НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
НАВЧАЛЬНО-НАУКОВИЙ ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ
Кафедра інформаційної безпеки

До захисту допущено
Завідувач кафедри

_____ **Дмитро ЛАНДЕ**
(підпис)

«_____»_____2024 р

Магістерська дисертація

на здобуття ступеня магістра

за освітньо-професійною програмою
«Системи, технології та математичні методи кібербезпеки»
зі спеціальності 125 «Кібербезпека»

на тему: Методи підвищення захисту в Kubernetes

Виконав (-ла): здобувач вищої освіти **II** курсу, групи ФБ-21мп
(шифр групи)

Зеленін Владислав Юрійович
(прізвище, ім'я, по батькові) (підпис)

Керівник к.т.н, доцент кафедри ІБ, Родіонов Андрій Миколайович ____
(посада, науковий ступінь, вчене звання, прізвище, ім'я, по батькові) (підпис)

Рецензент ст. викладач каф. ММСА, к.т.н. Савастьянов В.В. _____
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище, ім'я, по батькові) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Здобувач вищої освіти _____
(підпис)

Київ – 2024 року

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
НАВЧАЛЬНО-НАУКОВИЙ ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ
Кафедра інформаційної безпеки

Рівень вищої освіти – другий (магістерський)
Спеціальність – 125 «Кібербезпека та захист інформації»
Освітньо-професійна програма «Системи, технології та математичні методи кібербезпеки»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Дмитро ЛАНДЕ
(підпис)
«_____»_____2024 р.

ЗАВДАННЯ
на магістерську дисертацію здобувачу ступеня магістра

Зеленіну Владиславу Юрійовичу
(прізвище, ім'я, по батькові)

1. Тема дисертації: Методи підвищення захисту в Kubernetes,
Науковий керівник дисертації роботи: Родіонов Андрій Миколайович, к.т.н, доцент кафедри інформаційної безпеки,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від 11 листопада 2023р. № 5236-с

2. Термін подання здобувачем вищої освіти роботи 08 січня 2024 р.

3. Предмет дослідження: безпека платформи оркестрації Kubernetes

4. Вихідні дані до роботи: технічна документація, теоретичні дані, розроблена утиліта

5. Завдання дослідження: 1. Аналіз існуючих методів підвищення захисту в Kubernetes; 2. Визначення переваг і недоліків цих засобів; 3. Пропозиція вирішення проблеми неактуальної інформації стосовно безпеки кластера Kubernetes шляхом сканування його різними утилітами; 4. Реалізація оркестратора сканерів Kubernetes; 5. Порівняння отриманих результатів виконання сканерів з результатами окремих сканерів.

6. Орієнтовний перелік ілюстративного матеріалу: 45 ілюстрацій.

7. Орієнтований перелік публікацій: Зеленін В. МЕТОДИ ПІДВИЩЕННЯ ЗАХИСТУ В KUBERNETES / VI Міжнародна студентська наукова конференція «Сучасні аспекти та перспективні напрямки розвитку науки»

8. Дата видачі завдання 26.07.2023

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів дипломної роботи	Примітка
1.	Постановка задачі, огляд технічної літератури	26.07.23 – 01.09.23	Виконано
2.	Аналіз існуючих методів підвищення захисту в Kubernetes	01.09.23 – 09.09.23	Виконано
3.	Розробка архітектури запропонованої утиліти для підвищення захисту в Kubernetes	09.09.23 – 19.09.23	Виконано
4.	Розробка утиліти	19.09.23 – 10.12.23	Виконано
5.	Аналіз ефективності і актуальності реалізації	10.12.23 – 11.12.23	Виконано
6.	Оформлення дипломної роботи	11.12.23 – 31.12.23	Виконано

Здобувач вищої освіти _____ Владислав ЗЕЛЕНІН
(підпис) (Власне ім'я, ПРІЗВИЩЕ)

Керівник роботи _____ Андрій РОДІОНОВ
(підпис) (Власне ім'я, ПРІЗВИЩЕ)

РЕФЕРАТ

Робота обсягом 96 сторінок містить 45 ілюстрацій, 2 таблиці, 22 джерела літератури та 2 додатки.

Об’єктом дослідження є підвищення захищенності Kubernetes шляхом комбінації результатів сканування кластеру.

Предметом дослідження є рівень актуальності результатів утиліт для сканування конфігураційних файлів та кластеру Kubernetes на вразливості.

Метою роботи роботи є підвищення захищеності Kubernetes шляхом автоматизації перевірок кластеру.

У цьому дослідженні було проаналізовано існуючі методи підвищення захисту в Kubernetes та доведено практично і теоретично необхідність у впровадженні утиліти-оркестратора сканерів вразливостей з відкритим програмним кодом задля отримання актуальної інформації стосовно захищеності Kubernetes.

Ключові слова: Kubernetes, вразливості Kubernetes, сканер вразливостей Kubernetes.

ABSTRACT

The work, consisting of 96 pages, contains 45 illustrations, 2 tables, 22 literature sources, and 2 appendixes.

The object of the study is to increase the security of Kubernetes by combining the cluster scan results.

The subject of the research is the level of relevance of the results of utilities for scanning configuration files and the Kubernetes cluster for vulnerabilities.

The aim of the work is to improve the security of Kubernetes by automating cluster checks.

This study analyzed existing methods of enhancing protection in Kubernetes and theoretically and practically proved the necessity of implementing an open-source vulnerability scanner orchestrator utility to obtain current information regarding the security of Kubernetes.

Key words: Kubernetes, Kubernetes vulnerabilities, Kubernetes vulnerability scanner.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

K8s – скорочення від “Kubernetes”;

Pod – абстракція в Kubernetes, являє собою групу контейнерів застосунку і ресурси, які спільні для цих контейнерів;

Node – вузол, машини, на яких розгортаються контейнери;

ВМ – віртуальна машина;

ОС – операційна система.

ВСТУП

У сучасному світі, де технології безперервно розвиваються, безпека і інформаційних системах має особливе значення. Kubernetes, як один з провідних платформ для оркестрації контейнеризованими застосунками, стає ключовим елементом у створенні гнучких, великих IT інфраструктур. Однак, з розвитком і популяризацією Kubernetes зростає і кількість потенційних загроз безпеці, що робить дослідження цієї області вкрай актуальним.

Актуальність дослідження: Враховуючи широке використання Kubernetes у різноманітних секторах — від стартапів до великих корпорацій — питання безпеки набуває особливої ваги. Поглиблене вивчення цієї тематики допоможе не лише підвищити загальний рівень безпеки в IT галузі, але й сприятиме розвитку нових підходів та технологій в сфері кібербезпеки.

Метою роботи є підвищення захищеності Kubernetes шляхом автоматизації перевірок кластеру.

Об’єктом дослідження є підвищення захищенності Kubernetes шляхом комбінації результатів сканування кластеру.

Предметом дослідження є рівень актуальності результатів утиліт для сканування конфігураційних файлів та кластеру Kubernetes на вразливості.

Завданням дослідження є вивчення та аналіз безпеки платформи Kubernetes, розбір існуючих вразливостей згідно з OWASP Top 10; Порівняльний аналіз існуючих методів захисту протидії та виявленню вразливостей з використанням утиліт з відкритим вихідним кодом; Теоретичне та практичне доведення обмеженості використання однієї утиліти; Впровадження власного рішення у вигляді оркестратора сканерів вразливостей з відкритим вихідним кодом.

Наукова новизна дослідження: побудована утиліта для оркестрації сканерів вразливостей в Kubernetes.

Апробація результатів роботи: дослідження було представлено у вигляді доповіді на VI Міжнародній студентській науковій конференції «Сучасні аспекти та перспективні напрямки розвитку науки».

Публікації: Робота була опублікована у збірнику матеріалів [22] VI Міжнародної студентської наукової конференції «Сучасні аспекти та перспективні напрямки розвитку науки».

1 ОГЛЯД ПЛАТФОРМИ KUBERNETES

1.1 Загальні відомості про Kubernetes

Kubernetes — це адаптована, розширювана система з відкритим кодом, призначена для керування сервісами та контейнерними робочими навантаженнями, яка значно полегшує декларативне налаштування і автоматизацію. Він підтримує як автоматизовані процеси, так і чітку конфігурацію за підтримки екосистеми, що швидко розширюється та розвивається. Усі сервіси, підтримка та інструменти Kubernetes широкодоступні.

Для детального огляду на Kubernetes треба спочатку повернутись у минуле і зрозуміти чому він зараз настільки корисний.

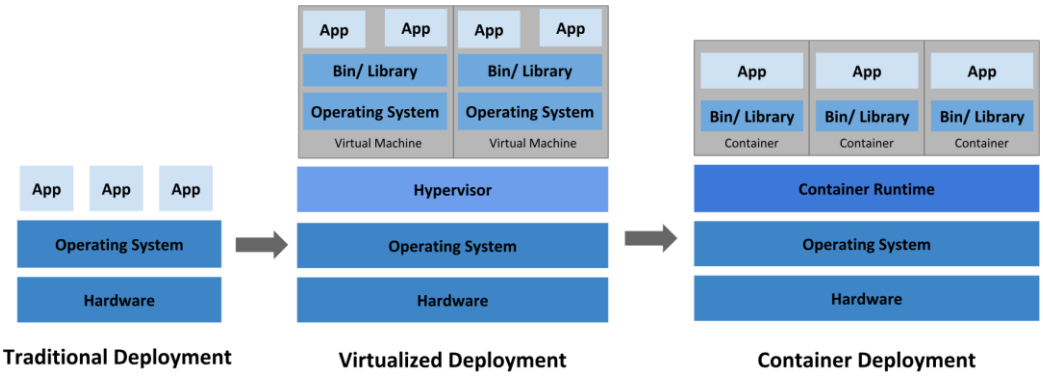


Рисунок 1.1 – Еволюція контейнерів

Традиційне розгортання: раніше компанії розгортали ПЗ на автономних фізичних серверах. Ці сервери не пропонували механізм встановлення обмежень на ресурси для окремих програм, що призводило до потенційних проблем із розподілом ресурсів. Наприклад: на спільному сервері одна програма може витрачати більше ресурсів, що сприяє «відставанню» інших програм. Одним з обхідних шляхів було використання окремого сервера, однак цей підхід був не дуже ефективний, оскільки

часто призводив до недостатнього використання ресурсів і збільшення витрат через необхідність використання кількох серверів.

Ера віртуалізації: як вирішення проблеми традиційного розгортання було запропоновано та введено «віртуалізацію». Це дозволяє кільком віртуальним машинам працювати на одному фізичному ЦП сервера. Віртуалізація дозволяє ПЗ бути ізольованим між різними машинами тим самим забезпечуючи деякий рівень безпеки.

Цей підхід значно підвищу ефективність використання ресурсів фізичного сервера та підвищує можливість масштабування. Програми, які знаходяться у віртуальній машині, можна безперешкодно оновлювати або додавати, що, у свою чергу, знижує витрати на апаратне забезпечення та пропонує інші переваги. Кожна машина працює як цілісна система, що містить усі її компоненти, включно з окремою ОС , на вершині віртуалізованої інфраструктури.

Ера контейнеризації: Контейнери схожі на ВМ, але мають послаблені властивості ізоляції, щоб спільно використовувати ОС між програмами, тому контейнери вважаються «легкими». Це не єдина схожість до ВМ, окрім цього можна зазначити що контейнери також мають свою власну файлову систему, процесний простір, пам’ять та інше. Оскільки вони відокремлені від основної інфраструктури, вони переносяться між хмарами та дистрибутивами ОС.

Тому контейнери є гарним способом «запакувати» і запустити ПЗ. У production середовищі треба керувати запущеними контейнерами і гарантувати відсутність простоїв (downtime). Наприклад: якщо з контейнером щось стається і він виходить з ладу інший контейнер має бути створений негайно. Було б значно легше якби це все виконувалось і контролювалось автоматично, наприклад якоюсь системою.

Саме для цього був створений всім відомий Kubernetes. Платформа надає можливість стійкої роботи розподілених систем. Система піклується про відновлення програми після відмови, її масштабування, надає шаблони розгортання та інше.

Тому Kubernetes можна використовувати для наступні можливості:

- **Service Discovery & Load Balancing:** Kubernetes може відкривати контейнер використовуючи його DNS ім’я або IP адресу. Якщо контейнер піддається великому мережевому трафіку, k8s автоматично балансує і перенаправляє мережевий трафік, забезпечуючи стабільність системи.
- **Store Management:** Kubernetes забезпечує гнучкість автоматичного підключення до будь-якого рішення зберігання даних: будь то локальне сховище, хмарна чи інші.
- **Automated Updates & Rollbacks:** За допомогою k8s можна вказати бажаний стан для своїх контейнерів. Потім автоматично налаштовується фактичний стан на запланований у керованому темпі. Наприклад, він може автономно створювати контейнери, відкидати старі та передавати їхні ресурси до нових контейнерів.
- **Optimized Resource Utilization:** Ви надаєте Kubernetes кластер вузлів для виконання контейнерних завдань. Вказуючи вимоги до ЦП і пам’яті для кожного контейнера, Kubernetes ефективно розміщує контейнери на вузлах, щоб оптимізувати використання ресурсів.
- **Self-healing:** Kubernetes автоматично перезавантажує контейнер, який вийшов з ладу, видаляє контейнери які не відповідають на зазначені health-checks.
- **Secret & Configuration Handling:** k8s пропонує захищене середовище, в якому можна зберігати конфіденційну інформацію. Цю інформацію можна завантажувати або оновлювати без створювання нових образів контейнеру.

- **Easy Scaling:** можна збільшувати або зменшувати потужність програми за допомогою простої команди, через інтерфейс користувача або автоматично залежно від споживання ЦП.[1]

1.1.1 Компоненти Kubernetes

Після визначення для чого саме потрібен Kubernetes можна перейти до більш детального опису як саме він працює і з яких компонентів складається.

- Коли розгортається Kubernetes, на виході отримуєте кластер;
- Кластер складається з набору робочих машин, які називаються нодами, які запускають контейнеризовані застосунки. Кожен кластер повинен мати щонайменше одну робочу ноду;
- Робочі ноди розміщують поди, які є компонентами робочого навантаження застосунку;
- Площина керування (control plane) управляє робочими нодами і подами в кластері. В production середовищі площина керування зазвичай працює на декількох комп'ютерах, а сам кластер на кількох нодах, тим самим забезпечуючи відмовостійкість і високу доступність.

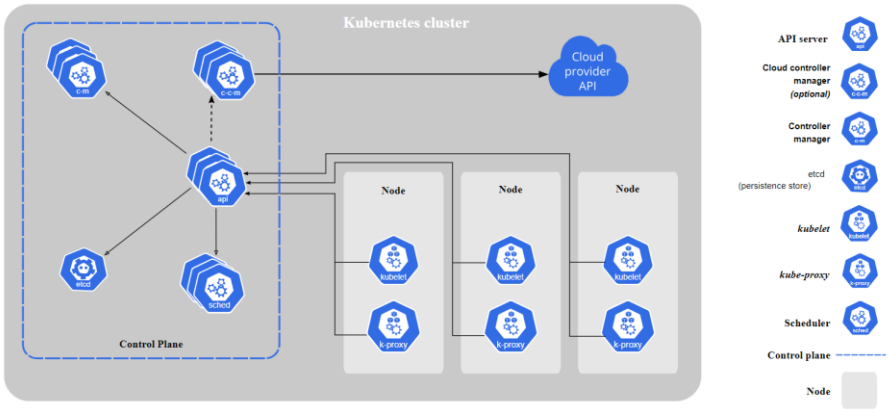


Рисунок 1.2 – Компоненти Kubernetes кластеру

Компоненти площини керування відповідають за прийняття глобальних рішень щодо кластера, ось наприклад планування, а також за виявлення подій кластера та реагування на них.

Ці компоненти можна запускати на будь-якій машині в кластері, однак для спрощення, сценарії налаштування зазвичай запускають усі компоненти рівня керування на одній машині.

Перейдемо саме до переліку вищезгаданих компонентів:

kube-apiserver – являє собою компонент площина керування, який відкриває Kubernetes API. Реалізація цього програмного інтерфейсу додатку є вищезгаданий апі-сервер. Він призначений для горизонтального розширення, мається на увазі, що він масштабується шляхом розгортання більшої кількості екземплярів ПЗ.

etcd – консистентний і високо-доступний key-value сховище, яке використовується Kubernetes для зберігання даних кластеру.

kube-scheduler - компонент керуючої площини, який стежить за свіж створеними контейнерами без призначеного вузла та обирає для них місце запуску. При прийнятті рішень у плануванні враховуються такі фактори: окремі та загальні потреби у ресурсах, обмеження з точки зору апаратури, програмного забезпечення чи політики, критерії спорідненості та анти спорідненості, локалізація даних, проблеми між робочими завданнями та кінцеві терміни.

kube-controller-manager - компонент площини керування, який запускає процеси контролера. Хоча кожен контролер на логічному рівні є відокремленим процесом, для спрощення всі вони зібрані в один бінарний файл і працюють у межах одного процесу.

Окрім компонентів площини керування можна виділити ще компоненти нод. Вони запущені на кожній ноді і надають Kubernetes середовище виконання.

kubelet - на кожному вузлі кластера діє агент, що забезпечує функціонування контейнерів у рамках поду. Kubelet оперує з заданим набором специфікацій Pod (PodSpec), які отримує за допомогою різноманітних інструментів, і відповідає за те, щоб контейнери, описані в цих специфікаціях, були активні та працездатні. Контейнери, які були створені поза Kubernetes, не підпадають під управління Kubelet.

kube-proxy - це мережевий проксі, що функціонує на кожному вузлі вашого кластера і втілює деякі аспекти концепції служб Kubernetes.[2]

1.1.2 Архітектура кластеру

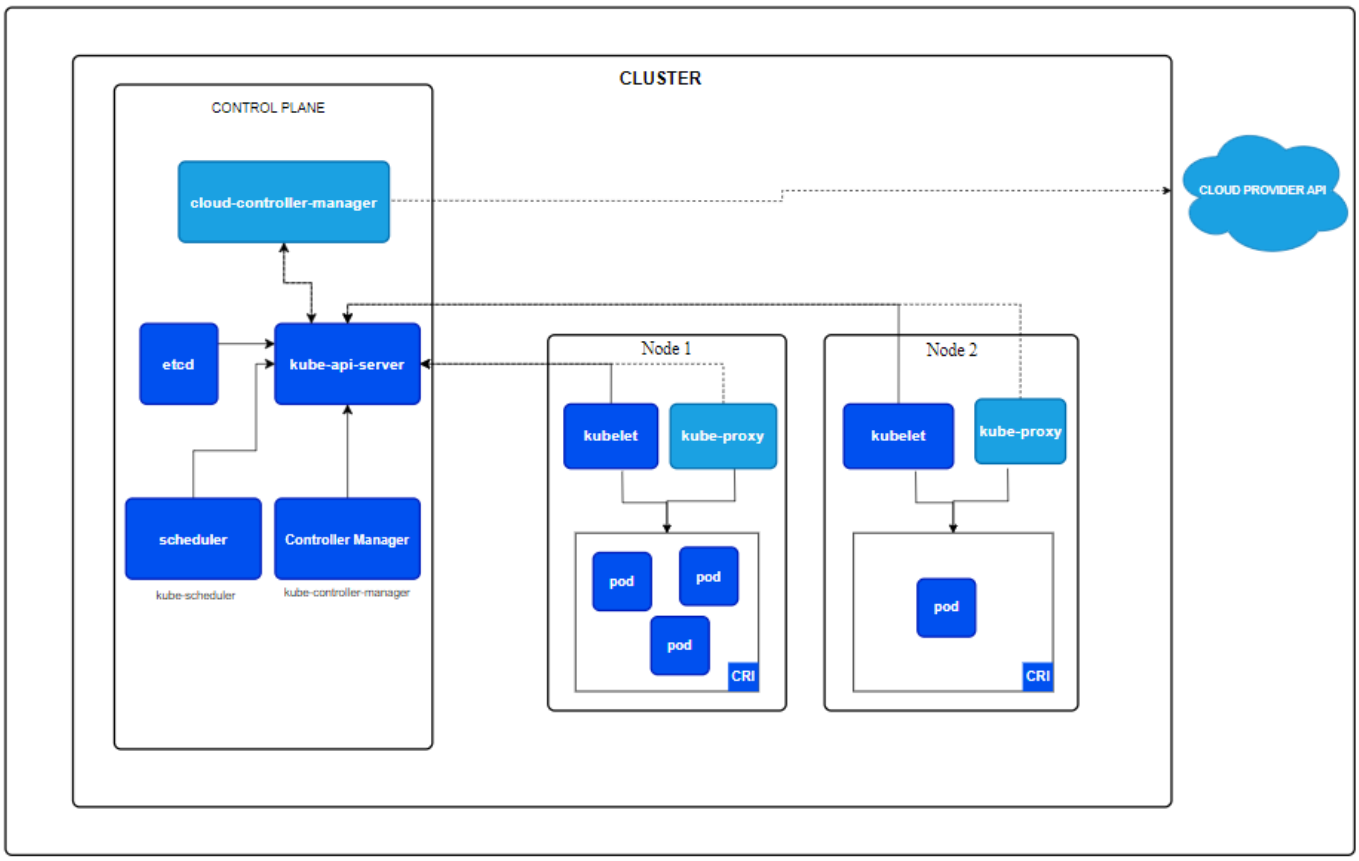


Рисунок 1.3 – Архітектура Kubernetes кластеру

Kubernetes виконує робочі завдання, розташовуючи контейнери в подах на нодах. Нода може представляти собою віртуальний або реальний комп'ютер, залежно від конфігурації кластера. Площина керування надає управління кожним вузлом, на якому запущені служби, необхідні для роботи подів. Зазвичай кластер має декілька нод. Проте в навчальних цілях або в умовах обмежених ресурсів може бути всього одна. Серед компонентів, розташованих на ноді, є kubelet, середовище виконання контейнера та kube-проксі.

Існує два ключові методи додавання нод (вузлів) до сервера API:

- Kubelet на вузлі автоматично реєструє себе на площині керування;
- Ви (або інша особа) самостійно додаєте об'єкт Node.

Після того, як об'єкт Node було створено вручну чи kubelet автоматично зареєстрував себе на вузлі, управляюча площина виконує перевірку коректності нового об'єкта Node. Наприклад, при спробі додати вузол на основі даного маніфесту JSON:

```
{
  "kind": "Node",
  "apiVersion": "v1.1",
  "metadata": {
    "name": "10.240.79.158",
    "labels": {
      "name": "some-k8s-node"
    },
    "annotations": {
      "description": "This is test Kubernetes node."
    }
  }
}
```

Kubernetes формує внутрішнє представлення вузла у вигляді об'єкта Node. Kubernetes переконується, що kubelet зареєстрований на сервері API, який відповідає імені в полі metadata.name об'єкта вузла. Якщо вузол відпрацьовує коректно (тобто всі потрібні служби активні), він може запускати Pod. В протилежному випадку, такий вузол відсіюється від участі в діях кластера до моменту його нормальної роботи.[3]

Kubernetes використовує модель API "hub-and-spoke". Усі запити до API від вузлів надходять до сервера. Жоден з інших компонентів управління не призначений для віддаленого доступу. Сервер API слухає віддалені запити на захищеному порту HTTPS із активованою автентифікацією клієнта. Рекомендується активувати авторизацію, особливо якщо дозволені анонімні запити або маркери доступу.

Вузлам слід мати кореневий сертифікат, доступний для всього кластера, щоб безпечно підключатися до сервера API із вірними клієнтськими даними. Варто забезпечити, щоб даними, які подає kubelet, були клієнтські сертифікати. Варто розглянути використання TLS для kubelet для автоматичного отримання клієнтських сертифікатів.

Модулі, що хочуть з'єднатися із сервером API, можуть це робити безпечно, використовуючи облікові дані служби. Таким чином, Kubernetes автоматично

вставляє кореневий сертифікат та відповідний маркер. Служба kubernetes (у просторі імен за замовчуванням) має віртуальну IP-адресу, що перенаправляє на кінцевий пункт HTTPS на сервері API.

Елементи площини керування також обмінюються даними з сервером API через цей порт. В результаті, стандартний спосіб з'єднання між вузлами та контейнерами до площини керування є безпечним та можуть функціонувати через потенційно ненадійні або публічні мережі.

Існують два ключових канали зв'язку від управлінської площини (сервер API) до вузлів. Перший канал — між сервером API та процесом kubelet, який діє на кожному вузлі в кластері. Другий канал — між сервером API та будь-яким вузлом, контейнером або службою через API-проксі. З'єднання від сервера API до kubelet використовуються для таких дій:

- Завантаження логів контейнерів;
- З'єднання (зазвичай через kubectl) до активних контейнерів;
- Використання можливості перенаправлення портів kubelet.

Ці з'єднання здійснюються через HTTPS-кінцеву точку kubelet. Зазвичай сервер API не виконує перевірку сертифіката kubelet, що робить з'єднання вразливим до атак «людина посередині» і небезпечним у відкритих або ненадійних мережах.

Для захисту цього з'єднання можна використовувати опцію --kubelet-certificate-authority, надаючи серверу API кореневий набір сертифікатів для перевірки сертифіката kubelet.

Якщо безпосереднє з'єднання є непридатним, рекомендується використовувати SSH-тунелювання між сервером API та kubelet для уникнення експозиції в небезпечних мережах.

Нарешті, для підвищення безпеки Kubelet API варто активувати процедури автентифікації та авторизації Kubelet.

Kubernetes використовує SSH-тунелі для захисту комунікацій між площиною керування та вузлами. У такому налаштуванні сервер API створює SSH-тунель до кожного вузла в кластері (з'єднуючись з SSH-сервером на порті 22) і направляє весь комунікаційний трафік, що націлено на kubelet, вузол, контейнер або службу, через цей тунель. Такий підхід забезпечує ізоляцію трафіку в мережі, де розташовані вузли.

1.1.3 Контейнери

Кожний контейнер, який ви розгортаєте, працює консистентно, а завдяки вбудованим залежностям можна бути впевненими у його однаковій роботі незалежно від місця запуску.

Контейнери ізолюють додатки від основної системи хоста, що спрощує їх розгортання у різноманітних хмарних сервісах або операційних системах. На кожному вузлі в кластері Kubernetes запущені контейнери, що належать цьому вузлу. Всі контейнери в модулі розташовані поруч і призначені для спільного виконання на тому самому вузлі.

Образ контейнера є готовим до роботи набором, який включає все необхідне для роботи програми: вихідний код, необхідне виконавче середовище, потрібні програмні та системні бібліотеки, а також базові параметри налаштувань. Контейнери повинні бути стабільними і незмінними. Ви не маєте модифікувати код вже запущеного контейнера. Якщо потрібно внести зміни в програму у контейнері, рекомендовано створювати новий образ із цими змінами та перезапускати контейнер з оновленого образу.

Ключовим компонентом, який надає Kubernetes можливість ефективно керувати контейнерами, є компонент, що відповідає за життєвий цикл та виконання контейнерів у системі. Kubernetes працює із різними реалізаціями виконавчого середовища контейнерів, такими як containerd, CRI-O та інші, що відповідають інтерфейсу Kubernetes CRI. Зазвичай, вам стає достатньо дозволити Kubernetes автоматично вибрати виконавче середовище контейнера для Pod'ів. Проте, якщо ви

бажаєте використовувати декілька середовищ виконання у своєму кластері, використовуйте RuntimeClass для вказівки конкретного середовища для конкретного Pod. За допомогою RuntimeClass можна також запускати модулі з однаковим середовищем виконання, але з різними налаштуваннями.[4]

1.1.4 Сервіси, балансування навантаження та мережа

В кластері Kubernetes кожен Pod має свою індивідуальну IP-адресу. Це дозволяє спрощувати взаємодію між модулями без необхідності ручного налаштування портів чи їх відображення. Така модель прирівнює Pods до віртуальних або фізичних машин з точки зору роботи з портами, назвами, виявленням служб та іншими параметрами.

Kubernetes встановлює кілька ключових вимог для мережевих реалізацій. Модулі повинні мати змогу спілкуватися між собою без використання NAT. Також системні процеси на вузлі мають змогу спілкуватися з модулями на тому ж вузлі. Деякі платформи дозволяють модулям працювати без NAT у головній мережі.

Дана модель є простою та відповідає концепції Kubernetes про легкий перехід від віртуальних машин до контейнерів. Адресаційний простір в Kubernetes призначений для Pod'ів, де контейнери в межах одного Pod спільно використовують мережеві ресурси.

Як контейнери взаємодіють в мережі залежить від використовуваного середовища виконання. Є можливість перенаправляти порти на вузол, які потім ведуть до вашого Pod, але це досить специфічна дія.

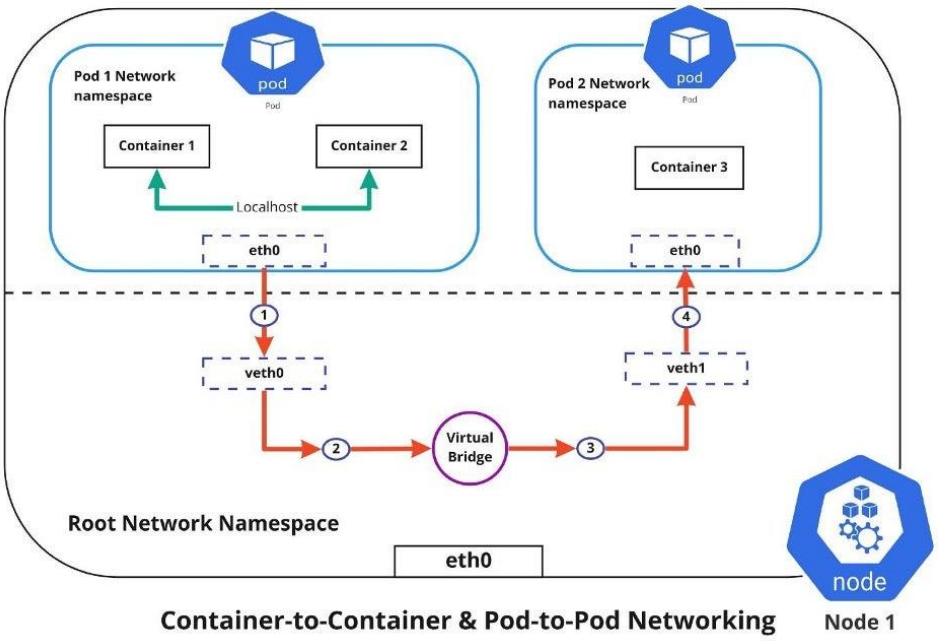


Рисунок 1.4 – Container to container & Pod-to-Pod Networking [5]

Мережева архітектура Kubernetes вирішує кілька ключових завдань:

- Забезпечення комунікації між контейнерами в межах одного Pod.
- Забезпечення комунікації між різними модулями в мережі кластера.
- Використання службового API для доступу до програм у Pods ззовні кластера.
- Використання Ingress для публікації HTTP-додатків та веб-ресурсів.
- Можливість розкриття служб лише для внутрішнього використання в кластері.

1.2 Безпека в Kubernetes

Впровадження захисту в середовищі Kubernetes є завданням, яке вимагає глибокого розуміння системи. Kubernetes представляє собою комплексну систему, що

об'єднує численні компоненти, і неможливо забезпечити її повний захист за допомогою одного захисного модуля чи інструмента безпеки.

Ефективний захист Kubernetes полягає в комплексному підході до ризиків безпеки, що можуть зачепити різноманітні компоненти і сервіси всередині кластера. До прикладу, необхідно мати знання щодо захисту вузлів, мережевих ресурсів, модулів, даних тощо.

Додатково, адміністратори Kubernetes мають бути обізнані з вбудованими інструментами безпеки, які пропонуються Kubernetes, а також з потребою інтеграції сторонніх інструментів безпеки для забезпечення всебічного захисту. Хоча Kubernetes не розглядається як спеціалізована платформа безпеки, він надає інструменти такого типу, як рольовий контроль доступу (RBAC).

Можливо, найпростіший спосіб підійти до безпеки Kubernetes — це подумати про типи ризиків, які впливають на кожну частину стеку Kubernetes, а потім визначити інструменти та ресурси, доступні для їх захисту.

Kubernetes має різні варіації конфігурацій та масштабів. Деякі з них є безпечнішими в порівнянні з іншими. В залежності від того, яку структурну концепцію Kubernetes ви обираєте під час створення системи, це може впливати на її безпеку. Наприклад, система, яка використовує кілька кластерів, може мати певні переваги в безпеці порівняно з системою на одному кластері. Однак використання кількох кластерів може також додати складнощі, що стає мінусом з огляду на безпеку. Додавання сторонніх інструментів, таких як системи моніторингу чи сервісні мережі, також може мати свої особливості для безпеки в архітектурі Kubernetes.

Більшість команд, плануючи впровадження Kubernetes, задають собі питання: користуватися керованою послугою Kubernetes, наприклад Amazon AKS чи Azure Kubernetes Service, або самостійно розгортати та адмініструвати Kubernetes на власній інфраструктурі.

Використання керованого сервісу Kubernetes зазвичай є простішим у налаштуванні та обслуговуванні, оскільки провайдер виконує деякі аспекти налаштування та обслуговування для вас. Проте обсяг функцій, які надаються різними керованими сервісами, може відрізнятися.

За умовами керованого Kubernetes:

1. Інфраструктура оптимізована для безпеки на професійному рівні.
2. Постачальник може надати передналаштовані інструменти, які зазвичай відповідають найкращим стандартам безпеки.

Однак у керованої системи є обмеження у контролі та конфіденційності, через що користувачам може бути важко досягти повного рівня безпеки. Наприклад, ви не зможете ізольвувати свої кластери від Інтернету. І хоча деякі керовані платформи сумісні з власною інфраструктурою, більшість використовують інфраструктуру у публічній хмарі, яка може бути піддана ризикам. Отже, для початківців, що не впевнені у принципах безпеки Kubernetes, керований сервіс може бути кращим вибором. Проте для тих, хто шукає додаткові можливості безпеки, наприклад, ізоляція від Інтернету, краще обрати самостійне розгортання.

Під час планування впровадження Kubernetes ви повинні вирішити, чи створювати один кластер чи декілька.

Здебільшого команди раніше користувалися одним кластером. Проте згідно з CNCF[1], багатокластерні налаштування набувають популярності, зокрема через можливість відокремлення робочих навантажень. Такий підхід знижує ймовірність перенесення проблем безпеки з одного навантаження на інше. Більше інформації про безпеку кластерів Kubernetes можна знайти в додаткових матеріалах.

[1] Cloud Native Computing Foundation

Однак, збільшення кількості кластерів також призводить до зростання складності. Це стосується кількості журналів, політик RBAC та мережевих налаштувань.

Якщо у вас вже є потужні інструменти автоматизації для адміністрування кластерів, ця складність може бути менш проблематичною, і вам може підійти багатокластерний підхід. Проте, якщо ви відчуваєте труднощі з таким налаштуванням, можливо, варто обрати однокластерний варіант для простоти моніторингу та управління.

Основним інструментом захисту API-сервера в Kubernetes є платформа керування доступом на основі ролей Kubernetes, або RBAC. За допомогою політик RBAC ви можете визначати користувачів і облікові записи служб, а потім призначати дозволи, які дозволяють їм виконувати певні дії за допомогою Kubernetes API.

Для додаткового захисту Kubernetes API і, відповідно, усього кластера Kubernetes, можна використовувати контролери доступу. В основі своїй, контролер доступу є кодом, який обробляє запити до API Kubernetes після проходження процедур автентифікації та авторизації, але перед фінальною обробкою цього запиту.

За допомогою контролерів доступу ви можете встановлювати обмеження на типи ресурсів, які може запитувати об'єкт від API, або блокувати виконання окремих команд, незалежно від дозволів, наданих через політики RBAC чи контексти безпеки [6].

1.2.1 Kubernetes Network Security

Безпека мережі в Kubernetes має велике значення, оскільки вона сприяє захисту конфіденційних даних та ключових застосунків, які працюють у кластері Kubernetes. Забезпечення безпечного мережевого спілкування допомагає запобігти несанкціонованому доступу, порушенню даних, втручанню та іншим шкідливим діям, які можуть загрожувати кластеру та його навантаженням.

Додатково, безпека мережі в Kubernetes сприяє забезпеченню конфіденційності, цілісності та доступності застосунків та даних у кластері, відповідаючи регулятивним вимогам та галузевим стандартам.

Більше того, впровадження заходів щодо безпеки мережі в кластері Kubernetes може сприяти підвищенню загального рівня безпеки, дозволяючи організаціям з впевненістю розгортати та запускати ключові застосунки в середовищах.

Існує багато способів покращити захист мережі, ось основні з них:

Network Security Policies - це правила, які визначають дозволене спілкування між подами у кластері. Вони сприяють контролю доступу до мережі та запобігають недозволеному доступу до конфіденційних даних. Існує два способи реалізації політик мережевої безпеки в кластері Kubernetes:

- **API політик мережі Kubernetes**: Kubernetes пропонує вбудовані API мережевої політики, які дозволяють організаціям визначати та застосовувати політики мережевої безпеки. Ці API дозволяють організаціям вказувати, які поди можуть спілкуватися між собою, на основі різних критеріїв, таких як простір імен, мітка та IP-адреса;
- **Постачальники політики мережевої безпеки сторонніх виробників**: Існують також постачальники політики мережевої безпеки сторонніх виробників, такі як Calico та Cilium, які пропонують додаткові функції безпеки та інтеграцію з Kubernetes. Ці постачальники дозволяють організаціям впроваджувати політики мережевої безпеки, контролювати мережевий трафік та відстежувати активність у мережі.

CNI Plugins - Плагіни інтерфейсу мережі контейнерів (CNI) відповідають за забезпечення мережевого з'єднання контейнерів у кластері. Отже, обираючи плагін, необхідно звертати особливу увагу на його можливості забезпечення безпеки. Ось деякі ключові характеристики безпеки, які можуть надавати плагіни CNI:

- **Розділення мережі**: Вони можуть створювати підмережі в межах основної мережі, щоб зменшити ризики та обмежити можливість поширення вірусів;
- **Захист даних**: Ці плагіни можуть використовувати методи шифрування для захисту конфіденційної інформації, що передається мережею;
- **Контроль за мережею**: Деякі з них мають інструменти для відстеження мережевого трафіку, що допомагає виявляти аномалії та потенційні загрози.

Service Mesh - мережеві решітки використовують допоміжний проксі для кожного екземпляра служби, який виступає як посередник між службою та мережею. Такий підхід надає кілька переваг з точки зору безпеки мережі:

- **Аутентифікація**: Мережеві решітки можуть забезпечувати взаємну аутентифікацію TLS (безпека транспортного рівня) між екземплярами служб, гарантуючи, що тільки довірені служби можуть спілкуватися між собою;
- **Авторизація**: Мережеві решітки можуть застосовувати детальні політики контролю доступу між екземплярами служб, дозволяючи адміністраторам вказувати, які служби можуть спілкуватися між собою та який вид спілкування дозволений;
- **Керування трафіком**: Мережеві решітки можуть направляти трафік між екземплярами служб на основі політик та правил, дозволяючи адміністраторам керувати потоком трафіку між службами та застосовувати політики безпеки мережі.

RBAC - це функція безпеки в Kubernetes, яка дозволяє адміністраторам регулювати доступ до ресурсів у кластері згідно з ролями. Завдяки цій функції, адміністратори мають можливість регулювати, хто і до яких ресурсів у кластері може отримати доступ, базуючись на певних ролях. Використовуючи RBAC, можна

встановити докладні правила для різних користувачів та частин системи, забезпечуючи тим самим, що лише ті, хто має на це право, зможуть виконувати певні дії.

Розглянемо конкретний приклад: за допомогою RBAC можна дозволити доступ до важливих системних ресурсів, таких як сервер API Kubernetes, тільки визначеній групі користувачів. Такий підхід допомагає уникнути несанкціонованих втручань, витоків даних та інших потенційних проблем. Також, завдяки автоматизації в RBAC, адміністрування прав доступу стає простішим і дотримання стандартів безпеки забезпечується ефективніше.

Enforce Zero Trust - Безпека на принципі "zero trust" полягає в тому, що жоден мережевий трафік не вважається надійним за замовчуванням. Кожен запит має бути перевірений та аутентифікований перед тим, як отримати доступ до мережі. Для посилення безпеки в мережі Kubernetes можна застосувати такі заходи:

- **Розподілення на сегменти**: Розділіть мережу на більш безпечні сегменти і контролюйте доступ до кожного з них;
- **Мінімізація видимості в мережі**: Обмежте доступ до сервера API Kubernetes і бази даних etcd лише для перевірених IP-адрес і мереж. Також використовуйте розділення мережі для ізоляції основних компонентів від робочих вузлів та користувацьких завдань;
- **Перевірка та підписування образів**: Переконайтеся, що всі образи, які використовуються в кластері, перевірені на наявність вразливостей і містять підпис. Це допоможе уникнути використання шкідливих образів;
- **Застосування двофакторної аутентифікації**: Вимагайте двофакторної аутентифікації при доступі до сервера API Kubernetes і бази даних etcd, щоб зменшити ризик незаконного доступу. [7]

1.2.2 Kubernetes Container Security

Безпека контейнерів має на меті захищати контейнеризовані системи, які відрізняються своєю складністю в порівнянні з традиційними сценаріями. Коли ми розглядаємо робочі середовища, вони працюють з безліччю контейнерів. Тому спеціалісти у галузі безпеки та адміністратори мають більше викликів при роботі з контейнерами.

Щоб контейнери були безпечними, потрібно створювати та підтримувати засоби контролю. Коли безпека вбудовується прямо в процес створення продукту, це гарантує, що від початкового стадію до завершення весь цикл продукту захищений.

Контейнери надають ряд переваг, але і приносять виклики з безпеки. Одним з основних викликів є збільшення можливостей для атак через багатість контейнерів, створених на базі різних образів, що мають свої ризики.

Крім того, усі контейнери спільно використовують одну архітектуру ядра. Тому захист лише основної системи недостатній. Необхідно також враховувати налаштування безпеки контейнерів та забезпечувати відокремлення між ними.

Ще одним аспектом є те, що в контейнеризованих середовищах може бути важко стежити за всім, що відбувається. Традиційні засоби відстеження можуть не впоратися з розпізнаванням дій контейнерів. Тому важливо забезпечити якнайкращий контроль над ситуацією, щоб вчасно реагувати на загрози.

Container Security Best Practices:

1. Securing Images;
2. Securing Registries;
3. Securing Deployment
4. Securing Container Runtime
5. Securing Kubernetes
6. Using Short-Live containers
7. Monitoring container activity

Часті помилки безпеки контейнерів, яким слід уникнути. Ось кілька поширених проколів у безпеці контейнерів, яких слід уникати:

Знехтування основними принципами безпеки – Незважаючи на новітній характер контейнерів, деякі базові норми безпеки залишаються актуальними. Важливо забезпечити своєчасне оновлення всіх систем, включаючи операційні системи та середовища запуску контейнерів.

Відсутність налаштувань інструментів та середовищ – Платформи оркестрації контейнерів мають свої специфічні можливості з безпеки. Щоб захистити систему, потрібно коректно налаштовувати ці інструменти. Не слід покладатися на стандартні налаштування безпеки. Наприклад, контейнерам слід надавати лише ті права, які їм необхідні для роботи.

Відсутність відстеження та тестування – Запускаючи контейнери, можна втратити контроль над станом додатків та середовищ. Щоб уникнути ризиків, необхідно налаштувати системи відстеження, реєстрації та тестування, особливо для розподілених систем.

Недостатній захист усіх етапів CI/CD – Слід пам'ятати про безпеку на всіх етапах розробки. Це можна досягти, реалізовуючи підхід "shift left", коли безпека враховується з самого початку розробки. Забезпечте єдність інструментів та політик на всіх етапах. [8]

1.2.3 Kubernetes Troubleshooting

Відлагодження проблем у кластері Kubernetes має три ключові елементи: розуміння причини проблеми, її управління та ліквідація, а також запобігання повторенню проблеми.

Розуміння - у контексті Kubernetes важливо зрозуміти, що саме призвело до проблеми. Для цього зазвичай потрібно:

- Перевірити останні зміни в задіяних кластерах, вузлах або подах, щоб зрозуміти джерело збою;

- Вивчити конфігурації YAML, репозиторії Github, а також протоколи віртуальних машин, де запущені проблемні компоненти;
- Оцінити події в Kubernetes і такі показники, як дискове навантаження, використання пам'яті;
- Порівняти аналогічні компоненти на наявність схожої поведінки і залежності між ними.

Управління - у архітектурі мікросервісів кожний компонент зазвичай розробляється окремою командою. Тому для швидкого вирішення проблем потрібна взаємодія між командами. Є три підходи до вирішення проблем:

- Тимчасові рішення на основі досвіду команди;
- Ручні інструкції - документовані кроки для вирішення типових проблем;
- Автоматизовані інструкції - автоматичні процеси для вирішення проблем.

Запобігання – це включає:

- Розробку правил і процедур після кожного інциденту;
- Визначення способів автоматизації реакції на проблеми;
- Визначення способів швидкого виявлення та реакції на проблеми;
- Забезпечення ефективного спілкування команд при вирішенні проблем.

Kubernetes є складною системою, а вирішення проблем, які виникають у ньому, вимагає глибоких знань. Навіть у невеликому локальному кластері Kubernetes може бути важко виявити і вирішити проблеми. Це пов'язано з тим, що збій може бути пов'язаний з окремим контейнером, декількома подами, контролером або ж компонентом управління.

У великому середовищі ці труднощі лише посилюються через велику кількість компонентів і низьку видимість. Командам потрібно використовувати різні

інструменти для збору інформації для вирішення проблем і, можливо, додаткові інструменти для їх діагностики та усунення.

Що ще ускладнює ситуацію, так це те, що Kubernetes часто використовується для розробки мікросервісних застосунків, де кожен мікросервіс розробляється окремою командою. У деяких випадках команди DevOps та розробники працюють над одним і тим же кластером Kubernetes, що призводить до неясності в розподілі обов'язків. Для короткого висновку – вирішення проблем в Kubernetes може бути складним і ресурсоємким процесом, якщо команди не співпрацюють належним чином та не використовують необхідні інструменти.

1.3 Модель порушника безпеки Kubernetes

Модель порушника безпеки являє собою процес оцінки, ідентифікації та впорядкування потенційних загроз для системи. Основна мета – це зрозуміти потенційних агресорів, їх ціль, інтереси, можливості та методи атаки, щоб можна було відповідно спроектувати заходи безпеки.

Модель порушника включає в себе наступні етапи:

- Визначення того, що саме намагаємось захистити;
- Оцінка та ідентифікації можливих загроз;
- Визначення слабких місць;
- Розробка методів зменшення ризиків.

Моделювання загроз допомагає командам ІБ фокусуватись на найважливіших ризиках та визначати, які заходи безпеки будуть найефиктивнішими в конкретному контексті.

Якщо ви маєте намір використовувати Kubernetes у production середовищі, однією з основних рекомендацій з точки зору безпеки є розробка моделі потенційних загроз.

Наразі існують проекти, які взялися за моделювання загроз для кластера Kubernetes.

Першим з них є NCC – у листопаді 2017 року опублікували свій підхід до моделювання загроз Kubernetes. Там вони виділили три основні групи загроз для кластера Kubernetes.

Зовнішні зловмисники:

- Порушники: люди, які не мають прямого доступу до кластеру, окрім можливості отримати доступ до запущених на ньому програм або портів керування через мережу.
- Засоби протидії: переконатися, що всі служби керування не піддаються впливу ненадійних мереж без засобів керування автентифікацією.

Шкідливі контейнери:

- Порушники: зловмисник має доступ до одного контейнера, ймовірно, через уразливість програми, і хоче розширити свій доступ, щоб захопити весь кластер.
- Засоби протидії: всі порти які відповідають за керування, видимі в мережі кластера, потребують автентифікації. Використовувати мережеві політики, щоб обмежити доступ між просторами імен і модулями. Переконатися, що облікові записи служби або не змонтовано в контейнерах, або мають обмежені права.

Зкомпрометовані користувачі:

- Порушники: зловмисник має дійсні облікові дані для виконання команд Kubernetes API, а також доступ до мережі.

- Засоби протидії: переконатися, що політики RBAC діють для всіх користувачів, надаючи наймеш привілейований доступ до ресурсів кластеру. Правила безпеки контейнерів мають діяти для всіх користувачів, щоб обмежити створення привілейованих контейнерів.

Наступник проектом є CNCF - у січні 2020 року CNCF опублікувала документацію та результати детального аналізу потенційних загроз для типового кластера Kubernetes. Метою цього дослідження було надати комплексний огляд можливих загроз та способів їх нейтралізації, які можна було б використати для виявлення типових шляхів атак на платформу та способів, якими зловмисник міг би використовувати слабкі місця у налаштуваннях Kubernetes для досягнення своїх цілей.

Конкретно, кожний елемент архітектури Kubernetes було досліджено з використанням методології STRIDE для виявлення можливих проблем безпеки на границях довіри в межах платформи. [9]

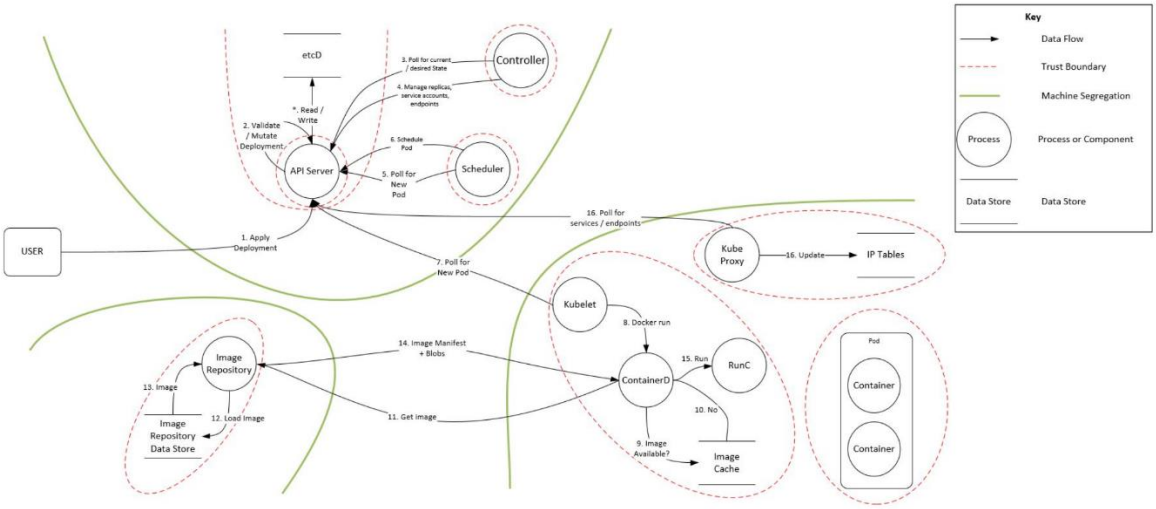


Рисунок 1.5 - Kubernetes Trust Boundaries

Основні вектори атак:

Таблиця 1.1 – Основні вектори атак

Вектор атаки	**Опис**

Кінець таблиці 1.1

Service Token	Зазвичай Service Token автоматично підключається до кожного модуля. У випадку компроментації контейнера, атакуючому може стати доступний інструмент для використання цих даних. Важливими заходами безпеки є строгі налаштування RBAC та відключення автоматичного підключення сервісних токенів.
Compromised Container	Головний фокус в кластері має бути на контейнери, оскільки скомпроментований контейнер забезпечує віддалену точку виконання команд для зловмисника.
Network Endpoints	Кожна кінцева точка Kubernetes має бути захищена від внутрішніх зловмисників. Також варто зазначити, якщо зловмиснику вдається скомпроментувати контейнер, він отримає доступ до кінцевих точок, якщо це дозволяє мережева політика подів.
RBAC Issues	Багато векторів атак залежать від неправильної конфігурації політик RBAC. Засоби пом’якшення мають покладатися на автоматизовані інструменти для перевірки таких політик.

З отриманої інформації можна скласти власну модель порушника ІБ.

Тип порушника: визначена на основі дослідження дозволів, які особи мають до даних та/або елементів ІТ-системи, а також здатності зловмисників взаємодіяти з частинами ІТ-системи з урахуванням її структурних та функціональних особливостей. Зловмисники можуть мати дозволений прямий або віртуальний доступ до частин ІТ-системи та/або зберіганої інформації або бути позбавленими такої можливості.

Види порушників: розвідувальні служби інших країн, радикальні групи, кримінальні елементи, зовнішні зловмисники, конкуренти, розробники, виробники та доставщики програмного та апаратного обладнання, спеціалісти, які підтримують ІТ-системи або інфраструктуру, користувачі системи та інші.

Потенціал порушників: особи з початковим (низьким) рівнем здатності, особи з помірним рівнем здатності та ті, хто має високий рівень здатності.

Мета порушників: отримання доступу до інформації без дозволу, виведення з ладу системи, розповсюдження конфіденційних даних та ін.

Можливі способи реалізації загроз безпеки інформації:

- Несанкціоноване втручання на апаратному рівні через вбудовані програми в апаратних елементах;
- Несанкціонована взаємодія на рівні програмного забезпечення, такими як бази даних, веб-браузери, веб-програми та інші загальні або спеціальні програми.

На основі аналізованих даних можна визначити базову модель зловмисника ІБ, ранжуючи загрози на шкалі від 1 до 4, де 1 символізує мінімальну загрозу, а 4 - ризик втрати інформації та здійснення нищівної атаки. Деталі моделі зловмисника ІБ представлені в таблиці.

Таблиця 1.2 – Модель порушника ІБ

Позначення	Категорія	Рівень загрози
Внутрішні		
В1	Сервісний штат	1
В2	Технічні співробітники	2
В3	Керівництво	3
В4	Служби безпеки	4
Зовнішні		
З1	Звичайні користувачі	1

Кінець таблиці 1.2

З2	Конкуренти	2
З3	Зловмисники (хакери)	3
З4	Служби безпеки іноземних держав	4

Висновки до розділу 1

В даному розділі було розглянуто загальні відомості про Kubernetes з детальним оглядом на безпекову частину даної платформи. Було зазначено вразливі місця, на які варто звернути увагу, якщо обираєте вищезгадану платформу оркестрації контейнерів. Було побудовано модель порушника ІБ.

2 АНАЛІЗ ЗАХИЩЕННОСТІ ПЛАТФОРМИ KUBERNETES

Керуючи великими об'ємами даних і додатків, Kubernetes використовує концепцію "кластеризації", де декілька серверів об'єднуються для спільної роботи, забезпечуючи високу доступність та надійність ресурсів. Це, в свою чергу, створює кілька точок входу і витоку даних, які можуть бути потенційними місцями атак для зловмисників. Тому існує ряд викликів у забезпеченні безпеки Kubernetes, що включає в себе забезпечення мережевої безпеки, управління доступом, оновлення без втрати даних та моніторинг з метою виявлення аномалій.

Більше того, з огляду на відкритий код Kubernetes, зловмисники постійно в пошуках нових вразливостей. Це ставить під загрозу не тільки індивідуальні компоненти системи, але й весь бізнес, який базується на цій платформі. Втрати від неавторизованого доступу або витоку даних можуть бути катастрофічними, особливо для великих організацій і корпорацій, які залежать від надійності та безпеки своїх IT-систем.

Тому поглиблений аналіз потенційних ризиків, вивчення новітніх практик захисту та розробка стратегій безпеки для Kubernetes стають критично важливими. Це не лише допоможе уникнути потенційних атак, але й підвищить довіру користувачів до середовища, що, у свою чергу, сприятиме збільшенню конверсії та доходів бізнесу.

Kubernetes, як платформа оркестрації контейнерів, має власний набір потенційних вразливостей, що можуть бути мішенями для зловмисників. Серед найбільш поширених типів атак на Kubernetes можна виділити:

- Проникнення через API-сервер: Некоректно налаштований API-сервер Kubernetes може бути доступний в Інтернеті без достатнього рівня захисту, що дозволяє атакувальникам отримувати, змінювати або видаляти ресурси в кластері.

- Витоки інформації через etcd: etcd є основною базою даних для Kubernetes, яка містить усю конфігурацію та секрети. Неналежно захищений etcd може стати джерелом витоку важливої інформації.
- Ескалація привілеїв через Pod: Якщо зловмисник може запустити Pod в кластері (наприклад, через компроментований контейнер), він може намагатися отримати привілеїї на більш високому рівні, ніж задумано.
- Атаки між контейнерами: Якщо різні контейнери запущені в одному namespace без відповідних політик мережевої безпеки, зловмисники можуть атакувати один контейнер з іншого.
- Неналежне використання RBAC: Якщо Role-Based Access Control (RBAC) неналежно налаштований, користувачі або додатки можуть отримати більше привілеїв, ніж їм потрібно, що стає потенційним місцем для атак.
- Використання небезпечних або компрометованих контейнерних образів: Зловмисники можуть використовувати відомі вразливості в контейнерах або додавати шкідливий код до образів.

Ці та інші потенційні загрози вимагають ретельного підходу до налаштування, моніторингу та захисту Kubernetes-середовища. Вивчення цих типів атак допоможе розробити ефективні стратегії захисту та механізми реагування на інциденти.

2.1 Огляд існуючих вразливостей і методи їх запобігання згідно з Kubernetes OWASP Top 10

Однією з основних проблем під час використання Kubernetes є забезпечення надійності заходів безпеки та врахування всіх потенційних загроз. Усвідомлюючи цю потребу, OWASP представив Kubernetes Top 10, висвітлюючи найімовірніші ризики.

OWASP Kuberenetes служать безцінною інформацією та порадами для фахівців із безпеки та розробників. Вони узгоджуються з іншими парадигмами безпеки, допомагаючи групам реагування на інциденти розуміти загрози, пов'язані з Kubernetes. Методології MITER ATT&CK також часто згадуються для документування стратегій зловмисників, пропонуючи синім командам зрозуміти оптимальні методи захисту. Крім того, можна звернутися до моделі загроз Kubernetes для повного розуміння всіх потенційних точок зламу та основних шляхів атаки.

Також класифікує ризики на основі їх загальної частоти або ймовірності. У нашому аналізі ми незначно їх реорганізували. Деякі ризики об'єднали в ширші категорії, як-от неправильні конфігурації, моніторинг або вразливості. Ми також пропонуємо певні інструменти або методи для перевірки ваших налаштувань і забезпечення ефективності вашої безпеки.

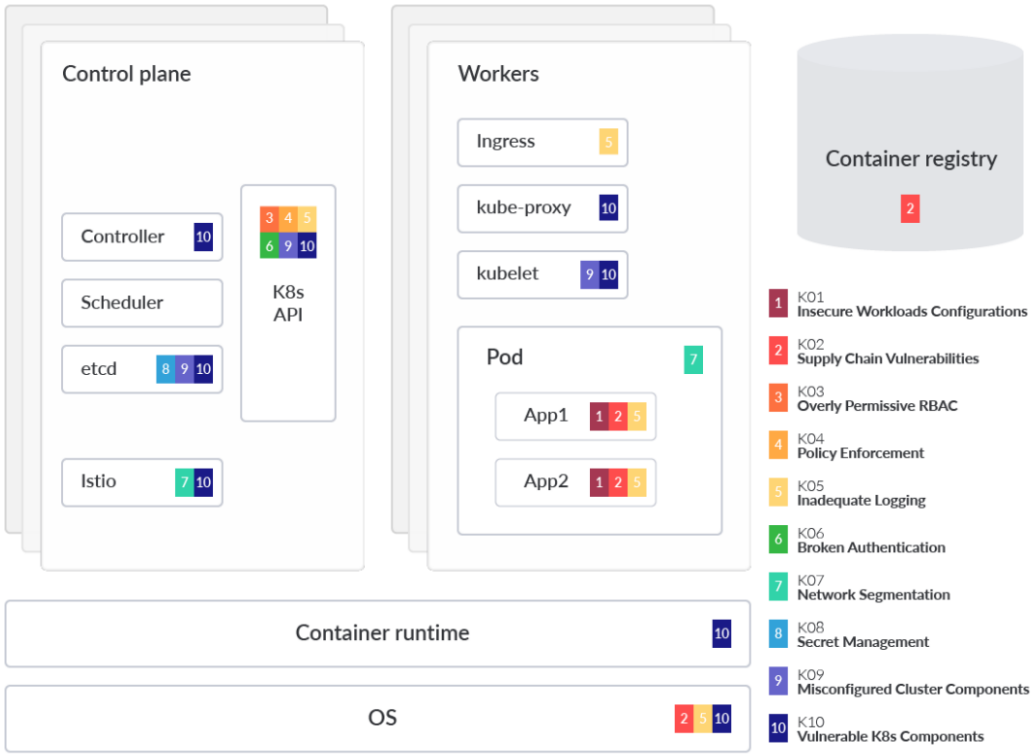


Рисунок 2.1 – Компоненти Kubernetes на які впливає кожен з ризиків [10]

2.1.1 Misconfigurations

2.1.1.1 Insecure Workload configuration

Безпека є головним пріоритетом для всіх основних хмарних служб. Такі гіганти, як AWS, GCP і Azure, включили численні функції безпеки, включаючи можливості ізольованого програмного середовища, віртуальні брандмауери та автоматичні оновлення для своїх основних служб. Це забезпечує максимальний захист бізнесу. Ці положення допомагають зменшити деякі типові проблеми безпеки, пов'язані з локальною інфраструктурою. Проте важливо зазначити, що хмарні платформи працюють за принципом спільної безпеки. Це означає, що хоча хмарний провайдер пропонує певні захисні заходи, користувачі також несуть частину відповідальності за безпеку, зокрема в своєму робочому середовищі. Ступінь відповідальності може відрізнятися залежно від моделі використання хмари та обраної конкретної служби.

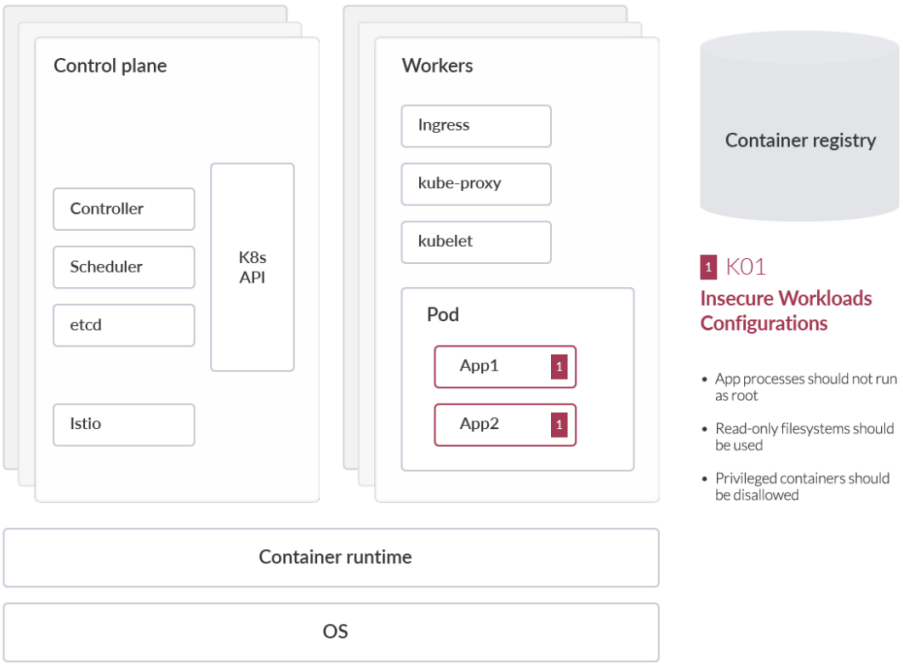


Рисунок 2.2 – Компоненти Kubernetes вразливі до «Insecure Workload configuration» [10]

Привілейовані контейнери мають бути заборонені. Якщо встановити для контейнера статус привілейованого в Kubernetes, контейнер може отримати доступ до додаткових ресурсів і можливостей ядра хоста. Робочі навантаження, які виконуються як root, у поєднанні з привілейованими контейнерами можуть бути руйнівними, оскільки користувач може отримати повний доступ до хосту.

Приклад неправильно налаштованого Kubernetes Pod наведений нижче:

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx
  namespace: default
spec:
  containers:
    - image: alpine:1.14.2
      command:
        - /bin/sh
        - "-c"
        - "sleep 45m"
      imagePullPolicy: IfNotPresent
      name: alpine-modified
      securityContext:
        privileged: true
```

Запуск модуля в привілейованому режимі означає, що модуль може отримати доступ до ресурсів хоста та можливостей ядра.

```
PS C:\Users\VladislavZelenin\Documents\hws\diploma_practice> kubectl get pods
NAME                                READY   STATUS    RESTARTS        AGE
nginx-deployment-7fb96c846b-2jpjr   1/1     Running   1 (9m32s ago)   154m
nginx-deployment-7fb96c846b-cxgnp   1/1     Running   1 (9m32s ago)   154m
nginx-deployment-7fb96c846b-md7ts   1/1     Running   1 (9m32s ago)   154m
PS C:\Users\VladislavZelenin\Documents\hws\diploma_practice> kubectl exec nginx-deployment-7fb96c846b-cxgnp -- whoami
root
PS C:\Users\VladislavZelenin\Documents\hws\diploma_practice>
```

Рисунок 2.3 – Результат виконання команди всередині Kubernetes Pod

Для запобігання можна налаштувати securityContext наступним чином:

```yaml
runAsUser: 1000  # Specify the user ID to run the container
runAsGroup: 1000 # Specify the group ID to run the container
```

В такому випадку под буде запускатися під звичайним користувачем. Іншим способом запобігання привілейованим под є створення/зміна .rego файлу з контролера доступу OPA Gatekeeper і має він виглядати наступним чином:

```
package kubernetes.customAdmission

deny[output] {
    container := input_containers[i]
    container.securityContext.privileged
    output := sprintf("Security alert! Container with privileged mode detected: %v,
securityContext: %v",
[container.name, container.securityContext])
}
```

І при спробі створення привілейованого контейнеру отримаємо наступне повідомлення: «Error from server (Security alert! Container with privileged mode detected: nginx, securityContext: {"privileged": true}): error when creating "STDIN": admission webhook "validating-webhook.openpolicyagent.org"».

2.1.1.2 Misconfigured cluster components

Наступною вразливістю яка відноситься до “Misconfigurations” є Misconfigured Cluster Components і торкається вона наступних компонентів:

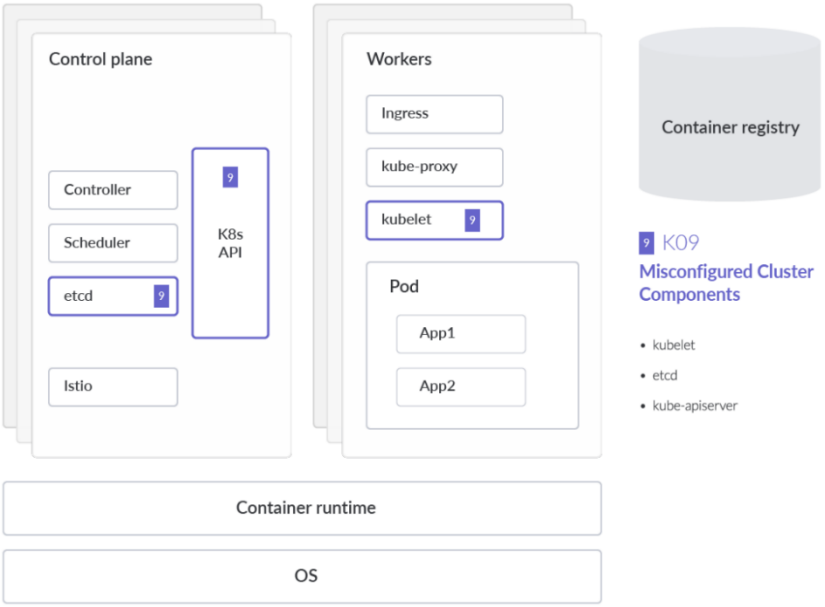


Рисунок 2.4 - Компоненти Kubernetes на які впливає «Misconfigured Cluster Components» [10]

Неправильні конфігурації в основних компонентах Kubernetes зустрічаються набагато частіше, ніж очікувалося. Щоб запобігти цьому, безперервний і

автоматичний аудит маніфестів IaC і K8s (YAML) замість необхідності перевіряти їх вручну зменшить кількість помилок конфігурації.

Однією з найнебезпечніших неправильних конфігурацій є налаштування анонімної автентифікації в Kubelet, яке дозволяє неавтентифіковані запити до Kubelet. Наполегливо рекомендуємо перевірити вашу конфігурацію Kubelet і переконатися, що для описаного нижче прапора встановлено значення false.

```
#bad
  --anonymous-auth=true
#good
  --anonymous-auth=false
```

2.1.1.3 Overly-permissive RBAC Configurations

Наступною вразливою складовою є **Overly-permissive RBAC Configurations**. Контроль доступу на основі ролей (RBAC) — це метод регулювання доступу до ресурсів комп’ютера або мережі на основі ролей окремих користувачів у вашій організації. Неправильна конфігурація RBAC може дозволити зловмиснику підвищити привілеї та отримати повний контроль над усім кластером.

На рисунку 2.5 зазначені компоненти Kubernetes на які впливає неправильна конфігурація RBAC:

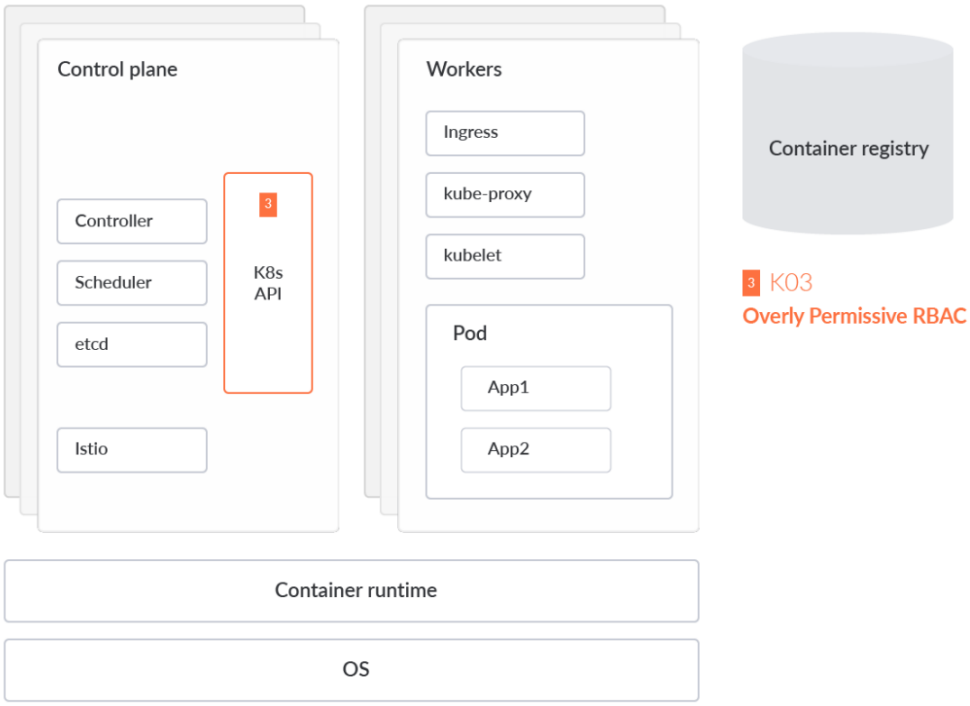


Рисунок 2.5 – Overly Permissive RBAC [10]

Налаштувати правила RBAC відносно просто. Наприклад, якщо ви хочете встановити м'яку політику, яка дозволяє операції CRUD лише для читання (такі як отримання, перегляд, список) на модулях у просторі імен мережі за замовчуванням кластера Kubernetes, одночасно обмежуючи дії «Створити», «Оновити» або «Видалити». на цих модулях політика виглядатиме так:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
Metadata:
  namespace: default
  name: pod-reader
Rules:
  - apiGroups: [""]
    resources: ["pods"]
    verbs: ["get", "watch", "list"]
```

Сценарій і приклад атаки: Рішення для моніторингу кластера з відкритим вихідним кодом було розгорнуто в приватному кластері Kubernetes групою інженерів інфраструктури. Це рішення поставляється з вбудованим веб-інтерфейсом, призначеним для аналізу трафіку та налагодження. Однак через недогляд у

пов’язаному маніфесті служби, де використовується тип: LoadBalancer, створюється балансувальник навантаження AWS ALB із загальнодоступною IP-адресою, ненавмисно роблячи інтерфейс користувача доступним через Інтернет.

Для цього уявного інструменту налаштування RBAC виглядають так:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: custom-sa-namespace-leader
  namespace: prod-env
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: admin
subjects:
  - apiGroup: rbac.authorization.k8s.io
    kind: User
    name: system:serviceaccount:prod-env:custom-default
```

Зловмисник виявляє відкритий веб-інтерфейс і отримує доступ до оболонки контейнера, що працює в кластері. Використовуючи маркер облікового запису служби за замовчуванням із простору імен prd, який використовує веб-інтерфейс, зловмисник імітує його. Це дозволяє їм викликати API Kubernetes і виконувати високорівневі операції, такі як розкриття секретів у просторі імен «kube-system». Це порушення безпеки є результатом roleRef, який надає цьому обліковому запису служби властиві привілеї адміністратора для всього кластера.

Та інший спосіб при непотрібному використанні дозволу LIST: відповідь списку містить усі елементи повністю, а не лише їх назву. Облікові записи з дозволом LIST не можуть отримати певний елемент з API, але отримають усі їх повністю, коли вони створять список.

kubectl приховує це за замовчуванням, показуючи лише назви об’єктів, але він містить усі атрибути цих об’єктів.

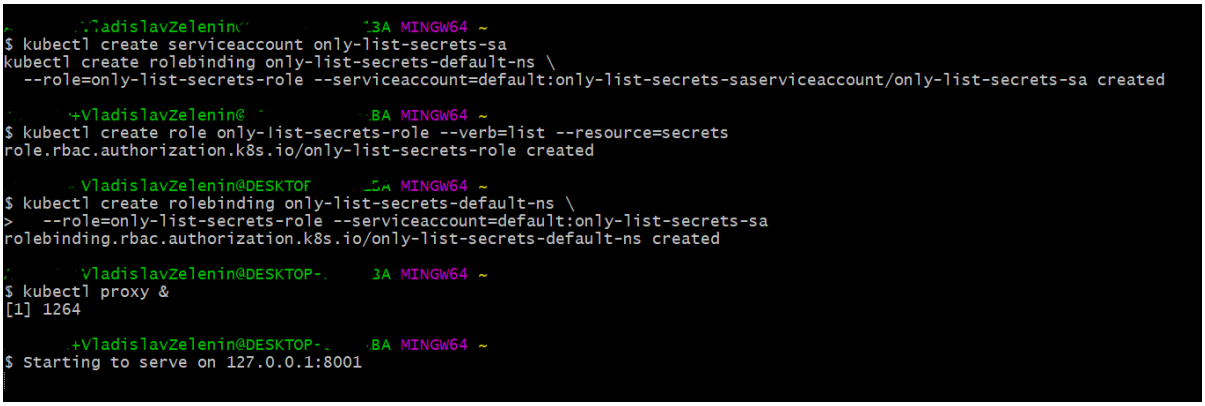


Рисунок 2.6 – Створення ролі і секретів

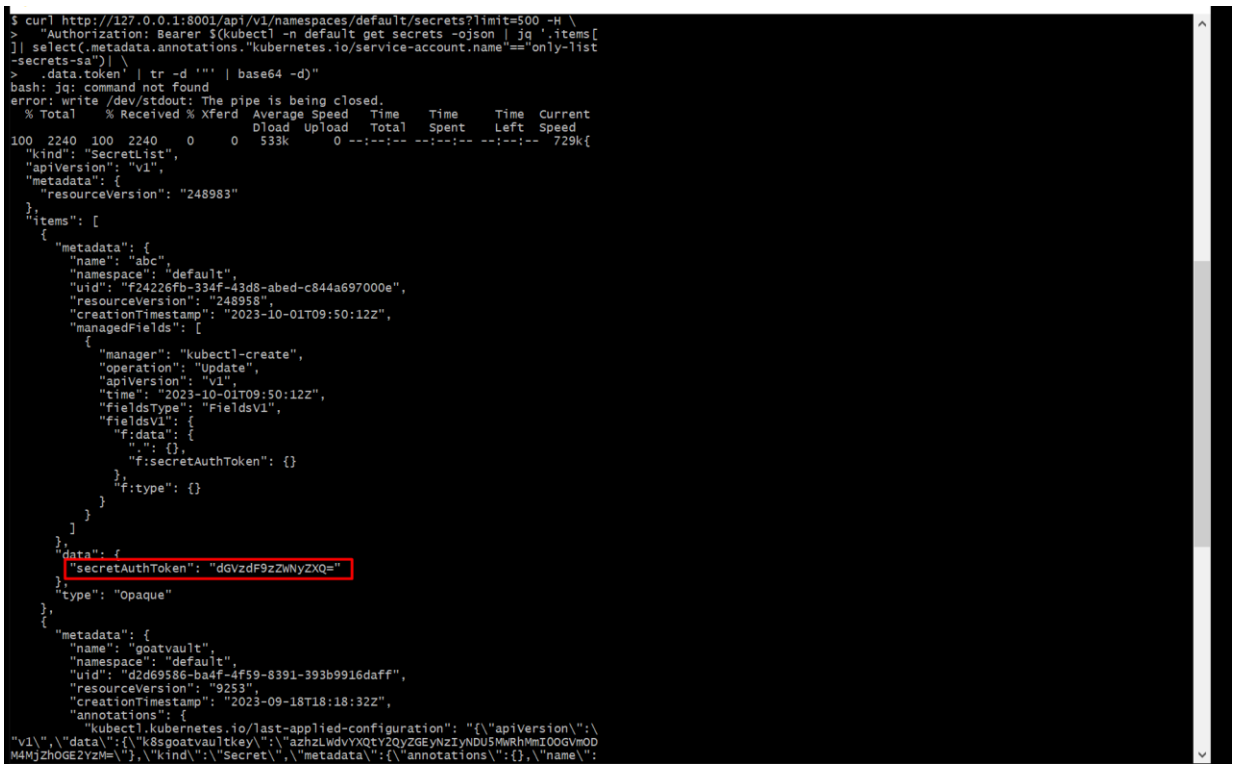


Рисунок 2.7 – Безперешкодне читання secretAuthToken

2.1.1.4 Missing network segmentation controls

Kubernetes за замовчуванням визначає так звану «плоску мережу».

Це дозволяє робочим навантаженням вільно спілкуватися між собою без попереднього налаштування. Однак вони можуть робити це без будь-яких обмежень. Якби зловмисник міг використати запущене робоче навантаження, він, по суті, мав би доступ до викрадання даних усіх інших модулів у кластері. Оператори кластерів, які зосереджені на архітектурі нульової довіри у своїй організації, захочуть уважніше

ознайомитися з мережевою політикою Kubernetes, щоб забезпечити належне обмеження послуг.

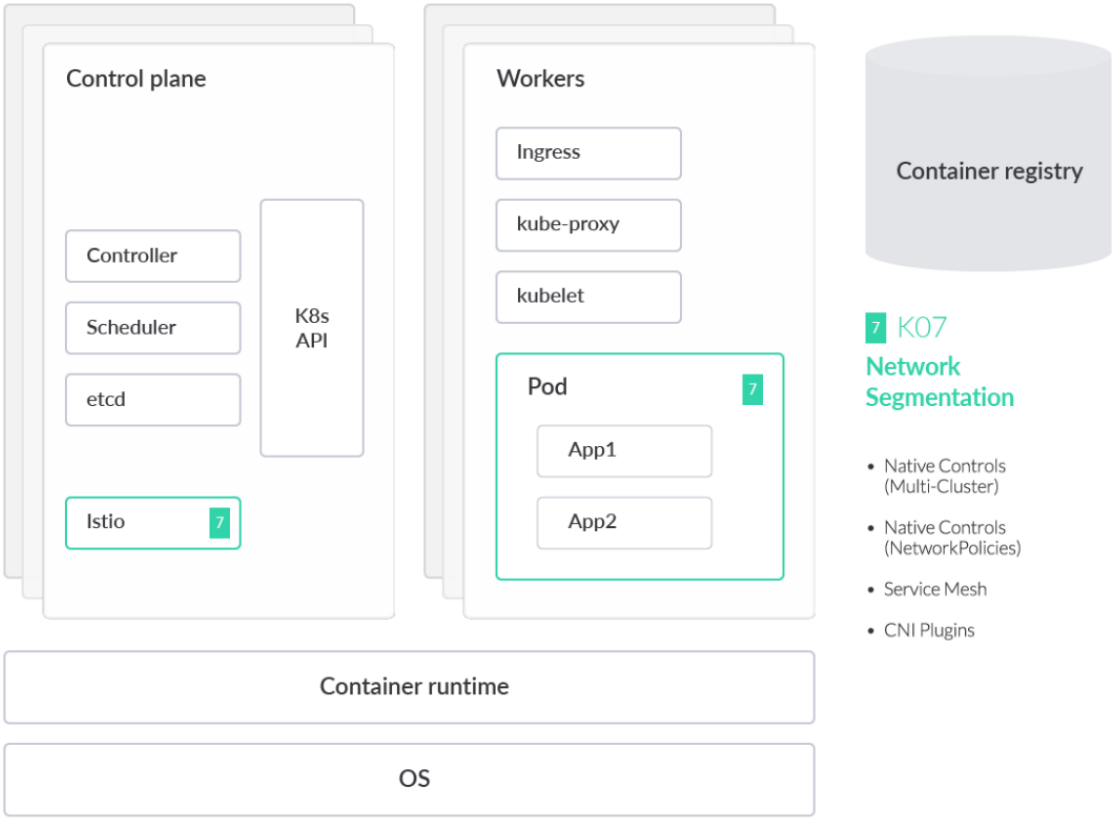


Рисунок 2.8 – Missing network segmentation controls [10]

Сценарій атаки: контейнер WordPress у кластері стає вразливим у середовищі без сегментації мережі. Враховуючи образ на основі Ubuntu, зловмисник використовує такі вбудовані мережеві інструменти, як dig і curl, для дослідження мережі. Зловмисник ідентифікує внутрішній API на порту 6379, зазвичай пов'язаному з Redis. Згодом вони перевіряють цей передбачуваний внутрішній мікросервіс Redis, який призначений лише для взаємодії з серверним API, за допомогою curl. Як наслідок, доступ до даних здійснюється незаконно, і вони змінюються.

Застосування суворої стратегії NetworkPolicy або сервісної мережі обмежило б будь-які спроби мережевого підключення від WordPress до Redis, перешкоджаючи таким вторгненням.

В іншому випадку менш критична веб-програма зламана в кластері, який також не має мережевого поділу. Нападнику вдається запитати URL-адресу метаданих, отримуючи файл kube-env. Цей файл містить ключі сертифікатів із детальним описом процедури початкового завантаження, що дозволяє зловмиснику потенційно зареєструвати себе як вузол, викрасти секрети та розширити свій доступ.

Наступна мережева політика може запобігти доступу користувачів до URL-адреси метаданих, пропонуючи рівень захисту від таких атак:

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: deny-metadata-access-unique
spec:
  egress:
    - to:
        - ipBlock:
            cidr: 0.0.0.0/0
            except:
              - 172.17.0.1/32
  podSelector: {}
  policyTypes:
    - Egress
```

2.1.2 Lack of visibility

2.1.2.1 Inadequate Logging and Monitoring

Inadequate Logging and Monitoring – Kubernetes за замовчуванням надає функцію журналювання аудиту. Журнал аудиту показує різноманітні події, пов’язані з безпекою, у хронологічному порядку. Ці дії можуть генеруватися користувачами, програмами, які використовують Kubernetes API, або самою площиною керування.

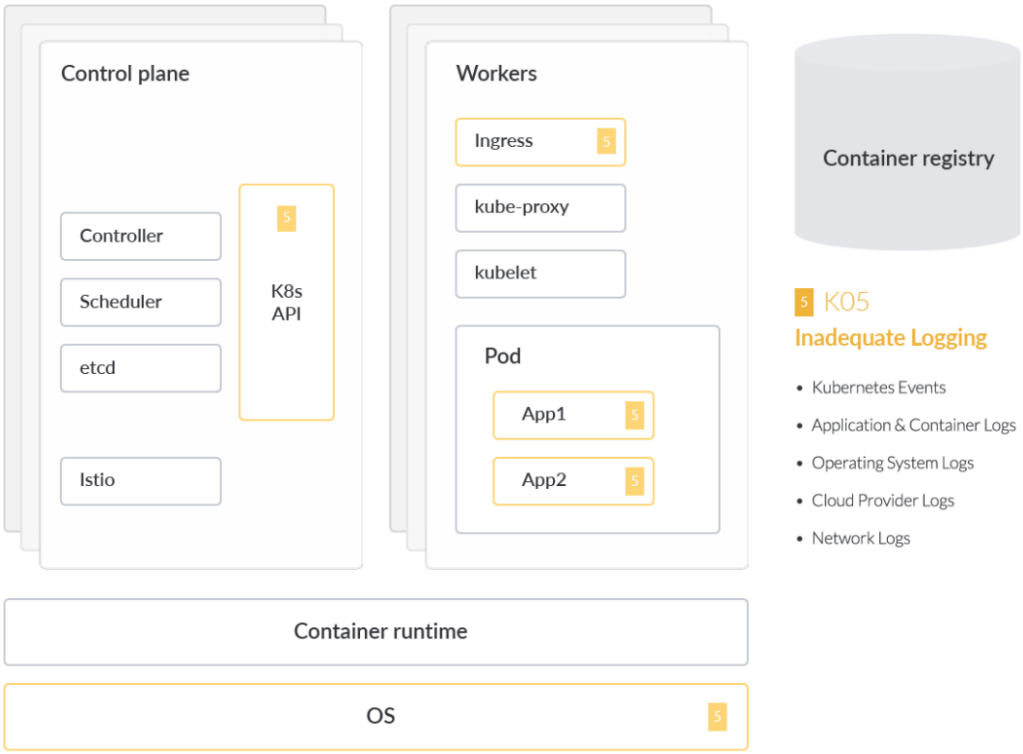


Рисунок 2.9 - Inadequate Logging and Monitoring [10]

Однак є й інші джерела журналів, на яких варто зосередитися, не лише журнали аудиту Kubernetes. Вони можуть включати журнали операційної системи конкретного хоста, журнали мережевої активності (наприклад, DNS, за якими можна відстежувати доповнення Kubernetes CoreDNS) і хмарні постачальники, які також працюють як основа для Kubernetes Cloud.

Без централізованого інструменту для зберігання всіх цих спорадичних джерел журналів нам було б важко використовувати їх у разі порушення. Ось де знадобляться такі інструменти, як Prometheus, Grafana та Falco.

2.1.2.2 Lack of Centralized Policy Enforcement

Наступною вразливістю K8S є **Lack of Centralized Policy Enforcement.** Застосування політик безпеки стає складним завданням, коли потрібно забезпечити виконання правил у мультикластерних і багатохмарних середовищах. За

замовчуванням командам із безпеки потрібно буде окремо керувати ризиками в кожному з цих гетерогенних середовищ.

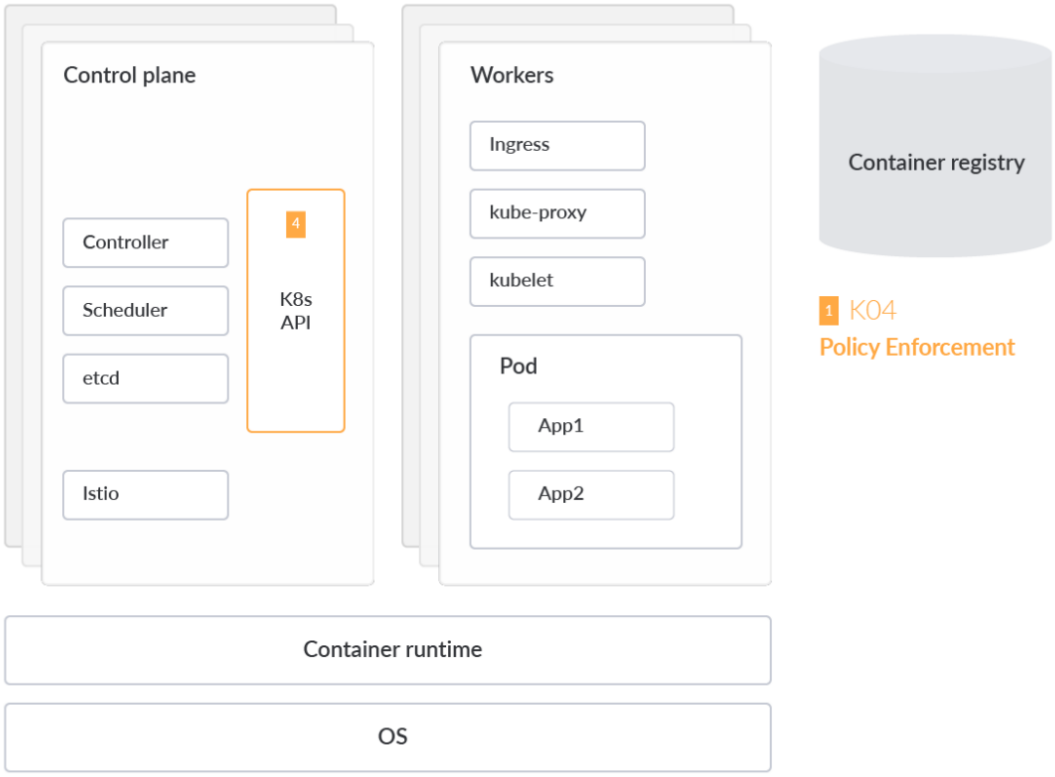


Рисунок 2.10 - Lack of Centralized Policy Enforcement [10]

Немає стандартного способу виявлення, виправлення та запобігання неправильним конфігураціям із централізованого розташування, тобто кластери потенційно можуть залишитися відкритими для компромісу.

Перед тим як перейти на сервер Kubernetes API, контролер доступу перехоплює запити. Кожен запит проходить автентифікацію та авторизацію перед тим, як буде прийнято рішення про дозвіл або заборону дії. Для ілюстрації розглянемо конфігурацію контролера допуску:

```
apiVersion: apiserver.config.k8s.io/v1
kind: UniqueAdmissionConfiguration
plugins:
  - name: CustomImagePolicyWebhook
    configuration:
      customImagePolicy:
```

```
      kubeConfigFile: "/etc/kubernetes/unique-kubeconfig.conf"
      allowTTL: 60
      denyTTL: 60
      retryBackoff: 600
      defaultAllow: false
```

Конфігурація ImagePolicyWebhook, яка посилається на файл у форматі kubeconfig, встановлює підключення до серверної частини. Мета цього контролера доступу полягає в тому, щоб забезпечити внутрішній зв'язок через TLS.

Параметри allowTTL і denyTTL визначають тривалість кешування (у секундах) для схвалення та відмови відповідно. Контролери доступу можуть обмежувати запити, які створюють, видаляють, змінюють об'єкти або підключаються до проксі-серверів.

На жаль, ресурсом AdmissionConfiguration потрібно керувати на кожному кластері незалежно. Якщо цей файл пропущено в одному з наших кластерів, умова політики втрачається. На щастя, такі інструменти, як Open Policy Agent (OPA) Kube-Mgmt, допомагають керувати політиками та даними примірників OPA в Kubernetes, усуваючи потребу в індивідуальному керуванні контролерами доступу.

Інструмент kube-mgmt автономно визначає політики та дані JSON у Kubernetes ConfigMaps та імпортує їх в OPA. Політики можна вимкнути за допомогою прапорця --enable-policy=false, а дані також можна вимкнути за допомогою --enable-data=false.

Контроль доступу становить важливий аспект стратегії безпеки контейнера, який забезпечує виконання необхідних політик у контексті Kubernetes і забезпечує останній рівень захисту для вашого кластера. Сканування зображень, яке обговорюватиметься пізніше, також можна застосувати через контролер доступу Kubernetes.

Стандартизація розгортання конфігурацій політики безпеки в усіх кластерах є обов'язковою, особливо якщо вони відображають ідентичні конфігурації. У сценаріях, коли конфігурації кластерів суттєво відрізняються, може знадобитися спеціальна

політика безпеки. Незалежно від сценарію важливо визначити, які політики безпеки реалізовано в кожному середовищі кластера. Тут Falco стає важливою.

Якщо припустити, що kube-mgmt не використовується та централізоване керування контролерами допуску відсутнє, користувач може ненавмисно створити ConfigMap, відкриваючи приватні облікові дані в маніфесті ConfigMap через відсутність контролера допуску в новоствореному кластері. Falco може попередити адміністраторів про таку поведінку за допомогою єдиного правила:

```
- rule: GenerateAlert_ConfigmapPrivateCredentialsChange
  desc: >
    Identify creation/modification of a configmap embedding a private credential.
  condition: k8s_evt and cm and kmodify and has_sensitive_data
  output: >-
    [K8s Alert] Configmap with sensitive data identified (User=%ka.user.name,
Action=%ka.verb,
    ConfigMap=%ka.req.configmap.name, Namespace=%ka.target.namespace)
  priority: critical
  source: k8s_audit_logs
  append: false
  exceptions:
    - name: trusted_configmaps
      fields:
        - ka.target.namespace
        - ka.req.configmap.name
```

У наведеному вище правилі Falco ми шукаємо журнали аудиту Kubernetes, щоб показати приклади приватних облікових даних, які можуть бути представлені в ConfigMaps у будь-якому просторі імен. Приватні облікові дані визначаються як будь-яка з наведених нижче умов

```
condition: (ka.req.configmap.obj matches "aws[_-]access[_-]key[_-]id" or
  ka.req.configmap.obj matches "aws[_-]s3[_-]access[_-]key[_-]id" or
  ka.req.configmap.obj matches "password" or
  ka.req.configmap.obj matches "passphrase" or
  ka.req.configmap.obj matches "private[_-]key" or
  ka.req.configmap.obj matches "api[_-]token")
```

2.1.2.3 Secrets Management Failures

Останньою вразливістю яка відноситься до Lack of Visibility є **Secrets Management Failures.**

Kubernetes представляє «секрети» як об’єкти, призначені для захисту конфіденційних деталей, таких як паролі чи маркери, від вбудовування в код програми. Посилаючись на секрети K8s у специфікації модуля, інженери можуть запобігти прямому вбудовуванню облікових даних і конфіденційної інформації в маніфести модуля або зображення контейнера, тим самим покращуючи безпеку та керованість.

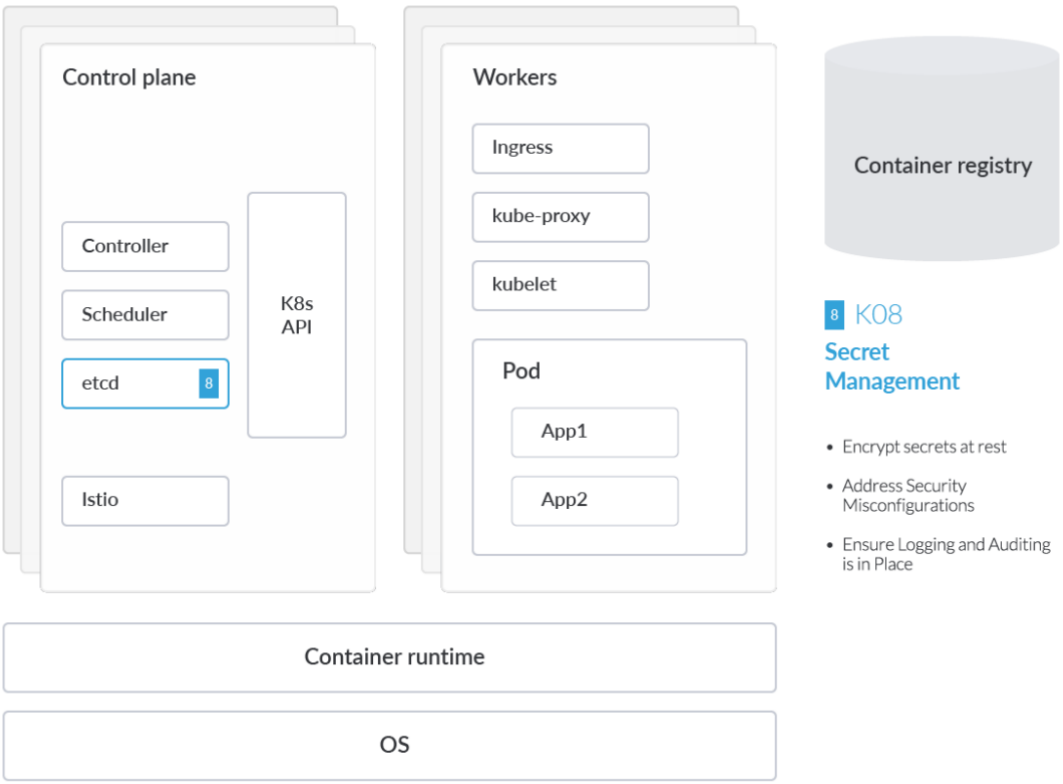


Рисунок 2.11 - Secrets Management Failures [10]

Незважаючи на безпечний дизайн, K8s Secrets не схильні до компромісів. По суті, внутрішній механізм секретів K8s служить абстракцією, при цьому дані зрештою зберігаються в базі даних etcd, що робить критично важливим перевірити, як обробляються ключі та облікові дані в секретах K8s. Компанії повинні стратегічно оцінити управління секретами, охоплюючи такі аспекти, як зберігання та доступ до секретів K8s. Крім того, Kubernetes розширює додаткові засоби контролю безпеки,

включаючи шифрування даних у стані спокою, контроль доступу та журналювання, щоб зміцнити рівні безпеки.

Завжди вдосконалюйте свою стратегію безпеки за допомогою додаткових практик та інструментів безпеки, забезпечуючи ретельний і надійний підхід до захисту вашого середовища Kubernetes.

Значна вразливість, пов'язана з використанням Kubernetes бази даних etcd, полягає в тому, що вона містить усі дані, доступні через Kubernetes API, потенційно надаючи зловмиснику широкий доступ до секретів. Отже, шифрування секретів у спокої стає першорядним.

Починаючи з версії 1.7, Kubernetes полегшив підтримку шифрування в стані спокою. Реалізація цієї функції шифрує секретні ресурси в etcd, захищаючи вміст секретів від неавторизованого доступу до ваших резервних копій etcd. Хоча ця функція зараз знаходиться в бета-версії та не активована за замовчуванням, вона вводить додатковий рівень захисту від сценаріїв, коли резервні копії незашифровані або зловмисник захищає доступ для читання до тощо.

Нижче наведено приклад, який ілюструє створення спеціального ресурсу EncryptionConfiguration:

```
apiVersion: apiserver.config.k8s.io/v1
kind: EnhancedEncryptionConfiguration
resources:
  - resources:
      - secrets
    providers:
      - aescbc:
          keys:
            - name: secureKeyAlpha
              secret: c2VjcmV0X3Rlc3Q=
            - name: secureKeyBeta
              secret: NjY2cXNkcXdlcXdyZnE=
      - secretbox:
          keys:
            - name: sboxKeyAlpha
              secret: ZGhqZ3M0NXQ1NHQ1dzR5Z2F3
      - identity: {}
```

2.1.3 Vulnerability Management

2.1.3.1 Supply chain vulnerabilities

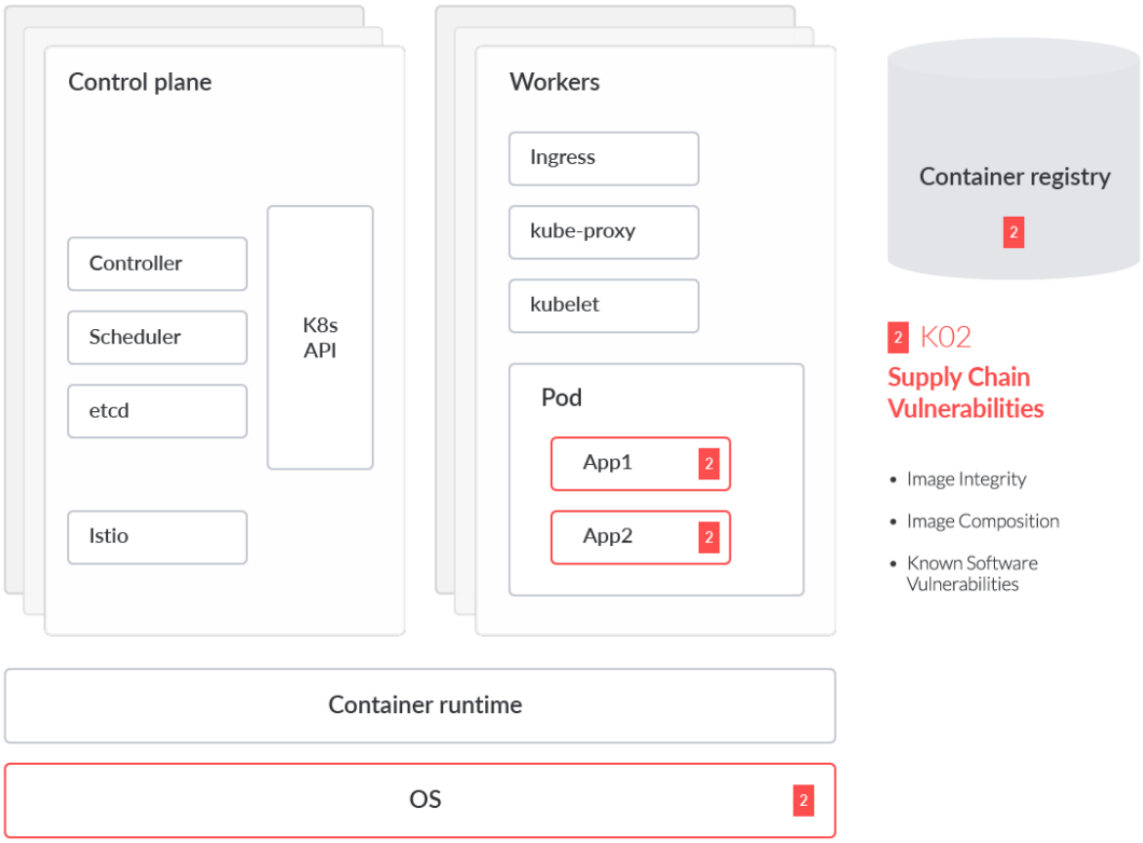


Рисунок 2.12 - Supply Chain Vulnerabilities [10]

Сплеск атак на ланцюги поставок, прикладом якого є злом SolarWinds, підкреслює переважаючу загрозу кібербезпеці. Програмне рішення «Orion» від SolarWinds було проникнуте групою загроз APT29, яку також називають Cosy Bear. Ця підступна атака нульового дня розгорнулася таємно, залишивши клієнтів SolarWinds із впровадженнями Orion непомітними для компромісу.

Інцидент із SolarWinds — одиничний приклад у ширшому діапазоні вразливостей безпеки. Що стосується Kubernetes, навіть одне робоче навантаження в контейнері може залежати від сотень сторонніх компонентів і залежностей, що ускладнює перевірку походження на кожному етапі. Ускладнення охоплюють кілька

областей, включаючи цілісність зображення, композицію зображення та наявність відомих уразливостей програмного забезпечення.

Розглянемо кожен із цих аспектів більш детально.

Navigating Dependencies - управління великою кількістю ресурсів у кластері може перетворитися на «лабіринтне» завдання, де нагляд за всіма взаємозв'язками між ними стає невловимим. Навіть кластери, які вважаються «маленькими», можуть мати більше послуг, ніж очікувалося, завдяки багатогранній природі контейнеризації та оркестровки. Підтримка чіткої карти всіх служб, ресурсів і залежностей стає дедалі складнішою при роботі з розподіленими командами в мультикластерних або багатохмарних ландшафтах.

За замовчуванням Kubernetes не пропонує інструмент для графічного представлення залежностей між вашими розгортаннями, службами, постійними претензіями на томи (PVC) тощо. KubeView, зразковий інструмент із відкритим кодом, дозволяє користувачам спостерігати та перевіряти внутрішньокластерні залежності за допомогою ілюструє об'єкти API та їхні взаємозв'язки. Він отримує дані безпосередньо з Kubernetes API, забезпечуючи кольорове представлення (червоний/зелений) стану та працездатності певних об'єктів, як-от Pods, ReplicaSets і Deployments.

Registry Considerations - реєстр слугує серверною програмою без збереження стану, що масштабується, яка зберігає та полегшує розповсюдження зображень контейнерів.

Ресурси Kubernetes, які використовують зображення (такі як пакети та розгортання), використовують секрети imagePull для підтримки облікових даних, необхідних для автентифікації різних реєстрів зображень. Подібно до багатьох проблем, розглянутих у цьому розділі, стандартне розгортання Kubernetes за своєю суттю не передбачає методу сканування зображень на наявність вразливостей.

Навіть за наявності приватного спеціального реєстру зображень сканування зображень на наявність вразливостей є ключовим. Kubernetes не пропонує інтегрований метод для цього за замовчуванням. Бажано сканувати ваші зображення під час процесу збирання у ваших конвеєрах CI/CD як компонент стратегії безпеки зсуву вліво.

Sysdig надає вичерпні технічні вказівки, включаючи приклади, щодо впровадження цього для поширених служб CI/CD, таким чином запроваджуючи додатковий рівень безпеки для запобігання вразливостям у ваших конвеєрах:

- GitHub actions
- Gitlab pipelines
- Azure pipelines
- Jenkins

Додатковий рівень безпеки передбачає процес підписання та перевірки зображень, надісланих до наших реєстрів або сховищ. Цей процес мінімізує атаки на ланцюг постачання, гарантуючи автентичність і цілісність. Він захищає нашу розробку та розгортання Kubernetes, пропонуючи розширений контроль над інвентарем контейнерів, що працює в будь-який момент.

2.1.3.2 Broken authentication mechanisms

Наступною вразливістю є Broken Authentication Mechanisms - декільком об’єктам потрібен доступ до API Kubernetes. Автентифікація є першою перешкодою для цих запитів. Автентифікація в Kubernetes API здійснюється через HTTP-запит, і метод автентифікації може відрізнятися від кластера до кластера. Якщо запит не може бути автентифікований, він відхиляється зі статусом HTTP 401.

OWASP рекомендує, щоб незалежно від обраного механізму автентифікації ми змусили людей надати другий метод автентифікації. Наприклад, якщо ви використовуєте IAM і 2FA не ввімкнено, можна виявити це під час виконання у вашому хмарному середовищі або середовищі Kubernetes, щоб пришвидшити

виявлення та відповідь. Для цього ми можемо використовувати Falco, механізм виявлення загроз з відкритим кодом, який запускає сповіщення під час виконання відповідно до набору правил у форматі YAML.

```
- rule: NonMFAConsoleAccessDetected
  desc: Flags an instance of a console login occurring without MFA enforcement.
  condition: >-
    aws.eventName="ConsoleLogin" and not aws.errorCode exists and
    jevt.value[/userIdentity/type]!="AssumedRole" and
    jevt.value[/responseElements/ConsoleLogin]="Success" and
    jevt.value[/additionalEventData/MFAUsed]="No"
  output: >-
    Alert: Console access without MFA identified (user=%aws.user, IP
address=%aws.sourceIP, Region=%aws.region)
  priority: critical
  source: aws_cloudtrail_audit
  append: false
  exceptions: []
```

Falco допомагає визначити, де існують незахищені входи. В даному випадку це вхід в консоль AWS без MFA. Однак, якби зловмисник міг отримати доступ до хмарної консолі без додаткової авторизації, він, імовірно, міг би отримати доступ до Amazon Elastic Kubernetes Service (EKS) через CloudShell.

2.1.3.3 Outdated and Vulnerable Kubernetes Components

І останньою вразливістю є **Outdated and Vulnerable Kubernetes Components**.

Ефективний vulnerability management в Kubernetes це доволі складний процес, але є декілька порад, які допоможуть з цим.

Адміністратори Kubernetes мають постійно стежити за найновішими базами даних CVE, стежити за новими розкриттями вразливостей і за потреби впроваджувати відповідні виправлення. Якщо цього не зробити, кластери Kubernetes можуть бути вразливими до розпізнаних уразливостей, таким чином спрощуючи завдання зловмисника щодо розгортання методів захоплення повного контролю над інфраструктурою та, можливо, розширення їхнього охоплення у вашому хмарному клієнті, де розгорнуто кластери. [10]

Управління CVE стає особливо складним через значну кількість компонентів з відкритим кодом у Kubernetes у поєднанні з ритмом випуску проекту. У версії

Kubernetes 1.25 в альфа-версії було введено новий канал безпеки, який упорядковує й оновлює масив CVE, що впливає на компоненти Kubernetes.

2.1 Існуючі підходи до моніторингу та виявлення загроз в Kubernetes

Моніторинг загроз Kubernetes передбачає нагляд і оцінку стану безпеки кластера Kubernetes разом із його підключеними елементами. Цей процес вимагає збору та ретельного аналізу даних із кількох джерел усередині кластера, щоб виявити можливі загрози безпеці, виявити порушення та підтвердити дотримання встановлених протоколів безпеки. Завдяки застосуванню ефективних стратегій моніторингу компанії можуть активно виявляти та усувати порушення безпеки, спроби неавторизованого доступу та розбіжності у своїй структурі Kubernetes.

Для забезпечення надійного моніторингу безпеки Kubernetes вирішальною є цілісна стратегія, яка охоплює численні аспекти кластера. Ось декілька важливих дій для ефективного нагляду за безпекою в Kubernetes:

Збір даних журналу: ініціюйте та налаштуйте журналювання в кластерах Kubernetes для документування критичних подій і операцій. Журнали допомагають пролити світло на можливі вторгнення в безпеку, помилки конфігурації та сумнівну діяльність кластера.

Активація аудиту: увімкніть функції аудиту в Kubernetes, щоб відстежувати та контролювати всі запити до сервера API. Ця практика допомагає виявити несанкціоновані спроби підключитися до активів кластера або змінити їх.

Нагляд за мережевим трафіком. Використовуйте інструменти перевірки мережі, щоб реєструвати та досліджувати мережевий трафік у кластері, допомагаючи визначити незвичайні тенденції зв’язку або потенційні мережеві атаки.

Виявлення та протидія вторгненням: налаштуйте системи для виявлення та запобігання вторгненням (IDS/IPS), адаптовані до налаштувань Kubernetes. Ці механізми можуть виявляти та зупиняти зловмисні дії, такі як несхвалені розгортання контейнерів або спроби підвищити привілеї.

Прийняття безпеки контейнерів: використовуйте технології безпеки контейнерів, які мають вбудовані функції моніторингу для Kubernetes. Ці технології можуть перевіряти зображення контейнерів на наявність слабких місць, виявляти аномалії під час роботи та контролювати поведінку контейнера на предмет будь-яких незвичних операцій.

Сканування вразливостей: регулярно перевіряйте елементи Kubernetes, як-от робочі вузли, рівні керування та програми, на наявність поширених уразливостей. Це має вирішальне значення для визначення та усунення потенційних недоліків безпеки або неправильних налаштувань.

Впровадження SIEM: об’єднайте журнали та дані про інциденти безпеки з Kubernetes у рішення SIEM, централізувавши нагляд за безпекою та дозволяючи зіставляти та досліджувати випадки безпеки в усій інфраструктурі.

Звичайні аудити та оцінки: виконуйте регулярні перевірки та оцінки безпеки, щоб виявити будь-які недоліки в заходах безпеки Kubernetes і підтвердити дотримання встановлених у секторі протоколів і юридичних повноважень.

Існує безліч інструментів і платформ для покращення моніторингу безпеки Kubernetes. Ось деякі з них:

Prometheus: це популярний інструмент моніторингу з відкритим вихідним кодом, який забезпечує надійний моніторинг і оповіщення для кластерів Kubernetes.

Grafana: як правило, у поєднанні з Prometheus, Grafana є складною платформою для візуалізації та приладної дошки, яка використовується для відстеження та перевірки показників і журналів із Kubernetes.

Sysdig: ця платформа зосереджена на моніторингу контейнерів і Kubernetes, пропонуючи миттєве уявлення про стан безпеки кластерів Kubernetes. Він має такі можливості, як ідентифікація загроз, керування вразливими місцями та моніторинг відповідності.

Aqua Security: відома своєю зосередженістю на безпеці Kubernetes, ця контейнерна платформа безпеки забезпечує захист під час роботи, керує вразливими місцями та забезпечує застосування політик для кластерів Kubernetes.

Falco: Falco як інструмент безпеки з відкритим вихідним кодом, рідний для Kubernetes, використовує функції аудиту Kubernetes для виявлення незвичайної активності та можливих ризиків безпеки в кластері. [11]

Цей список інструментів не є вичерпним і існує велика кількість інших засобів моніторингу вразливостей Kubernetes, далі буде наведено список із найбільш застосованих засобів виявлення вразливостей:

Першою потужним засобом пошуку вразливостей є **kubelinter -** це інструмент для аналізу файлів Kubernetes YAML та діаграм Helm, який забезпечує їх відповідність різноманітним кращим практикам, особливо зосереджуючись на готовності та безпеці. Він проводить ретельні перевірки за замовчуванням, які дають цінні відомості про ваші файли Kubernetes YAML та діаграми Helm, допомагаючи командам рано та регулярно виявляти помилки в налаштуваннях безпеки та кращих практиках DevOps. До прикладів належать запуск контейнерів без прав root, використання мінімальних привілеїв та безпечне зберігання конфіденційних даних у секретах. KubeLinter можна налаштовувати, дозволяючи включати чи відключати перевірки та створювати унікальні перевірки відповідно до політики вашої організації. Якщо перевірка лінту не пройде, KubeLinter видає рекомендації щодо виправлення потенційних проблем та повертає ненульовий код виходу.

Для прикладу візьмемо підготовлений вразливий под наступного вигляду:

```
apiVersion: v1
kind: Pod
metadata:
```

```
  name: custom-ctx-demo-pod
spec:
  securityContext:
    runAsUser: 1001
    runAsGroup: 3001
    fsGroup: 2001
  volumes:
    - name: custom-ctx-vol
      emptyDir: {}
  containers:
    - name: custom-ctx-container
      image: alpine
      resources:
        requests:
          memory: "128Mi"
          cpu: "500m"
      command: [ "sh", "-c", "sleep 2h" ]
      volumeMounts:
        - name: custom-ctx-vol
          mountPath: /data/custom
      securityContext:
        allowPrivilegeEscalation: true
        readOnlyRootFilesystem: true
```

Проблема безпеки наступна: контейнер у цьому модулі не працює як файлова система лише для читання, що може дозволити йому записувати в кореневу ФС.

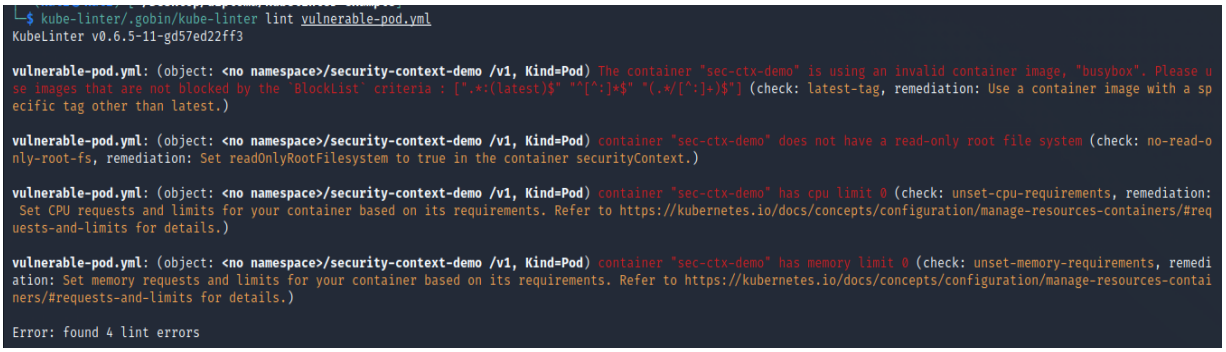


Рисунок 2.13 – Результат пошуку вразливостей за допомогою kube-linter

З результатів виконання команди лінтеру, можна зробити висновок, що він орієнтований на аналіз статичних кофігурацій поду k8s. Також він може використовуватись в CI/CD пайплайнах.

Наступною утилітою, яка допоможе знайти вразливостій в конфігурації yaml файлу є **kube-audit.** Kubeaudit, розроблений Shopify, є інструментом командного рядка для оцінки відповідності кластерів Kubernetes стандартам безпеки. Цей інструмент включає 14 окремих аудиторів, кожен з яких відповідає за перевірку

певних аспектів безпеки, таких як перевірка на використання надмірних привілеїв у контейнерах, виявлення небезпечних налаштувань за замовчуванням та перевірка включення безпеки AppArmor. Кожен аудитор у Kubeaudit має детальну документацію з прикладами та рекомендаціями. Результати аудиту можна сортувати за рівнями безпеки, що дозволяє користувачам визначати пріоритетність залежно від ступеню ризику. В цілому, kubeaudit забезпечує гнучкість у конфігурації та налаштуваннях, що робить його корисним інструментом у вирішенні питань безпеки в середовищі Kubernetes. [13]

Після успішного встановлення вищезазначеної утиліти протестуємо її на тій самій конфігурації що і попередня.

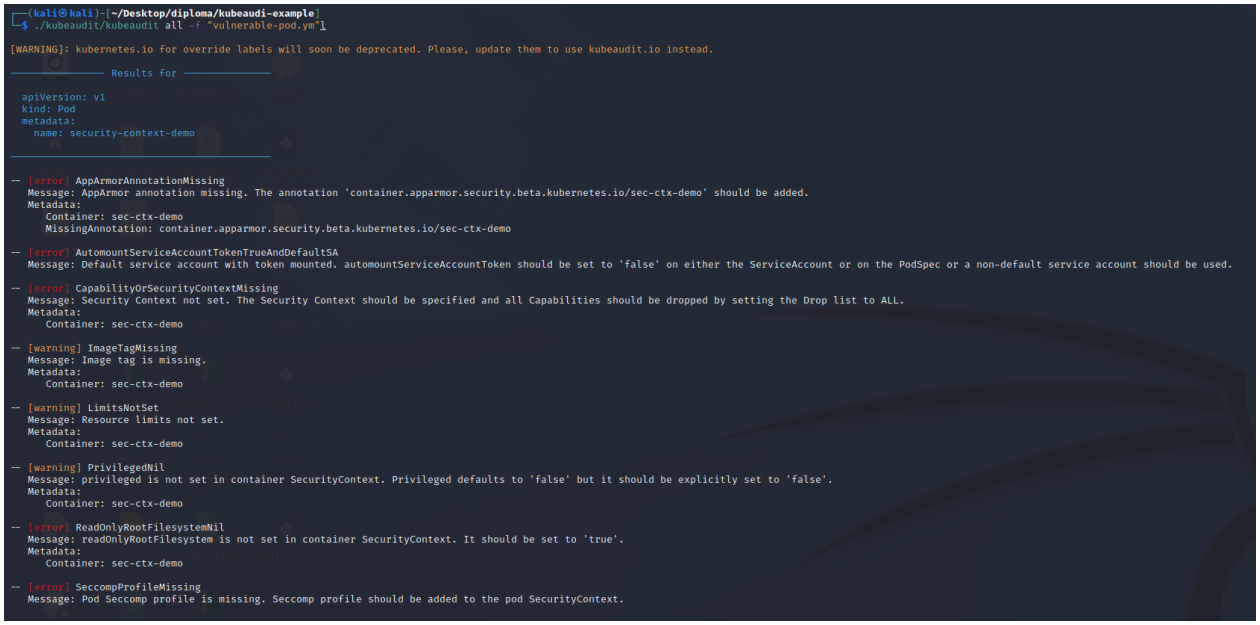


Рисунок 2.14 – Пошук вразливостей за допомогою утиліти kube-audit

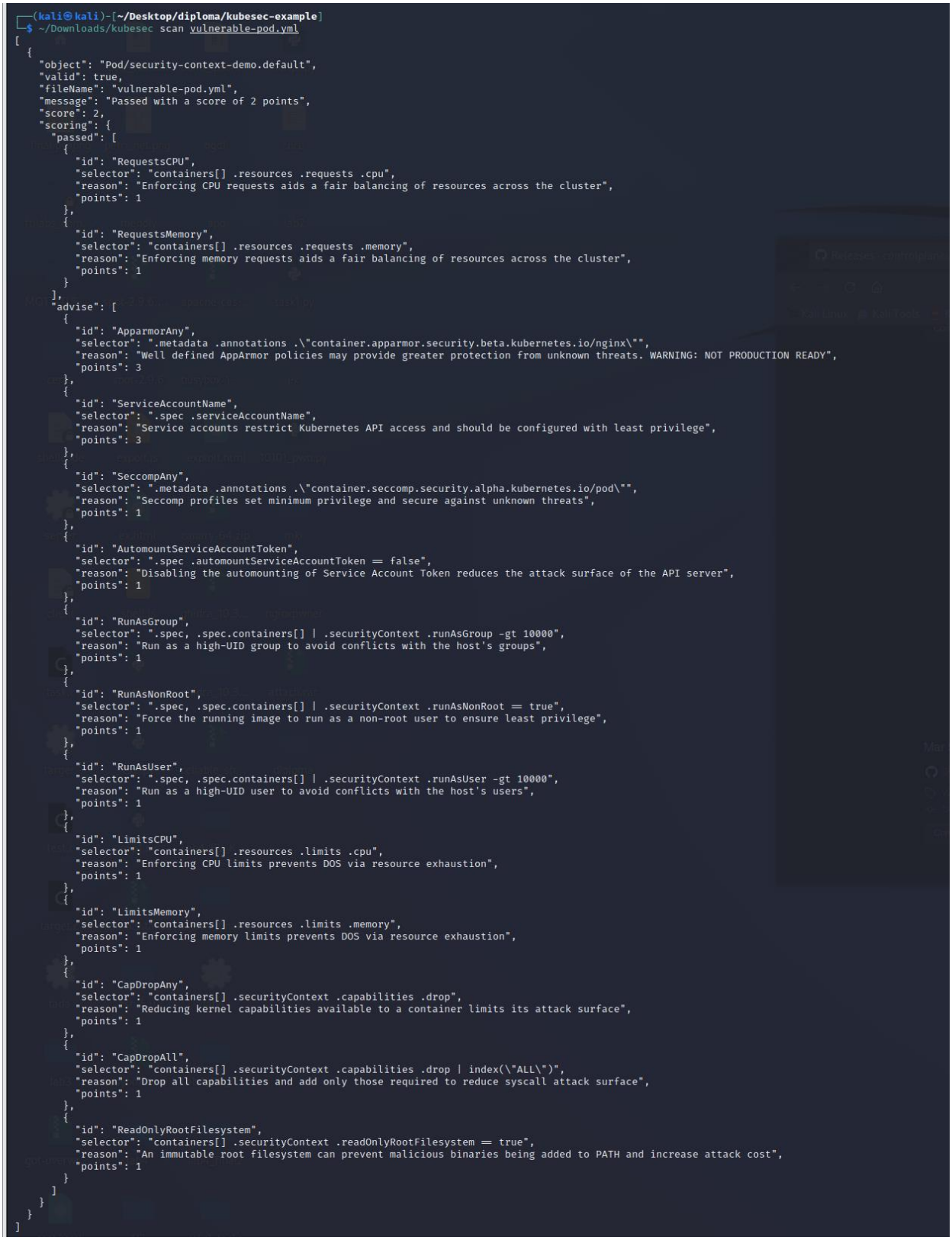
Як видно на рисунку 2.14 утиліта kube-audit надає більш детальну інформацію ніж kube-linter.

Останньою утилітой, яка сканує конфігураційні файли подів, яку варто розглянути є kube-sec.

Kubesec є інструментом для аналізу безпеки, який працює з ресурсами Kubernetes, використовуючи інтерфейс з відкритим вихідним кодом. Цей інструмент обробляє YAML маніфест Kubernetes, аналізуючи елементи, такі як модулі та

розгортання, в пошуках потенційних вразливостей. Зокрема, Kubesec зосереджується на виявленні слабких місць, наприклад, використання привілейованих налаштувань, запуск контейнерів без прав користувача та інших загальновідомих загроз. Для кожного знайденого ризику інструмент визначає ступінь серйозності, надаючи зведений рейтинг безпеки. [14]

Результат сканування:



```
┌──(kali㉿kali)-[~/Desktop/diploma/kubesec-example]
└─$ ~/Downloads/kubesec scan vulnerable-pod.yml
[
  {
    "object": "Pod/security-context-demo.default",
    "valid": true,
    "fileName": "vulnerable-pod.yml",
    "message": "Passed with a score of 2 points",
    "score": 2,
    "scoring": {
      "passed": [
        {
          "id": "RequestsCPU",
          "selector": "containers[] .resources .requests .cpu",
          "reason": "Enforcing CPU requests aids a fair balancing of resources across the cluster",
          "points": 1
        },
        {
          "id": "RequestsMemory",
          "selector": "containers[] .resources .requests .memory",
          "reason": "Enforcing memory requests aids a fair balancing of resources across the cluster",
          "points": 1
        }
      ],
      "advise": [
        {
          "id": "ApparmorAny",
          "selector": ".metadata .annotations .\"container.apparmor.security.beta.kubernetes.io/nginx\"",
          "reason": "Well defined AppArmor policies may provide greater protection from unknown threats. WARNING: NOT PRODUCTION READY",
          "points": 3
        },
        {
          "id": "ServiceAccountName",
          "selector": ".spec .serviceAccountName",
          "reason": "Service accounts restrict Kubernetes API access and should be configured with least privilege",
          "points": 3
        },
        {
          "id": "SeccompAny",
          "selector": ".metadata .annotations .\"container.seccomp.security.alpha.kubernetes.io/pod\"",
          "reason": "Seccomp profiles set minimum privilege and secure against unknown threats",
          "points": 1
        },
        {
          "id": "AutomountServiceAccountToken",
          "selector": ".spec .automountServiceAccountToken == false",
          "reason": "Disabling the automounting of Service Account Token reduces the attack surface of the API server",
          "points": 1
        },
        {
          "id": "RunAsGroup",
          "selector": ".spec, .spec.containers[] | .securityContext .runAsGroup -gt 10000",
          "reason": "Run as a high-UID group to avoid conflicts with the host's groups",
          "points": 1
        },
        {
          "id": "RunAsNonRoot",
          "selector": ".spec, .spec.containers[] | .securityContext .runAsNonRoot == true",
          "reason": "Force the running image to run as a non-root user to ensure least privilege",
          "points": 1
        },
        {
          "id": "RunAsUser",
          "selector": ".spec, .spec.containers[] | .securityContext .runAsUser -gt 10000",
          "reason": "Run as a high-UID user to avoid conflicts with the host's users",
          "points": 1
        },
        {
          "id": "LimitsCPU",
          "selector": "containers[] .resources .limits .cpu",
          "reason": "Enforcing CPU limits prevents DOS via resource exhaustion",
          "points": 1
        },
        {
          "id": "LimitsMemory",
          "selector": "containers[] .resources .limits .memory",
          "reason": "Enforcing memory limits prevents DOS via resource exhaustion",
          "points": 1
        },
        {
          "id": "CapDropAny",
          "selector": "containers[] .securityContext .capabilities .drop",
          "reason": "Reducing kernel capabilities available to a container limits its attack surface",
          "points": 1
        },
        {
          "id": "CapDropAll",
          "selector": "containers[] .securityContext .capabilities .drop | index(\"ALL\")",
          "reason": "Drop all capabilities and add only those required to reduce syscall attack surface",
          "points": 1
        },
        {
          "id": "ReadOnlyRootFilesystem",
          "selector": "containers[] .securityContext .readOnlyRootFilesystem == true",
          "reason": "An immutable root filesystem can prevent malicious binaries being added to PATH and increase attack cost",
          "points": 1
        }
      ]
    }
  }
]
```

Рисунок 2.15 – Результат сканування утилітою kubesec

Результат сканування представленний у вигляді JSON і має деталізований опис вразливостей поду з їх оцінкою.

Не менш потужною утилітою, яка пропонує більше 30 різних типів перевірок на вразливості є kube-score. Цей інструмент який виконує статичний аналіз коду визначень об’єктів кубернетес. Результатом виконання є список рекомендацій, що можна покращити, щоб зробити свою програму більш безпечною та стійкою.

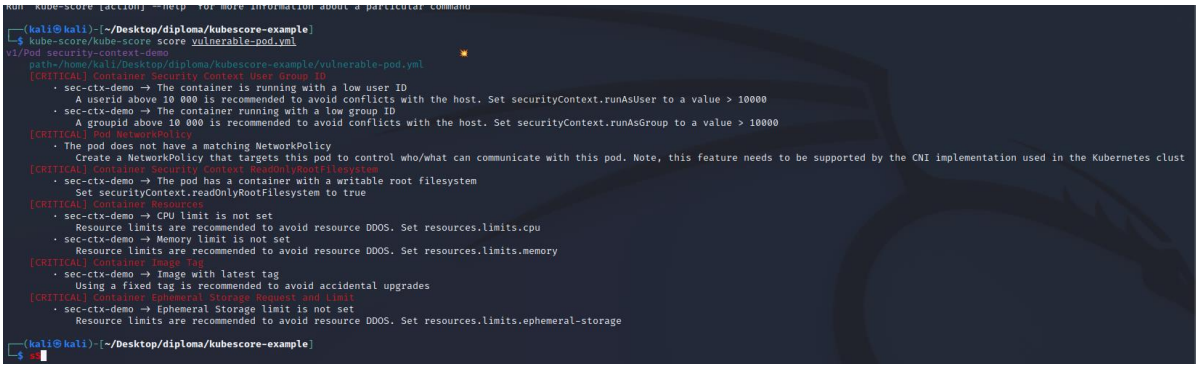


Рисунок 2.16 – Статичний аналіз вразливої конфігурації за допомогою kube-score [16]

Існує велика кількість різних засобів сканування вразливостей починаючи від статичного аналізу, закінчуючи динамічним, тому було розглянуто основні.

2.3 Порівняльний аналіз існуючих підходів до аналізу вразливостей Kubernetes

У цьому розділі варто порівняти ключові функції, сильні сторони та обмеження різних інструментів аналізу вразливостей Kubernetes, згаданих у попередньому розділі. Цей порівняльний аналіз допоможе зрозуміти, як кожен інструмент впливає на систему безпеки Kubernetes і де вони можуть бути недосконалими. Порівняльний аналіз наведений у таблиці 2.1

Таблиця 2.1 – Порівняльний аналіз засобів пошуку вразливостей в Kubernetes

Утиліта	Основна функціональність	Переваги	Недоліки	Застосування
Моніторинг				
Prometheus	Моніторинг і база даних часових рядів	Масштабованість, потужна мова запитів	Обмежене виявлення аномалій безпеки	Моніторинг продуктивності
Grafana	Візуалізація даних моніторингу	Надійні візуалізації, потужна екосистема плагінів	Покладається на зовнішні джерела даних (наприклад Prometheus)	Візуалізація даних у моніторингу
Falco	Моніторинг виявлення загроз	Механізм правил для аномальних дій	Вимоги до налаштування, крива навчання	Моніторинг поведінкової активності
Sysdig	Безпека і моніторинг Kubernetes	Глибока видимість, підтримка відповідності	Складне налаштування, накладні витрати на продуктивність	Комплексний моніторинг безпеки
Сканування				
KubeLinter	Статичний аналіз Kubernetes YAML	Перевірка неправильної конфігурації	Обмежується статичним аналізом	Аналіз конфігурації перед розгортанням
Kubeaudit	Аудит Kubernetes	Автоматизовані перевірки аудиту	Обмежений діапазон виявлення загроз	Відповідність передовим практикам Kubernetes
Kubesec	Аналіз ризиків безпеки	Оцінки ризику, інтеграція CI/CD	Аналіз лише конфігурації	Оцінка ризиків конфігурації
Kubescore	Статичний аналіз коду	Оцінка на основі найкращих практик	Фокус на статичнму аналізу	Оцінка конфігурації об’єкта Kubernetes

У цій таблиці наведено стислий огляд основних функцій кожного інструменту, ключових переваг, обмежень і сценаріїв, для яких вони найкраще підходять. Це

допоможе зрозуміти, як кожен інструмент по-різному впливає на безпеку Kubernetes і чому одного інструменту може бути недостатньо для комплексного аналізу безпеки. Порівняння підкреслює додатковий характер цих інструментів, висвітлюючи потенційні переваги їх інтеграції в єдине рішення безпеки.

Порівняльний аналіз різних інструментів аналізу вразливостей Kubernetes показує, що кожен інструмент має свої переваги та зосереджується на конкретних аспектах безпеки. Однак ця спеціалізація також виявляє значні обмеження, коли покладатись лише на один конкретний інструмент для комплексного покриття безпеки. Тому всі ці засоби можна поділити на такі обмеження:

- **Вузький фокус**: багато інструментів спеціалізується на аналізі конфігурації, моніторингу або вивяленні загроз. Така вузька спрямованість означає, що один інструмент не зможе надати повну картину безпеки середовищаж
- **Сліпі зони в охопленні безпеки:** кожен засіб має властиві «сліпі зони». Наприклад, інструмент статичного аналізу може пропустити загрози під час виконання, а інструмент моніторингу – усунути не належним чином вразливості конфігурації;
- **Складність керування одночасно кількома векторами загроз:** середовища k8s стикаються з різними загрозами, починаючи від неправильної конфігурації і до загроз під час виконання та мережевих атак;
- **Інтеграція:** навіть найбільш досконалі інструменти можуть не інтегруватись з іншими в екосистемі Kubernetes, що призводить до прогалин у безпеці;
- **Обмеження ресурсів:** використання одного інструменту може призвести до надмірного або недостатнього використання програмних

ресурсів. Деякі інструменти можуть потребувати більше ресурсів, тоді як інші – значно менше;

- **Ландшафт загроз:** динамічний і швидкозмінний характер загроз означає, що жоден інструмент не може постійно працювати з усіма вразливими місцями та векторами атак;
- **Проблема масштабованості:** у міру того, як кластери k8s збільшуються в розмірах і ускладнюються, одному інструменту може бути важко ефективино масштабуватись, що призводить до вузьких місць продуктивності та зниження ефективності у великих середовищах.

Підсумовуючи, хоча кожен інструмент аналізу вразливостей Kubernetes надає цінну інформацію та можливості, покладаючись на один інструмент, в результаті ви отримуєте фрагментоване уявлення про безпеку та потенційні вразливості. Це вимагає багатогранного підходу, який використовує сильні сторони різних інструментів для досягнення повного покриття безпеки в середовищах Kubernetes.

Висновки до розділу 2

У цьому розділі представлено детальний аналіз безпеки Kubernetes, зосереджено на виявленні ключових вразливостей, вивченні поточних підходів до моніторингу та виявлення загроз, а також порівняльному аналізі ефективності різних методологій безпеки.

Було розглянуто 10 найбільш популярних вразливостей згідно до OWASP TOP 10 Kubernetes Vulnerabilities. Було розглянуто проблеми неправильної конфігурації, важливість підтримки видимості в середовищах Kubernetes і ефективні методи керування вразливістю. Згодом було розглянуто існуючі методи, які

використовуються для моніторингу та виявлення загроз у Kubernetes, підкреслюються їхні можливості та обмеження.

Розділ завершується порівняльним дослідженням, у якому порівнюються різні підходи до аналізу вразливостей Kubernetes, що забезпечує цілісне уявлення про поточний ландшафт безпеки Kubernetes. Цей комплексний аналіз служить основою для розробки більш надійних стратегій безпеки в середовищах Kubernetes.

3 ПОБУДОВА СКАНЕРА ВРАЗИЛОВСТЕЙ ДЛЯ ПІДВИЩЕННЯ ЗАХИЩЕННОСТІ KUBERNETES

Враховуючи обмеження підходу з одним інструментом сканування вразливостей Kubernetes, стає зрозуміло, що багатогранний підхід є не лише корисним, але й необхідним. У цьому буде розглянуто і реалізовано, чому інтеграція кількох інструментів для роботи в тандемі пропонує більш надійне рішення для безпеки Kubernetes.

- **Комплексне покриття:** поєднуючи інструменти, які спеціалізуються на різних аспектах безпеки Kubernetes, таких як моніторинг під час виконання, статичний аналіз, виявлення загроз і відповідність, багатогранний підхід забезпечує більш повне покриття. Ця інтеграція допомагає закрити сліпі зони, які один інструмент може залишити без уваги;
- **Покращене виявлення загроз і реагування**: різні інструменти пропонують різні методи виявлення загроз і реагування. Інтеграція цих методів дозволяє створити більш складний механізм виявлення, використовуючи сильні сторони кожного інструменту для виявлення ширшого спектру загроз безпеці та реагування на них;
- **Покращене використання ресурсів:** багатогранний підхід може оптимізувати використання ресурсів шляхом використання конкретних можливостей кожного інструменту. Наприклад, один інструмент може бути ефективнішим у використанні ресурсів для моніторингу часу виконання, тоді як інший може краще підходити для статичного аналізу з мінімальним впливом на продуктивність;

- **Гнучкість і масштабованість**: інтеграція кількох інструментів забезпечує більшу гнучкість у адаптації до мінливих вимог і проблем масштабованості. У міру того, як середовище Kubernetes росте та розвивається, інфраструктуру безпеки можна коригувати, додаючи або змінюючи інструменти, не обмежуючись можливостями одного рішення;
- **Адаптованість до загроз, що розвиваються**: ландшафт кібербезпеки постійно розвивається. Багатогранний підхід із здатністю інтегрувати нові інструменти та методи має кращі позиції для адаптації до нових загроз і вразливостей у міру їх появи;
- **Покращена відповідність і звітність**: різні інструменти пропонують різні функції відповідності та звітності. Поєднуючи ці інструменти, організації можуть досягти більш надійної відповідності, більш ефективно відповідаючи різноманітним нормативним вимогам;
- **Надмірність і надійність**: використання кількох інструментів забезпечує резервування, гарантуючи, що якщо один інструмент виходить з ладу або пропускає загрозу, інші можуть компенсувати це, тим самим підвищуючи загальну надійність системи безпеки;
- **Цілісна безпека**: багатогранний підхід сприяє цілісному уявленню про безпеку, заохочуючи до більш стратегічної та проактивної позиції. Він підкреслює важливість вирішення проблем безпеки на різних рівнях і етапах середовища Kubernetes, від розробки до розгортання та виконання.

Підсумовуючи, багатогранний підхід узгоджується зі складною та динамічною природою середовищ Kubernetes. Він забезпечує більш ефективну та комплексну стратегію для аналізу вразливостей і посилює загальну безпеку. У наступному розділі буде обговорено стратегію інтеграції для об’єднання цих інструментів у єдине рішення безпеки.

3.1 Архітектура сканера

Запропонований сканер розроблено як комплексний і універсальний інструмент для аналізу вразливостей Kubernetes, який поєднує в собі сильні сторони добре відомих статичних сканерів уразливостей в єдину програму. Структуру та функціональні можливості сканера можна описати наступним чином:

Основні компоненти

Unified Application Framework: основна програма, написана на Python, діє як центральний оркестратор, який визначає, який конкретний сканер використовувати на основі характеру вразливості або вимог аналізу. Програма має модульну конструкцію для легкої інтеграції кількох статичних сканерів уразливостей, що забезпечує гнучкість і масштабованість.

Інтеграція з ChatGPT для порад на основі ШІ: додаток міститиме додаткову функцію для звернення до OpenAI's ChatGPT для отримання експертних порад щодо усунення конкретних вразливостей. Ця інтеграція штучного інтелекту має на меті надати додаткову інформацію та рекомендації, покращуючи процес прийняття рішень щодо керування вразливістю.

Інтеграція сканера: сканери, вибрані для інтеграції, будуть добре відомими інструментами, визнаними за їхню ефективність у виявленні вразливостей у середовищах Kubernetes. Кожен сканер буде модулем у програмі, який можна запускати незалежно або в поєднанні з іншими.

Розгортання та експлуатація

Докеризація: весь додаток сканера разом із інтегрованими компонентами буде контейнеризовано за допомогою Docker. Такий підхід забезпечує легкість розгортання, узгодженість у різних середовищах та ізоляцію від хост-системи.

Dockerfile і створення зображень: Dockerfile буде створено для створення комплексного образу Docker для сканера. Він включатиме кроки для встановлення необхідних утиліт і залежностей, необхідних для роботи інтегрованих сканерів і основної програми. Dockerfile також виконуватиме клонування сканерів із відповідних репозиторіїв GitHub, забезпечуючи використання останніх версій цих інструментів.

Оперативний робочий процес

Робочий процес сканера буде простим, але гнучким, що дозволить користувачам вибирати конкретні сканери та/або використовувати функцію рекомендацій за допомогою ШІ. Після сканування програма збирає та представляє результати разом із будь-якими порадами, створеними ШІ (якщо використовується інтеграція OpenAI).

Переваги:

- Комплексний аналіз: об’єднуючи кілька сканерів, програма забезпечує більш ретельний аналіз, ніж будь-який окремий інструмент;
- Можливість налаштування та масштабування: модульний характер дозволяє легко оновлювати та додавати нові сканери чи функції;
- Прийняття рішень на основі штучного інтелекту: інтеграція з ChatGPT забезпечує додатковий рівень інтелекту, пропонуючи пропозиції та інформацію щодо усунення вразливостей;
- Простота розгортання та використання: Докеризація спрощує розгортання в різних середовищах і підвищує портативність.

Ця запропонована структура для сканера вразливостей Kubernetes представляє інноваційний підхід, який використовує потужність багатьох інструментів і штучного інтелекту, щоб забезпечити розширене, зручне рішення для аналізу безпеки в середовищах Kubernetes.

Діаграма послідовностей буде виглядати наступним чином:

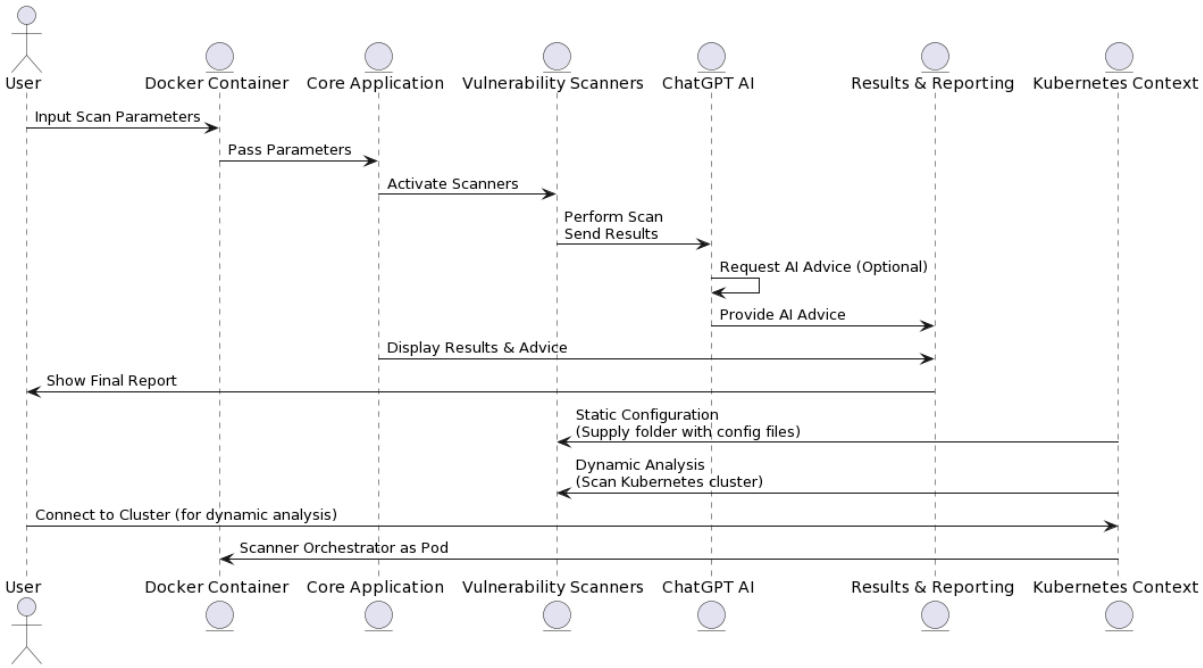


Рисунок 3.1 – Діаграма послідовностей запропонованого сканеру

Діаграма розгортання:

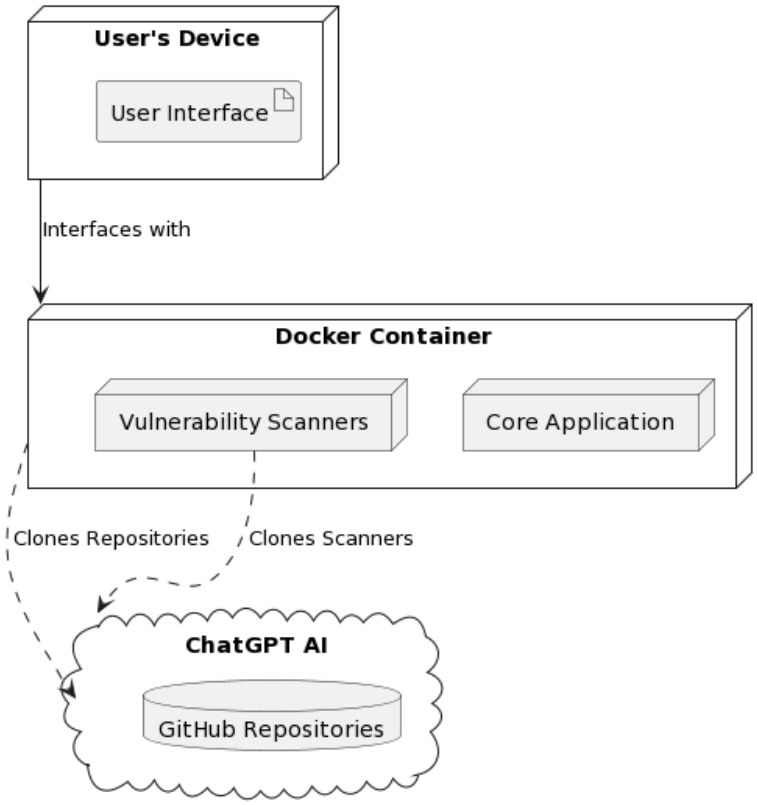


Рисунок 3.2 – Діаграма розгортання

І діаграма активностей:

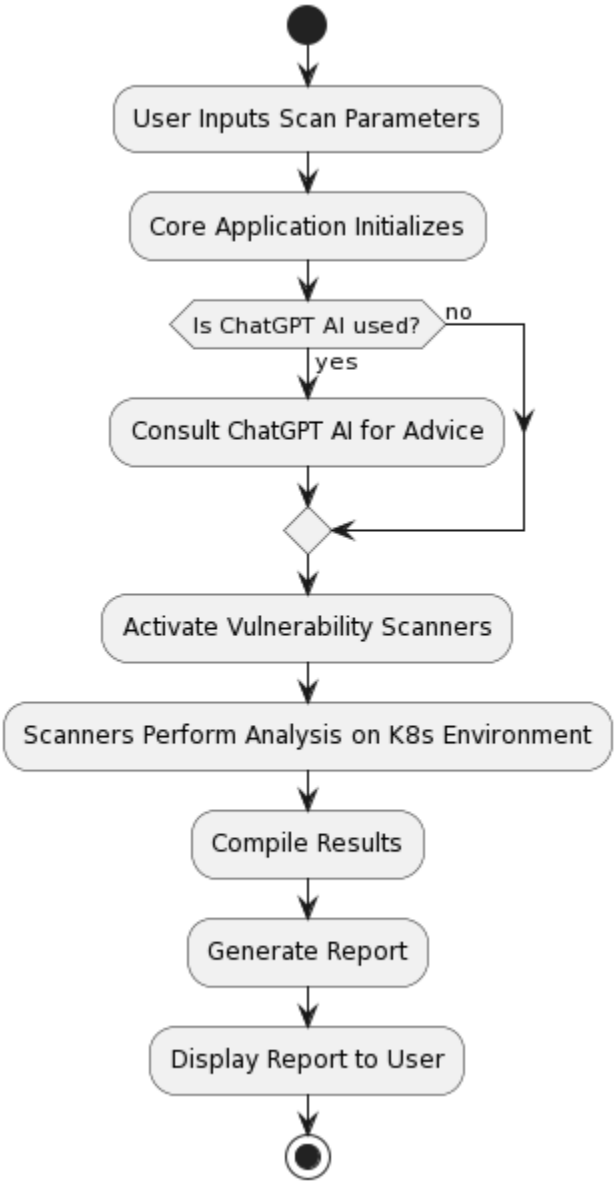


Рисунок 3.3 – Діаграма активностей

Архітектура сканеру, який розгорнутий в Kubernetes pod:

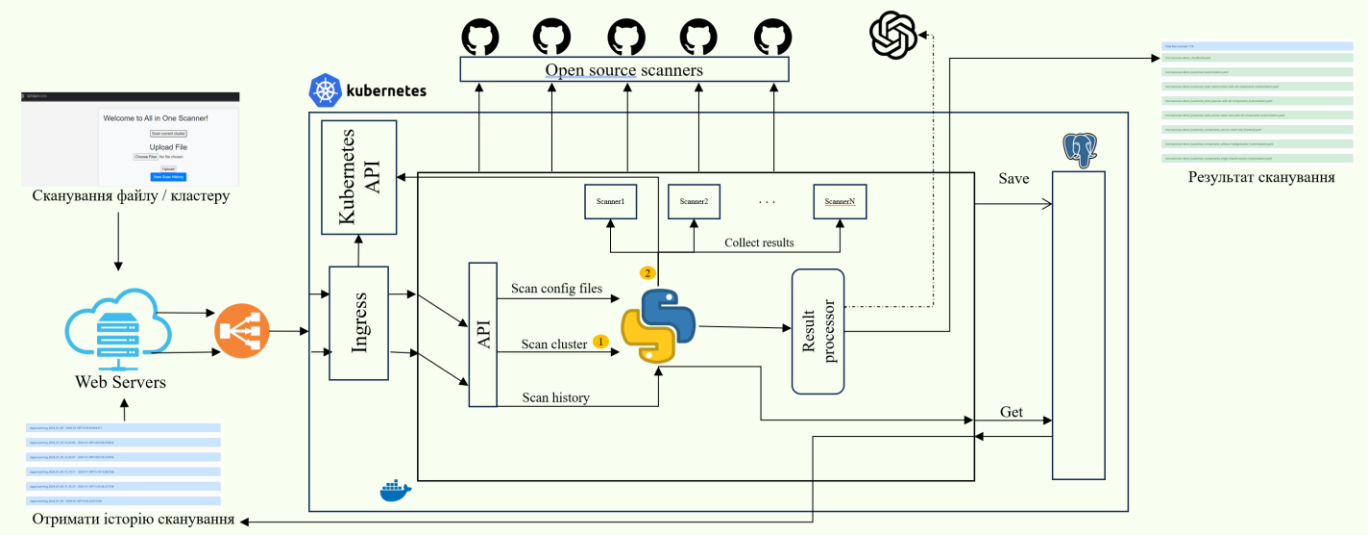


Рисунок 3.4 – Архітектура сканеру, розгорнутого в Kubernetes поді

Діаграма взаємодій між модулями утиліти виглядає наступним чином:

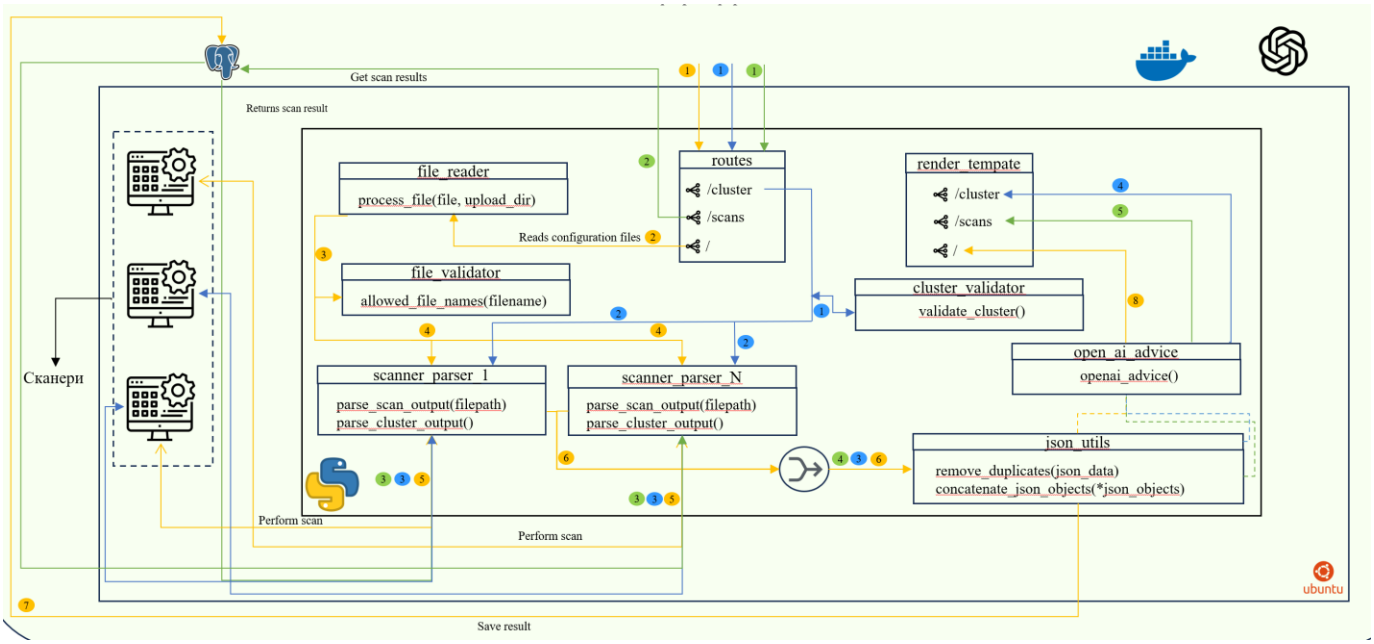


Рисунок 3.5 – Діаграма взаємодії між модулями утиліти

3.2 Програмна реалізація статичного сканера вразливостей

Необхідною умовою для використання розробленого сканера вразливостей є наявність заздалегідь встановленого докеру [17].

Для запуску сканеру треба ввести команду docker-compose up –build в терміналі поточна директорія якого знаходиться у теці зі сканером.

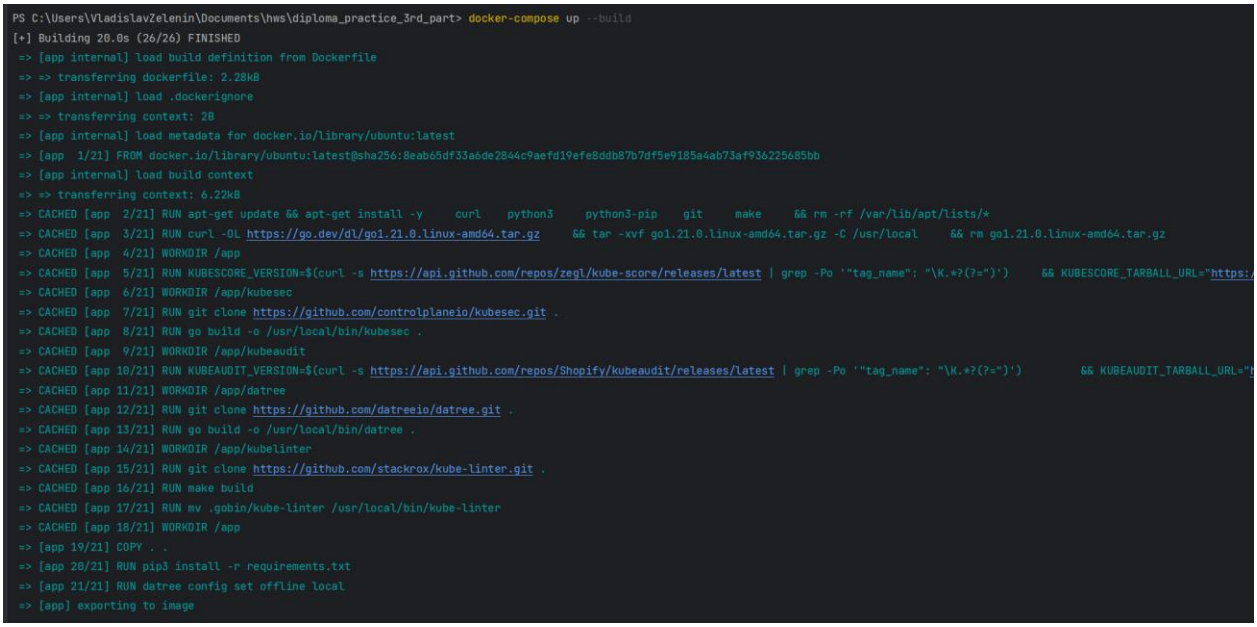


Рисунок 3.6 – Результат виконання команди, яка запускає сканер у докер контейнері

Docker-compose файл використовує Dockerfile для побудови контейнера. У цьому файлі міститься вся конфігурація, потрібна для роботи сканера. Докер контейнер запускається на базі ubuntu і встановлює необхідні пакети і утиліти для подальшого завантаження і встановлення сканерів.

```
FROM ubuntu:latest

RUN apt-get update && apt-get install -y \
    curl\
    python3 \
    python3-pip\
    git \
    make \
    && rm -rf /var/lib/apt/lists/*

RUN curl -OL https://go.dev/dl/go1.21.0.linux-amd64.tar.gz \
    && tar -xvf go1.21.0.linux-amd64.tar.gz -C /usr/local \
    && rm go1.21.0.linux-amd64.tar.gz

ENV PATH="/usr/local/go/bin:${PATH}"
# Set the working directory in the container
WORKDIR /app
```

Рисунок 3.7 – Встановлення необхідних пакетів

Тому для того, щоб об’єднати будь-яку кількість сканерів в одну єдину утиліту достатньо мати встановленими python, go, make. На рисунку 3.3 саме це і відбувається.

Далі відбувається клонування з git репозиторіїв сканерів, все обмежується лише місцем на диску та кількістю виділених ресурсів. Завдяки тому, що клонується репозиторій або останній реліз маємо актуальні версії.

Приклад встановлення сканерів зазначений на рисунку 3.8:

```
# Clone & install kube-score
RUN KUBESCORE_VERSION=$(curl -s https://api.github.com/repos/zegl/kube-score/releases/latest | grep -Po '"tag_name": "\K.*?(?=")') \
    && KUBESCORE_TARBALL_URL="https://github.com/zegl/kube-score/releases/download/${KUBESCORE_VERSION}/kube-score_${KUBESCORE_VERSION#v}_linux_amd64.tar.gz" \
    && curl -L -o kube-score.tar.gz "$KUBESCORE_TARBALL_URL" \
    && tar -xvzf kube-score.tar.gz -C /usr/local/bin/ \
    && chmod +x /usr/local/bin/kube-score

# Clone & install kubesec
WORKDIR /app/kubesec
RUN git clone https://github.com/controlplaneio/kubesec.git .
RUN go build -o /usr/local/bin/kubesec .

# Clone & install kubeaudit
WORKDIR /app/kubeaudit
RUN KUBEAUDIT_VERSION=$(curl -s https://api.github.com/repos/Shopify/kubeaudit/releases/latest | grep -Po '"tag_name": "\K.*?(?=")') \
        && KUBEAUDIT_TARBALL_URL="https://github.com/Shopify/kubeaudit/releases/download/$KUBEAUDIT_VERSION/kubeaudit_${KUBEAUDIT_VERSION#v}_linux_amd64.tar.gz" \
        && curl -L -o kubeaudit.tar.gz "$KUBEAUDIT_TARBALL_URL" \
        && tar -xvzf kubeaudit.tar.gz -C /usr/local/bin/ \
        && chmod +x /usr/local/bin/kubeaudit

# Clone & install datree
WORKDIR /app/datree
RUN git clone https://github.com/datreeio/datree.git .
RUN go build -o /usr/local/bin/datree .

# Clone & install kube-linter
WORKDIR /app/kubelinter
RUN git clone https://github.com/stackrox/kube-linter.git .
RUN make build
RUN mv .gobin/kube-linter /usr/local/bin/kube-linter

WORKDIR /app

COPY . .
```

Рисунок 3.8 – Встановлення сканерів з відкритим вихідним кодом

Далі залишається запустити лише python застосунок який саме і є головним оркестратором. Опціонально треба додати в Dockerfile поле ENV OPENAI_API_KEY="sk-", де в лапках буде знаходитись ключ до OpenAI API.

Після запуску застосунку маємо таку домашню сторінку, яка знаходиться на 127.0.0.1:8080:

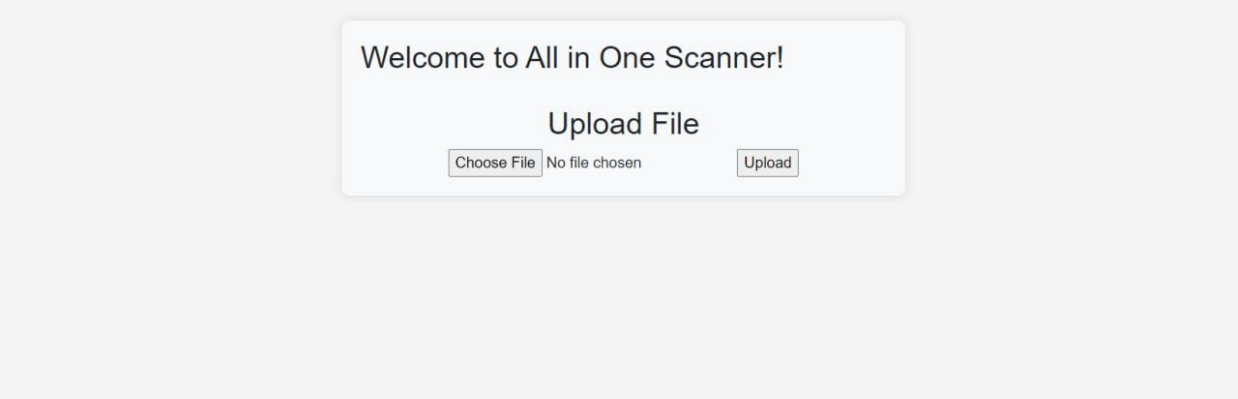


Рисунок 3.9 – Домашня сторінка сканеру

Для завантаження конфігураційного файлу треба натиснути на Choose_file, обрати файл з розширенням .yaml або .yml або обрати директорію для сканування та натиснути Upload.

Далі після натискання кнопки Upload починається сам процес аналізу конфігураційних файлів, їх сканування N-ою кількістю сканерів та об’єднання всіх репортів в один єдиний та виключення з них дублікатів. Якщо ж є OpenAI API Key буде доданий додатковий репорт, який містить в собі детальну структуровану інформацію про вразливу конфігурацію в одному місці і виправлену конфігурацію.

Після сканування всіх файлів відбувається автоматичне перенаправлення на іншу сторінку з результатами.

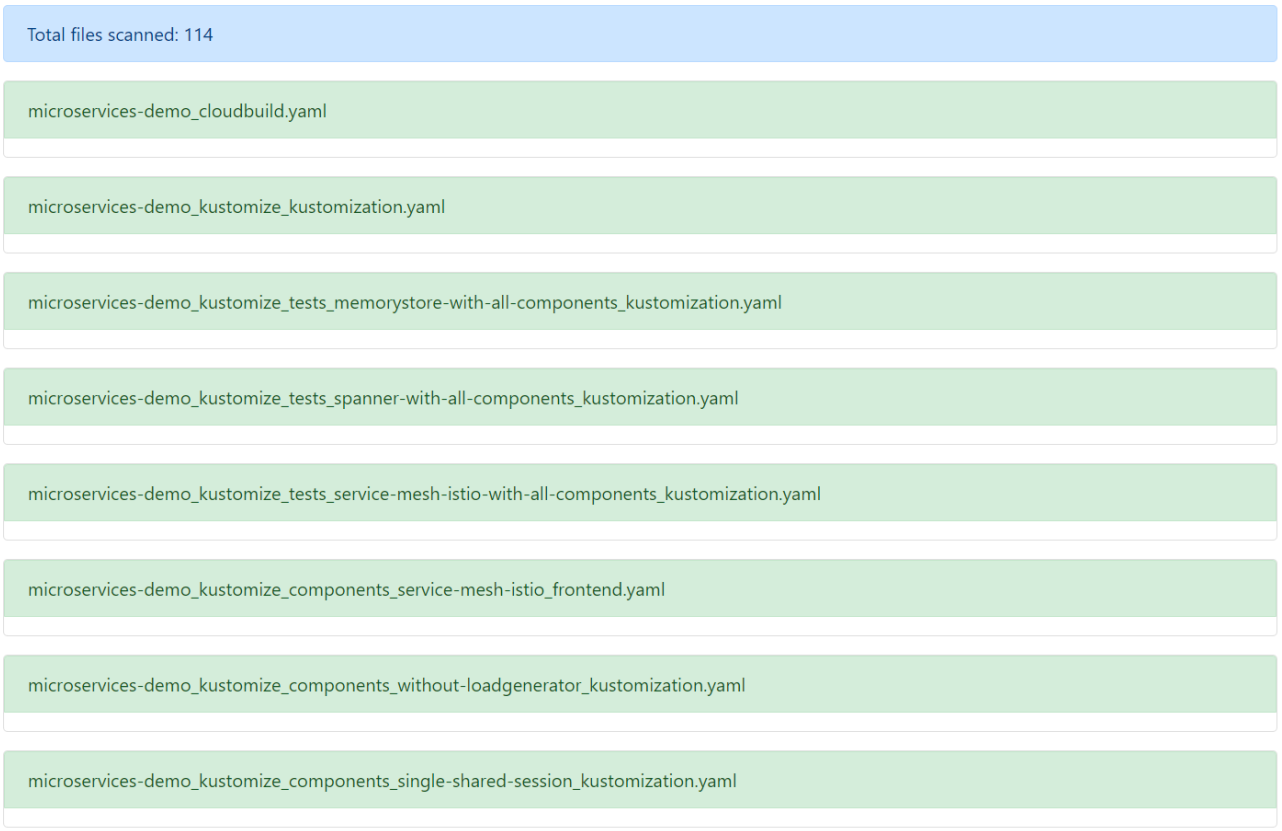


Рисунок 3.10 – Сторінка з результатами сканування microservices demo [21]

Блоки які виділені зеленим кольором – не клікабельні і означають, що всі перевірки пройдені успішно. Якщо блок червоного кольору – значить є проблеми з конфігурацією:

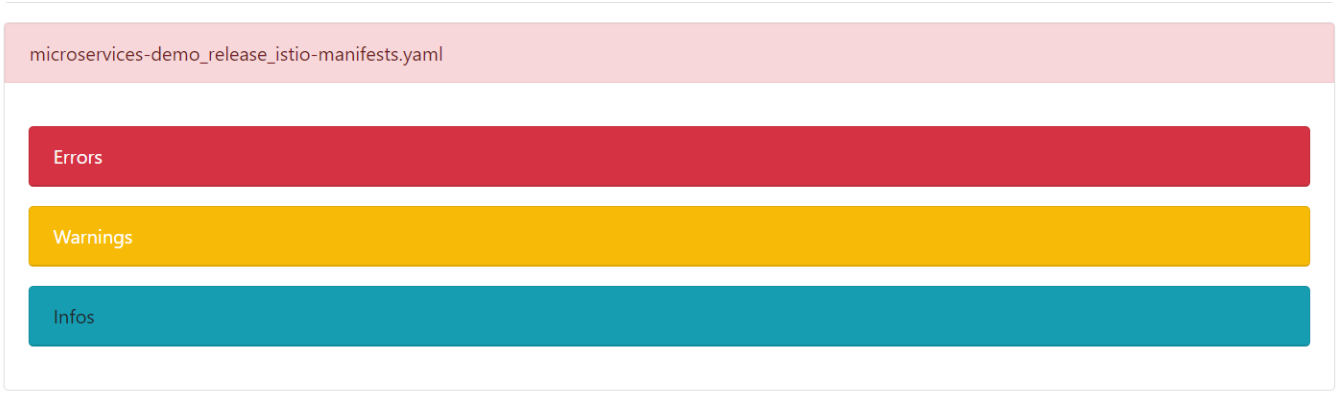


Рисунок 3.11 – Проблемна конфігурація

Кожна з кнопок Errors, Warnings, Info клікабельна та відображає в собі всю інформацію поділену на рівні помилок.

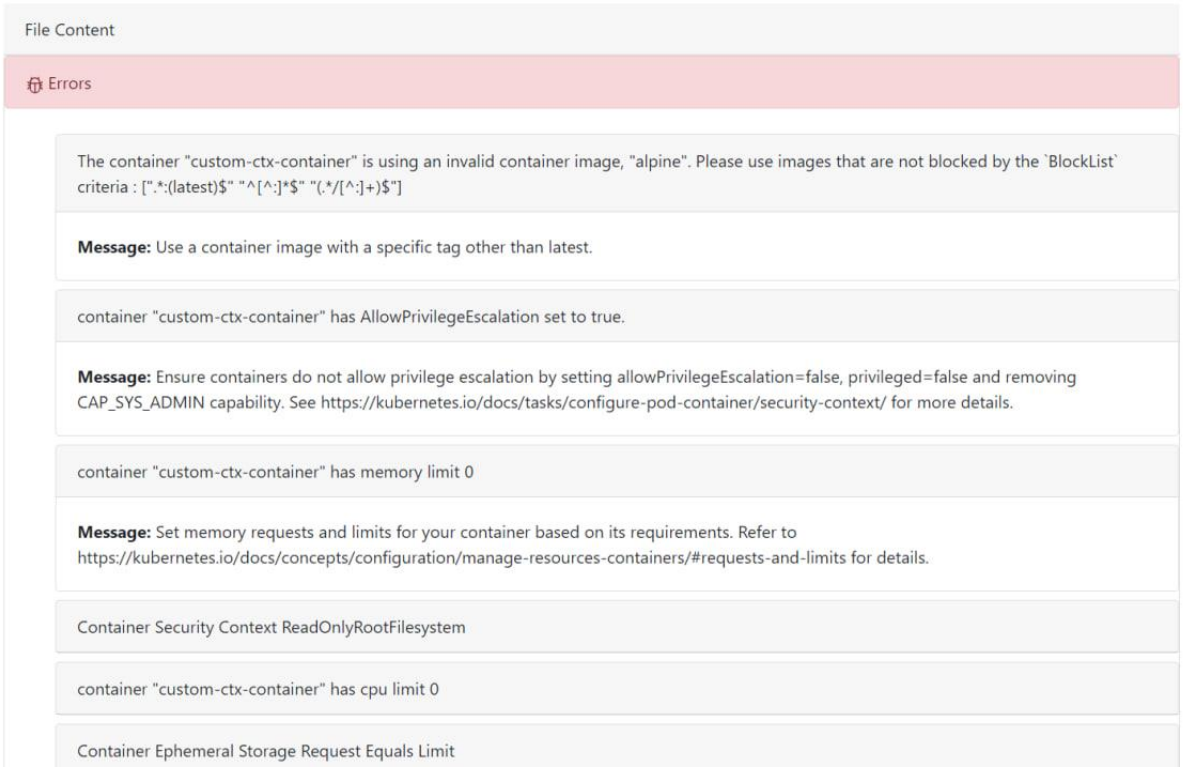


Рисунок 3.12 – Репорт з помилками рівня “Error”

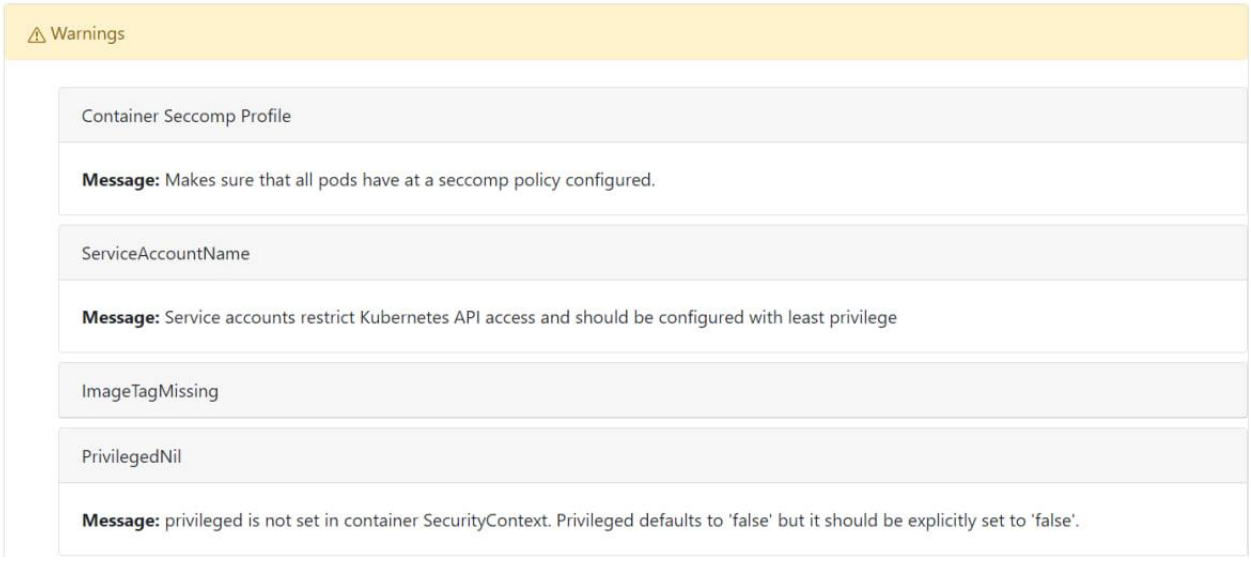


Рисунок 3.13 – Репорт з помилками рівня “Warn”

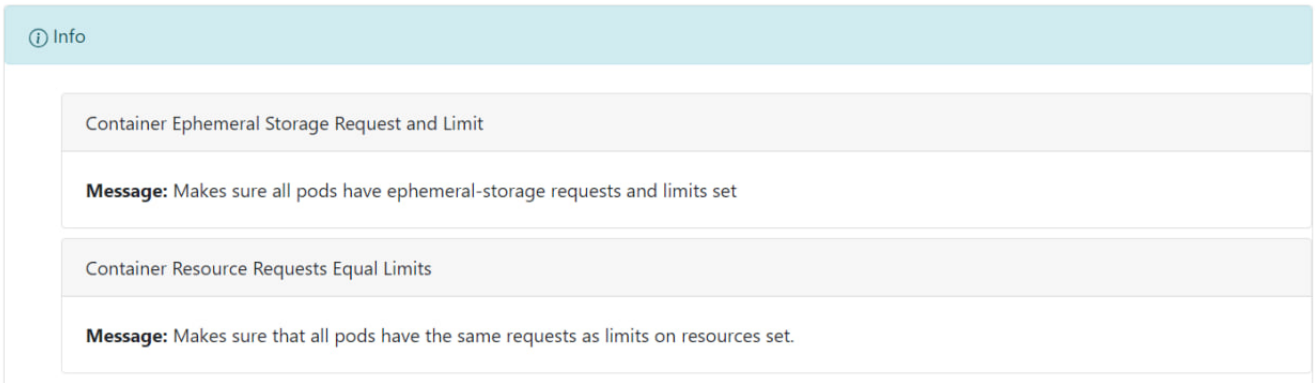


Рисунок 3.14 – Репорт з рівнем “Info”

В результаті тестування було практично доведено, що використання декількох або великої кількості сканерів об’єднаних в один значно пришвидшує тестування конфігурації і зменшує вірогідність пропускання сканером критичної помилки. Завдяки тому, що всі сканери встановлюються з git репозиторія при кожному перезапуску контейнера (мається на увазі його перестворення) маємо останні версії усіх утиліт, які використовуються. Використання Open AI API не є безкоштовним, але завдяки оптимізації запиту маємо наступну ціну кожного: 0.035$, що згідно курсу НБУ від 11.12.2023 в гривні дорівнює ~1,28 (одна гривня двадцять вісім копійок)

Із недоліків можу виділити повільний запит до OpenAI API, який займає до 6 секунд поки ШІ генерує відповідь, тому загальний час на виконання сканування з 5 сканерами і увімкненим OpenAI займає ~7.5 секунди, у випадку з вимкненим ~2 секунди.

Застосунок який є оркестратором сканерів після завантаження в нього файлу викликає системні команди для збереження результатів виконання сканування конкретної утиліти у текстовий файл. Приклад команди:

```
os.system(f'kubeaudit all -f "{filepath}" --no-color -p json > kubeaudit_output.txt')
```

Оскільки, за замовчуванням, контейнер запускається з правами root застосунок має можливість використовувати системні команди.

Після збереження результатів всіх сканувань починається обробка отриманих результатів, а точніше приведення їх до потрібного JSON формату:

```
{
  "vulnerability": "VulnerabilityName",
  "level": "error",
  "message": "Message with vulnerability info"
}
```

Наступним чином:

```
def parse_kubelinter_output(file_path):
    try:
        with open(file_path, 'r') as file:
            data = json.load(file)  # Load JSON content

        parsed_data = []

        for report in data.get("Reports", []):
            vulnerability = report.get("Diagnostic", {}).get("Message", "")
            remediation = report.get("Remediation", "")

            transformed_item = {
                "vulnerability": vulnerability,
                "level": "error",  # Default level is error
                "message": remediation
            }

            parsed_data.append(transformed_item)

        # Convert the parsed data to JSON
        result_json = json.dumps(parsed_data, indent=4)

        return result_json
    except:
        return json.dumps([], indent=4)
```

Рисунок 3.15 – Приведення отриманих результатів сканування до потрібного формату

Кожен сканер має свій власний формат, тому для отримання потрібного нам формату треба з отриманого результату витягувати потрібні нам дані програмно.

Тому для того щоб додати новий сканер потрібно:

1. Додати його клонування і встановлення в Dockerfile;
2. Командою сканера записати результат сканування в окремий файл;
3. Реалізувати сканер результату та привести до необхідного формату.

З цього можна зробити висновок, що розширення ПЗ не є складним і застосунок легко піддається масштабованості.

Для розгортання сканеру в Kubernetes потрібно застосувати наступну конфігурацію:

```
apiVersion: v1
kind: Pod
metadata:
  name: app
spec:
  containers:
    - name: app
      image: <your-image-name>
      ports:
        - containerPort: 8080
      volumeMounts:
        - name: app-volume
          mountPath: /app
  volumes:
    - name: app-volume
      hostPath:
        path: /path/on/host
```

Де image – зображення Docker яке знаходиться в Docker registry.

3.3 Програмна реалізація динамічного сканера вразливостей

Для реалізації динамічного сканера вразливостей перш за все треба оновити UI, в якому додано кнопку «Scan current cluster», яка сканує поточний кластер. Оновлений зовнішній вигляд інтерфейсу виглядає наступним чином:

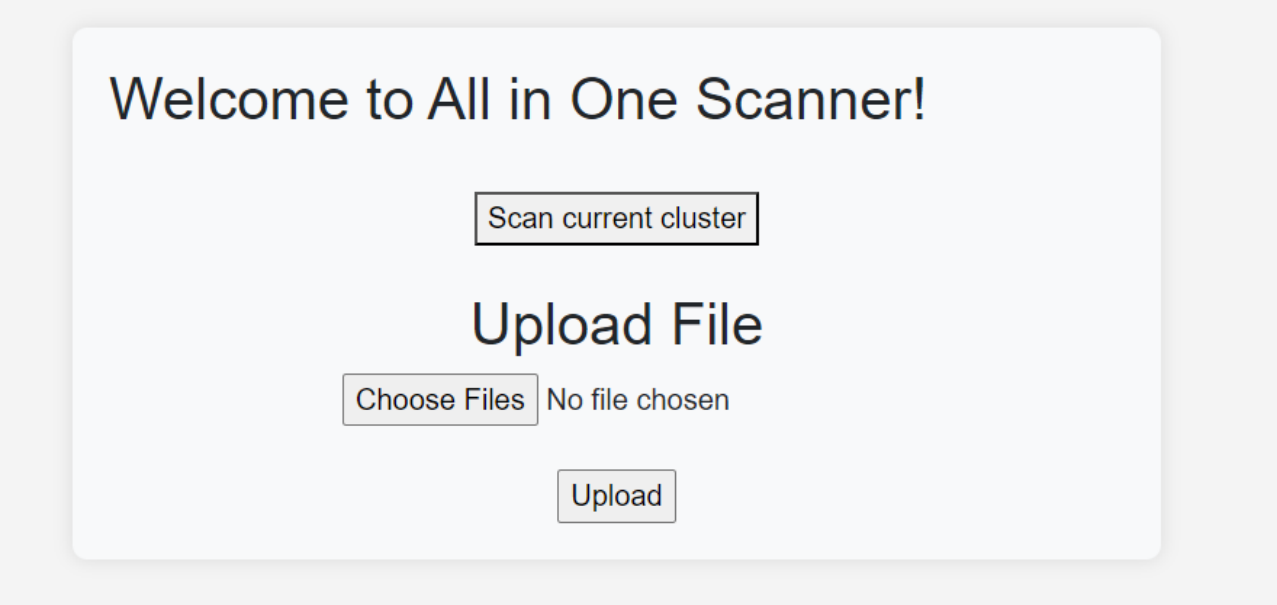


Рисунок 3.16 – Оновлений вигляд інтерфейсу

У випадку, якщо в кластері немає подів, або кластер не запущений на інтерфейсі буде виведено помилку:

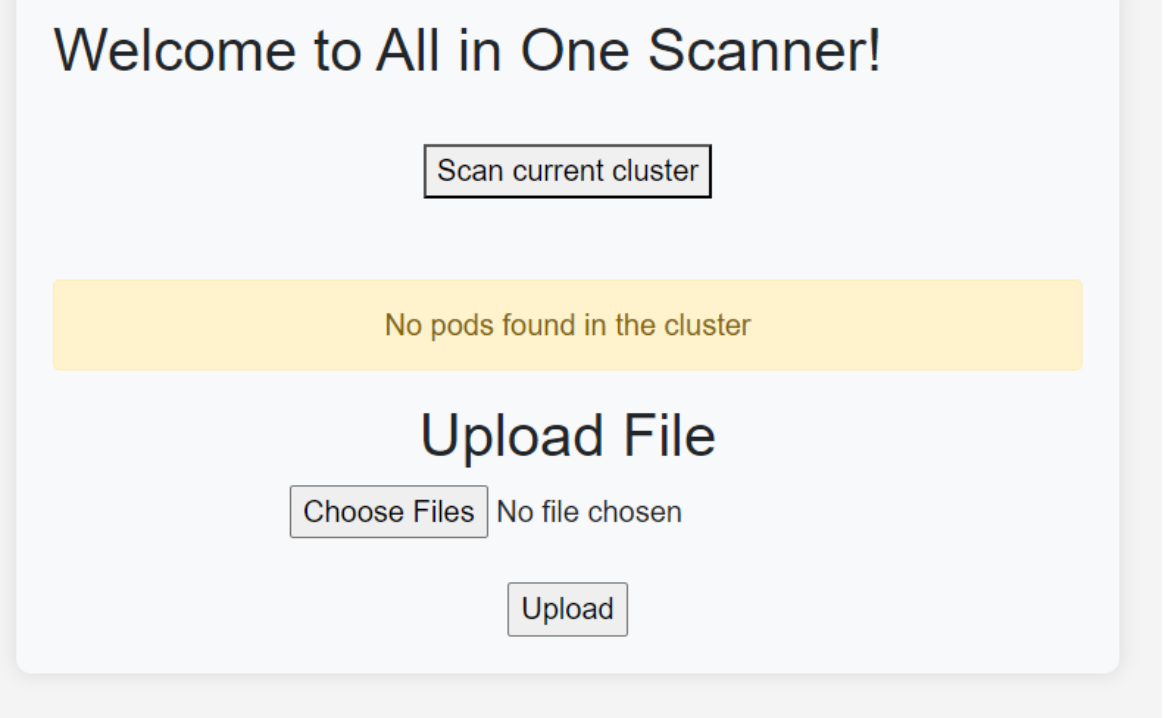


Рисунок 3.17 – Помилка у випадку вимкненого кластеру або відсутності подів в ньому

Якщо є хоча б один под то сканування успішно запуститься по аналогії з тим, як це працює у випадку зі статичним сканером, тому отримуємо схожий результат:

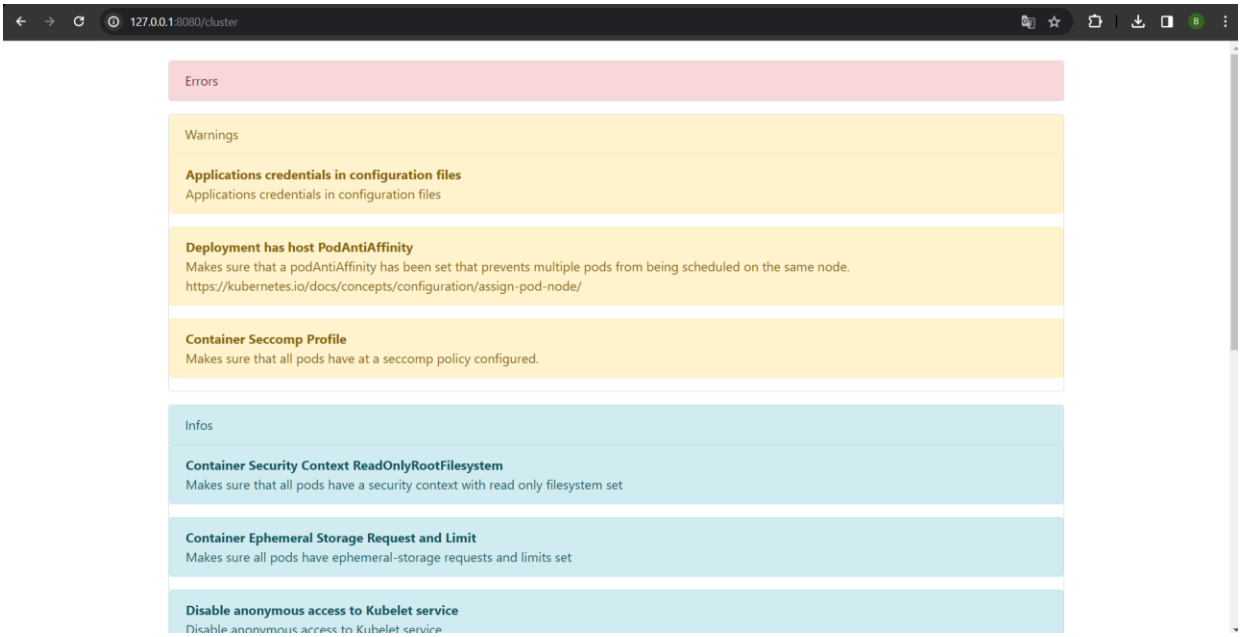


Рисунок 3.18 – Результат сканування кластеру

Більшість сканерів, які були розглянуті у розділі зі статичним аналізом використовуються і у випадку з динамічним, тому що вони здатні сканувати і конфігураційні файлі, і працювати з безпосередньо кластером. Завдяки тому, що програмна реалізація розділу 3.2 від самого початку було розроблена уніфікованою для різних сканерів та здатною легко масштабуватись – додавання нових типів утиліт є простою задачею та потребує лише такі кроки:

1. Додати команди для встановлення утиліти в Dockerfile;
2. Створити новий метод який буде виконувати команду сканування конфігурації/кластера;
3. Додати парсер результату в JSON об’єкт.

Отож маємо такий набір утиліт в одному сканері: kube-score [16], kubesec [14], kubeaudit [13], datree [18], kube-linter [19], kubescape [20], kube-bench [15].

3.4 Порівняння отриманих результатів

Для підтвердження необхідності використання єдиного оркестратора сканерів проведемо декілька тестів. Випробування проводилися в трьох різних умовах для імітації різних робочих середовищ:

1. Більшість сканерів були застарілими, а інші були оновлені до останньої версії;
2. Усі сканери оновлено до останньої версії;
3. Усі сканери були застарілих версій.

Кожен тест виконувався на ідентичному наборі файлів із microservices-demo [21], що забезпечувало узгодженість у тестах.

Критерії успішного сканування включали точність виявлення вразливостей і мінімізацію помилкових спрацьовувань. Налаштування передбачало підготовку контрольованого середовища, де аналізувалися як статичні файли конфігурації, так і динамічно кластер Kubernetes.

Випробування дали такі результати:

1. З поєднанням застарілих і оновлених сканерів: 84 успішних сканування з 30 помилками;
2. З усіма оновленими сканерами: 70 успішних сканувань із 44 помилками;
3. З усіма застарілими сканерами: 98 успішних сканувань із 16 помилками;

Порівняльний аналіз підкреслює переваги використання організованого підходу порівняно з окремими сканерами окремо. Зведені результати оркестратора більш повні, що зменшує кількість уразливостей, які може пропустити один сканер.

Наслідки цих результатів для практики безпеки Kubernetes є значними. Здатність оркестратора керувати набором різноманітних інструментів сканування та

використовувати їхні спільні переваги підвищує рівень безпеки середовищ Kubernetes.

Щоб оглянути більш детальні результати тестування та повний аналіз, див. Додаток А, який містить вичерпні дані до результатів.

Висновки до розділу 3

У цьому розділі був розроблений оркестратор для динамічного та/або статичного сканування вразливостей в Kubernetes. Було детально розглянуто архітектуру сканера та намальовані діаграми взаємодій, послідовностей для розуміння, як саме працює розроблена утиліта. Було надано рекомендації стосовно розширення існуючого сканера і проведено декілька тестів, які практично доводять необхідність у впровадженні єдиного рішення, яке об’єднує різні утиліти. Тести було проведено на наборі із 7 сканерів, у випадку, якщо їх буде більше то результати можуть відрізнятися у кращу сторону.

ВИСНОВКИ

Комплексне дослідження, проведене в цій роботі, значно сприяє розумінню та підвищенню безпеки в середовищах Kubernetes. Основною метою дослідження була розробка утиліти, зокрема оркестратора сканера вразливостей для Kubernetes, щоб надавати своєчасну та актуальну інформацію про стан безпеки платформи. Завдяки ретельному аналізу дослідження успішно підкреслило життєво важливу потребу в такій утиліті в сучасному середовищі кіберзагроз, що швидко розвивається.

У цій роботі було ретельно оцінено поточний стан безпеки в Kubernetes, заглиблено в уразливості та проблеми, з якими стикається платформа. Дослідження продемонструвало ефективність запропонованої утиліти для сканування конфігураційних файлів і кластерів Kubernetes, пропонуючи більш надійний і проактивний підхід до виявлення та вирішення проблем безпеки. Така проактивна позиція має важливе значення для зменшення ризиків і підвищення загальної безпеки середовищ Kubernetes.

Крім того, дослідження підтвердило як теоретично, так і практично необхідність впровадження оркестратора сканера вразливостей з відкритим кодом. Інтегрувавши цю утиліту в екосистему Kubernetes, організації можуть досягти вищого рівня гарантії безпеки, йдучи в ногу з постійним прогресом технологій і кіберзагрозами.

Важливість цієї роботи виходить за межі технічної сфери, оскільки вона надає цінну інформацію для адміністраторів і розробників, які відповідають за підтримку та захист платформ Kubernetes. Висновки та рекомендації, викладені в цій дисертації, є не тільки своєчасними, але й критично важливими для забезпечення безпечної роботи Kubernetes у різних секторах.

Підсумовуючи, дослідження, втілене в цій дисертації, є значним кроком вперед у захисті Kubernetes від потенційних уразливостей. Це підкреслює важливість

постійної пильності та інновацій у практиці кібербезпеки, особливо в складних і широко використовуваних системах, таких як Kubernetes. Очікується, що результати цього дослідження прокладуть шлях для майбутніх досягнень у цій галузі, сприяючи розвитку більш стійких і безпечних інфраструктур інформаційних технологій.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Kubernetes Overview [Електронний ресурс] – Режим доступу до ресурсу: https://kubernetes.io/docs/concepts/overview/#why-you-need-kubernetes-and-what-can-it-do
2. Kubernetes Components Overview [Електронний ресурс] – Режим доступу до ресурсу: https://kubernetes.io/docs/concepts/overview/components/#control-plane-components
3. Kubernetes Node Overview [Електронний ресурс] – Режим доступу до ресурсу: https://kubernetes.io/docs/concepts/architecture/nodes/
4. Containers [Електронний ресурс] – Режим доступу до ресурсу: https://kubernetes.io/docs/concepts/containers/
5. A visual guide to Kubernetes networking fundamentals By Nived Velayudhan [Електронний ресурс] – 01.06.2022 – Режим доступу до ресурсу: https://opensource.com/article/22/6/kubernetes-networking-fundamentals
6. Kubernetes Cluster Security, Component by Component [Електронний ресурс] – Режим доступу до ресурсу: https://sysdig.com/learn-cloud-native/kubernetes-security/cluster-security/
7. 6 Ways to Improve Kubernetes Network Security [Електронний ресурс] – TIGERA Guiders – Режим доступу до ресурсу: https://www.tigera.io/learn/guides/kubernetes-security/kubernetes-network-security/
8. 6 Ways to Improve Kubernetes Network Security [Електронний ресурс] – TIGERA Guiders – Режим доступу до ресурсу: https://www.tigera.io/learn/guides/container-security-best-practices/
9. The Current State of Kubernetes Threat Modelling by Marco Lancic [Електронний ресурс] – 30.06.2020 – Режим доступу до ресурсу - https://blog.marcolancini.it/2020/blog-kubernetes-threat-modelling/
10. OWASP Kubernetes Top 10 A Risk Assessment BY NIGEL DOUGLAS [Електронний ресурс] – 21.02.2023 – Режим доступу до ресурсу: https://sysdig.com/blog/top-owasp-kubernetes/
11. Guide for Kubernetes Security Monitoring [Електронний ресурс] – 05.09.2023 - Режим доступу до ресурсу: https://www.practical-devsecops.com/kubernetes-security-monitoring/
12. Kube-hunter [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/aquasecurity/kube-hunter/

13. Kube-audit [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/Shopify/kubeaudit
14. Kubesec [Електронний ресурс] – Режим доступу до ресурсу: https://kubesec.io/
15. Kube-Bench [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/aquasecurity/kube-bench/blob/main/job.yaml
16. Kube-score [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/zegl/kube-score
17. Docker [Електронний ресурс] – Режим доступу до ресурсу: https://www.docker.com/
18. Datree [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/datreeio/datree
19. Kube-linter [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/stackrox/kube-linter
20. Kubescape [Електронний ресурсу] – Режим доступу до ресурсу: https://github.com/kubescape/kubescape
21. Microservices-demo [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/GoogleCloudPlatform/microservices-demo
22. Апробація роботи [Електронний ресурс] – Режим доступу до ресурсу: https://archive.liga.science/index.php/conference-proceedings/issue/view/inter-19.01.2024

ДОДАТОК А ПОРІВНЯННЯ РЕЗУЛЬТАТІВ СКАНУВАННЯ НА РІЗНИХ ВЕРСІЯХ

Перший тест:

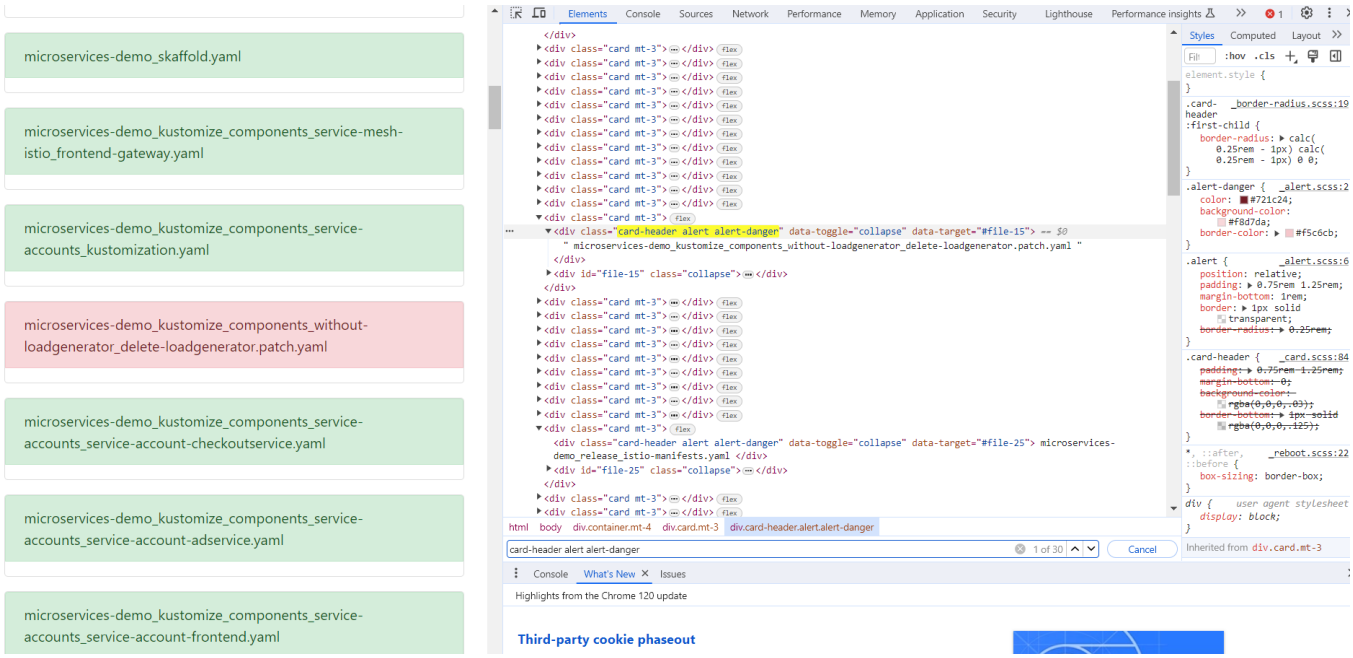


Рисунок А.1 – Результат першого тестування (вразливості)

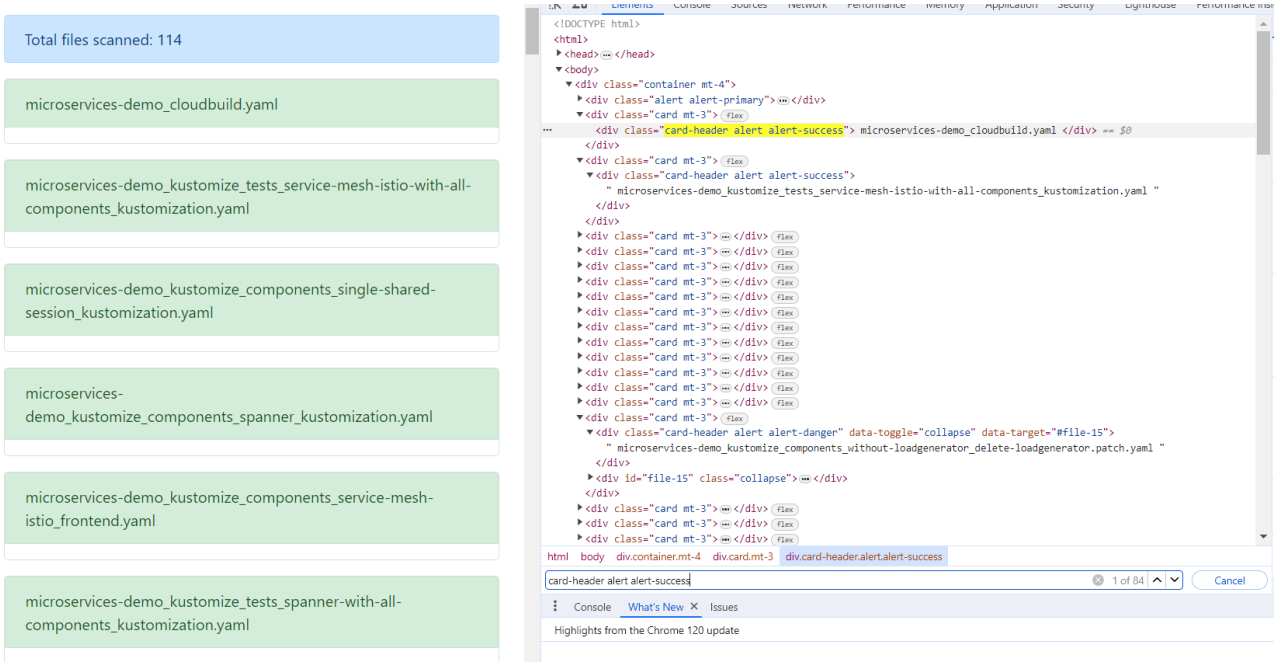


Рисунок А.2 – Результат першого тестування (пройдені тести)

Другий тест:

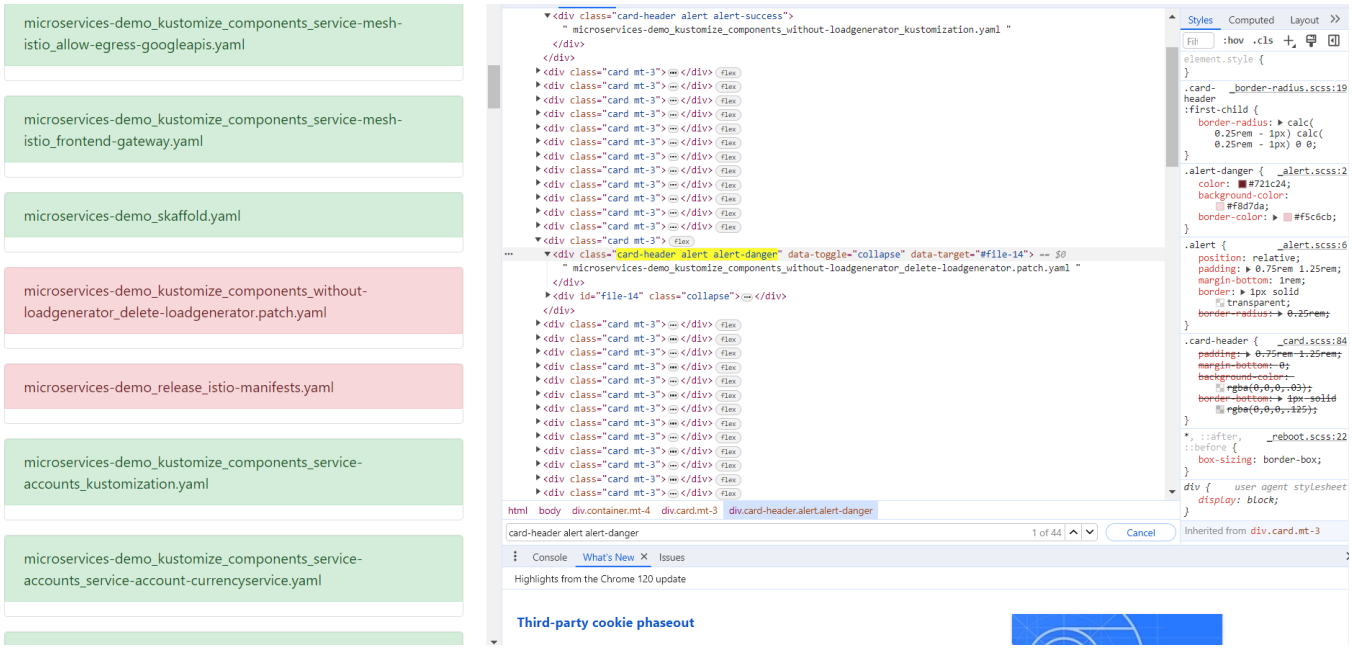


Рисунок А.3 – Результат другого тестування (вразливості)

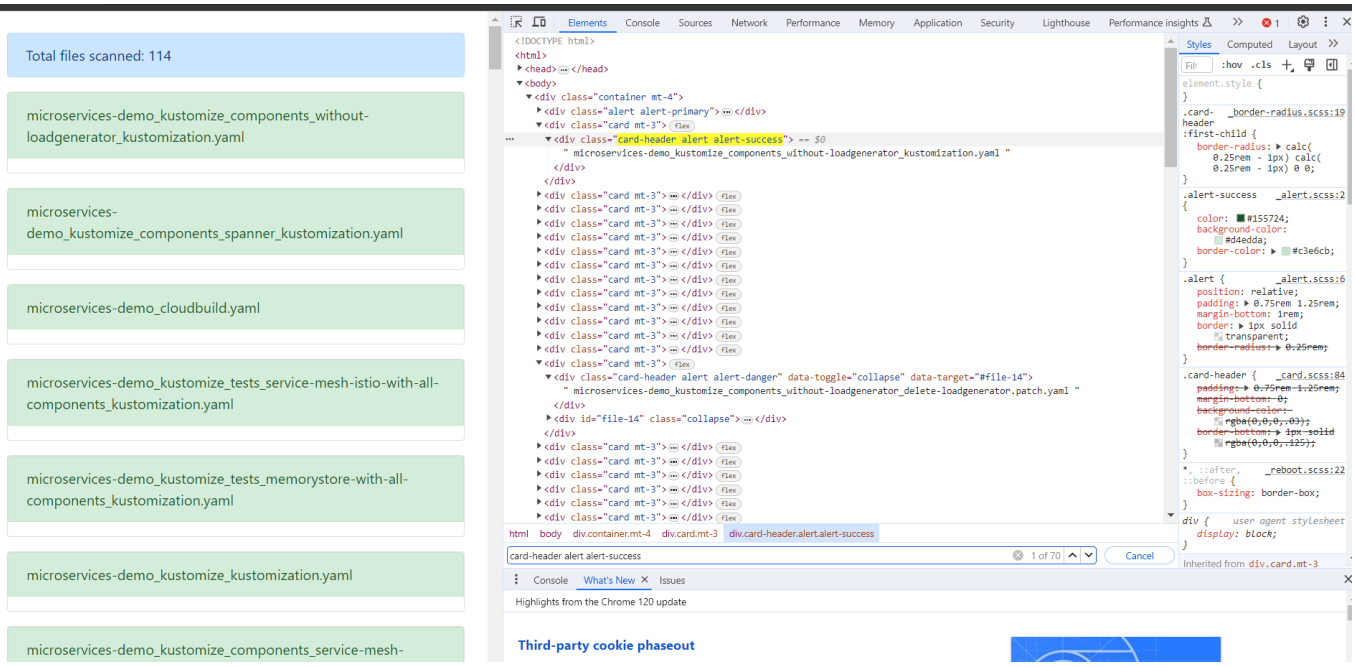


Рисунок А.4 – Результат другого тестування (пройдені тести)

Третій тест:

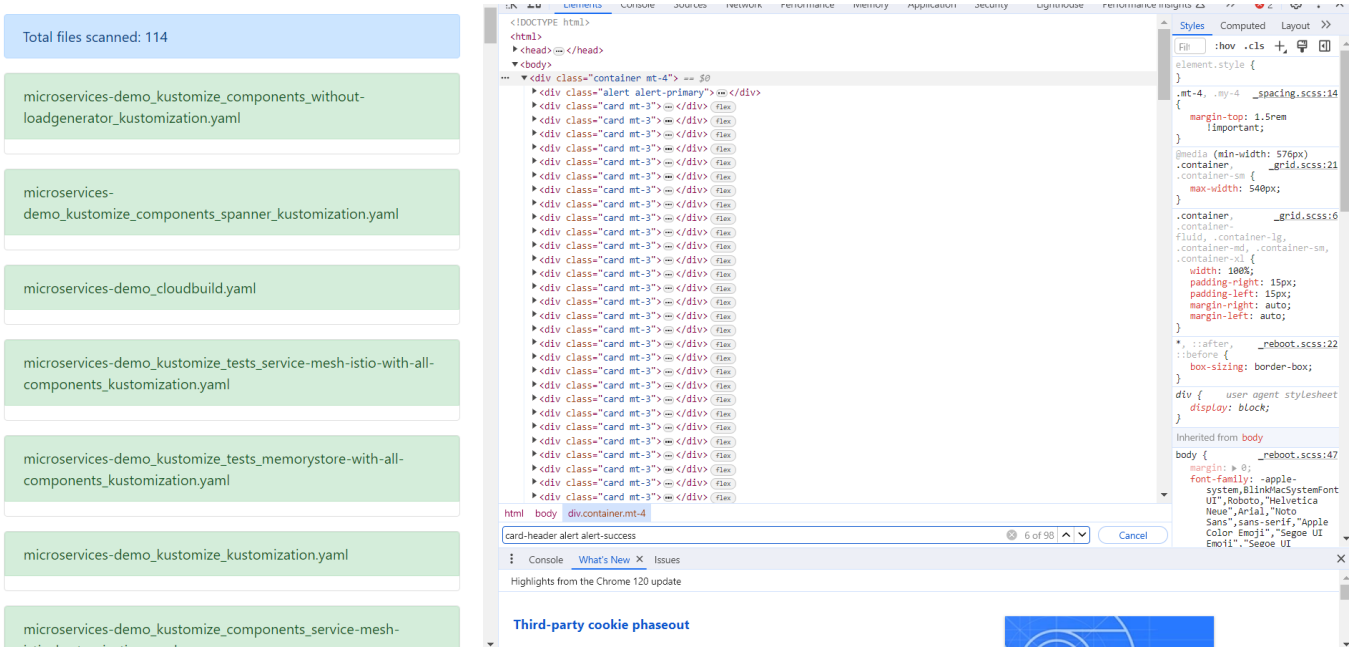


Рисунок А.5 – Результат третього тестування (успішні тести)

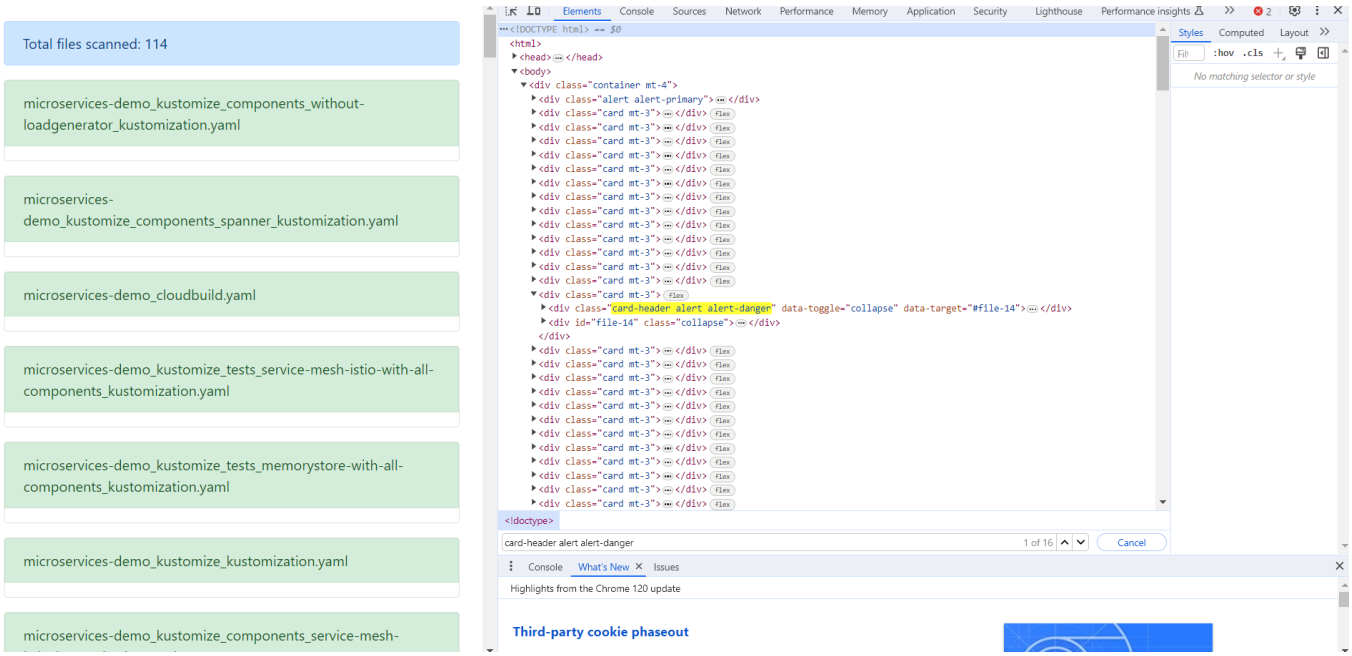


Рисунок А.6 – Результат третього тестування (вразливості)

ДОДАТОК Б ПРОГРАМНИЙ КОД ЗАПРОПОНОВАНОЇ УТИЛІТИ

https://github.com/vladislav2402/diploma