртуалізації на основі контейнерів вища порівняно з віртуалізацією на основі гіпервізора [4].

Однак, незважаючи на всі зазначені переваги, віртуалізація на основі контейнерів не може підтримувати різні середовища так, як це робить віртуалізація на основі гіпервізора, оскільки всі середовища контейнерів мають бути того самого типу, що й хост. Наприклад, контейнери Windows не можна запускати поверх хоста Linux.

Сервісно-орієнтоване програмування (СОП) — це контейнерна технологія, яка здатна «створювати, постачати та запускати розподілені програми» [1]. Воно використовується в деяких популярних програмах, зокрема Spotify, Yelp і Ebay.

Хоча контейнерні технології існують уже понад 10 років, сервісно-орієнтоване програмування нині є однією з найуспішніших технологій, оскільки воно має такі нові можливості, яких раніше не було:

♦ надає інтерфейси для простого та безпечного створення контейнерів та керування ними;

♦ розробники можуть запаковувати програми в легкі контейнери, які матимуть змогу працювати практично будь-де без змін. Крім того, СОП може розгорнути більше віртуальних середовищ, ніж інші технології, працюючи на тому самому обладнанні [3];

♦ добра взаємодія зі сторонніми інструментами, які спрощують процес керування та розгортання контейнерів.

Інструменти DevOps, зокрема Puppet, An sible та Vagrant, можуть інтегруватися з сервісно-орієнтованим програмуванням, що спрощує розгортання його контейнерів у хмарі. До того ж, багато інструментів оркестрування, такі як Mesos, Shipyard та Kubernetes також підтримують контейнери СОП. Ці інструменти забезпечують абстрактний рівень керування ресурсами та планування.

Сервісно-орієнтоване програмування має у своєму складі два основних компоненти: Docker Engine та Docker Hub. Перший— це віртуалізація з відкритим вихідним кодом вирішення, а другий є платформою «Програмне забезпечення як послуга» для обміну образами.

Рушій Docker — це легкий та портативний інструмент для пакування [1], який ґрунтується на

віртуалізації на основі контейнерів. Отже, його архітектура (рис. 3) аналогічна архітектурі контейнерної віртуалізації загалом. Контейнери сервісно-орієнтованого програмування працюють поверх демона, який відповідає за виконання та керування всіма контейнерами. Клієнт, який надає користувачам інтерфейс для взаємодії з контейнерами, приймає команди від користувачів, а потім відправляє їх демону через RESTful API. Використання цього методу зв'язку дає можливість клієнту працювати на тому самому хості, що і контейнери, або навіть на різних хостах.

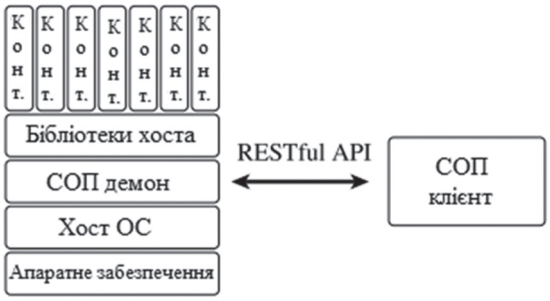

Рис. 3. Архітектура рушія

Однією з основних проблем на етапі запуску сервісів у віртуальних середовищах, насамперед у передбаченій на багато користувачів хмарній системі, є безпека. Віртуальні машини, що надаються методами віртуалізації на основі гіпервізора, вважаються безпечнішими за контейнери, оскільки вони додають додатковий рівень ізоляції між застосунками та хостом. Програма, яка працює всередині віртуальної машини, може взаємодіяти лише з ядром віртуальної машини, але не з ядром хоста. Отже, щоб програма вийшла з віртуальної машини, вона має обійти ядро віртуальної машини та гіпервізор, перш ніж зможе атакувати ядро хоста. Проте контейнери можуть безпосередньо зв'язуватися з ядром хоста, що дає можливість зловмиснику значно заощадити сили під час злому хост-системи. Це спричинює занепокоєння щодо безпеки контейнерів. Для того аби протистояти зазначеним атакам, вирішення віртуалізації на рівні ОС має задовольняти такі вимоги: ізоляція процесу, ізоляція файлової системи, ізоляція пристроїв, ізоляція IPC, ізоляція мережі та обмеження ресурсів [2].

Розглянемо ці вимоги детальніше.

1. ***Ізоляція процесу***. Основна мета ізоляції процесів — запобігти використанню скомпрометованих контейнерів інтерфейсів керування процесами для створення завад іншим контейнерам. Сервісно-орієнтоване програмування досягає ізоляції процесів, беручи процеси, запущені в контейнерах, у простори імен і обмежуючи їхні дозволи та видимість процесами, що запущені в інших контейнерах і на базовому хості. Цей механізм працює за допомогою просторів імен PID, які ізолюють простір ідентифікаторів процесів контейнера від простору імен хоста. Оскільки простори імен PID є ієрархічними [3], процес може бачити інші процеси лише у власному просторі імен або у своїх «дочірніх» просторах імен. Простір імен PID також дозволяє кожному контейнеру мати свій власний процес, подібний до процесу ініціалізації (PID 1), який призводить до завершення всіх процесів у просторі імен, якщо він завершується. Цей процес допомагає адміністратору повністю закрити контейнер у разі виявлення чогось підозрілого.

2. ***Ізоляція файлової системи***. Щоб досягти такої ізоляції, файлові системи хоста та контейнери мають бути захищені від несанкціонованого доступу та модифікації. Сервісно-орієнтоване програмування використовує простори імен монтування, також звані просторами імен файлових систем для ізоляції ієрархії файлової системи, пов'язаної з різними контейнерами. Простір імен монтування надає процесам кожного контейнера різне уявлення дерева файлової системи та обмежує всі події монтування, що відбуваються всередині контейнера, впливом тільки всередині контейнера.

3. ***Ізоляція пристрою***. У Unix ядро та програми отримують доступ до обладнання через вузли пристроїв, які переважно є спеціальними файлами, що виступають як інтерфейси для драйверів пристроїв. Якщо контейнер може отримати доступ до деяких важливих вузлів пристрою, зокрема /dev/mem (фізична пам'ять), /dev/sd* (сховище) або /dev/tty (термінал), він може серйозно пошкодити хост-систему. Важливо обмежити набір вузлів пристроїв, до яких контейнер може отримати доступ. Крім того, СОП монтує образи контейнерів за допомогою вузла. Тобто, навіть якщо вузол пристрою було попередньо створено всередині образу, процеси в контейнері, які використовують зображення, не можуть застосовувати його для зв'язку з ядром. За замовчуванням, СОП не дає розширених привілеїв своїм контейнерам. Отже, вони не можуть отримати доступ до будь-яких пристроїв. Однак, якщо оператор вважає контейнер «привілейованим», СОП надає доступ до всіх пристроїв контейнера.

4. ***Ізоляція міжпроцесної взаємодії***. Міжпроцесна взаємодія (МВ) — це набір об'єктів для обміну даними між процесами, такими як семафори, черги повідомлень і сегменти розподільної пам'яті. СОП призначає простір імен МВ кожному контейнеру, водночас запобігаючи взаємодії процесів у контейнері з процесами інших контейнерах.

5. ***Мережна ізоляція***. Мережна ізоляція важлива для запобігання мережним атакам, зокрема Man-in-the-Middle (MitM) та ARP-спуфінг. Контейнери мають бути налаштовані в такий спосіб,

щоб вони не мають змоги підслуховувати або маніпулювати мережним трафіком інших контейнерів та хоста. Для кожного контейнера СОП створює незалежний мережний стек, використовуючи мережні простори імен. Отже, кожен контейнер має свої IP-адреси, таблиці IP-маршрутизації, мережні пристрої тощо.

6. ***Обмеження ресурсів.*** Відмова в обслуговуванні (DoS) — одна з найпоширеніших атак на систему, розраховану на велику кількість користувачів, коли процес або група процесів намагаються використовувати всі ресурси системи, порушуючи нормальну роботу іншого процесу. Щоб запобігти цьому виду атаки, має бути можливість обмежити ресурси, котрі виділяються кожному контейнеру. Контрольні групи — ключовий компонент, який СОП використовує для вирішення цієї проблеми. Вони контролюють кількість ресурсів, зокрема ЦП, пам'ять та дискове введення-виведення, які може застосовувати будь-який контейнер, гарантуючи, що кожен контейнер отримує свою справедливу частку ресурсів і запобігаючи використанню всіх ресурсів будь-яким контейнером. Вони також дають змогу налаштовувати ліміти та обмеження, пов'язані з ресурсами, виділеними кожному контейнеру. Наприклад, одним із таких обмежень є обмеження кількості доступних ЦП для певного контейнера [5].

Висновки

Отже, віртуалізація на основі контейнерів може забезпечити віртуальні середовища з більшою щільністю та кращу продуктивність, ніж віртуалізація на основі гіпервізора. Проте останній вважається більш безпечним, ніж перший. Сервісно-орієнтоване програмування є однією з найпопулярніших технологій віртуалізації на основі контейнерів. Аналіз показує, що контейнери СОП є досить безпечними, навіть якщо запускається конфігурація за замовчуванням. Рівень безпеки контейнерів СОП також можна підвищити, якщо оператор запускає їх як «непривілейований» і дозволяє використовувати додаткові вирішення для зміцнення в ядрі Linux, наприклад AppArmor або SELinux.

Список використаної літератури

1. ***Burniske C.*** *Containers: The next generation of virtualization? [Електронний ресурс]. URL: http://ark-invest.com/webx0/ containers-next-generation-virtualization. [Accessed 22 November 2014].*

2. ***Security*** *of OS-level virtualization technologies / E. Reshetova, J. Karhunen, T. Nyman and N. Asokan // Proceedings of the 2014 NordSec Conference. Norway, 2014. P. 77–93.*

3. ***Милл И., Хобсон Сейерс Э.*** *Docker на практике / пер. с англ. Д. А. Беликов. Москва: ДМК Пресс, 2020. 516 с.*

4. ***Regola N., Ducom J.-C.*** *Recommendations for virtualization technologies in high performance computing // IEEE Second International Conference on Cloud Computing Technology and Science (CloudCom). Nov. 2010. P. 409–416.*

5. ***Моуэт Э.*** *M89 Использование Docker / пер. с англ. А. В. Снастина; науч. ред. А. А. Маркелов. Москва: ДМК Пресс, 2017. 354 с.*

О. Б. Придыбайло, И. В. Замрий, А. Г. Захаржевский, К. В. Полонский

АНАЛИЗ БЕЗОПАСНОСТИ СЕРВИСНО-ОРИЕНТИРОВАННОГО ПРОГРАММИРОВАНИЯ

За последние несколько лет стремительно возросло использование технологий виртуализации. Поэтому потребность в эффективных и безопасных решениях для виртуализации становится все более очевидной. Виртуализация на основе контейнеров и виртуализация на основе гипервизора — это два основных типа появившихся на рынке технологий виртуализации. Из этих двух классов виртуализация на основе контейнеров может обеспечить более легкую и эффективную виртуальную среду, но не без проблем безопасности. В статье анализируется уровень безопасности сервисно-ориентированного программирования, основанного на контейнеризации приложений. Рассмотрим две области: внутреннюю безопасность сервисно-ориентированного программирования и то, как она взаимодействует с функциями безопасности ядра Linux, такими как SELinux и AppArmor, для усиления защиты хост-системы.

Анализ показывает, что сервисно-ориентированное программирование обеспечивает высокий уровень изоляции и ограничения ресурсов для своих контейнеров с помощью пространства имен, контрольных групп и файловой системы копирования при записи, даже если конфигурация по умолчанию. Оно также поддерживает несколько функций безопасности ядра, помогающих усилить безопасность хоста. Единственная проблема, обнаруженная в сервис-ориентированном программировании, была связана с его сетевой моделью по умолчанию. Виртуальный ethernet-мост, который сервисно-ориентированное программирование использует как свою сетевую модель по умолчанию, уязвим к ARP-спуфингу и атакам заполнения MAC, поскольку он не обеспечивает ни одного фильтра сетевого трафика, проходящего через мост. Однако эту проблему можно решить, если администратор вручную добавит к мосту фильтрацию, например ebtables, или изменит сетевое подключение на более безопасное, скажем виртуальную сеть.

Также следует подчеркнуть, что если оператор запускает контейнер как «привилегированный», СОП предоставляет полный доступ к контейнеру, который почти такой же, как и для процессов, запущенных на хосте. Поэтому безопаснее эксплуатиро-

вать контейнеры как «непривилегированные». Кроме того, несмотря на то, что контейнеры могут обеспечить большую плотность виртуальных сред и лучшую производительность, они обладают большей поверхностью атаки, чем виртуальные машины, поскольку контейнеры могут непосредственно общаться с ядром хоста. Однако можно снизить поверхность атаки, сохраняя эти преимущества. К примеру, этого можно добиться путем размещения контейнеров внутри виртуальных машин.

Ключевые слова: сервисно-ориентированное программирование; контейнеризация; виртуализированные приложения; виртуальная машина; сервер; контейнер; пространство имен; контрольные группы.

O. B. Prydybailo, I. V. Zamrii, A. G. Zakharzhevskyi, K. V. Polonskyi

SAFETY ANALYSIS OF SERVICE-ORIENTED PROGRAMMING

Over the last few years, the use of virtualization technologies has increased dramatically. Therefore, the need for effective and secure virtualization solutions is becoming increasingly apparent. Container-based virtualization and hypervisor-based virtualization are the two main types of virtualization technologies that have appeared on the market. Of these two classes, container-based virtualization can provide a lighter and more efficient virtual environment, but not without security issues. This article analyzes the security level of service-oriented programming, which is based on application containerization. Let's look at two areas: the internal security of service-oriented programming and how it interacts with Linux kernel security features such as SELinux and AppArmor to enhance host security. The analysis shows that service oriented programming provides a high level of isolation and resource limiting for its containers using namespaces, cgroups, and its copy-on-write file system, even with the default configuration. It also supports several kernel security features, which help to hardening the security of the host. The only problem we found with service oriented programming was related to its default networking model. The virtual ethernet bridge which service oriented programming uses as its default networking model, is vulnerable to ARP spoofing and MAC flooding attacks since it does not provide any filter on the network traffic passing through the bridge. However, this problem can be solved if the administrator manually adds filtering, such as ebtables, to the bridge, or changes the networking connectivity to a more secure one, such as virtual network. It is also worth highlighting that if the operator runs a container as «privileged», service oriented programming grants full access permissions to the container, which is nearly the same as that of processes running natively on the host. Therefore, it is more secure to operate containers as «non-privileged». Furthermore, even though containers can provide higher density of virtual environments and better performance, they have a bigger attack surface than virtual machines since containers can directly communicate with the host kernel. However, it is possible to reduce the attack surface while maintaining these advantages. For example, this can be achieved by placing containers inside virtual machines.

Keywords: service-oriented programming; containerization; virtualized applications; virtual machine; server; container; namespace; control groups.