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)

«_____»_____2022 р.

Дипломна робота

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Системи, технології та математичні методи кібербезпеки»

спеціальності 125 «Кібербезпека»

на тему: Розробка автоматичних засобів забезпечення безпеки Docker

_____

_____

Виконав (-ла): здобувач вищої освіти **IV** курсу, групи __ФБ-84___
(шифр групи)

Кравченко Валентин Володимирович 
(прізвище, ім'я, по батькові) (підпис)

Керівник Гальчинський Леонід Юрійович, к.т.н., доцент
(посада, науковий ступінь, вчене звання, прізвище, ім'я, по батькові) (підпис)

Рецензент _____ _____
(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 – 2022 року

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

Кафедра інформаційної безпеки

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 125 «Кібербезпека»

Освітньо-професійна програма «Системи, технології та математичні методи кібербезпеки»

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ Дмитро ЛАНДЕ

(підпис)

«___»_____2022 р.

ЗАВДАННЯ

на дипломну роботу здобувачу вищої освіти

Кравченко Валентину Володимировичу

1. Тема роботи: «Розробка автоматичних засобів забезпечення безпеки Docker» керівник роботи к.т.н., доцент кафедри Інформаційної безпеки Гальчинський Леонід Юрійович, затверджені наказом по університету від « » _____ 2022 р. №

2. Термін подання здобувачем вищої освіти роботи 10 червня 2022 р.

3. Вихідні дані до роботи: Існуючи методи та засоби забезпечення безпеки Docker, методи програмної взаємодії з системою Linux та Docker, документація Docker.

4. Зміст роботи: Огляд контейнерної віртуалізації, її недоліків, переваг та областей застосування; огляд задачі забезпечення безпеки Docker, методів та засобів які для цього використовуються; розробка програмного забезпечення для автоматичного налаштування безпеки Docker.

5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо): презентація

6. Дата видачі завдання 14 листопада 2021 р.

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів дипломної роботи	Примітка
1	Визначення напрямку роботи.	05.09.2021-27.09.2021	
2	Вибір теми роботи, чітке визначення мети роботи, а також поставлених задач.	27.09.2021-15.11.2021	
3	Визначення наукової літератури яка буде використовуватися для наукового підгрунтя.	15.11.2021-10.12.2021	
4	Розгляд контейнерної віртуалізації, визначення її переваг та недоліків. Написання першого розділу дипломної роботи	10.12.2021-16.01.2022	
5	Огляд відомих вразливостей у Docker, їх впливу на цільову систему.	17.01.2022-25.02.2022	
6	Розгляд відомих засобів та методів забезпечення безпеки Docker, виокремлення найбільш важливих для поставлених задач.	25.02.2022-19.03.2022	
7	Написання другого розділу бакалаврської дипломної роботи, ознайомлення зі скануючими утилітами.	19.03.2022-02.05.2022	
8	Проходження переддипломної практики	02.05.2022 - 29.05.2022	
9	Написання третього розділу дипломної роботи та розробка програмного забезпечення	13.05.2022-31.05.2022	
10	Створення презентації для передзахисту дипломної роботи	01.06.2022-13.06.2022	
11	Передзахист дипломної роботи	14.06.2022	
12	Доопрацювання	15.06.2022-20.06.2022	
13	Захист дипломної роботи	21.06.2022	

Здобувач вищої освіти  Валентин КРАВЧЕНКО

Керівник роботи _____ Леонід ГАЛЬЧИНСЬКИЙ

(підпис)

РЕФЕРАТ

Дипломна робота має обсяг 81 сторінок, містить 22 рисунки, 1 таблицю, 11 додатків та 18 джерел посилань.

Контейнери стали однією з найбільш затребуваних технологій компаніями при розгортанні великої інфраструктури у хмарних середовищах або на локальних фізичних машинах через велику кількість її переваг. У сьогоднішній час Docker є лідером серед інструментаріїв керування контейнерами. Через це забезпечення безпеки Docker є однією з найбільш приоритетних задач будь-якої компанії, яка використовує контейнерну віртуалізацію, оскільки недотримання безпеки може нести дуже великі втрати як для компаній так і для користувачів, які використовують технологію контейнерів для персональних цілей.

Об’єктом дослідження є забезпечення безпеки Docker.

Предметом дослідження є розробка автоматичного програмного забезпечення.

Метою дослідження є аналіз існуючих вразливостей Docker, методів та засобів забезпечення безпеки Docker та розробка універсального автоматичного програмного забезпечення для налаштування безпеки Docker після його встановлення або вдосконалення вже існуючих налаштувань, яке допомогло би пришвидшити процес базового налаштування безпеки та допомогти захистити свою систему користувачам, які навіть не знайомі з Docker.

Робота містить опис відомих вразливостей Docker; розбір відомих методів та засобів забезпечення безпеки Docker, виокремлення найбільш важливих для поставленої мети дослідження методів та засобів; розгляд відомих утиліт для сканування системи Docker; розробка універсального автоматичного програмного забезпечення для налаштування безпеки Docker на цільовій системі.

Ключові слова: Docker, Python, Docker Bench for Security, DockerFile, Docker daemon, конфігурація Docker.

ABSTRACT

Thesis has a volume of 81 pages, contains 22 pictures, 1 table, 11 appendices and 18 sources of references.

Containers have become one of the most sought-after technologies by companies in deploying large infrastructure in cloud environments or on local physical machines because of its many benefits. Today, Docker is a leader in container management tools. Because of this, Docker's security is one of the top priorities of any company that uses container virtualization, as security breaches can be very costly for both companies and users who use container technology for personal use.

The object of the study is to ensure the security of Docker.

The subject of research is the development of automated software.

The aim of the study is to analyze existing Docker vulnerabilities, Docker security methods and tools, and to develop universal automated software to configure Docker security after installing or upgrading existing settings to help speed up the basic security configuration process and help protect your system. not familiar with Docker.

The work describes Docker's known vulnerabilities; analysis of known methods and tools for security Docker, highlighting the most important for the purpose of the research methods and tools; review of well-known utilities for scanning the Docker system; development of universal automated software to configure Docker security on the target system.

Keywords: Docker, Python, Docker Bench for Security, DockerFile, Docker daemon, Docker configuration.

ЗМІСТ

Перелік умовних позначень, символів, одиниць, скорочень і термінів

ОС – Операційна Система

Docker - інструментарій для управління ізольованими Linux-контейнерами.

Docker Engine – див. Docker

Docker daemon - це постійний фоновий процес, який керує контейнерами на одному хості.

DockerFile - це текстовий документ, який містить усі команди, які користувач може викликати в командному рядку, щоб зібрати образ. Образ Docker містить код програми, бібліотеки, інструменти, залежності та інші файли, необхідні для запуску контейнеру.

TLS — це криптографічний протокол, призначений для забезпечення безпеки зв'язку через комп'ютерну мережу. Використовує асиметричне шифрування і сертифікати X.509.

JSON — є відкритим стандартним форматом файлу та форматом обміну даними, який використовує зрозумілий людині текст для зберігання та передачі об’єктів даних, що складаються з пар атрибут-значення та масивів (або інших значень, які можна серіалізувати).

YAML — є зрозумілою для людини мовою серіалізації даних. Він зазвичай використовується для файлів конфігурації та в програмах, де дані зберігаються або передаються.

Деплоймент - метод швидкого створення та випуску комплексних програм.

API - це програмний посередник, який дозволяє двом додаткам спілкуватися один з одним.

Orchestrator - – це програмне забезпечення, яке виконує автоматизовану конфігурацію, координацію та керування комп’ютерними системами та програмним забезпеченням.

Distroless образи – ці образи містять лише вашу програму та її залежності під час виконання. Вони не містять менеджерів пакетів, оболонок або будь-яких інших програм, які були б очікувано знайти в стандартному дистрибутиві Linux.

Вступ

На початкових етапах компанії розгортали свої проекти на фізичних серверах. Але даний підхід мав критичний недолік – не було можливості визначати границі ресурсів для програм на фізичних серверах, що призводило до проблем з розподіленням ресурсів. Рішенням цієї проблеми був запуск кожної програми на окремому фізичному сервері, але це не було повноцінним рішенням проблеми, бо призводило до нового недоліку – не існувало можливості масштабувати систему, оскільки тримати велику кількість фізичних серверів дуже дорого для компанії, а ресурси на цих серверах найчастіше використовувались не повністю.

У якості рішення була винайдена віртуалізація. Ця технологія дозволяла запускати декілька віртуальних машин на одному процесорі фізичного серверу. До того ж віртуалізація дозволяла ізолювати програми між віртуальними машинами, що також підвищувало рівень безпеки у систем, оскільки інформація одної програми не була доступна іншій програмі. Віртуалізація також дозволяла більш корисно розподіляти ресурси фізичного серверу забезпечуючи кращу масштабованість.

Контейнери є прямими наслідками віртуальних машин, а також мають ряд переваг перед віртуальними машинами: вони можуть поділяти операційну систему між програмами, пришвидшують створення та розгортання програм, набагато краще працюють з хмарними технологіями (які зараз використовуються більшістю компаній по всьому світу), тощо. Саме завдяки своїм перевагам контейнери стали одним з найбільш використовуваних видів віртуалізації, через це їх безпека є дуже критичною, оскільки може вести до великих збитків компаній.

Ця дипломна робота спрямована на дослідження безпеки контейнерів та розробку універсального автоматичного програмного забезпечення для налаштування цієї безпеки.

1. Системи Віртуалізації.

1.1. Порівняння контейнерної та апаратної віртуалізації та їх архітектур.

Віртуалізація – створення ізольованих середовищ у рамках одного фізичного пристрою. Кожне середовище при цьому є окремим пристроєм зі своїми характеристиками, наприклад доступна пам’ять, процесор та тощо. Таке середовище називається набором логічних ресурсів чи віртуальною машиною.

Прикладом використання віртуалізації є запуск декількох операційних систем на одному комп’ютері. Наданням ресурсів для цих операційних систем займається базова операційна система – гіпервізор. Тобто гіпервізор є спеціальною програмою яка займається створенням віртуальних машин та їх керуванням. Гіпервізор забезпечує ізоляцію операційних систем одна від одної, їх захист та безпеку, а також розділення ресурсів між запущеними операційними системами. В залежності від типу використовуваної віртуалізації гіпервізор може спілкуватися напряму с залізом комп’ютера без запитів до хост-системи, або через основну операційну систему, яка встановлена на хост-систему. У першому випадку використовується Апаратна віртуалізація, у другому – програмна віртуалізація.
Віртуалізація значно безпечніша ніж встановлення операційних систем у одну машину, оскільки у будь-який момент є можливість видалити віртуальну машину та встановити новий образ.

При цьому існує також контейнерна віртуалізація яка не пов’язана з запуском операційної системи у ізольованому середовищі. При використанні контейнерної віртуалізації ізоляція відбувається на рівні процесу операційної системи. Сьогодні можливість такої віртуалізації можлива лише у системах базованих на Linux оскільки ядро у цій системі має cgroups та namespaces можливості. Ці можливості дозволяють запускати усього один процес у власному середовищі, з власною мережею, своїм диском та своєю файловою системою. Оскільки при такій віртуалізації запуск відбувається у основній операційній системі та на основному ядрі, то не можливо у Linux запустити Windows. Цей вид віртуалізації дуже широко використовується на рівні

сервісів, які є складовими програмного продукту. Найбільш відомою реалізацією є проект Docker.

Кожна з реалізацій віртуалізації має свої переваги, які є важливим фактором у тих чи інших випадках. До того ж слід зазначити, що у багатьох випадках контейнері середовища запускаються на віртуальних машинах в якості хост серверів, особливо у хмарних технологіях, які зараз використовуються у більшості компанії.

1.1.1. Архітектура контейнерної віртуалізації.

Основною величиною контейнерної віртуалізації є контейнер. Контейнер – це ізольоване, невимогливе до ресурсів сховище для запуску прикладних програм у операційній системі хоста. Контейнери містять лише прикладні програми, деякі невимогливі API операційних систем та сервіси, працюючі у режимі користувача. Структуру можна побачити на рисунку 1.1.[1]

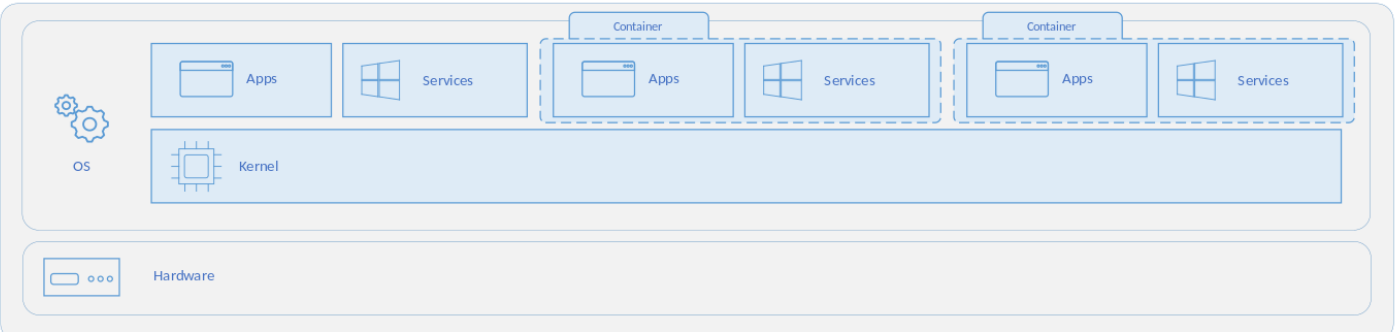

Рисунок 1.1 [1]

1.1.2. Архітектура віртуальної машини

Віртуальні машини запускаються під управлінням повної операційної системи, яка має власне ядро. Структуру можна побачити на рисунку 1.2.[1]

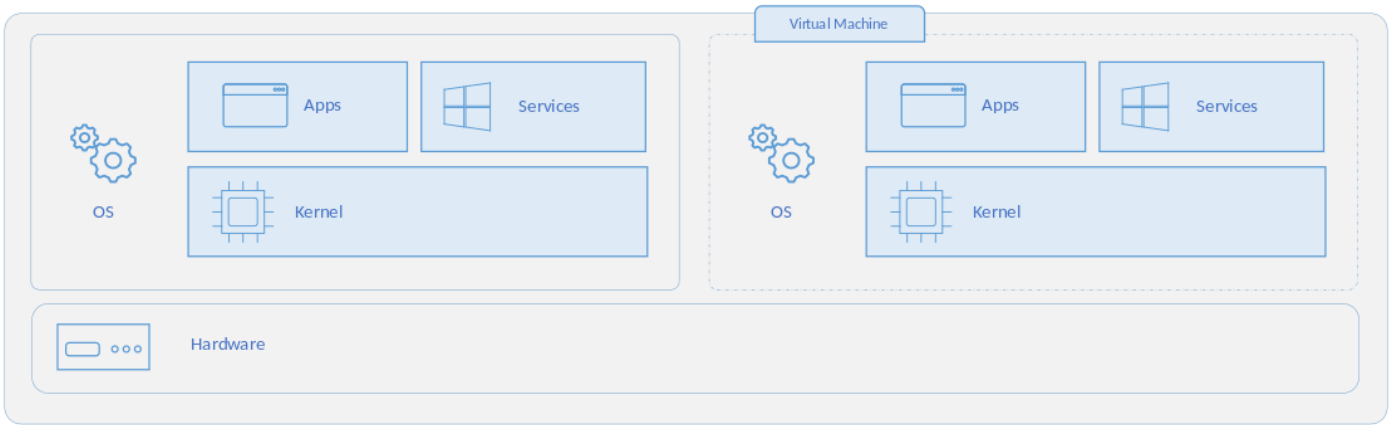

Рисунок 1.2 [1]

1.1.3. Відмінності у технологіях контейнера та віртуальної машини.

Функції	Віртуальна машина	Контейнер
Ізоляція	Дозволяє повністю ізолюватись від операційної системи хост-системи та інших віртуальних машин. Найбільш корисним такий підхід є при необхідності встановлення строгих меж безпеки. Наприклад при запуску конкуруючих продуктів(компанії) у одному кластері або сервері.	Не може надавати строгих меж безпеки як у випадку віртуальної машини, зазвичай надає базову ізоляцію від хоста та інших контейнерів. Але завдяки додатковим засобам безпека може бути підвищена(наприклад при використанні Hyper-V ізоляції).
Операційна система	Запускає повну операційну систему, яка має власне ядро. Такий підхід найчастіше вимагає більше системних ресурсів, таких як центральний процесор, пам’ять, сховище).	Запускає частини операційної системи у режимі користувача. Оскільки контейнери мають дуже гнучке налаштування, то система буде мати лише найбільш важливі сервіси, що дозваляє економити системні ресурси.
Сумісність з гостьовою системою	Майже будь-яка операційна система може бути використана.	Використовує ту саму версію операційної системи як і хост. Але можливе встановлення більш ранніх версій при використанні додаткових засобів.

Деплоймент	Деплоймент окремих віртуальних машин з використанням гіпервізорів(наприклад Hyper-V). Можливий деплоймент будь-якої кількості віртуальних машин з використанням PowerShell.	У технології Docker деплояться окремі контейнери з використанням командного рядка. При використанні K8S технології можливий деплоймент будь-якої кількості контейнерів.
Оновлення Операційних Систем	Для оновлення віртуальних машин потрібно оновлювати кожну віртуальну машину окремо. Якщо порівнювати такий підхід з використанням K8S для оновлення встановлених контейнерів, то оновлення віртуальних машин займає значно більший час.	Оновлення окремого контейнера виконується по таких кроках: 1. Змінити DockerFile контейнера для того, щоб він використовував актуальні версії образів. 2. Перезібрати образ контейнеру використовуючи оновлений DockerFile. 3. Надіслати новий образ до свого репозіторію на DockerHub(необов'язковий пункт при роботі на локальній системі для тестування). 4. Запустити деплоймент оновленого образу. Однак контейнерні засоби також мають потужні можливості до запуску оновлення на великому обсязі

		контейнерів, наприклад це використовуються у таких технологіях як K8S.
Сховище	Використовує віртуальні жорсткі диски для локального сховища, або загальні файлові ресурси для розділення пам’яті між декількаами віртуальними машинами.	Використовує жорсткий диск хост-системи для зберігання образів та DockerFiles, а також для переносу файлів хост-системи у контейнер, а також використовує сутність volume для зберігання власних даних.
Розпроділення навантаження	У відмовостойкому кластері віртуальна машина переміщується на інші сервери.	Контейнери не можуть переміщуватися самостійно. Але orchestrator може по заданих правилах, або автоматично переміщувати контейнери між серверами(нодами) кластерів при змінах у навантаженні, чи при відмові окремих контейнерів.
Відмовостійкість	При відмові віртуальної машини аналогічна віртуальна машина буде запущена на іншому сервері кластеру.	Коли нода кластера відмовляє, усі контейнери які були на цій ноді перерозподіляються між працюючими нодами кластеру відповідно до навантаження. Також якщо відмовляє певний контейнер, то його аналог буде піднятий

		на будь-якій с працюючих нод кластеру.
Мережі	Використовує віртуальні мережеві адаптери.	Використовується ізольований вид віртуальних мережевих адаптерів, у цьому випадку брандмауер хоста використовується спільно с контейнерами, що дозволяє економити ресурси.

таблиця 1.1 [1]

1.2. Віртуалізація Мереж.

Віртуалізація мережі – це абстрагування мережевих ресурсів, які традиційно надавалися фізичними засобами, та їх реалізація програмним чином. При віртуалізації мережі декілька фізичних мереж поєднуються у одну віртуальну мережу, або одна фізична мережа поділяється на окремі та незалежні віртуальні мережі. Основною метою віртуалізації мережи є встановлення рівню абстракції між фізичним обладнанням та діяльністю, яка використовує це обладнання.

Використання віртуальних мереж дозволяє оптимізувати використання мережевих ресурсів, а також покращити швидкість, гнучкість та надійність мереж. [14]

Існує два види віртуалізації мережи з використанням VLAN технології: зовнішня та внутрішня віртуалізація. Зовнішня віртуалізація дозволяє поєднувати системи, які фізично підключені до однієї локальної мережі(LAN), у окремі віртуальні локальні мережі(VLAN) чи, навпаки, поділяти окремі LAN у одну VLAN. Внутрішня віртуалізація мережі на відміну від внутрішньої віртуалізації, яка діє на системі за межами одного сервера, діє в межах одного сервера для емуляції фізичної мережі. Найчастіше внутрішня віртуалізація використовується для підвищення ефективності роботи серверу та потребує налаштування серверу за допомогою програмних контейнерів.

Іншим прикладом віртуалізації мереж є принцип накладання мереж. Для нього використовується протокол інкапсуляції під назвою VXLAN, який забезпечує основу для накладання віртуальних мереж рівня 2 на мережі рівня 3. Аналогом до протокола VXLAN може стати Generic Network Virtualization Encapsulation protocol (GENEVE) [15], який надає інший спосіб інкапсуляції, розроблений для забезпечення незалежності площини керування між кінцевими точками тунелю.

Також можливе створення віртуалізацій мереж з використанням платформ спеціалізованих для цього, наприклад VMware NSX. VMware NSX Data Center переносить компоненти мережі та безпеки, такі як комутація, брандмауер та маршрутизація, які визначаються та використовуються у програмному забезпеченні.

Переваги при використанні мережевої віртуалізації[16]:

1. Віртуальна мережа дозволяє економити на придбанні апаратного забезпечення. Наприклад COTS(Commercial off-the-shelf) блоки, які є прикладом товарного обладнання, зазвичай є значно дешевшими ніж власне обладнання.
2. Взаємодія з віртуальними мережами відбувається через спільну консоль керування, де адміністратор має повну інформацію щодо мережи. Це є значно швидшим ніж керування кожним окремим обладнанням при використанні фізичних мереж.
3. Мережева інфраструктура на базі програмного забезпечення дозволяє значно легше та швидше масштабувати екземпляри контейнерів при збільшенні навантаження на ресурси.
4. Використання віртуальних мереж значно поліпшує рівень мережевої безпеки організації, оскільки програмне забезпечення безпеки дозволяє бачити більшу частину мережи та повідомляти про вразливості та події безпеки.
5. Можливість переміщати та розподіляти навантаження незалежно від фізичної топології.

1.3. Аналіз вразливостей які присутні у технології контейнерної віртуалізації.

Менш ніж за десять років концепція контейнерів вийшла на передній край передових обчислювальних технологій, став однією з невід'ємних частин хмарних технологій. Контейнери підтримуються багатьма найбільш відомими хмарними інфраструктурами та використовуються у більшості мобільних програм.

Однак таке широке використання спричиняє багато приводів до пошуку та застосування вразливостей у безпеці задля атак на крупні компанії. Тому дуже важливим є коректне та повне впровадження безпеки для контейнеризованих застосунків.

В цьому пункті метою є розглянути приклади відомих атак на контейнери, які були знайдені у Docker та використовувались зловмисниками.

1.3.1. runC. CVE-2019-5736.

У проекті з відкритим вихідним кодом runC, який забезпечував можливість виконання майже усіх контейнерних технологій різних виробників, була знайдена вразливість, яка дозволяла зловмиснику отримати root-доступ до хосту. Для отримання доступу необхідно було переписати двійковий файл runC хоста використовуючи контейнер із шкідливим програмним забезпеченням.

Ця вразливість підкреслює ризик використання образів, які можуть бути підконтрольні зловмисникам, а також ситуацій коли зловмисник отримує можливість вийти за ізоляцію контейнеру над яким він отримав контроль та отримати доступ до хоста. [2]

1.3.2. PHP Runtime для Apache OpenWhisk. CVE-2018-11756.

Дія Docker, виконуєма як безсерверна функція (наприклад wsk action create <name> -docker <image>), де DockerFile, використовуємий для створення образу, наслідує один із зачеплених тегів, може дозволити підробленому параметру перезаписати безсерверну функцію, яка запущена всередині контейнеру. При цьому наступні виконання вихідної функції в

цьому контейнері буде використовувати замінену реалізацію, якщо атака була успішно реалізована. Атака яка використовує цю вразливість дуже залежить від коду всередині контейнеру, код має бути вразливим до, наприклад, перехвату параметрів, віддаленого виконання коду, або небезпечного використання «eval()». [2]

Вразливість була можливою через проблему з дозволами, привілеями та контролем доступу.

1.3.3. util.c у runV. CVE-2018-9862.

util.c у runV для Docker некоректно обробляв числове ім’я користувача, що дозволяло зловмисникам отримати доступ root, використавши наявність початкового числового значення у рядку/etc/passwd, а після виконавши команду «docker exec» з цим значення у аргументі –u. [2]

1.4. Аналіз вразливостей які присутні у мережевій технології контейнерної віртуалізації.

Одна з причин, чому контейнери та сервіси Docker настільки потужні, полягає в тому, що їх можливо поєднувати разом та підключати к робочим навантаженням не пов’язаним з Docker. Контейнерам і службам Docker навіть не потрібно знати, що вони розгорнуті на Docker. Незалежно від того працюють вузли Docker під керуванням Linux, Windows або суміші цих двох операційних систем, ви можете використовувати Docker для керування ними. Такий принцип роботи базується на тому, що контейнерні мережі можуть виконувати функції комунікації між контейнерами, між контейнером та хостом та між серверами, що знаходяться за межами хоста.

Мережева підсистема Docker є підключаємою та використовує драйвери. Існує 6 драйверів за замовчуванням, які забезпечують основну мережеву функціональність:

1. Bridge: Мережевий драйвер за замовчуванням. Мостові мережі зазвичай використовуються, коли ваші прикладні програми працюють в автономних контейнерах, яким необхідна взаємодія. Контейнери з таким мережевим драйвером не можуть взаємодіяти з контейнерами поза межами хоста.

2. Host: При використанні цього мережевого драйвера мережевий стек контейнера не ізольований від хоста, та контейнеру не виділяється власна IP адреса. Цей мережевий драйвер може бути корисний для оптимізації продуктивності та в ситуаціях, коли контейнер обробляє великий діапазон портів, оскільки він не потребує NAT та для кожного порта не створюється «userland-proxy». Цей мережевий драйвер схильний до конфліктів портів, оскільки мережеві інтерфейси використовуються спільно з хостом чи іншими контейнерами.

3. Overlay: Цей драйвер мережі поєднує декілька демонів Docker разом та дозволяє глобальним сервісам взаємодіяти одне з одним. Також може бути використаний для забезпечення зв'язку між глобальним сервісом та окремим контейнером, або між двома окремими контейнерами на різних демонах Docker. Цей драйвер позбавляє від необхідності виконувати маршрутизацію між цими контейнерами на рівні операційної системи.

4. IPvlan: Мережі IPvlan дозволяють користувачу повний контроль над адресацією IPv4 та IPv6. Драйвер VLAN розвиває цей принцип дозволяючи операторам повний контроль над тегами другого рівня та маршрутизацією IPvlan L3 для ситуацій, коли необхідна інтеграція з оверлейними мережами.

5. MACvlan: Мережі MACvlan дозволяють назначати MAC-адресу контейнеру, щоб у мережі він приймав вигляд фізичного пристрою. Демон Docker направляє трафік к контейнерам базуючись на їх MAC-адреси. Цей драйвер найкраще показує себе при використанні з застарівшими програмами, які очікують прямого підключення до фізичної мережі, а не маршрутизації через мережевий стек хоста.

6. None: При використанні цього драйвера для контейнера відключаються усі мережеві підключення. Найчастіше використовується у комбінації з користувацьким мережевим драйвером, або для тестувань.

Кожен з описаних вище драйверів має свої переваги та використовується у конкретних ситуаціях. Але при неправильному налаштуванні або використанні кожен з них може стати потенційними

джерелом вразливостей для програмного застосунку, який запущений у контейнері. [13]

1.4.1. IPv6 вразливість перевірки введення. CVE-2020-13401.

У Docker є вразливість при створенні мережевих мостів, які приймають IPv6 об’явлення маршрутизаторів за замовчуванням. Ця вразливість дозволяє зловмиснику, який може виконувати код у контейнері, підробити IPv6 об’явлення маршрутизаторів для проведення атаки man-in-the-middle проти мережі хоста або іншого контейнера. [2]

1.5. Розгляд втрат, які можуть спричинити вразливості при використанні зловмисником.

Дефекти у компонентах контейнерів можуть дозволяти зловмиснику отримати контроль над всією системою, забезпечити доступ к конфіденційним даним, що приводить до фінансових та репутаційних збитків, а також збоях у роботі системи.

Дуже гарним прикладом буде нещодавно знайдена вразливість з Apache log4j CVE-2021-44228[6], яка дозволяла запустити віддалене виконання коду на будь-якій системі яка користується цією логінг платформою. Оскільки будь-який контейнер, який використовує Java мову програмування, був під загрозою, то компанії понесли великі збитки від атак зловмисника. Зловмисник міг безпосередньо підключитися до контейнера та запустити шкідливий код, або добути інформацію стосовно системи, наприклад версію, архітектуру та ім’я операційної системи, а також конфіденційні дані, наприклад оскільки AWS(амазон веб сервіси) також були під впливом вразливості, то великий перелік секретних ключів компаній, токени сесій, профілей у AWS, конфігураційних файлів, тощо. Це вже великі збитки для компаній, але це лише невеликий перелік шкоди, яку вона може завдати.

Висновки до розділу 1

Виходячи з результатів досліджень проведених у розділі 1 було зроблено два важливих підсумки:

1. Контейнерна віртуалізація широко використовується у сучасних інфраструктурах та її використання буде збільшуватись через значні переваги перед іншими методами віртуалізації.
2. Безпека контейнерних застосунків має надзвичайно великий приорітет, оскільки вразливості можуть бути використані зловмисниками для різних цілей та завдають критичної шкоди інфраструктурі та даним компаній.

Ці підсумки призводять до висновку, що ознайомлення та використання відомих методів та засобів забезпечення безпеки Docker є необхідними при використанні контейнерної віртуалізації у інфраструктурі.

Ці знання є важливими не тільки для спеціалізованих на безпеці департаментів компанії, а і для, наприклад, DevOps підрозділів для уникнення найбільш поширених вразливостей у безпеці на базовому рівні.

2. Розгляд найбільш відомих методів та засобів, які використовуються у безпеці Docker

Система Docker дуже багаторівнева і комплексна, усі її складові щільно пов’язані і співпрацюють разом. Через це вразливість навіть у базовій безпеці Docker може бути шляхом зловмисника до контролю над усією системою.

Саме тому методи та засоби забезпечення безпеки Docker є важливими для розуміння кожним користувачем який використовує систему Docker. У цьому розділі метою є розібрати основні методи та засоби забезпечення Docker, їх використання і причини з яких вони є необхідними.

2.1. Базовий образ.

Перший рядок DockerFile – це інструкція FROM, яка вказує на базовий образ на основі якого створюється новий образ.

Завжди використовуйте образи із довіреного реєстру, оскільки базові образи сторонніх розробників можуть мати шкідливий код. Саме з цієї причини більшість організацій наказують використовувати попередньо схвалені, або так звані «золоті» базові образи.

Дуже корисною практикою є збірка образів з нуля, або використання мінімальних базових образів, наприклад distroless. Це значно зменшує площу атаки, оскільки зменшується непотрібний код. [3]

2.2. Використання багатоетапних збірок.

Багатоетапна зборка – це спосіб виключення непотрібних складових у кінцевому образі. Початковий етап може включати усі пакети та інструментарій, але більшість цих інструментів не потрібні під час виконання контейнеру. Наприклад, якщо контейнер використовує написаний виконуваний файл на C++, то йому потрібен компілятор C++ для створення виконуваної програми, але під час виконання контейнеру компілятор буде надлишковим. [3]

2.3. Монтування томів

При створенні контейнеру базовими командами всередину контейнеру монтуються томи з хоста. Дуже важливим є виключення монтування таких

конфіденційних каталогів, як /etc або /bin, оскільки атака контейнера тоді веде також до витоку конфіденційних даних хоста. [3]

2.4. Команда RUN

Команда RUN у DockerFile дозволяє виконувати будь-яку команду. Якщо зловмисник може скомпрометувати DockerFile з налаштуваннями за замовчуванням, то він може виконати будь-який код за його бажанням. Потрібно бути впевненим, що привелеї на редагування DockerFile є тільки у потрібних учасників організації, а також прискіпливо перевіряти код при будь-яких змінах у ньому. Також гарною практикою є ведення журналу аудиту для DockerFile. [3]

2.5. AppArmor

AppArmor – це модуль безпеки Linux який захищає операційну систему та її програми від погроз безпеки. Docker автоматично генерує та загружає профіль за замовчуванням для контейнерів з ім’ям docker-default. Бінарний файл Docker генерує цей профіль у tmpfs, а потім загружає у ядро.
Але Docker має можливості для використання користувацьких конфігурації AppArmor при використанні з security-opt опцією. [4]

2.6. Зміна root директорії

Оскільки існує багато атак на контейнери, які призводять до витоку конфіденційних даних хост-системи, зміна root директорії у хост-системі є дуже гарним рішенням для безпеки Docker контейнерів. При запуску та роботі у Docker використовуючи нову root директорію ризики втратити конфіденційні данні значно зменшуються, оскільки при атаці зловмисника він отримає доступ до чистої Debian системи, у котрій не існує жодних конфіденційних даних, які не використовуються у Docker. [3], [9]

2.7. Docker секрети

З точки зору служб глобальних сервісів Docker, секрет – це згусток даних, наприклад паролі, закриті ключі SSH, сертифікати SSL, тощо, які не повинні передаватися по мережі чи зберігатися у незашифрованому виді у DockerFile або у вихідному коді вашої програми. Секрети використовуються для централізованого керування цими даними та безпечної передачі їх тільки

тим контейнерам, яким необхідний доступ к ним. Секрети шифруються під час транспортування. Секрети також використовуються для забезпечення рівня абстракції між контейнером та набором облікових даних. Наприклад коли у вас є окреме середовище розробки, тестування та виробництва для програми. Кожне з цих середовищ може мати окремі облікові дані, які будуть зберігатися з одним секретним ім’ям. Секрети мають ще багато різних імплементації у Docker, тому вони є незамінними при налаштуванні контейнерної системи. [10]

2.8. Шифрування оверлей мереж

Мережі у Docker можуть бути зашифровані для додаткового захисту даних. При використанні шифрування для оверлей драйверу Docker встановлює IPSEC шифрування на рівні vxlan. При запуску оверлей шифрування Docker створює IPSEC-тунелі між усіма вузлами на яких заплановані завдання для сервісів, які підключені до оверлейної мережі. У цих тунелях використовуються AES алгоритми, вузли менеджера автоматично відновлюють ключі кожні 12 годин. [17], [18]

2.9. Оцінка безпеки за допомогою додаткових засобів.

Під час розгляду та аналізу методів і засобів які використовуються для забезпечення безпеки роботи Docker мною було знайдено програмне забезпечення(сканери) які використовуються для автоматичного аналізу системи Docker та пошуку вразливостей. Після тестування більшості доступних у інтернеті рішень мною були виокремлені такі сканери[11]:

1. Docker Bench for Security
2. Prisma Cloud
3. Cilium
4. Anchore
5. OpenSCAP Workbench
6. Snyk
7. Lynis
8. Clair

Сканери для Docker можно поділити на два типи:

1. Сканери для образів, які використовує Docker.
2. Комплексні сканери.

До першого типу відносяться такі сканери як Snyk, Prisma Cloud і т.д.. До другого типу відносяться сканери Docker Bench for Security, Lynis і т.д..

Сканери другого типу надають більш поверхневу інформацію щодо безпеки образів, але вони також надають інформацію стосовно хост системи, конфігурації Docker, розгорнуту інформацію щодо ресурсів контейнерів та додаткового функціоналу.

Snyk є вбудованим у DockerHub сканером та викликається вбудованою у Docker командою, він має 10 безкоштовних сканувань протягом місяця після логіну у DockerHub, якщо необхідна більша кількість сканувань, то необхідно додатково зареєструватися на Snyk після чого ви отримуєте доступ до безлімітних сканувань образів.

Цей сканер був обраний для аналізу через ряд його переваг над іншими сканерами для образів Docker:

1. Він є вбудованим у Docker, що прибирає будь-яку необхідність додаткової установки, налаштування програмного забезпечення.

2. Основною перевагою його конкурентів є не функціонал, а швидкість виконання, оскільки у дипломній роботі час виконання не є критичною, то ця перевага конкурентів не була вирішальною.

3. Snyk цілком безкоштовний для приватного використання сканер, ціни конкурентів досягають 20-50 долларів на місяць.

4. Команда підтримки Snyk дуже активна і завжди допомагає з вирішенням поточних проблем.

5. Є функціонал який дозволяє сканувати git репозиторії.

Для приклада використання Snyk сканеру був використаний побудований мною образ який використовує python:3.7-alpine образ, як базовий. Як можна побачити на рисунках 2.1 та 2.2 сканер показує, що знайдена достатньо велика кількість вразливостей та пропонує спосіб їх виправлення.

Рисунок 2.1

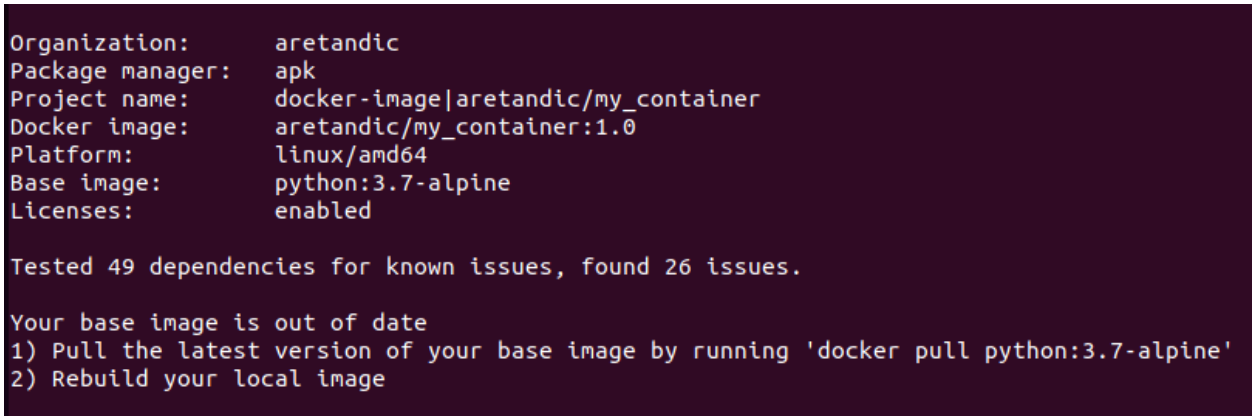

Рисунок 2.2

Docker Bench for Security це одне з найпопулярніших рішень для сканування Docker на данний момент. Аналіз який надає данний сканер базується на CIS Docker Benchmark v1.4.0[7]. Оскільки данне програмне забезпечення є проектом з відкритим кодом, то він постійно доповнюється новим функціоналом. Є різні варіанти використання данного сканеру, наприклад, запуск використовуючи Docker контейнери. Але оскільки сам Docker образ давно не оновлювався, то був використаний відкритий код, який знаходиться на гіт хабі та має більший функціонал.

Цей сканер був обраний для аналізу через ряд його переваг над іншими комплексними сканерами:

1. Docker Bench for Security написаний як shell script, що робить його функціонал дуже простим для розуміння і розбору з точки зору програмного коду.

2. Це проект з відкритим кодом, що дозволяє користувачам додавати новий функціонал до вже існуючого. На данний момент для Docker Bench for Security існує чимало користувацьких плагінів, що додають нові параметри сканування та можуть бути вільно встановленні і використанні.

3. Docker Bench for Security постійно оновлюється та отримує новий функціонал. За час виконання дипломної роботи на сканер вийшло 30 оновлень.

4. Команда підтримки проекту постійно займається покращенням проекту. Наприклад, під час роботи над дипломною роботою мною був знайдений баг у логіці коду, я написав про це команді у проекті на github і після обговорення нами було знайдено рішення як цю помилку виправити.

5. Сканер базується на CIS Docker Benchmark v1.4.0 [7], який є одним з найкращих джерел кращих практик, методів та засобів забезпечення безпеки у Docker.

6. Велика кількість можливих способів запуску сканування. Наприклад, як контейнер, як скріпт на хост системі і т.д..

7. Можливість гнучкого налаштування сканування під необхідні потреби.

8. Доступ до функціоналу цілком безкоштовний, коли деякі конкуренти пропонують аналогічний функціонал лише при підписці.

9. Данний сканер має дуже низьку кількість залежностей, що дозволяє використовувати його без встановлення додаткового функціоналу.

10. Данний параметер не є критичним у дипломнії роботі, але сканер є однією з найбільш швидкодійних утиліт.

Завантаження та запуск коду Docker Bench for Security[5].

При запуску даний сканер надає інформацію по таким пунктам:

1. Конфігурація хоста.

```
[INFO] 1 - Host Configuration
[INFO] 1.1 - Linux Hosts Specific Configuration
[WARN] 1.1.1 - Ensure a separate partition for containers has been created (Automated)
[INFO] 1.1.2 - Ensure only trusted users are allowed to control Docker daemon (Automated)
[INFO]       * Users:
[WARN] 1.1.3 - Ensure auditing is configured for the Docker daemon (Automated)
[WARN] 1.1.4 - Ensure auditing is configured for Docker files and directories -/run/containerd (Automated)
[WARN] 1.1.5 - Ensure auditing is configured for Docker files and directories - /var/lib/docker (Automated)
[WARN] 1.1.6 - Ensure auditing is configured for Docker files and directories - /etc/docker (Automated)
[WARN] 1.1.7 - Ensure auditing is configured for Docker files and directories - docker.service (Automated)
[INFO] 1.1.8 - Ensure auditing is configured for Docker files and directories - containerd.sock (Automated)
[INFO]       * File not found
[WARN] 1.1.9 - Ensure auditing is configured for Docker files and directories - docker.socket (Automated)
[WARN] 1.1.10 - Ensure auditing is configured for Docker files and directories - /etc/default/docker (Automated)
[INFO] 1.1.11 - Ensure auditing is configured for Dockerfiles and directories - /etc/docker/daemon.json (Automated)
[INFO]       * File not found
[WARN] 1.1.12 - 1.1.12 Ensure auditing is configured for Dockerfiles and directories - /etc/containerd/config.toml (A
utomated)
[INFO] 1.1.13 - Ensure auditing is configured for Docker files and directories - /etc/sysconfig/docker (Automated)
[INFO]       * File not found
[WARN] 1.1.14 - Ensure auditing is configured for Docker files and directories - /usr/bin/containerd (Automated)
[WARN] 1.1.15 - Ensure auditing is configured for Docker files and directories - /usr/bin/containerd-shim (Automated)
[WARN] 1.1.16 - Ensure auditing is configured for Docker files and directories - /usr/bin/containerd-shim-runc-v1 (Au
tomated)
[WARN] 1.1.17 - Ensure auditing is configured for Docker files and directories - /usr/bin/containerd-shim-runc-v2 (Au
tomated)
[WARN] 1.1.18 - Ensure auditing is configured for Docker files and directories - /usr/bin/runc (Automated)
[INFO] 1.2 - General Configuration
[NOTE] 1.2.1 - Ensure the container host has been Hardened (Manual)
[PASS] 1.2.2 - Ensure that the version of Docker is up to date (Manual)
[INFO]       * Using 20.10.16, verify is it up to date as deemed necessary
```

Рисунок 2.3

2. Конфігурація Docker daemon.

```
[INFO] 2 - Docker daemon configuration
[NOTE] 2.1 - Run the Docker daemon as a non-root user, if possible (Manual)
[WARN] 2.2 - Ensure network traffic is restricted between containers on the default bridge (Scored)
[PASS] 2.3 - Ensure the logging level is set to 'info' (Scored)
[PASS] 2.4 - Ensure Docker is allowed to make changes to iptables (Scored)
[PASS] 2.5 - Ensure insecure registries are not used (Scored)
[PASS] 2.6 - Ensure aufs storage driver is not used (Scored)
[INFO] 2.7 - Ensure TLS authentication for Docker daemon is configured (Scored)
[INFO]      * Docker daemon not listening on TCP
[INFO] 2.8 - Ensure the default ulimit is configured appropriately (Manual)
[INFO]      * Default ulimit doesn't appear to be set
[WARN] 2.9 - Enable user namespace support (Scored)
[PASS] 2.10 - Ensure the default cgroup usage has been confirmed (Scored)
[PASS] 2.11 - Ensure base device size is not changed until needed (Scored)
[WARN] 2.12 - Ensure that authorization for Docker client commands is enabled (Scored)
[WARN] 2.13 - Ensure centralized and remote logging is configured (Scored)
[WARN] 2.14 - Ensure containers are restricted from acquiring new privileges (Scored)
[WARN] 2.15 - Ensure live restore is enabled (Scored)
[WARN] 2.16 - Ensure Userland Proxy is Disabled (Scored)
[PASS] 2.17 - Ensure that a daemon-wide custom seccomp profile is applied if appropriate (Manual)
[INFO] Ensure that experimental features are not implemented in production (Scored) (Deprecated)
```

Рисунок 2.4

3. Конфігураційні файли Docker daemon.

```
[INFO] 3 - Docker daemon configuration files
[PASS] 3.1 - Ensure that the docker.service file ownership is set to root:root (Automated)
[PASS] 3.2 - Ensure that docker.service file permissions are appropriately set (Automated)
[PASS] 3.3 - Ensure that docker.socket file ownership is set to root:root (Automated)
[PASS] 3.4 - Ensure that docker.socket file permissions are set to 644 or more restrictive (Automated)
[PASS] 3.5 - Ensure that the /etc/docker directory ownership is set to root:root (Automated)
[PASS] 3.6 - Ensure that /etc/docker directory permissions are set to 755 or more restrictively (Automated)
[INFO] 3.7 - Ensure that registry certificate file ownership is set to root:root (Automated)
[INFO]      * Directory not found
[INFO] 3.8 - Ensure that registry certificate file permissions are set to 444 or more restrictively (Automated)
[INFO]      * Directory not found
[INFO] 3.9 - Ensure that TLS CA certificate file ownership is set to root:root (Automated)
[INFO]      * No TLS CA certificate found
[INFO] 3.10 - Ensure that TLS CA certificate file permissions are set to 444 or more restrictively (Automated)
[INFO]       * No TLS CA certificate found
[INFO] 3.11 - Ensure that Docker server certificate file ownership is set to root:root (Automated)
[INFO]       * No TLS Server certificate found
[INFO] 3.12 - Ensure that the Docker server certificate file permissions are set to 444 or more restrictively (Automated)
[INFO]       * No TLS Server certificate found
[INFO] 3.13 - Ensure that the Docker server certificate key file ownership is set to root:root (Automated)
[INFO]       * No TLS Key found
[INFO] 3.14 - Ensure that the Docker server certificate key file permissions are set to 400 (Automated)
[INFO]       * No TLS Key found
[PASS] 3.15 - Ensure that the Docker socket file ownership is set to root:docker (Automated)
[PASS] 3.16 - Ensure that the Docker socket file permissions are set to 660 or more restrictively (Automated)
[INFO] 3.17 - Ensure that the daemon.json file ownership is set to root:root (Automated)
[INFO]       * File not found
[INFO] 3.18 - Ensure that daemon.json file permissions are set to 644 or more restrictive (Automated)
[INFO]       * File not found
[PASS] 3.19 - Ensure that the /etc/default/docker file ownership is set to root:root (Automated)
[INFO] 3.20 - Ensure that the /etc/sysconfig/docker file permissions are set to 644 or more restrictively (Automated)
[INFO]       * File not found
[INFO] 3.21 - Ensure that the /etc/sysconfig/docker file ownership is set to root:root (Automated)
[INFO]       * File not found
[PASS] 3.22 - Ensure that the /etc/default/docker file permissions are set to 644 or more restrictively (Automated)
[PASS] 3.23 - Ensure that the Containerd socket file ownership is set to root:root (Automated)
[PASS] 3.24 - Ensure that the Containerd socket file permissions are set to 660 or more restrictively (Automated)
```

Рисунок 2.5

4. Образи контейнерів та файли для їх збірки.

```
[INFO] 4 - Container Images and Build File
[WARN] 4.1 - Ensure that a user for the container has been created (Automated)
[WARN]      * Running as root: competent_elion
[NOTE] 4.2 - Ensure that containers use only trusted base images (Manual)
[NOTE] 4.3 - Ensure that unnecessary packages are not installed in the container (Manual)
[NOTE] 4.4 - Ensure images are scanned and rebuilt to include security patches (Manual)
[WARN] 4.5 - Ensure Content trust for Docker is Enabled (Automated)
[WARN] 4.6 - Ensure that HEALTHCHECK instructions have been added to container images (Automated)
[WARN]      * No Healthcheck found: [imiell/bad-dockerfile:latest]
[WARN]      * No Healthcheck found: [centos:centos7.2.1511]
[PASS] 4.7 - Ensure update instructions are not used alone in the Dockerfile (Manual)
[NOTE] 4.8 - Ensure setuid and setgid permissions are removed (Manual)
[INFO] 4.9 - Ensure that COPY is used instead of ADD in Dockerfiles (Manual)
[INFO]      * ADD in image history: [imiell/bad-dockerfile:latest]
[INFO]      * ADD in image history: [centos:centos7.2.1511]
[NOTE] 4.10 - Ensure secrets are not stored in Dockerfiles (Manual)
[NOTE] 4.11 - Ensure only verified packages are installed (Manual)
[NOTE] 4.12 - Ensure all signed artifacts are validated (Manual)
```

Рисунок 2.6

5. Час виконання контейнера.

```
[INFO] 5 - Container Runtime
[PASS] 5.1 - Ensure that, if applicable, an AppArmor Profile is enabled (Automated)
[WARN] 5.2 - Ensure that, if applicable, SELinux security options are set (Automated)
[WARN]      * No SecurityOptions Found: competent_elion
[PASS] 5.3 - Ensure that Linux kernel capabilities are restricted within containers (Automated)
[PASS] 5.4 - Ensure that privileged containers are not used (Automated)
[PASS] 5.5 - Ensure sensitive host system directories are not mounted on containers (Automated)
[PASS] 5.6 - Ensure sshd is not run within containers (Automated)
[PASS] 5.7 - Ensure privileged ports are not mapped within containers (Automated)
[PASS] 5.8 - Ensure that only needed ports are open on the container (Manual)
[PASS] 5.9 - Ensure that the host's network namespace is not shared (Automated)
[WARN] 5.10 - Ensure that the memory usage for containers is limited (Automated)
[WARN]       * Container running without memory restrictions: competent_elion
[WARN] 5.11 - Ensure that CPU priority is set appropriately on containers (Automated)
[WARN]       * Container running without CPU restrictions: competent_elion
[WARN] 5.12 - Ensure that the container's root filesystem is mounted as read only (Automated)
[WARN]       * Container running with root FS mounted R/W: competent_elion
[PASS] 5.13 - Ensure that incoming container traffic is bound to a specific host interface (Automated)
[WARN] 5.14 - Ensure that the 'on-failure' container restart policy is set to '5' (Automated)
[WARN]       * MaximumRetryCount is not set to 5: competent_elion
[PASS] 5.15 - Ensure that the host's process namespace is not shared (Automated)
[PASS] 5.16 - Ensure that the host's IPC namespace is not shared (Automated)
[PASS] 5.17 - Ensure that host devices are not directly exposed to containers (Manual)
[INFO] 5.18 - Ensure that the default ulimit is overwritten at runtime if needed (Manual)
[INFO]       * Container no default ulimit override: competent_elion
[PASS] 5.19 - Ensure mount propagation mode is not set to shared (Automated)
[PASS] 5.20 - Ensure that the host's UTS namespace is not shared (Automated)
[PASS] 5.21 - Ensure the default seccomp profile is not Disabled (Automated)
[NOTE] 5.22 - Ensure that docker exec commands are not used with the privileged option (Automated)
[NOTE] 5.23 - Ensure that docker exec commands are not used with the user=root option (Manual)
[PASS] 5.24 - Ensure that cgroup usage is confirmed (Automated)
[WARN] 5.25 - Ensure that the container is restricted from acquiring additional privileges (Automated)
[WARN]       * Privileges not restricted: competent_elion
[WARN] 5.26 - Ensure that container health is checked at runtime (Automated)
[WARN]       * Health check not set: competent_elion
[INFO] 5.27 - Ensure that Docker commands always make use of the latest version of their image (Manual)
[WARN] 5.28 - Ensure that the PIDs cgroup limit is used (Automated)
[WARN]       * PIDs limit not set: competent_elion
[INFO] 5.29 - Ensure that Docker's default bridge 'docker0' is not used (Manual)
[INFO]       * Container in docker0 network: competent_elion
[PASS] 5.30 - Ensure that the host's user namespaces are not shared (Automated)
```

Рисунок 2.7

6. Операції Безпеки Docker.

```
[INFO] 6 - Docker Security Operations
[INFO] 6.1 - Ensure that image sprawl is avoided (Manual)
[INFO]       * There are currently: 2 images
[INFO] 6.2 - Ensure that container sprawl is avoided (Manual)
[INFO]       * There are currently a total of 1 containers, with 1 of them currently running
```

Рисунок 2.8

7. Конфігурація глобальних образів Docker.

```
[INFO] 7 - Docker Swarm Configuration
[PASS] 7.1 - Ensure swarm mode is not Enabled, if not needed (Automated)
[PASS] 7.2 - Ensure that the minimum number of manager nodes have been created in a swarm (Automated) (Swarm mode not enabled)
[PASS] 7.3 - Ensure that swarm services are bound to a specific host interface (Automated) (Swarm mode not enabled)
[PASS] 7.4 - Ensure that all Docker swarm overlay networks are encrypted (Automated)
[PASS] 7.5 - Ensure that Docker's secret management commands are used for managing secrets in a swarm cluster (Manual) (Swarm mode not enabled)
[PASS] 7.6 - Ensure that swarm manager is run in auto-lock mode (Automated) (Swarm mode not enabled)
[PASS] 7.7 - Ensure that the swarm manager auto-lock key is rotated periodically (Manual) (Swarm mode not enabled)
[PASS] 7.8 - Ensure that node certificates are rotated as appropriate (Manual) (Swarm mode not enabled)
[PASS] 7.9 - Ensure that CA certificates are rotated as appropriate (Manual) (Swarm mode not enabled)
[PASS] 7.10 - Ensure that management plane traffic is separated from data plane traffic (Manual) (Swarm mode not enabled)
```

Рисунок 2.9

Оскільки сканування було виконано на базовому дистрибуті Ubuntu, де взагалі не налаштована безпека Docker, то сумарна оцінка безпеки була дуже низька.

```
Section C - Score

[INFO] Checks: 117
[INFO] Score: 8
```

Рисунок 2.10

Висновки до розділу 2

Проаналізувавши методи та засоби забезпечення безпеки Docker описані у пунктах 2.1 – 2.8 та результати скануваннь отримані у пункті 2.9, був зроблений висновок, що велику частину налаштувань безпеки для Docker можно автоматизувати, що значно пришвидшить налаштування безпеки при встановленні Docker на нових системах чи при модифікації безпеки у системах які вже використовують Docker та мають вже встановлені налаштування безпеки. Оскільки багато світових кампаній використовують Docker у комбінації з Kubernetes для інфраструктури, то автоматизація яка дозволяє автоматично налаштовувати безпеку Docker буде дуже корисна не лише для локального використання Docker, а і для глобального. Розробку програмного забезпечення було прийнято роботи базуючись на CIS Docker Benchmark v1.4.0[7] та скануючій утиліті Docker Bench for Security.

3. Розробка підходів для автоматичного виправлення виявленних вразливостей.

Розглянемо основні пункти які перевіряються при скануванні утилітою Docker Bench for Security. Нас будуть цікавити пункти 1, 2, 3 та 4, оскільки 5, 6, 7, або не можуть бути автоматизовані, або застосовуються при дуже специфічних налаштуваннях Docker, наприклад при використанні Docker Swarm.

Надалі у демонстрації коду будуть використовуватись деякі допоміжні функції, тут буде коротка інформація щодо них для кращого розуміння написаного коду.

1. run_command_as_sudo(command) – дана команда виконує будь-яку команду як root користувач та повертає вивід який ця команда видає.
2. apt_install(pkgs) –дана команда дозволяє встановити будь-які пакети, які будуть передані як параметр(у вигляді масиву).
3. open_file(file, text, parameter) – дана функція дозволяє відкрити будь-який файл у режимі «r», «w», «a» чи у власностворенних режимах які були запрограмовані під певні цілі.
4. checking_file_exists(file_path) – перевіряє чи існує файл у системі хосту та повертає True або False в залежності від результату.

3.1. Конфігурація хоста

При встановленні Docker на новий хост створюється велика кількість нових ресурсів. Ці ресурси можна поділити на три основних групи:

1. Конфігураційні файли.
2. Окремий користувач та група Docker.
3. Бінарні файли які використовуються для різноманітного функціоналу Docker.

Цей пункт сканування аналізує дані ресурси і подає інформацію щодо рекомендацій для забезпечення їх безпеки.

Я виокремив 3 пункти автоматизації які забезпечують коректне налаштування безпеки для цих ресурсів.

3.1.1. Оновлення Docker Engine до останньої версії.

Будь-яке програмне забезпечення повинне бути завжди останньої версії, оскільки розробники можуть додавати новий функціонал, закривати проломи у безпеці і т.д..

Саме тому першою дією моєї автоматизації є перевірка на наявність оновлень Docker та встановлення їх якщо вони присутні. Для цього використовується функція docker_update().

```
def docker_update():
    out = run_command_as_sudo("rm /etc/docker/daemon.json")
    print(out)
    restart_docker()
    out = run_command_as_sudo("apt-get update")
    print(out)
    apt_install(["apt", "install", "-y", "docker-ce", "docker-ce-cli", "containerd.io", "docker-compose-plugin"])
```

Ця функція видаляє старий конфігураційний файл Docker та відправляє запит «apt-get update», а потім встановлює останні версії, якщо у користувача встановлена остання версія, то автоматизація віддасть інформацію, що Docker вже останньої версії.

3.1.2. Налаштування аудіту для основних файлів Docker Engine.

Дуже важливим є правильне налаштування аудиту для важливих файлів які використовуються під час роботи з Docker, так завжди можно зрозуміти джерело помилки та атаки зловмисника.

Для аудіту був використаний пакет auditd, аудит встановлюється використовуючи функцію add_audit(), яку можно подивитись у додатку 6.

Ця функція додає необхідні правила аудиту для переліку файлів та перезапускає auditd для їх застосування.

3.1.3. Налаштування дозволів до використання Docker Engine.

Для правильного налаштування безпеки потрібно, щоб дозвіл до використання Docker мав тільки root користувач. Це допомогає запобігати несанкціонованого доступу до функціоналу Docker.

Була розроблена функція restrict_access_to_docker_group(), яка прибирає усіх користувачів, які мають доступ до групи docker та не були додані у список користувачів, які повинні мати доступ до групи.

```
def restrict_access_to_docker_group(users_to_ignore):
    out = run_command_as_sudo("members docker")
    out = out[0]
    if out is None:
        return "No users are in docker group"
    else:
        users = out.split()
        for user in users:
            if user not in users_to_ignore:
                run_command_as_sudo("gpasswd -d " + str(user) + " docker")
            else:
                print("User " + str(user) + " is set to ignore list")
    return "All users which are not in ignore list are deleted from group"
```

3.2. Конфігурація Docker daemon

Docker daemon є основним менеджмент ресурсом у Docker Engine, він займається менеджментом контейнерів, образів, плагінів і т.д.. Саме цьому конфігурація Docker daemon є дуже важливою складовою налаштування безпеки.

При запуску Docker daemon дивиться на дві речі - це параметри з якими він запускається та конфігураційні файли.

Важливо розуміти, що параметри фактично є частинами конфігураційних файлів, але надають можливість запустити Docker daemon з якоюсь особливою конфігурацією яка необхідна користувачу лише зараз. З цього випливає також те, що якщо якийсь параметр буде передаватись і у команді, і з конфігураційного файлу, то виникне конфлікт який не дозволить запуститись Docker daemon.

Був обраний підхід до налаштування через конфігураційні файли, бо це більш наглядно, а також дозволяє бути впевненим, що при наступному запуску daemon усі налаштування безпеки залишаться присутні.

Було поділено налаштування Docker daemon на 3 пункти:

3.2.1. Налаштування основних параметрів Docker daemon.

Основним конфігураційним файлом Docker є /etc/docker/daemon.json. Docker daemon має дуже велику кількість різних параметрів, тому у даному пункті буде описане встановлення основних параметрів, які не потребують окремих дій для їх правильної конфігурації.

Для цього була використана поміжна функція updating_daemon_configuration(dict_of_parameters)

```
def updating_daemon_configuration(dict_of_parameters):
    if not checking_file_exist("/etc/docker/daemon.json"):
        default_text = open_file("daemon_default.json", None, 'r')
        open_file("/etc/docker/daemon.json", default_text, 'w')
    daemon_json = open_file("/etc/docker/daemon.json", None, 'r')
    daemon_json = json.loads(daemon_json)
    if "disable-legacy-registry" in daemon_json:
        del daemon_json["disable-legacy-registry"]
    for parameter in dict_of_parameters.keys():
        if dict_of_parameters[parameter]["value"] == "True" or dict_of_parameters[parameter]["value"] == "False":
            dict_of_parameters[parameter]["value"] = bool(dict_of_parameters[parameter]["value"])
        daemon_json[parameter] = dict_of_parameters[parameter]["value"]
    daemon_json = json.dumps(daemon_json, indent=6)
    open_file("/etc/docker/daemon.json", daemon_json, 'w')
```

В дану функцію потрібно передати словник python з необхідними параметрами та їх значеннями.

Також функція перевіряє чи існує daemon.json у системі, бо на новій системі при встановленні Docker daemon.json буде відсутній та Docker буде запускатись з стандартними параметрами. Якщо daemon.json відсутній – функція створить його з стандартними значеннями які знаходяться у файлі daemon_default.json (додаток 10).

Також дуже важливим є момент з видаленням «disable-legacy-registry» параметром, тому що це параметер, який більше не використовується і може бути шкідливим для користувача.

Після всіх перевірок функція створює новий json файл з параметрами які потрібно було оновити.

3.2.2. Генерація TLS сертифікатів та встановлення параметрів для використання TLS з Docker.

Docker підтримує використання його функціоналу з мережі через HTTPS, тому важливим є встановлення TLS сертифікатів та їх використання.

Для прикладу на локальній системі були створені самопідписані сертифікати та дозволив tcp з’єднання з tcp://127.0.0.1:2376. Для цього автоматизація використовує три основних функції.

1. updating_daemon_configuration(dict_of_parameters) – Описана у минулому пункті функція, яка використовується для встановлення tls параметрів після налаштування.
2. certificates_creation.py(додаток 8) – це окремий python модуль, який був написаний для створення tls сертифікатів. Він створює сертифікати базуючись на config.yaml файлі який дінамічно створюється базуючись на параметраї які задав користувач. Також є можливість перестворення сертифікатів базуючись на старих приватних ключах та старому csr файлі, якщо такий наявний у користувача.
3. setting_tls_certificate() – функція яка безпосередньо відправляє виклики до описаних у попередніх двох пунктах функцій, а також правильно конфігурує побічні файли які використовуються при tls верифікації у Docker.

Результатом виконання цих функцій є створення cert.pem, key.pem, ca.pem та налаштування daemon для їх використання.

3.2.3. Встановлення плагіна для авторизації у Docker.

Ще одною важливою складовою конфігурації Docker daemon є плагіни для авторизації. Вони встановлюються окремо саме у системі Docker та потребують деяких налаштувань конфігураційних файлів для правильної роботи.

Був обраний плагін openpolicyagent/opa-docker-authz-v2:0.4. Для прикладу був налаштований так, щоб плагін читав з конфігураційного файлу Docker(config.json) встановлені заголовки, та базуючи на встановленому у них айді користувача(у моєму випадку це буде test_user), та базуючись на конфігураційному файлі плагіну вирішував чи буде користувач мати тільки Read-only доступ до Docker команд(наприклад, користувач не зможе запускати контейнери, якщо цей флаг встановлено).

Для цього автоматизація використовує функцію installing_authorization_plugin().

```
def installing_authorization_plugin():
    os.system('docker plugin install --grant-all-permissions openpolicyagent/opa-docker-authz-v2:0.4 '
              'opa-args="-policy-file /opa/policies/authz.rego"')
```

```
    if not checking_file_exist("/root/.docker"):
        out = run_command_as_sudo("mkdir /root/.docker")
        print(out)
    if checking_file_exist("/root/.docker/config.json"):
        config_json = open_file("/root/.docker/config.json", None, 'r')
        config_json = json.loads(config_json)
        config_json["HttpHeaders"] = {"Authz-User": "test_user"}
    else:
        config_json = dict()
        config_json["HttpHeaders"] = {"Authz-User": "test_user"}
    config_json = json.dumps(config_json, indent=6)
    open_file("/root/.docker/config.json", config_json, 'w')
    content = open_file("authzrego_default", None, 'r')
    if not checking_file_exist("/etc/docker/policies"):
        out = run_command_as_sudo("mkdir /etc/docker/policies")
        print(out)
    open_file("/etc/docker/policies/authz.rego", content, 'w')
    updating_daemon_configuration({"authorization-plugins": {"value": ["openpolicyagent/opa-docker-authz-
v2:0.4"]}})
    restart_docker()
```

Функція використовує конфігураційний файл authzrego_default (додаток 9) для встановлення конфігурації плагіну, а також додає до конфігураційного файлу Docker необхідні заголовки. Після чого перезапускає докер для застосування встановлених конфігурацій.

3.3. Конфігураційні файли Docker daemon

Як було згадано у попередніх пунктах Docker використовує багато конфігураційних файлів під час своєї роботи, цей пункт відповідає за правильне встановлення дозволів та власників на ці файли задля запобігання несанкціонованого доступу.

Для цього автоматизація використовує функцію setting_correct_owners_and_permissions(), яка базуючись на заданому словнику python, який має назви файлів та рекомендовані значення дозволів та власників для даних файлів, встановлює ці значення.

3.4. Образи контейнерів та файли для їх збірки

Основною метою використання Docker завжди був запуск контейнерів і саме тому безпека контейнерів є однією з найважливіших у безпеці Docker. Нажаль повністю автоматизувати створення безпечних контейнерів

неможливо, оскільки команди які буде запускати контейнер при створені може контролювати лише сам користувач.

Але були виокремлені два пункти які можуть бути автоматизовані та дуже сильно впливають на безпеку Docker контейнерів.

3.4.1. Оновлення DockerFile відповідно до вимог безпеки.

Кожен образ який використовує Docker будується базуючись на DockerFile, саме у DockerFile встановлюється перелік базових образів, команд, додатковий функціонал(наприклад HealthCheck) та інше. Розглянемо, що може бути оновлено у DockerFile для забезпечення кращої безпеки на прикладі знайденого у інтернеті bad-dockerfile[12]. Даний DockerFile має велику кількість вразливостей.

Для роботи з даним пунктом використовується функція build_container(file_path, update_image_check)

```
def build_container(file_path, update_image_check):
    out = run_command_as_sudo("docker build -t imiell/bad-dockerfile " + file_path)
    print(out)
    out = out[0]
    out = out.splitlines()
    needed_line = ""
    for line in out:
        if "Successfully built " in line:
            needed_line = line
    if needed_line == "":
        print("Image wasn't successfully build")
    elif update_image_check == 1:
        image = needed_line.split("Successfully built ")[1]
        update_images(str(image), file_path + "/Dockerfile")
        out = run_command_as_sudo("docker image rm -f " + str(image))
        print(out)
        out = run_command_as_sudo("docker build -t imiell/bad-dockerfile " + file_path)
        print("Image tags updated and image successfully built")
    else:
        print("Image successfully built")
    print(out)
```

Більш детально про використання цієї функції буде розказано далі.

Виділимо основні проблеми даного DockerFile, які спричиняють таку велику кількість вразливостей:

1. Застарілі версії базових образів.

DockerFile завжди використовує базові образи. Їх включення позначається командою FROM. Зазвичай це образи систем, наприклад у DockerFile який розбирається використовується centos:centos7.2.1511, який не є останньою версією, а тому має багато вразливостей, які у наступних версіях вже виправлені.

Для того, щоб автоматично виправити даний базовий образ на новий у автоматизації використовується функція update_image().

```
def update_images(image, file_path):
    checked_image = image
    checked_image_dockerfile = file_path
    out = run_command_as_sudo("docker image history {}".format(checked_image))
    images_list = list()
    out = out[0]
    out = out.splitlines()
    for line in out:
        if line is not None:
            print(line)
            line_index = line.split()
            images_list.append(line_index[0])
    out = run_command_as_sudo("docker image ls")
    out = out[0]
    images_dict = dict()
    out = out.splitlines()
    for line in out:
        if line is not None:
            print(line)
            line_index = line.split()
            images_dict[line_index[2]] = line_index[0]
    needed_to_check_image = list()
    for image in images_list:
        if image in images_dict.keys() and image != "IMAGE" and image != checked_image:
            needed_to_check_image.append(images_dict[image])
    dockerfile = open_file(checked_image_dockerfile, None, 'readlines')
    for line in dockerfile:
        if "FROM" in line:
            for image in needed_to_check_image:
                if image in line:
                    index = dockerfile.index(line)
                    line = line.split(":")
```

```
            line = line[0] + ":latest\n"
            dockerfile[index] = line
dockerfile = "".join(dockerfile)
open_file(checked_image_dockerfile, dockerfile, 'w')
```

Задля того, щоб дана функція працювала, як заплановано, потрібно, щоб у Docker вже був образ побудований на застарілому базовому образі, оскільки функція дивиться інформацію цього образу та співставляє використанні при його побудові образи та образи які встановленні у Docker системі користувача, тим самим виокремлюючи базові образи та їх назви. Отримавши їх назви функція замінить у DockerFile тег базового образу на «latest»(це тег який використовується для найновіших версій). Тепер після побудови образу базуючись на оновленому DockerFile – буде використана остання версія базових образів.

Через те, що не завжди у користувача який буде використовувати автоматизацію буде вже побудований образ, була додана побудова образу у функціонал автоматизації(функція build_container описана вище). Нажаль цей функціонал погано працює на DockerFile, який використовується як приклад, через те, що образ цього DockerFile не завжди будується коректно через велику кількість запитів на сторонні ресурси через curl та wget, які іноді відбиваються timeout через що уся робота автоматизації зупиняється. Тому при роботі з DockerFile який використовується для прикладу цей параметр був переданий як стала.

2. Відсутність окремого користувача для контейнеру.

Якщо у DockerFile не вказаний певний користувач, то контейнер буде запущений від імені root користувача. Це не дуже гарна практика, оскільки веде до великої кількості атак, наприклад «втеча із контейнера». Тому найкращою практикою є створення окремих користувачів для контейнерів лише з необхідними правами. [3]
Для цього у моїй автоматизації використовується функція add_container_user().

```
def add_container_user(file_path):
    command = "RUN useradd -d /home/base_user -m -s /bin/bash base_user\nUSER base_user\n"
    dockerfile = open_file(file_path, None, 'readlines')
    index = None
    for line in dockerfile:
        if "CMD" in line:
            index = dockerfile.index(line) - 2
    if index is not None:
        dockerfile.insert(index + 1, command)
        dockerfile = "".join(dockerfile)
        open_file(file_path, dockerfile, 'w')
    else:
        print("Your dockerfile is invalid. It don't have CMD command.")
```

Ця функція додає у цільовий DockerFile необхідні команди для створення непривілейованого користувача.

3. Використання ADD команд замість COPY.

Команди ADD та COPY мають однакову мету використання – копіювання файлів та директорій у контейнер, але ADD має більше можливостей, які можуть бути використані зловмисниками, зокрема це використання джерелом файлів і контейнерів посилань. COPY в цьому випадку є значно безпечнішою командою, бо вона може копіювати лише локальні файли, що значно зменшує ризик завантаження небажаного програмного забезпечення.

У автоматизації була використана функцію use_copy_instead_add(file_path).

```
def use_copy_instead_add(file_path):
    dockerfile = open_file(file_path, None, 'readlines')
    for line in dockerfile:
        if "ADD" in line:
            index = dockerfile.index(line)
            line_array = line.split()
            line_array[0] = "COPY"
            line = " ".join(line_array) + "\n"
            dockerfile[index] = line
    dockerfile = "".join(dockerfile)
    open_file(file_path, dockerfile, "w")
```

Ця функція перевіряє DockerFile на наявність використання команди ADD та замінює їх на COPY.

Користувачу треба бути обережним з цим функціоналом оскільки він може замінити ADD команду там де це не бажано користувачем.

4. Відсутність Healthcheck функціоналу.

HealthCheck функціонал важлива частина будь-якого Docker контейнеру, оскільки вона забезпечує чіткий контроль за доступністю контейнеру.

Для автоматичного встановлення HealthCheck функціоналу використовується функція healthcheck_add(file, interval, timeout), яка додає необхідні команди до цільового DockerFile базуючись на параметрах заданих користувачем.

```
def healthcheck_add(file, interval, timeout):
    command = "curl -f http://localhost/"
    healthcheck = "HEALTHCHECK --interval={}s --timeout={}s CMD {}\n".format(interval, timeout, command)
    dockerfile = open_file(file, None, 'readlines')
    index = None
    for line in dockerfile:
        if "FROM" in line:
            index = dockerfile.index(line)
            break
    if index is not None:
        dockerfile.insert(index + 1, healthcheck)
        dockerfile = "".join(dockerfile)
        open_file(file, dockerfile, 'w')
    else:
        print("Your dockerfile is invalid. It don't have FROM command.")
```

3.4.2. Docker Content Trust

Для підпису образів використовується Docker Content Trust технологія. Docker Content Trust використовує два різних ключа. Перший – це ключ тегів. Ключ генерується для кожного нового сховища, яке публікує користувач. Це ключі які можна передавати іншим та експортувати тим, кому необхідна можливість підписувати вміст від імені реєстру. Другий ключ – це автономний ключ. Він повинен надійно зберігатись і не повинен бути розповсюджений. Цей ключ використовується для створення ключів тегів. [6][8]

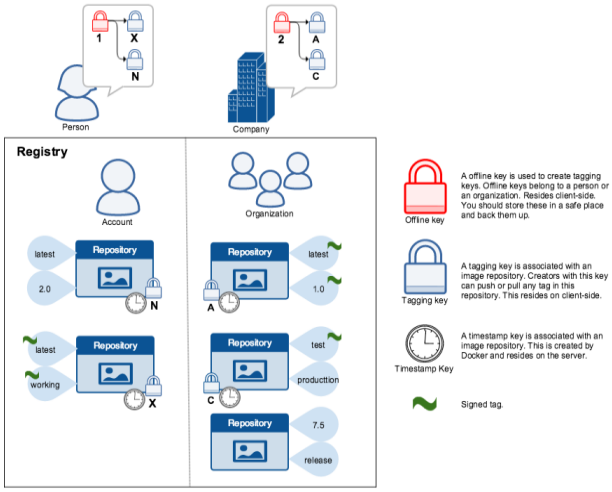

рис. 3.1 [8]

Даний метод дозволяє захиститися від трьох ключових сценаріїв[6]:

1. Захист від підробки образів. Якщо зловмисник буде намагатися, що зловмисний образ це ваш образ, то він не зможе підписати образ із тим самим автономним ключем репозиторія, оскільки при підписі нашого образу використовується наш автономний ключ, який відомий тільки нашому користувачу.

2. Захист від атак повторного відтворення. Атаки повторного відтворення – це атаки при використанні яких зловмисник намагається видати стару версію нашого образу, яка наприклад має вразливості, за останню версію образу. Це буде неможливо через використання часових міток у Docker Content Trust технології.

3. Захист від компрометації ключів. Якщо ваш тег ключ був скомпрометований, то ви завжди можете запустити ротацію ключів використовуючи ваш автономний ключ. Така ротація може бути запущена лише власником вихідного автономного ключа.

Автоматизація включає Docker Content Trust додаючи необхідний параметр у змінні середовища. Після цього Docker Content Trust буде включений при будь-якому наступному запуску Docker. Для цього використовується функція enabling_docker_trust().

```
def enabling_docker_trust():
    variable = "DOCKER_CONTENT_TRUST=1\n"
    open_file("/etc/environment", variable, "a")
```

3.5. Огляд результатів отриманих після використання розробленого програмного забезпечення.

Після виконання розробленого програмного забезпечення було проведено повторне сканування результати якого можно побачити у додатках 1, 2, 3, 4, 5.

Сумарний результат значно виріс порівняно з початковими значеннями.

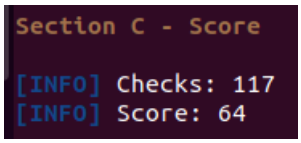

рис. 3.2

Нажаль не є можливим одночасно показати правильне виконання функціоналу пункту 3.4.1 та функціоналу пункту 3.4.2 через те що оглядаємий у 3.4.1 DockerFile не може бути підписаний. Тому у додатках 4 та 5 можно побачити два різних виводи в залежності від того, який з пунктів 3.4.1 та 3.4.2 виконувався.

3.6. Розробка Graphical User Interface.

Після розробки програмного забезпечення були закриті усі задачі які були поставлені щодо автоматизації налаштування безпеки Docker. Але залишалась не менш важлива проблема.

Оскільки основною метою дипломної роботи була розробка універсального програмного забезпечення для автоматичного налаштування базової безпеки Docker, яке може використовуватись як для персонального використання так і для великих корпорацій. Але розроблена мною програма не була простою для розуміння непідготовленим користувачем, а підходила для вже досвідченого користувача Docker, який розбирається у основних параметрах безпеки, а також може зрозуміти яку структуру має програмне забезпечення та до чого воно звертається.

Для вирішення цієї проблеми було прийняте рішення розробити графічний інтерфейс(рисунок 3.3) у якому користувач може більш щільно взаємодіяти з програмним забезпеченням, використовувати лише необхідний функціонал, а також змінювати параметри за власним розсудом.

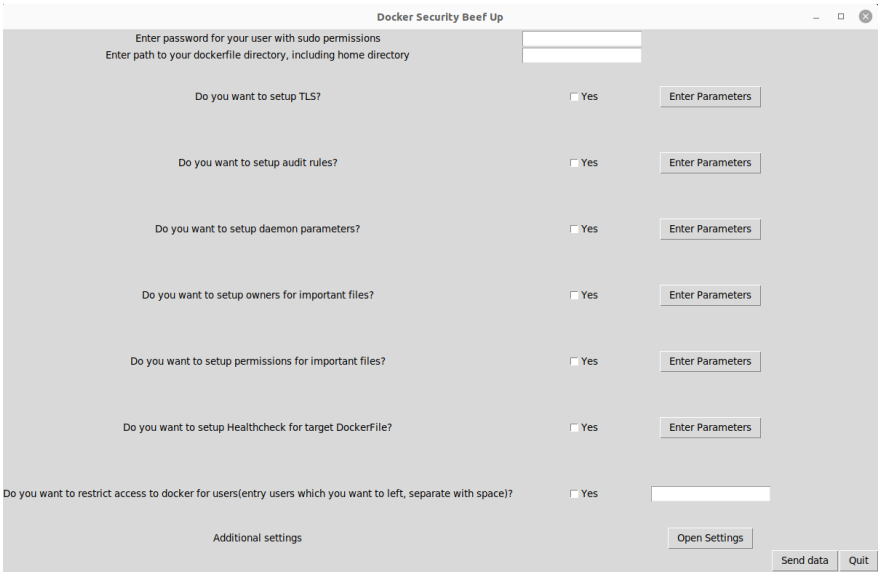

Рисунок 3.3

Для впровадження поставлених задач мною було виокремлено три проблеми:

1. Донесення інформації до користувача без необхідності додаткового пошуку інформації з його сторони.

2. Впровадження системи видалення параметрів з конфігурації яка буде застосована, або зміна їх на відмінні від стандартних значення.

3. Введення системи перевірок, яка б забезпечила захист системи користувача від данних, які можуть бути введенні помилково і в результаті викликати помилки у цільовій системі.

3.6.1. Введення функціоналу підказок.

Для вирішення першої проблеми було вирішено розробити систему підказок для користувачів для швидкого отримання надійної інформації. Був розроблений окремий клас ToolTip.

```
class ToolTip(object):

  def __init__(self, widget):
    self.widget = widget
    self.tool_tip_window = None
    self.id = None
    self.x_axis = self.y_axis = 0

  def show_tool_tip(self, text):
    self.text = text
    if self.tool_tip_window or not self.text:
      return
    x_axis, y_axis, cx_axis, cy_axis = self.widget.bbox("insert")
```

```
        x_axis = x_axis + self.widget.winfo_rootx() + 56
        y_axis = y_axis + cy_axis + self.widget.winfo_rooty() + 26
        self.tool_tip_window = tk.Toplevel(self.widget)
        self.tool_tip_window.wm_overrideredirect(1)
        self.tool_tip_window.wm_geometry("+%d+%d" % (x_axis, y_axis))
        label = tk.Label(self.tool_tip_window, text=self.text, justify="left", background="#ffffe0", relief="solid",
borderwidth=1, font=("tahoma", "8", "normal"))
        label.pack(ipadx=1)

    def hide_tool_tip(self):
        tool_tip_window = self.tool_tip_window
        self.tool_tip_window = None
        if tool_tip_window:
            tool_tip_window.destroy()

def tool_tip_create(widget, text):
    tool_tip = ToolTip(widget)

    def enter(event):
        tool_tip.show_tool_tip(text)

    def leave(event):
        tool_tip.hide_tool_tip()

    widget.bind('<Enter>', enter)
    widget.bind('<Leave>', leave)
```

Цей клас використовується для створення підказок для кожного функціоналу, а також для параметрів які використовуються у програмному забезпеченні, базуючись на тексті який вказаний у default_dicts.json(додаток 11). Завдяки цьому користувач завжди може отримати актуальну інформацію щодо функціоналу та параметрів при наведенні на назву курсором комп’ютерної миші. Приклад роботи підказок зображений на рисунку 3.4.

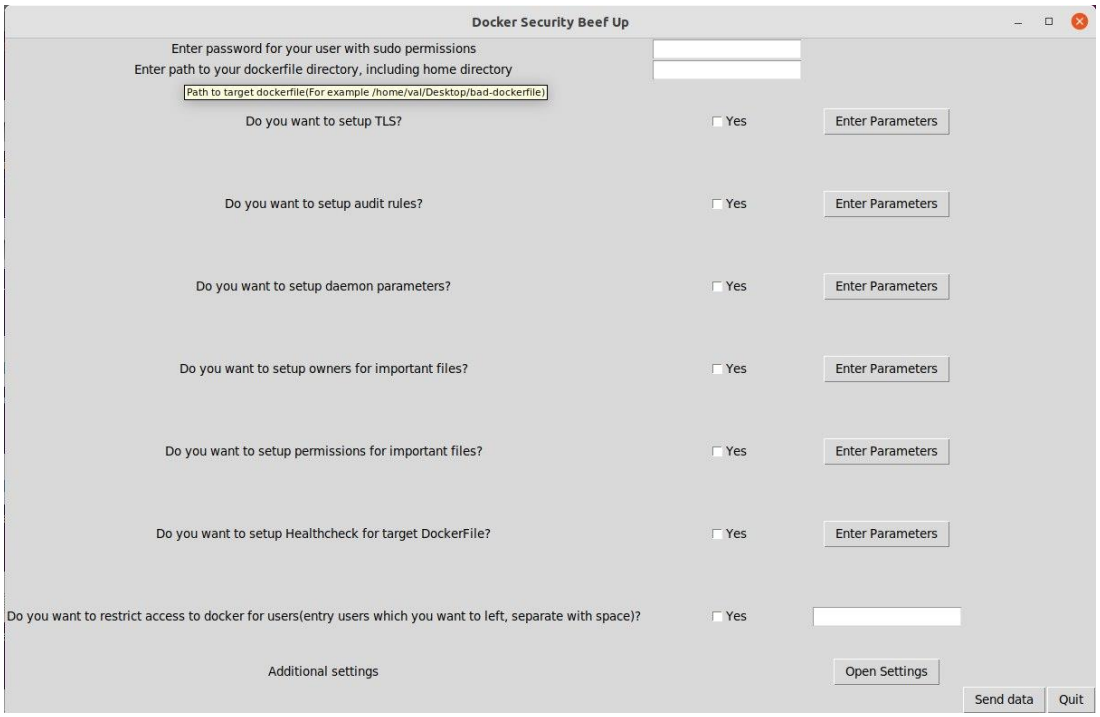

Рисунок 3.4

3.6.2. Модифікація параметрів програмного забезпечення.

Для вирішення другої проблеми було вирішено впровадити можливість власноруч задавати параметри для кожного функціоналу де вони необхідні. Були додані окремі вікна з переліком усіх параметрих для кожного функціоналу у яких користувач може прочитати інформацію щодо параметрів використовуючи описану у пункті 3.6.1 систему підказок, а також задати власні значення або виключити бажані параметри(використовуючи ключові слова). Якщо користувач не бажає змінювати значення параметрів, то будуть використані рекомендовані значення відповідно до CIS Docker Benchmark v1.4.0[7], які записані у default_dicts.json(додаток 11). Приклад впровадженого функціоналу зображений на рисунку 3.5.

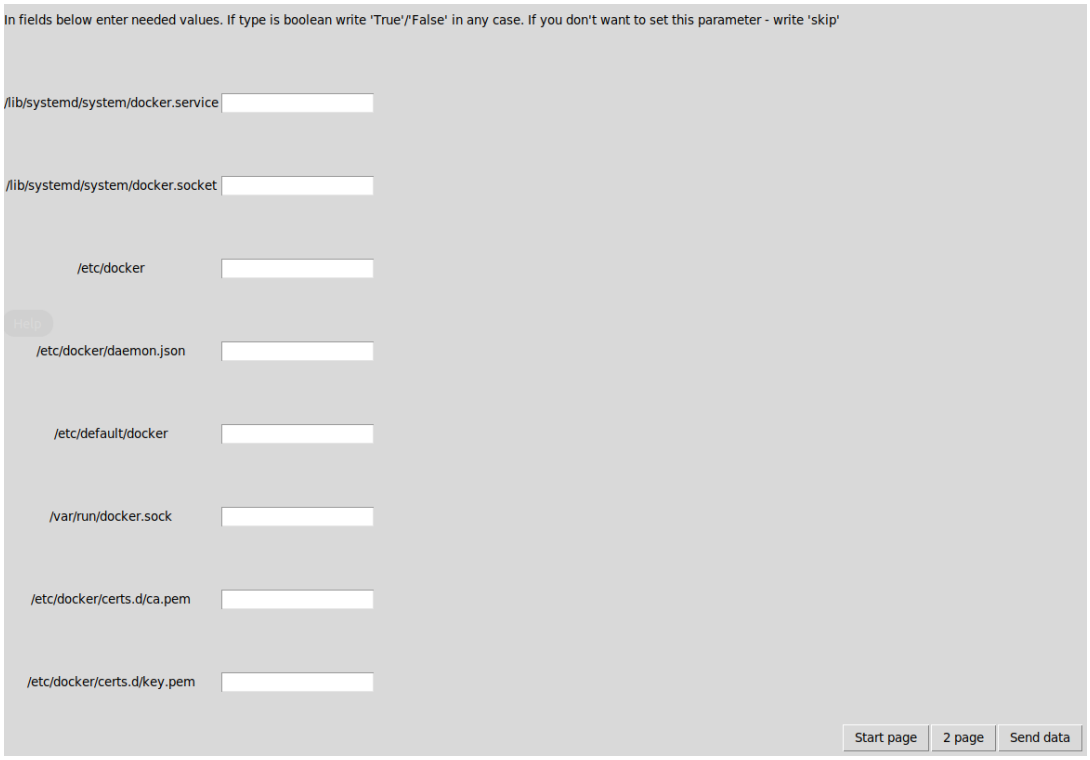

Рисунок 3.5

3.6.3. Система захисту від небажаних змін.

Було прийнято рішення розробити систему валідації оскільки розроблена система є комплексною і багато параметрів та функціоналу залежні одне від одного. Для цього була впроваджена система перевірок, яка відключає деякий функціонал якщо параметри не відповідають вимогам. Наприклад, якщо користувач не вказує шлях до цільового DockerFile, то функціонал описаний у пункті 3.4.1 не буде виконуватись.

Висновки до розділу 3

В цьому розділі було розроблено автоматичне програмне забезпечення для налаштування безпеки Docker.

Функціональність програмного забезпечення показала чудові результати піднявши загальну оцінку безпеки Docker у 8 раз(з 8 до 64 по оцінках скануючої утиліти Docker Bench for Security).

Для оцінки універсальності та функціональності розробленого програмного забезпечення, а також взаємодії користувача з ним, було проведено опитання декількох моїх одногрупників щодо їх оцінки на це програмне забезпечення. У всіх з них не було нарікань до нього і вони відмічали, що використання цього програмного забезпечення дуже просте навіть для користувачів, які не знайомі з методами та засобами безпеки Docker.

За результатами опитувань та власного тестування був зроблений висновок, що поставлені задачі були виконані повністю. Однак важливим є те, що данне програмне забезпечення може бути модифіковано додатковим функціоналом, а також інтегровано зі скануючими утилітами, що на мою думку є чудовим цілями на подальше розвинення проекту поза рамками дипломної роботи.

Висновки

У ході виконання дипломної роботи були розглянуті види віртуалізації. Була створена порівняльна характеристика двох найбільш використовуємих технологій віртуалізаціїї, яка показує переваги та недоліки цих технологій у порівнянні.

Було досліджено вплив вразливостей у контейнерній віртуалізації на їх використання, а також розглянуто декілька прикладів вразливостей, які гарно відображають збитки, що можуть бути наслідками використання цих вразливостей зловмисниками.

Було розглянуто найбільш відомі методи та засоби забезпечення безпеки Docker. Базуючись на цих методах і засобах було розроблено програмне забезпечення, яке значно пришвидшує встановлення та модифікацію налаштувань безпеки. Таким чином була вирішена проблема витрачання великого часу на налаштування безпеки на нових системах чи у великих інфраструктурах, а також проблема неправильного налаштування безпеки некомпетентними користувачами, які тільки почали працювати з Docker.

Був розроблений графічний інтерфейс, який дозволяє користувачам більш повно використовувати програмне забезпечення в залежності від поставлених цілей. Розроблене програмне забезпечення повноцінно реалізує поставлені задачі та мету дипломної роботи.

Базуючись на отриманих результатах дипломної роботи було прийнято рішення подати заявку в УКРПАТЕНТ на отримання авторського права на розроблене програмне забезпечення у співавторстві з науковим керівником. Заявка на данний момент знаходиться у процессі розгляду.

Джерела

1. Microsoft documentation. https://github.com/MicrosoftDocs/Virtualization-Documentation/blob/main/virtualization/windowscontainers/about/containers-vs-vm.md, 29 October 2021.
2. National Vulnerability Database, https://nvd.nist.gov/, Always updating.
3. Container Security by Liz Rice, O`Reilly Media, https://learning.oreilly.com/library/view/container-security/9781492056690/, April 2020.
4. Docker, AppArmor security profiles, https://docs.docker.com/engine/security/apparmor/.
5. GitHub, Docker Bench for Security, https://github.com/docker/docker-bench-security
6. Securing Docker by Scott Gallagher, Packt>, https://www.packtpub.com/product/securing-docker/9781785888854, 30 March 2016.
7. Benchmark for Docker, https://www.cisecurity.org/benchmark/docker
8. Docker, Content Trust Overview, https://docs.docker.com/engine/security/trust/.
9. Simplified, Creating Ubuntu chroot environment, https://www.simplified.guide/ubuntu/build-chroot-environment.
10. Docker, managing sensitive data with secrets, https://docs.docker.com/engine/swarm/secrets/ .
11. TechBeacon, 10+ top open-source tools for Docker security, https://techbeacon.com/security/10-top-open-source-tools-docker-security.
12. GitHub, Bad-Dockerfile, https://github.com/ianmiell/bad-dockerfile
13. Docker, Networking Overview, https://docs.docker.com/network/.
14. Red Hat Topic, https://www.redhat.com/en/topics/virtualization/what-is-network-virtualization, 23 March 2021.
15. GeekForGeek Article, https://www.geeksforgeeks.org/network-virtualization-in-cloud-computing/, 17 March 2021.

16. sdxCentral research, https://www.sdxcentral.com/networking/nfv/definitions/network-virtualization-and-how-it-works/, 22 July 2021.
17. Docker swarm: overlay network encryption, https://lovethepenguin.com/docker-swarm-overlay-network-encryption-and-mtls-5fba4ce3e266, 4 February 2021.
18. Dockerlabs, Docker Networking security Basics, https://dockerlabs.collabnix.com/advanced/security/networking/.

Додатки

1.

```
[INFO] 1 - Host Configuration
[INFO] 1.1 - Linux Hosts Specific Configuration
[WARN] 1.1.1 - Ensure a separate partition for containers has been created (Automated)
[INFO] 1.1.2 - Ensure only trusted users are allowed to control Docker daemon (Automated)
[INFO]       * Users:
[PASS] 1.1.3 - Ensure auditing is configured for the Docker daemon (Automated)
[PASS] 1.1.4 - Ensure auditing is configured for Docker files and directories -/run/containerd (Automated)
[PASS] 1.1.5 - Ensure auditing is configured for Docker files and directories - /var/lib/docker (Automated)
[PASS] 1.1.6 - Ensure auditing is configured for Docker files and directories - /etc/docker (Automated)
[PASS] 1.1.7 - Ensure auditing is configured for Docker files and directories - docker.service (Automated)
[INFO] 1.1.8 - Ensure auditing is configured for Docker files and directories - containerd.sock (Automated)
[INFO]       * File not found
[PASS] 1.1.9 - Ensure auditing is configured for Docker files and directories - docker.socket (Automated)
[PASS] 1.1.10 - Ensure auditing is configured for Docker files and directories - /etc/default/docker (Automated)
[PASS] 1.1.11 - Ensure auditing is configured for Dockerfiles and directories - /etc/docker/daemon.json (Automated)
[PASS] 1.1.12 - 1.1.12 Ensure auditing is configured for Dockerfiles and directories - /etc/containerd/config.toml (Automated)
[INFO] 1.1.13 - Ensure auditing is configured for Docker files and directories - /etc/sysconfig/docker (Automated)
[INFO]       * File not found
[PASS] 1.1.14 - Ensure auditing is configured for Docker files and directories - /usr/bin/containerd (Automated)
[PASS] 1.1.15 - Ensure auditing is configured for Docker files and directories - /usr/bin/containerd-shim (Automated)
[PASS] 1.1.16 - Ensure auditing is configured for Docker files and directories - /usr/bin/containerd-shim-runc-v1 (Automated)
[PASS] 1.1.17 - Ensure auditing is configured for Docker files and directories - /usr/bin/containerd-shim-runc-v2 (Automated)
[PASS] 1.1.18 - Ensure auditing is configured for Docker files and directories - /usr/bin/runc (Automated)
[INFO] 1.2 - General Configuration
[NOTE] 1.2.1 - Ensure the container host has been Hardened (Manual)
[PASS] 1.2.2 - Ensure that the version of Docker is up to date (Manual)
[INFO]       * Using 20.10.16, verify is it up to date as deemed necessary
```

2.

```
[INFO] 2 - Docker daemon configuration
[NOTE] 2.1 - Run the Docker daemon as a non-root user, if possible (Manual)
[PASS] 2.2 - Ensure network traffic is restricted between containers on the default bridge (Scored)
[PASS] 2.3 - Ensure the logging level is set to 'info' (Scored)
[PASS] 2.4 - Ensure Docker is allowed to make changes to iptables (Scored)
[PASS] 2.5 - Ensure insecure registries are not used (Scored)
[PASS] 2.6 - Ensure aufs storage driver is not used (Scored)
[PASS] 2.7 - Ensure TLS authentication for Docker daemon is configured (Scored)
[PASS] 2.8 - Ensure the default ulimit is configured appropriately (Manual)
[WARN] 2.9 - Enable user namespace support (Scored)
[PASS] 2.10 - Ensure the default cgroup usage has been confirmed (Scored)
[PASS] 2.11 - Ensure base device size is not changed until needed (Scored)
[
[PASS] 2.12 - Ensure that authorization for Docker client commands is enabled (Scored)
[PASS] 2.13 - Ensure centralized and remote logging is configured (Scored)
[PASS] 2.14 - Ensure containers are restricted from acquiring new privileges (Scored)
[PASS] 2.15 - Ensure live restore is enabled (Scored)
[PASS] 2.16 - Ensure Userland Proxy is Disabled (Scored)
[PASS] 2.17 - Ensure that a daemon-wide custom seccomp profile is applied if appropriate (Manual)
[INFO] Ensure that experimental features are not implemented in production (Scored) (Deprecated)
```

3.

```
[INFO] 3 - Docker daemon configuration files
[PASS] 3.1 - Ensure that the docker.service file ownership is set to root:root (Automated)
[PASS] 3.2 - Ensure that docker.service file permissions are appropriately set (Automated)
[PASS] 3.3 - Ensure that docker.socket file ownership is set to root:root (Automated)
[PASS] 3.4 - Ensure that docker.socket file permissions are set to 644 or more restrictive (Automated)
[PASS] 3.5 - Ensure that the /etc/docker directory ownership is set to root:root (Automated)
[PASS] 3.6 - Ensure that /etc/docker directory permissions are set to 755 or more restrictively (Automated)
[PASS] 3.7 - Ensure that registry certificate file ownership is set to root:root (Automated)
[PASS] 3.8 - Ensure that registry certificate file permissions are set to 444 or more restrictively (Automated)
[PASS] 3.9 - Ensure that TLS CA certificate file ownership is set to root:root (Automated)
[PASS] 3.10 - Ensure that TLS CA certificate file permissions are set to 444 or more restrictively (Automated)
[PASS] 3.11 - Ensure that Docker server certificate file ownership is set to root:root (Automated)
[PASS] 3.12 - Ensure that the Docker server certificate file permissions are set to 444 or more restrictively (Automated)
[PASS] 3.13 - Ensure that the Docker server certificate key file ownership is set to root:root (Automated)
[PASS] 3.14 - Ensure that the Docker server certificate key file permissions are set to 400 (Automated)
[PASS] 3.15 - Ensure that the Docker socket file ownership is set to root:docker (Automated)
[PASS] 3.16 - Ensure that the Docker socket file permissions are set to 660 or more restrictively (Automated)
[PASS] 3.17 - Ensure that the daemon.json file ownership is set to root:root (Automated)
[PASS] 3.18 - Ensure that daemon.json file permissions are set to 644 or more restrictive (Automated)
[PASS] 3.19 - Ensure that the /etc/default/docker file ownership is set to root:root (Automated)
[INFO] 3.20 - Ensure that the /etc/sysconfig/docker file permissions are set to 644 or more restrictively (Automated)
[INFO]       * File not found
[INFO] 3.21 - Ensure that the /etc/sysconfig/docker file ownership is set to root:root (Automated)
[INFO]       * File not found
[PASS] 3.22 - Ensure that the /etc/default/docker file permissions are set to 644 or more restrictively (Automated)
[PASS] 3.23 - Ensure that the Containerd socket file ownership is set to root:root (Automated)
[PASS] 3.24 - Ensure that the Containerd socket file permissions are set to 660 or more restrictively (Automated)
```

4.

```
[INFO] 4 - Container Images and Build File
[PASS] 4.1 - Ensure that a user for the container has been created (Automated)
[NOTE] 4.2 - Ensure that containers use only trusted base images (Manual)
[NOTE] 4.3 - Ensure that unnecessary packages are not installed in the container (Manual)
[NOTE] 4.4 - Ensure images are scanned and rebuilt to include security patches (Manual)
[WARN] 4.5 - Ensure Content trust for Docker is Enabled (Automated)
[WARN] 4.6 - Ensure that HEALTHCHECK instructions have been added to container images (Automated)
[WARN]      * No Healthcheck found: a4f14816ef46
[WARN]      * No Healthcheck found: [centos:centos7.2.1511]
[PASS] 4.7 - Ensure update instructions are not used alone in the Dockerfile (Manual)
[NOTE] 4.8 - Ensure setuid and setgid permissions are removed (Manual)
[INFO] 4.9 - Ensure that COPY is used instead of ADD in Dockerfiles (Manual)
[INFO]      * ADD in image history: [imiell/bad-dockerfile:latest]
[INFO]      * ADD in image history: [centos:centos7.2.1511]
[NOTE] 4.10 - Ensure secrets are not stored in Dockerfiles (Manual)
[NOTE] 4.11 - Ensure only verified packages are installed (Manual)
[NOTE] 4.12 - Ensure all signed artifacts are validated (Manual)
```

5.

```
[INFO] 4 - Container Images and Build File
[INFO] 4.1 - Ensure that a user for the container has been created (Automated)
[INFO]      * No containers running
[NOTE] 4.2 - Ensure that containers use only trusted base images (Manual)
[NOTE] 4.3 - Ensure that unnecessary packages are not installed in the container (Manual)
[NOTE] 4.4 - Ensure images are scanned and rebuilt to include security patches (Manual)
[PASS] 4.5 - Ensure Content trust for Docker is Enabled (Automated)
[WARN] 4.6 - Ensure that HEALTHCHECK instructions have been added to container images (Automated)
[WARN]      * No Healthcheck found: a4f14816ef46
[WARN]      * No Healthcheck found: [centos:centos7.2.1511]
[PASS] 4.7 - Ensure update instructions are not used alone in the Dockerfile (Manual)
[NOTE] 4.8 - Ensure setuid and setgid permissions are removed (Manual)
[INFO] 4.9 - Ensure that COPY is used instead of ADD in Dockerfiles (Manual)
[INFO]      * ADD in image history: [imiell/bad-dockerfile:latest]
[INFO]      * ADD in image history: [centos:centos7.2.1511]
[NOTE] 4.10 - Ensure secrets are not stored in Dockerfiles (Manual)
[NOTE] 4.11 - Ensure only verified packages are installed (Manual)
[NOTE] 4.12 - Ensure all signed artifacts are validated (Manual)
```

6. Main.py

```
import subprocess
import os
import os.path
import json
from gui import SampleApp
from certificates_creation import Secure


sudo_password = ""



def run_command_as_sudo(command):
    command = command.split()
    process = subprocess.Popen(['sudo', '-S'] + command, stdin=subprocess.PIPE, stdout=subprocess.PIPE,
                               universal_newlines=True)
    global sudo_password
    out = process.communicate(str(sudo_password) + "\n")
    if process.returncode is None:
        process.kill()
```

```
    return out


def apt_install(pkgs):
    subprocess.run(["sudo", "apt", "install", "-y"] + pkgs, check=True)


def docker_update():
    out = run_command_as_sudo("rm /etc/docker/daemon.json")
    print(out)
    restart_docker()
    out = run_command_as_sudo("apt-get update")
    print(out)
    apt_install(["docker-ce", "docker-ce-cli", "containerd.io", "docker-compose-plugin"])


def open_file(file, text, parameter):
    if parameter == 'a' or parameter == 'w':
        file_object = open(file, parameter)
        file_object.write(text)
        file_object.close()
    elif parameter == 'r':
        file_object = open(file, parameter)
        text_from_file = file_object.read()
        file_object.close()
        return text_from_file
    elif parameter == 'readlines':
        file_object = open(file, 'r')
        content = file_object.readlines()
        file_object.close()
        return content


def checking_file_exist(path):
    file_exists = os.path.exists(path)
    return file_exists


def add_audit(files_dict):
    apt_install(["auditd", "audispd-plugins"])
    audit_list = list()
    commands_dict = {"-w {} -p wa\n": ["/usr/bin/docker", "/var/lib/docker", "/etc/docker",
"/etc/default/docker",
```

```
                                "/etc/docker/daemon.json", "/usr/bin/docker-containerd", "/usr/bin/docker-runc",
                                "/etc/containerd/config.toml", "/run/containerd"],
                    "-w {} -k docker\n": ["/lib/systemd/system/docker.service", "/usr/bin/dockerd",
                                          "/usr/bin/containerd", "/usr/bin/containerd-shim",
                                          "/usr/bin/containerd-shim-runc-v1", "/usr/bin/containerd-shim-runc-v2",
                                          "/usr/bin/runc", "/lib/systemd/system/docker.socket"]}
    for command in commands_dict.keys():
        for file in files_dict.keys():
            if files_dict[file]["value"] == 1 and file in commands_dict[command]:
                audit_list.append(command.format(str(file)))
    for text in audit_list:
        open_file("/etc/audit/rules.d/audit.rules", text, 'a')
    out = run_command_as_sudo("systemctl restart auditd")
    print(out)


def restrict_access_to_docker_group(users_to_ignore):
    out = run_command_as_sudo("members docker")
    out = out[0]
    if out is None:
        return "No users are in docker group"
    else:
        users = out.split()
        for user in users:
            if user not in users_to_ignore:
                run_command_as_sudo("gpasswd -d " + str(user) + " docker")
            else:
                print("User " + str(user) + " is set to ignore list")
    return "All users which are not in ignore list are deleted from group"


def updating_daemon_configuration(dict_of_parameters):
    if not checking_file_exist("/etc/docker/daemon.json"):
        default_text = open_file("daemon_default.json", None, 'r')
        open_file("/etc/docker/daemon.json", default_text, 'w')
    daemon_json = open_file("/etc/docker/daemon.json", None, 'r')
    daemon_json = json.loads(daemon_json)
    if "disable-legacy-registry" in daemon_json:
        del daemon_json["disable-legacy-registry"]
    for parameter in dict_of_parameters.keys():
        if dict_of_parameters[parameter]["value"] == "True" or dict_of_parameters[parameter]["value"] ==
"False":
            dict_of_parameters[parameter]["value"] = bool(dict_of_parameters[parameter]["value"])
```

```
        daemon_json[parameter] = dict_of_parameters[parameter]["value"]
    daemon_json = json.dumps(daemon_json, indent=6)
    open_file("/etc/docker/daemon.json", daemon_json, 'w')


def chmod_chown_run(target, tag, data):
    command = ""
    if tag == "owners":
        command = "chown"
    elif tag == "permissions":
        command = "chmod"
    if command == "chown" or command == "chmod":
        out = run_command_as_sudo(str(command) + " -R " + str(data) + " " + str(target))
        return out
    else:
        return KeyError


def setting_correct_owners_and_permissions(targets_dict, tag):
    for target in targets_dict.keys():
        out = chmod_chown_run(target, tag, targets_dict[target]["value"])
        print(out)


def enabling_docker_trust():
    variable = "DOCKER_CONTENT_TRUST=1\n"
    open_file("/etc/environment", variable, "a")


def healthcheck_add(file, interval, timeout):
    command = "curl -f http://localhost/"
    healthcheck = "HEALTHCHECK --interval={}s --timeout={}s CMD {}\n".format(interval, timeout,
command)
    dockerfile = open_file(file, None, 'readlines')
    index = None
    for line in dockerfile:
        if "FROM" in line:
            index = dockerfile.index(line)
            break
    if index is not None:
        dockerfile.insert(index + 1, healthcheck)
        dockerfile = "".join(dockerfile)
        open_file(file, dockerfile, 'w')
```

```
    else:
        print("Your dockerfile is invalid. It don't have FROM command.")


def update_images(image, file_path):
    checked_image = image
    checked_image_dockerfile = file_path
    out = run_command_as_sudo("docker image history {}".format(checked_image))
    images_list = list()
    out = out[0]
    out = out.splitlines()
    for line in out:
        if line is not None:
            print(line)
            line_index = line.split()
            images_list.append(line_index[0])
    out = run_command_as_sudo("docker image ls")
    out = out[0]
    images_dict = dict()
    out = out.splitlines()
    for line in out:
        if line is not None:
            print(line)
            line_index = line.split()
            images_dict[line_index[2]] = line_index[0]
    needed_to_check_image = list()
    for image in images_list:
        if image in images_dict.keys() and image != "IMAGE" and image != checked_image:
            needed_to_check_image.append(images_dict[image])
    dockerfile = open_file(checked_image_dockerfile, None, 'readlines')
    for line in dockerfile:
        if "FROM" in line:
            for image in needed_to_check_image:
                if image in line:
                    index = dockerfile.index(line)
                    line = line.split(":")
                    line = line[0] + ":latest\n"
                    dockerfile[index] = line
    dockerfile = "".join(dockerfile)
    open_file(checked_image_dockerfile, dockerfile, 'w')


def add_container_user(file_path):
```

```
    command = "RUN useradd -d /home/base_user -m -s /bin/bash base_user\nUSER base_user\n"
    dockerfile = open_file(file_path, None, 'readlines')
    index = None
    for line in dockerfile:
        if "CMD" in line:
            index = dockerfile.index(line) - 2
    if index is not None:
        dockerfile.insert(index + 1, command)
        dockerfile = "".join(dockerfile)
        open_file(file_path, dockerfile, 'w')
    else:
        print("Your dockerfile is invalid. It don't have CMD command.")


def use_copy_instead_add(file_path):
    dockerfile = open_file(file_path, None, 'readlines')
    for line in dockerfile:
        if "ADD" in line:
            index = dockerfile.index(line)
            line_array = line.split()
            line_array[0] = "COPY"
            line = " ".join(line_array) + "\n"
            dockerfile[index] = line
    dockerfile = "".join(dockerfile)
    open_file(file_path, dockerfile, "w")


def setting_tls_certificate():
    if not checking_file_exist("/etc/docker/certs.d"):
        out = run_command_as_sudo("mkdir /etc/docker/certs.d")
        print(out)
    x = Secure("config.yaml")
    dict_of_parameters = dict()
    pem_key_file, pem_ca_certificate_file, pem_certificate_file = x.key_certificate_ca_dump()
    dict_of_parameters["tlscert"] = dict()
    dict_of_parameters["tlscert"]["value"] = pem_certificate_file
    dict_of_parameters["tlskey"] = dict()
    dict_of_parameters["tlskey"]["value"] = pem_key_file
    dict_of_parameters["tlscacert"] = dict()
    dict_of_parameters["tlscacert"]["value"] = pem_ca_certificate_file
    dict_of_parameters["tls"] = dict()
    dict_of_parameters["tls"]["value"] = True
    dict_of_parameters["tlsverify"] = dict()
```

```
    dict_of_parameters["tlsverify"]["value"] = True
    dict_of_parameters["hosts"] = dict()
    dict_of_parameters["hosts"]["value"] = ["unix:///var/run/docker.sock", "tcp://127.0.0.1:2376"]
    updating_daemon_configuration(dict_of_parameters)
    if not checking_file_exist("/etc/systemd/system/docker.service.d"):
        out = run_command_as_sudo("mkdir /etc/systemd/system/docker.service.d")
        print(out)
    text = """[Service]
ExecStart=
ExecStart=/usr/bin/dockerd"""
    if checking_file_exist("/etc/systemd/system/docker.service.d/hosts.conf"):
        content = open_file("/etc/systemd/system/docker.service.d/hosts.conf", None, 'r')
        if text not in content:
            open_file("/etc/systemd/system/docker.service.d/hosts.conf", text, 'a')
        else:
            print("Needed parameters already specified in configuration file")
    else:
        open_file("/etc/systemd/system/docker.service.d/hosts.conf", text, 'w')


def installing_authorization_plugin():
    os.system('docker plugin install --grant-all-permissions openpolicyagent/opa-docker-authz-v2:0.4 '
              'opa-args="-policy-file /opa/policies/authz.rego"')
    if not checking_file_exist("/root/.docker"):
        out = run_command_as_sudo("mkdir /root/.docker")
        print(out)
    if checking_file_exist("/root/.docker/config.json"):
        config_json = open_file("/root/.docker/config.json", None, 'r')
        config_json = json.loads(config_json)
        config_json["HttpHeaders"] = {"Authz-User": "test_user"}
    else:
        config_json = dict()
        config_json["HttpHeaders"] = {"Authz-User": "test_user"}
    config_json = json.dumps(config_json, indent=6)
    open_file("/root/.docker/config.json", config_json, 'w')
    content = open_file("authzrego_default", None, 'r')
    if not checking_file_exist("/etc/docker/policies"):
        out = run_command_as_sudo("mkdir /etc/docker/policies")
        print(out)
    open_file("/etc/docker/policies/authz.rego", content, 'w')
    updating_daemon_configuration({"authorization-plugins": {"value": ["openpolicyagent/opa-docker-authz-
v2:0.4"]}})
    restart_docker()
```

```
def restart_docker():
    out = run_command_as_sudo("systemctl daemon-reload")
    print(out)
    out = run_command_as_sudo("systemctl restart docker")
    print(out)


def build_container(file_path, update_image_check):
    out = run_command_as_sudo("docker build -t imiell/bad-dockerfile " + file_path)
    print(out)
    out = out[0]
    out = out.splitlines()
    needed_line = ""
    for line in out:
        if "Successfully built " in line:
            needed_line = line
    if needed_line == "":
        print("Image wasn't successfully build")
    elif update_image_check == 1:
        image = needed_line.split("Successfully built ")[1]
        update_images(str(image), file_path + "/Dockerfile")
        out = run_command_as_sudo("docker image rm -f " + str(image))
        print(out)
        out = run_command_as_sudo("docker build -t imiell/bad-dockerfile " + file_path)
        print("Image tags updated and image successfully built")
    else:
        print("Image successfully built")
    print(out)


def main():
    app = SampleApp()
    app.mainloop()
    file_path = str(app.dockerfile_path) + "/Dockerfile"
    global sudo_password
    sudo_password = str(app.sudo_password)
    if app.settings_dict["Update Docker"]["value"] == 1:
        print("Docker update started\n----------")
        docker_update()
        print("----------\nDocker update finished")
    if app.value_audit_check.get() == 1:
```

```
    print("Audit adding started\n----------")
    add_audit(app.audit_files_dict)
    print("----------\nAudit adding finished")
if app.settings_dict["Add separate USER to target DockerFile"]["value"] == 1:
    print("Adding separate user to Docker file started\n----------")
    add_container_user(file_path)
    print("----------\nAdding separate user to Docker file finished")
if app.settings_dict["Use COPY Command instead ADD in target DockerFile"] == 1:
    print("Changing all ADD commands on COPY started\n----------")
    use_copy_instead_add(file_path)
    print("----------\nChanging all ADD commands on COPY finished")
if app.value_healthcheck.get() == 1:
    print("Adding Healthcheck started\n----------")
    healthcheck_add(file_path, app.healthcheck_dict["interval"]["value"],
                    app.healthcheck_dict["timeout"]["value"])
    print("----------\nAdding Healthcheck finished")
if app.settings_dict["Build an image"]["value"] == 1:
    print("Building image started\n----------")
    build_container(file_path, app.settings_dict["Update base image to latest version in target DockerFile"])
    print("----------\nBuilding image finished")
if app.value_restrict_access.get() == 1:
    print("Restricting access for docker group started\n----------")
    restrict_access_to_docker_group(app.restrict_string.split())
    print("----------\nRestricting access for docker group finished")
if app.value_daemon_check.get() == 1:
    print("Updating daemon configuration started\n----------")
    updating_daemon_configuration(app.daemon_parameters_dict)
    print("----------\nUpdating daemon configuration finished")
if app.value_tcp_check.get() == 1:
    print("Setting TLS configuration started\n----------")
    setting_tls_certificate()
    print("----------\nSetting TLS configuration finished")
if app.value_owners_check.get() == 1:
    print("Setting owners for files started\n----------")
    setting_correct_owners_and_permissions(app.all_targets_owners_dict, "owners")
    print("----------\nSetting owners for files finished")
if app.value_permissions_check.get() == 1:
    print("Setting permissions for files started\n----------")
    setting_correct_owners_and_permissions(app.all_targets_permissions_dict, "permissions")
    print("----------\nSetting permissions for files finished")
if app.settings_dict["Install Authorization Plugin"]["value"] == 1:
    print("Installing authorization plugin started\n----------")
    installing_authorization_plugin()
```

```
        print("---------------------------------\nInstalling authorization plugin finished")
    if app.settings_dict["Enable Docker Trust"]["value"] == 1:
        print("Enabling Docker Trust started\n---------------------------------")
        enabling_docker_trust()
        print("---------------------------------\nEnabling Docker Trust finished")
    restart_docker()



if __name__ == "__main__":
    main()
```

7.gui.py

```
import tkinter as tk
import yaml
import json



class ToolTip(object):

    def __init__(self, widget):
        self.widget = widget
        self.tool_tip_window = None
        self.id = None
        self.x = self.y = 0

    def show_tool_tip(self, text):
        self.text = text
        if self.tool_tip_window or not self.text:
            return
        x, y, cx, cy = self.widget.bbox("insert")
        x = x + self.widget.winfo_rootx() + 57
        y = y + cy + self.widget.winfo_rooty() + 27
        self.tool_tip_window = tk.Toplevel(self.widget)
        self.tool_tip_window.wm_overrideredirect(1)
        self.tool_tip_window.wm_geometry("+%d+%d" % (x, y))
        label = tk.Label(self.tool_tip_window, text=self.text, justify="left", background="#ffffe0", relief="solid",
borderwidth=1,
                         font=("tahoma", "8", "normal"))
        label.pack(ipadx=1)

    def hide_tool_tip(self):
        tool_tip_window = self.tool_tip_window
        self.tool_tip_window = None
        if tool_tip_window:
```

```
            tool_tip_window.destroy()


def tool_tip_create(widget, text):
    tool_tip = ToolTip(widget)

    def enter(event):
        tool_tip.show_tool_tip(text)

    def leave(event):
        tool_tip.hide_tool_tip()

    widget.bind('<Enter>', enter)
    widget.bind('<Leave>', leave)


def set_variable_dictionary(values_dict, flag):
    sub_window = tk.Toplevel()
    sub_window.geometry("1900x1080")

    def get_dict():
        temporary_dict = {**values_dict}
        if flag == "checkbutton":
            for key in temporary_dict.keys():
                values_dict[key]["value"] = values_dict[key]["value"].get()
        elif flag == "entry":
            for key in temporary_dict.keys():
                if values_dict[key]["entry"].get() == "":
                    pass
                elif values_dict[key]["entry"].get().lower() == "false":
                    values_dict[key]["value"] = False
                elif values_dict[key]["entry"].get().lower() == "true":
                    values_dict[key]["value"] = True
                else:
                    values_dict[key]["value"] = values_dict[key]["entry"].get()
                if values_dict[key]["entry"].get().lower() == "skip":
                    del values_dict[key]
                else:
                    del values_dict[key]["entry"]
        sub_window.destroy()

    sub_window.wm_title("Selection")
    if flag == "entry":
```

```
        window_label = tk.Label(sub_window, text="In fields below enter needed values. If type is boolean write "
                                                 "'True'/'False' in any case. If you don't want to set this parameter "
                                                 "- write 'skip'", height=5)
    elif flag == "checkbutton":
        window_label = tk.Label(sub_window, text="In fields below set check button for options which you want to use",
                                height=5)
    window_label.grid(column=0, row=0)
    button_frame = tk.Frame(sub_window)
    sub_window_frame_1 = tk.Frame(sub_window, height=600, width=1200)
    button_frame.grid(column=1, row=2)
    counter_buttons = 1
    sub_window_frame_1.grid(column=0, row=1, sticky="nsew")
    button = tk.Button(button_frame, text="Start page", command=sub_window_frame_1.lift)
    button.grid(column=0, row=0)
    sub_window_frame = sub_window_frame_1
    counter = 1
    for key in values_dict.keys():
        if counter > 8:
            counter = 1
            counter_buttons += 1
            temporary_frame = tk.Frame(sub_window, height=600, width=1900)
            temporary_frame.grid(column=0, row=1, sticky="nsew")
            temporary_button = tk.Button(button_frame, text=str(counter_buttons) + " page",
                                         command=temporary_frame.tkraise)
            temporary_button.grid(column=counter_buttons, row=0)
            sub_window_frame = temporary_frame
        label = tk.Label(sub_window_frame, text=key, height=5)
        tool_tip_create(label, text=values_dict[key]["instruction"])
        if flag == "checkbutton":
            var = tk.IntVar()
            check_button = tk.Checkbutton(sub_window_frame, text="Yes", variable=var, onvalue=1, offvalue=0, height=5,
                                          width=5)
            label.grid(column=0, row=counter)
            check_button.grid(column=1, row=counter)
            values_dict[key]["value"] = var
        elif flag == "entry":
            entry = tk.Entry(sub_window_frame)
            label.grid(column=0, row=counter)
            entry.grid(column=1, row=counter)
            values_dict[key]["entry"] = entry
```

```
        counter += 1
    sub_window_frame_1.lift()
    button = tk.Button(button_frame, text="Send data", command=get_dict)
    button.grid(column=counter_buttons + 1, row=0)
    sub_window.mainloop()


def get_dict_from_json(dict_name):
    dict_json = open("default_dicts.json", "r")
    dict_json = json.loads(dict_json.read())
    buffer_json = dict_json[dict_name]
    return buffer_json


class SampleApp(tk.Tk):
    def __init__(self):
        tk.Tk.__init__(self)
        self.wm_title("Docker Security Beef Up")
        self.tcp_values_dict = dict()
        self.daemon_parameters_dict = dict()
        self.audit_files_dict = dict()
        self.all_targets_owners_dict = dict()
        self.all_targets_permissions_dict = dict()
        self.healthcheck_dict = dict()
        self.settings_dict = dict()
        self.restrict_string = ""
        self.dockerfile_path = ""
        self.sudo_password = ""
        self.value_tcp_check = tk.IntVar()
        self.value_audit_check = tk.IntVar()
        self.value_daemon_check = tk.IntVar()
        self.value_owners_check = tk.IntVar()
        self.value_permissions_check = tk.IntVar()
        self.value_healthcheck = tk.IntVar()
        self.value_restrict_access = tk.IntVar()
        self.label_healthcheck = tk.Label(self, text='Do you want to setup Healthcheck for target DockerFile?')
        self.label_dockerfile_path = tk.Label(self, text='Enter path to your dockerfile directory, '
                                                         'including home directory')
        self.label_sudo_password = tk.Label(self, text='Enter password for your user with sudo permissions')
        self.label_tcp_check = tk.Label(self, text='Do you want to setup TLS?')
        self.label_audit_check = tk.Label(self, text='Do you want to setup audit rules?')
        self.label_daemon_check = tk.Label(self, text='Do you want to setup daemon parameters?')
        self.label_owners_check = tk.Label(self, text='Do you want to setup owners for important files?')
```

```
        self.label_permissions_check = tk.Label(self, text='Do you want to setup permissions for important files?')
        self.label_restrict_access = tk.Label(self, text='Do you want to restrict access to docker for users(entry '
                                                         'users which you want to left, separate with space)?')
        self.label_settings = tk.Label(self, text="Additional settings")
        tool_tip_create(self.label_tcp_check, text="Enables TLS functionality and generate tls certificates")
        tool_tip_create(self.label_audit_check, text="Creates audit rules for selected files")
        tool_tip_create(self.label_daemon_check, text="Adds new parameters in daemon configuration")
        tool_tip_create(self.label_owners_check, text="Changes owners for files on entered ones")
        tool_tip_create(self.label_permissions_check, text="Changes permissions for files on entered ones")
        tool_tip_create(self.label_restrict_access, text="Deletes all unwanted users from docker group")
        tool_tip_create(self.label_healthcheck, text="Adds healthcheck to target DockerFile with entered
parameters")
        tool_tip_create(self.label_dockerfile_path, text="Path to target dockerfile(For example "
                                                         "/home/val/Desktop/bad-dockerfile)")
        tool_tip_create(self.label_sudo_password, text="Password of user with sudo permissions")
        self.entry_dockerfile_path = tk.Entry(self)
        self.entry_sudo_password = tk.Entry(self)
        self.entry_restrict_access = tk.Entry(self)
        self.button_audit_check = tk.Checkbutton(self, text="Yes", variable=self.value_audit_check, onvalue=1,
                                                 offvalue=0, height=5, width=20)
        self.button_tcp_check = tk.Checkbutton(self, text="Yes", variable=self.value_tcp_check, onvalue=1,
offvalue=0,
                                               height=5, width=20)
        self.button_daemon_check = tk.Checkbutton(self, text="Yes", variable=self.value_daemon_check,
onvalue=1,
                                                  offvalue=0, height=5, width=20)
        self.button_owners_check = tk.Checkbutton(self, text="Yes", variable=self.value_owners_check,
onvalue=1,
                                                  offvalue=0, height=5, width=20)
        self.button_permissions_check = tk.Checkbutton(self, text="Yes", variable=self.value_permissions_check,
                                                       onvalue=1, offvalue=0, height=5, width=20)
        self.button_healthcheck = tk.Checkbutton(self, text="Yes", variable=self.value_healthcheck, onvalue=1,
                                                 offvalue=0, height=5, width=20)
        self.button_restrict_access = tk.Checkbutton(self, text="Yes", variable=self.value_restrict_access,
onvalue=1,
                                                     offvalue=0, height=5, width=20)
        self.button_tcp_parameters = tk.Button(self, text="Enter Parameters", command=self.tcp_parameters_set)
        self.button_audit_parameters = tk.Button(self, text="Enter Parameters",
command=self.audit_parameters_set)
        self.button_daemon_parameters = tk.Button(self, text="Enter Parameters",
command=self.daemon_parameters_set)
        self.button_owners_parameters = tk.Button(self, text="Enter Parameters",
command=self.files_owners_set)
```

```
        self.button_healthcheck_parameters = tk.Button(self, text="Enter Parameters",
command=self.healthcheck_set)
        self.button_permissions_parameters = tk.Button(self, text="Enter Parameters",
                                               command=self.files_permissions_set)
        self.button_settings = tk.Button(self, text="Open Settings", command=self.settings_set)
        self.send_button = tk.Button(self, text="Send data", command=self.get_values)
        self.quit_button = tk.Button(self, text="Quit", command=self.destroy)
        self.app_init()

    def app_init(self):
        self.label_sudo_password.grid(column=0, row=0)
        self.entry_sudo_password.grid(column=1, row=0)
        self.label_dockerfile_path.grid(column=0, row=1)
        self.entry_dockerfile_path.grid(column=1, row=1)
        self.label_tcp_check.grid(column=0, row=2)
        self.button_tcp_check.grid(column=1, row=2)
        self.label_audit_check.grid(column=0, row=3)
        self.button_audit_check.grid(column=1, row=3)
        self.label_daemon_check.grid(column=0, row=4)
        self.button_daemon_check.grid(column=1, row=4)
        self.label_owners_check.grid(column=0, row=5)
        self.button_owners_check.grid(column=1, row=5)
        self.label_permissions_check.grid(column=0, row=6)
        self.button_permissions_check.grid(column=1, row=6)
        self.label_healthcheck.grid(column=0, row=7)
        self.button_healthcheck.grid(column=1, row=7)
        self.button_tcp_parameters.grid(column=2, row=2)
        self.button_audit_parameters.grid(column=2, row=3)
        self.button_daemon_parameters.grid(column=2, row=4)
        self.button_owners_parameters.grid(column=2, row=5)
        self.button_permissions_parameters.grid(column=2, row=6)
        self.button_healthcheck_parameters.grid(column=2, row=7)
        self.label_restrict_access.grid(column=0, row=8)
        self.button_restrict_access.grid(column=1, row=8)
        self.entry_restrict_access.grid(column=2, row=8)
        self.label_settings.grid(column=0, row=9)
        self.button_settings.grid(column=2, row=9)
        self.send_button.grid(column=3, row=10)
        self.quit_button.grid(column=4, row=10)

    def get_values(self):
        if self.daemon_parameters_dict == {}:
            self.daemon_parameters_dict = get_dict_from_json("daemon_parameters_dict")
```

```
    if self.audit_files_dict == {}:
        self.audit_files_dict = get_dict_from_json("audit_files_dict")
    if self.all_targets_owners_dict == {}:
        self.all_targets_owners_dict = get_dict_from_json("all_targets_owners_dict")
    if self.all_targets_permissions_dict == {}:
        self.all_targets_permissions_dict = get_dict_from_json("all_targets_permissions_dict")
    if self.healthcheck_dict == {}:
        self.healthcheck_dict = get_dict_from_json("healthcheck_dict")
    if self.settings_dict == {}:
        self.settings_dict = get_dict_from_json("settings_dict")
    self.dockerfile_path = self.entry_dockerfile_path.get()
    self.sudo_password = self.entry_sudo_password.get()
    if self.sudo_password == "":
        raise ValueError("Missing password for user with sudo permissions, program will not work")
    if self.dockerfile_path == "":
        self.value_healthcheck.set(0)
        self.settings_dict["Use COPY Command instead ADD in target DockerFile"]["value"] = 0
        self.settings_dict["Add separate USER to target DockerFile"]["value"] = 0
    if self.value_tcp_check.get() == 1:
        tcp_values_dict_reformatted = dict()
        for key in self.tcp_values_dict:
            if self.tcp_values_dict[key] == "BitLength":
                self.tcp_values_dict[key]["value"] = int(self.tcp_values_dict[key]["value"])
            if self.tcp_values_dict[key]["title"] not in tcp_values_dict_reformatted.keys():
                tcp_values_dict_reformatted[self.tcp_values_dict[key]["title"]] = dict()
                tcp_values_dict_reformatted[self.tcp_values_dict[key]["title"]][key] = self.tcp_values_dict[key][
                    "value"]
            else:
                tcp_values_dict_reformatted[self.tcp_values_dict[key]["title"]][key] = self.tcp_values_dict[key][
                    "value"]
        temporary_values_dict = tcp_values_dict_reformatted
        yaml_dumped = yaml.dump(temporary_values_dict)
        with open("config.yaml", "w") as file:
            file.write(yaml_dumped)
            file.close()
    if self.value_restrict_access.get() == 1:
        self.restrict_string = self.entry_restrict_access.get()
    self.destroy()

def audit_parameters_set(self):
    self.audit_files_dict = dict()
    self.audit_files_dict = get_dict_from_json("audit_files_dict")
    set_variable_dictionary(self.audit_files_dict, "checkbutton")
```

```
    def tcp_parameters_set(self):
        self.tcp_values_dict = dict()
        self.tcp_values_dict = get_dict_from_json("tcp_values_dict")
        set_variable_dictionary(self.tcp_values_dict, "entry")

    def daemon_parameters_set(self):
        self.daemon_parameters_dict = dict()
        self.daemon_parameters_dict = get_dict_from_json("daemon_parameters_dict")
        set_variable_dictionary(self.daemon_parameters_dict, "entry")

    def files_owners_set(self):
        self.all_targets_owners_dict = dict()
        self.all_targets_owners_dict = get_dict_from_json("all_targets_owners_dict")
        set_variable_dictionary(self.all_targets_owners_dict, "entry")

    def files_permissions_set(self):
        self.all_targets_permissions_dict = dict()
        self.all_targets_permissions_dict = get_dict_from_json("all_targets_permissions_dict")
        set_variable_dictionary(self.all_targets_permissions_dict, "entry")

    def healthcheck_set(self):
        self.healthcheck_dict = dict()
        self.healthcheck_dict = get_dict_from_json("healthcheck_dict")
        set_variable_dictionary(self.healthcheck_dict, "entry")

    def settings_set(self):
        self.settings_dict = dict()
        self.settings_dict = get_dict_from_json("settings_dict")
        set_variable_dictionary(self.settings_dict, "checkbutton")


if __name__ == "__main__":
    app = SampleApp()
    app.mainloop()
    print(app.value_healthcheck.get())
```

8. certificates_creation.py

```
from OpenSSL import crypto
import os
import yaml
from OpenSSL.crypto import TYPE_RSA, TYPE_DSA, FILETYPE_PEM, load_certificate_request, PKCS12,
FILETYPE_ASN1, \
    load_privatekey, X509Req
```

```
import random



class Secure:
    def __init__(self, config_file):
        self.config_file = config_file
        config_yaml = self.config_parsing()
        if config_yaml['CERT'].get('KeyType') == 'RSA':
            self.key_type = TYPE_RSA
        elif config_yaml['CERT'].get('KeyType') == 'DSA':
            self.key_type = TYPE_DSA
        self.bit_length = config_yaml['CERT'].get('BitLength')
        self.not_before_time = config_yaml['CSR'].get('validfrom')
        self.not_after_time = config_yaml['CSR'].get('validto')
        self.valid_from = config_yaml['CERT'].get('validfrom')
        self.valid_to = config_yaml['CERT'].get('validto')
        self.digest_type = config_yaml['CERT'].get('digestType')
        self.cert_dir = config_yaml['CERT'].get('CertDir')
        self.certificate = config_yaml['REUSE'].get('Certificate')
        self.old_private_key = config_yaml['REUSE'].get('OldPrivateKey')
        self.old_private_key_type = config_yaml['REUSE'].get('OldPrivateKeyType')
        self.csr_file = config_yaml['CSR'].get('OldCsrFile')
        self.old_scr_file = config_yaml['CSR'].get('Oldcsr_fileType')

    def config_parsing(self):
        with open(self.config_file) as config:
            try:
                config_yaml = yaml.safe_load(config)
            except Exception as ConfigException:
                print("Failed to read Configuration %s" % (ConfigException))
        return config_yaml

    def get_private_key(self):
        if not self.old_private_key:
            key = crypto.PKey()
            key.generate_key(self.key_type, self.bit_length)
        else:
            with open(self.old_private_key) as key_file:
                if self.old_private_key_type == 'PEM':
                    key = load_privatekey(FILETYPE_PEM, key_file.read())
                elif self.old_private_key_type == 'DER':
                    key = load_privatekey(FILETYPE_ASN1, key_file.read())
        return key
```

```
def creation_csr(self, key):
    if not self.csr_file:
        config_yaml = self.config_parsing()
        csr = X509Req()
        csr.get_subject().countryName = config_yaml['CSR'].get('countryName')
        csr.get_subject().stateOrProvinceName = config_yaml['CSR'].get('stateOrProvinceName')
        csr.get_subject().commonName = config_yaml['CSR'].get('commonName')
        csr.get_subject().localityName = config_yaml['CSR'].get('localityName')
        csr.get_subject().emailAddress = config_yaml['CSR'].get('emailAddress')
        csr.get_subject().organizationName = config_yaml['CSR'].get('organizationName')
        csr.get_subject().organizationalUnitName = config_yaml['CSR'].get('organizationalUnitName')
        csr.set_pubkey(key)
        csr.sign(key, self.digest_type)
    else:
        with open(self.csr_file) as csr_file:
            if self.old_scr_file == 'PEM':
                csr = load_certificate_request(FILETYPE_PEM, csr_file.read())
            elif self.old_scr_file == 'DER':
                csr = load_certificate_request(FILETYPE_ASN1, csr_file.read())
            else:
                raise TypeError("Unknown Certificate Type %s" % (self.old_scr_file))
    return csr

def creation_certificate(self, csr, key):
    certificate = crypto.X509()
    certificate.get_subject().countryName = csr.get_subject().countryName
    certificate.get_subject().stateOrProvinceName = csr.get_subject().stateOrProvinceName
    certificate.get_subject().commonName = csr.get_subject().commonName
    certificate.get_subject().localityName = csr.get_subject().localityName
    certificate.get_subject().emailAddress = csr.get_subject().emailAddress
    certificate.get_subject().organizationName = csr.get_subject().organizationName
    certificate.get_subject().organizationalUnitName = csr.get_subject().organizationalUnitName
    if self.valid_from and self.valid_to:
        valid_from_bytes = bytes(self.valid_from, 'utf-8')
        valid_to_bytes = bytes(self.valid_to, 'utf-8')
        certificate.set_notBefore(valid_from_bytes)
        certificate.set_notAfter(valid_to_bytes)
    certificate.set_pubkey(key)
    certificate.sign(key, self.digest_type)
    return certificate

def creation_ca_certificate(self, csr, certificate, key):
```

```
        serialnumber = random.getrandbits(64)
        ca_certificate = crypto.X509()
        ca_certificate.set_serial_number(serialnumber)
        ca_certificate.gmtime_adj_notBefore(0)
        ca_certificate.gmtime_adj_notAfter(31536000)
        ca_certificate.set_subject(csr.get_subject())
        ca_certificate.set_issuer(certificate.get_subject())
        ca_certificate.set_pubkey(key)
        subject_alt_name = crypto.X509Extension(b'subjectAltName', False, b'DNS:xxx,IP:127.0.0.1')
        ca_certificate.add_extensions([subject_alt_name])
        ca_certificate.sign(key, "sha512")
        return ca_certificate

    def key_certificate_ca_dump(self):
        key = self.get_private_key()
        csr = self.creation_csr(key)
        certificate = self.creation_certificate(csr, key)
        ca_certificate = self.creation_ca_certificate(csr, certificate, key)
        pem_file_key = os.path.join(self.cert_dir, "key.pem")
        pem_file_ca_certificate = os.path.join(self.cert_dir, "ca.pem")
        pem_file_certificate = os.path.join(self.cert_dir, "cert.pem")
        if not self.old_private_key:
            with open(pem_file_key, "w") as pem_f_key:
                pem_f_key.write(crypto.dump_privatekey(FILETYPE_PEM, key).decode("utf-8"))
        if not self.csr_file:
            with open(pem_file_ca_certificate, "w") as pem_f_ca_certificate:
                pem_f_ca_certificate.write(crypto.dump_certificate(FILETYPE_PEM, ca_certificate).decode("utf-
8"))
        with open(pem_file_certificate, "w") as pem_f_certificate:
            pem_f_certificate.write(crypto.dump_certificate(FILETYPE_PEM, certificate).decode("utf-8"))
        return pem_file_key, pem_file_ca_certificate, pem_file_certificate
```

9. authzrego_default

```
package docker.authz

default allow = false

# allow if the user is granted read/write access.
allow {
    user_id := input.Headers["Authz-User"]
    user := users[user_id]
    not user.readOnly
}
```

```
# allow if the user is granted read-only access and the request is a GET.
allow {
    user_id := input.Headers["Authz-User"]
    users[user_id].readOnly
    input.Method == "GET"
}

# users defines permissions for the user. In this case, we define a single
# attribute 'readOnly' that controls the kinds of commands the user can run.
users = {
    "test_user": {"readOnly": false}
}
```

10. daemon_default.json

```
{
    "authorization-plugins": [],
    "dns": [],
    "dns-opts": [],
    "dns-search": [],
    "exec-opts": [],
    "experimental": false,
    "storage-driver": "",
    "storage-opts": [],
    "labels": [],
    "log-driver": "",
    "mtu": 0,
    "pidfile": "",
    "graph": "",
    "cluster-store": "",
    "cluster-advertise": "",
    "max-concurrent-downloads": 3,
    "max-concurrent-uploads": 5,
    "shutdown-timeout": 15,
    "debug": true,
    "hosts": [],
    "log-level": "",
    "swarm-default-advertise-addr": "",
    "group": "",
    "default-ulimits": {},
    "bridge": "",
    "fixed-cidr": "",
    "raw-logs": false,
    "registry-mirrors": [],
```

```
    "insecure-registries": []
}
```

11. default_dicts.json

```
{
    "healthcheck_dict": {
        "interval": {
            "value": "30",
            "instruction": "This parameter defines an interval in which healthcheck will be proceeded"
            },
        "timeout": {
            "value": "3",
            "instruction": "This parameter defines a timeout of healthcheck"
            }
    },
    "settings_dict": {
        "Enable Docker Trust": {
            "value": 0,
            "instruction": "Enables Docker Trust"
        },
        "Update Docker": {
            "value": 0,
            "instruction": "Starts update of docker to latest version"
        },
        "Install Authorization Plugin": {
            "value": 0,
            "instruction": "Installs and enables authorization plugin for docker(default authorization parameters written in authzrego_default.txt)"
        },
        "Use COPY Command instead ADD in target DockerFile": {
            "value": 0,
            "instruction": "Changes all ADD commands on COPY in target DockerFile(This may broke your DockerFile)"
        },
        "Add separate USER to target DockerFile": {
            "value": 0,
            "instruction": "Creates separate non-root user in target Dockerfile"
        },
        "Update base image to latest version in target DockerFile": {
            "value": 0,
            "instruction": "Updates all base images in target DockerFile to latest tag(This may broke your DockerFile)"
        },
        "Build an image": {
```

```
        "value": 0,
        "instruction": "Builds image from target DockerFile"
    }
},
"tcp_values_dict": {
    "BitLength": {
        "value": 2048,
        "instruction": "Bit Length of key",
        "title": "CERT"
    },
    "CertDir": {
        "value": "/etc/docker/certs.d",
        "instruction": "Directory there Keys will be stored",
        "title": "CERT"
    },
    "KeyType": {
        "value": "RSA",
        "instruction": "Type of the key",
        "title": "CERT"
    },
    "digestType": {
        "value": "sha256",
        "instruction": "Type of digest which will be used for signing key",
        "title": "CERT"
    },
    "validfrom": {
        "value": "20220510000000Z",
        "instruction": "Date from which certificate is valid(write as 'yearmonthdaytimeZ'",
        "title": "CERT"
    },
    "validto": {
        "value": "20221010000000Z",
        "instruction": "Date until which certificate is valid(write as 'yearmonthdaytimeZ'",
        "title": "CERT"
    },
    "OldCsrFile": {
        "value": null,
        "instruction": "Path to old CSR file. Fill the field if you want to use it in generation of certificates.",
        "title": "CSR"
    },
    "OldCsrFileType": {
        "value": null,
        "instruction": "Type of old CSR file. Fill the field if you want to use it in generation of certificates.",
```

```
    "title": "CSR"
  },
  "commonName": {
    "value": "dyploma",
    "instruction": "Common Name",
    "title": "CSR"
  },
  "countryName": {
    "value": "UA",
    "instruction": "Country Name",
    "title": "CSR"
  },
  "emailAddress": {
    "value": "kravchenkovalentinv@gmail.com",
    "instruction": "Email address",
    "title": "CSR"
  },
  "localityName": {
    "value": "KPI",
    "instruction": "Locality Name",
    "title": "CSR"
  },
  "organizationName": {
    "value": "PTI",
    "instruction": "Organization Name",
    "title": "CSR"
  },
  "organizationalUnitName": {
    "value": "Certification Authority",
    "instruction": "Organization Unit Name",
    "title": "CSR"
  },
  "stateOrProvinceName": {
    "value": "Kyiv",
    "instruction": "State/Province Name",
    "title": "CSR"
  },
  "OldPrivateKey": {
    "value": null,
    "instruction": "Path to old private key. Fill the field if you want to use it in generation of certificates.",
    "title": "REUSE"
  },
  "OldPrivateKeyType": {
```

```
        "value": null,
        "instruction": "Type of old private key. Fill the field if you want to use it in generation of certificates.",
        "title": "REUSE"
    }
},
"audit_files_dict": {
    "/usr/bin/docker": {
        "value": 1,
        "instruction": ""
    },
    "/var/lib/docker": {
        "value": 1,
        "instruction": ""
    },
    "/etc/docker": {
        "value": 1,
        "instruction": ""
    },
    "/lib/systemd/system/docker.service": {
        "value": 1,
        "instruction": ""
     },
    "/etc/default/docker": {
        "value": 1,
        "instruction": ""
    },
    "/etc/docker/daemon.json": {
        "value": 1,
        "instruction": ""
    },
    "/usr/bin/docker-containerd": {
        "value": 1,
        "instruction": ""
    },
    "/usr/bin/docker-runc": {
        "value": 1,
        "instruction": ""
    },
    "/usr/bin/dockerd": {
        "value": 1,
        "instruction": ""
    },
    "/etc/containerd/config.toml": {
```

```
            "value": 1,
            "instruction": ""
        },
        "/run/containerd": {
            "value": 1,
            "instruction": ""
        },
        "/usr/bin/containerd": {
            "value": 1,
            "instruction": ""
        },
        "/usr/bin/containerd-shim": {
            "value": 1,
            "instruction": ""
        },
        "/usr/bin/containerd-shim-runc-v1": {
            "value": 1,
            "instruction": ""
        },
        "/usr/bin/containerd-shim-runc-v2": {
            "value": 1,
            "instruction": ""
        },
        "/usr/bin/runc": {
            "value": 1,
            "instruction": ""
        },
        "/lib/systemd/system/docker.socket": {
            "value": 1,
            "instruction": ""
        }
    },
    "daemon_parameters_dict": {
        "log-driver": {
            "value": "syslog",
            "instruction": "Default driver for container logs (Default 'syslog')"
            },
        "live-restore": {
            "value": true,
            "instruction": "Enable/Disable(true/false) keeping containers alive during daemon downtime(Default is
enable)"
            },
        "userland-proxy": {
```

```
            "value": false,
            "instruction": "Enable/Disable(true/false) use of userland proxy for loopback traffic (Default is disable)"
            },
        "no-new-privileges": {
            "value": true,
            "instruction": "Enable/Disable(true/false) Restrict gaining by default for new containers(Default is
enable)"
            },
        "experimental": {
            "value": false,
            "instruction": "Enable/Disable(true/false) experimental features(Default is disable)"
            },
        "icc": {
            "value": false,
            "instruction": "Enable/Disable(true/false) inter-container communication(Default is disable)"
            },
        "log-level": {
            "value": "info",
            "instruction": "the logging level ('debug'|'info'|'warn'|'error'|'fatal')(Default is 'info')"
            },
        "insecure-registries": {
            "value": [],
            "instruction": "Replaces the daemon insecure registries with a new set of insecure registries. If some
existing insecure registries in daemon’s configuration are not in newly reloaded insecure registries, these
existing ones will be removed from daemon’s config."
            },
        "default-ulimits":{
            "value": {
                "nofile": {"Hard": 64000, "Name": "nofile", "Soft": 64000},
                "nproc": {"Hard": 64000, "Name": "nproc", "Soft": 64000}
                },
            "instruction": "Default ulimits for containers(By default set on nofile and nproc)"
            }
    },
    "all_targets_owners_dict": {
        "/lib/systemd/system/docker.service": {
            "value": "root:root",
            "instruction": ""
        },
        "/lib/systemd/system/docker.socket": {
            "value": "root:root",
            "instruction": ""
        },
```

```
    "/etc/docker": {
        "value": "root:root",
        "instruction": ""
    },
    "/etc/docker/daemon.json": {
        "value": "root:root",
        "instruction": ""
    },
    "/etc/default/docker": {
        "value": "root:root",
        "instruction": ""
    },
    "/var/run/docker.sock": {
        "value": "root:docker",
        "instruction": ""
    },
    "/etc/docker/certs.d/ca.pem": {
        "value": "root:root",
        "instruction": ""
    },
    "/etc/docker/certs.d/key.pem": {
        "value": "root:root",
        "instruction": ""
    },
    "/etc/docker/certs.d/cert.pem": {
        "value": "root:root",
        "instruction": ""
    }
},
"all_targets_permissions_dict": {
    "/lib/systemd/system/docker.service": {
        "value": "644",
        "instruction": ""
    },
    "/lib/systemd/system/docker.socket": {
        "value": "644",
        "instruction": ""
    },
    "/etc/docker": {
        "value": "755",
        "instruction": ""
    },
    "/etc/docker/daemon.json": {
```

```
            "value": "644",
            "instruction": ""
        },
        "/etc/default/docker": {
            "value": "644",
            "instruction": ""
        },
        "/var/run/docker.sock": {
            "value": "660",
            "instruction": ""
        },
        "/etc/docker/certs.d/ca.pem": {
            "value": "444",
            "instruction": ""
        },
        "/etc/docker/certs.d/key.pem": {
            "value": "400",
            "instruction": ""
        },
        "/etc/docker/certs.d/cert.pem": {
            "value": "444",
            "instruction": ""
        }
    }
}
```