УДК 004.056

ВИЯВЛЕННЯ БЕЗПЕКОВИХ ПРОБЛЕМ В ДОКЕРФАЙЛАХ

М. В. Супрун[1], Л. Ю. Гальчинський[1]

[1] Навчально-науковий Фізико-технічний інститут

Анотація

Контейнеризація є дуже популярним та ефективним підходом в розробці застосунків, метою якого є портативність, масштабованість та безпека. Яскравим представником даної технології є Docker. Питання безпеки є актуальним для даної технології і потребує комплексного підходу. Особливої увагу заслуговують підходи для забезпечення безпеки конфігураційних файлів контейнерів – докерфайлів. Вони можуть часто містити в собі безпекові місконфігурації, що в подальшому можуть призводити до компроментації всього контейнеру.

Ключові слова: докерфайл, security smell, code smell, безпекова місконфігурація

Вступ

Dockerfile – це текстовий файл, який містить інструкції для автоматизованої конфігурації образів Docker. Ці інструкції вказують, як створити образ, який можна потім використовувати для запуску контейнерів Docker. Для даної технології характерна наявність так званих code smell. Code smell є індикатором наявності потенційних проблем в програмному проекті. Їх підмножиною є security smell. Security smell – це повторювані шаблони коду, які вказують на безпекові слабкості, які потенційно можуть привести до порушень безпеки та потребують додаткової перевірки.

1. Аналіз проблеми

Поглиблений аналіз джерел поширеності security smell в докерфайлах, зокрема [1] показав наступне:

- Майже 98% докерфайлів, розміщених в публічних репозиторіях звичайних проектів в тій чи іншій мірі містять security smell.
- Докерфайли, які були зібрані з проектів docker-library та відносяться до категорії Docker Official Images мали б відповідати вищим стандартам безпеки. Однак вони виявилися все ще в значній мірі уражені security smell, хоч і в меншій кількості – 81%

З цього випливає, що проблема присутності security smell у докерфайлах з різнем ступенем якості є гострою. Розробники масово необізнані в безпечних практиках написання докерфайлів. Далі постає задача дослідити які бувають security smell, їх природу та вплив на безпеку.

2. Існуючі Security smell в докерфайлах

Було проаналізовано існуючі security smells в докерфайлах, їх наведено в переліку нижче. Серед значного корпусу досліджень найбільш повний аналіз подано [1], [2], що показано в наступному переліку нижче.

1) **Використання apt-get update поодинці**

   Якщо використовувати apt-get update в окремій інструкції, то це призводить до створення кешованого шару, який використовується при збиранні Docker образу. Таким чином останні оновлення для пакетів, які можуть містити безпекові патчі можуть не потрапити до образу при збірці. Тому рекомендовано впевнитися що інструкція apt-get update використовується в парі з іншими інструкціями, наприклад apt-get install

2) **Невикористання прапору –no-install-recommends**

   Без цієї опції непотрібні apt пакети для Docker образу будуть додані під час інсталяції. Це може збільшити поверхню атаки Docker образу. За замовчуванням apt-get install завантажує окрім основних залежностей, ще й рекомендовані (допоміжні). Ця опція вказує менеджеру пакетів не встановлювати рекомендовані залежності для вказаних пакетів.

3) **Не закріплена версія пакету**

   Встановлення пакетів без закріплення їх версії спричиняє схожі проблеми як і security smell «Використання apt-get update поодинці». Закріплення версії допомагає змусити збірку докерфайлу отримати конкретні версії пакету незалежно від кешу в попередніх Docker шарах. Даний security smell об'єднує в собі чотири види пакетів, для яких може бути незакріплена версія, а саме:

   - apt пакети
   - apk пакети
   - pip пакети

- npm пакети
- gem пакети

4) **Використання ADD замість COPY**

ADD інструкції можуть скачувати і розпаковувати віддалені файли за певних URLs без якої-небудь перевірки, що потенційно приносить безпекові ризики. ADD інструкції мають бути замінені еквівалентними командами wget або COPY. Інструкція COPY просто копіює файли з локального хосту до файлової системи контейнеру без автоматичного розархівування. Якщо необережно використовуючи інструкцію ADD, ми можемо завантажити, розпакувати небажані файлі без попереднього сканування, які можуть виявитися зловмисними, або вразливими.

5) **Останній користувач – root**

Запуск контейнеру під привілейованим користувачем може дозволити атаку на ескалацію привілеїв. Таким чином, рутовий користувач має бути замінений на нерутового користувача в докерфайлах. Щоб не допустити появу даної місконфігурації варто явно визначати користувача контейнеру за допомогою інструкції USER. Після цього контейнер буде по-замовчуванню працювати від імені користувача визначеного в цій директиві. Без даної директиви користувач за замовчуванням – root.

6) **Наявні секрети**

Секрети, які зберігаються в докерфайлах можуть призводити до великого ризику, так як вони передаються до Docker образу під час збірки і є видимими для кожного користувача образу. Навіть якщо на якомусь з кроків секрет був видалений, Docker зберігає історію по всіх шарах і зловмисник зможе за допомогою деяких команд переглянути початкові шари, де був використаний секрет. Альтернативною може бути використання таких технік як Docker secrets. Docker secrets надає можливість централізовану керувати секретами і безпечно їх доставляти до тих контейнерів, які їх потребують.

7) **Відсутня інструкція HEALTHCHECK**

Команда HEALTHCHECK має бути включена для моніторингу здоров'я Docker контейнеру, це відповідає безпековому контролю – доступності. Тому рекомендується щоб інструкція HEALTHCHECK була додана до образу контейнера.

8) **Використання «latest» тегу в базовому образі**

Якщо покладатися на тег «latest», це може призвести до неявного успадкування оновлених пакетів які у гіршому випадку негативно вплинуть на надійність вашого контейнеру, а в гіршому випадку будуть містити вразливості в собі. Рекомендовано конкретно зазначаємо версію базового образу у форматі major.minor.

9) **Відкритий 22 порт**

Відкриті порти в контейнері є відкритими дверима до системи. Рекомендується відкривати лише необхідні порти, які потрібна для коректної роботи системи і уникати відкриття 22 порту. Варто зазначити, що хоч докерфайл дає можливість визначити відкритий порт за допомогою команди EXPOSE, ця команда є лише інформативною і використовується в цілях документування. Відкриття портів відбувається явно під час запуску контейнера і використовує іншу команду, для прикладу:

```
docker run --publish-all
```

2.0.1. Додаткові безпекові рекомендації

Тепер маючи знання про існуючі security smell в докерфайлах – їх легко можна ідентифікувати, а значить і уникнути.
Окрім цього було вирішено копнути глибше і знайти спосіб зробити контейнери ще безпечнішими при роботі з докерфайлами. Тому було проаналізовано практики від CIS Docker Benchmark [3] та OWASP [4], а також інші менш відомі ресурси. Таким чином було визначено перелік рекомендацій, виконання яких додатково покращить безпеку контейнеру.

- Впевніться, що контейнер використовує лише довірений base image
- Впевніться, що setuid та setgid дозволи видалено
- Впевніться, що тільки верифіковані пакети встановлені
- Уникайте curl bashing в RUN директиві
- Зробіть виконувані файли власником адміністратора та недоступними для запису
- Використовуйте багатоетапні збірки
- Використовуйте базові образи – Distroless або з нуля

3. Аналіз інструментів для виявлення безпекових проблем

Маючи ці два переліки постала задача перевірити, які популярні інструменти для безпеки Docker здатні виявляти ці проблеми. Перелік security smell та перелік додаткових рекомендацій були використані як критерії для оцінки інструментів. Тобто було проаналізовано чи може інструмент виявляти конкретний security smell або недотримання рекомендації. Наступні інструменти були оцінені:

- Dockle,
- Hadolint,
- Docker Bench for Security.

Фаворитом виявився Dockle, який міг виявляти 8 з 16 безпекових проблем і набрав найбільше балів серед проаналізованих інструментів відповідно даної методики.

Друге місце посів Haolint, який міг виявляти 6 з 16 безпекових проблем, і відповідно третє – Docker Bench for Security, який міг виявляти лише 5 з 16 безпекових проблем, зокрема це зумовлено тим, що даний інструмент призначений для більш ширшого спектру первірок і кількість перевірок Docker, специфічних для докерфайлів є малою.

Але очевидно, що для забезпечення достатнього рівня безпеки докерфайлу та виявлення повної множини безпекових проблем (ті які були розглянуті) цього недостатньо. Задля цього було вирішено створити власний інструмент.

4. Запропонований алгоритм

У відкритому доступі є дані про розробки пропрієтарних програм для технологій докерфайлів пошуку, зокрема [5]. Проте відсутні дані про такі програми для виявлення проблем з security smell.

Алгоритм даної задачі можна умовно поділити на дві частини:

- парсинг докерфайлу з метою його структуризації
- застосування правил для виявлення місконфігурацій поверх отриманої структури.

Першим та ключовим кроком є структуризація докерфайлу. Оскільки докерфайл містить вбудовані shell сценарії в деяких директивах – їх треба коректно відокремити. Без цього кроку аналіз докерфайлу методом звичайного проходження зверху до низу буде неефективний, хоча цілком можливий. У структурованому представленні докерфайлу легше навігуватися та швидше знаходити необхідні патерни, особливо шукаючи патерни в shell сценаріях.

Один з можливих методів досягнення цього є побудова абстрактного синтаксичного дерева. Абстрактне синтаксичне дерево (AST) – це графове представлення вихідного коду, зазвичай використовується компіляторами для читання коду та побудови двійкових файлів. Формально AST це промарковане напрямлене дерево з коренем. Конвертація вихідного коду в AST здійснюється за допомогою певного парсера, який аналізує кожну специфічну конструкцію у коді і добудовує нові вузли з них..

Другим кроком є визначення правил, які будуть застосовані для виявлення місконфігурацій. Правила – це шаблон, патерн, який має детектуватися в докерфайлі і свідчити про наявність місконфігурації або невиконання певної рекомендації. Набір цих правил може бути сформований з переліку існуючих security smell, а також переліку додаткових безпекових рекомендацій.

Висновки

Було проаналізовану проблему security smell в докерфайлах. Стало зрозуміло, що дане питання є поширеним та критичним. При створенні докерфайлів розробниками недостатньо уваги приділяється безпековій складовій. Щоб глибше проаналізувати проблему було розглянуто перелік існуючих security smell, а також додатково визначено рекомендації для підвищення рівня безпеки докерфайлу. Після чого постала задача їх виявлення. З проведеної оцінки популярних інструментів зроблено висновок, що універсального і в той же час ефективного інструменту який би виявляв широкий спектр security smell та невиконання безпекових рекомендацій немає.

Тому постала задача створення власного інструменту, задля чого було запропоновано потенційний алгоритм його роботи. Метою інструменту має бути ефективне виявлення повної множини безпекових проблем, які було розглянуто.

Перелік використаних джерел

1. *Giorgi L. A. D.* Security Misconfigurations Detection and Repair in Dockerfile // Politecnico di Torino, Corso di laurea magistrale in Ingegneria Informatica (Computer Engineering). — 2022. — URL: https://webthesis.biblio.polito.it/secure/23448/1/tesi.pdf.
2. *Quang-Cuong Bui Malte Laukötter R. S.* DOCKERCLEANER: Automatic Repair of Security Smells in Dockerfiles. — 2023. — DOI: 10.1109/ICSME58846.2023.00026. — URL: https://cuong.buiquang.com/papers/icsme23.pdf.
3. *CIS.* CIS Docker Benchmark. — URL: https://www.cisecurity.org/benchmark/docker.
4. *OWASP.* Docker Security Cheat Sheet. — 2024. — URL: https://cheatsheetseries.owasp.org/cheatsheets/Docker_Security_Cheat_Sheet.html.
5. Комп'ютерна програма «Docker Security Beef Up»(«DSBU»). Авторське право і суміжні права, бюлетень № 71, 2022 с.261 : 113416 / В. В. Кравченко, Л. Ю. Гальчинський. — 23.06.2022.