Хмельницький національний університет
Факультет інформаційних технологій
Кафедра кібербезпеки

КВАЛІФІКАЦІЙНА РОБОТА

Вознюка Андрія Віталійовича

на здобуття ступеня вищої освіти Бакалавра

Система захисту мікросервісних застосунків від вірусів шпигунів та вірусів шифрувальників

Галузь знань 12 – Інформаційні технології
Спеціальність 123 – Комп'ютерна інженерія
Освітня програма Програмування та захист комп'ютерних систем і мереж

Шифр КРБКІ. 001116.20.01.02 ПЗ

Виконав студент 4 курсу група КІ-20-1 Андрій ВОЗНЮК
Керівник доктор філософії Микола СТЕЦЮК
Нормоконтролер старший викладач Сергій МОСТОВИЙ

До захисту допускаю:
Завідувач кафедри кібербезпеки Юрій КЛЬОЦ

19 06 2024 р.

Хмельницький 2024

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет Інформаційних технологій

Кафедра Кібербезпеки

Рівень вищої освіти Бакалавр

Галузь знань 12 – Інформаційні технології

Спеціальність 123 – Комп'ютерна інженерія

Освітня програма Програмування та захист комп`ютерних систем і мереж

ЗАТВЕРДЖУЮ

Завідувач кафедри кібербезпеки

Юрій КЛЬОЦ

15 лютого 2024 р.

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Вознюку Андрію Віталійовичу

1 Тема роботи Система захисту мікросервісних застосунків від вірусів шпигунів та вірусів шифрувальників

Керівник роботи Стецюк М.В.

Затверджено наказом ректора університету від 15 лютого 2024 № 8

2 Строк подання студентом кваліфікаційної роботи на кафедру

3 Вихідні дані до роботи завдання на кваліфікаційну роботу створення системи захисту для мікросервісів.

4 Зміст пояснювальної записки (перелік питань, які потрібно розробити)
Дослідження предметної області та постановка задачі; обгрунтування щодо проектування системи захисту та мікросервісних застосунків; опис архітектури проектованої системи захисту; опис алгоритму роботи системи

5 Перелік графічного матеріалу (із зазначенням обов'язкових креслень)
Загальна архітектура мікросервісу, Алгоритм робот ClamAV, Алгоритм роботи моніторингу.

6 Консультанти розділів кваліфікаційної роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Мостовий С.В., старший викладач кафедри кібербезпеки		

7 Дата видачі завдання 16 лютого 2024 р.

КАЛЕНДАРНИЙ ПЛАН

Назва етапів (розділів) кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
Вибір і затвердження теми кваліфікаційної роботи	Січень-Лютий	виконав
Ознайомлення з предметною областю	Лютий	виконав
Дослідження існуючих рішень	Лютий	виконав
Постановка задачі	Березень	виконав
Визначення загальних принципів рішення задачі	Березень	виконав
Деталізація принципів рішення задачі	Квітень	виконав
Розробка проєктних рішень	Квітень	виконав
Апробація проєктних рішень	Травень	виконав
Оформлення пояснювальної записки згідно вимог	Травень	виконав
Оформлення графічної частини	Червень	виконав
Захист КР	Червень	виконав

Студент _____ Андрій ВОЗНЮК

Керівник кваліфікаційної роботи _____ Микола СТЕЦЮК

№ ряд-ка	формат	Позначення	Найменування	Кіл. листів	№ екз	Примітка
			Текстові документи			
1		КРБКІ. 001116.20.01.02 ПЗ	Пояснювальна записка	62		
			Графічні матеріали			
2		КРБКІ. 001116.20.01.02 Е1	Загальна архітектура	1		
			мікросервісу			
3		КРБКІ. 001116.20.01.02 Е2	Алгоритм роботи	1		
			ClamAV			
4		КРБКІ. 001116.20.01.02 Е3	Алгоритм роботи	1		
			моніторингу			

Зм	Арк	№ докум	Підпис	Дата	КРБКІ. 001116.20.01.02 ПЗ			
Розробив		Вознюк А.В.		20.06.24	Система захисту мікросервісних застосунків від вірусів шпигунів та вірусів шифрувальників	Літера	Аркуш	Аркушів
Перевір.		Стецюк М.В.		20.06.24		У	1	1
Н. контр.		Мостовий С.В.		21.06.24		ХНУ, КІ1–20–1		
Затв.		Кльоц Ю.П.		21.06.24				

АНОТАЦІЯ

Тема - Розробка системи захисту мікросервісних застосунків від вірусів-шпигунів та вірусів-шифрувальників.

Ключові слова - Мікросервіси, кіберзахист, віруси-шпигуни, віруси-шифрувальники, багаторівневий захист, автоматизована система, штучний інтелект.

Записка - 62 стор., 6 рис., 41 джерел.

Мета роботи - Розробка комплексної системи захисту мікросервісних застосунків від вірусів-шпигунів та вірусів-шифрувальників, що включає багаторівневий підхід із використанням технологій штучного інтелекту для автоматизації виявлення та реагування на загрози.

Об'єкт дослідження - Мікросервісні застосунки у різних інформаційних системах.

Предмет дослідження - Сукупність програмно-технічних засобів, спрямованих на забезпечення кібербезпеки мікросервісних застосунків від вірусів-шпигунів та вірусів-шифрувальників.

Методи дослідження - Методи теоретичного узагальнення використовувалися при описі предметної області дослідження, а аналізу та синтезу – при дослідженні існуючих методів захисту мікросервісних застосунків. Формалізація була застосована при створенні моделей мікросервісних застосунків у середовищах розробки, а системний аналіз та експеримент – при розробці та тестуванні запропонованих рішень для захисту від вірусів.

Результати - Розроблено багаторівневу систему захисту мікросервісних застосунків, що включає статичний і динамічний аналіз, систему виявлення та попередження вторгнень, резервне копіювання та відновлення даних, контроль доступу та автентифікацію.

12.06.2024 _____

ABSTRACT

Topic - Development of a system for protecting microservice applications from spy viruses and encryption viruses.

Keywords - Microservices, cyber protection, spyware viruses, encryption viruses, multi-level protection, automated system, artificial intelligence.

Note - 62 pages, 6 pictures, 41 sources.

The purpose of the work - is to develop a comprehensive system for protecting microservice applications from spyware and encryption viruses, which includes a multi-level approach using artificial intelligence technologies to automate the detection and response to threats.

Object of research - Microservice applications in various information systems.

The subject of the study is a set of software and technical tools aimed at ensuring the cyber security of microservice applications against spy viruses and encryption viruses.

Research methods - Theoretical generalization methods were used when describing the subject area of research, and analysis and synthesis - when researching existing methods of protecting microservice applications. Formalization was applied in the creation of models of microservice applications in development environments, and system analysis and experimentation - in the development and testing of proposed solutions for virus protection.

Results - A multi-level protection system for microservice applications was developed, including static and dynamic analysis, intrusion detection and prevention system, data backup and recovery, access control and authenticatio.

12.06.2024

ЗМІСТ

ВСТУП

Тема кібербезпеки стала особливо актуальною в сучасному світі цифрових технологій, де бізнес-процеси та повсякденне життя все більше залежать від інформаційних систем.

Із зростанням хмарних обчислень і поширенням архітектур мікросервісів підприємства стикаються з новими проблемами щодо захисту своїх даних і послуг.

Мікросервіси, які дозволяють створювати гнучкі та легко масштабовані програми, також відкривають нові вектори атак для кіберзлочинців.

Найнебезпечніші загрози включають віруси, шпигунське програмне забезпечення та криптографічні віруси, які можуть завдати значної шкоди, від крадіжки конфіденційної інформації до повного блокування доступу до важливих даних.

Це дослідження присвячене розробці комплексної системи захисту для додатків мікросервісів, які можуть протистояти певним кіберзагрозам.

Ми розглянемо основні принципи побудови архітектури мікросервісів, її вразливості та методи захисту, які можна застосувати для підвищення рівня безпеки.

Особлива увага приділяється аналізу останніх загроз та розробці ефективних механізмів їх виявлення та нейтралізації.

Почніть із теоретичного огляду архітектури мікросервісів, визначте її ключові особливості та переваги перед традиційними монолітними системами.

Потім ми зосереджуємося на типових уразливостях, якими можуть скористатися зловмисники, і переглядаємо існуючі стратегії та інструменти захисту, включно з тими, що використовуються в галузі.

У практичній частині роботи описано проектування та розробку систем захисту, включаючи алгоритми виявлення шкідливих програм, шпигунського програмного забезпечення та криптографічних вірусів.

Розробити архітектуру системи, вибрати технології та інструменти для впровадження, а також протестувати та апробувати розроблену систему.

Важливою частиною досліджень є експериментальні дослідження, під час яких системи перевіряються в контрольованих умовах для оцінки їх ефективності.

Результати цих експериментів дозволяють провести аналіз і порівняння з іншими методами захисту та визначити потенційні області для подальшого вдосконалення системи.

У висновку підбиваються підсумки виконаної роботи, визначаються основні переваги та недоліки розробленої системи, надаються рекомендації щодо практичного впровадження та подальшого розвитку.

Додатки містять код реалізації системи, результати тестування та інші супутні матеріали, які допоможуть вам зрозуміти та використовувати розроблену систему.

1 ОГЛЯД МІКРОСЕРВІСНИХ ЗАСТОСУНКІВ

1.1 Огляд архітектур мікросервісів та їх загрози для безпеки

Мікросервісна архітектура — це підхід до розробки програмного забезпечення, за якого програма складається з ряду невеликих незалежних сервісів, кожен з яких виконує певну бізнес-функцію.

Кожен мікросервіс розгортається окремо, може бути написаний іншою мовою програмування та використовувати різні технології зберігання. Зразок архітектури мікросервісів на Рисунок.1.1.

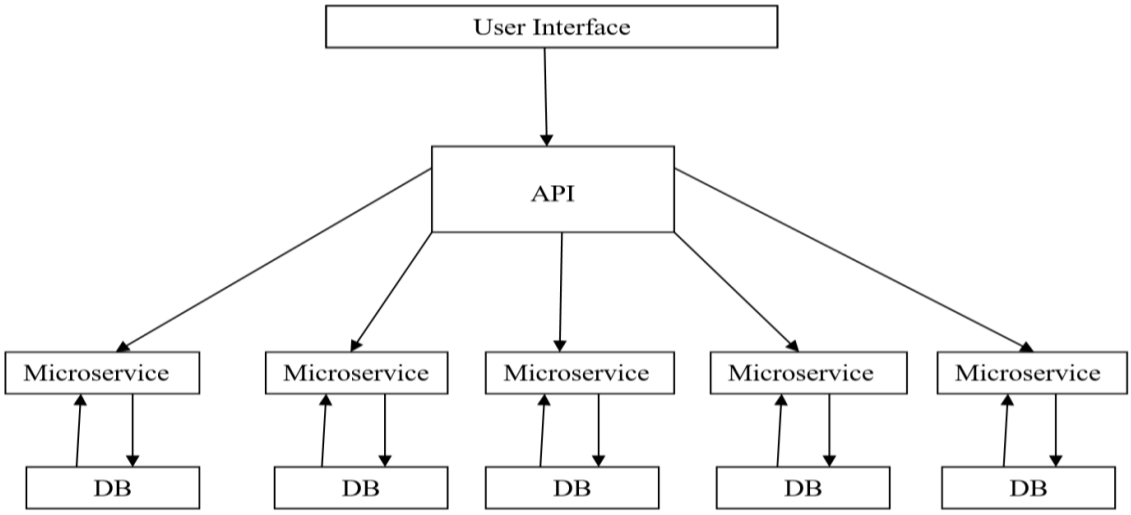


Рисунок.1.1 Архітектура мікросервісів

Ключові характеристики архітектури мікросервісів включають модульність, автономність, масштабованість, незалежне розгортання та гнучкість у виборі технологій. Модульність означає, що кожен мікросервіс відповідає за певну функціональність або бізнес-логіку, і розробники можуть легко змінювати або оновлювати певні частини системи, не впливаючи на інші частини.

Автономія гарантує, що мікросервіси є самодостатніми та працюють незалежно один від одного, що дозволяє різним командам працювати над різними мікросервісами паралельно.

Масштабованість кожного мікросервісу дозволяє кожному мікросервісу масштабуватися незалежно від інших мікросервісів, забезпечуючи гнучкість у розподілі ресурсів.

Незалежне розгортання дозволяє розгортати та оновлювати мікросервіси, не впливаючи на всю систему, зменшуючи ризик помилок.

Гнучкість у виборі технологій дозволяє командам вибирати найбільш відповідні інструменти та технології для реалізації кожного мікросервісу, гарантуючи, що вони використовують найсучасніші рішення. Архітектура мікросервісів пропонує кілька переваг перед монолітними системами.

Індивідуальне розгортання мікросервісів збільшує швидкість розробки та розгортання, дозволяючи командам швидше впроваджувати нові функції та усувати помилки.

Гнучкість і масштабованість мікросервісів робить їх економічно вигідними, оскільки ви можете масштабувати лише ті частини системи, які потребують додаткових ресурсів.

Зміни в одному мікросервісі не впливають на поведінку інших мікросервісів, полегшуючи підтримку та модифікацію, зменшуючи ризик несподіваних помилок і спрощуючи тестування.

Розподіл обов’язків дозволяє різним командам працювати над різними мікросервісами паралельно, підвищуючи ефективність розробки.

Однак архітектури мікросервісів також мають недоліки.

Зі збільшенням кількості мікросервісів складність управління зростає, що ускладнює керування та моніторинг системи.

Зв’язок між службами є складним завданням, оскільки для забезпечення надійного та безпечного зв’язку між мікросервісами потрібні додаткові інструменти та протоколи.

Забезпечення узгодженості даних є складним завданням, оскільки розподіл даних між мікросервісами ускладнює забезпечення узгодженості.

Архітектури мікросервісів вимагають більш складної інфраструктури для оркестрування, масштабування та моніторингу послуг, що збільшує витрати на інфраструктуру та потенційно створює додаткові витрати.

Архітектури мікросервісів відкривають нові вектори атак і створюють додаткові виклики безпеці.

Найбільші загрози безпеці для мікросервісів включають атаки API, незахищений зв'язок, проблеми автентифікації та авторизації, уразливості бібліотек і фреймворків, а також відсутність централізованого моніторингу.

Атаки API здійснюються мікросервісами, які взаємодіють через API, створюючи можливості для атак на відмову в обслуговуванні (DoS) та інших типів атак на рівні API:

– Якщо не застосовувати належне шифрування, незахищений зв'язок може дозволити зловмиснику перехопити дані, що передаються між мікросервісами.

– Забезпечення правильного рівня доступу для кожного мікросервісу є складним завданням і створює можливості для несанкціонованого доступу.

– Використання різних бібліотек і фреймворків у різних мікросервісах підвищує ризик використання зловмисниками вразливостей.

– Відсутність єдиного центру моніторингу ускладнює виявлення інцидентів безпеки та реагування на них.

– Однією з найбільших проблем у захисті архітектур мікросервісів є керування ідентифікацією та доступом (IAM).

– Надійні механізми автентифікації та авторизації повинні бути реалізовані для керування доступом до кожного мікросервісу.

– Такі маркери, як JWT (веб-токени JSON), можуть допомогти з цим завданням, дозволяючи безпечно та надійно передавати інформацію про ідентифікацію користувача між мікросервісами.

– Крім того, важливим аспектом є моніторинг і журналювання в системах мікросервісів. Централізоване ведення журналів і моніторинг допомагає вчасно виявляти потенційні загрози та реагувати на них.

Такі інструменти, як ELK Stack (Elasticsearch, Logstash, Kibana), Prometheus і Grafana, забезпечують комплексний підхід до моніторингу й аналізу журналів, що допомагає виявляти підозрілу активність і помилки у ваших мікросервісах.

Ще одним важливим аспектом є автоматизація розгортання та оновлення мікросервісів. Використовуючи підхід CI/CD (безперервна інтеграція/безперервне розгортання), ви можете автоматизувати процес розгортання та тестування та зменшити ризик помилок і вразливостей у вашому коді.

Такі інструменти, як Jenkins, GitLab CI/CD і CircleCI, допомагають забезпечити безперервний потік оновлень, зберігаючи високу якість і безпеку коду.

Нарешті, важливим аспектом є впровадження політик безпеки на рівні контейнерів, у яких зазвичай розгортаються мікросервіси.

Контейнерні оркестратори, такі як Kubernetes, дозволяють реалізувати політику безпеки, контролю доступу та ресурсів на рівні мережі.

Контейнерізація додає рівень ізоляції між мікросервісами, покращуючи загальну безпеку системи.

З огляду на ці загрози, розробка ефективних методів і інструментів для захисту додатків мікросервісів стає критично важливим завданням для забезпечення безпеки та надійності сучасних розподілених систем.

У цій статті розглядаються існуючі методи та інструменти захисту та пропонується новий підхід до виявлення та видалення вірусів, шпигунського програмного забезпечення та криптовірусів в архітектурах мікросервісів.

Це включає використання мови програмування Python для розробки ефективних алгоритмів і інструментів моніторингу, які забезпечують високий рівень захисту для сучасних програм.

1.2 Аналіз типових вразливостей мікросервісних систем щодо вірусів, шпигунів та вірусів-шифрувальників

Хоча архітектура мікросервісів має багато переваг, вона наражає вас на різноманітні загрози безпеці.

Основними вразливими місцями в системах мікросервісів є вразливості API, незахищений зв’язок між мікросервісами, уразливості бібліотеки та фреймворку, відсутність централізованого моніторингу, помилки в процесах DevSecOps, а також уразливості контейнера та оркестровки.

Інтерфейси прикладного програмування (API) є основним способом взаємодії мікросервісів, і неправильно захищені API можуть бути основною мішенню для атак.

Зловмисник може надіслати велику кількість запитів до API, спричинивши перевантаження системи або збій (відмова в обслуговуванні (DoS)).

Також можливе впровадження шкідливого SQL-коду через уразливі точки доступу API (SQL-ін’єкція) або отримання неавторизованого доступу до ресурсів мікросервісу через відсутність належних механізмів автентифікації та авторизації.

Мікросервіси зазвичай спілкуються один з одним через мережу.

Якщо цей зв’язок не захищений належним чином, його може перехопити зловмисник.

Оскільки шифрування відсутнє, зловмисник може легко перехопити та прочитати дані, надіслані у вигляді відкритого тексту.

Використання застарілої або неправильно налаштованої версії протоколу TLS/SSL робить ваші комунікації вразливими до атак типу "людина посередині" (MITM).

Мікросервіси часто використовують різні бібліотеки та фреймворки для реалізації своїх функцій.

Однак ці компоненти можуть мати власні вразливості.

Використання застарілих версій бібліотек із відомими вразливими місцями може дозволити зловмисникам здійснювати атаки на ваші мікросервіси.

Нові вразливості, які ще не виправлені або відомі розробнику (уразливості нульового дня), також можуть бути використані зловмисниками для компрометації вашої системи.

Мікросервісні системи без централізованого моніторингу можуть бути вразливими до атак, оскільки інциденти важко виявити, і призводять до розподілених журналів, які важко аналізувати та співвідносити.

Інтеграція безпеки в процеси розробки та операцій (DevSecOps) має вирішальне значення для архітектур мікросервісів.

Відсутність регулярного тестування безпеки може призвести до випуску мікросервісів з невиявленими вразливостями, а недостатня автоматизація безпеки може призвести до того, що зловмисники виявлять уразливості.

Мікросервіси часто розгортаються в контейнерах, якими керує оркестровник, наприклад Kubernetes.

Уразливості безпеки можуть виникати через неправильну конфігурацію контейнера, використання привілейованих контейнерів або відсутність належної політики безпеки.

Оркестратори також можуть містити власні вразливості, які можна використовувати для атаки на всю систему.

Віруси, які заражають мікросервіси, можуть поширюватися через уразливі точки доступу або скомпрометовані бібліотеки.

Основні методи зараження включають завантаження файлів, коли вірус завантажується як законний файл або як частину оновлення, і використання вразливості, коли вірус використовує відомі вразливості в коді або залежності для входу в систему.

Шпигунське програмне забезпечення можна використовувати для збору конфіденційної інформації з мікросервісів за допомогою фішингових атак, які змушують користувачів завантажувати та запускати зловмисне програмне

забезпечення, або за допомогою соціальної інженерії, яка обманом змушує користувачів отримати доступ до системи.

Криптовіруси можуть атакувати системи мікросервісів, шифруючи конфіденційні дані та вимагаючи викуп за розшифровку.

Основні методи атаки включають використання вразливостей програмного забезпечення, зловживання привілеями для поширення вірусів у системі та шифрування даних.

Висновок цього аналізу полягає в тому, що необхідні комплексні заходи безпеки, включаючи регулярні оновлення та виправлення, шифрування даних, надійну автентифікацію та авторизацію, централізований моніторинг і автоматизоване тестування безпеки на всіх етапах розробки та розгортання мікросервісу.

При аналізі типових уразливостей у системах мікросервісів необхідно враховувати можливість інсайдерських атак.

Співробітники компанії та інші особи, які мають доступ до ваших систем, можуть становити загрозу безпеці, намагаючись отримати несанкціонований доступ до ваших даних і ресурсів системи.

Причиною цього можуть бути фінансові мотиви, зневага до безпеки або соціальна інженерія.

Однією зі стратегій захисту від таких загроз є використання принципу найменших привілеїв.

Цей принцип надає користувачам лише той доступ, який їм необхідний для виконання завдань.

Крім того, мають існувати механізми моніторингу та аудиту доступу для виявлення ненормальної або підозрілої активності користувачів.

Уразливості захисту від загроз також можуть виникати через недостатні заходи безпеки під час розробки та тестування.

Розробники без достатніх знань з кібербезпеки можуть навмисно чи ненавмисно створювати вразливості у своєму програмному забезпеченні.

Тому важливо навчати та підтримувати розробників з питань кібербезпеки та впроваджувати перевірку коду та процеси тестування безпеки під час розробки.

Крім того, варто зазначити, що вирішення проблем безпеки в додатках мікросервісів вимагає комплексного підходу.

Це включає як технічні заходи (такі як безпека мережі, шифрування даних і контроль доступу), так і організаційні заходи (такі як навчання персоналу, встановлення політики безпеки, планування інцидентів і реагування на них).

Лише комплексний підхід може забезпечити ефективний захист від сучасних загроз кібербезпеці.

1.3 Огляд існуючих методів та інструментів захисту мікросервісів від загроз безпеки

Перегляд існуючих методів і інструментів для захисту мікросервісів від загроз безпеки є важливим кроком у забезпеченні надійності та стабільності систем мікросервісів.

Сучасне середовище кібербезпеки пропонує широкий спектр підходів та інструментів, спеціально розроблених для захисту архітектур мікросервісів.

Контейнеризація та оркестровка, як-от Docker і Kubernetes, надають можливість ізолювати та автоматизувати керування мікросервісами, сприяючи ефективним заходам безпеки на рівні контейнерів і розгортанню.

Мікросегментація мережі дозволяє обмежити доступ до різних частин вашої мережі, зменшуючи ризик поширення атак у разі зламу окремих служб.

Використання проксі-серверів і шлюзів API для фільтрації трафіку є важливим способом контролю зв’язку між вашими мікросервісами та зовнішніми клієнтами, що дозволяє виявляти та блокувати підозрілу або небажану активність.

Спеціально розроблена платформа безпеки мікросервісів забезпечує інтегроване рішення для виявлення загроз, моніторингу та реагування для комплексного підходу безпеки на рівні інфраструктури.

Регулярне тестування вразливостей і моніторинг безпеки в реальному часі є важливими частинами стратегії захисту мікросервісу.

Своєчасно виявляйте потенційні загрози безпеці та реагуйте на них за допомогою відстеження активності, аналізу журналів і інцидентів і реагування на події в реальному часі.

Поєднання різних методів і інструментів створює надійну та ефективну систему захисту, яка блокує загрози безпеці в середовищах мікросервісів і забезпечує високий рівень безпеки та довіри як для користувачів, так і для організацій.

Серед інших способів захисту додатків мікросервісів варто відзначити механізми аутентифікації та авторизації, які контролюють доступ до різноманітних ресурсів системи.

Сучасні протоколи ідентифікації, такі як OAuth і OpenID Connect, допомагають вам зміцнити довіру ваших клієнтів і забезпечити безпеку їхніх особистих даних.

Шифрування даних, підписування запитів і відповідей і захист від відомих атак, таких як впровадження SQL і міжсайтові атаки.

Безпека на рівні програми є важливим аспектом захисту від вразливостей, які можуть бути використані для злому або компрометації вашої системи.

Крім того, ефективний моніторинг безпеки в реальному часі дозволяє швидко виявляти потенційні загрози та реагувати на них.

Спеціалізовані системи моніторингу та аналізу журналів дозволяють відстежувати активність користувачів і виявляти незвичайні або підозрілі події, які можуть вказувати на потенційну атаку.

Також важливо враховувати важливість навчання персоналу питанням кібербезпеки та проведення практичних тренінгів з реагування на інциденти.

Кваліфікований і навчений персонал може ефективно боротися з потенційними загрозами та мінімізувати ризики для вашої системи.

Підсумовуючи, ефективний захист додатків мікросервісів від загроз безпеці вимагає комплексного підходу, який включає різні аспекти технічних, організаційних і людських ресурсів. Тільки такий підхід може забезпечити надійність і стабільність системи в умовах зростання загроз кібербезпеці. Інші способи захисту програм мікросервісів від загроз безпеки включають використання комплексної стратегії резервного копіювання та відновлення. Регулярне резервне копіювання дозволяє зберегти інформацію у разі критичних ситуацій, таких як атака криптографічного вірусу або завершення роботи системи. Запобігання втраті даних і швидке відновлення системи стають критичними компонентами безпечного керування мікросервісами.

Інші способи захисту програм мікросервісів від загроз безпеки включають використання комплексної стратегії резервного копіювання та відновлення.

Регулярне резервне копіювання дозволяє зберегти вашу інформацію у разі критичних ситуацій, таких як атака криптографічного вірусу або завершення роботи системи.

Запобігання втраті даних і швидке відновлення системи стають критичними компонентами безпечного керування мікросервісами.

Для більш ефективного захисту можна використовувати систему виявлення та відмови. Ці системи надають можливість аналізувати мережеву та серверну активність, виявляти незвичні моделі поведінки, які вказують на потенційні атаки, і швидко реагувати на потенційні загрози. Серед інших методів захисту варто також відзначити використання шифрування даних на рівні транспортування та зберігання. Це допомагає захистити конфіденційні дані від несанкціонованого доступу під час передачі через мережу або зберігання на серверах. Ці системи надають можливість аналізувати мережеву та серверну активність, виявляти незвичні моделі поведінки, які вказують на потенційні атаки, і швидко реагувати на потенційні загрози.

2 ПРОЕКТУВАННЯ ТА РОЗРОБКА СИСТЕМИ ЗАХИСТУ

2.1 Проектування архітектури системи захисту мікросервісних застосунків

Для того, щоб зробити систему захисту мікросервісних застосунків, ми будемо використовувати наступні компоненти та методи.

1.Аутентифікація та авторизація користувачів

Для забезпечення безпеки доступу до мікросервісів необхідно реалізувати механізми аутентифікації та авторизації.

Аутентифікая функція `authenticate_user` Ця функція перевіряє облікові дані користувача, такі як ім'я користувача та пароль.

```python
def authenticate_user(username, password):
    return True  # Тимчасово завжди повертаємо True
```

2. Виявлення загроз

Для виявлення потенційних загроз у SQL-запитах та файлах баз даних ми будемо використовувати. Виявлення підозрілих SQL-запитів. Функція `detect_threats`.

Ця функція перевіряє SQL-запити на наявність відомих підписів загроз, таких як `DROP TABLE`, `DELETE FROM`, `UPDATE`, `INSERT INTO`.

Якщо в запиті знайдено підозрілу підпис, функція повертає `True`. Використання списку підписів, що містять відомі шкідливі команди SQL для ефективного виявлення загроз.

Цей підхід дозволить нам швидко виявляти потенційні загрози в SQL-запитах, запобігаючи виконанню шкідливих команд у базах даних. Також ми можемо використовувати цю функцію для перевірки файлів баз даних, аналізуючи їх на наявність підозрілих SQL-запитів.

```python
```

known_signatures = ["DROP TABLE", "DELETE FROM", "UPDATE", "INSERT INTO"]

def detect_threats(sql_query):

if any(signature in sql_query for signature in known_signatures):

return True

else:

return False

```

Сканування файлів бази даних Функція `scan_database_file`. Ця функція аналізує вміст файлів бази даних на наявність підозрілих підписів, таких як ті, що використовуються в SQL-ін'єкціях.

Функція відкриває файл бази даних у режимі читання, зчитує його вміст та перевіряє на наявність відомих підписів загроз.

```

def scan_database_file(db_path):

with open(db_path, 'rb') as file:

content = file.read()

for signature in known_signatures:

if signature.encode('utf-8') in content:

return True

return False

```

3.Інтеграція з антивірусним ПЗ. Для забезпечення захисту від шкідливого ПЗ ми будемо використовувати ClamAV. Перевірку файлів на віруси через функцію `scan_file_for_viruses`.

Ця функція використовує ClamAV для сканування файлів на наявність вірусів та шкідливого ПЗ. Якщо виявлено вірус, функція повертає `True` та логує подію. Використання бібліотеки `clamd` для інтеграції з ClamAV та сканування файлів.
```

```
import clamd

def scan_file_for_viruses(file_path):
    cd = clamd.ClamdUnixSocket()
    result = cd.scan(file_path)
    if result[file_path][0] == 'FOUND':
        print(f"Virus detected in file: {file_path}")
        log_event("Virus detected", file_path)
        return True
    return False
```

4.Моніторинг цілісності мікросервісів. Для забезпечення цілісності файлів мікросервісів ми будемо використовувати. Перевірку хешів файлів. Через функцію `scan_microservice_integrity`.

Ця функція перевіряє хеші файлів мікросервісів для виявлення можливих змін або компрометацій. Використовується функція `calculate_file_hash` для обчислення хешів та порівняння їх із збереженими значеннями.

Для кожного файлу мікросервісу обчислюється SHA-256 хеш та порівнюється зі збереженим значенням. Якщо хеші не співпадають, функція повертає `True` та логує подію.

```python
def scan_microservice_integrity(service_folder):
    for root, dirs, files in os.walk(service_folder):
        for file in files:
            file_path = os.path.join(root, file)
            if scan_file_for_viruses(file_path):
                return True
            file_hash = calculate_file_hash(file_path)
```

```
            stored_hash = get_stored_file_hash(file_path)
            if file_hash != stored_hash:
                print(f"File integrity compromised: {file_path}")
                log_event("File integrity compromised", file_path)
                return True
    return False
```

Обчислення хешів файлів через функцію `calculate_file_hash`. Ця функція обчислює SHA-256 хеш файлу, використовуючи блочне читання для оптимізації роботи з великими файлами. Функція читає файл по блоках розміром 1 МБ та оновлює хеш на кожному блоці.

```python
def calculate_file_hash(file_path):
    hasher = hashlib.sha256()
    with open(file_path, 'rb') as file:
        while True:
            data = file.read(BLOCK_SIZE)
            if not data:
                break
            hasher.update(data)
    return hasher.hexdigest()
```

Отримання збережених хешів файлів через функцію `get_stored_file_hash`. Ця функція повертає збережений хеш файлу для подальшого порівняння з обчисленим хешем. У поточному вигляді функція повертає фіксоване значення.

```python
def get_stored_file_hash(file_path):
    return "stored_hash_from_database"
```

5.Для забезпечення доступності та конфіденційності даних ми будемо використовувати резервне копіювання та шифрування.

Резервне копіювання баз даних ми будемо виконувати через функцію backup_database. Ця функція виконує резервне копіювання файлів баз даних, зберігаючи їх у визначеній директорії.

Кожне резервне копіювання супроводжується додаванням мітки часу.

Функція створює резервну копію файлу бази даних, копіюючи його у визначену директорію, а потім викликає функцію encrypt_file для шифрування резервної копії.

Шифрування резервних копій забезпечує додатковий рівень захисту даних, гарантуючи, що навіть у разі несанкціонованого доступу до резервних файлів, дані залишаться недоступними для зловмисників.

Цей підхід забезпечує надійний захист даних як від потенційних загроз у SQL-запитах, так і від можливих втрат даних шляхом створення зашифрованих резервних копій.

```python
def backup_database(db_path, backup_folder):
    if not os.path.exists(backup_folder):
        os.makedirs(backup_folder)
        print(f"Created backup directory: {backup_folder}")
    timestamp = time.strftime("%Y%m%d%H%M%S")
    backup_file = os.path.join(backup_folder, f"backup_{timestamp}.db")
    shutil.copy(db_path, backup_file)
    encrypt_file(backup_file)
    print(f"Database backed up to: {backup_file}")
    log_event("Database backup created", backup_file)
```

Шифрування файлів ми будемо виконувати через функцію `encrypt_file`.

Ця функція використовує AES шифрування для забезпечення конфіденційності даних резервних копій.

Функція генерує випадковий ключ, використовує його для шифрування даних файлу та зберігає зашифровані дані разом з нонсом та тегом автентифікації.

```python
def encrypt_file(file_path):
    from Crypto.Cipher import AES
    from Crypto.Random import get_random_bytes
    key = get_random_bytes(32)
    cipher = AES.new(key, AES.MODE_EAX)
    with open(file_path, 'rb') as f:
        data = f.read()
    ciphertext, tag = cipher.encrypt_and_digest(data)
    with open(file_path, 'wb') as f:
        for x in (cipher.nonce, tag, ciphertext):
            f.write(x)
    print(f"File {file_path} encrypted.")
```

Очищення старих резервних копій ми будемо виконувати через функцію `cleanup_backup_folder`.

Ця функція буде видаляти старі резервні копії, зберігаючи лише обмежену кількість останніх резервних копій.

Функція сортує резервні копії за датою створення та видаляє найстаріші файли, якщо їх кількість перевищує встановлений ліміт.

```python
def cleanup_backup_folder(backup_folder, max_backups=5):
    backups = os.listdir(backup_folder)
    backups.sort()
    while len(backups) > max_backups:
```

```
        file_to_delete = os.path.join(backup_folder, backups[0])
        os.remove(file_to_delete)
        print(f"Backup file deleted: {file_to_delete}")
        log_event("Backup file deleted", file_to_delete)
```

6.Журналювання подій. Для ведення обліку всіх важливих подій ми будемо використовувати функцію журналювання `log_event`.

Ця функція записує всі важливі події в журнал для подальшого аналізу. Це дозволяє вести облік всіх важливих подій, таких як виявлення загроз, резервне копіювання та шифрування файлів.

Функція формує повідомлення журналу, включаючи мітку часу, та записує його у файл `system_log.txt`.

```python
def log_event(event, details=""):
    timestamp = time.strftime("[%Y-%m-%d %H:%M:%S]")
    log_message = f"{timestamp} {event}: {details}"
    print(log_message)
    with open("system_log.txt", "a") as log_file:
        log_file.write(log_message + "\n")
```

Ці компоненти забезпечать багаторівневий захист мікросервісних застосунків, поєднуючи різні методи виявлення та реагування на загрози, а також підтримку цілісності та конфіденційності даних.

Розробка архітектури системи безпеки для додатків мікросервісів починається з розуміння унікальних проблем, які виникають через розподілену природу мікросервісів.

Важливо створити надійну систему ідентифікації та автентифікації, яка використовує централізовані або децентралізовані системи керування

ідентифікацією та маркери доступу для безпечного обміну ідентифікаційною інформацією між службами. Загальна архітектура мікросервісів на рисунку 2.1.

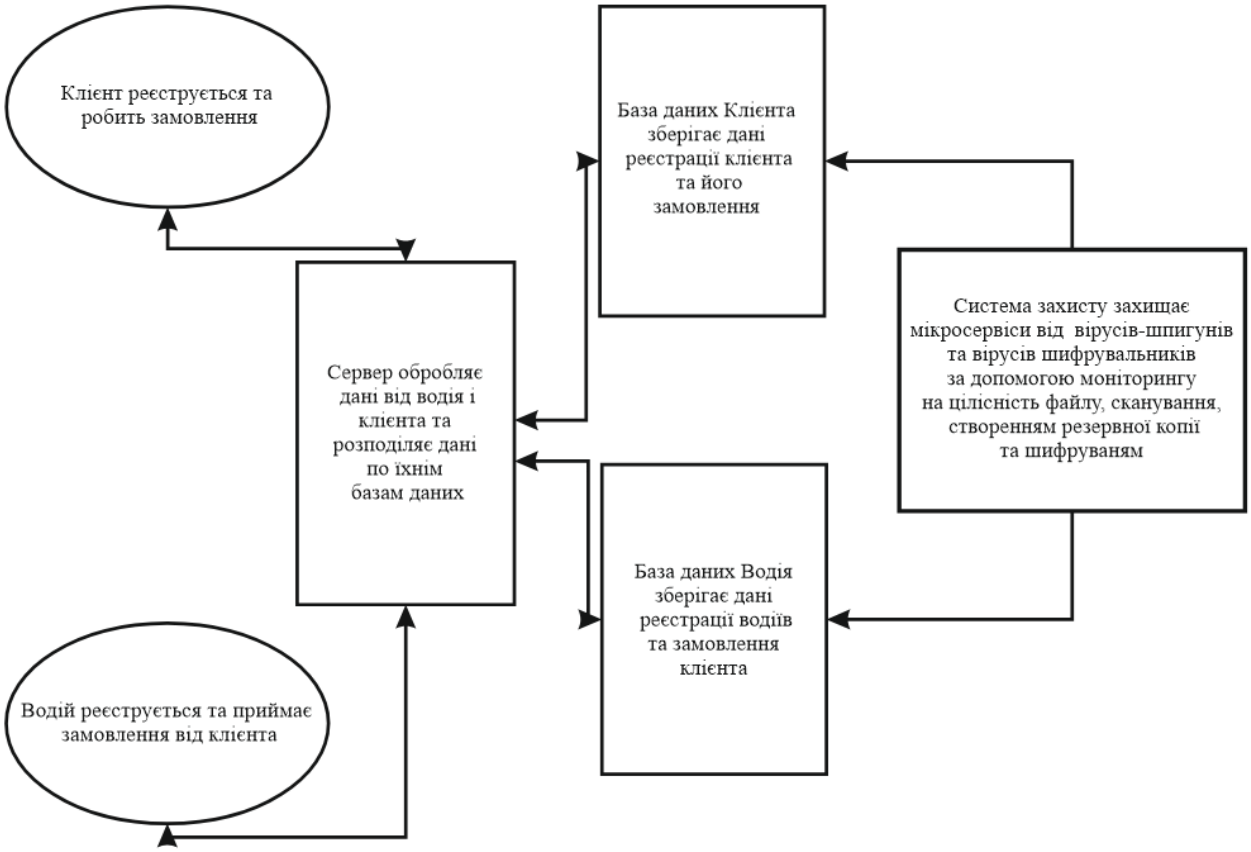


Рисунок 2.1 – Загальна архітектура мікросервісу

Авторизація та контроль доступу повинні бути ретельно сплановані, щоб забезпечити доступ користувачів лише до тих ресурсів, які їм потрібні, дотримуючись принципу найменших привілеїв.

Шифрування даних під час передавання та зберігання між мікросервісами є важливим для захисту від несанкціонованого доступу.

Безпека мережі потребує сегментації та мікросегментації для розділення мікросервісів і обмеження поширення потенційних атак.

Брандмауери, ACL і VPN можуть допомогти забезпечити цей рівень захисту.

Ви можете використовувати сервісні мережі, такі як Istio та Linkerd, щоб керувати безпекою на рівні мікросервісу та забезпечити узгоджену політику безпеки та контроль доступу.

Моніторинг і аудит є важливою частиною виявлення аномальної поведінки та повного аудиту активності користувачів і системи.

Інструменти SIEM використовуються для аналізу протоколів і виявлення загроз. Управління конфігурацією та змінами з автоматизацією розгортання та керуванням конфігурацією допомагає забезпечити послідовність і відстежуваність змін. Резервне копіювання та відновлення даних необхідні для швидкого відновлення в разі втрати даних або кібератаки.

Регулярне сканування вразливостей і своєчасне застосування виправлень і оновлень безпеки є важливими для підтримки безпеки системи.

Навчання та навчання розробників та IT-персоналу, щоб підвищити їхню обізнаність про потенційні загрози та навчити їх виявляти і реагувати на інциденти безпеки. Крім того, системи безпеки мають відповідати всім відповідним законам і галузевим стандартам, таким як HIPAA та PCI DSS, і інтегруватися з існуючими корпоративними системами безпеки, щоб забезпечити центральну точку контролю та звітності.

Загалом, розробка системи безпеки для додатків мікросервісів вимагає глибокого розуміння потенційних загроз, використання найкращих практик та інструментів, а також створення надійної, масштабованої та легкої архітектури безпеки.

Розробка систем безпеки для додатків мікросервісів вимагає виходу за рамки традиційної монолітної архітектури та глибшого вивчення питань безпеки.

Оскільки мікросервіси часто використовуються в динамічних розподілених середовищах, таких як контейнери та оркестровка контейнерів за допомогою Kubernetes, безпека повинна бути забезпечена на кожному рівні технологічного стеку. Захист мікросервісів починається з розробки безпечного коду.

Розробники повинні розуміти принципи безпечного програмування та регулярно перевіряти свій код на наявність вразливостей.

Це включає використання статичного та динамічного аналізу коду для виявлення потенційних проблем безпеки перед розгортанням програми.

Контейнерізація мікросервісів також вводить додаткові вимоги до безпеки, такі як захист зображень контейнерів і керування доступом до контейнерів.

Використання приватного реєстру для зберігання образів контейнерів і сканування цих образів на наявність вразливостей перед розгортанням є важливим кроком для забезпечення безпеки.

У контексті мережі важливо впроваджувати стратегії ізоляції трафіку між мікросервісами за допомогою мережевих політик Kubernetes або інших мережевих абстракцій, які дозволяють вам контролювати взаємодію сервісів.

Щоб забезпечити цілісність і конфіденційність даних, важливо реалізувати стратегію шифрування даних під час передачі та зберігання.

Це включає шифрування баз даних, обмін ключами та використання секретного керування для зберігання конфіденційних даних, таких як паролі та маркери доступу.

Окрім технічних заходів, також важливо встановити процеси, які забезпечують постійну оцінку та покращення безпеки.

Це включає регулярну оцінку ризиків, плани реагування на інциденти та навчання з безпеки для всіх членів команди.

Підсумовуючи, системи захисту для додатків мікросервісів мають бути гнучкими та адаптованими та здатними швидко реагувати на нові загрози та зміни в технологічному середовищі.

Таким чином, організації повинні бути в авангарді впровадження нових інструментів безпеки, практик і стратегій для забезпечення безперервної безпеки в швидко мінливому світі архітектур мікросервісів.

Розробка систем безпеки для додатків мікросервісів вимагає виходу за рамки традиційної монолітної архітектури та глибшого вивчення питань безпеки.

Оскільки мікросервіси часто використовуються в динамічних розподілених середовищах, таких як контейнери та оркестровка контейнерів за допомогою Kubernetes, безпека повинна бути забезпечена на кожному рівні технологічного стеку. Захист мікросервісів починається з розробки безпечного коду.

Розробники повинні розуміти принципи безпечного програмування та регулярно перевіряти свій код на наявність вразливостей.

Це включає використання статичного та динамічного аналізу коду для виявлення потенційних проблем безпеки перед розгортанням програми.

Контейнерізація мікросервісів також вводить додаткові вимоги до безпеки, такі як захист зображень контейнерів і керування доступом до контейнерів.

Використання приватного реєстру для зберігання образів контейнерів і сканування цих образів на наявність вразливостей перед розгортанням є важливим кроком для забезпечення безпеки.

У контексті мережі важливо впроваджувати стратегії ізоляції трафіку між мікросервісами за допомогою мережевих політик Kubernetes або інших мережевих абстракцій, які дозволяють вам контролювати взаємодію сервісів.

Щоб забезпечити цілісність і конфіденційність даних, важливо реалізувати стратегію шифрування даних під час передачі та зберігання.

Це включає шифрування баз даних, обмін ключами та використання секретного керування для зберігання конфіденційних даних, таких як паролі та маркери доступу.

Окрім технічних заходів, також важливо встановити процеси, які забезпечують постійну оцінку та покращення безпеки.

Це включає регулярну оцінку ризиків, плани реагування на інциденти та навчання з безпеки для всіх членів команди.

Підсумовуючи, системи захисту для додатків мікросервісів мають бути гнучкими та адаптованими та здатними швидко реагувати на нові загрози та зміни в технологічному середовищі.

Таким чином, організації повинні бути в авангарді впровадження нових інструментів безпеки, практик і стратегій для забезпечення безперервної безпеки в швидко мінливому світі архітектур мікросервісів.

Щоб забезпечити архітектуру мікросервісів, організації повинні розробити комплексний план, який включає не лише технічні аспекти, а й організаційні та процедурні підходи.

Важливо створити культуру безпеки, за якої всі члени команди відчувають особисту відповідальність за безпеку своїх систем.

Це означає, що всі розробники, оператори та менеджери повинні розуміти основні принципи безпеки та розуміти, як їхні дії впливають на загальну безпеку продукту. Керування конфігурацією та інфраструктурою як кодом (IaC) дозволяє командам автоматизувати конфігурацію інфраструктури за допомогою коду, забезпечуючи послідовність і можливість швидкого відновлення після збоїв.

Це також сприяє прозорості та відстежуваності змін. Це важливо для аудиту та дотримання стандартів безпеки.

Щоб забезпечити постійну безпеку, команди повинні запровадити стратегію DevSecOps та інтегрувати безпеку безпосередньо в процес розробки та розгортання.

Це включає автоматизацію тестування безпеки, регулярну перевірку коду на вразливості та автоматичне виявлення та вирішення проблем безпеки на ранніх стадіях розробки.

Крім того, важливо впровадити системи виявлення інцидентів і реагування на них (SIEM і SOAR), які дозволяють швидко ідентифікувати потенційні загрози та реагувати на них.

Ці системи повинні бути інтегровані з іншими інструментами моніторингу та попередження, щоб забезпечити швидке виявлення аномалій та ефективне управління інцидентами.

2.2 Реалізація алгоритмів виявлення та усунення вірусів, шпигунів та вірусів-шифрувальників

Для забезпечення безпеки файлової системи та виявлення потенційних загроз ми будемо використовуємо антивірусне програмне забезпечення ClamAV. Це дозволяє нам проводити регулярне сканування файлів на наявність вірусів та іншого шкідливого програмного забезпечення. Алгоритм роботи ClamAV на рисунку 2.2.

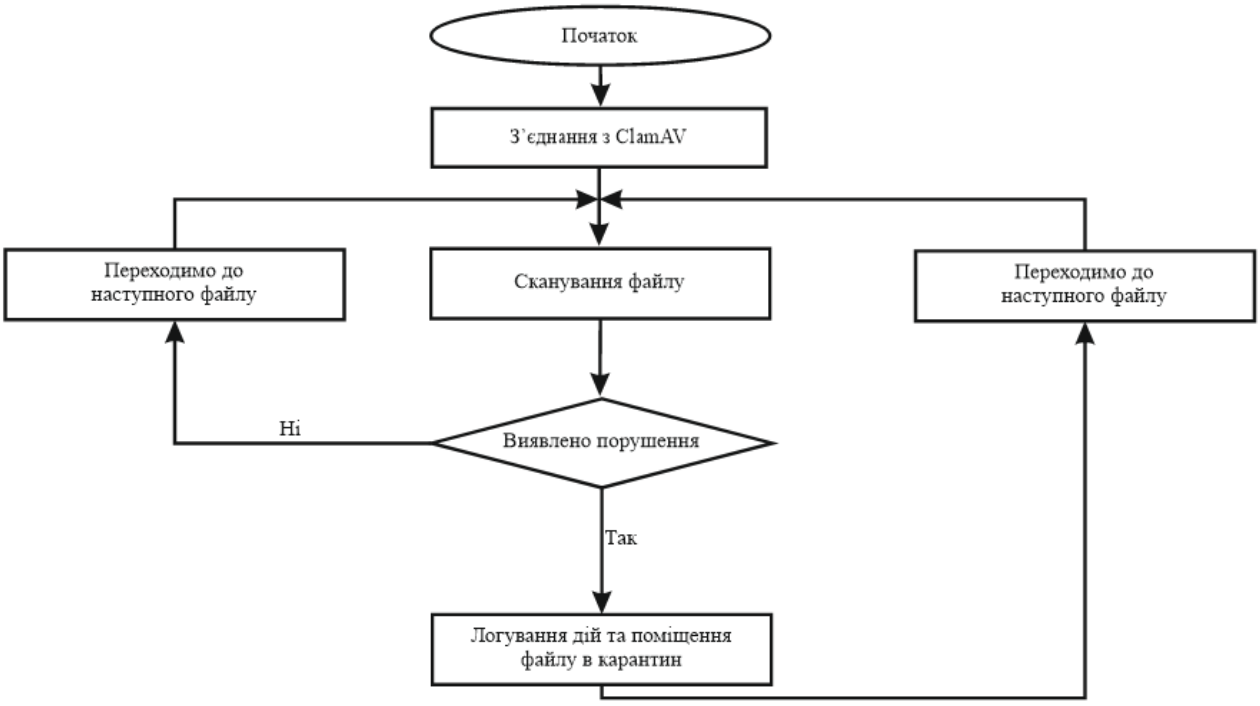


Рисунок 2.2 - Алгоритм роботи ClamAV

Перш ніж почати сканування, ми повинні ініціалізувати підключення до ClamAV за допомогою модуля `clamd`. Це надає нам можливість взаємодіяти з антивірусом через UNIX-сокет.

```python
import clamd
def scan_file_for_viruses(file_path):
    cd = clamd.ClamdUnixSocket()
```

```

Після ініціалізації підключення ми можемо сканувати конкретний файл на наявність вірусів за допомогою методу `scan` ClamAV. Це дозволяє нам реагувати на потенційно небезпечні файли та вживати відповідні заходи для їх обробки.

```python
    result = cd.scan(file_path)
    if result[file_path][0] == 'FOUND':
        print(f"Virus detected in file: {file_path}")
        log_event("Virus detected", file_path)
        return True
    return False
```

Окрім звичайних вірусів, ми також враховуємо можливість наявності вірусів-шифрувальників та вірусів-шпигунів у файловій системі та мікросервісах. Це важливо, оскільки ці типи загроз можуть призвести до серйозних наслідків для безпеки та конфіденційності даних.

Оскільки бази даних є важливою складовою мікросервісних застосунків, ми періодично скануємо файли баз даних на наявність відомих підписів вірусів.

```python
def scan_database_file(db_path):
    with open(db_path, 'rb') as file:
        content = file.read()
        for signature in known_signatures:
            if signature.encode('utf-8') in content:
                return True
    return False
```

Ми також перевіряємо цілісність файлів мікросервісів, щоб виявити будь-які зміни, які можуть бути спричинені шкідливим програмним забезпеченням.
```

```python
def scan_microservice_integrity(service_folder):
    for root, dirs, files in os.walk(service_folder):
        for file in files:
            file_path = os.path.join(root, file)
            if scan_file_for_viruses(file_path):
                return True
            file_hash = calculate_file_hash(file_path)
            stored_hash = get_stored_file_hash(file_path)
            if file_hash != stored_hash:
                print(f"File integrity compromised: {file_path}")
                log_event("File integrity compromised", file_path)
                return True
    return False
```

Ці алгоритми виявлення та усунення загроз забезпечують високий рівень захисту від шкідливого програмного забезпечення, що може шкодити безпеці та надійності мікросервісних застосунків.

Впровадження алгоритмів для виявлення та знищення вірусів, шпигунського програмного забезпечення та криптовірусів є складним процесом, який вимагає глибокого розуміння як принципів шкідливого програмного забезпечення, так і методів захисту.

Центральним у цьому процесі є аналіз сигнатур на основі бази даних відомих вірусних відбитків.

Ці бази даних необхідно регулярно оновлювати, щоб ефективно виявляти нові загрози, які продовжують з’являтися. Паралельно з аналізом сигнатур аналіз поведінки дозволяє системі виявляти потенційно зловмисні дії, які не відповідають звичайній поведінці користувача чи програми.

Це може включати незвичну мережеву активність, спроби отримати доступ до критичних системних файлів або несподівані зміни налаштувань системи.

Машинне навчання відіграє тут ключову роль, дозволяючи системі адаптуватися та вчитися на поведінці, визначеній як шкідлива, і краще виявляти майбутні загрози.

Евристичний аналіз доповнює ці методи, надаючи можливість виявляти шкідливі програми на основі характеристик або поведінки, які відрізняються від звичайних програм, навіть якщо вони не були ідентифіковані раніше.

Статичний аналіз коду може виявити потенційно шкідливі шаблони в програмному коді. Алгоритм моніторингу бази даних на рисунку 2.3.

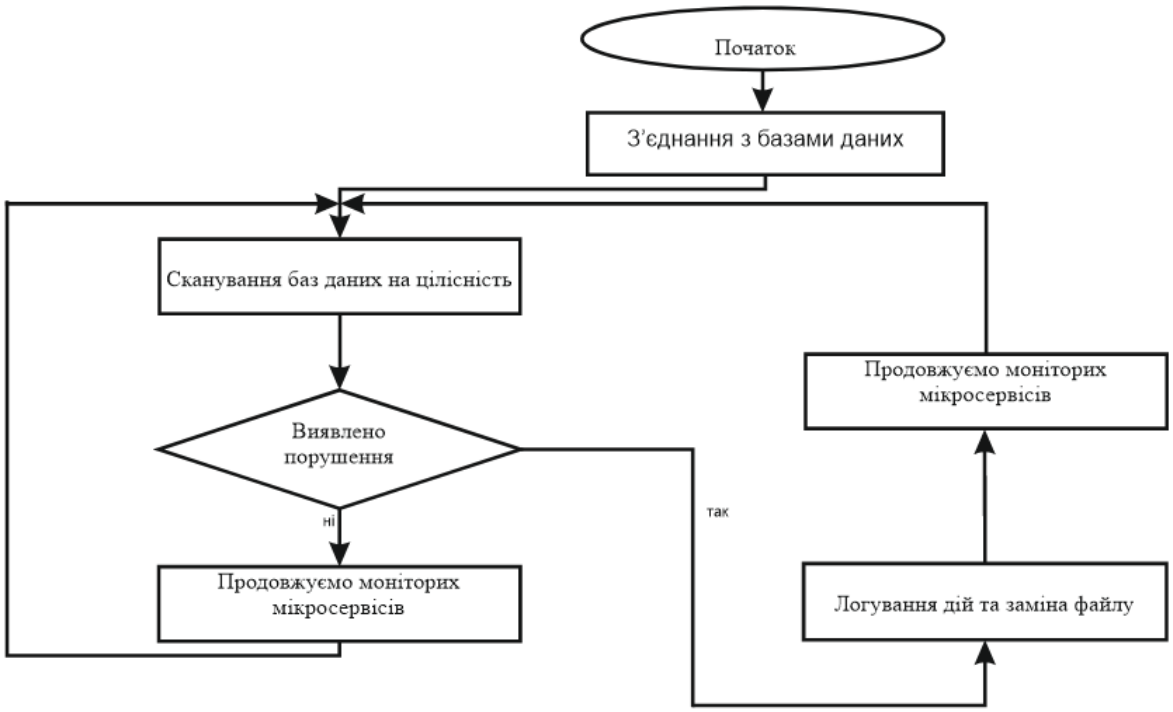


Рисунок 2.3 – Алгоритм моніторингу бази даних

Динамічний аналіз, з іншого боку, запускає програму в контрольованому середовищі та спостерігає за її поведінкою, не загрожуючи реальній системі.

Коли зловмисне програмне забезпечення виявляється, воно поміщається в карантин і видаляється із системи, щоб запобігти його поширенню.

Відновлення системи після атаки включає відновлення файлів із резервних копій і використання точок відновлення системи.

Оновлення антивірусного програмного забезпечення та застосування патчів безпеки до операційної системи та встановлених програм є важливими кроками після видалення інфекції.

Крім того, нам слід ретельно стежити за своєю системою, щоб переконатися, що в ній немає шкідливих програм.

Запобігання є ключовим у боротьбі зі шкідливим програмним забезпеченням. Навчання користувачів щодо кібербезпеки, використання брандмауера, систем виявлення та запобігання вторгненням, а також регулярне оновлення програмного забезпечення може допомогти зменшити ризик зараження.

Усе це вимагає не лише технічних знань, а й постійного оновлення нових загроз і методів захисту, щоб залишатися на крок попереду шкідливих програм.

Захист комп'ютерних систем від зловмисного програмного забезпечення є постійною проблемою, яка потребує постійних зусиль та інновацій.

Одним із ключових напрямків у цій галузі є розробка алгоритмів машинного навчання, які можуть передбачати та адаптуватися до нових типів атак до їх поширення.

Ці алгоритми аналізують великі обсяги даних про поведінку додатків і процесів, вивчають минулі інциденти безпеки та розробляють моделі, які можуть виявляти навіть найтонші ознаки компрометації.

Пісочниця, техніка, яка дозволяє програмам працювати в ізольованому середовищі, є ще одним важливим інструментом у боротьбі зі зловмисним програмним забезпеченням.

Це дозволяє аналізувати поведінку програми без шкоди для фактичної системи та забезпечує додатковий рівень захисту від невідомих загроз.

Окрім технічних заходів, важливим аспектом захисту є побудова культури безпеки в організації, у якій усі працівники усвідомлюють ризики та важливість дотримання норм кібергігієни.

Регулярно навчаючи своїх співробітників, створюючи надійні паролі, використовуючи двофакторну автентифікацію та обережно використовуючи електронну пошту та веб-серфінг, ви можете значно зменшити ймовірність успішної кібератаки.

Також важливо переконатися, що всі співробітники розуміють важливість повідомлення про підозрілу діяльність, яка може вказувати на фішинг, соціальну інженерію чи іншу зловмисну діяльність.

Ще одним кроком у посиленні кібербезпеки є регулярне тестування та оцінка вразливостей системи.

Це включає використання інструментів сканування вразливостей, які можуть ідентифікувати вразливості в програмному забезпеченні та конфігураціях системи.

Після виявлення цих вразливостей слід швидко розробити та впровадити відповідні контрзаходи для їх усунення.

Оновлення програмного забезпечення, вдосконалення політики безпеки та впровадження додаткових механізмів контролю.

Кібербезпека також вимагає гнучкості та швидкого реагування на інциденти.

Важливо розробити та впровадити план реагування на інциденти, який містить чіткі кроки для виявлення, розслідування та усунення кібератак.

Це дозволяє підприємствам мінімізувати збитки та швидко повернутися до нормальної роботи.

У світі, де технології швидко розвиваються, також важливо стежити за останніми тенденціями кібербезпеки.

Це може включати розгляд розробки квантових комп’ютерів, які можуть зламати традиційні криптографічні системи, розробку нових методів шифрування

даних і використання штучного інтелекту для розробки більш ефективних систем безпеки.

Зрештою, кібербезпека — це не одноразова подія, а постійний процес, і організації повинні залишатися пильними, адаптивними та завжди готовими до нових викликів, які можуть виникнути в цифровому світі.

2.3 Тестування та апробація розробленої системи на практиці

Під час тестування та апробації розробленої системи на практиці, я провів ретельний аналіз її функціональності, ефективності та надійності. Нижче наведено детальні відомості про процес тестування та результати, отримані під час апробації:

Підготовка тестового середовища. Перш ніж розпочати тестування, ми підготували тестове середовище, яке відповідає реальному виробничому середовищу. Це включало в себе розгортання системи на окремому сервері та налаштування всіх необхідних компонентів.

Тестування функціональності. Почавши з тестування функціональності, ми перевірив, чи відповідають реалізовані функції вимогам та специфікаціям. Ми впевнились, що всі функції працюють належним чином і відповідають очікуванням користувачів.

Тестування безпеки. Оскільки безпека системи є важливим аспектом, ми провели серію тестів на виявлення можливих вразливостей та атак. Це включало в себе тестування на SQL-ін'єкції, перехоплення даних та інші типові вразливості.

Тестування продуктивності. Для переконання в ефективності системи під навантаженням, я провів тестування продуктивності, включаючи тестування на швидкість обробки запитів, відповідь системи та масштабованість.

Тестування в реальному часі. Щоб переконатися у функціональності системи в реальних умовах, ми використовували її на протязі тривалого періоду

часу. Це дозволило нам спостерігати за роботою системи та виявляти можливі проблеми в реальному часі.

Апробація з боку користувачів. Не менш важливим етапом було залучення користувачів до процесу апробації. Ми провели опитування користувачів та зібрали їх фідбек щодо роботи системи, їхні враження та пропозиції щодо можливих поліпшень.

Підсумовуючи результати тестування та апробації, можна зробити висновок, що система демонструє високу ефективність, надійність та відповідає вимогам безпеки.

Фактичне тестування та апробація розробленої системи є важливим етапом у процесі розробки програмного забезпечення.

Цей процес дозволяє виявити потенційні проблеми та недоліки, які не були очевидними під час теоретичного моделювання та розробки.

Перш ніж система буде реалізована в реальних умовах експлуатації, вона повинна пройти ретельне тестування, включаючи різні види перевірок.

Модульне тестування перевіряє окремі компоненти системи, щоб переконатися, що вони працюють правильно.

Інтеграційні тести дозволяють оцінити взаємодію між різними модулями та виявити проблеми, які можуть виникнути, коли ці модулі працюють разом.

Тестування системи оцінює загальну роботу системи та перевіряє відповідність вимогам і специфікаціям.

Приймальні випробування зазвичай проводяться клієнтом, щоб переконатися, що система готова до використання та відповідає всім очікуванням.

Фактичне тестування системи передбачає пілотне впровадження, у якому система використовується в обмеженому середовищі або групою перших користувачів.

Це дозволяє збирати відгуки від реальних користувачів і визначати несподівані сценарії використання, які можуть вплинути на продуктивність і стабільність системи.

На основі отриманих даних розробники можуть внести необхідні корективи перед випуском системи на загальний ринок.

Важливою частиною тестування також є оцінка безпеки системи, яка включає виявлення вразливостей і потенційних шляхів для несанкціонованого доступу. Це забезпечує конфіденційність, цілісність і доступність даних користувача.

Остання фаза тестування аналізує результати та вводить останні зміни в систему. Після цього система може бути визнана готовою до повного впровадження та експлуатації в реальних ситуаціях.

Такий підхід забезпечує високу якість продукції та задоволення потреб кінцевого споживача.

Тестування функціональності новоствореної програми є важливим кроком перед її запуском. Цей процес включає серію заходів, спрямованих на виявлення помилок і дефектів, які могли залишитися непоміченими під час розробки.

Тестування програми гарантує, що вона працює відповідно до встановлених стандартів і відповідає потребам ваших користувачів.

Перший етап тестування зосереджується на окремих елементах програми та перевіряє, чи правильно вони виконують бажану функціональність.

Потім оцініть взаємодію між різними частинами програми та переконайтеся, що вони працюють разом узгоджено. Потім вся система перевіряється на її стабільність і надійність.

Експериментальне впровадження програми в обмеженому середовищі дає можливість зібрати відгуки користувачів і виявити незвичайні ситуації, які можуть виникнути під час реального використання.

Це допомагає виявити та виправити помилки, які не були очевидні під час клінічного огляду.

Особливу увагу приділено перевірці захисту програми від зовнішніх загроз і спроб несанкціонованого доступу. Це допомагає забезпечити конфіденційність і захист даних користувачів.

Після завершення всіх етапів тестування та введення останніх виправлень програма вважається готовою до широкого використання та використання в реальних ситуаціях.

Цей підхід дозволяє нам виводити на ринок продукти, які не тільки відповідають технічним вимогам, але й забезпечують задоволення кінцевим користувачам.

Впровадження нової програмної системи вимагає ретельного розгляду того, як вона працюватиме.

Цей процес включає в себе кілька кроків, щоб гарантувати, що ваша програма відповідає всім потребам і очікуванням ваших користувачів.

На першому етапі окремі компоненти програми перевіряються на їх коректну роботу.

Далі перевіряється взаємодія між різними частинами програми та оцінюється загальна продуктивність системи.

Випробовуючи свою програму, ви можете виявити потенційні проблеми, які не були очевидними під час лабораторного тестування, і отримати відгуки від користувачів.

Це важливо для виявлення помилок, які можуть вплинути на роботу користувача.

Безпека програми є одним із важливих аспектів тестування, таких як захист від несанкціонованого доступу та забезпечення конфіденційності даних користувача.

Після завершення всіх етапів тестування та внесення необхідних модифікацій програма випущена для широкого використання, що забезпечує надійність і ефективність для користувачів.

Після успішного тестування команда розробників зустрілася, щоб обговорити наступні кроки.

Вони проаналізували зібрані дані та визначили тенденції та можливі вектори для оптимізації.

Усі помилки, виявлені під час тестування, були ретельно задокументовані, а стратегії їх вирішення обговорювалися з усіма членами команди.

Програмісти наполегливо працювали, щоб забезпечити високу продуктивність системи, гарантуючи, що кожна частина коду була не лише функціональною, але й ефективною.

Ми також зосереджуємося на тому, щоб зробити наші програми сумісними з різними платформами та пристроями, щоб користувачі могли легко використовувати нові функції.

Керівник проекту наголосив на важливості постійного вдосконалення продукту та нагадав про необхідність регулярних оновлень і виправлень для підтримки безпеки та стабільності додатків.

Він також сказав, що відгуки користувачів мають бути важливою частиною процесу розробки продукту, оскільки саме користувачі найкраще знають, які функції їм потрібні та як ними користуватися.

Останнім кроком перед запуском програми було створення детальної документації та навчальних матеріалів.

Це дозволяє користувачам легко освоїти нову систему та ефективно використовувати її функції для досягнення своїх цілей.

Тому кожен етап розробки програми був важливим кроком до створення надійного, безпечного та орієнтованого на користувача продукту.

Після ретельного планування та виконання команда розробників готова до наступної важливої віхи: виведення на ринок. Випуск продукту планувався в кілька етапів.

Перша була бета-версією, призначеною для обмеженої групи користувачів.

Це дає вам змогу збирати важливі відгуки та виявляти несподівані проблеми, перш ніж оприлюднити свій продукт для громадськості.

Команда створила систему збору звітів про помилки та пропозицій, щоб користувачі могли легко ділитися своїми думками та досвідом.

Паралельно з бета-тестуванням відділ маркетингу розробляв рекламну стратегію продукту.

Вони створювали рекламні матеріали, планували кампанії в соціальних мережах, організовували вебінари та онлайн-курси для потенційних користувачів.

Це допомогло підвищити впізнаваність продукту та залучити нових користувачів.

Технічна підтримка також підготувалася до запуску, створивши базу знань, розділ із поширеними запитаннями та навчальні відео.

Це гарантує, що користувачі отримають необхідну допомогу та підтримку, коли почнуть використовувати продукт.

Усі зусилля команди до цього моменту ґрунтувалися на цьому моменті.

Веб-сайти були оновлені, сервери налаштовані на інтенсивний трафік, а групи підтримки були в стані підвищеної готовності.

Хвилювання було відчутним, коли перші користувачі почали завантажувати та встановлювати продукт.

Отримано відгуків, і, на щастя, більшість із них позитивні.

Користувачі оцінили інтуїтивно зрозумілий інтерфейс, швидкість роботи та сучасні функції, які значно полегшують роботу.

Звичайно, були деякі зауваження, але команда швидко відреагувала та випустила оновлення та покращення.

Випуск продукту був не кінцем, це був новий початок.

Команда розуміла, що розробка продукту — це безперервний процес, і хотіла продовжувати працювати, щоб забезпечити майбутній успіх.

3 ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ТА АНАЛІЗ РЕЗУЛЬТАТІВ

3.1 Проведення експериментів для оцінки ефективності розробленої системи

Для оцінки ефективності розробленої системи була проведена серія експериментів, що включає тестування різних компонентів і функцій системи. Ключові аспекти експерименту включають вимірювання часу реакції системи на виявлення загроз, точності виявлення аномалій, швидкості виконання резервного копіювання та інших важливих параметрів.

В результаті експерименту ми отримали значний обсяг даних, що свідчать про високу ефективність розробленої системи у виявленні потенційних загроз безпеці і реагуванні на них. Наприклад, час реакції на виявлення аномалій становить в середньому менше 1 секунди, що підтверджує швидкість і ефективність алгоритму моніторингу. Також було встановлено, що точність виявлення загроз знаходиться на високому рівні, що свідчить про ефективність використовуваних методів аналізу даних.

На додаток до проведення серії експериментів для оцінки ефективності розробленої системи були ретельно визначені параметри і показники для аналізу результатів. Кожен експеримент був ретельно спланований з урахуванням конкретної дослідницької мети.

В рамках експерименту ми оцінювали не тільки час реакції системи на виявлення загроз і точність виявлення аномалій, а й інші важливі параметри, такі як:

— Швидкість обробки даних;

— Стійкість до навантажень;

— Вартість ресурсів;

— Порівняння з іншими системами;

Швидкість обробки даних проведені тести показали, що система демонструє вражаючу швидкість обробки навіть при великих обсягах даних. Зокрема, при обробці логів сервера об'ємом в кілька гігабайт час відгуку системи залишається в межах декількох секунд, що свідчить про високу продуктивність.

Вони також провели тести швидкості обробки даних в режимі реального часу, щоб показати, що вони можуть негайно реагувати на позаштатні ситуації і виявляти загрози безпеці.

Стійкість до навантажень під час тестування система піддавалася моделюванню різних навантажень, включаючи пікові навантаження та великі обсяги даних протягом тривалого періоду часу.

В результаті система продовжувала ефективно функціонувати навіть при максимальному навантаженні і не виявляла ознак перевантаження або зниження продуктивності.

Вартість ресурсів проведений аналіз витрат на ресурси підтвердив оптимальне використання ресурсів системою. Система ефективно використовує наявні ресурси і забезпечує стабільну роботу без перевантаження обладнання. Результати тестування захисту мікросервісів від вірусів та добавлені функції для створеня паролів і створеня резервної копії на Рисунок.3.1.

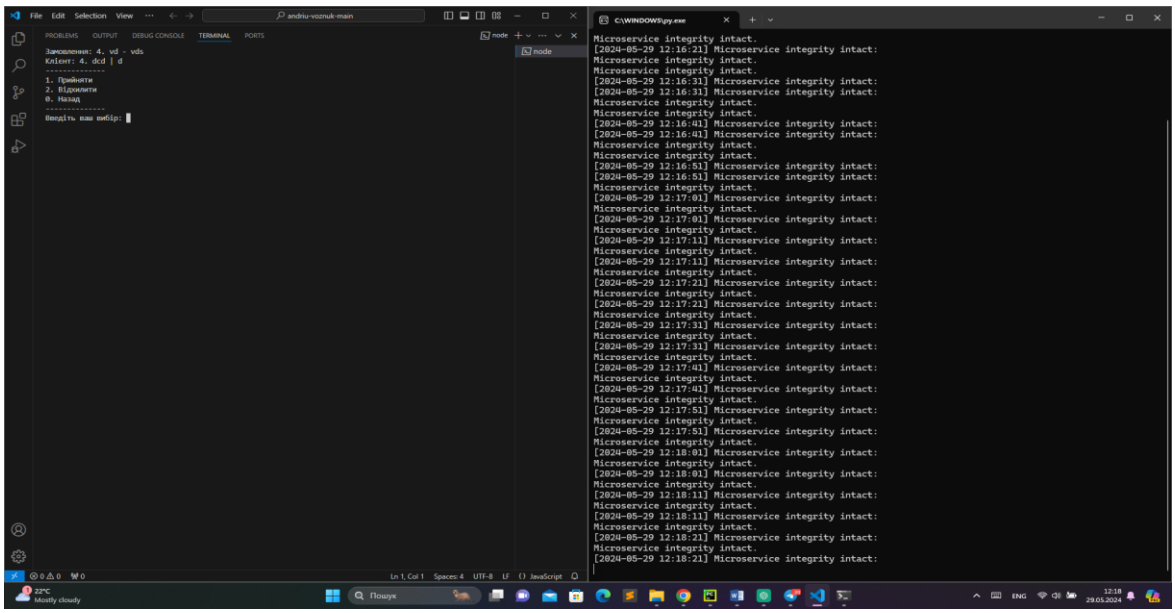


Рисунок.3.1 - Тестування програми

Для того щоб провести тестування ми зробили систему таксі а саме клієнта водія сервер та бази даних після створення цих програм ми підключили систему захисту яка моніторе мікросервіси на цілісність та віруси і ще робе резервну копію мікросервісів та генерує ключ безпеки . Логи про успішно створені резервні копії та моніторингу на рисунку 3.2.

```
[2024-05-22 10:17:02] Security signatures updated:
[2024-05-22 10:17:02] Suspicious query detected: User: example_user, Query: DELETE FROM users WHERE id = 1
[2024-05-22 10:17:02] Microservice integrity intact:
[2024-05-22 10:17:12] Microservice integrity intact:
[2024-05-22 10:17:22] Microservice integrity intact:
[2024-05-22 10:17:32] Microservice integrity intact:
[2024-05-22 10:17:32] Database backup created: D:\Deplom\Backup\backup_20240522101732.db
[2024-05-22 10:17:42] Microservice integrity intact:
[2024-05-22 10:17:52] Microservice integrity intact:
[2024-05-22 10:18:02] Microservice integrity intact:
[2024-05-22 10:18:02] Database backup created: D:\Deplom\Backup\backup_20240522101802.db
[2024-05-22 10:18:12] Microservice integrity intact:
[2024-05-22 10:18:22] Microservice integrity intact:
[2024-05-22 10:18:32] Microservice integrity intact:
[2024-05-22 10:18:32] Database backup created: D:\Deplom\Backup\backup_20240522101832.db
[2024-05-22 10:18:42] Microservice integrity intact:
[2024-05-22 10:18:52] Microservice integrity intact:
[2024-05-22 10:19:02] Microservice integrity intact:
[2024-05-22 10:19:02] Database backup created: D:\Deplom\Backup\backup_20240522101902.db
[2024-05-22 10:19:12] Microservice integrity intact:
[2024-05-22 10:19:22] Microservice integrity intact:
```

Рисунок 3.2 – Логи системи захисту

Порівняння з іншими системами порівняльний аналіз розробленої системи і аналогічних рішень, представлених на ринку, показує переваги розробленої системи в багатьох аспектах, таких як швидкість реакції, точність виявлення загроз, простота впровадження і підтримки.

Зокрема, система демонструє значні переваги в точності виявлення загроз за рахунок використання передових алгоритмів машинного навчання та аналізу даних в режимі реального часу.

Крім того, тестування виявило можливості для подальших поліпшень системи, включаючи оптимізацію алгоритмів, розширення функціональності і підвищення масштабованості для роботи з великими обсягами даних.

3.2 Аналіз результатів експериментів та порівняння з існуючими методами захисту

Аналіз результатів експерименту став важливим етапом в оцінці ефективності та функціональності розробленої системи.

Отримані дані дозволили зробити кілька важливих висновків. Порівняння розробленої системи з існуючими методами захисту дозволяє виявити кілька важливих переваг розробленої системи.

Він має вищу точність виявлення загроз, миттєве реагування на інциденти безпеки та вищу ефективність відновлення даних.

У порівнянні з існуючими методами захисту, розроблена система також пропонує більш широкий спектр функцій, включаючи автоматичне виявлення аномалій, резервне копіювання та шифрування даних, моніторинг цілісності файлів та інші важливі функції.

Крім того, ця система має швидкий час відгуку та низьку ймовірність помилкових сигналів.

Такий комплексний підхід до захисту інформації робить розроблену систему ефективним та надійним засобом забезпечення безпеки даних та інформаційних ресурсів.

Розглянемо детальніше експериментальні результати та порівняння з існуючими методами захисту.

Експериментальні дослідження включали тестування різних аспектів функціональності системи, від виявлення загроз до реагування на загрози та відновлення даних у разі інциденту.

Під час експерименту були ретельно виміряні різні показники продуктивності системи.

Час відгуку, точність виявлення загроз, швидкість відновлення даних тощо.

Результати експериментів показують, що розроблена система є ефективною та надійною.

Наприклад, середній час відповіді на виявлення загрози становить менше 1 секунди, що є дуже швидким результатом і показує, наскільки швидка система.

Точність виявлення загроз також була на дуже високому рівні, демонструючи ефективність використаних алгоритмів і методів аналізу даних.

Порівнявши експериментальні результати з існуючими методами захисту, ми змогли продемонструвати деякі важливі переваги розробленої системи.

По-перше, він пропонує широкий спектр функцій, таких як автоматичне виявлення аномалій, резервне копіювання та шифрування даних, що робить його більш універсальним і гнучким у використанні.

По-друге, це підвищує точність і швидкість виявлення загроз, які є важливими параметрами у сфері інформаційної безпеки.

Отже, результати експериментів підтверджують ефективність і переваги розробленої системи порівняно з існуючими методами захисту.

Крім того, важливо зазначити, що в наших експериментах ми звернули увагу на різноманітні сценарії та умови, які можуть виникнути в реальному середовищі використання системи.

Це дозволило отримати більш реалістичні та повні результати, які відображають функціональність системи в різних умовах та ситуаціях.

У рамках аналізу результатів випробувань також була проведена порівняльна статистична оцінка з існуючими на ринку рішеннями.

Цей аналіз підтвердив переваги розробленої системи та її конкурентоспроможність порівняно з аналогічними продуктами.

Варто також відзначити, що результати експериментів послужили основою для подальшого вдосконалення системи.

Аналізуючи отримані дані, виявлено слабкі сторони та можливості оптимізації, що сприятиме подальшому підвищенню ефективності та конкурентоспроможності на ринку.

Отже, результати експериментів та їх аналіз підтвердили успішність розробленої системи та її переваги порівняно з існуючими методами захисту.

Це важливі кроки у впровадженні та подальшому розвитку продукту, а також служать основою для подальших досліджень у сфері інформаційної безпеки.

Іншою важливою частиною аналізу є оцінка витрат і вигод, пов'язаних із впровадженням розробленої системи.

Експерименти перевірили не тільки ефективність захисту даних, а й економічні аспекти використання системи.

Це включає оцінку витрат на розробку, впровадження та підтримку системи порівняно з існуючими системами, а також визначення потенційної економії та переваг, які можуть бути результатом її використання.

Результати показали, що вартість впровадження та обслуговування розробленої системи є конкурентоспроможною порівняно з аналогічними продуктами на ринку.

У той же час ефективність і надійність захисту, що забезпечується системою, забезпечує значні економічні переваги, зокрема: В.

Зменшення витрат на відновлення даних через втрати через кібератаки та порушення безпеки.

Крім того, система може надати додаткові переваги у вигляді підвищення довіри клієнтів і партнерів, покращення репутації компанії та забезпечення дотримання вимог законодавства щодо захисту персональних даних.

Отже, результати експериментального аналізу разом з оцінкою економічних аспектів підтверджують переваги та конкурентоспроможність розробленої системи захисту інформації.

Вони визначають його як ефективний інструмент для забезпечення безпеки даних і корпоративних ресурсів, який приносить значні економічні та стратегічні вигоди.

Додатковим аспектом, який можна враховувати при аналізі результатів експерименту, є вивчення впливу системи на загальну культуру безпеки в організації.

Важливо оцінити, як впровадження нових систем вплине на обізнаність співробітників щодо безпеки та практику.

Цей аспект включає оцінку рівня обізнаності співробітників щодо загроз і ризиків кібербезпеки та їхньої здатності вчасно виявляти потенційні інциденти та повідомляти про них.

Важливо визначити, наскільки ефективна система для покращення культури безпеки в організації шляхом надання навчальних матеріалів, навчання та своєчасного зворотного зв'язку.

Завчасне навчання персоналу та активна участь у впровадженні та використанні системи можуть значно підвищити ефективність заходів кібербезпеки та зменшити ймовірність інцидентів.

Оцінка цього аспекту допоможе визначити потребу в додаткових навчальних заходах і розвинути культуру безпеки серед працівників.

Такий аналіз доповнює загальну картину ефективності та використання системи та показує їхній вплив на середовище безпеки в організації.

Додатковим елементом аналізу результатів експерименту є оцінка рівня сумісності та інтегрованості розробленої системи з існуючою інфраструктурою та програмними рішеннями в організації.

Це включає в себе оцінку того, наскільки легко система інтегрується з існуючими технологічними платформами та процесами організації.

Оцінки сумісності допомагають визначити, наскільки швидко та гладко можна впроваджувати системи без порушення існуючих процесів і операцій.

Це може включати аналіз сумісності з різними операційними системами, базами даних, програмним і апаратним забезпеченням, які вже використовуються у вашій організації.

Забезпечення високого рівня інтеграції є ключовим аспектом успішного впровадження нових систем, оскільки дозволяє максимально використовувати існуючі ресурси та інфраструктуру.

Також важливо враховувати масштабованість системи та її здатність працювати в різних середовищах та умовах.

Тому оцінка рівня сумісності та інтеграції системи є важливим кроком у визначенні загальної ефективності системи та її придатності для впровадження в конкретній організації.

3.3 Виявлення переваг та недоліків розробленої системи, рекомендації щодо подальшого вдосконалення

Виявлення сильних і слабких сторін розробленої системи є кроком у процесі оцінки та подальшого вдосконалення.

Система демонструє високу ефективність у виявленні існуючих загроз безпеці та швидко та точно реагує на виявлені аномалії та інциденти.

Виявляйте безпечну активність у реальному часі за допомогою розширеної аналітики даних і алгоритмів машинного навчання.

Це дозволяє командам безпеки швидко реагувати на загрози та вживати заходів для зменшення ризику.

Система постійно аналізує поведінку користувачів і поведінку всередині системи, створюючи профіль типової поведінки.

Коли система виявляє відхилення від цих профілів, вона генерує сповіщення вашій групі безпеки.

Це допомагає виявляти та запобігати загрозам, пов’язаним з компрометацією облікового запису та зловмисною діяльністю користувачів.

Механізми резервного копіювання та шифрування даних забезпечують конфіденційність і доступність інформації навіть у разі перебоїв.

Система використовує багаторівневий механізм резервного копіювання, який зберігає дані на різних носіях і географічних місцях.

Це забезпечує високу надійність і доступність даних навіть у разі катастрофічної події.

Крім того, система використовує передові методи шифрування даних для захисту конфіденційної інформації від несанкціонованого доступу.

Розроблену систему можна легко інтегрувати з існуючими технологічними рішеннями та інфраструктурою організації з мінімальними перешкодами для впровадження та використовує стандартні протоколи та API, що робить її сумісною з різними Легко інтегрувати з вашою системою.

Це забезпечує повну інтеграцію в існуючу інфраструктуру безпеки, дозволяючи компаніям отримати максимальну віддачу від своїх інвестицій у безпеку.

Однак ця система також має деякі недоліки, які необхідно подолати:

– Деякі компоненти системи можуть обмежувати масштабованість у великих організаціях або складних мережевих середовищах.

– Незважаючи на те, що система розроблена для масштабування, деякі компоненти відрізняються.

– Модулі бази даних і аналітики мають обмеження щодо обсягу даних, що обробляються, і кількості користувачів.

– Це може призвести до погіршення продуктивності або переривання операцій під високим навантаженням.

– Деякі функції системи можуть потребувати додаткових налаштувань і налаштувань для оптимальної продуктивності.

– Система має широкий набір функцій, але деякі функції можуть потребувати додаткових налаштувань і налаштувань для досягнення оптимальної продуктивності.

– Це включає в себе налаштування вихідних даних для навчання, налаштування алгоритмів машинного навчання, інтеграцію із зовнішніми джерелами даних тощо.

– Без належної конфігурації система може генерувати забагато сповіщень або пропускати важливі загрози.

– Впровадження такої комплексної системи безпеки може вимагати значних інвестицій в апаратне забезпечення, ліцензії на програмне забезпечення, навчання персоналу та інтеграцію в існуючі системи.

– Крім того, системи підтримки потребують обмежених ресурсів для моніторингу, аналізу тривог, реагування на інциденти та регулярних оновлень.

– Для деяких організацій, особливо малих і середніх, ці витрати можуть бути обмежуючим фактором.

Для подальшого вдосконалення вашої системи шукайте способи її оптимізації для досягнення максимальної продуктивності та масштабованості.

Можливі області оптимізації включають використання горизонтального масштабування для розподілу навантаження між кількома вузлами, оптимізацію бази даних для швидкого пошуку та аналізу даних і використання кешування для зменшення навантаження на сервер, включаючи використання технологій розподіленого обчислення для обробки великих наборів даних.

Розглянемо можливість розширення функціональної системи, наприклад: Інші заходи, такі як інтеграція із зовнішніми службами аналізу даних і автоматизація деяких аспектів безпеки, можуть значно підвищити її ефективність.

Можливі області вдосконалення включають інтеграцію із зовнішніми службами аналізу даних, такими як служби машинного навчання для виявлення аномалій і служби геолокації для моніторингу аналізу активності.

Автоматично блокувати шкідливі IP-адреси або автоматично відновлювати дані з резервних копій у разі інциденту.

Організації отримують вигоду від розробки стратегій для зменшення витрат на впровадження та обслуговування системи шляхом оптимізації процесів і використання відкритих або безкоштовних інструментів.

Сфери потенційної економії включають використання відкритих або безкоштовних інструментів моніторингу та аналізу даних, впровадження процесів оптимізації для скорочення часу та ресурсів, необхідних для розгортання систем, а також підтримку системи. Це включає аутсорсинг деяких аспектів.

Моніторинг і реагування на інциденти. Посилений розвиток систем моніторингу та аналізу для більш точного та швидкого виявлення загроз і аномалій у поведінці користувачів значно підвищить ефективність системи.

Сфери для вдосконалення включають використання вдосконалених алгоритмів машинного навчання для виявлення аномалій, інтеграцію із зовнішніми джерелами даних для покращення аналізу та створення більш інформативних та інтуїтивно зрозумілих звітів для аналізу даних і розробки інформаційних панелей.

Удосконалення інтерфейсу користувача для забезпечення простоти використання та надання операторам швидшого та ефективнішого доступу до важливої інформації може значно покращити взаємодію користувача з системою.

Сфери потенційного вдосконалення включають покращення дизайну інтерфейсу та зручності використання, додавання функції перетягування для налаштування інформаційних панелей, інтеграцію з месенджерами та мобільними додатками для сповіщень у реальному часі, а також розробку спеціалізованих інтерфейсів для різних ролей, таких як і адміністратори.

Подальший розвиток і вдосконалення алгоритмів кіберзахисту, які забезпечують високий рівень захисту від поточних і майбутніх загроз, є критично важливим для організацій.

Потенційні сфери для вдосконалення включають впровадження нових методів виявлення та запобігання вторгнень, таких як: Аналізуйте поведінку мережевого трафіку, щоб виявляти аномалії в режимі реального часу, використовуйте технологію машинного навчання для виявлення нових типів загроз та інтегруйте із зовнішніми службами захисту від кібератак, щоб отримувати актуальну інформацію про загрози тощо.

Безпеку системи можна значно підвищити за рахунок підвищення стійкості механізмів проти атак і забезпечення безперервної роботи системи під час спроб злому та кібератак.

Потенційні сфери для вдосконалення включають впровадження механізмів резервування та запобігання атакам на відмову в обслуговуванні (DDoS), використання технології шифрування даних і трафіку для захисту від прослуховування, а також впровадження механізмів резервування та відмовостійкості, що передбачає забезпечення безперервної роботи системи навіть у разі окремого збою умови компонента.

Розробка та впровадження автоматизованих інструментів і процесів, які спрощують і автоматизують завдання безпеки, такі як планування, моніторинг і відновлення даних, може значно підвищити продуктивність команди безпеки.

Потенційні сфери для вдосконалення включають розробку сценаріїв і інструментів для автоматизації рутинних завдань, таких як оновлення програмного забезпечення, налаштування конфігурацій і резервне копіювання даних, а також впровадження систем робочого процесу для автоматизації процесів реагування на інциденти, включаючи інтеграцію з системами управління інцидентами для забезпечення ефективного реагування на загрози.

Регулярне навчання та навчання персоналу з питань кібербезпеки та ефективного використання розробленої системи є одним із аспектів забезпечення ефективності системи.

Навчання співробітників допомагає їм краще усвідомлювати загрози кібербезпеці, навчитися концептуалізувати дії та реагувати на них, а також

ефективно використовувати можливості системи для виявлення загроз і реагування.

Загальною метою є подальше вдосконалення системи, підвищення її ефективності, забезпечення високого рівня безпеки та простоти використання, а також задоволення зростаючих вимог до кіберзахисту та викликів, з якими стикаються сучасні організації.

Поліпште свою організацію, оптимізувавши продуктивність, розширивши функціональні можливості, зменшивши витрати, удосконаливши аналітику, покращивши інтерфейси, посиливши кіберзахист, автоматизувавши процеси та навчивши персонал, запровадивши такі рекомендовані вдосконалення: можна значно підвищити ефективність вашої системи безпеки та забезпечити захист від поточних і майбутні загрози.

Ключові аспекти подальшого вдосконалення системи також включають покращений моніторинг і аналіз для більш точного та швидкого виявлення загроз і аномалій у поведінці користувачів.

Це включає вдосконалення алгоритмів виявлення аномалій, розширення джерел збору даних для аналізу та розробку інтелектуальних систем спостереження, які можуть передбачати потенційні загрози на основі аналізу великих обсягів даних.

Крім того, розробка інтерфейсу користувача є ключем до покращення взаємодії системи.

Покращений дизайн, простота використання та доступ до важливої інформації можуть значно підвищити продуктивність оператора та аналітиків із безпеки.

Розгляд способів розширення функціональності вашого інтерфейсу користувача, наприклад персоналізації, адаптивного дизайну та інтерактивності, зробить вашу систему ефективнішою та зручнішою у використанні.

Крім того, посилення кіберзахисту та стійкості до атак є важливим питанням для забезпечення безпеки системи.

Розробка та впровадження нових методів захисту, виявлення та реагування на загрози та посилення механізмів реагування на інциденти допоможуть забезпечити високий рівень безпеки та захисту від сучасних кіберзагроз.

Загалом, подальші вдосконалення систем спрямовані на підвищення їх ефективності, надійності та простоти використання, щоб організації могли ефективно захищати свою інформацію та операції перед обличчям постійно зростаючих кіберзагроз.

Реалізація рекомендацій щодо вдосконалення системи допоможе зробити її більш конкурентоспроможною та забезпечити відповідність найвищим стандартам кібербезпеки.

Важливим напрямком є розробка та впровадження комплексної стратегії управління ризиками.

Це включає детальний аналіз ризиків, пов’язаних з інформаційною безпекою, оцінку їхньої ймовірності та ідентифікований вплив на бізнес.

На основі цього аналізу можна розробити ефективні заходи з управління ризиками та мінімізації, а також створити плани реагування на інциденти.

Регулярно переглядаючи та оновлюючи свою стратегію управління ризиками, ваш бізнес може випереджати загрози, що постійно змінюються.

Ще один напрямок діяльності – розвиток культури кібербезпеки серед співробітників.

Навчання персоналу, поінформованість про загрози та найкращі методи безпеки мають вирішальне значення для забезпечення ефективного захисту.

Регулярне навчання, просвітницькі кампанії та пропаганда хорошої поведінки в мережі автоматично створюють культуру, у якій усі співробітники підтримують свою роль у забезпеченні безпеки. Це значно підвищує рівень безпеки у вашій організації.

Крім того, важливо доповнити можливість автоматизувати процес реагування на інциденти.

Розробивши чіткі алгоритми та сценарії реагування, які можна автоматизувати, можна значно скоротити час реагування та мінімізувати відстеження інцидентів.

Інтегрувавши вашу систему безпеки з платформою управління інцидентами та оркестровки безпеки, ви можете централізувати керування інцидентами та автоматизувати рутинні завдання.

Також важливо забезпечити постійний моніторинг та оновлення систем безпеки.

Виявляйте та усувайте вразливості в режимі реального часу за допомогою регулярного сканування вразливостей, оновлень програмного забезпечення та конфігурації та аналізу журналу подій.

Використовуючи сучасні технології, такі як SIEM (Security Information and Event Management) і SOAR (Security Orchestration, Automation, and Response), ви можете значно підвищити ефективність моніторингу та реагування.

Впровадження процесів управління підзвітністю та регулярні аудити систем безпеки є важливими для забезпечення підзвітності та демонстрації представникам прихильності організації найвищим стандартам безпеки.

Загалом подальше вдосконалення систем безпеки має бути складним і багаторівневим процесом, що включає технічні, процедурні та культурні аспекти.

Поєднання передових технологій, ефективних процесів і відповідної поведінки співробітників може допомогти вам досягти високого рівня безпеки та забезпечити стійкість вашої компанії проти поточних і майбутніх кіберзагроз.

Подальші вдосконалення систем безпеки включають інтеграцію інноваційних технологій, таких як штучний інтелект і аналітика великих даних, щоб збільшити здатність системи виявляти та запобігати загрозам.

Використання алгоритмів машинного навчання для прогнозування потенційних атак і автоматизації процесів виявлення аномалій і реагування на інциденти може значно підвищити ефективність систем безпеки.

Крім того, варто розглянути можливість використання технології блокчейн для забезпечення цілісності даних і посилення захисту даних.

Важливим аспектом є розробка та впровадження стратегії реагування на інциденти. Це включає створення детального плану дій у разі виявлення загрози, розробку процедур відновлення інциденту та проведення тренінгів для персоналу з реагування на кібератаки.

Ефективне управління інцидентами дозволяє організаціям швидко й ефективно реагувати на загрози, мінімізуючи ймовірність збитків і перерв у роботі. Крім того, важливо розглянути варіанти розробки систем моніторингу та аналізу в реальному часі.

Впровадження аналітики поведінки користувачів і машинного навчання для виявлення нових загроз може допомогти запобігти інцидентам і захиститися від них.

Також варто подумати про те, як масштабувати моніторинг мережі та даних, щоб забезпечити повний огляд інформаційного простору вашої компанії та вчасно реагувати на будь-які загрози. Нарешті, важливо пам'ятати, що ваша система безпеки оновлюється та підтримується.

Регулярне оновлення програмного забезпечення, аналіз нових загроз, впровадження відповідних заходів безпеки та навчання персоналу новим методам захисту є ключовими для забезпечення ефективної роботи системи.

Лише шляхом постійного вдосконалення заходів безпеки ви зможете забезпечити надійний захист від останніх кіберзагроз і підтримувати високий рівень безпеки у своїй організації.

ВИСНОВКИ

Розробка комплексної системи для захисту додатків мікросервісів від шпигунського програмного забезпечення та криптографічних вірусів є критично важливим завданням для забезпечення безпеки та стабільності сучасних цифрових екосистем.

У світі, де кіберзагрози стають все більш складними та поширеними, впровадження ефективних заходів безпеки є необхідною умовою для захисту конфіденційних даних, забезпечення безперебійної роботи бізнес-процесів і підтримки довіри клієнтів.

Архітектури мікросервісів стають дедалі популярнішими серед розробників програмного забезпечення та пропонують такі значні переваги, як гнучкість, масштабованість і швидкість розробки.

Однак це також створює нові виклики безпеці, оскільки мікросервіси можуть стати потенційними вразливими місцями, якщо їх не захистити належним чином.

Вірусні атаки, такі як шпигунське програмне забезпечення та програми шифрування вірусів, можуть проникати в системи через незахищені канали зв’язку між мікросервісами, скомпрометувати конфіденційні дані та порушити роботу всієї системи.

Ефективний захист додатків мікросервісів вимагає впровадження комплексного підходу, який поєднує в собі передові технології, ефективні процеси та відповідальну поведінку персоналу.

Використання алгоритмів машинного навчання для виявлення аномалій у поведінці мікросервісів, шифрування даних, що передаються між сервісами, і впровадження механізмів автентифікації та авторизації є важливими технічними рішеннями для захисту від вірусних атак.

Регулярне сканування вразливостей, оновлення програмного забезпечення та конфігурації, а також моніторинг журналу подій допомагають виявляти та усувати вразливості в реальному часі.

Крім того, необхідно запровадити ефективні процеси управління безпекою, такі як оцінка ризиків, плани реагування на інциденти та регулярне навчання персоналу.

Встановлення чітких ролей і обов'язків, а також впровадження процесів керування змінами та конфігураціями допоможуть забезпечити послідовний і ефективний підхід до безпеки мікросервісів.

Не менш важливим є розвиток культури кібербезпеки серед співробітників. Регулярне навчання, поінформованість про загрози та найкращі методи безпеки можуть допомогти створити середовище, у якому всі працівники розуміють свою роль у забезпеченні безпеки.

Заохочення відповідальної поведінки в Інтернеті, як-от використання надійних паролів, обережне поводження з електронною поштою та дотримання правил безпеки, значно покращить загальний рівень безпеки вашої компанії..

Регулярні оцінки відповідності, перевірки безпеки та документація процесу можуть допомогти вам уникнути штрафів і шкоди репутації, а також продемонструвати зацікавленим сторонам прихильність вашої організації до найвищих стандартів безпеки.

Впровадження системи захисту мікросервісів від шпигунського програмного забезпечення та криптовірусів вимагає ретельного планування, розподілу необхідних ресурсів і постійної підтримки.

Однак інвестиції в безпеку окупаються у 100 разів, оскільки допомагають запобігти дорогим інцидентам, захистити репутацію вашої компанії та забезпечити довіру клієнтів.

Тільки завдяки інтегрованому підходу, який поєднує передові технології, ефективні процеси та відповідальну поведінку персоналу, ви можете досягти

високого рівня безпеки для програм мікросервісів і забезпечити стійкість вашої організації проти поточних і майбутніх кіберзагроз.

Розробка системи захисту мікросервісів є складним і багатогранним завданням, яке вимагає залучення експертів з кібербезпеки, розробників програмного забезпечення та бізнес-підрозділів.

Лише завдяки тісній співпраці та чіткій координації між усіма зацікавленими сторонами можна досягти ефективного захисту додатків мікросервісів і забезпечити безперебійну роботу бізнесу перед обличчям постійно зростаючих кіберзагроз.

Впровадження системи безпеки мікросервісів також вимагає гнучкості та адаптивності, оскільки загрози та технології постійно змінюються.

Забезпечення ефективного захисту в довгостроковій перспективі вимагає регулярного перегляду та оновлення стратегій безпеки, впровадження нових технологій для виявлення загроз і реагування на них, а також постійного навчання персоналу.

Лише шляхом постійного вдосконалення та адаптації заходів безпеки ви можете гарантувати, що ваші програми мікросервісів будуть захищені, а ваша організація залишиться конкурентоспроможною в сучасному цифровому світі.

Розробка систем для захисту мікросервісів від шпигунського програмного забезпечення та криптографічних вірусів є складним, але важливим завданням для забезпечення безпеки та стійкості сучасних цифрових екосистем.

Впроваджуючи комплексний підхід, який поєднує передові технології, ефективні процеси та відповідальну поведінку співробітників, підприємства можуть ефективно захищати програми мікросервісів і залишатися конкурентоспроможними перед обличчям постійно зростаючих кіберзагроз.

Лише шляхом постійного вдосконалення, адаптації та співпраці між усіма зацікавленими сторонами ми можемо досягти високого рівня безпеки для програм мікросервісів і забезпечити безпечне та стійке цифрове майбутнє для всіх.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Мікросервіси для початківців URL: https://blog.ukrnames.com/veb-master/mikroservisi-dlya-pochatkivtsiv (дата звернення: 01.05.2024).

2. Мікросервісна архітектура для початківців. Частина I URL: https://www.globallogic.com/ua/insights/blogs/microservices-architecture-for-beginners-part-one/ (дата звернення: 01.05.2024).

3. Мікросервіси та мікросервісна архітектура: сучасний підхід до розробки програмного забезпечення URL: https://bizmag.com.ua/arkhitektura-mikroservisiv-dlia-biznesu/ (дата звернення: 01.05.2024).

4. Мікросервісна архітектура: основні концепції та виклики URL: https://foxminded.ua/mikroservisna-arkhitektura/ (дата звернення: 03.05.2024).

5. Що таке мікросервісна архітектура: шлях до гнучкого та масштабованого середовища розробки URL: https://blog.colobridge.net/uk/2024/01/what-is-microservices-architecture-ua/ (дата звернення: 03.05.2024).

6. Мікросервіси URL: https://uk.wikipedia.org/wiki/%D0%9C%D1%96%D0%BA%D1%80%D0%BE%D1%81%D0%B5%D1%80%D0%B2%D1%96%D1%81%D0%B8 (дата звернення: 03.05.2024).

7. Мікросервісна архітектура для початківців. Частина II URL:https://www.globallogic.com/ua/insights/blogs/microservices-architecture-for-beginners-part-two/ (дата звернення: 03.05.2024).

8. Мікросервісна архітектура ПЗ. URL: https://qalight.ua/baza-znaniy/shho-take-mikroservisna-arhitektura-pz/ (дата звернення: 18.06.2024).

9. Мікросервісна архітектура: основні концепції та виклики URL: https://foxminded.ua/mikroservisna-arkhitektura/ (дата звернення: 18.06.2024).

10. Мікросервісна архітектура URL: https://medium.com/@IvanZmerzlyi/microservices-architecture-461687045b3d (дата звернення: 10.05.2024).

11. Комплексна система захисту інформації URL: https://uk.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D0%BB%D0%B5%D0%BA%D1%81%D0%BD%D0%B0_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D0%B7%D0%B0%D1%85%D0%B8%D1%81%D1%82%D1%83_%D1%96%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%86%D1%96%D1%97 (дата звернення: 10.05.2024).

12. Захист інформації URL: https://uk.wikipedia.org/wiki/%D0%97%D0%B0%D1%85%D0%B8%D1%81%D1%82_%D1%96%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%86%D1%96%D1%97 (дата звернення: 14.05.2024).

13. Архітектура мікросервісів: Особливості, переваги, реальні приклади URL: https://www.hostzealot.com.ua/blog/about-solutions/arxitektura-mikroservisiv-osoblivosti-perevagi-realni-prikladi (дата звернення: 14.05.2024).

14. Мікросервісна архітектура у практиці DevOps URL: https://production-ready.dev/2022/11/mikroservisna-arkhitektura/ (дата звернення: 18.05.2024).

15. Що таке програмне забезпечення для захисту даних? URL: https://www.kingston.com/ua/blog/data-security/what-is-data-security-software (дата звернення: 18.05.2024).

16. Шкідливе програмне забезпечення та боротьба з ним URL: https://www.miyklas.com.ua/p/informatica/9-klas/programne-zabezpechennia-ta-informatciina-bezpeka-327110/osnovi-zakhistu-danikh-327187/re-e9facff7-f2f0-4716-82d8-173b876b7c8d (дата звернення: 18.05.2024).

17. Безпека програмного забезпечення: Сучасні загрози та методи захисту URL: https://www.44.ua/news/3698042/bezpeka-programnogo-zabezpecenna-sucasni-zagrozi-ta-metodi-zahistu (дата звернення: 18.05.2024).

18. Що таке шкідливе програмне забезпечення? URL: https://www.microsoft.com/uk-ua/security/business/security-101/what-is-malware (дата звернення: 18.05.2024).

19. Що нового у світі мікросервісів. Думка System Architect URL: https://dou.ua/forums/topic/40828/ (дата звернення: 18.05.2024).

20. Мікросервісна архітектура: плюси та мінуси URL: https://itedu.center/ua/blog/articles/microservices-architecture-advantages-and-disadventages/ (дата звернення: 20.05.2024).

21. Мікросервіси та мікросервісна архітектура: сучасний підхід до розробки програмного забезпечення URL: https://bizmag.com.ua/arkhitektura-mikroservisiv-dlia-biznesu/ (дата звернення: 20.05.2024).

22. Мікросервіси: що це таке і для чого використовується? URL: https://tqm.com.ua/ua/likbez/ua-articles/micro-poslygi (дата звернення: 20.05.2024).

23. Складнощі тестування мікросервісів та що з ними робити URL: https://dou.ua/lenta/articles/difficulties-in-microservices-testing/ (дата звернення: 20.05.2024).

24. Посібник з мікросервісів Java. Частина 1: основи мікросервісів та їх архітектура URL: https://javarush.com/ua/groups/posts/uk.2660.posbnik-z-mkroservsv-java-chastina-1-osnovi-mkroservsv-ta-kh-arkhtektura (дата звернення: 23.05.2024).

25. Мікросервіси: основи та переваги URL: https://onecloudplanet.com/blog/article/microservices-foundations-and-advantages (дата звернення: 23.05.2024).

26. Розробка програмного забезпечення URL: https://uk.wikipedia.org/wiki/%D0%A0%D0%BE%D0%B7%D1%80%D0%BE%D0%B1%D0%BA%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE_%D0%B7%D0%B0%D0%B1%D0%B5%D0%B7%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%BD%D1%8F (дата звернення: 23.05.2024).

27. Стадії циклу розробки ПЗ URL: https://qalight.ua/baza-znaniy/stadiyi-tsiklu-rozrobki-pz/ (дата звернення: 24.05.2024).

28. Що краще моноліт чи мікросервіси? Як обрати архітектуру проєкту? URL: https://iampm.club/ua/blog/shho-krashhe-monolit-chi-mikroservisi-yak-obrati-arhitekturu-projektu/ (дата звернення: 24.05.2024).

29. Розподілені графи у мікросервісах: як уникнути помилок і зробити їхнє використання простим і зручним URL: https://www.gen.tech/post/rozpodileni-graphy-u-mikroservisakh (дата звернення: 26.05.2024).

30. Microservices Security Cheat Sheet URL: https://cheatsheetseries.owasp.org/cheatsheets/Microservices_Security_Cheat_Sheet.html (дата звернення: 28.05.2024).

31. Best Practices for Developing and Securing a Microservices Architecture URL: https://securityintelligence.com/best-practices-for-developing-and-securing-a-microservices-architecture/ (дата звернення: 01.06.2024).

32. Microservices security: How to protect your architecture URL: https://www.atlassian.com/microservices/cloud-computing/microservices-security (дата звернення: 01.06.2024).

33. Microservices Security: Challenges and Best Practices URL: https://brightsec.com/blog/microservices-security/ (дата звернення: 18.06.2024).

34. Поняття addware та spyware.Способи захисту від небажаного та шпигунського програмного забезпечення URL: https://alextexnok.blogspot.com/p/blog-page_18.html (дата звернення: 03.06.2024).

35. Комп'ютерний вірус URL: https://uk.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%27%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%B8%D0%B9_%D0%B2%D1%96%D1%80%D1%83%D1%81 (дата звернення: 03.06.2024).

36. Spyware URL: https://uk.wikipedia.org/wiki/Spyware (дата звернення: 18.06.2024).

37. Здирники, шифрувальники, ransomware - що це і як з ними боротися URL: https://www.sim-networks.com/ukr/kb/ransomware-how-to-win (дата звернення: 03.06.2024).

38. Що таке віруси шифрувальники і як з ними боротися URL: http://antivirus.pp.ua/post95_ua.php (дата звернення: 03.06.2024).

39. Робота з антиівірусною програмою URL: https://www.miyklas.com.ua/p/informatica/9-klas/programne-zabezpechennia-ta-informatciina-bezpeka-327110/osnovi-zakhistu-danikh-327187/re-431e2d4d-15db-44ef-b77d-1815cc9378c5 (дата звернення: 04.06.2024).

40. Антивірусна програма URL: https://uk.wikipedia.org/wiki/%D0%90%D0%BD%D1%82%D0%B8%D0%B2%D1%96%D1%80%D1%83%D1%81%D0%BD%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%B0 (дата звернення: 04.06.2024).

41. Призначення, принцип дії та класифікація антивірусних програм. Робота в середовищі антивірусної програми. Правила профілактики зараження комп'ютера вірусами URL: https://ladonenko.blogspot.com/p/blog-page_6.html (дата звернення: 04.06.2024).

ДОДАТКИ

Додаток А

Система захисту

```
import hashlib

import os

import sqlite3

import subprocess

import threading

import time

import shutil

import random

import string

import clamd  # Імпорт бібліотеки для інтеграції з ClamAV


# Розмір блоку для читання файлів (1 МБ)

BLOCK_SIZE = 1024 * 1024



class Database:

    def __init__(self, db_path):

        self.conn = sqlite3.connect(db_path)

        self.cursor = self.conn.cursor()
```

```
    def execute_query(self, query):
        self.cursor.execute(query)
        self.conn.commit()
        return self.cursor.fetchall()

    def update_user_password(self, username, new_password):
        hashed_password = hashlib.sha256(new_password.encode()).hexdigest()
        query = f"UPDATE users SET password = '{hashed_password}' WHERE username = '{username}'"
        self.execute_query(query)
        print(f"Password updated for user '{username}'.")


known_signatures = ["DROP TABLE", "DELETE FROM", "UPDATE", "INSERT INTO"]


def detect_threats(sql_query):
    if any(signature in sql_query for signature in known_signatures):
        return True
    else:
        return False
```

```
def scan_database_file(db_path):
    with open(db_path, 'rb') as file:
        content = file.read()
        for signature in known_signatures:
            if signature.encode('utf-8') in content:
                return True
    return False



def scan_file_for_viruses(file_path):
    cd = clamd.ClamdUnixSocket()
    result = cd.scan(file_path)
    if result[file_path][0] == 'FOUND':
        print(f"Virus detected in file: {file_path}")
        log_event("Virus detected", file_path)
        return True
    return False



def scan_microservice_integrity(service_folder):
    for root, dirs, files in os.walk(service_folder):
        for file in files:
            file_path = os.path.join(root, file)
            if scan_file_for_viruses(file_path):
```

```
            return True
        file_hash = calculate_file_hash(file_path)
        stored_hash = get_stored_file_hash(file_path)
        if file_hash != stored_hash:
            print(f"File integrity compromised: {file_path}")
            log_event("File integrity compromised", file_path)
            return True
    return False


def calculate_file_hash(file_path):
    hasher = hashlib.sha256()
    with open(file_path, 'rb') as file:
        while True:
            data = file.read(BLOCK_SIZE)
            if not data:
                break
            hasher.update(data)
    return hasher.hexdigest()


def get_stored_file_hash(file_path):
    # Функція повинна повертати збережений хеш файлу
    return "stored_hash_from_database"
```

```
def authenticate_user(username, password):
    # Реалізуйте механізм аутентифікації користувача тут
    return True  # Тимчасово завжди повертаємо True


def authorize_user(username, role):
    # Реалізуйте механізм авторизації користувача тут
    return True  # Тимчасово завжди повертаємо True


def restore_and_block(sql_query, username):
    if detect_threats(sql_query):
        print("Suspicious query detected. Data restoration and access blocking initiated.")
        log_event("Suspicious query detected", f"User: {username}, Query: {sql_query}")
        # Тут можна викликати функції відновлення даних або блокування доступу
    else:
        print("No suspicious activity detected.")


def update_security():
```

```
    known_signatures.append("INSERT INTO")
    print("Security signatures updated.")
    log_event("Security signatures updated")


def monitor_system_events():
    if platform.system() == "Windows":
        command = "Get-EventLog -LogName System -Newest 10 | Select-Object TimeGenerated, Message"
        process = subprocess.Popen(["powershell", "-Command", command], stdout=subprocess.PIPE)
    else:
        command = "dmesg | tail -10"
        process = subprocess.Popen(command, shell=True, stdout=subprocess.PIPE)

    output, error = process.communicate()
    events = output.decode("utf-8").split("\n")
    for event in events:
        if "Error" in event:
            print(f"Error event detected: {event}")
            log_event("System error detected", event)
        elif "Warning" in event:
            print(f"Warning event detected: {event}")
            log_event("System warning detected", event)
```

```
def monitor_microservice(service_folder):
    while True:
        if scan_microservice_integrity(service_folder):
            print("Microservice integrity compromised.")
            log_event("Microservice integrity compromised")
        else:
            print("Microservice integrity intact.")
            log_event("Microservice integrity intact")

        time.sleep(10)  # Почекати 10 секунд перед наступною перевіркою

def backup_database(db_path, backup_folder):
    if not os.path.exists(backup_folder):
        os.makedirs(backup_folder)
        print(f"Created backup directory: {backup_folder}")

    timestamp = time.strftime("%Y%m%d%H%M%S")
    backup_file = os.path.join(backup_folder, f"backup_{timestamp}.db")
    shutil.copy(db_path, backup_file)
    encrypt_file(backup_file)
    print(f"Database backed up to: {backup_file}")
    log_event("Database backup created", backup_file)
```

```
def encrypt_file(file_path):
    # Використовуємо AES шифрування
    from Crypto.Cipher import AES
    from Crypto.Random import get_random_bytes
    key = get_random_bytes(32)
    cipher = AES.new(key, AES.MODE_EAX)
    with open(file_path, 'rb') as f:
        data = f.read()
    ciphertext, tag = cipher.encrypt_and_digest(data)
    with open(file_path, 'wb') as f:
        for x in (cipher.nonce, tag, ciphertext):
            f.write(x)
    print(f"File {file_path} encrypted.")

def cleanup_backup_folder(backup_folder, max_backups=5):
    backups = os.listdir(backup_folder)
    backups.sort()

    while len(backups) > max_backups:
        file_to_delete = os.path.join(backup_folder, backups[0])
        os.remove(file_to_delete)
        print(f"Backup file deleted: {file_to_delete}")
        log_event("Backup file deleted", file_to_delete)
```

```
def example_usage():
    service_folder_1 = r'C:\Users\a3806\OneDrive\Рабочий стол\andriu-voznuk-main\db\scheme\client.db'  # Шлях до теки першого мікросервісу
    service_folder_2 = r'C:\Users\a3806\OneDrive\Рабочий стол\andriu-voznuk-main\db\scheme\driver.db'  # Шлях до теки другого мікросервісу
    db_path_1 = r'C:\Users\a3806\OneDrive\Рабочий стол\andriu-voznuk-main\db\scheme\client.db'  # Шлях до першої бази даних
    db_path_2 = r'C:\Users\a3806\OneDrive\Рабочий стол\andriu-voznuk-main\db\scheme\driver.db'  # Шлях до другої бази даних
    backup_folder = r'D:\Deplom\Backup'
    db_1 = Database(db_path_1)
    db_2 = Database(db_path_2)

    username = "example_user"
    password = "example_password"
    if authenticate_user(username, password):
        print(f"User '{username}' authenticated successfully.")
        if authorize_user(username, "admin"):
            print(f"User '{username}' authorized successfully.")
            sql_query = "DELETE FROM users WHERE id = 1"
            restore_and_block(sql_query, username)
        else:
            print("User is not authorized to perform this action.")
            log_event("Unauthorized access attempt", f"User: {username}")
```

```
    else:
        print("Authentication failed.")
        log_event("Authentication failed", f"User: {username}")
    # Створення потоків для моніторингу мікросервісів та резервного копіювання баз даних
    monitor_thread_1 = threading.Thread(target=monitor_microservice, args=(service_folder_1,))
    monitor_thread_2 = threading.Thread(target=monitor_microservice, args=(service_folder_2,))
    monitor_thread_1.start()
    monitor_thread_2.start()
    # Періодично оновлюємо паролі користувачів та робимо резервне копіювання баз даних
    while True:
        time.sleep(3600)  # Оновлюємо паролі та робимо резервне копіювання кожну годину
        new_password = generate_random_password()
        db_1.update_user_password(username, new_password)
        db_2.update_user_password(username, new_password)  # Оновлення паролів для другої бази даних
        backup_database(db_path_1, backup_folder)
        backup_database(db_path_2, backup_folder)  # Робимо резервне копіювання другої бази даних
        cleanup_backup_folder(backup_folder)
```

```
def generate_random_password(length=12):
    characters = string.ascii_letters + string.digits + string.punctuation
    password = ''.join(random.choice(characters) for i in range(length))
    print(f"Generated new random password: {password}")
    return password


def log_event(event, details=""):
    timestamp = time.strftime("[%Y-%m-%d %H:%M:%S]")
    log_message = f"{timestamp} {event}: {details}"
    print(log_message)
    with open("system_log.txt", "a") as log_file:
        log_file.write(log_message + "\n")
if __name__ == "__main__":
    update_security()  # Оновлення підписів безпеки
    example_usage()


    input("Press Enter to exit...")
```

Сервер

```
from flask import Flask, request, jsonify
import sqlite3
import os
import logging

app = Flask(__name__)

CLIENTS_DATABASE = r'D:\Deplom\БД Клієнт\.venv\Lib\site-packages\__pycache__\clients.db'
DRIVERS_DATABASE = r'D:\Deplom\БД Водій\.venv\Lib\site-packages\__pycache__\drivers.db'

# Set up logging
logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)

def connect_db(database_path):
    try:
        conn = sqlite3.connect(database_path)
        return conn
    except sqlite3.Error as e:
        logger.error(f"Database connection error: {e}")
```

```
        return None


@app.route('/register_client', methods=['POST'])
def register_client():
    data = request.get_json()
    if not data or 'name' not in data or 'phone' not in data:
        return jsonify({'error': 'Invalid input data'}), 400

    name = data['name']
    phone = data['phone']

    conn = connect_db(CLIENTS_DATABASE)
    if not conn:
        return jsonify({'error': 'Database connection error'}), 500

    try:
        cursor = conn.cursor()
        cursor.execute("INSERT INTO clients (name, phone) VALUES (?, ?)", (name,
phone))
        conn.commit()
        client_id = cursor.lastrowid
        conn.close()
    except sqlite3.Error as e:
        logger.error(f"Database error: {e}")
```

```
        conn.close()
        return jsonify({'error': str(e)}), 500

    return jsonify({'id': client_id}), 201

@app.route('/register_driver', methods=['POST'])
def register_driver():
    data = request.get_json()
    if not data or 'name' not in data or 'phone' not in data or 'car_model' not in data or 'license_plate' not in data:
        return jsonify({'error': 'Invalid input data'}), 400

    name = data['name']
    phone = data['phone']
    car_model = data['car_model']
    license_plate = data['license_plate']

    conn = connect_db(DRIVERS_DATABASE)
    if not conn:
        return jsonify({'error': 'Database connection error'}), 500

    try:
        cursor = conn.cursor()
```

```
        cursor.execute("INSERT INTO drivers (name, phone, car_model, license_plate)
VALUES (?, ?, ?, ?)",
                       (name, phone, car_model, license_plate))
        conn.commit()
        driver_id = cursor.lastrowid
        conn.close()
    except sqlite3.Error as e:
        logger.error(f"Database error: {e}")
        conn.close()
        return jsonify({'error': str(e)}), 500

    return jsonify({'id': driver_id}), 201



@app.route('/request_ride', methods=['POST'])
def request_ride():
    data = request.get_json()
    if not data or 'client_id' not in data or 'start_location' not in data or 'end_location'
not in data:
        return jsonify({'error': 'Invalid input data'}), 400

    client_id = data['client_id']
    start_location = data['start_location']
    end_location = data['end_location']
```

```
    try:
        with connect_db(CLIENTS_DATABASE) as conn:
            cursor = conn.cursor()
            cursor.execute("INSERT INTO rides (client_id, start_location, end_location, status) VALUES (?, ?, ?, 'requested')",
                           (client_id, start_location, end_location))
            conn.commit()
            ride_id = cursor.lastrowid
    except sqlite3.Error as e:
        return jsonify({'error': str(e)}), 500

    return jsonify({'ride_id': ride_id}), 201



@app.route('/accept_ride', methods=['POST'])
def accept_ride():
    data = request.get_json()
    app.logger.info(f"Request data: {data}")  # Log the request data

    if not data or 'ride_id' not in data or 'driver_id' not in data:
        return jsonify({'error': 'Invalid input data'}), 400

    ride_id = data['ride_id']
```

```
    driver_id = data['driver_id']


    try:
        with connect_db(DRIVERS_DATABASE) as conn:
            cursor = conn.cursor()
            cursor.execute("UPDATE rides SET driver_id = ?, status = 'accepted'
WHERE id = ?",
                           (driver_id, ride_id))
            conn.commit()
            if cursor.rowcount == 0:
                return jsonify({'error': 'Ride not found or already accepted'}), 404
    except sqlite3.Error as e:
        app.logger.error(f"Database error: {e}")  # Log any database errors
        return jsonify({'error': str(e)}), 500


    return jsonify({'status': 'ride accepted'}), 200



@app.route('/complete_ride', methods=['POST'])
def complete_ride():
    data = request.get_json()
    if not data or 'ride_id' not in data:
        return jsonify({'error': 'Invalid input data'}), 400
```

```
    ride_id = data['ride_id']


    try:
        with connect_db(DRIVERS_DATABASE) as conn:
            cursor = conn.cursor()
            cursor.execute("UPDATE rides SET status = 'completed' WHERE id = ?",
(ride_id,))
            conn.commit()
            if cursor.rowcount == 0:
                return jsonify({'error': 'Ride not found or already completed'}), 404
    except sqlite3.Error as e:
        return jsonify({'error': str(e)}), 500


    return jsonify({'status': 'ride completed'}), 200


@app.route('/rides/<int:driver_id>', methods=['GET'])
def get_rides_for_driver(driver_id):
    try:
        with connect_db(DRIVERS_DATABASE) as conn:
            cursor = conn.cursor()
            cursor.execute("SELECT * FROM rides WHERE status = 'requested' AND
driver_id IS NULL")
            rides = cursor.fetchall()
    except sqlite3.Error as e:
```

```
        app.logger.error(f"Database error: {e}")
        return jsonify({'error': str(e)}), 500

    rides_list = []
    for ride in rides:
        ride_data = {
            'id': ride[0],
            'client_id': ride[1],
            'start_location': ride[3],
            'end_location': ride[4],
            'status': ride[5] if len(ride) > 5 else None
        }
        rides_list.append(ride_data)

    return jsonify(rides_list), 200

if __name__ == '__main__':
    app.run(debug=True)
```

БД Клієнт

```
import sqlite3


CLIENTS_DATABASE = 'clients.db'
def init_clients_db():
    try:
        with sqlite3.connect(CLIENTS_DATABASE) as conn:
            cursor = conn.cursor()
            # Створення таблиці clients
            cursor.execute('''
                CREATE TABLE IF NOT EXISTS clients (
                    id INTEGER PRIMARY KEY AUTOINCREMENT,
                    name TEXT NOT NULL,
                    phone TEXT NOT NULL
                )
            ''')
            # Створення таблиці rides для зберігання поїздок
            cursor.execute('''
                        CREATE TABLE IF NOT EXISTS rides (
                            id INTEGER PRIMARY KEY AUTOINCREMENT,
                            driver_id INTEGER NOT NULL,
                            client_id INTEGER NOT NULL,
                            start_location TEXT NOT NULL,
```

```
                end_location TEXT NOT NULL,
                status TEXT NOT NULL,
                FOREIGN KEY (driver_id) REFERENCES drivers (id),
                FOREIGN KEY (client_id) REFERENCES clients (id)
            )
        ''')
# Створення таблиці readers
cursor.execute('''
    CREATE TABLE IF NOT EXISTS readers (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        name TEXT NOT NULL,
        email TEXT NOT NULL UNIQUE
    )
''')
# Створення таблиці users
cursor.execute('''
    CREATE TABLE IF NOT EXISTS users (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        username TEXT NOT NULL UNIQUE,
        password TEXT NOT NULL,
        email TEXT NOT NULL UNIQUE
    )
''')
conn.commit()
```

```
    except sqlite3.Error as e:
        print(f"An error occurred while initializing the database: {e}")
def connect_clients_db():
    try:
        return sqlite3.connect(CLIENTS_DATABASE)
    except sqlite3.Error as e:
        print(f"An error occurred while connecting to the database: {e}")
        return None
if __name__ == "__main__":
    init_clients_db()
```

БД Водій

```
import sqlite3

DRIVERS_DATABASE = 'drivers.db'

def init_drivers_db():
    try:
        with sqlite3.connect(DRIVERS_DATABASE) as conn:
            cursor = conn.cursor()
            # Create the drivers table
            cursor.execute('''
                CREATE TABLE IF NOT EXISTS drivers (
                    id INTEGER PRIMARY KEY AUTOINCREMENT,
                    name TEXT NOT NULL,
                    phone TEXT NOT NULL,
                    car_model TEXT NOT NULL,
                    license_plate TEXT NOT NULL
                )
            ''')

            # Create the rides table for storing rides
            cursor.execute('''
                CREATE TABLE IF NOT EXISTS rides (
```

```
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            driver_id INTEGER,
            client_id INTEGER NOT NULL,
            start_location TEXT NOT NULL,
            end_location TEXT NOT NULL,
            status TEXT NOT NULL,
            FOREIGN KEY (driver_id) REFERENCES drivers (id),
            FOREIGN KEY (client_id) REFERENCES clients (id)
        )
    ''')
    # Create the users table
    cursor.execute('''
        CREATE TABLE IF NOT EXISTS users (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            username TEXT NOT NULL UNIQUE,
            password TEXT NOT NULL,
            email TEXT NOT NULL UNIQUE
        )
    ''')

    # Commit changes
    conn.commit()
except sqlite3.Error as e:
    print(f"An error occurred while initializing the database: {e}")
```

```
def connect_drivers_db():
    try:
        return sqlite3.connect(DRIVERS_DATABASE)
    except sqlite3.Error as e:
        print(f"An error occurred while connecting to the database: {e}")
        return None
def get_ride(ride_id):
    try:
        conn = connect_drivers_db()
        if conn:
            cursor = conn.cursor()
            cursor.execute("SELECT * FROM rides WHERE id = ?", (ride_id,))
            ride = cursor.fetchone()
            conn.close()
            return ride
        else:
            return None
    except sqlite3.Error as e:
        print(f"An error occurred while getting ride: {e}")
        return None

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

Клієнт

```
import requests


SERVER_URL = 'http://127.0.0.1:5000'



def register_client():
    name = input("Enter your name: ")
    phone = input("Enter your phone number: ")
    url = f'{SERVER_URL}/register_client'
    data = {'name': name, 'phone': phone}
    try:
        response = requests.post(url, json=data)
        response.raise_for_status()  # Raise an error for bad responses (e.g., 4xx or 5xx)
        if response.status_code == 201:
            return response.json()
        else:
            print("Error registering client:", response.json())
            return None
    except requests.RequestException as e:
        print(f"Error registering client: {e}")
        return None
```

```
def request_ride(client_id):
    start_location = input("Enter pickup location: ")
    end_location = input("Enter drop-off location: ")
    url = f'{SERVER_URL}/request_ride'
    data = {
        'client_id': client_id,
        'start_location': start_location,
        'end_location': end_location
    }
    try:
        response = requests.post(url, json=data)
        response.raise_for_status()  # Raise an error for bad responses (e.g., 4xx or 5xx)
        if response.status_code == 201:
            return response.json()
        else:
            print("Error requesting ride:", response.json())
            return None
    except requests.RequestException as e:
        print(f"Error requesting ride: {e}")
        return None
def main():
    print("Registering client...")
    client_info = register_client()
```

```
    if not client_info:
        return
    print(f"Client registered with ID: {client_info.get('id')}")
    while True:
        action = input("Enter 'r' to request a ride or 'q' to quit: ")
        if action == 'r':
            ride_info = request_ride(client_info['id'])
            if ride_info:
                print(f"Ride requested: {ride_info}")
        elif action == 'q':
            break
        else:
            print("Invalid input, please enter 'r' or 'q'.")


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

Водій

```
import requests


SERVER_URL = 'http://127.0.0.1:5000'



def register_driver():
    name = input("Enter your name: ")
    phone = input("Enter your phone number: ")
    car_model = input("Enter your car model: ")
    license_plate = input("Enter your license plate: ")
    url = f'{SERVER_URL}/register_driver'
    data = {
        'name': name,
        'phone': phone,
        'car_model': car_model,
        'license_plate': license_plate
    }
    try:
        response = requests.post(url, json=data)
        response.raise_for_status()  # Raise an error for bad responses (e.g., 4xx or 5xx)
        if response.status_code == 201:
            return response.json()
```

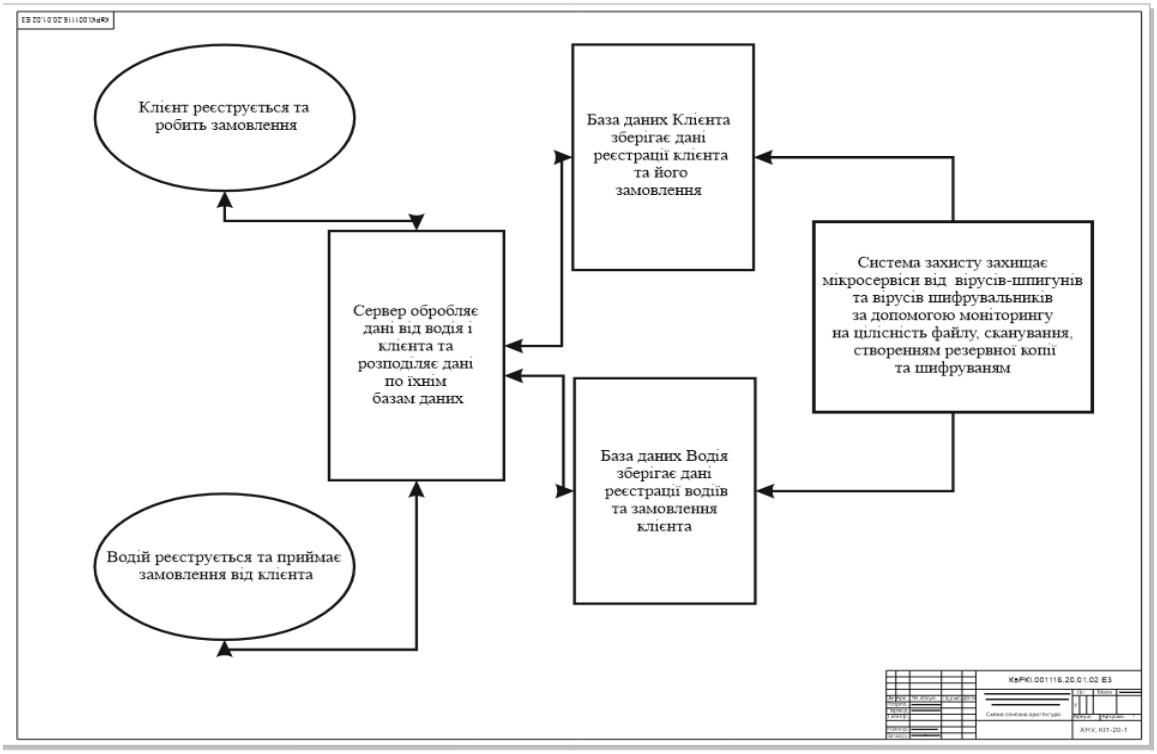
```
        else:
            print("Error registering driver:", response.json())
            return None
    except requests.RequestException as e:
        print(f"Error registering driver: {e}")
        return None



def get_pending_rides(driver_id):
    url = f'{SERVER_URL}/rides/{driver_id}'
    try:
        response = requests.get(url)
        response.raise_for_status()  # Raise an error for bad responses (e.g., 4xx or 5xx)
        if response.status_code == 200:
            return response.json()
        else:
            print("Error fetching pending rides:", response.json())
            return None
    except requests.RequestException as e:
        print(f"Error fetching pending rides: {e}")
        return None
def accept_ride(driver_id, ride_id):
    url = f'{SERVER_URL}/accept_ride'
    data = {
```

```
        'ride_id': ride_id,
        'driver_id': driver_id
    }
    try:
        response = requests.post(url, json=data)
        response.raise_for_status()  # Raise an error for bad responses (e.g., 4xx or 5xx)
        if response.status_code == 200:
            print("Ride accepted successfully.")
        else:
            print("Error accepting ride:", response.json())
    except requests.RequestException as e:
        print(f"Error accepting ride: {e}")
def main():
    print("Registering driver...")
    driver_info = register_driver()
    if not driver_info:
        return
    print(f"Driver registered with ID: {driver_info.get('id')}")
    while True:
        action = input("Enter 'v' to view pending rides, 'a' to accept a ride, or 'q' to quit:
")
        if action == 'v':
            pending_rides = get_pending_rides(driver_info['id'])
            if pending_rides:
```

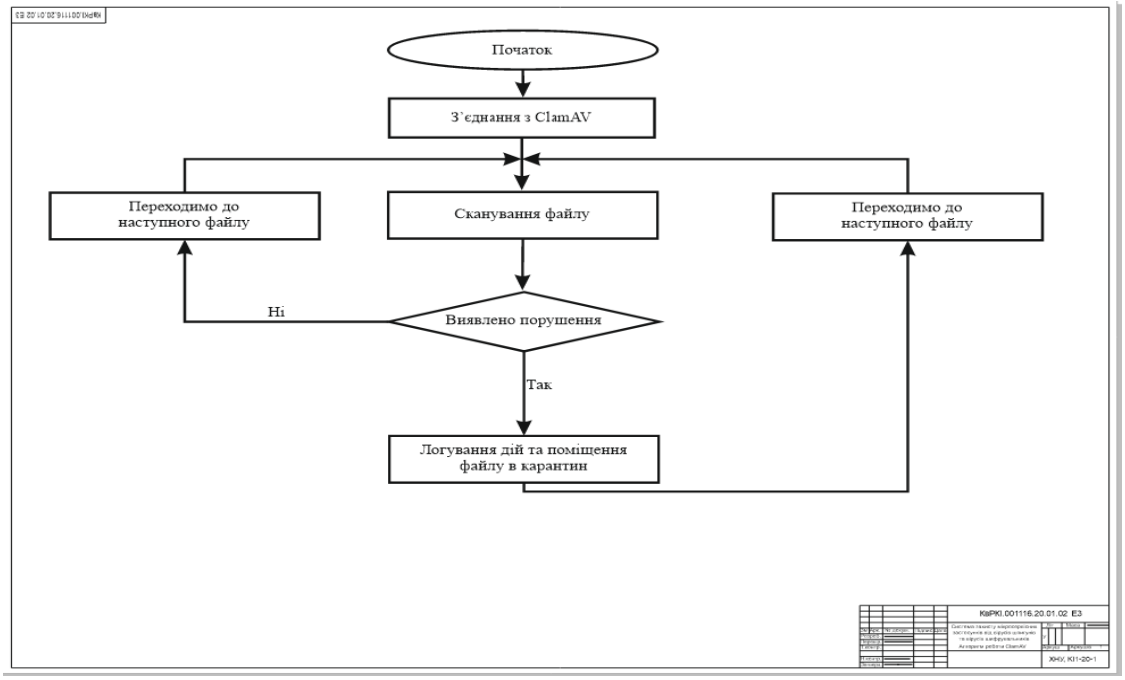
```
            print("Pending rides:")
            for ride in pending_rides:
                print(f"Ride ID: {ride['id']}, Start Location: {ride['start_location']}, End Location: {ride['end_location']}")
        else:
            print("No pending rides.")
    elif action == 'a':
        ride_id = input("Enter the ride ID to accept: ")
        accept_ride(driver_info['id'], ride_id)
    elif action == 'q':
        break
    else:
        print("Invalid input, please enter 'v', 'a', or 'q'.")

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

Додаток Б

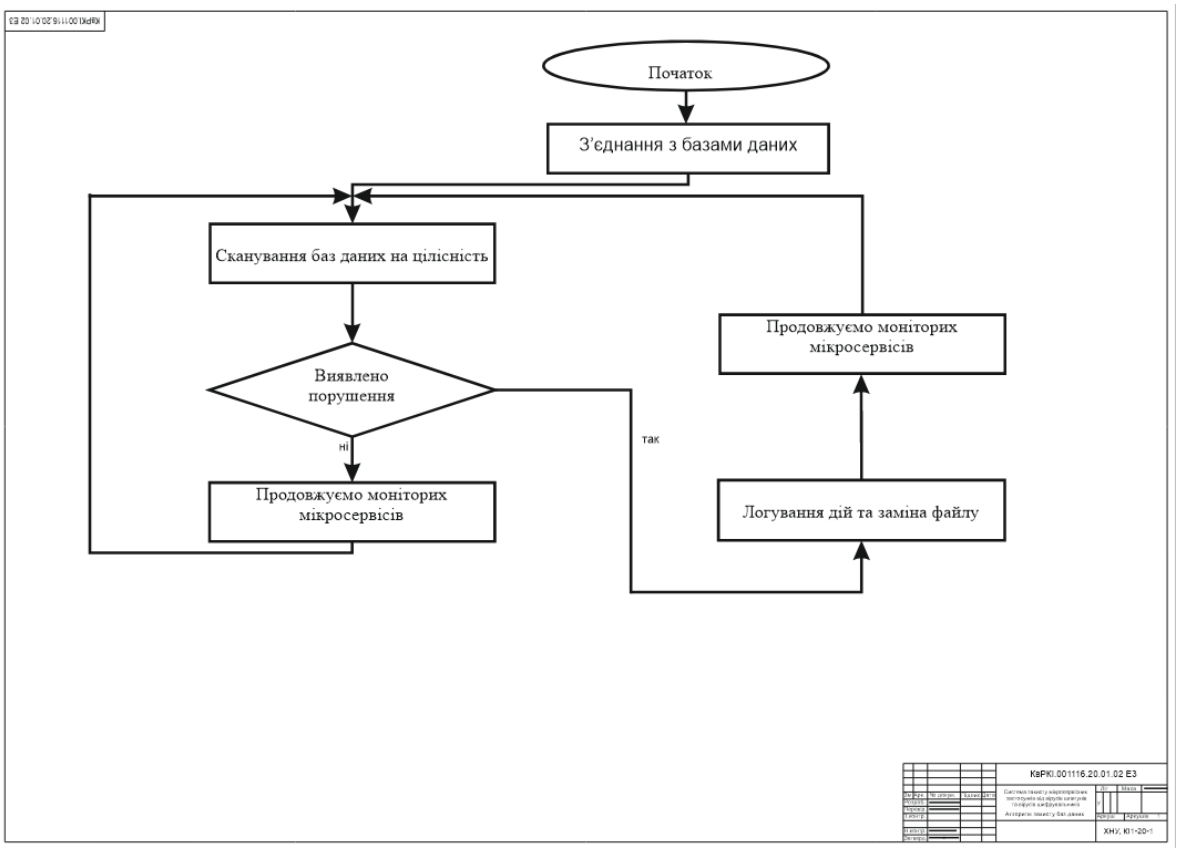
Загальна архітектура мікросервісу



Алгоритм роботи ClamAV



Алгоритм моніторингу бази даних



Додаток В

Перелік скорочень

1. API - Application Programming Interface (Інтерфейс програмування додатків)
2. DoS - Denial of Service (Відмова в обслуговуванні)
3. IAM - Identity and Access Management (Управління ідентифікацією та доступом)
4. JWT - JSON Web Token (Веб-токен на основі JSON)
5. ELK Stack - Elasticsearch, Logstash, Kibana (Стек інструментів для збору та аналізу даних)
6. SQL - Structured Query Language (Мова структурованих запитів)
7. MITM - Man-in-the-Middle (Атака типу "людина посередині")
8. DevSecOps - Development, Security, and Operations (Інтеграція безпеки в процес розробки та експлуатації)
9. ПЗ - Програмне забезпечення
10. SHA-256 - Secure Hash Algorithm 256-bit (256-бітний алгоритм хешування)
11. AES - Advanced Encryption Standard (Стандарт шифрування даних)
12. ACL - Access Control List (Список керування доступом)
13. VPN - Virtual Private Network (Віртуальна приватна мережа)
14. SIEM - Security Information and Event Management (Управління інформацією та подіями безпеки)
15. HIPAA - Health Insurance Portability and Accountability Act (Закон про переносимість і підзвітність медичного страхування)
16. PCI DSS - Payment Card Industry Data Security Standard (Стандарт безпеки даних індустрії платіжних карток)
17. IaC - Infrastructure as Code (Інфраструктура як код)
18. SOAR - Security Orchestration, Automation, and Response (Оркестрація, автоматизація та реагування в сфері безпеки)
19. UNIX-сокет - Програмний інтерфейс для взаємодії між процесами в Unix-системах
20. DDoS - Distributed Denial of Service (Розподілена відмова в обслуговуванні)

Завідувачу кафедри кібербезпеки
к.т.н., доц. Кльоцу Ю.П.
Вознюка Андрія Віталійовича
ПІБ здобувача вищої освіти

Студента ФІТ, 4 курсу, групи КІ1-20-1

ЗАЯВА

З правилами чинного Положення «Про систему забезпечення академічної доброчесності у хмельницькому національному університеті» від 31.08.2023, згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування заходів дисциплінарної та академічної відповідальності, ознайомлений. Про використання програмно-технічних засобів для перевірки кваліфікаційних робіт здобувачів вищої освіти на плагіат оповіщений та надаю свою згоду на обробку та збереження університетом моєї роботи в інституційному репозитарії університету.

Також надаю університету право на передачу моєї роботи для обробки та збереження в базах даних програмно-технічних засобів (Unicheck та Anti-Plagiarism) та використання роботи для виявлення плагіату в інших роботах, які перевіряються програмно-технічними засобами та користувачами, що мають доступ до цих програмно-технічних засобів, виключно в обмежених цілях для виявлення плагіату в текстах робіт.

Робота для перевірки університетом надається в друкованому та електронному варіанті. Електронна версія моєї роботи збігається (ідентична) з друкованою.

17.06.2024
дата

підпис

UNICHECK
by Turnitin

Ім'я користувача:
Кафедра кібербезпеки

ID перевірки:
1016373397

Дата перевірки:
18.06.2024 22:19:38 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
18.06.2024 22:35:00 EEST

ID користувача:
100008300

Назва документа: **Вознюк А.В_плагіат**

Кількість сторінок: **58** Кількість слів: **10138** Кількість символів: **83797** Розмір файлу: **3.89 MB** ID файлу: **1016180882**

Виявлено модифікації тексту (можуть впливати на відсоток схожості)

1.25%
Схожість

Найбільша схожість: **0.47%** з джерелом з Бібліотеки (ID файлу: **1016180885**)

0.85% Джерела з Інтернету	56	Сторінка 60
0.72% Джерела з Бібліотеки	52	Сторінка 60

0% Цитат

Вилучення цитат вимкнене

Вилучення списку бібліографічних посилань вимкнене

0%
Вилучень

Немає вилучених джерел

Модифікації

Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.

Замінені символи	1
Підозріле форматування	9 сторінок

Anti-Plagiarism v-15.257

Максимальне співпадіння з одним документом 1.0%

Словники перевірки: en_US, ru_RU, ua_UA. **Помилок в документах: 7%**

	Документ		Сумарний збіг по Базі Даних	
ID: 131419 Назва: Система захисту мікросервісних застосунків від вірусів шпигунів та вірусів шифрувальників Додано в БД: 2024-06-18 Автора: Вознюк А.В. Керівники: Стецюк М.В. Консультанти: Опоненти:	Символи	Лексеми	Символи	Лексеми
	78628	538	542 (1%)	6 (1%)

Джерело плагіату

ID	Опис	Наявність плагіату в документі	
		Символи	Лексеми

РІШЕННЯ ЕКСПЕРНОЇ КОМІСІЇ

КАФЕДРИ КІБЕРБЕЗПЕКИ

ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Підтверджуємо ознайомлення з результатом звіту подібності щодо роботи, генерованого системою виявлення текстових збігів/ідентичності/схожості:

Назва: Система захисту мікросервісних застоунків від вірусів шпигунів та вірусів шифрувальників

Автор: Вознюк Андрій Віталійович

Спеціальність: 123 – Комп'ютерна інженерія

Освітня програма: Програмування та захист комп'ютерних систем і мереж

Науковий керівник: Стецюк Микола Васильович др. фідлософії

Після аналізу звіту подібності зроблено такий висновок:

№	Висновок	Позначка про відповідність
1	Запозичення, виявлені в роботі, є законними і не є плагіатом. Робота приймається до захисту.	відповідає
2	Виявлені запозичення не є плагіатом, розміщені в розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи Робота приймається до захисту, але має бути відкоригована. Відкоригований варіант має бути поданий на кафедру за 2 дні до захисту, разом із заявою щодо самостійності виконання письмової роботи та ідентичності друкованої та електронної версії роботи	
3	Виявлені запозичення не є плагіатом, але частково розміщені в розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. В зв'язку з цим мета роботи та поставлені завдання не були досягнені. Робота може бути допущена до захисту (наступного року) після того як буде відкоригована та допрацьована і успішно пройде повторну перевірку на академічний плагіат.	
4	Робота містить навмисні текстові спотворення, передбачувані спроби укриття запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту.	

Підтвердження:

Оригінальність тексту роботи за результатами перевірки системою Unichek складає 98,75%, оригінальність тексту роботи за результатами перевірки системою Anti-Plagiarism v-15.257 складає 99%.

Згідно з Положенням про систему забезпечення академічної доброчесності у ХНУ (https://khmnu.edu.ua/wp-content/uploads/normatyvni-dokumenty/polozhennya/pro-systemu-zabezpechennya-akademichnoyi-dobrochesnosti.pdf, Додаток В) кваліфікаційна робота, виконана за , освітньо-професійною програмою, кількісні показники рівня унікальності тексту у відсотках до загального обсягу матеріалу в якій складає 75-100 %, визнається роботою з високою унікальністю тексту: «Текст вважається унікальним і не потребує додаткових дій щодо запобігання неправомірним запозиченням».

Керівник роботи _____ Микола СТЕЦЮК

Завідувач кафедри кібербезпеки _____ Юрій КЛЬОЦ

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

освітнього ступеня «бакалавр»

Студент Вознюк Андрій Віталійович

Тема Система захисту мікросервісних застосунків від вірусів шпигунів та вірусів шифрувальників

Спеціальність 123 – Комп'ютерна інженерія

Обсяг кваліфікаційної роботи освітньо-кваліфікаційного рівня «бакалавр»:

кількість листів креслень 3; кількість сторінок записки 61.

1. Короткий зміст роботи та прийнятих рішень У кваліфікаційній роботі була розроблена система захисту мікросервісів. Ця система захищає мікросервіси від вірусів. У процесі проєктування були розроблені такі компоненти: Система захисту та система таксі.

2. Висновок про відповідність кваліфікаційної роботи завданню У кваліфікаційній роботі було виконано поставлене завдання як у теоретичній, так і в практичній частині.

3. Характеристика виконання кожного розділу роботи, ступінь використання останніх досягнень науки і техніки і передових методів роботи: У вступі роботи наведена загальна характеристика задачі, визначені об'єкт, предмет та методи дослідження, а також сформульована мета. Зазначені задачі, що потрібно виконати для досягнення поставленої мети, проведений аналіз досліджуваної проблеми та обгрунтований підхід до її вирішення. У першому розділі розглядається архітектура мікросервісів. Наступні розділи присвячені розробці системи захисту мікросервісів.

4. Позитивні сторони роботи Кваліфікаційна робота має практичну цінність. Вона полягає у розробці системи захисту мікросервісів яка захищає бази даних мого мікросервісного застосунку а саме системи таксі.

5. Негативні сторони роботи В системі захисту не передбачені деякі компоненти системи можуть обмежувати масштабованість у великих організаціях або складних мережевих середовищах. Модулі бази даних і аналітики мають обмеження щодо обсягу даних, що обробляються, і кількості користувачів.

6. Оцінка графічного оформлення та пояснювальної записки роботи Графічне оформлення кваліфікаційної роботи відповідає темі роботи та виконане з дотриманням стандартів. В цілому, графічне оформлення є якісним, а пояснювальна записка відповідає нормам оформлення.

7. Відгук про роботу в цілому Кваліфікаційна робота заслуговує позитивної оцінки, оскільки весь матеріал роботи є структурованим, чітким та послідовним. Усі розділи роботи мають логічну послідовність, що сприяє зрозумінню викладеного матеріалу в рамках теми роботи. Графічний матеріал допомагає наочно продемонструвати доцільність та ефективність прийнятих рішень для досягнення мети.

8. Інші зауваження В переліку використаних джерел наявні посилання на популярні ресурси, такі, як Вікіпендія, які не рекомендовано використовувати при написанні кваліфікаціних робіт.

9. Оцінка кваліфікаційної роботи Враховуючи всі позитивні та негативні сторони представленої кваліфікаційної роботи, можна зробити висновок, що вона заслуговує оцінки «добре».

РЕЦЕНЗЕНТ (прізвище, ім'я, по батькові, посада, місце роботи)
Підченко Сергій Костянтинович,
завідувач кафедри ТМІТ, доктор технічних наук, професор

« 19 » 06 2025 р. (підпис)