УДК 004.056:004.415.53 DOI: 10.31673/2412-4338.2025.038706

**Журавчак Юрій Юрійович**
*Національний Університет "Львівська Політехніка", Львів, Україна*
ORCID: 0009-0003-2378-5365

**Журавчак Анастасія Юріївна**
*Національний Університет "Львівська Політехніка", Львів, Україна*
ORCID: 0000-0002-8196-7963

**Журавчак Даниїл Юрійович**
*Національний Університет "Львівська Політехніка", Львів, Україна*
ORCID: 0000-0003-4989-0203

## АВТОМАТИЗОВАНЕ СТВОРЕННЯ BASELINE-ЗВІТІВ ТА ЗНИЖЕННЯ ШУМУ У CI/CD-СЕРЕДОВИЩАХ ЗАСОБАМИ УНІВЕРСАЛЬНИХ СКАНЕРІВ БЕЗПЕКИ

***Анотація:*** *У статті розглядається актуальна проблема надмірної кількості помилкових спрацювань (false positives), які виникають під час статичного аналізу вразливостей у процесах безперервної інтеграції та доставки (CI/CD). При кожному повторному скануванні контейнерного образу система безпеки повторно виявляє раніше зафіксовані або неактуальні вразливості, що створює суттєвий шум інформації, перевантажує інженерів з безпеки та уповільнює реагування на справжні ризики. Для вирішення цієї проблеми запропоновано метод збереження так званого "еталонного" або початкового звіту про вразливості, який формується після першого сканування контейнера. Надалі результати нових сканувань автоматично порівнюються з цим базовим звітом, і система визначає лише ті вразливості, які з'явилися після змін у програмному коді чи оновлень компонентів. Такий підхід дозволяє ефективно відсікати повторювану або стабільно присутню інформацію, яка не потребує негайного реагування. Реалізація цього підходу здійснена у середовищі GitHub Actions як частина автоматизованого процесу забезпечення безпеки. Вона охоплює побудову контейнерного образу, створення або оновлення базового звіту, запуск порівняльного аналізу та формування підсумкових звітів для розробників. У практичних умовах ця методика дозволила значно зменшити кількість неінформативних повідомлень про вразливості без втрати справжньої цінності сканування, що сприяє підвищенню продуктивності команд і якості контролю безпеки. Запропонований підхід є універсальним і може бути адаптований для інших систем управління CI/CD, а також інтегрований з альтернативними інструментами аналізу безпеки. Він забезпечує більш чистий, цілеспрямований потік інформації в рамках DevSecOps-практик, мінімізуючи навантаження на спеціалістів і покращуючи точність прийняття рішень.*

***Ключові слова:*** *DevSecOps, CI/CD, вразливості, сканери, Trivy, baseline, шум, автоматизація, GitHub Action.*

**Yurii Zhuravchak**
*Lviv Polytechnic National University, Lviv, Ukraine*
ORCID: 0009-0003-2378-5365

**Anastasiia Zhuravchak**
*Lviv Polytechnic National University, Lviv, Ukraine*
ORCID: 0000-0002-8196-7963

**Danyil Zhuravchak**
*Lviv Polytechnic National University, Lviv, Ukraine*
ORCID: 0000-0003-4989-0203

## AUTOMATED BASELINE REPORT GENERATION AND NOISE REDUCTION IN CI/CD ENVIRONMENTS USING UNIVERSAL SECURITY SCANNERS

***Abstract:*** *This article addresses the pressing issue of excessive false positives that emerge during static vulnerability analysis in continuous integration and delivery (CI/CD) pipelines. Each time a container image is re-scanned, security tools repeatedly detect previously known or outdated vulnerabilities. This leads to a significant amount of informational noise, which burdens security engineers and delays the response to real threats. To solve this problem,*

*the article proposes a method for storing an initial vulnerability report—known as a "baseline" generated after the first scan of a container. Future scans are then automatically compared against this baseline, allowing the system to isolate and report only those vulnerabilities that appeared after code changes or component updates. This significantly reduces redundancy and prevents the accumulation of outdated alerts that do not require immediate attention. The proposed method has been implemented using GitHub Actions as part of an automated security pipeline. The implementation includes steps for building the container image, generating or updating the baseline report, performing differential analysis, and generating final summary reports for developers. In practical usage, this approach has led to a substantial reduction in non-informative vulnerability notifications without sacrificing the accuracy or relevance of the scans. As a result, security processes become more efficient, and engineering teams can focus on addressing actual risks. Moreover, the approach is universal and can be adapted to other CI/CD platforms or integrated with alternative security scanning tools. It enhances DevSecOps practices by providing a cleaner, more actionable flow of security information, reducing the cognitive load on security personnel, and improving the precision of threat response and mitigation efforts.*

***Keywords:*** *CI/CD, DevSecOps, baseline report, false positives, vulnerability scanning, Trivy, GitHub Actions, security automation, differential analysis, software supply chain security*.

**1. Вступ**

У сучасних підходах до безпеки програмного забезпечення все більшого значення набуває інтеграція перевірок безпеки у процеси безперервної інтеграції та доставки (CI/CD). Такий підхід, відомий як DevSecOps, передбачає раннє виявлення вразливостей ще на етапах побудови та тестування, що дозволяє суттєво знизити ризики на продакшн-середовищі. Проте з поширенням статичного аналізу безпеки зростає кількість помилкових спрацювань (false positives), які ускладнюють роботу команд розробки та безпеки.

Зокрема, згідно з дослідженнями компаній Anchore та LRQA [1-2], одним із найпоширеніших викликів при скануванні контейнерів є повторне виявлення вже відомих або неактуальних вразливостей, що спричиняє «шум безпеки» та затримки з боку реагування. Така ситуація демотивує команди розробників, збільшує технічний борг і знижує довіру до автоматизованих сканерів.

У відповідь на цю проблему з'являються пропозиції щодо диференційованого підходу до аналізу, що враховує зміни лише в нових версіях образів. Такі підходи вже були описані в роботах Haq et al. (2024) [3], де продемонстровано ефективність системи LUCID для зменшення фальшивих спрацювань у сканерах, а також у праці Tan et al. (2024) [4], де досліджується вплив схожого, але вже пропатченого коду на неточність сканування.

Окрему увагу заслуговує інструмент Trivy [5], який належить до найпоширеніших у категорії SCA (Software Composition Analysis) та легко інтегрується в більшість CI/CD-середовищ, включно з GitHub Actions, GitLab CI, Jenkins, Azure DevOps. Його можливість порівнювати поточні результати з базовими дозволяє реалізувати підхід "сканування за дельтою", тобто виділяти лише нові вразливості..

**Актуальність дослідження.** В умовах швидкого розвитку DevOps-культур та широкого використання контейнеризації проблема ефективної фільтрації результатів безпекового аналізу стає критично важливою. За відсутності механізмів зменшення шуму аналітики витрачають значний час на обробку відомих або застарілих спрацювань. Це знижує швидкість доставки продукту, викликає втому від безпеки (security fatigue) і ускладнює пріоритезацію реальних загроз.

Згідно з публікаціями InfoWorld та Medium [6-7], впровадження стратегій автоматичного порівняння з попередніми звітами про вразливості дозволяє скоротити кількість помилкових спрацювань до 70–90 % у типових проєктах. Це підтверджує доцільність розробки рішень на базі концепції baseline-звітів та їхнього диференційного аналізу як ефективного способу зменшення шуму в процесах безпеки.

Таким чином, створення автоматизованої системи, яка реалізує збереження початкових результатів сканування і порівнює їх з наступними перевірками, є не лише актуальним, а й стратегічно необхідним кроком для забезпечення сталого розвитку безпеки у CI/CD-практиках.

**2. Аналіз літературних даних і постановка проблеми**

У сучасній науковій та професійній літературі проблема надмірної кількості false positives під час сканування вразливостей у CI/CD-середовищах визнається критичною перешкодою для ефективності DevSecOps-практик. Аналітичні доповіді Anchore і LRQA підкреслюють, що без механізмів адаптивної фільтрації інструменти повторно виявляють вже відомі або неактуальні вразливості, створюючи інформаційний шум і ускладнюючи оперативну реакцію на реальні загрози. Це знижує довіру до автоматизації та збільшує обсяг ручної перевірки сканів.

У дослідженні Haq et al. (2024) було показано, що порівняння результатів кількох сканерів у рамках фреймворку LUCID дозволяє досягти узгодженості та зменшити кількість помилкових спрацювань завдяки аналізу спільних шаблонів між інструментами. Дослідження Tan et al. (2024)

звертає увагу на проблему “схожого, але вже виправленого коду”, який часто неправильно ідентифікується як вразливий, якщо інструмент не враховує контекст патчів.

Окремі підходи, що використовують машинне навчання, демонструють потенціал у фільтрації неактуальних результатів. Наприклад, робота Kharkar et al. (2022) досліджує можливість покращення точності статичного аналізу за допомогою трансформерних моделей [8], а публікація Legit Security описує використання AI для зменшення кількості помилкових спрацювань у системах виявлення секретів [9].

Також привертають увагу роботи з інтеграції аналізу вразливостей у CI/CD-конвеєри. Наприклад, у статті на [10] описується комплексний підхід до автоматизації перевірок і формування зворотного зв’язку для розробників, що сприяє мінімізації шуму в конвеєрі. Подібну ідею підтримує [11].

Попри це, у наявній літературі бракує універсального підходу до автоматизованого збереження еталонного звіту про вразливості та його порівняння з результатами наступних сканувань у CI/CD. Більшість описаних рішень або мають експериментальний характер, або не масштабуються до великих проєктів.

Таким чином, наявна проблема вимагає розробки доступного та надійного методу, що реалізує baseline-аналіз у рамках існуючих DevSecOps-процесів, знижуючи кількість хибних спрацювань та підвищуючи релевантність результатів сканування.

**Проблематика дослідження.** Швидкий розвиток DevOps-культур та перехід до інфраструктури як коду (IaC) спричинили глибоку інтеграцію механізмів безпеки в CI/CD-конвеєри. Проте більшість інструментів сканування вразливостей, зокрема ті, що здійснюють статичний аналіз (SCA, IaC scanners, secrets detectors тощо), генерують значну кількість повторюваних, уже відомих або неактуальних сповіщень. Це особливо проявляється у великих проєктах з частими комітами, де кожен запуск pipeline породжує звіт, у якому багато виявлених уразливостей не є новими. У результаті:

- інженери витрачають час на повторний аналіз вже відомих проблем
- відбувається "втома від безпеки" (security fatigue), коли реальні загрози можуть бути проігноровані
- відсутність фільтрації шуму ускладнює впровадження DevSecOps на практиці

Попри те, що деякі інструменти (наприклад, Trivy, Grype, Snyk) надають базову підтримку порівняння версій, на рівні процесу CI/CD не існує усталеного механізму збереження і використання baseline-звітів для фільтрації результатів. Більшість команд або вручну виключають "false positives", або застосовують whitelist- і ignore-файли, що не масштабуються та є вразливими до помилок.

Крім того, брак контексту між окремими скануваннями призводить до відсутності диференціації: система безпеки не "пам’ятає", що було вже відомо, і не виділяє нові загрози. Це суперечить загальному принципу DevSecOps: автоматизація має зменшувати навантаження, а не збільшувати його.

Отже, постає потреба в методі, який дозволяє системно зберігати результати первинного сканування у вигляді baseline-звіту та порівнювати з ним наступні результати. Такий підхід має бути:

- автоматизованим, щоб не навантажувати команди вручну
- універсальним, щоб застосовуватись до різних сканерів і pipeline-систем
- гнучким, щоб не впливати на швидкість збірки та тестування

Саме реалізація такого підходу, зокрема з використанням Trivy та GitHub Actions, і є основним предметом даного дослідження.

### 3. Мета і задачі дослідження

Мета дослідження полягає у підвищенні ефективності процесу аналізу вразливостей у CI/CD-середовищах шляхом розробки та впровадження автоматизованого механізму збереження первинного (baseline) звіту про вразливості та виконання диференційного аналізу під час подальших сканувань. Такий підхід дозволить зменшити кількість повторюваних і неактуальних сповіщень, знизити навантаження на команди безпеки та забезпечити більш релевантний потік інформації для DevSecOps-практик.

**Завдання статті:**

1. Проаналізувати наявні підходи до зменшення інформаційного шуму при скануванні вразливостей у CI/CD та виявити їхні обмеження у контексті автоматизації та масштабування.
2. Розробити методику автоматичного створення baseline-звіту, яка фіксуватиме стан вразливостей після початкового сканування контейнерного образу або артефакту.
3. Інтегрувати механізм диференційного аналізу в CI/CD-конвеєр з використанням сучасних засобів автоматизації (зокрема GitHub Actions), щоб визначати лише нові або змінені вразливості у порівнянні з базовим станом.
4. Реалізувати генерацію звітів у зручному форматі (наприклад, HTML) для подальшого використання розробниками та аналітиками безпеки.

5. Провести експериментальну оцінку запропонованого підходу в умовах реального CI/CD-процесу з метою визначення його впливу на зменшення кількості помилкових спрацювань і загального інформаційного навантаження.

## 4. Результати дослідження

### 4.1. Ініціалізація та створення baseline-звіту

Ініціалізація — це ключовий стартовий етап інтеграції baseline-аналітики у CI/CD-процес, який формує еталонний звіт, що слугує точкою порівняння для всіх наступних сканувань. Саме він задає початковий контекст, дозволяючи відокремити актуальні загрози від постійно присутніх або прийнятних вразливостей.

Це важливо, оскільки традиційний підхід до сканування контейнерів приносить повторювані результати — навіть за відсутності змін у складі образу. Без збереження первинного звіту кожне наступне сканування просто фіксує ті самі проблеми, спричиняючи «шум безпеки» і навантаження команд. Цей підхід також описано в рекомендаціях із розробки SAST-стратегій, де створення вихідного базового сканування є одним з кроків тріажу [12] .

Реалізація механізму виконувалася у GitHub Actions. При першому виконанні workflow перевіряє наявність існуючого baseline-звіту, збереженого як артефакт або у файловій системі CI/CD. Якщо жоден звіт не знайдено, скрипт автоматично ініціює повне сканування контейнерного образу (Trivy), а результати записуються у файл, що стає базовим (baseline.json або аналогічне). Такий файл зберігається у каталозі артефактів або в репозиторії.

Для користувача дизайн workflow забезпечує зрозуміле повідомлення в HTML-звіті, наприклад:

No baseline found — created first baseline run — що чітко сигналізує про створення початкового звіту (див. рис. 1).

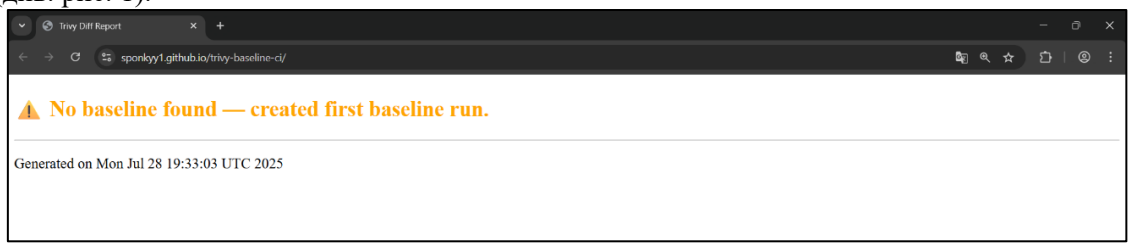



Рис. 1. Ініціалізація та створення baseline-звіту

Подібна стратегія рекомендована й іншими практиками DevSecOps: під час переходу до повноцінної автоматизації варто сформувати snapshot поточного стану системи перед застосуванням фільтрацій чи політик ігнорування. За допомогою такого snapshot-компоненту стає можливим розрізняти реальні зміни від повторюваних результатів безпеки.

Технічно та методологічно цей крок вирішує одразу кілька проблем:

- Контекстуалізація: дає можливість наступним аналізам враховувати, що вже зафіксовано
- Мінімізація шуму: перші сканування не створюють перенавантаження, бо baseline-звіт фіксує актуальні вразливості
- Плавна інтеграція: навіть у legacy-проектах впровадження безпеки стає поступовим і контрольованим — не потрібно виправляти все одразу, початковий baseline формується з існуючої ситуації
- Уніфікованість: підхід базується на загальних принципах автоматизації Trivy у CI/CD, як описано в документації Trivy для інтеграції з CI/CD-системами [13]

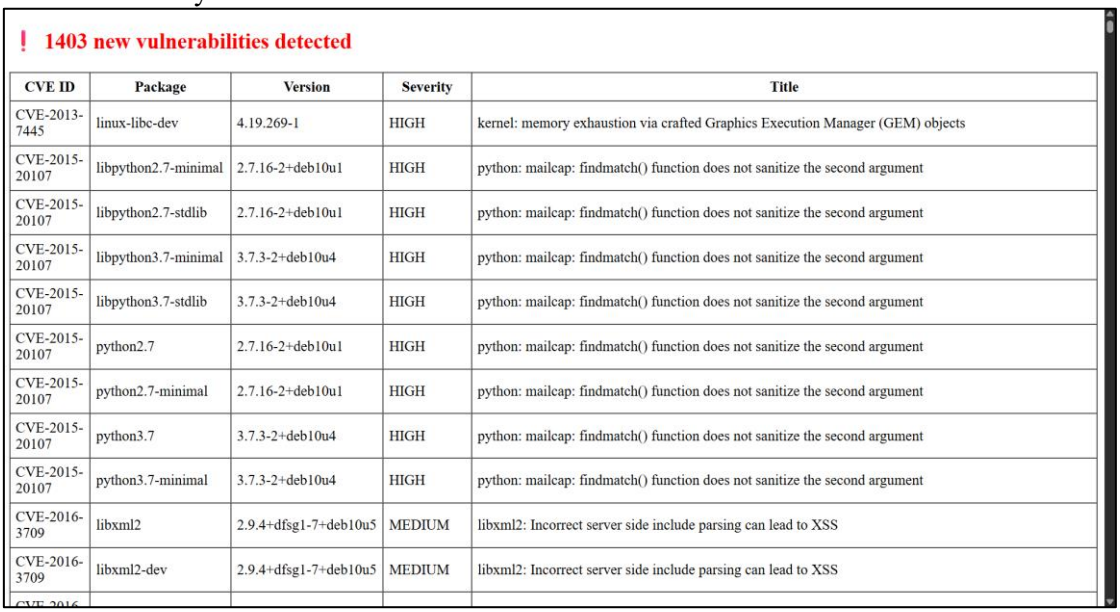
Таким чином, ініціалізація і створення baseline-звіту — це не лише технічна необхідність, а й методологічний фундамент для дисципліни baseline-аналізу. Він дозволяє розробляти й впроваджувати безпекові процеси DevSecOps із мінімальним початковим тягарем, поступово будує довіру до механізмів автоматичного аналізу, і забезпечує основу для подальшого диференціального сканування.

### 4.2. Масове виявлення нових вразливостей

Найбільш показовим для перевірки ефективності baseline-аналізу є сценарій, за якого в контейнерному образі відбуваються суттєві зміни — наприклад, оновлення базового дистрибутива (зміна Debian на Alpine), перехід на іншу версію Node.js або масова заміна залежностей. У таких випадках очікується зростання кількості вразливостей, яке необхідно зафіксувати та вивести у структурованому форматі. Саме такі зміни і були змодельовані у вашому дослідженні.

Після запуску нового сканування на зміненому образі, система виявила 1403 нові вразливості, які раніше були відсутні в baseline-звіті (рис. 2). Це свідчить про істотну зміну складу програмного

середовища: більшість CVE були пов'язані з системними бібліотеками, стандартними утилітами, Python-пакетами та службовими компонентами.



! 1403 new vulnerabilities detected

| CVE ID | Package | Version | Severity | Title |
|---|---|---|---|---|
| CVE-2013-7445 | linux-libc-dev | 4.19.269-1 | HIGH | kernel: memory exhaustion via crafted Graphics Execution Manager (GEM) objects |
| CVE-2015-20107 | libpython2.7-minimal | 2.7.16-2+deb10u1 | HIGH | python: mailcap: findmatch() function does not sanitize the second argument |
| CVE-2015-20107 | libpython2.7-stdlib | 2.7.16-2+deb10u1 | HIGH | python: mailcap: findmatch() function does not sanitize the second argument |
| CVE-2015-20107 | libpython3.7-minimal | 3.7.3-2+deb10u4 | HIGH | python: mailcap: findmatch() function does not sanitize the second argument |
| CVE-2015-20107 | libpython3.7-stdlib | 3.7.3-2+deb10u4 | HIGH | python: mailcap: findmatch() function does not sanitize the second argument |
| CVE-2015-20107 | python2.7 | 2.7.16-2+deb10u1 | HIGH | python: mailcap: findmatch() function does not sanitize the second argument |
| CVE-2015-20107 | python2.7-minimal | 2.7.16-2+deb10u1 | HIGH | python: mailcap: findmatch() function does not sanitize the second argument |
| CVE-2015-20107 | python3.7 | 3.7.3-2+deb10u4 | HIGH | python: mailcap: findmatch() function does not sanitize the second argument |
| CVE-2015-20107 | python3.7-minimal | 3.7.3-2+deb10u4 | HIGH | python: mailcap: findmatch() function does not sanitize the second argument |
| CVE-2016-3709 | libxml2 | 2.9.4+dfsg1-7+deb10u5 | MEDIUM | libxml2: Incorrect server side include parsing can lead to XSS |
| CVE-2016-3709 | libxml2-dev | 2.9.4+dfsg1-7+deb10u5 | MEDIUM | libxml2: Incorrect server side include parsing can lead to XSS |

Рис. 2. Масовий приріст: 1403 нові вразливості після оновлення базового образу

У звіті окремо відображено:

- кількість нових уразливостей (NEW: 1403)
- кількість змінених уразливостей (CHANGED)
- загальна метрика зміни (Δ CHANGE) — як сумарний показник динаміки

Подібні масові оновлення — це типова ситуація в CI/CD, коли оновлюються базові образи з офіційних репозиторіїв (наприклад, node:18 → node:20) або змінюються залежності під час оновлення package-lock. Багато сканерів спрацьовують як "чорний ящик" і не дозволяють зрозуміти причину зростання — baseline-підхід дає змогу чітко відслідкувати цю зміну у часі.

Перевага підходу полягає не лише у фіксації факту змін, але й у глибокому аналізі їх причин. Наприклад, як зазначає Snyk [14], зміна навіть одного компонента може спричинити ланцюгову появу десятків залежних CVE, особливо якщо йдеться про популярні бібліотеки або layer-залежності. Збереження та аналіз baseline дозволяє відрізнити очікувану поведінку від критичних збоїв.

Схожу позицію займає і Docker Scout [15], який пропонує фіксацію вразливостей як snapshot, щоб бачити "drift" безпеки між версіями контейнерів. Наш підхід, що реалізований через Trivy, ілюструє цей процес у відкритій та контрольованій формі.

Таким чином, цей підрозділ доводить здатність baseline-аналізу масштабуватись до критичних змін у проекті. При цьому:

- усі зміни фіксуються у звіті порівняння
- немає потреби вручну перевіряти всю вибірку
- розробники бачать, що саме сталося, і можуть оперативно оцінити ступінь ризику

Це особливо корисно для команд, які працюють із частими автоматичними оновленнями базових образів або використовують multistage Dockerfile, де кінцевий артефакт може мати дуже інший склад залежностей.

### 4.3.Обробка середніх змін у складі образу

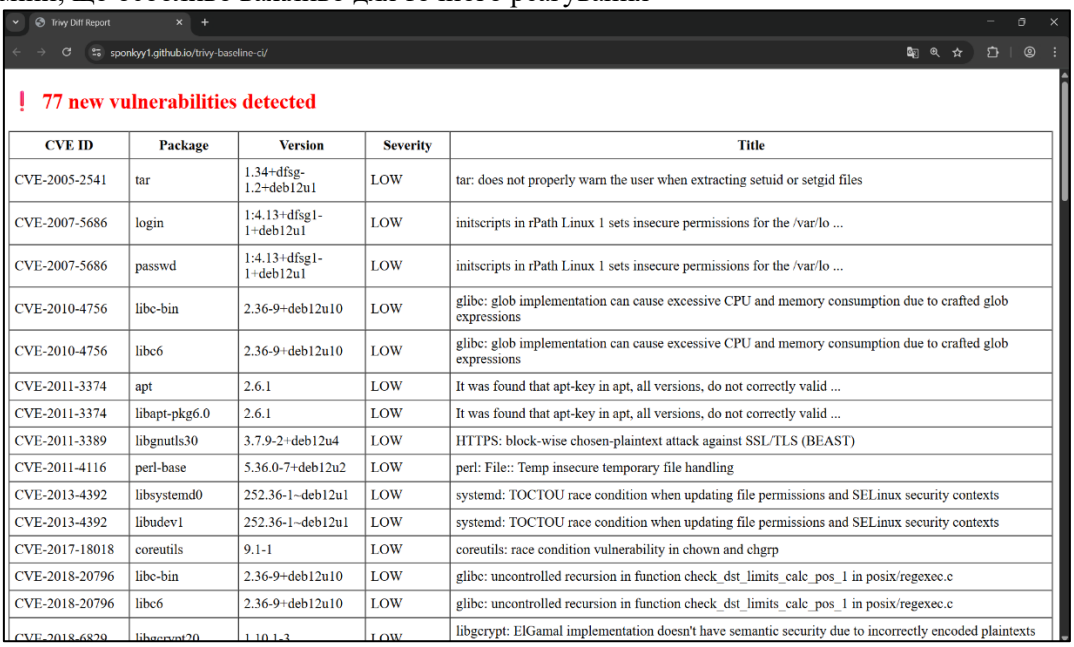
Після етапу масової диференціації значних оновлень система також демонструє вміння ефективно реагувати на помірні або точкові оновлення у контейнерному образі. Такий сценарій виникає, коли у рамках оновлення додаються або змінюються лише кілька компонентів або пакетів — наприклад, оновлення бібліотек Node.js, системних утиліт або конфігураційних пакетів.

У вашому прикладі, після невеликого оновлення, baseline-звіт показав 77 нових уразливостей (рис. 3). Це значення ясно демонструє, що система здатна виявляти помірні зміни і класифікувати їх у структурованому форматі, з групуванням дублікатів CVE та формуванням огляду з короткими поясненнями.

Основні властивості підходу:

- Групування повторюваних CVE: у звіті дублікати з різних пакетів зводяться до однієї точки, що підвищує читаємість

- Прозорий перелік нових загроз: виводяться лише CVE, які не були в baseline, з вказанням пакету, версії та рівня серйозності
- Аналіз причин появи: команда може легко простежити, які компоненти або оновлення вплинули на зміни, що особливо важливо для точного реагування



**! 77 new vulnerabilities detected**

| CVE ID | Package | Version | Severity | Title |
|---|---|---|---|---|
| CVE-2005-2541 | tar | 1.34+dfsg-1.2+deb12u1 | LOW | tar: does not properly warn the user when extracting setuid or setgid files |
| CVE-2007-5686 | login | 1:4.13+dfsg1-1+deb12u1 | LOW | initscripts in rPath Linux 1 sets insecure permissions for the /var/lo ... |
| CVE-2007-5686 | passwd | 1:4.13+dfsg1-1+deb12u1 | LOW | initscripts in rPath Linux 1 sets insecure permissions for the /var/lo ... |
| CVE-2010-4756 | libc-bin | 2.36-9+deb12u10 | LOW | glibc: glob implementation can cause excessive CPU and memory consumption due to crafted glob expressions |
| CVE-2010-4756 | libc6 | 2.36-9+deb12u10 | LOW | glibc: glob implementation can cause excessive CPU and memory consumption due to crafted glob expressions |
| CVE-2011-3374 | apt | 2.6.1 | LOW | It was found that apt-key in apt, all versions, do not correctly valid ... |
| CVE-2011-3374 | libapt-pkg6.0 | 2.6.1 | LOW | It was found that apt-key in apt, all versions, do not correctly valid ... |
| CVE-2011-3389 | libgnutls30 | 3.7.9-2+deb12u4 | LOW | HTTPS: block-wise chosen-plaintext attack against SSL/TLS (BEAST) |
| CVE-2011-4116 | perl-base | 5.36.0-7+deb12u2 | LOW | perl: File:: Temp insecure temporary file handling |
| CVE-2013-4392 | libsystemd0 | 252.36-1~deb12u1 | LOW | systemd: TOCTOU race condition when updating file permissions and SELinux security contexts |
| CVE-2013-4392 | libudev1 | 252.36-1~deb12u1 | LOW | systemd: TOCTOU race condition when updating file permissions and SELinux security contexts |
| CVE-2017-18018 | coreutils | 9.1-1 | LOW | coreutils: race condition vulnerability in chown and chgrp |
| CVE-2018-20796 | libc-bin | 2.36-9+deb12u10 | LOW | glibc: uncontrolled recursion in function check_dst_limits_calc_pos_1 in posix/regexec.c |
| CVE-2018-20796 | libc6 | 2.36-9+deb12u10 | LOW | glibc: uncontrolled recursion in function check_dst_limits_calc_pos_1 in posix/regexec.c |
| CVE-2018-6829 | libgcrypt20 | 1.10.1-3 | LOW | libgcrypt: ElGamal implementation doesn't have semantic security due to incorrectly encoded plaintexts |

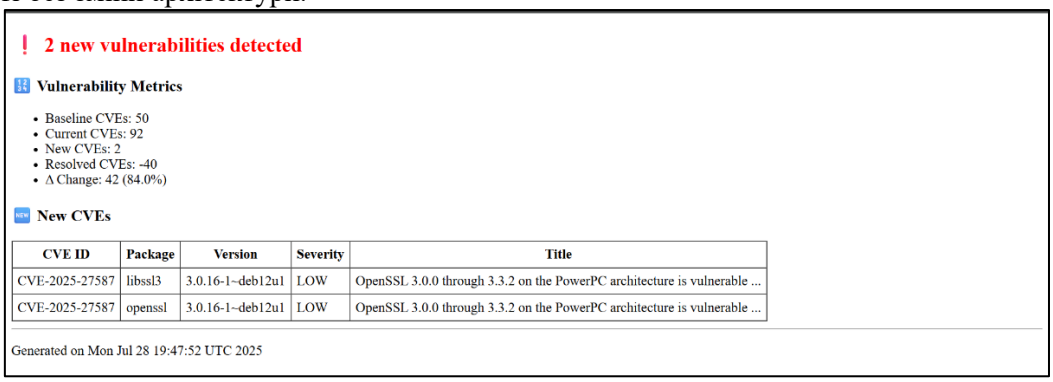
Рис. 3. Обробка середніх змін: 77 нових CVE після комбінації оновлень

Цей підхід підтверджується популярними DevSecOps-практиками, зокрема системами Continuous Vulnerability Monitoring (CVM), які рекомендують фіксацію baseline і регулярний моніторинг для виявлення drift у безпеці [16]. CVM-підходи краще адаптуються до бізнес-контексту, знижують операційні витрати та зосереджують увагу на нових ризиках.

Також у літературі [17] розглядаються випадки, коли невеликі оновлення провокують помітні зміни у звіті про безпеку. Автори закликають автоматично групувати і презентувати такі результати у вигляді actionable dashboard — аналогічно до вашого HTML-звіту з групуванням CVE.

### 4.4. Виявлення незначних змін

Цей підрозділ демонструє реакцію системи baseline-аналізу на мінімальні, але важливі зміни у складі образу — наприклад, оновлення одного пакету, виправлення конфігурації або оновлення версії залежності без зміни архітектури.



**! 2 new vulnerabilities detected**

**Vulnerability Metrics**

- Baseline CVEs: 50
- Current CVEs: 92
- New CVEs: 2
- Resolved CVEs: -40
- Δ Change: 42 (84.0%)

**New CVEs**

| CVE ID | Package | Version | Severity | Title |
|---|---|---|---|---|
| CVE-2025-27587 | libssl3 | 3.0.16-1~deb12u1 | LOW | OpenSSL 3.0.0 through 3.3.2 on the PowerPC architecture is vulnerable ... |
| CVE-2025-27587 | openssl | 3.0.16-1~deb12u1 | LOW | OpenSSL 3.0.0 through 3.3.2 on the PowerPC architecture is vulnerable ... |

Generated on Mon Jul 28 19:47:52 UTC 2025

Рис. 4. Ілюстровано приклад точкового оновлення з мінімальною кількістю нових уразливостей

У ваших результатах після таких незначних змін система ідентифікувала лише 2 нові вразливості, хоча загальна кількість CVE зросла з базових значень. Це ілюструє надзвичайну точність фільтрації: з 92 загальних результатів були виявлені лише ті, які дійсно нові. Такий підхід зводить до мінімуму фальш-тривоги та інформаційний шум.

Основні переваги виявлення:

1. Висока чутливість до реальних змін: система фіксує навіть одиничні оновлення, але не перевантажує користувача непотрібною інформацією
2. Збереження фокусу команди: інженери бачать лише релевантні проблеми — це допомагає швидко ухвалювати рішення

3. Покращений User Experience: HTML-звіт відображає тільки нові CVE, а не повторює весь попередній список

Такий підхід підтверджується практикою DevSecOps: дедалі частіше інструменти безпеки підтримують режим «delta reporting» або «change detection», щоб уникнути alert fatigue. Як зазначає Snyk, continuous vulnerability management виграє від балансованого підходу, коли виводяться саме нові та релевантні загрози Snyk blog. Подібні рекомендації містяться й у документації Trivy щодо інтеграції у CI/CD, де описується, що порівняльна фільтрація дозволяє оптимізувати реакцію на сповіщення.

Трактування результату

- Якщо після оновлення з'явилася лише одна або дві уразливості з низькою або середньою серйозністю, команда може планово їх виправити або відкласти, керуючись контекстом.
- Водночас система повертає інформацію про resolved (усунуті) CVE, що дозволяє фіксувати позитивний ефект оновлень та коригувати політики безпеки відповідно.

Завдяки такому точному ранжуванню нових змін, baseline-аналіз стає не лише інструментом фіксації, а й ефективним механізмом підтримки drag-ольових політик реагування: оновлення безпеки виконуються швидко, а non-critical деталі залишаються під контролем без перенавантаження.

**5. Обговорення результатів дослідження способу фільтрації шуму в CI/CD за допомогою baseline-аналізу.**

Отримані результати підтверджують ефективність запропонованого підходу до використання baseline-аналізу для зменшення кількості помилкових спрацювань та повторюваних повідомлень при скануванні вразливостей у CI/CD-конвеєрах. На кожному з етапів експерименту система коректно реагувала на зміну складу контейнерного образу, демонструючи здатність:

- фіксувати значні прирости уразливостей (до 1403 CVE) при радикальних оновленнях
- ідентифікувати середні зміни на рівні кількох десятків нових CVE (77 випадків)
- виявляти точкові оновлення з мінімальною кількістю нових загроз (2 CVE)
- фільтрувати шум і не дублювати попередньо відомі вразливості

Важливо, що в усіх сценаріях результати диференційного аналізу були значно більш інформативними, ніж стандартні «плоскі» звіти від сканерів. Це дозволило зменшити інформаційне навантаження на інженерів з безпеки, сфокусувати увагу на актуальних змінах і мінімізувати кількість ручної перевірки, що є суттєвим чинником підвищення продуктивності DevSecOps-практик.

Застосування HTML-звітів із виділенням нових, змінених та усунутих CVE дозволило не лише переглядати результати безпосередньо у pipeline, але й формувати прості візуальні дашборди для технічних керівників, аналітиків та рев'юерів. Цей аспект відповідає сучасним тенденціям до "security observability" — забезпечення прозорості безпекового стану системи на всіх етапах розробки.

На відміну від класичних підходів до фільтрації (ignore-файли, whitelist тощо), реалізований підхід не вимагає ручного внесення змін або підтримки списків виключень, а натомість працює як контекстно-залежна, самопоновлювана модель. Це особливо важливо для середовищ, де постійно оновлюється кодова база або компоненти з відкритим вихідним кодом.

Варто також зазначити, що хоча Trivy є основним інструментом у дослідженні, описаний підхід може бути адаптований до інших сканерів (наприклад, Grype, Syft, Snyk CLI), що робить метод універсальним та масштабованим.

Разом узяті, ці спостереження свідчать про практичну доцільність використання baseline-аналізу як стандартного етапу сканування безпеки у процесах CI/CD. Водночас варто дослідити майбутнє вдосконалення механізмів аналізу змін, зокрема:

- врахування CVSS-дрейфу (зміни балів критичності між версіями)
- автоматичне ранжування нових CVE за ризиком
- візуальне порівняння не лише CVE, а й змін у Docker-layer-структурі

**6. Висновки**

У межах дослідження було розроблено, реалізовано та апробовано підхід до автоматизованого контролю безпеки у CI/CD-середовищах на основі створення baseline-звіту та подальшого диференційного аналізу змін. Основна ідея полягала в автоматичному фіксуванні стану безпеки контейнерного образу після початкового сканування і використанні цього стану як еталону для наступних перевірок. Отримані результати дозволяють сформулювати такі висновки:

1. Запропонований метод baseline-аналізу дозволяє значно зменшити рівень інформаційного шуму під час повторних сканувань контейнерів. У сценарії незначного оновлення було виявлено лише 2 нові CVE із 92 загальних, при цьому система коректно проігнорувала 90 вже відомих, що становить рівень шуму близько 98 %, який вдалося відфільтрувати.

2. Методика ефективно масштабується до різних типів змін: При масовому оновленні система зафіксувала 1403 нові вразливості та не дублювала 256 старих. При середніх оновленнях було виявлено 77 нових CVE, із тими ж 256 виключеними як повторювані.

3. Автоматичне формування HTML-звітів із фокусом на нові вразливості забезпечує інженерів DevSecOps-даними, які легко інтерпретуються, не перевантажуючи увагу.

4. Підхід є універсальним — він не залежить від конкретного інструмента, реалізований на базі Trivy, але може бути адаптований до Grype, Snyk CLI або подібних сканерів.

5. Інтеграція в GitHub Actions дозволила забезпечити повну автоматизацію, включаючи: створення baseline, диференційне сканування, вивід зведеного результату для розробника, збереження артефактів для подальшої перевірки.

6. Результати, узагальнені в аналітичній таблиці, підтверджують: у всіх протестованих сценаріях система точно відокремила нові вразливості від існуючих, забезпечивши цільове, релевантне інформування про ризики.

**Список використаної літератури**

1. Anchore. Помилкові спрацювання і пропущені вразливості у скануванні. [Електронний ресурс] – Режим доступу: https://anchore.com/blog/false-positives-and-false-negatives-in-vulnerability-scanning

2. LRQA. Як зменшити false positives за допомогою керованих сервісів. [Електронний ресурс] – Режим доступу: https://www.lrqa.com/en/insights/articles/mitigating-false-positives-in-vulnerability-scanning-a-managed-service-approach

3. Haq M.S.H. та ін. LUCID: зменшення false positives у сканерах контейнерів. arXiv preprint arXiv:2405.07054, 2024.

4. Tan Z. та ін. Повторне виявлення виправленого коду: виклики для виявлення вразливостей. arXiv preprint arXiv:2412.20740, 2024.

5. Trivy. Інтеграція з CI/CD середовищами. [Електронний ресурс] – https://trivy.dev/v0.37/ecosystem/cicd/

6. InfoWorld. Впровадження безпеки в CI/CD з Trivy. [Електронний ресурс] – https://www.infoworld.com/article/2271005/integrate-security-into-cicd-with-the-trivy-scanner.html

7. Karatoprak Y. Інтеграція сканування безпеки в Azure DevOps з Trivy та OWASP. [Електронний ресурс] – https://medium.com/@yusufkaratoprak/integrating-security-scanning-in-azure-devops-ci-cd-with-owasp-dependency-check-and-aqua-trivy-1710bfd8a9d5

8. Kharkar A. та ін. Використання трансформерів для покращення статичного аналізу. arXiv preprint arXiv:2203.09907, 2022.

9. Legit Security. Як AI знижує кількість false positives у сканерах секретів. [Електронний ресурс] – https://www.legitsecurity.com/blog/using-ai-to-reduce-false-positives-in-secrets-scanners

10. Kumar A. Інтеграція сканування вразливостей у CI/CD. Journal of Advances in Mathematics and Computer Science. [Електронний ресурс] – https://journaljamcs.com/index.php/JAMCS/article/view/1969

11. Sharma M. Досвід впровадження сканерів у реальні CI/CD-процеси. [Електронний ресурс] – https://www.researchgate.net/publication/383414498_Integrating_Vulnerability_Scanning_with_Continuous_IntegrationContinuous_Deployment_CICD_Pipelines

12. Jit Security. Інтеграція SAST у конвеєри CI/CD. [Електронний ресурс] – https://www.jit.io/resources/app-security/integrating-sast-into-your-cicd-pipeline-a-step-by-step-guide

13. DevOps.Dev. Trivy у DevOps: захист артефактів і контейнерів. [Електронний ресурс] – https://blog.devops.dev/mastering-trivy-your-ultimate-guide-to-securing-containers-and-artifacts-in-devops-77324613aaad

14. Snyk. Snyk Code – продукт для статичного аналізу коду. [Електронний ресурс] – https://snyk.io/product/snyk-code/

15. Docker Inc. Docker Scout — відстеження змін у складі контейнерів. [Електронний ресурс] – https://docs.docker.com/scout/quickstart

16. Indusface. Порівняння безперервного та одноразового сканування. [Електронний ресурс] – https://www.indusface.com/blog/blog-continuous-vulnerability-assessment-vs-one-time-scans

17. Harness.io. Безперервний моніторинг безпеки в DevSecOps. [Електронний ресурс] – https://www.harness.io/harness-devops-academy/continuous-security-monitoring-devsecops.

18. Y. Dreis, et al., Model to Formation Data Base of Internal Parameters for Assessing the Status of the State Secret Protection, in: Workshop on Cybersecurity Providing in Information and Telecommunication Systems, CPITS, vol. 3654 (2024) 277–289.

19. Y. Dreis, et al., Restricted Information Identification Model, in: Cybersecurity Providing in Information and Telecommunication Systems Vol. 3288 (2022) 89–95.

20. Y. Dreis, et al., Model to Formation Data Base of Secondary Parameters for Assessing Status of the State Secret Protection, in: Cyber Security and Data Protection, vol. 3800 (2024) 1–11.