Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя
(повне найменування вищого навчального закладу)
Факультет комп'ютерно-інформаційних систем і програмної інженерії
(назва факультету)
Кафедра кібербезпеки
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

магістр
(освітній рівень)

на тему: "Дослідження сучасних інструментів моніторингу вразливих npm-пакетів у Node.js-проектах"

Виконав: студент VI курсу, групи СБм-61

Спеціальності:
125 «Кібербезпека та захист інформації»
(шифр і назва напряму підготовки, спеціальності)
Стебельський Максим Віталійович
підпис (прізвище та ініціали)

Керівник Загородна Н. В.
підпис (прізвище та ініціали)

Нормоконтроль Стадник М. А.
підпис (прізвище та ініціали)

Завідувач кафедри Загородна Н.В.
підпис (прізвище та ініціали)

Рецензент
підпис (прізвище та ініціали)

м. Тернопіль – 2025

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)

Кафедра кібербезпеки
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Загородна Н.В.
(підпис) (прізвище та ініціали)
«__» _____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня Магістр
(назва освітнього ступеня)

за спеціальністю 125 Кібербезпека та захист інформації
(шифр і назва спеціальності)

Студенту Стебельському Максиму Віталійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Дослідження сучасних інструментів моніторингу вразливих npm-пакетів у Node.js-проектах

Керівник роботи Загородна Наталія Володимирівна, кандидат технічних наук завідувач кафедри КБ
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від « 24 » листопада 2025 року № 4/7-1024

2. Термін подання студентом завершеної роботи 12.12.2025

3. Вихідні дані до роботи

4. Зміст роботи (перелік питань, які потрібно розробити)
Проаналізувати загрози ланцюжка постачання Node.js та сучасні засоби моніторингу
Провести аналіз ефективності інструментів сигнатурного та поведінкового контролю
Спроектувати комплексну систему захисту та розробити політики блокування загроз
Охорона праці та безпека в надзвичайних ситуаціях
Висновки

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)
Тема кваліфікаційної роботи. Проблематика та актуальність. Мета та завдання дослідження.
Класифікація загроз ланцюжка постачання Node.js. Порівняльний аналіз інструментів.
Архітектура спроектованої системи захисту. Алгоритм блокування та політики безпеки.
Порівняльна характеристика ефективності. Висновки та практична цінність.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Охорона праці	Осухівська Г.М., к.т.н., доцент		
Безпека в надзвичайних ситуаціях	Стручок В.С., ст. викладач кафедри ОХ		

7. Дата видачі завдання 19.09.2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1.	Ознайомлення з завданням до кваліфікаційної роботи	19.09 – 22.09	Виконано
2.	Підбір джерел та аналіз загроз безпеці ланцюжка постачання Node.js	23.09 – 30.09	Виконано
3.	Класифікація векторів атак та огляд методів SCA (Software Composition Analysis)	01.10 – 08.10	Виконано
4.	Оформлення першого розділу	09.10 – 15.10	Виконано
5.	Дослідження інструментів моніторингу (npm audit, Snyk, Socket) та визначення метрик	16.10 – 20.10	Виконано
6.	Порівняльний аналіз ефективності засобів захисту	21.10 – 23.10	Виконано
7.	Оформлення другого розділу	24.10 – 30.12	Виконано
8.	Проектування комплексної системи захисту та розробка політик безпеки	03.12 – 11.12	Виконано
9.	Оформлення третього розділу	21.11 – 30.11	Виконано
10.	Виконання завдання до підрозділу «Охорона праці та безпека в надзвичайних ситуаціях»	01.12 – 02.12	Виконано
11.	Оформлення кваліфікаційної роботи	03.12 – 11.12	Виконано
12.	Нормоконтроль	12.12 – 14.12	Виконано
13.	Перевірка на плагіат	15.12 – 18.12	Виконано
14.	Попередній захист кваліфікаційної роботи	19.12 – 19.12	Виконано
15.	Захист кваліфікаційної роботи	23.12.2025	

Студент _____ Стебельський М.В.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Загородна Н. В.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Дослідження сучасних інструментів моніторингу вразливих npm-пакетів у Node.js-проектах // ОР «Магістр» // Стебельський Максим Віталійович // Тернопільський національний технічний університет імені Івана Пулюя, факультет комп'ютерно-інформаційних систем і програмної інженерії, кафедра кібербезпеки, група СБм-61 // Тернопіль, 2025 // С. 59, рис. – 11, табл. – 1, кресл. – 13, додат. – 1.

Ключові слова: Node.js, npm, SCA, Reachability, Malware, Supply Chain.

Кваліфікаційна робота присвячена аналізу загроз ланцюжка постачання в екосистемі Node.js та дослідженню ефективності інструментів захисту.

Перший розділ закладає теоретичні основи безпеки ланцюжка постачання. У ньому розглядаються архітектура реєстру npm та специфіка управління залежностями, з акцентом на класифікацію векторів атак, таких як атаки з підміною назв пакетів або повна підміна пакетів, що становлять ключові загрози для екосистеми.

Другий розділ присвячено порівняльному аналізу сучасних засобів моніторингу. У ньому визначено типові метрики ефективності сканерів, досліджено роботу інструментів сигнатурного контролю, систем аналізу досяжності коду та платформ поведінкового аналізу, виявлено їхні переваги та обмеження.

Третій розділ зосереджено на обґрунтуванні комплексного підходу до захисту. У ньому запропоновано схему інтеграції засобів безпеки у CI/CD-конвеєр, описано налаштування політик автоматичного блокування загроз та продемонстровано переваги гібридної моделі, що поєднує статичний та поведінковий аналіз.

ABSTRACT

Research of Modern Tools for Monitoring Vulnerable npm-packages in Node.js projects // Thesis of educational level «Master» // Maksym Stebelskyi // Ternopil Ivan Puluj National Technical University, Faculty of Computer Information Systems and Software Engineering, Department of Cybersecurity, group СБм-61 // Ternopil, 2025 // p. 59, figs. – 11, tbls. – 1, drws. – 13, apps. – 1.

Keywords: Node.js, npm, SCA, Reachability, Malware, Supply Chain.

The thesis is devoted to analyzing threats to the supply chain in the Node.js ecosystem and researching the effectiveness of protection tools.

The first section lays the theoretical foundations of supply chain security. It examines the architecture of the npm registry and the specifics of dependency management, with an emphasis on the classification of attack vectors such as domain spoofing and package tampering, which pose key threats to the ecosystem.

The second section is devoted to a comparative analysis of modern monitoring tools. It defines typical metrics for scanner effectiveness, examines the operation of signature control tools, code reachability analysis systems, and behavioral analysis platforms, and identifies their advantages and limitations.

The third section focuses on justifying a comprehensive approach to protection. It proposes a scheme for integrating security measures into the CI/CD pipeline, describes how to configure automatic threat blocking policies, and demonstrates the advantages of a hybrid model that combines static and behavioral analysis.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

API	—	Application Programming Interface
AST	—	Abstract Syntax Tree
CI/CD	—	Continuous Integration / Continuous Delivery
CLI	—	Command Line Interface
CVE	—	Common Vulnerabilities and Exposures
CVSS	—	Common Vulnerability Scoring System
IDE	—	Integrated Development Environment
NPM	—	Node Package Manager
NVD	—	National Vulnerability Database
SAST	—	Static Application Security Testing
SBOM	—	Software Bill of Materials
SCA	—	Software Composition Analysis

ВСТУП

Актуальність теми. Екосистема Node.js є домінуючою платформою для сучасної веб-розробки, проте її відкрита архітектура та залежність від зовнішніх модулів створюють критичні ризики безпеки. Зростання кількості атак на ланцюжок постачання (Supply Chain Attacks), таких як експлуатація друкарських помилок (typosquatting), впровадження шкідливого коду (malware) та компрометація облікових записів розробників, робить традиційні методи захисту недостатніми. Стандартні інструменти аудиту, що базуються виключно на сигнатурному аналізі метаданих, не здатні виявляти загрози «нульового дня» та генерують високий рівень хибних спрацювань. Це зумовлює необхідність впровадження комплексних підходів, які поєднують статичний аналіз коду, перевірку контексту виконання та поведінковий моніторинг. Тому дослідження та імплементація гібридних систем моніторингу в середовищі CI/CD є актуальним науково-прикладним завданням.

Мета і задачі дослідження. Метою роботи є підвищення рівня захищеності Node.js-проектів шляхом розробки комплексної архітектури моніторингу, що інтегрує методи статичного та поведінкового аналізу для автоматизованого блокування загроз ланцюжка постачання.

Завданнями дослідження є

– проведення аналізу сучасних векторів атак на екосистему npm та виявлення недоліків існуючих методів захисту;

– здійснення порівняльного аналізу ефективності інструментів моніторингу за критеріями повноти покриття загроз та точності детектування;

– проектування ешелонованої системи безпеки з інтеграцією в автоматизований конвеєр розробки;

– обґрунтування ефективності запропонованої гібридної моделі захисту.

Об’єкт дослідження. Процес забезпечення безпеки ланцюжка постачання програмного забезпечення в екосистемі Node.js.

Предмет дослідження. Методи, моделі та програмні засоби моніторингу вразливостей, аналізу досяжності коду та поведінкового контролю npm-пакетів.

Наукова новизна одержаних результатів кваліфікаційної роботи. Удосконалено методологію захисту Node.js-додатків шляхом поєднання контекстного статичного аналізу (Reachability), сигнатурних підходів та поведінкового моніторингу, що, дозволяє виявляти та блокувати не лише відомі вразливості, а й загрози «нульового дня» (Malware Injection) та мінімізувати кількість хибних спрацювань.

Практичне значення одержаних результатів. Розроблено архітектуру комплексної системи захисту та сформовано набори конфігураційних політик для інструментів Snyk та Socket, які можуть бути безпосередньо імплементовані в робочі процеси DevSecOps для автоматизації контролю безпеки програмних продуктів.

Апробація результатів магістерської роботи. Основні результати проведених досліджень обговорювались на: XIII науково-технічній конференції «Інформаційні моделі, системи та технології» (м. Тернопіль, Україна).

Публікації. Основні результати кваліфікаційної роботи опубліковано у працях конференції (див. Додаток А).

РОЗДІЛ 1 АНАЛІЗ СТАНУ ПРОБЛЕМИ БЕЗПЕКИ ЛАНЦЮЖКА ПОСТАЧАННЯ В ЕКОСИСТЕМІ NODE.JS

1.1 Концепція модульної розробки в середовищі Node.js та роль npm

В основі сучасної веб-розробки лежить простий принцип: не писати весь код з нуля, а збирати програму з готових блоків. Цю ідею компонентної розробки запропонував Дуглас Макілрой ще в 1968 році, але саме з появою Node.js вона стала головним підходом у веб-програмуванні.

Сьогодні платформа Node.js разом із менеджером пакетів npm є стандартом для створення серверних додатків та веб-сервісів [1]. Роль npm полягає в тому, щоб спростити обмін кодом. Замість того, щоб писати власну функцію для роботи з датами або HTTP-запитами, розробник просто завантажує готовий модуль однією командою. Це дозволяє командам працювати набагато швидше і випускати продукти раніше за конкурентів.

Масштаби використання цієї системи величезні. Станом на липень 2025 року в реєстрі npm доступно вже понад 3,5 мільйона пакетів [2]. Це робить його найбільшою бібліотекою програмного забезпечення у світі.

Система управління залежностями в Node.js будується навколо кількох ключових елементів:

– Файл маніфесту (package.json).

– Семантичне версіонування (SemVer).

– Папка node_modules.

Ключовим компонентом є файл маніфесту (package.json). Це своєрідний «паспорт» проекту, у якому розробник чітко декларує, які саме бібліотеки (прямі залежності) необхідні для функціонування програми.

Механізм оновлення регулюється семантичним версіонуванням (SemVer) [3]. Це стандартизований набір правил (наприклад, запис ^1.2.0), які визначають, які оновлення пакетів менеджер може встановлювати автоматично, а які зміни є критичними і можуть порушити роботу коду.

Фізичне збереження всіх завантажених файлів відбувається у директорії node_modules. Саме сюди потрапляють не лише прямі залежності, вказані розробником, а й усі транзитивні пакети, від яких вони залежать.

Коли розробник встановлює один пакет, npm автоматично завантажує і всі пакети, від яких залежить цей пакет. Це називається транзитивними залежностями. На рисунку 1.1 зображено схему, яка відображає, як кілька прямих залежностей призводять до встановлення великої кількості транзитивних пакетів у директорії node_modules.

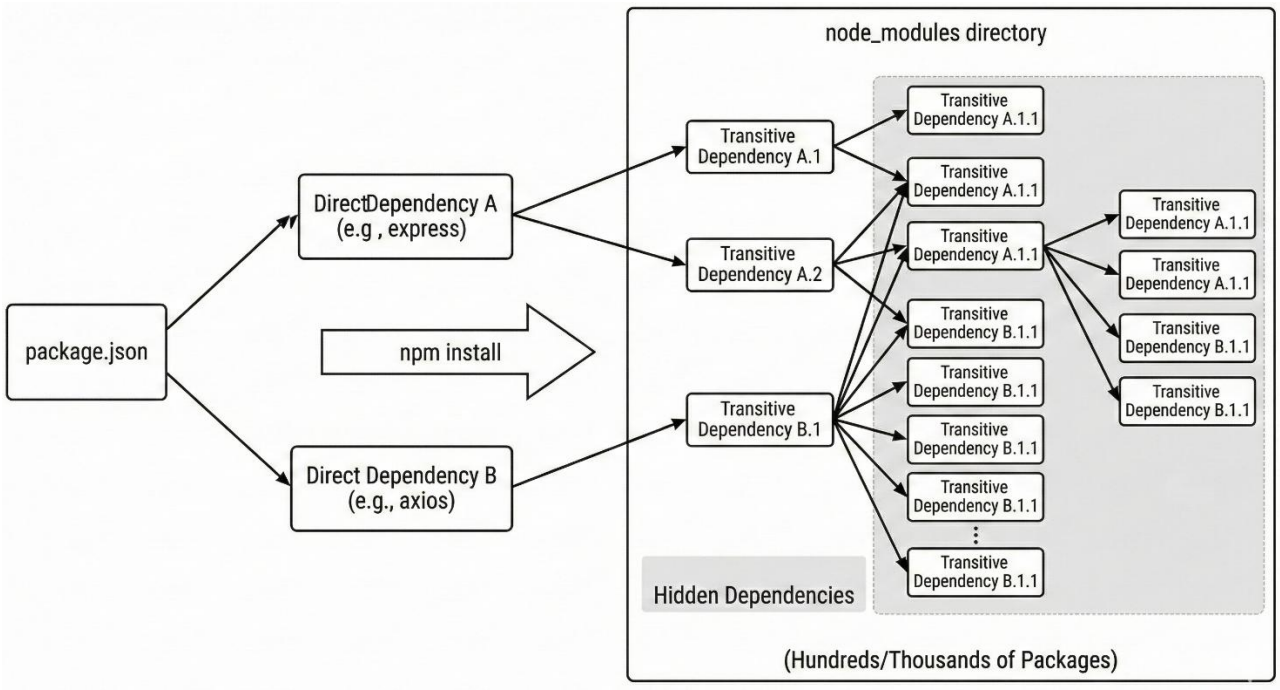

Рисунок 1.1 – Схема формування дерева залежностей

На практиці це призводить до того, що простий проект може містити сотні або тисячі сторонніх бібліотек. Часто розробник навіть не знає про їх існування, бо в package.json він вказав лише 10-20 прямих залежностей, а в папці node_modules опинилося 500 папок із кодом.

Статистика 2024–2025 років показує цікаву картину: у сучасному додатку власний код, написаний розробниками компанії, складає лише 10–30%. Решта 70–90% open-source код, завантажений з інтернету. Виходить ситуація, коли компанії витрачають великі бюджети на захист свого коду, але майже не

контролюють левову частку програми, яка фактично виконується на їхніх серверах. Цей ефект називають проблемою "чорної скриньки". Ми довіряємо авторам пакетів за замовчуванням. Але як показують звіти Sonatype за 2024 рік, цією довірою все частіше користуються зловмисники: з 2019 року було виявлено понад 700 000 шкідливих пакетів. Це означає, що механізм, який зробив Node.js таким популярним, став його головною вразливістю.

1.2 Класифікація загроз безпеці програмного забезпечення

Аналіз звітів провідних аналітичних компаній у сфері кібербезпеки, таких як CISA, Palo Alto Networks та GitLab, дозволяє констатувати суттєву трансформацію моделі загроз у період 2024–2025 років [4]. Якщо протягом попереднього десятиліття зусилля зловмисників були здебільшого спрямовані на пошук та експлуатацію вразливостей безпосередньо у коді фінальних додатків, то на сучасному етапі вектор атак змістився на компрометацію інструментів розробки та середовища виконання. Це явище, відоме як атаки на ланцюжок постачання (Supply Chain Attacks), базується на використанні довірчих відносин між розробниками та зовнішніми реєстрами пакетів, що робить традиційні методи захисту периметра неефективними. Для побудови адекватної системи захисту необхідно чітко класифікувати існуючі загрози, розділяючи їх на ненавмисні дефекти коду та цілеспрямовані атаки на інфраструктуру.

Першою та найбільш поширеною категорією залишаються вразливості коду (CVE). Це ненавмисні помилки, допущені розробниками бібліотек, які можуть призвести до порушення конфіденційності, цілісності або доступності системи. Навіть перевірені часом фреймворки не застраховані від критичних помилок. Яскравим прикладом є вразливість CVE-2025-55182 у бібліотеці React Server Components (версії 19.0.0–19.2.0), яка дозволяла неавтентифікованим зловмисникам виконувати довільний код (RCE) через небезпечну десеріалізацію даних у HTTP-запитах [5]. Іншим критичним випадком стала вразливість CVE-2024-21892 безпосередньо у середовищі виконання Node.js, що уможливлювала

підвищення привілеїв у системі через ін'єкцію коду. Такі дефекти, хоч і є небезпечними, зазвичай швидко виправляються спільнотою після їх виявлення.

Значно більшу загрозу становлять цілеспрямовані атаки на ланцюжок постачання, де зловмисники свідомо впроваджують шкідливий код (malware) в екосистему npm. Згідно з дослідженнями, основними векторами таких атак у звітному періоді стали:

– Експлуатація друкарських помилок (Typosquatting).

– Плутанина залежностей (Dependency Confusion).

– Компрометація облікових записів (Account Takeover).

– Шкідливі скрипти життєвого циклу.

– Protestware.

Механізм атак на основі схожості назв розрахований на неуважність розробника та полягає у реєстрації пакетів зі схожими назвами на популярні бібліотеки. Наприклад, зловмисники розміщують пакети urlib замість urllib або створюють клони react-dom з непомітними друкарськими помилками, що містять шкідливе навантаження.

Атака типу Dependency Confusion спрямована на корпоративний сектор, де використовуються внутрішні приватні пакети [6]. Зловмисник реєструє у публічному реєстрі пакет з ідентичною назвою, але вищою версією, змушуючи клієнт npm автоматично завантажити шкідливий код із зовнішнього джерела замість безпечного внутрішнього.

Вектор компрометації облікових записів набув особливої актуальності у 2025 році. Хакери отримують доступ до облікових записів авторів популярних пакетів через фішинг або підбір паролів. Це дозволяє легітимно опублікувати нову версію пакета, що вже містить шкідливий код. Жертвами таких атак ставали популярні бібліотеки coa, rc, ua-parser-js, а також мейнтейнери пакетів debug та chalk.

Особливу технічну небезпеку становлять шкідливі скрипти життєвого циклу [7]. Зловмисники використовують вбудовані у package.json хуки preinstall та postinstall для виконання довільного коду. Головна загроза полягає у тому, що

шкідливий код активується автоматично під час виконання команди npm install, ще до того, як розробник імпортує бібліотеку у проект.

Специфічним видом загрози є Protestware, коли автор пакету навмисно псує функціонал власної розробки на знак політичного протесту. Відомі прецеденти (наприклад, з пакетом node-ipc), коли оновлення містили код, що видаляв файли на комп'ютерах користувачів за географічною ознакою, створюючи ризики для бізнесу незалежно від його локації.

Оскільки звичайні інструменти аналізу покладаються на бази вже відомих вразливостей, вони просто не здатні виявити шкідливі пакети, які використовують стандартні механізми npm або є загрозами «нульового дня», що ще не внесені до баз даних.

1.3 Аналіз механізмів реалізації сучасних кіберзагроз

Показовим явищем, що продемонструвало новий рівень автоматизації атак у середовищі Node.js, стала поява шкідливого програмного забезпечення, що самореплікується [7]. Цей клас загроз змінив фокус зловмисників: замість пошуку вразливостей у коді вони почали використовувати механізми довірених залежностей, що дозволяє компрометувати значну кількість популярних пакетів у геометричній прогресії.

Механізм дії подібних кіберзагроз можна розділити на такі ключові етапи:

– Проникнення.

– Виконання.

– Ексфільтрація даних.

– Самовідтворювання.

– Активація механізму «Dead Man's Switch».

Для цілісного розуміння логіки роботи цього механізму доцільно розглянути структурну схему взаємодії його компонентів. На рисунку 1.2 візуалізовано замкнений життєвий цикл атаки, що демонструє безперервну

послідовність дій від первинного проникнення до автоматизованої експансії через реєстр пакетів.

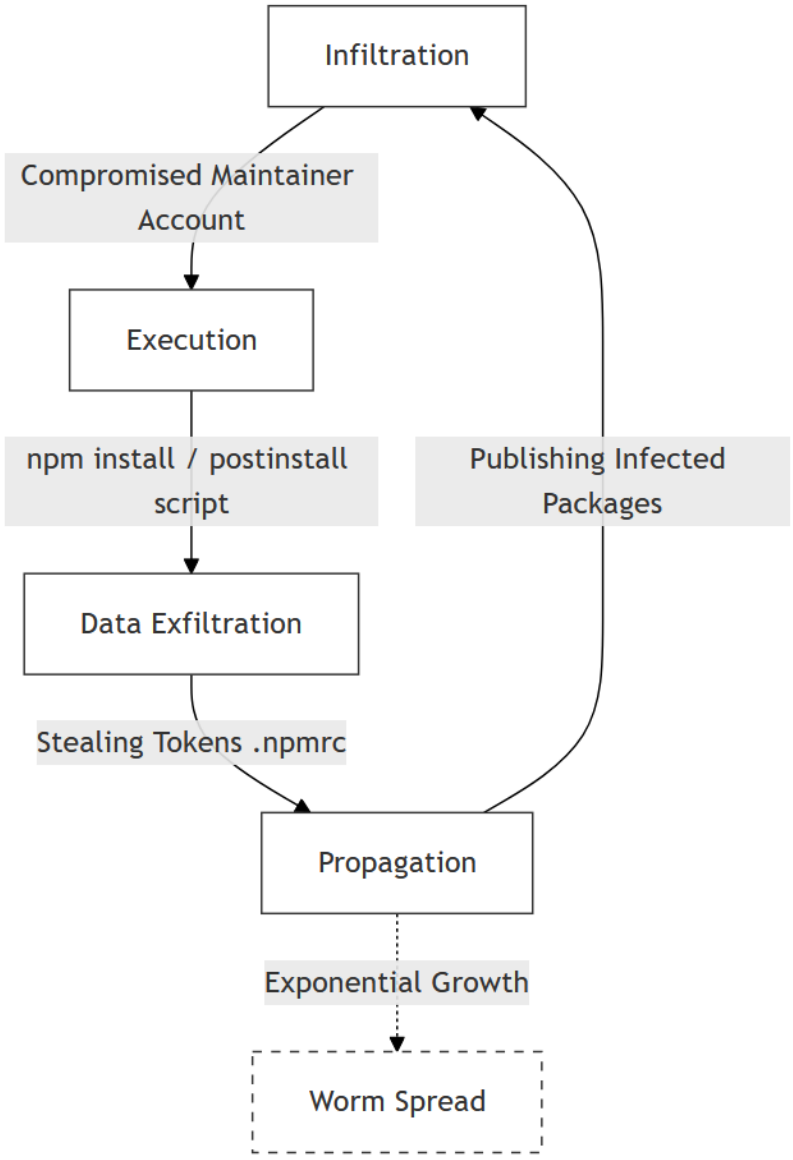

Рисунок 1.2 – Циклічна модель атаки типу «хробак» у ланцюжку постачання

На рисунку зображено схему атаки, яка функціонує за принципом замкненого циклу. Процес ініціюється через компрометацію облікового запису (Infiltration), що призводить до виконання шкідливого коду (Execution) на машині жертви. Це дозволяє здійснити викрадення секретних ключів (Data Exfiltration), які автоматично використовуються для публікації нових заражених пакетів (Propagation). Таким чином, кожна нова жертва стає джерелом

подальшого розповсюдження, що призводить до експоненціального росту загрози (Worm Spread).

Етап проникнення розпочинається з компрометації одного легітимного пакета, що зазвичай відбувається через фішинг мейнтейнера або злам його токена публікації.

Наступним кроком є виконання. Інфікований пакет містить скрипт у полі postinstall файлу package.json. Головна небезпека полягає в тому, що шкідливий код запускається автоматично одразу після команди npm install, завантажуючи основне тіло шкідливого ПЗ ще до того, як розробник почне використовувати бібліотеку у своєму коді.

Після запуску відбувається ексфільтрація даних: шкідливе ПЗ сканує змінні середовища та файли конфігурації (наприклад, .npmrc, .aws/credentials) на предмет наявності секретів. Основною ціллю є токени публікації npm, ключі AWS та GitHub Personal Access Tokens.

Використовуючи викрадені токени, загроза переходить до стадії самовідтворювання. Відбувається автоматична публікація нових версій усіх пакетів, до яких має доступ скомпрометований розробник, із впровадженням у них шкідливого коду. Це створює каскадний ефект, експоненційно збільшуючи зону ураження.

Окремою особливістю сучасних загроз є наявність механізму «Dead Man's Switch», який передбачає знищення даних користувача у разі втрати зв'язку з командним сервером (C2), що фактично перетворює шкідливе ПЗ на інструмент шантажу.

Паралельно з автоматизованими атаками, критичні вразливості виявляються і в архітектурі популярних фреймворків. Прикладом є вразливість CVE-2025-55182 у React Server Components, яка дозволила неавтентифікованим зловмисникам виконувати довільний код на сервері через маніпуляції з серіалізацією даних у HTTP-запитах. Цей факт підкреслює, що навіть перевірені бібліотеки від технологічних гігантів не застраховані від критичних помилок.

1.4 Огляд існуючих підходів до аналізу складу програмного забезпечення

Масове використання компонентів із відкритим вихідним кодом (Open Source) створило потребу в автоматизованих засобах контролю, які отримали назву Software Composition Analysis (SCA). Це методологія управління ризиками, що дозволяє інтегрувати перевірку безпеки безпосередньо в процес розробки (DevSecOps) та забезпечити видимість усіх сторонніх елементів у програмному продукті.

Реалізація методології DevSecOps неможлива без глибокого розуміння середовища, в якому відбувається життєвий цикл розробки — процесів CI/CD (Continuous Integration / Continuous Delivery). Ця парадигма передбачає створення автоматизованого конвеєра, де кожна зміна в коді проходить серію перевірок перед потраплянням у продуктивне середовище. Безперервна інтеграція (CI) забезпечує автоматичну збірку проекту та запуск модульних тестів при кожному збереженні коду в репозиторій (commit). Безперервна доставка (CD) автоматизує розгортання протестованого артефакту на сервери.

У контексті екосистеми Node.js цей процес має свою специфіку, яка створює унікальні вектори загроз. Типовий сценарій CI-конвеєра розпочинається з команди встановлення залежностей (зазвичай npm ci або npm install). Саме цей етап є найбільш критичним з точки зору безпеки ланцюжка постачання. Як було зазначено в попередніх підрозділах, зловмисники часто використовують скрипти життєвого циклу (preinstall, postinstall), які виконуються автоматично під час інсталяції пакетів. Це означає, що шкідливий код запускається безпосередньо на сервері збірки (Build Server). Небезпека полягає в тому, що CI-середовище зазвичай має доступ до критичних секретів проекту: ключів доступу до хмарних провайдерів (AWS, Azure), токенів баз даних та API-ключів для публікації релізів. Компрометація CI-сервера через шкідливий npm-пакет дозволяє зловмиснику ексфільтрувати ці секрети за лічені секунди, отримавши повний контроль над інфраструктурою компанії.

Крім того, висока швидкість роботи CI/CD конвеєрів, яка є їхньою головною перевагою, парадоксальним чином стає вразливістю. Традиційні підходи до безпеки, що передбачають ручний аудит коду або тривале сканування вразливостей перед релізом, стають «вузьким місцем», яке гальмує процес розробки. Якщо перевірка займає більше часу, ніж сама збірка, розробники часто відключають інструменти безпеки заради збереження швидкості. Це призводить до ситуації, коли шкідливий компонент може пройти всі етапи тестування (оскільки функціональні тести перевіряють працездатність, а не безпеку) і бути автоматично розгорнутим на тисячах серверів.

Відповіддю на ці виклики стала концепція «Shift Left Security». Суть підходу полягає в переміщенні перевірок безпеки з фінальних етапів розробки на якомога ранні стадії — безпосередньо в момент написання коду або його збірки в CI. Сучасні SCA-інструменти повинні інтегруватися в цей процес як автоматичні контрольні точки. Вони мають працювати в режимі реального часу, блокуючи збірку при виявленні нових вразливостей або підозрілої поведінки залежностей, ще до того, як артефакт буде створено. Таким чином, аналіз складу програмного забезпечення трансформується з пасивного звіту в активний механізм захисту інфраструктури розробки.

На сучасному етапі розвитку SCA-системи виконують чотири критичні функції, кожна з яких відіграє свою роль у захисті периметра розробки:

– Ідентифікація компонентів та побудова SBOM.

– Виявлення відомих вразливостей (CVE Detection).

– Юридичний аналіз та комплаєнс (License Analysis).

– Автоматизоване управління політиками (Policy Enforcement).

Функція ідентифікації компонентів є першим кроком роботи будь-якого SCA-інструменту та полягає у створенні так званого «програмного паспорта» виробу — SBOM [8]. Система сканує файли маніфестів (наприклад, package.json та package-lock.json у Node.js), рекурсивно розбираючи дерево залежностей. Важливою особливістю є здатність ідентифікувати не лише прямі залежності, які

розробник додав власноруч, а й транзитивні бібліотеки, що підтягуються автоматично і часто залишаються поза увагою команди.

Етап виявлення відомих вразливостей базується на зіставленні знайдених версій пакетів із глобальними базами даних загроз. Основними джерелами інформації виступають NVD (National Vulnerability Database), GitHub Advisory Database та OSV [9]. Кожній знайденій вразливості присвоюється рівень критичності згідно зі стандартом CVSS, що дозволяє розробникам пріоритизувати виправлення помилок від критичних до низьких [10].

Окрім безпекових ризиків, юридичний аналіз дозволяє контролювати правові аспекти використання стороннього коду. SCA-інструменти автоматично визначають тип ліцензії кожного пакета (MIT, Apache 2.0, GPL тощо) та перевіряють їх сумісність із політикою компанії. Це запобігає ситуаціям, коли використання бібліотеки з вірусною ліцензією зобов'язує компанію відкрити вихідний код власного комерційного продукту.

Механізм управління політиками забезпечує можливість налаштування «шлюзів якості» у CI/CD конвеєрах. Це означає, що система може автоматично заблокувати збірку проекту або заборонити злиття гілок, якщо виявлено вразливість критичного рівня або пакет із забороненою ліцензією.

Попри високу ефективність у виявленні вже відомих помилок, класичні SCA-інструменти мають суттєві архітектурні обмеження в умовах сучасних загроз. По-перше, вони є реактивними: механізм детекції спрацьовує лише тоді, коли вразливість вже зареєстрована в базі CVE. Це робить такі системи неефективними проти атак «нульового дня» або випадків впровадження шкідливого коду у нові версії пакетів. По-друге, існує проблема великої кількості хибних спрацювань, коли сканери повідомляють про вразливості, які технічно не можуть бути використані в контексті конкретного додатку. Зазначені обмеження зумовлюють необхідність переходу від простого порівняння версій до більш складних методів, таких як аналіз досяжності (Reachability Analysis) та поведінковий аналіз (Behavioral Analysis), які здатні оцінювати реальний контекст виконання коду.

РОЗДІЛ 2 АНАЛІЗ ТА ПОРІВНЯННЯ СУЧАСНИХ ІНСТРУМЕНТІВ МОНІТОРИНГУ NPM ПАКЕТІВ

Ефективна протидія атакам на ланцюжок постачання вимагає не лише розуміння природи загроз, розглянутої у попередньому розділі, але й обґрунтованого вибору інструментарію для їх виявлення та нейтралізації. Оскільки сучасний ринок пропонує широкий спектр рішень, від вбудованих утиліт командного рядка до комплексних хмарних платформ — виникає необхідність у їх дослідженні та об'єктивному порівнянні.

У даному розділі буде визначено ключові метрики ефективності систем захисту, такі як точність та повнота, а також проведено порівняльний аналіз трьох основних класів інструментів: базових засобів сигнатурного контролю метаданих, систем поглибленого статичного аналізу коду (SAST) та новітніх платформ поведінкового моніторингу.

2.1 Критерії та метрики оцінювання ефективності інструментів моніторингу

Ефективність сканера вразливостей визначається його здатністю коректно класифікувати об'єкти аналізу як безпечні або небезпечні. У контексті SCA-інструментів (Software Composition Analysis), де обсяги аналізованих залежностей можуть сягати тисяч одиниць, критично важливо мінімізувати помилки класифікації.

Для кількісної оцінки якості роботи сканерів у наукових дослідженнях використовується матриця помилок (Confusion Matrix) [11]. Вона дозволяє систематизувати результати роботи алгоритму за чотирма ключовими станами:

– True Positive (TP).

– False Positive (FP).

– True Negative (TN).

– False Negative (FN).

На рисунку 2.1 візуалізовано структуру матриці, що демонструє взаємозв’язок між прогнозованим станом системи та реальним класом об'єкта.

	Predicted: VULNERABLE (Система: Загроза)	Predicted: SAFE (Система: Безпечно)
Actual: VULNERABLE (Насправді: Загроза)	TRUE POSITIVE (TP) Загрозу виявлено	FALSE NEGATIVE (FN) Загрозу пропущено
Actual: SAFE (Насправді: Безпечно)	FALSE POSITIVE (FP) Хибна тривога	TRUE NEGATIVE (TN) Коректне ігнорування

Рисунок 2.1 – Матриця помилок класифікації загроз

Стан True Positive (TP), або істинно-позитивний результат, характеризує ситуацію, коли інструмент коректно ідентифікував реально існуючий вразливий пакет або шкідливий код. Це показник того, що система виконує свою основну функцію захисту.

Протилежним явищем є False Positive (FP) — хибно-позитивне спрацювання, коли сканер помилково позначає безпечний пакет як вразливий. Високий рівень FP є серйозною проблемою для процесу розробки, оскільки змушує інженерів витрачати значний час на ручну перевірку неіснуючих загроз. Це призводить до психологічного явища «втоми від безпеки» (security fatigue), коли розробники через недовіру починають ігнорувати всі звіти системи.

Стан True Negative (TN) відображає коректну роботу системи у штатному режимі, коли інструмент правильно ігнорує безпечні пакети, не генеруючи зайвого інформаційного шуму.

Найбільш критичною помилкою для систем безпеки є False Negative (FN), або хибно-негативний результат. У цьому випадку інструмент пропускає реальну загрозу, залишаючи небезпечний або інфікований пакет непоміченим. Саме такі помилки призводять до успішної реалізації атак на ланцюжок постачання.

На основі описаних станів розраховуються дві головні метрики ефективності — точність та повнота.

Точність демонструє частку реальних загроз серед усіх позитивних спрацювань системи. Висока точність свідчить про низький рівень хибних спрацювань. Метрика визначається за формулою 2.1:

$$Precision = \frac{TP}{TP+FP} \quad (2.1)$$

Повнота показує здатність системи знайти всі існуючі загрози у проекті, мінімізуючи пропуски. Ця метрика розраховується за формулою 2.2:

$$Recall = \frac{TP}{TP+FN} \quad (2.2)$$

Окрім математичних метрик, при порівняльному аналізі сучасних інструментів необхідно враховувати додаткові якісні критерії, що визначають їх практичну цінність у CI/CD конвеєрах:

– Reachability Analysis (аналіз досяжності).

– Time-to-Detect (швидкість оновлення баз даних).

– Behavioral Analysis (поведінковий аналіз).

Критерій Reachability Analysis визначає здатність інструменту перевірити, чи дійсно викликається вразливий фрагмент коду бібліотеки в контексті конкретного додатку. Це дозволяє автоматично відсіювати вразливості, які технічно неможливо експлуатувати.

Показник Time-to-Detect характеризує часовий інтервал між публікацією інформації про експлойт або виявленням шкідливого пакету та появою відповідної сигнатури в базі даних інструменту.

Функція поведінкового аналізу забезпечує виявлення аномалій, таких як несанкціонований доступ до мережі або файлової системи, без прив'язки до бази відомих CVE. Це є ключовим фактором для захисту від загроз «нульового дня».

2.2 Дослідження інструментів сигнатурного аналізу метаданих

Базовим ешелоном захисту екосистеми Node.js на сучасному етапі розвитку є інструменти, що базуються на методології сигнатурного аналізу метаданих. Цей підхід фундаментально відрізняється від статичного аналізу коду (SAST), оскільки передбачає перевірку не вихідного коду бібліотек, а їх декларативних описів (маніфестів версій та дерев залежностей). Найбільш поширеними та доступними представниками цього класу інструментів є вбудована утиліта npm audit та система автоматизації оновлень GitHub Dependabot. Обидва рішення тісно інтегровані з інфраструктурою GitHub та використовують спільну глобальну базу знань про загрози (GitHub Advisory Database), проте суттєво відрізняються сценаріями застосування, конфігурацією та алгоритмами реагування на інциденти.

2.2.1 Архітектура та алгоритм функціонування npm audit

Інструмент npm audit є нативним рішенням для платформи Node.js, яке було інтегроване безпосередньо в інтерфейс командного рядка (CLI) пакетного менеджера починаючи з версії npm 6. Ця інтеграція стала наслідком поглинання компанією npm, Inc. проекту Node Security Platform (nsp) — піонера у сфері безпеки Node.js, та подальшого придбання npm корпорацією GitHub [12]. Така консолідація дозволила створити єдиний стандарт перевірки безпеки, доступний кожному розробнику «з коробки» без необхідності встановлення стороннього ПЗ.

Процес сканування проекту інструментом npm audit є детермінованим і складається з чотирьох послідовних етапів:

– Генерація повного дерева залежностей.

– Серіалізація та відправка метаданих до реєстру.

– Звіряння з базою загроз на стороні сервера.

– Формування звіту та рекомендацій щодо виправлення.

На рисунку 2.2 візуалізовано послідовність інформаційної взаємодії між локальним середовищем розробника та хмарною інфраструктурою під час аудиту.

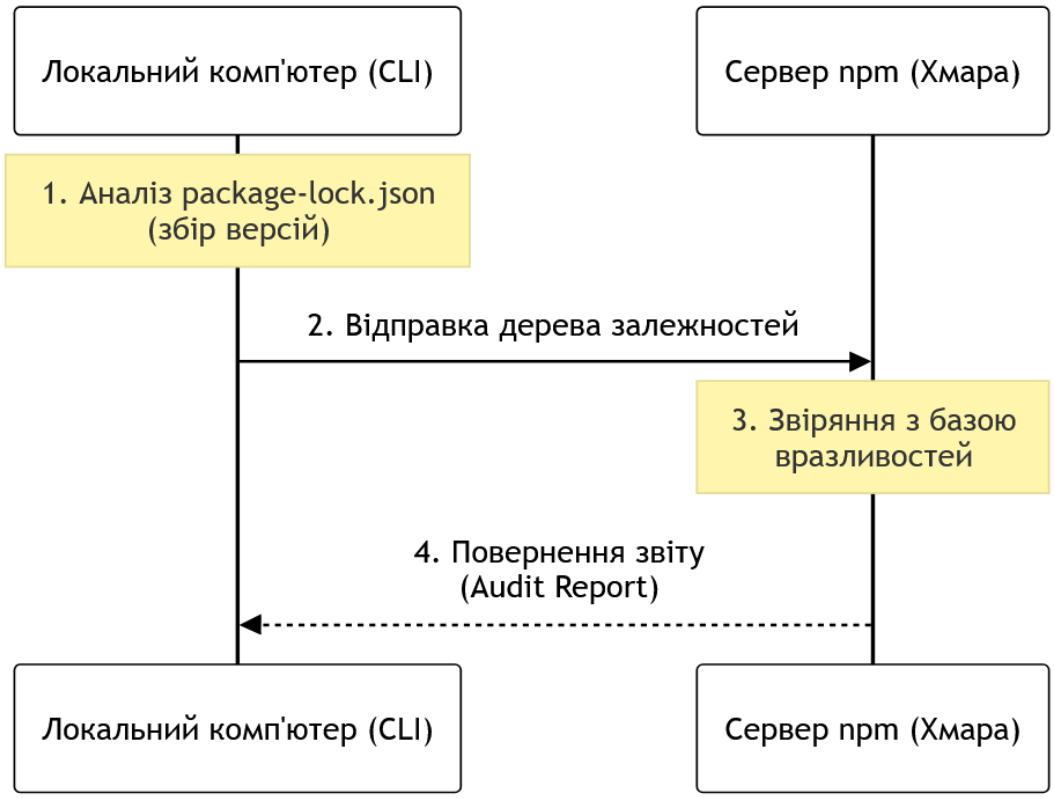

Рисунок 2.2 – Діаграма послідовності процесу аудиту залежностей

На етапі генерації дерева залежностей клієнт аналізує файл package-lock.json (або npm-shrinkwrap.json). На відміну від package.json, де версії вказані діапазонами (наприклад, ^1.2.0), lock-файл містить точну хеш-суму та версію кожного встановленого пакета, включно з усіма рівнями вкладеності (транзитивними залежностями). Це дозволяє побудувати точний граф використання бібліотек.

Далі відбувається процедура відправки метаданих. Сформований JSON-опис дерева залежностей надсилається до центрального реєстру npm через захищений API. Критично важливою архітектурною особливістю є те, що вихідний код проекту не передається і не покидає локальну машину розробника

— аналізуються виключно назви пакетів та їхні версії, що знімає ризики витоку інтелектуальної власності.

Звіряння з базою загроз відбувається на серверах npm. Отриманий список версій порівнюється із записами в GitHub Advisory Database, яка агрегує дані з NVD (National Vulnerability Database) та результати досліджень GitHub Security Lab [13]. Кожній знайденій вразливості присвоюється рівень критичності (Severity): Low, Moderate, High або Critical, що часто корелює з оцінкою CVSS (Common Vulnerability Scoring System). Фінальним кроком є формування звіту. Результат повертається користувачеві у вигляді структурованої таблиці в терміналі.

Окрім діагностики, інструмент пропонує команду npm audit fix, яка намагається автоматично оновити вразливі залежності до безпечних версій, що не порушують правила семантичного версіонування (SemVer). Однак, для мажорних оновлень (breaking changes) все одно потрібне ручне втручання (npm audit fix --force).

Суттєвим недоліком npm audit є високий рівень «інформаційного шуму». Оскільки інструмент перевіряє повний список залежностей, значна частина виявлених вразливостей (за різними оцінками до 85%) припадає на devDependencies — інструменти збірки, тестування (наприклад, Jest, Webpack) або лінтери. Оскільки цей код не потрапляє у фінальний білд і не виконується на стороні користувача або сервера в продакшні, реальний ризик експлуатації таких вразливостей є мінімальним. Це призводить до ігнорування звітів розробниками.

2.2.2 Автоматизація процесу оновлень за допомогою GitHub Dependabot

Якщо npm audit є інструментом для локальної перевірки або блокування збірки в CI/CD (через повернення коду помилки), то GitHub Dependabot реалізує стратегію безперервного фонового моніторингу [14]. Це хмарний сервіс, інтегрований безпосередньо в платформу GitHub, який автоматизує управління так званим «технічним боргом» залежностей. Конфігурація роботи інструменту

здійснюється через файл dependabot.yml, де визначаються цільові екосистеми, розклад перевірок та обмеження на кількість відкритих запитів..

Функціонал Dependabot поділяється на дві складові:

– Dependabot Security Updates.

– Dependabot Version Updates.

Перший компонент працює в реактивному режимі та ініціює оновлення лише при виявленні вразливостей у дереві залежностей. Другий компонент діє проактивно, оновлюючи застарілі пакети за заданим розкладом для підтримання актуальності технологічного стека. Принцип дії базується на скануванні файлів маніфестів (package.json, package-lock.json) безпосередньо в репозиторії. Алгоритм роботи виглядає наступним чином:

– Моніторинг змін.

– Детекція та аналіз сумісності.

– Генерація Pull Request (PR).

На рисунку 2.3 наведено алгоритм прийняття рішення системою Dependabot при виявленні вразливості, включаючи етап перевірки сумісності.

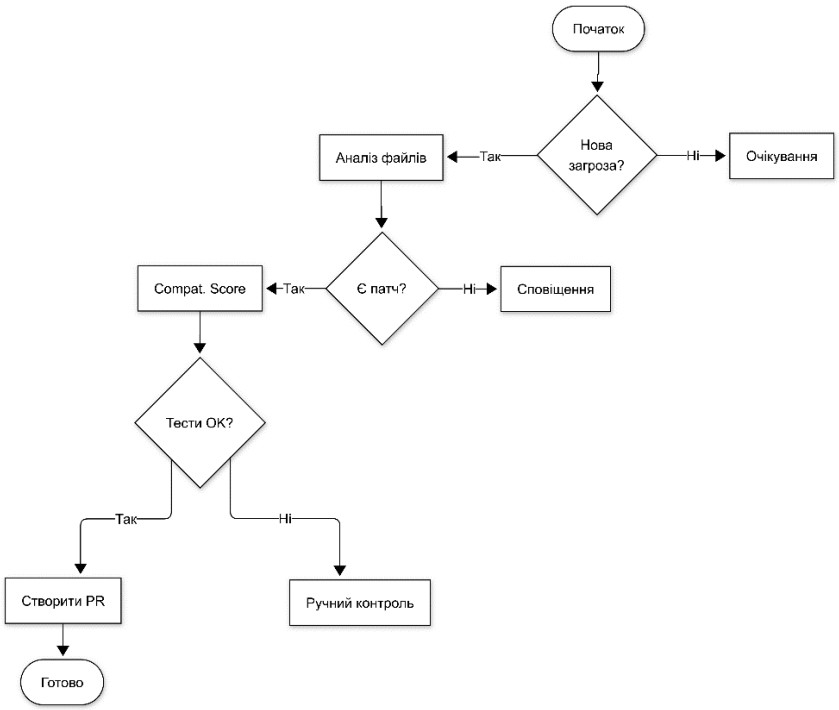

Рисунок 2.3 – Алгоритм автоматизованого усунення вразливостей системою Dependabot

Етап моніторингу змін передбачає, що система постійно відстежує публікацію нових версій пакетів у реєстрі npm та появу нових записів CVE у базі вразливостей GitHub Advisory Database. Це дозволяє реагувати на загрози миттєво, навіть якщо розробник не запускає локальний аудит.

Під час детекції та аналізу сумісності, при виявленні вразливої версії, Dependabot аналізує можливість оновлення. Унікальною особливістю є розрахунок оцінки сумісності (Compatibility Score). Система аналізує тисячі інших репозиторіїв, де це оновлення вже відбулося, і перевіряє, чи не призвело воно до падіння тестів у CI/CD конвеєрах. Якщо, наприклад, 95 % проектів успішно пройшли тести після оновлення, Dependabot позначає цей апдейт як безпечний.

Фінальним кроком є генерація Pull Request (PR), що є ключовою відмінністю від npm audit. Замість того, щоб просто повідомити про проблему, Dependabot діє проактивно: створює окрему гілку, оновлює версію пакета у файлах маніфестів і відкриває запит на злиття. Сформований запит містить вичерпну інформацію для прийняття рішення: опис вразливості з посиланням на CVE, нотатки до випуску (Release Notes) нової версії та оцінку сумісності.

Незважаючи на високий рівень автоматизації, Dependabot має ті ж фундаментальні обмеження, що й npm audit. Його модель захисту є реактивною: система здатна відреагувати на загрозу лише після того, як вразливість стала публічно відомою і була внесена в базу даних. Крім того, інструмент не аналізує вміст файлів («сліпий» до коду). Це робить його неефективним проти атак на ланцюжок постачання типу Malware Injection або Typosquatting, коли шкідливий пакет є абсолютно новим («нульового дня») і ще не класифікований як вразливий. Зловмисники часто використовують цей архітектурний недолік, маскуючи шкідливий код у мінорних оновленнях популярних бібліотек, які автоматичні системи на кшталт Dependabot вважають безпечними та пропонують до встановлення. Саме тому критично важливо доповнювати базовий моніторинг версій інструментами, здатними аналізувати фактичну поведінку завантаженого коду.

2.3 Аналіз систем поглибленого статичного контролю та технології Reachability (Snyk)

Якщо інструменти, розглянуті у попередньому підрозділі, працюють виключно на рівні метаданих (перевірка версій), то наступний ешелон захисту представлений системами, що комбінують SCA (Software Composition Analysis) із глибоким статичним аналізом коду [15]. Лідером у цьому сегменті є платформа Snyk, яка пропонує технологію Reachability Analysis (аналіз досяжності) [16]. Цей підхід дозволяє визначити, чи дійсно вразливий код бібліотеки використовується програмою, що значно підвищує точність оцінки ризиків.

Архітектуру процесу виявлення та класифікації вразливостей за допомогою технології Reachability наведено на Рисунку 2.4.

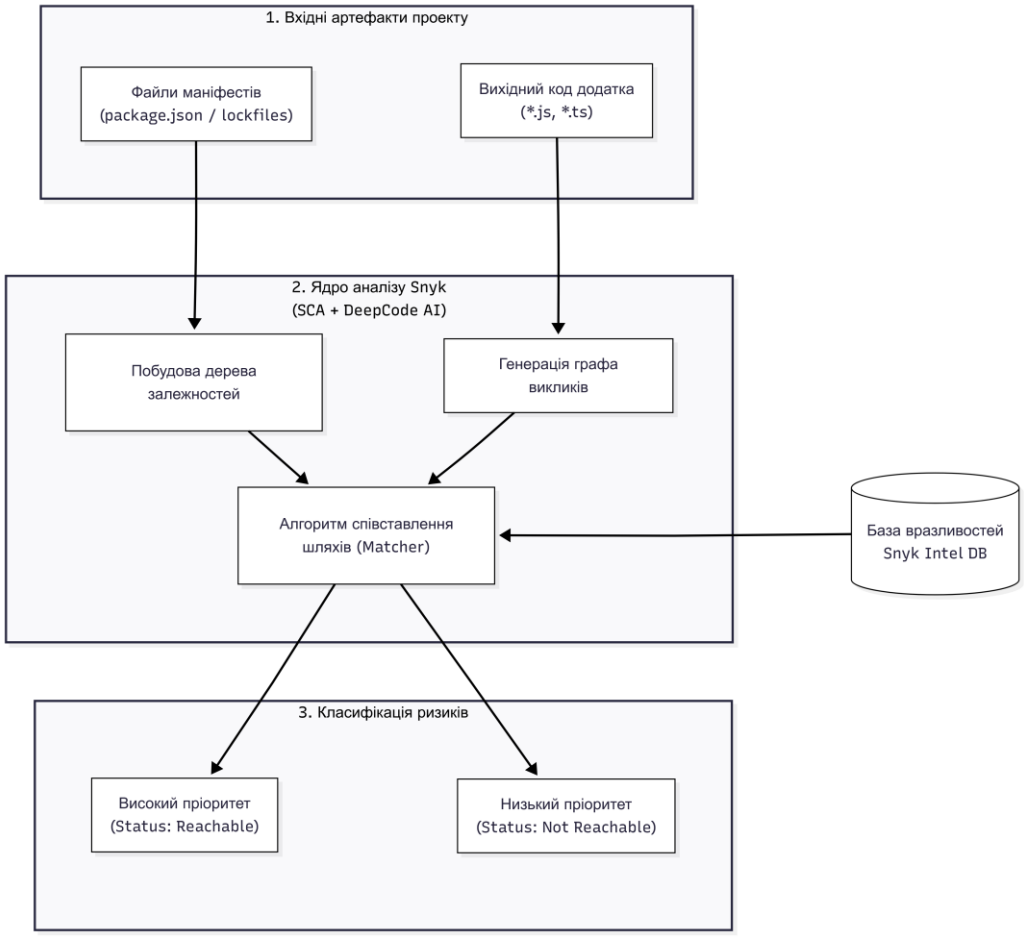

Рисунок 2.4 — Архітектура процесу аналізу досяжності вразливостей у середовищі Snyk

Як відображено на структурній схемі, процес аналізу є комплексним та складається з трьох послідовних етапів обробки даних:

– Збір та аналіз вхідних артефактів проекту.

– Обробка даних ядром аналізу (Core Engine).

– Класифікація ризиків та пріоритизація.

На першому етапі система паралельно сканує два типи даних. Аналізуються файли маніфестів (package.json, package-lock.json) для точної ідентифікації версій встановлених пакетів, а також власний вихідний код додатка (.js, .ts файли) для розуміння логіки його роботи.

Центральним елементом процесу є другий етап — робота ядра аналізу. Тут відбувається синтез отриманої інформації. На основі маніфестів будується повне дерево залежностей (Dependency Tree), яке звіряється із зовнішньою базою знань Snyk Intel DB для виявлення відомих вразливостей. Паралельно з цим алгоритми DeepCode AI будують граф викликів (Call Graph), що моделює потоки виконання програми. Завершує цей етап модуль співставлення (Matcher), який накладає дані про виявлені вразливості на побудований граф викликів.

Фінальним етапом є сегрегація загроз на основі результатів роботи матчера. Вразливості, до яких знайдено маршрут виконання від коду розробника, отримують статус Reachable (Високий пріоритет). Ті ж вразливості, що знаходяться у «мертвих зонах» графа і не викликаються додатком, маркуються як Not Reachable (Низький пріоритет).

Такий підхід дозволяє суттєво зменшити кількість помилкових спрацювань (False Positives). Статистика показує, що розробники часто використовують лише 10-20% функціоналу підключених бібліотек. Технологія Reachability дозволяє команді безпеки сфокусуватися саме на цих реальних векторах атак, ігноруючи вразливості в невикористовуваному коді.

Однак варто зазначити, що даний метод має обмеження. Побудова графа викликів для динамічної мови JavaScript може бути неповною через використання конструкцій eval() або складного метапрограмування. Крім того, цей метод не виявляє шкідливі скрипти (malware), які виконуються на етапі

інсталяції пакету, оскільки вони знаходяться поза контекстом run-time графа викликів.

2.4 Аналіз платформи поведінкового моніторингу Socket та виявлення загроз нульового дня

Якщо інструменти попередніх поколінь фокусуються на пошуку помилок у коді (вразливостей), то сучасний ландшафт кіберзагроз вимагає засобів, здатних виявляти зловмисні наміри. Атаки на ланцюжок постачання останніх років показали, що зловмисники часто не чекають на появу вразливостей CVE, а свідомо впроваджують шкідливий код у легітимні пакети. Для протидії цьому класу загроз виник новий напрямок — Supply Chain Security (Безпека ланцюжка постачання), що базується на методах поведінкового аналізу. Еталонним представником цього підходу є платформа Socket [7].

Фундаментальна відмінність цього підходу від традиційних SCA-сканерів полягає в об'єкті аналізу. Замість того, щоб сканувати Git-репозиторій, система аналізує безпосередньо артефакт пакету (tarball), який завантажується з реєстру. Це дозволяє виявляти загрози, які не класифікуються як CVE, але становлять критичну небезпеку:

– Скрипти інсталяції (Install Scripts).

– Прихована телеметрія (Telemetry).

– Ексфільтрація змінних середовища (Env Exfiltration).

– Тайпосквотінг (Typosquatting).

Скрипти інсталяції дозволяють виконання довільних команд під час встановлення пакету (через хуки preinstall, postinstall), що є найпоширенішим вектором атаки. Прихована телеметрія та ексфільтрація передбачають спроби відправки даних або зчитування ключів API та токенів зі змінних середовища. Тайпосквотінг полягає у використанні назв пакетів, візуально схожих на популярні бібліотеки, для введення розробника в оману.

Архітектуру та алгоритм роботи системи поведінкового аналізу наведено на Рисунку 2.5.

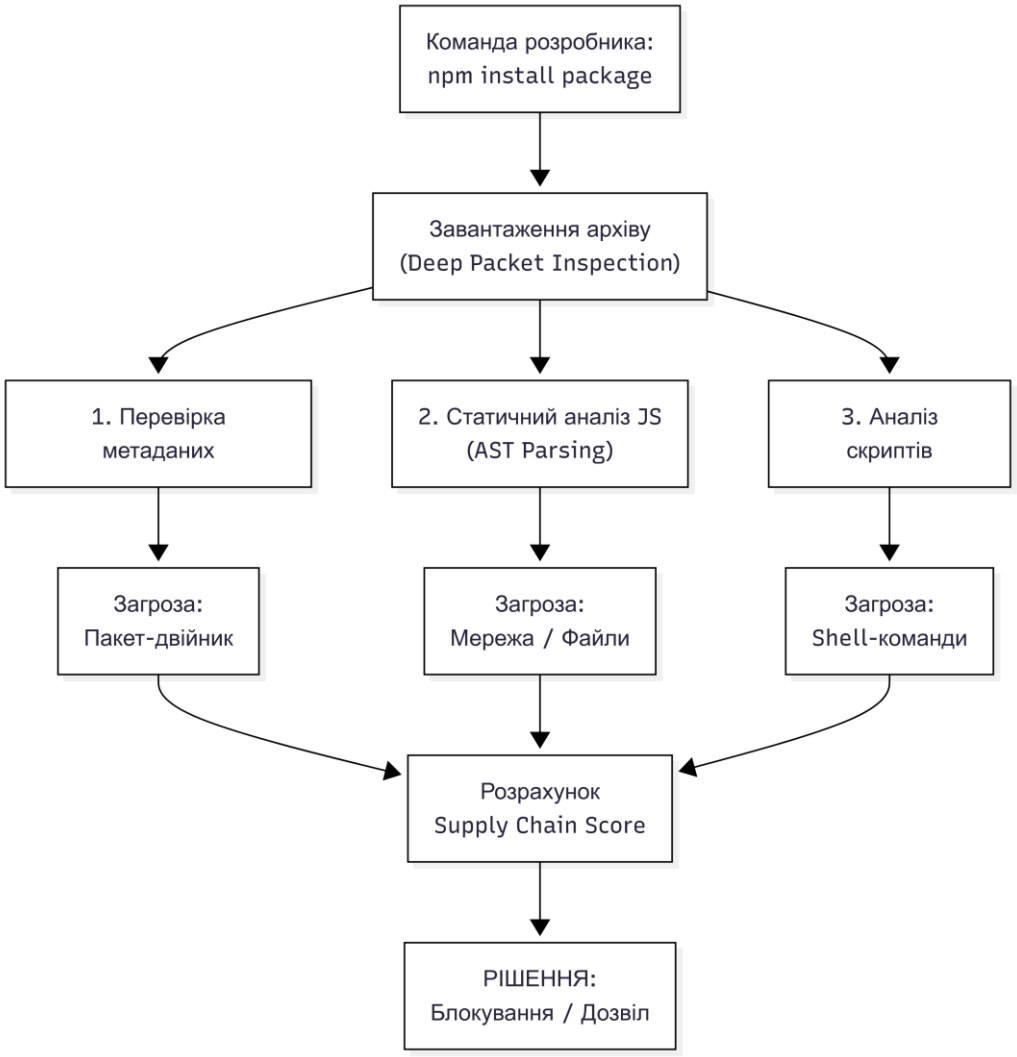

Рисунок 2.5 — Алгоритм поведінкового аналізу пакету на предмет небезпечних можливостей

На етапі перехоплення запиту система фіксує намір розробника встановити новий пакет або оновити існуючий. Після цього завантажується архів пакету і розпочинається етап глибокої інспекції. Він включає статичний аналіз усіх файлів .js та .json без виконання коду. На відміну від SAST-систем, тут аналізується абстрактне синтаксичне дерево (AST) конкретного пакету на наявність небезпечних патернів.

Ключовим етапом є детекція можливостей. Система класифікує поведінку пакету, виявляючи звернення до чутливих API. Якщо код містить виклики для роботи з файловою системою (fs), мережею (net, http) або процесами оболонки (child_process), пакет отримує відповідні маркери.

Фінальним кроком є оцінка ризиків. На основі виявлених можливостей, репутації автора та історії оновлень пакету присвоюється оцінка здоров'я (Supply Chain Score). Якщо пакет запитує привілеї, нетипові для його функціоналу (наприклад, бібліотека для форматування тексту намагається встановити мережеве з'єднання), операція встановлення блокується. Такий підхід дозволяє ефективно виявляти загрози «нульового дня», для яких ще не існує записів у базах вразливостей.

2.5 Узагальнення результатів порівняльного аналізу засобів захисту

На основі проведеного дослідження сучасних інструментів моніторингу безпеки в екосистемі Node.js було сформульовано наступні узагальнення:

– Методи сигнатурного контролю метаданих забезпечують базовий рівень гігієни проекту, проте характеризуються високим рівнем інформаційного шуму та неспроможністю виявляти нові вектори атак.

– Технології поглибленого статичного аналізу з функцією Reachability дозволяють мінімізувати кількість хибних спрацювань шляхом фільтрації вразливостей, що не використовуються у контексті виконання програми.

– Платформи поведінкового моніторингу є критично важливим елементом захисту від атак на ланцюжок постачання, оскільки забезпечують виявлення аномальної активності пакетів без прив'язки до баз відомих вразливостей.

Встановлено, що жоден із розглянутих підходів не є самодостатнім, тому для забезпечення комплексної безпеки необхідно застосування гібридної моделі захисту. Проектування та імплементація такої системи в рамках CI/CD-конвеєра буде розглянута у третьому розділі.

РОЗДІЛ 3 ПРОЕКТУВАННЯ КОМПЛЕКСНОЇ СИСТЕМИ ЗАХИСТУ ЛАНЦЮЖКА ПОСТАЧАННЯ В ЕКОСИСТЕМІ NODE.JS

Результати аналізу у другому розділі доводять неефективність використання окремих інструментів безпеки. Базові сканери вразливостей, системи аналізу коду та платформи поведінкового моніторингу мають власні обмеження та «сліпі зони». Жоден із цих засобів не здатен самостійно гарантувати цілісність програмного продукту.

Забезпечення надійного захисту від сучасних загроз вимагає переходу до комплексної стратегії. Необхідно поєднати різні методи контролю в єдину структуру.

Цей розділ присвячено проектуванню такої комплексної системи захисту. Головною метою є інтеграція сильних сторін розглянутих інструментів у автоматизований конвеєр розробки (CI/CD). Такий підхід дозволяє створити ешелонову оборону, де кожен етап створення програмного забезпечення підлягає автоматичному контролю. Це дозволяє мінімізувати вплив людського фактору та зупиняти атаки на ранніх стадіях.

3.1 Розробка архітектури системи безпеки в середовищі CI/CD

Фундаментом надійної системи захисту є концепція ешелонованої оборони (Defense in Depth) [17]. Вона передбачає створення кількох незалежних бар'єрів безпеки. Якщо загроза долає один рівень, її повинен зупинити наступний. У розробці програмного забезпечення це реалізується через впровадження автоматичних перевірок на кожному етапі руху коду.

Проектована архітектура розподіляє інструменти захисту на три логічні рівні. Перший рівень охоплює локальне середовище розробника. Другий рівень стосується сервера автоматичної збірки (CI). Третій рівень відповідає за моніторинг вже працюючого продукту. Така структура дозволяє виявляти прості помилки миттєво, а складні атаки блокувати до моменту оновлення основного

коду. Схему запропонованої архітектури та взаємодію її компонентів наведено на рисунку 3.1.

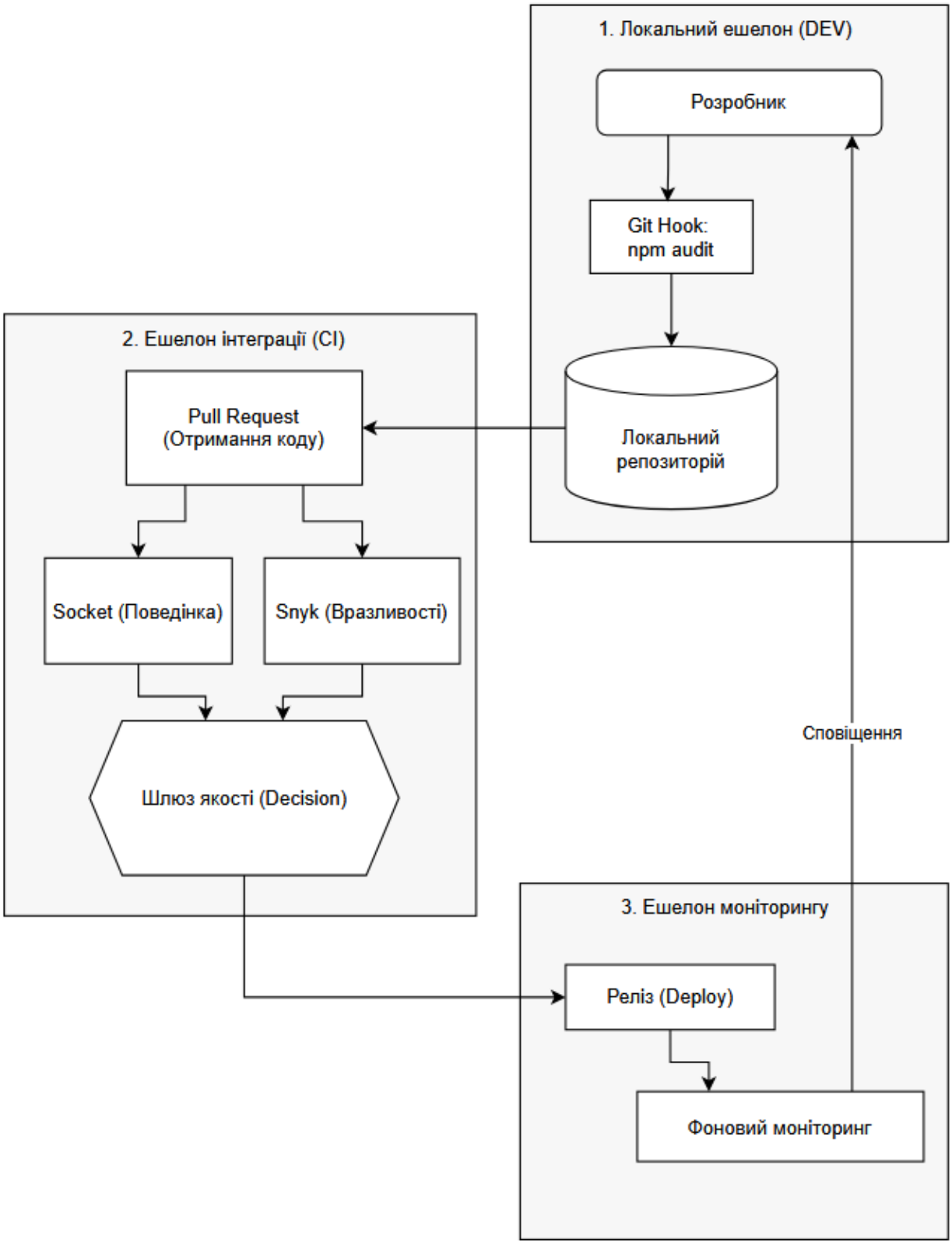

Рисунок 3.1 — Архітектура ешелонованої системи захисту ланцюжка постачання

Функціонування спроектованої системи здійснюється за алгоритмом, що складається з трьох послідовних етапів:

– Забезпечення локального контролю (Pre-Commit Stage).

– Поглиблений аналіз на етапі інтеграції (CI Stage).

– Безперервний моніторинг після релізу (Monitoring Stage).

На першому етапі головною метою є блокування очевидних загроз безпосередньо на робочій станції розробника. Тут застосовуються легкі інструменти сигнатурного аналізу, такі як npm audit. Інтеграція відбувається через систему git-хуків. Якщо розробник намагається додати до репозиторію пакет із критичною вразливістю, система автоматично відхиляє цю операцію. Це дозволяє попередити потрапляння небезпечного коду в історію проекту.

Другий етап виступає центральним бар'єром безпеки. Після відправки змін на сервер запускається комплексне сканування. Воно включає роботу двох паралельних механізмів. Платформа Socket виконує поведінковий аналіз для виявлення шкідливих скриптів та спроб доступу до мережі. Одночасно інструмент Snyk будує граф викликів для перевірки досяжності вразливостей. При виявленні загрози будь-яким із сканерів процес злиття коду блокується.

Третій етап забезпечує захист продукту в довгостроковій перспективі. Оскільки нові вразливості можуть бути виявлені вже після розгортання додатку, система продовжує роботу у фоновому режимі. Інструменти класу Dependabot щоденно перевіряють актуальність встановлених пакетів. У разі появи нових записів у базах загроз система автоматично формує запит на оновлення відповідного компонента.

3.2 Налаштування політик безпеки та алгоритмів блокування загроз

Ефективність спроектованої архітектури залежить від коректності налаштування правил фільтрації. Якщо політики будуть занадто м'якими, загрози потраплять у продакшн. Якщо занадто жорсткими — процес розробки зупиниться через постійні блокування легітимних інструментів. Тому на цьому

етапі розробляється конфігурація так званих «шлюзів якості» (Quality Gates), які функціонують в автоматичному режимі. Для реалізації гібридного захисту налаштовуються два незалежні набори правил: політики поведінкового контролю (для Socket) та критерії допустимості вразливостей (для Snyk).

Основою конфігурації для платформи Socket є файл socket.yml, що розміщується в корені репозиторію. У рамках дослідження розроблено профіль «Zero Trust» (Нульова довіра). Його особливість полягає в тому, що він блокує будь-які потенційно небезпечні можливості пакетів (доступ до мережі, файлів, запуск скриптів), якщо вони не були явно дозволені [18]. Алгоритм блокування базується на трьох критичних правилах:

– Блокування скриптів інсталяції (install-scripts).

– Заборона прихованої телеметрії (telemetry).

– Контроль доступу до мережі (network), файлової системи (filesystem) та змінних середовища (env).

Перше обмеження є найбільш пріоритетним. Будь-який пакет, що містить хуки preinstall або postinstall, автоматично позначається системою як небезпечний. Це правило дозволяє нейтралізувати більшість атак типу Malware Injection. Друге та третє обмеження захищають конфіденційність даних.

Для забезпечення безперебійної роботи легітимних пакетів (наприклад, бібліотеки axios, яка за своєю суттю повинна мати доступ до мережі), у конфігурації застосовано секцію overrides. Приклад розробленої комплексної конфігурації наведено у Лістингу 3.1.

Лістинг 3.1 – Розширена конфігурація політик безпеки (socket.yml) з механізмом виключень

```
version: 2
projectIgnorePaths:
  - "test/**"
  - "docs/**"
  - "*.test.js"
issueRules:
  "src/install-scripts": error
  "src/telemetry": error
  "src/eval": error
```

Продовження лістингу 3.1

```
"src/network": error
  "src/filesystem": error
  "src/env": error
overrides:
  "axios":
    "src/network": ignore
  "dotenv":
    "src/env": ignore
  "winston":
    "src/filesystem": ignore
```

У наведеному фрагменті секція issueRules встановлює глобальну заборону на небезпечні дії. Проте секція overrides дозволяє гнучко адаптувати політику під потреби проекту, дозволяючи мережеву активність лише для пакету axios та доступ до змінних середовища для dotenv. Такий підхід мінімізує кількість хибних блокувань (False Positives).

Для інструменту Snyk налаштовується логіка, спрямована на зменшення «інформаційного шуму». Політика безпеки для статичного аналізу базується на наступних принципах:

– Фільтрація за рівнем досяжності.

– Виключення допоміжних компонентів.

– Пріоритизація за рівнем критичності.

Принцип фільтрації за рівнем досяжності означає, що блокування відбувається лише у випадку, якщо вразливість має статус Reachable. Це гарантує, що розробники не витрачатимуть час на виправлення "мертвого" коду, який ніколи не виконується в додатку. Виключення допоміжних компонентів дозволяє ігнорувати вразливості в пакетах, що використовуються виключно для тестування або лінтингу (dev-dependencies).

Технічна реалізація цих правил здійснюється не через одиничну команду, а через спеціалізований скрипт автоматизації (wrapper script) для середовища CI/CD. Цей скрипт забезпечує запуск сканера з необхідними параметрами, генерацію детального звіту у форматі JSON для подальшого аудиту та контроль коду завершення процесу. Код реалізації наведено у Лістингу 3.2.

Лістинг 3.2 – Скрипт автоматизації перевірки вразливостей з генерацією артефактів звітності

```
#!/bin/bash

export SNYK_TOKEN=$SECRET_API_TOKEN
LOG_FILE="./snyk_security_report.json"

echo "Starting Contextual Security Analysis..."

snyk test --all-projects \
    --severity-threshold=high \
    --reachable \
    --dev=false \
    --fail-on=all \
    --json > $LOG_FILE

EXIT_CODE=$?

if [ $EXIT_CODE -eq 0 ]; then
  echo "[SUCCESS] No critical reachable vulnerabilities found."
  exit 0
else
  echo "[FAILURE] Critical threats detected! Pipeline stopped."
  echo "See details in $LOG_FILE"
  jq '.uniqueCount' $LOG_FILE
  exit 1
fi
```

У наведеному скрипті реалізовано розширену логіку перевірки. Параметр --severity-threshold=high фокусує увагу на критичних загрозах, а --reachable активує аналіз графа викликів. Важливим доповненням є збереження результатів у файл $LOG_FILE у форматі JSON, що дозволяє зберігати історію перевірок як артефакти збірки. Скрипт автоматично перериває виконання конвеєра (повертає exit 1), якщо знайдено хоча б одну проблему, що відповідає заданим критеріям.

Інтеграція описаних політик формує єдиний алгоритм прийняття рішення: процес перевірки вважається успішним лише за умови, що поведінковий аналіз не виявив аномалій, а статичний аналіз підтвердив відсутність критичних досяжних вразливостей. Додатково, інтеграція з утилітою jq забезпечує виведення короткої статистики безпосередньо в консоль, що значно спрощує первинний аналіз причин зупинки збірки без необхідності ручного розбору JSON-файлу.

3.3 Порівняльний аналіз функціональних можливостей засобів захисту

Обґрунтування доцільності впровадження спроектованої в підрозділі 3.1 архітектури вимагає детального аналізу покриття векторів атак різними класами інструментів. Для цього проведено порівняння функціональних можливостей трьох ешелонів захисту: сигнатурного аналізу (npm audit), контекстного статичного аналізу (Snyk) та поведінкового моніторингу (Socket). Зазначені інструменти репрезентують принципово різні підходи до виявлення небезпек, що дозволяє виявити специфічні переваги та недоліки кожного методу в умовах реальних інцидентів. Аналіз проводився за критерієм здатності інструменту протидіяти п'яти основним типам загроз, актуальним для екосистеми Node.js у 2024-2025 роках:

– Відомі вразливості (Known CVE) [19].

– Транзитивні вразливості (Indirect Dependencies).

– Атаки на ланцюжок постачання (Malware Injection).

– Тайпосквотінг (Typosquatting).

– Ризики ліцензійної чистоти (License Compliance).

Для підтвердження результатів порівняльного аналізу змодельовано роботу спроектованої ешелонованої системи захисту в тестовому CI/CD-конвеєрі. Метою експерименту є демонстрація того, які типи загроз виявляються або залишаються непоміченими при використанні кожного з методів аналізу окремо. Такий підхід дозволяє доповнити теоретичне порівняння інструментів практичною ілюстрацією їхньої роботи в автоматизованому процесі розробки. Моделювання виконувалося в умовах безперервної інтеграції, де всі перевірки здійснюються автоматично як окремі етапи CI/CD-конвеєра. Це дає змогу оцінити ефективність кожного механізму з точки зору DevSecOps-практик та мінімізувати вплив людського фактору на прийняття рішень. На першому етапі виконується сигнатурний аналіз залежностей за допомогою npm audit. Даний механізм перевіряє проєкт на наявність відомих вразливостей із ідентифікаторами CVE шляхом зіставлення використаних версій пакетів із

базами даних загроз. Результат виконання цього етапу в середовищі CI/CD наведено на рисунку 3.2.

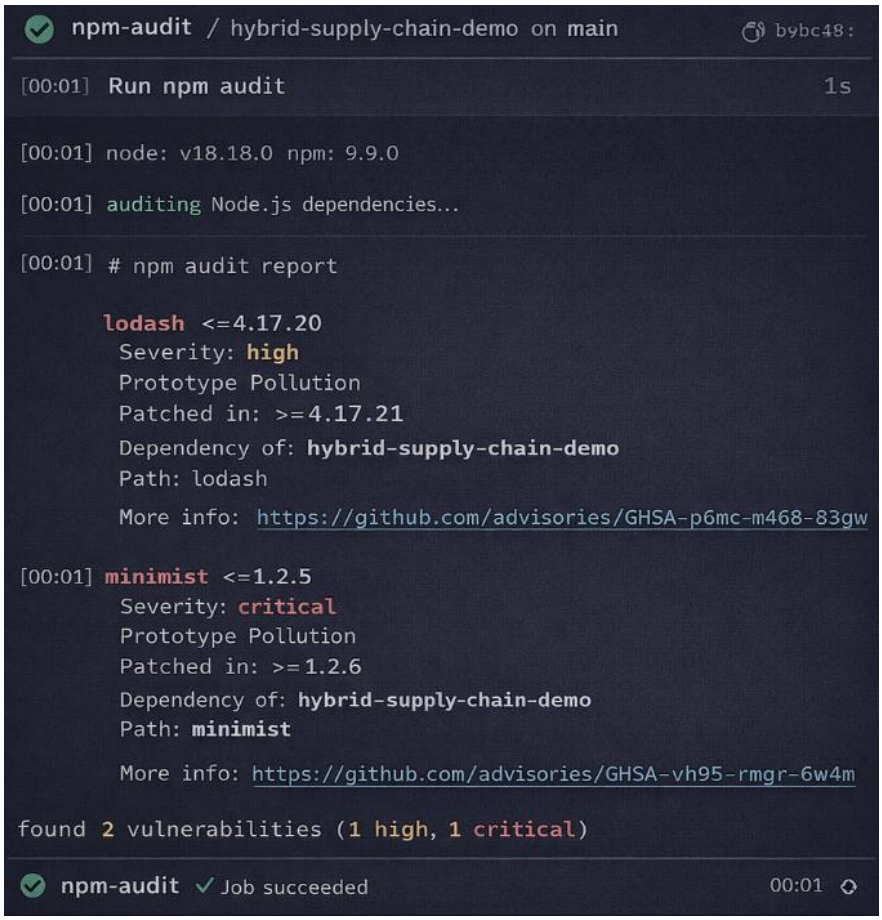

Рисунок 3.2 — Результат автоматизованого сигнатурного аналізу залежностей у CI/CD-конвеєрі за допомогою npm audit

Другий етап захисту реалізує контекстний статичний аналіз із використанням технології Reachability. На цьому рівні система визначає, чи є вразливий код фактично досяжним у потоках виконання додатку, що дозволяє зменшити кількість хибних спрацювань та сфокусувати увагу лише на реальних загрозах. Такий підхід дозволяє уникнути блокування збірки через вразливості в неексплуатованих компонентах та зосередити ресурси команди на усуненні дійсно критичних проблем. Водночас контекстний аналіз не враховує поведінкові аспекти роботи пакетів, що обмежує його ефективність у протидії атакам ланцюжка постачання. Результат роботи механізму у CI/CD-конвеєрі показано на рисунку 3.3.

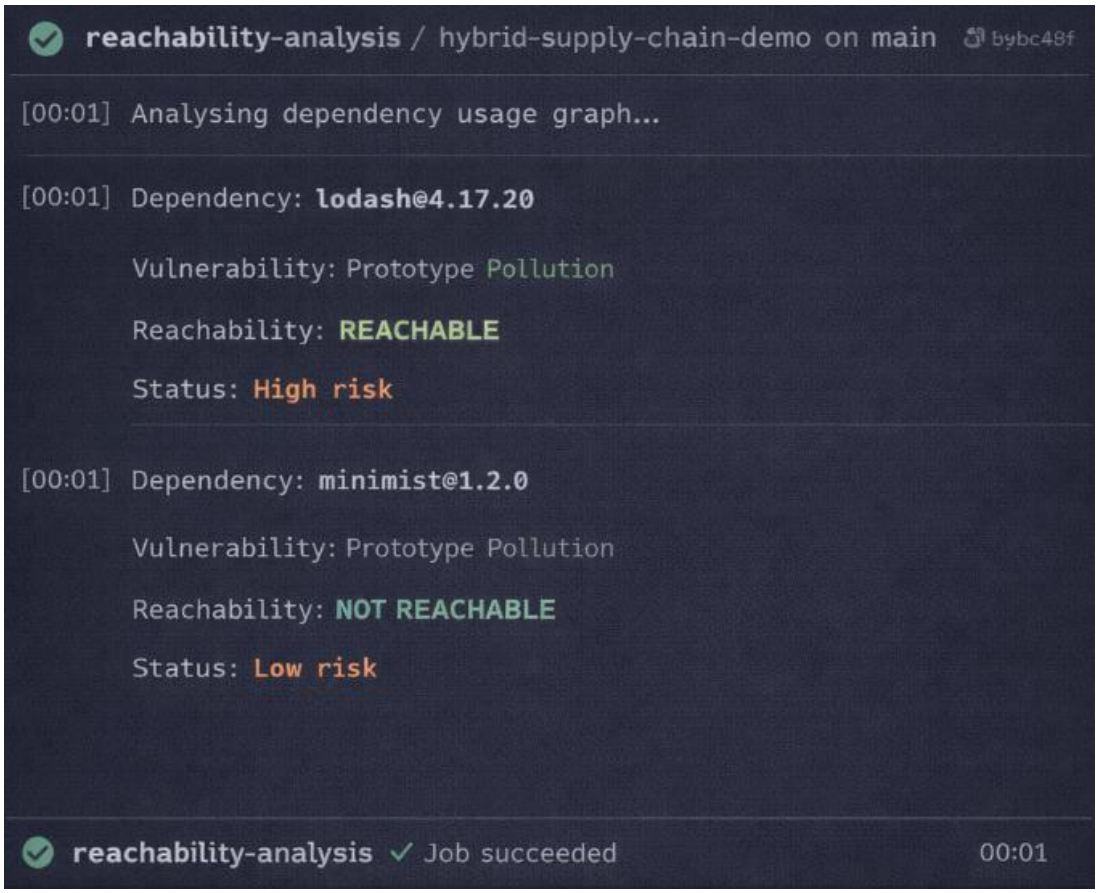

Рисунок 3.3 — Результат контекстного статичного аналізу залежностей (Reachability Analysis) у CI/CD-конвеєрі

Фінальний етап перевірки базується на поведінковому аналізі пакетів та реалізує політики контролю ланцюжка постачання. Його основним завданням є виявлення шкідливих або нетипових дій, таких як виконання інсталяційних скриптів, доступ до мережі, файлової системи або змінних середовища, незалежно від наявності відповідних записів у базах CVE. Аналіз здійснюється шляхом оцінки реальних можливостей пакета та їх відповідності заявленому функціональному призначенню. На відміну від сигнатурних та статичних методів, поведінковий підхід дозволяє ефективно виявляти загрози нульового дня та атаки типу Malware Injection, які не можуть бути ідентифіковані за допомогою баз відомих вразливостей. У разі виявлення аномальної або потенційно небезпечної поведінки система автоматично застосовує політику блокування. Приклад зупинки процесу збірки на цьому етапі наведено на рисунку 3.4.

Рисунок 3.4 — Блокування шкідливого пакета на етапі CI/CD в результаті поведінкового аналізу ланцюжка постачання

Наведені результати демонструють, що кожен із розглянутих методів аналізу ефективний лише в межах власного класу загроз. Сигнатурний аналіз дозволяє виявити відомі вразливості, однак не враховує їх можливості експлуатації та ігнорує поведінкові атаки. Контекстний статичний аналіз зменшує кількість хибних спрацювань, проте не здатен виявляти шкідливі пакети без сигнатур. Поведінковий аналіз, у свою чергу, блокує загрози нульового дня, але без попередньої фільтрації може бути надмірно жорстким. Для узагальнення отриманих результатів та систематизації можливостей інструментів було виконано порівняльний аналіз, результати якого наведено у таблиці 3.1.

Результати порівняльного аналізу зведено у матрицю покриття загроз (Таблиця 3.1), яка відображає здатність кожного з розглянутих інструментів протидіяти основним векторам атак у межах екосистеми Node.js. У таблиці систематизовано функціональні можливості сигнатурного, контекстного та поведінкового підходів з точки зору повноти покриття загроз, що дозволяє наочно оцінити сильні та слабкі сторони кожного методу. Такий формат подання

результатів спрощує порівняння інструментів та слугує основою для подальшого обґрунтування доцільності їх інтеграції у єдину гібридну модель захисту.

Таблиця 3.1 – Матриця покриття векторів атак інструментами моніторингу

Вектор атаки	npm audit (Сигнатурний)	Snyk (Контекстний)	Socket (Поведінковий)
Відомі вразливості (CVE)	Базовий рівень	Високий рівень	Обмежений рівень
Аналіз досяжності коду	Не підтримується	Повна підтримка	Не підтримується
Шкідливі скрипти (Malware)	Не виявляє	Низька ефективність	Висока ефективність
Тайпосквотінг	Не виявляє	Не виявляє	Ефективне виявлення
Прихована телеметрія	Не контролює	Не контролює	Блокування активності
Ліцензійні ризики	Базовий аналіз	Глибокий аналіз	Базовий аналіз

На основі таблиці 3.1 та результатів експериментального моделювання можна зробити наступні висновки щодо практичного застосування кожного інструменту. Інструмент npm audit є ефективним виключно як базовий засіб інвентаризації. Його головна перевага — нативність (вбудований у Node.js), проте він працює лише з метаданими. Це означає, що він не бачить різниці між бібліотекою, яка просто лежить у папці node_modules, і бібліотекою, яка реально виконується. Як наслідок, інструмент не здатний захистити від нових загроз (0-day) і генерує високий рівень хибних попереджень.

Snyk закриває головний недолік сигнатурного методу — відсутність контексту. Завдяки побудові графа викликів (Call Graph), він дозволяє відсіяти "мертвий код". Це робить його ідеальним інструментом для роботи з відомими вразливостями (CVE), оскільки він фокусує увагу розробника тільки на реальних проблемах. Однак, як і npm audit, він покладається на бази даних, тому є

неефективним проти принципово нових шкідливих пакетів, які ще не були досліджені експертами.

Socket працює в ортогональній площині. Він не шукає помилки в коді, а аналізує поведінку. Це єдиний інструмент, здатний зупинити атаку типу Malware Injection (коли в пакет вбудовано вірус), оскільки він блокує сам факт виконання підозрілих дій (доступ до мережі, файлів). Проте, використовувати лише Socket недостатньо, оскільки він менш ефективний у глибокому аналізі старих вразливостей (CVE) у транзитивних залежностях. Таким чином, аналіз доводить, що жоден із інструментів не покриває 100% матриці загроз самостійно. Це зумовлює об'єктивну необхідність інтеграції розглянутих засобів у єдину ешелоновану систему, де функціональні обмеження одного компонента компенсуються сильними сторонами іншого, забезпечуючи безперервний захист на всіх рівнях абстракції.

3.4 Обґрунтування ефективності та доцільності впровадження гібридної моделі захисту

На основі проведеного аналізу та результатів експериментального моделювання стає очевидною необхідність інтеграції розглянутих інструментів у єдину систему захисту. Використання кожного з них окремо не забезпечує повного покриття актуальних векторів атак, тоді як їх поєднання дозволяє компенсувати функціональні обмеження окремих методів. Практична цінність запропонованої в роботі архітектури полягає у формуванні багаторівневої системи фільтрації загроз у CI/CD-конвеєрі, яка включає:

– Базовий сигнатурний контроль залежностей.

– Контекстний аналіз вразливостей.

– Поведінковий контроль ланцюжка постачання.

Перший рівень реалізує базовий сигнатурний контроль залежностей за допомогою npm audit. Він використовується для швидкої ідентифікації відомих

вразливостей (CVE) на ранніх етапах розробки, зокрема у локальному середовищі розробника та під час початкових перевірок у CI/CD.

Другий рівень забезпечує контекстний аналіз вразливостей за допомогою технології Reachability (Snyk) та доповнює сигнатурний підхід перевіркою фактичної можливості експлуатації коду. Пакети, що пройшли базовий контроль, аналізуються на наявність помилок, які реально можуть бути використані через існуючі шляхи виконання.

Третій рівень реалізує поведінковий контроль ланцюжка постачання та виступає фінальним бар'єром захисту. Він застосовується до пакетів, які успішно пройшли попередні перевірки, і забезпечує виявлення загроз, що не можуть бути ідентифіковані сигнатурними або статичними методами.

Таким чином, запропонована гібридна модель не є простим набором інструментів, а формує керований алгоритм прийняття рішень у CI/CD-конвеєрі. Саме поєднання сигнатурного, контекстного та поведінкового аналізу дозволяє досягти значно вищого рівня захисту порівняно з використанням кожного з методів окремо, що відповідає сучасним підходам до управління ризиками в DevSecOps-практиках [20].

4 ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

4.1 Охорона праці

При дослідженні сучасних інструментів моніторингу вразливих npm-пакетів у Node.js-проектах технічне забезпечення передбачає інтенсивну експлуатацію персональних електронно-обчислювальних машин, серверного обладнання та периферійних пристроїв. Живлення вказаної техніки здійснюється від однофазної електричної мережі змінного струму промислової частоти 50 Гц напругою 220 В. Оскільки тіло людини є провідником електричного струму, експлуатація електроустановок пов’язана з ризиком ураження, що може призвести до електричних травм або ударів різного ступеня тяжкості. Крім того, робота сучасного комп’ютерного обладнання супроводжується генеруванням електромагнітних полів широкого спектра частот та накопиченням зарядів статичної електрики, що вимагає впровадження комплексних захисних заходів. Організація системи електробезпеки на робочому місці регламентується НПАОП 40.1-1.21-98 «Правила безпечної експлуатації електроустановок споживачів» [21] та ПУЕ:2017 «Правила улаштування електроустановок» [22].

Відповідно до вимог стандарту ДСТУ EN 61140:2019 [23], персональні комп’ютери класифікуються як електрообладнання І класу захисту. Це означає, що безпека персоналу забезпечується не лише основною ізоляцією струмопровідних частин, але й обов’язковим приєднанням усіх доступних для дотику металевих неструмопровідних частин до захисного провідника стаціонарної проводки.

Для захисту людини від ураження електричним струмом при прямому або непрямому дотику до частин, що перебувають під напругою, реалізовано наступні технічні заходи:

– Захисне заземлення (занулення) металевих корпусів обладнання.

– Автоматичне вимкнення живлення пристроями захисного вимкнення (ПЗВ).

– Забезпечення недоступності струмопровідних частин та подвійна ізоляція.

Реалізація захисного заземлення є основним заходом безпеки для обладнання I класу. Електроживлення робочих місць здійснюється через електричну мережу з трьома провідниками мережу за системою TN-S або TN-C-S, яка включає фазний провідник (L), робочий нульовий провідник (N) та захисний нульовий провідник (PE). Металеві корпуси системних блоків, моніторів та іншого обладнання через заземлюючий контакт штепсельних розеток із заземлюючим контактом мають надійне гальванічне з'єднання з PE-провідником. Фізичний зміст цього заходу полягає у створенні шляху з малим електричним опором для протікання струму. У випадку пробою ізоляції (наприклад, замикання фази на корпус системного блоку) струм не піде через тіло людини, опір якого становить близько 1000 Ом, а потече через захисний провідник, опір якого значно менший (до 4 Ом) [24]. Це перетворює замикання на корпус в однофазне коротке замикання, що викликає миттєве зростання сили струму та спрацювання автоматичного вимикача, який знеструмлює пошкоджену ділянку.

Додатковим, але важливим етапом захисту є автоматичне вимкнення живлення, що реалізується встановленням у розподільчих щитах пристроїв захисного вимкнення (ПЗВ) або диференційних автоматів. Принцип дії цих приладів базується на постійному порівнянні струму, що входить у коло через фазний провідник, зі струмом, що повертається через нульовий провідник. У нормальному режимі роботи ці струми рівні, і векторна сума магнітних потоків у трансформаторі струму дорівнює нулю. Однак, при виникненні витоку струму (наприклад, при дотику людини до струмопровідної частини або погіршенні стану ізоляції) виникає дисбаланс. Якщо диференційний струм витоку перевищує порогове значення (зазвичай 30 мА для ліній живлення штепсельних розеток), ПЗВ розмикає електричне коло за час, що не перевищує 0,4 секунди. Такий час реакції є достатнім для запобігання фібриляції шлуночків серця людини.

Крім того, комп’ютерна техніка є джерелом електромагнітних полів (ЕМП) широкого частотного спектра, які можуть негативно впливати на нервову, імунну та ендокринну системи організму. Нормування рівнів ЕМП на робочих місцях здійснюється відповідно до ДСанПіН 3.3.6.096-2002 [25]. Допустимі рівні напруженості електричного поля на робочому місці не повинні перевищувати 25 В/м у діапазоні частот 5 Гц – 2 кГц та 2,5 В/м у діапазоні 2 кГц – 400 кГц. Гранично допустима щільність магнітного потоку становить 250 нТл.

Для зниження рівня електромагнітного випромінювання до безпечних значень застосовуються такі методи захисту:

– Використання сертифікованого обладнання з вбудованим екрануванням.

– Раціональне розміщення техніки та організація кабельної мережі.

– Дотримання безпечних відстаней до джерел випромінювання.

Зниження напруженості полів досягається завдяки використанню системних блоків та моніторів, що відповідають міжнародним стандартам безпеки (TCO Certified). Металеві корпуси системних блоків діють як екран Фарадея, ефективно поглинаючи та відводячи електричну складову поля в землю через систему заземлення. Дотримання рекомендованої відстані від екрана до очей (600–700 мм) також є ефективним засобом захисту, оскільки напруженість поля швидко зменшується пропорційно квадрату відстані від джерела.

Ще одним фактором небезпеки є статична електрика, що виникає внаслідок тертя шарів одягу із синтетичних матеріалів, контакту з діелектричними поверхнями меблів та роботи рухомих частин обладнання (вентиляторів). Накопичення електростатичних зарядів може призводити до іскрових розрядів, які не лише викликають рефлекторну реакцію та дискомфорт у персоналу, але й здатні вивести з ладу чутливі електронні компоненти мікросхем.

Для нейтралізації статичної електрики застосовується комплекс наступних заходів:

– Підтримання нормованих параметрів мікроклімату.

– Забезпечення безперервного електричного ланцюга заземлення.

– Використання антистатичних матеріалів в оздобленні приміщення.

Найбільш ефективним та доступним методом боротьби з електростатикою є підтримання відносної вологості повітря в приміщенні на рівні 40–60 %. При такій вологості на поверхнях діелектриків утворюється мікроскопічна плівка конденсату, яка підвищує поверхневу провідність матеріалів і сприяє природному стіканню зарядів у землю, запобігаючи їх накопиченню до небезпечних потенціалів. У поєднанні з надійним заземленням усіх металевих частин обладнання та використанням антистатичних покриттів для підлоги, це дозволяє повністю усунути ризики, пов’язані зі статичною електрикою.

Таким чином, при дослідженні та виконанні кваліфікаційної роботи було в повному обсязі враховано правила електробезпеки, захисту від електромагнітних полів та статичної електрики. Дотримання цих технічних та організаційних заходів дозволило створити безпечні умови праці, мінімізувати вплив шкідливих факторів та забезпечити надійне функціонування обчислювальної техніки для успішного виконання поставлених наукових завдань.

4.2 Забезпечення цивільного захисту та пожежної безпеки в умовах надзвичайних ситуацій

Забезпечення цивільного захисту та пожежної безпеки під час виконання науково-дослідної роботи є важливою складовою загальної системи безпеки життєдіяльності. Відповідно до положень Кодексу цивільного захисту України [26], цивільний захист охоплює комплекс організаційних, інженерно-технічних та профілактичних заходів, спрямованих на запобігання надзвичайним ситуаціях, зменшення їх негативних наслідків, а також захист життя і здоров’я людини. У процесі виконання магістерського дослідження, що здійснюється з використанням обчислювальної техніки та електроустановок, існує низка потенційних загроз техногенного та воєнного характеру [28], які потребують завчасної підготовки та чіткого дотримання встановлених правил безпеки [29].

Організація цивільного захисту на об’єкті, де виконується науково-дослідна робота, передбачає обов’язкове проходження здобувачем вступного та

первинного інструктажів з питань охорони праці, пожежної безпеки та дій у надзвичайних ситуаціях. Під час інструктажу дослідник повинен бути ознайомлений з планом евакуації будівлі, сигналами оповіщення цивільного захисту, правилами користування первинними засобами пожежогасіння та місцями їх розташування. Особливу увагу слід приділяти інформуванню про розташування захисних споруд цивільного захисту та безпечних маршрутів прямування до них, що є критично важливим у разі виникнення надзвичайних ситуацій воєнного характеру.

Робоче місце дослідника, оснащене персональним комп'ютером, мережевим обладнанням, джерелами безперебійного живлення та іншими електронними пристроями, відповідно до чинних Правил пожежної безпеки в Україні (НАПБ А.01.001-2014) [27], належить до приміщень з підвищеною пожежною небезпекою. Основними факторами ризику є наявність електроустановок під напругою, значна кількість кабельних з'єднань, а також можливість перегріву обладнання внаслідок тривалої безперервної роботи або недостатньої вентиляції. Додаткову небезпеку становить накопичення пилу в системах охолодження, що може призвести до порушення теплового режиму та виникнення осередків займання.

З метою запобігання пожежам необхідно дотримуватися комплексу профілактичних заходів, зокрема здійснювати регулярний візуальний огляд стану електричних кабелів, розеток і подовжувачів, не допускати використання пошкодженого або саморобного електрообладнання, а також уникати перевантаження електромережі шляхом підключення великої кількості споживачів до одного джерела живлення. У робочих приміщеннях забороняється використання побутових електронагрівальних приладів, не передбачених умовами роботи, а також залишення ввімкненого обладнання без нагляду після завершення робочого часу.

У разі виникнення ознак пожежі або загоряння, таких як поява диму, запах горілої ізоляції, іскріння або підвищення температури обладнання, персонал повинен діяти негайно та послідовно. Алгоритм дій передбачає:

– Повідомлення пожежно-рятувальної служби за номером «101» із зазначенням адреси та характеру загрози.

– Оповіщення осіб, які перебувають у приміщенні, про небезпеку.

– Знеструмлення електрообладнання шляхом вимкнення його з розеток або через розподільний щит, якщо це не створює загрози життю.

– Організовану евакуацію згідно з планом, використовуючи сходові клітини та аварійні виходи.

– Гасіння осередку пожежі первинними засобами пожежогасіння за умови відсутності безпосередньої небезпеки для життя.

Для ліквідації загорянь електроустановок та комп'ютерної техніки, що перебувають під напругою, необхідно застосовувати вуглекислотні вогнегасники (призначені для гасіння пожеж класу Е), які є безпечними для електронного обладнання та не залишають залишкових забруднень після використання. Застосування водяних або пінних засобів пожежогасіння в таких умовах є забороненим через ризик ураження електричним струмом.

В умовах воєнного стану важливим елементом системи цивільного захисту є чітке виконання дій при оголошенні сигналу оповіщення «Увага всім!». Сигнал повітряної тривоги вимагає негайного припинення виконання будь-яких робіт та переходу до укриття. З огляду на специфіку науково-дослідної діяльності у сфері інформаційних технологій, доцільним є завчасне впровадження заходів щодо захисту результатів дослідження від втрати або пошкодження. У разі оголошення тривоги необхідно оперативно зберегти програмний код, аналітичні матеріали та звітну документацію на захищені носії або хмарні сервіси, після чого повністю вимкнути електроживлення обладнання.

Подальші дії передбачають підготовку до евакуації, зокрема взяття особистих документів, засобів зв'язку та індивідуальної аптечки, а також прямування найкоротшим безпечним маршрутом до найближчого укриття. Перебування в укритті повинно тривати до моменту офіційного оголошення сигналу «Відбій», після чого повернення до робочого приміщення здійснюється лише за відсутності додаткових загроз.

Оскільки науково-дослідна робота пов’язана з експлуатацією електроустановок, існує ймовірність ураження електричним струмом. У такій ситуації першочерговим завданням є негайне припинення дії струму на потерпілого шляхом вимкнення джерела живлення або відокремлення його від струмопровідних частин за допомогою сухих діелектричних предметів. Після цього необхідно оцінити стан потерпілого, перевірити наявність свідомості та дихання і, у разі їх відсутності, розпочати серцево-легеневу реанімацію до прибуття медичної допомоги. Дотримання вимог цивільного захисту та пожежної безпеки, а також систематичне виконання профілактичних заходів і алгоритмів дій у надзвичайних ситуаціях, дозволяє суттєво знизити рівень ризику для життя та здоров’я дослідника, зберегти матеріальні та інформаційні ресурси й забезпечити безперервність науково-дослідного процесу навіть в умовах підвищеної небезпеки.

ВИСНОВКИ

У кваліфікаційній роботі досліджено проблему захисту ланцюжка постачання в екосистемі Node.js та проаналізовано обмеження існуючих засобів моніторингу. Запропоновано та обґрунтовано комплексну стратегію захисту, яка, на відміну від стандартних сигнатурних методів, інтегрує контекстний аналіз досяжності коду та поведінковий контроль залежностей у єдиний автоматизований CI/CD-конвеєр розробки.

Аналітичне порівняння та моделювання сценаріїв реагування підтвердили, що використання гібридної моделі, що поєднує сигнатурний аналіз залежностей, контекстний аналіз досяжності коду та поведінковий контроль, дозволяє нівелювати недоліки окремих інструментів. Доведено, що такий підхід забезпечує проактивне виявлення загроз «нульового дня», таких як впровадження шкідливого коду (malware) та підміна назв пакетів, які залишаються непоміченими для традиційних сканерів вразливостей.

Загалом, застосування запропонованої архітектури та налаштованих політик безпеки забезпечує можливість автоматизувати процеси прийняття рішень щодо блокування небезпечних компонентів. Це дозволяє суттєво знизити рівень інформаційного шуму, вирішити проблему «втоми від безпеки» у розробників та посилити керованість процесів захисту без уповільнення темпів розробки програмного забезпечення.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. The Node.js Project. (2024). Node.js documentation. https://nodejs.org/en/docs/ (дата звернення: 25.09.2025).

2. Sonatype. (2024). State of the software supply chain 2024. https://www.sonatype.com/state-of-the-software-supply-chain (дата звернення: 24.09.2025).

3. Preston-Werner, T. (n.d.). Semantic versioning 2.0.0. Semantic Versioning. https://semver.org/ (дата звернення: 26.09.2025).

4. CISA. (2021). Defending against software supply chain attacks. https://www.cisa.gov/publication/software-supply-chain-attacks (дата звернення: 28.09.2025).

5. NIST. (2025). CVE-2025-55182. National Vulnerability Database. https://nvd.nist.gov/vuln/detail/CVE-2025-55182 (дата звернення: 29.09.2025).

6. Birsan, A. (2021, February 9). Dependency confusion: How I hacked into Apple, Microsoft and dozens of other companies. Medium. https://medium.com/@alex.birsan/dependency-confusion (дата звернення: 02.10.2025).

7. Socket. (2024). Supply chain attack trends: Malware, typosquatting, and more. https://socket.dev/blog (дата звернення: 03.10.2025).

8. CISA. Software Bill of Materials (SBOM). https://www.cisa.gov/sbom (дата звернення: 05.10.2025).

9. OWASP. (2021). OWASP Top 10: 2021. https://owasp.org/Top10/ (дата звернення: 06.10.2025).

10. FIRST. (n.d.). CVSS v3.1: Specification document. https://www.first.org/cvss/v3.1/specification-document (дата звернення: 07.10.2025).

11. Загородна, Н. В. (2023). Методи та засоби забезпечення інформаційної безпеки в сучасних комп’ютерних системах [Навчальний посібник]. ТНТУ.

12. npm. (n.d.). npm-audit: Run a security audit. npm Docs. https://docs.npmjs.com/cli/v8/commands/npm-audit (дата звернення: 16.10.2025).

13. GitHub. (n.d.). GitHub advisory database. https://github.com/advisories (дата звернення: 17.10.2025).

14. GitHub. (n.d.). Keeping your dependencies secure automatically. GitHub Docs. https://docs.github.com/en/code-security/dependabot (дата звернення: 17.10.2025).

15. Snyk. (2024). The state of open source security 2024. https://snyk.io/reports/open-source-security/ (дата звернення: 18.10.2025).

16. Snyk. (2023). Reachability analysis: How to prioritize vulnerabilities. https://snyk.io/learn/reachability-analysis/ (дата звернення: 19.10.2025).

17. NIST. (2022). Secure software development framework (SSDF) version 1.1 (SP 800-218). National Institute of Standards and Technology. https://csrc.nist.gov/pubs/sp/800/218/final (дата звернення: 02.11.2025).

18. NIST. (2020). Zero trust architecture (SP 800-207). National Institute of Standards and Technology. https://csrc.nist.gov/publications/detail/sp/800-207/final (дата звернення: 05.11.2025).

19. Tylor, M. (2020). Prototype pollution in Node.js. Medium. https://medium.com/node-security/prototype-pollution (дата звернення: 10.11.2025).

20. ISO/IEC. (2013). Information technology — Security techniques — Information security management systems — Requirements (ISO/IEC 27001:2013).

21. Держнаглядохоронпраці. (1998). Правила безпечної експлуатації електроустановок споживачів (НПАОП 40.1-1.21-98).

22. Міненерговугілля України. (2017). Правила улаштування електроустановок. Форт.

23. ДП "УкрНДНЦ". (2019). Захист від ураження електричним струмом. Загальні аспекти щодо установок та обладнання (ДСТУ EN 61140:2019).

24. Осухівська, Г. М. (2022). Охорона праці в галузі: конспект лекцій. ТНТУ.

25. МОЗ України. (2002). Державні санітарні норми і правила при роботі з джерелами електромагнітних полів (ДСанПіН 3.3.6.096-2002). https://zakon.rada.gov.ua/laws/show/z0203-03 (дата звернення: 01.12.2025).

26. Верховна Рада України. (2012). Кодекс цивільного захисту України (№ 5403-VI). https://zakon.rada.gov.ua/laws/show/5403-17 (дата звернення: 02.12.2025).

27. МВС України. (2014). Правила пожежної безпеки в Україні (НАПБ А.01.001-2014). https://zakon.rada.gov.ua/laws/show/z0252-15 (дата звернення: 02.12.2025).

28. Стручок, В. С. (2022). Безпека в надзвичайних ситуаціях: методичний посібник. ФОП Паляниця В. А. https://elartu.tntu.edu.ua/handle/lib/39196 (дата звернення: 02.12.2025).

29. Стручок, В. С. (2022). Техноекологія та цивільна безпека. Частина «Цивільна безпека»: навчальний посібник. ФОП Паляниця В. А. http://elartu.tntu.edu.ua/handle/lib/39424 (дата звернення: 02.12.2025).

30. Kozak, R., Skorenkyy, Y., Kramar, O., Lechachenko, T., & Brevus, H. (2025). Cybersecurity provisioning for Industry 4.0 digital twin with AR components. In *CEUR Workshop Proceedings, 3rd International Workshop on Computer Information Technologies in Industry* (Vol. 4, pp. 166-178).

31. Derkach, M., Matiuk, D., Skarga-Bandurova, I., & Zagorodna, N. (2025). CrypticWave: A zero-persistence ephemeral messaging system with client-side encryption.

32. Stadnyk, M., Fryz, M., & Scherbak, L. The Feature Extraction and Estimation of a Steady-state Visual Evoked Potential by the Karhunen-Loeve Expansion. *Eastern-European Journal of Enterprise Technologies*, *1*(4), 56-62.

33. Revniuk, O., Zagorodna, N., Kozak, R., & Yavorskyy, B. (2025). Development of an information system for the quantitative assessment of web application security based on the OWASP ASVS standard. *Вісник Тернопільського національного технічного університету*, *118*(2), 56-65.

Додаток А Публікація

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ТЕРНОПІЛЬСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ імені ІВАНА ПУЛЮЯ
НАУКОВЕ ТОВАРИСТВО ім. ШЕВЧЕНКА

XIII

НАУКОВО-ТЕХНІЧНА
К О Н Ф Е Р Е Н Ц І Я

Інформаційні моделі, системи та технології

17-18 грудня 2025 року

ТЕРНОПІЛЬ – 2025

УДК 004.056.5
М. Стебельський; Н. Загородна, к.т.н., доцент
(Тернопільський національний технічний університет імені Івана Пулюя, Україна)

ПРОБЛЕМАТИКА ІНТЕГРАЦІЇ ЗАСОБІВ SCA У ПРОЦЕСИ DEVSECOPS ДЛЯ NODE.JS ПРОЕКТІВ

UDC 004.056.5
M. Stebelskyi; N. Zagorodna, Ph.D, Assoc. Prof

PROBLEMATICS OF INTEGRATING SCA TOOLS INTO DEVSECOPS PROCESSES FOR NODE.JS PROJECTS

У сфері сучасної веб-розробки широкого розповсюдження набула платформа Node.js. Вона являє собою середовище виконання JavaScript, що дозволяє створювати високопродуктивні та масштабовані мережеві системи. Невід'ємною складовою цієї інфраструктури виступає пакетний менеджер npm, який забезпечує доступ до найбільшого у світі реєстру бібліотек відкритого коду. Проте широке використання сторонніх модулів та глибока вкладеність залежностей роблять проекти критично вразливими до атак на ланцюжок постачання (Supply Chain Attacks). Оскільки значна частина коду базується на open-source компонентах, забезпечення автоматизованого контролю їх безпеки є пріоритетним завданням, адже ручний аудит у таких масштабах стає неможливим [1].

Проблематика багатьох існуючих інструментів SCA (Software Composition Analysis) полягає у фокусуванні на кількісних показниках та високому рівні «шуму». Більшість стандартних рішень здійснюють поверхневий огляд, не перевіряючи, чи дійсно вразливий компонент використовується у робочому коді програми. Це призводить до блокування релізів через бібліотеки, які фактично не становлять загрози в конкретному проекті. Крім того, типові сканери реагують лише на вже задокументовані в офіційних базах вразливості, залишаючи поза увагою приховані загрози, такі як автоматичний запуск шкідливих скриптів безпосередньо під час встановлення пакетів. Відсутність розумної пріоритезації ризиків ускладнює роботу в швидких циклах розробки, що призводить до явища «втоми від сповіщень», коли розробники ігнорують звіти безпеки заради збереження темпів роботи [2].

Для побудови надійного захисту необхідний перехід від формальної наявності сканера до аналізу його реальної ефективності в умовах CI/CD. Ключовими критеріями стають не лише повнота бази вразливостей, а й точність побудови дерева залежностей, швидкість роботи та здатність мінімізувати помилкові тривоги. Такий підхід дозволяє перетворити моніторинг із блокуючого фактора на інструмент превентивного виявлення загроз [3].

Проведення комплексного порівняльного аналізу функціональності сучасних open-source та пропрієтарних рішень є необхідним етапом для побудови безпечної архітектури. Розробка методики вибору та інтеграції інструментів моніторингу, що базується на об'єктивних метриках точності та продуктивності, дозволяє суттєво знизити ризики компрометації Node.js-додатків та оптимізувати процеси DevSecOps.

Література

1. Supply chain attacks and NPM security [Електронний ресурс] // MDN Web Docs. – 2024. – Режим доступу: https://developer.mozilla.org/en- US/docs/Web/Security/Attacks /Supply_chain_attacks.
2. Defending Against Software Supply Chain Attacks [Електронний ресурс] // Cybersecurity and Infrastructure Security Agency (CISA) & NIST. – 2021. – Режим доступу: https://www.cisa. gov/sites/default/files/publications/defending_against_software_supply_chain_attacks_508.pdf.
3. Ensor M. Shifting left on security: Securing software supply chains [Електронний ресурс] / M. Ensor, D. Stevens // Google Cloud Whitepaper. – 2021. – Режим доступу: https://cloud.google. com/files/shifting-left-on-security.pdf.