**УДК 004.491.42**
**DOI https://doi.org/10.32782/IT/2024-3-16**

***Євгеній СЄРГЄЄВ***
*аспірант кафедри комп'ютерної інженерії та інформаційних систем, Хмельницький національний університет, вул. Інститутська, 11, Хмельницький, 29000*
***ORCID:** 0009-0008-9877-9863*
***Scopus Author ID:** 59129893600*

***Антоніна КАШТАЛЬЯН***
*кандидат технічних наук, докторанка, доцент, доцент кафедри фізики та електротехніки, Хмельницький національний університет, вул. Інститутська, 11, Хмельницький, Україна, 29000*
***ORCID:** 0000-0002-4925-9713*
***Scopus Author ID:** 57218242499*

***Василь КОВАЛЬЧУК***
*аспірант кафедри комп'ютерної інженерії та інформаційних систем, Хмельницький національний університет, вул. Інститутська, 11, Хмельницький, Україна, 29000*
***ORCID:** 0009-0008-7013-5919*

***Олег САВЕНКО***
*доктор технічних наук, професор, професор кафедри комп'ютерної інженерії та інформаційних систем, Хмельницький національний університет, вул. Інститутська, 11, Хмельницький, Україна, 29000*
***ORCID:** 0000-0002-4104-745X*
***Scopus Author ID:** 54421023400*

***Олег ІВАНЧЕНКО***
*доктор технічних наук, доцент, професор кафедри програмного забезпечення комп'ютерних систем, Національний технічний університет «Дніпровська політехніка», просп. Дмитра Яворницького 19, Дніпро, Україна, 49005*
***ORCID:** 0000-0002-5921-5757*
***Scopus Author ID:** 57190132131*

**Бібліографічний опис статті:** Сєргєєв, Є., Каштальян, А., Ковальчук, В., Савенко, О., Іванченко, О. (2024). Ефективність і вдосконалення SAST у контексті SQL Injection вразливостей. *Information Technology: Computer Science, Software Engineering and Cyber Security*, 3, 149–158, doi: https://doi.org/10.32782/IT/2024-3-16

# ЕФЕКТИВНІСТЬ І ВДОСКОНАЛЕННЯ SAST У КОНТЕКСТІ SQL INJECTION ВРАЗЛИВОСТЕЙ

*Виявлення вразливостей безпеки на ранніх етапах розробки є критично важливим для забезпечення надійності програмного забезпечення. Статичний аналіз безпеки (SAST) широко використовується для виявлення потенційних вразливостей у коді. Однак складність сучасних методів та використання динамічних конструкцій у коді створюють виклики для SAST-інструментів, особливо у виявленні вразливостей типу SQL Injection, які можуть призвести до несанкціонованого доступу до даних.*

***Метою статті** є дослідити ефективність методу статичного аналізу безпеки (SAST) у виявленні вразливостей типу SQL Injection та на основі експериментального аналізу запропонувати удосконалення цього методу для підвищення його ефективності.*

***Методологія** полягає у проведенні експериментального аналізу існуючих SAST-інструментів на здатність виявляти SQL Injection вразливості. Було використано набір тестових методів з відомими вразливостями для оцінки ефективності. На основі отриманих результатів ідентифіковано основні проблеми та розроблено удосконалення до методу статичного аналізу, які були впроваджені і протестовані для оцінки їхньої ефективності. Застосовано наукові методи синтезу, аналізу та порівняння.*

***Наукова новизна** полягає у розробці та впровадженні удосконалень до методу статичного аналізу безпеки, які підвищують ефективність виявлення вразливостей типу SQL Injection. Запропоновано нові*

*алгоритми аналізу динамічних конструкцій у коді та обробки складних шаблонів запитів до баз даних, що раніше були недоступні для стандартних SAST-інструментів.*

***Висновки**. Запропоновані удосконалення до методу статичного аналізу безпеки дозволяють значно покращити виявлення вразливостей типу SQL Injection, що підтверджується результатами експериментального аналізу. Це підкреслює важливість розвитку і впровадження передових технік у SAST-інструменти для забезпечення високого рівня безпеки програмного забезпечення.*

***Ключові слова:** вразливості, SAST-інструменти, SQL ін'єкції.*

***Yevhenii SIERHIEIEV***
*PhD student at the Department at Computer Engineering and Information Systems, Khmelnytskyi National University, 11, Instytutska Str., Khmelnytskyi, Ukraine, 29000, ysierhieiev@gmail.com*
***ORCID:** 0009-0008-9877-9863*
***Scopus Author ID:** 59129893600*

***Antonina KASHTALIAN***
*PhD, Associate Professor at the Department of Physics and Electrical Engineering, Doctoral Staff, Khmelnytskyi National University, 11, Instytutska Str., Khmelnytskyi, Ukraine, 29000, yantonina@ukr.net*
***ORCID:** 0000-0002-4925-9713*
***Scopus Author ID:** 57218242499*

***Vasily KOVALCHUK***
*PhD Student at the Department of Computer Engineering and Information Systems, Khmelnitsky National University, 11, Instytutska Str., Khmelnytskyi, Ukraine, 29000, vasuli458@gmail.com*
***ORCID:** 0009-0008-7013-5919*

***Oleg SAVENKO***
*Doctor of Technical Sciences, Professor, Professor at the Department of Computer Engineering and Information Systems, Khmelnytskyi National University, 11, Instytutska Str., Khmelnytskyi, Ukraine, 29000, savenko_oleg_st@ukr.net*
***ORCID:** 0000-0002-4104-745X*
***Scopus Author ID:** 54421023400*

***Oleg IVANCHENKO***
*Doctor of Technical Sciences, Associate Professor, Professor at the Department of Computer Systems and Software, Dnipro University of Technology, 19, Dmytra Yavornytskoho Ave., Dnipro, Ukraine, 49005, vmsu12@gmail.com;*
***ORCID:** 0000-0002-5921-5757*
***Scopus Author ID:** 57190132131*

**To site this article:** Sierhieiev, Ye., Kashtalian, A., Kovalchuk, V., Savenko, O., Ivanchenko, O. (2024). Effektyvnist' i vdoskonalennia SAST v konteksti vrazlyvostei SQL Injection [Effectiveness and improvement of SAST in the context of SQL Injection vulnerabilities]. *Information Technology: Computer Science, Software Engineering and Cyber Security,* 3, 149–158, doi: https://doi.org/10.32782/IT/2024-3-16

## EFFECTIVNESS AND IMPROVEMENT OF SAST IN THE CONTEXT OF SQL INJECTION VULNERABILITIES

*Identifying security vulnerabilities early in development is critical to ensuring software reliability. Static Security Analysis (SAST) is widely used to identify potential vulnerabilities in code. However, the complexity of modern applications and the use of dynamic constructs in the code create challenges for SAST tools, especially in detecting SQL Injection vulnerabilities that can lead to unauthorized access to data.*

***The article aims** to investigate the effectiveness of the static security analysis method (SAST) in detecting vulnerabilities of the SQL Injection type and, based on experimental analysis, to propose improvements to this method to increase its effectiveness.*

***The methodology** consists of conducting an experimental analysis of existing SAST tools for the ability to detect SQL Injection vulnerabilities. A set of test applications with known vulnerabilities was used to evaluate performance.*

*Based on the obtained results, the main problems were identified and improvements to the static analysis method were developed, which were implemented and tested to evaluate their effectiveness. Scientific methods of synthesis, analysis, and comparison are applied.*

***The scientific novelty** consists in the development and implementation of improvements to the method of static security analysis, which increases the effectiveness of detecting vulnerabilities of the SQL Injection type. New algorithms for the analysis of dynamic structures in the code and processing of complex query patterns to databases, which were previously unavailable for standard SAST tools, are proposed.*

***Conclusions.** The proposed improvements to the method of static security analysis allow to significantly improve the detection of vulnerabilities of the SQL Injection type, which is confirmed by the results of experimental analysis. This emphasizes the importance of developing and implementing advanced techniques in SAST tools to ensure a high level of software security.*

***Key words:** vulnerabilities, SAST tools, SQL injections.*

**Актуальність проблеми.** Сучасне програмне забезпечення для виявлення вразливостей класифікується за групами відповідно до реалізованих у ньому конкретних методів пошуку вразливостей. Проте, важливим є не лише реалізація одного чи групи методів в спеціалізованих інструментах, а й розробка нових методів, які дозволять ефективніше знаходити та усувати вразливості. Тому, пропонується приділити значну увагу дослідженню існуючих аналітичних підходів та технологій, що використовуються для виявлення вразливостей, дослідити ефективність методу статичного аналізу безпеки (SAST) у виявленні вразливостей типу SQL Injection та на основі експериментального аналізу запропонувати удосконалення цього методу для підвищення його ефективності.

**Аналіз останніх досліджень і публікацій.** Дослідженню виявлення вразливостей в комп'ютерних системах для виявлення зловмисного програмного забезпечення (ЗПЗ) і комп'ютерних атак приділяють увагу багато дослідників. Науковці розробили багато різних методів виявлення вразливостей. Відомо, що невиявлені вразливості можуть призвести до значних витрат для IT компаній. Дослідження Myriam Dunn Cavelty показують, що швидке виправлення є важливим для уникнення втрат, пов'язаних з вразливістю та її публічним розголошенням (Cavelty, 2024). Вказано на значні витрати, пов'язані з недостатньою інфраструктурою тестування програмного забезпечення; тому необхідний інструмент для виявлення вразливостей.

Виявлення вразливостей SQL-ін'єкцій стало популярною темою досліджень у галузі інформаційної безпеки. Досліджено, що метод статичного аналізу безпеки (SAST) базується на детальному перегляді вихідного коду, бінарних файлів та бібліотек без їх виконання для виявлення потенційних вразливостей (Luo, 2022). Цей метод дозволяє ідентифікувати такі проблеми, як SQL-ін'єкції, XSS та інші типи вразливостей ще на етапі розробки (Wang, 2015). SAST використовує кореляційний аналіз, метрики вразливостей та моделювання загроз для визначення потенційних шляхів атак. Це підвищує ефективність виявлення вразливостей шляхом створення карт вразливостей та дерев атак на основі інформації з CWE (Common Weakness Enumeration) та CVE (Common Vulnerabilities and Exposures) (Charoenwet, 2024).

Однак, виявлено, що метод SAST має свої недоліки. Основні проблеми включають високу залежність від технічних ресурсів та потребу в постійному оновленні інструментів та методик, щоб відповідати новітнім викликам у сфері кібербезпеки. Крім того, SAST не може виявити вразливості, які проявляються лише під час виконання програми, що обмежує його ефективність у виявленні деяких типів атак (Do,2022). Запропоновано методику виявлення вразливості SQL-ін'єкції на основі трансформації програми для вирішення цієї проблеми, яка складається з двох етапів: трансформація програми та виявлення вразливості (Yuan, 2023).

Інструменти статичного аналізу (SAST) є практичним вибором для активного виявлення вразливостей у програмному забезпеченні під час розробки. Національна база даних про вразливості (NVD) у США (NVD, 2023) збирає загальні вразливості та експлойти (CVE), класифіковані через загальні слабкі місця (CWE) (Chen, 2023). NVD також класифікує ступінь небезпеки вразливостей, пов'язаних з конкретними CVE. SAST-інструменти розроблені для виявлення вразливостей, пов'язаних з CWE. Хоча багато SAST-інструментів можуть виявляти однакові вразливості, лише деякі з них можуть виявляти більш специфічні та складні CWE. NIST надає тестові набори SARD, які орієнтовані на найпоширеніші мови програмування для оцінки SAST-інструментів (NIST, 2023).

Серед проаналізованих досліджень можна виділити дві підгрупи: дослідження, що фокусуються на зручності використання і дослідження,

що фокусуються на адаптації та інтеграції інструментів у процеси компаній.

Також досліджено різні контексти, в яких розробники використовують статичні інструменти, такі як середовище розробки, огляд та безперервна інтеграція (Vassallo, 2018). Під час дослідження налаштування для цих контекстів було з'ясовано, що більшість розробників використовують однакові налаштування у всіх середовищах (IDE, безперервна інтеграція чи огляд). Було проведене опитування серед сорока двох учасників, а для підтвердження своїх висновків проведено інтерв'ю з одинадцятьма розробниками з шести компаній.

Ще один важливий експеримент на виявлення вразливостей безпеки за допомогою інструментів статичного аналізу описано в роботі (Smith, 2020). Було запрошено десять розробників з одного проекту виконати чотири завдання з використанням розширеної версії FindBugs. Учасників попросили усно пояснити свої думки, які автори використали для формулювання запитань. Потім вони використовували метод сортування карток для отримання відповідних висновків.

Крім того, досліджено, що SAST можна використовувати для виявлення 76% дефектів перевірки коду, що значно більше, ніж поточні інструменти, які їх успішно виявляють. Було виявлено, що інструменти SAST можуть бути ефективнішими і виявляти більше вразливостей при перевірці коду (Mehrpour, 2022).

Доведено, що SAST є практичним вибором для активного виявлення вразливостей програмного забезпечення під час його розробки (Shen, 2021). Також зазначено, що інтеграція методів статичного аналізу безпеки (SAST) може забезпечити більш комплексний підхід до забезпечення безпеки програмного забезпечення, поєднуючи автоматизовані рев'ю коду з детальним виявленням вразливостей (Tufano, 2023). Подальші дослідження повинні зосередитись на удосконаленні моделей для специфічних задач автоматизації та їх інтеграції з SAST для підвищення рівня безпеки програмних продуктів.

Виявлено розрив між заявленими можливостями SAST та їх фактичною ефективністю, що підкреслює необхідність покладатися на емпіричні дані (Esposito, 2024). Проаналізовано, що жоден інструмент не покриває всі типи вразливостей, тому рекомендується використовувати комбінацію різних SAST та доповнювати їх іншими методами, такими як ручний аналіз коду або машинне навчання, тобто майбутні зусилля мають бути спрямовані на підвищення повноти виявлення (зменшення пропущених вразливостей), навіть якщо це призведе до збільшення кількості помилкових спрацьовувань, оскільки невиявлені вразливості можуть мати серйозні наслідки для безпеки програмного забезпечення.

**Метою дослідження** є оцінка ефективності методу статичного аналізу безпеки (SAST) у виявленні вразливостей типу SQL Injection у програмному забезпеченні. Дослідження спрямоване на аналіз поточної здатності існуючих SAST-інструментів виявляти ці вразливості та розробку удосконалень, які підвищують їхню ефективність. Це досягається шляхом проведення експериментального аналізу, що дозволяє виявити основні обмеження та проблеми сучасних методів статичного аналізу.

Виклад основного матеріалу дослідження

Проаналізуємо SQL ін'єкцію, що сьогодні є не тільки широко поширеною вразливістю, а ще й однією з найнебезпечніших, за версією OWASP (Top 10 Application Security Risks). SQL (Structrued Query Language) – мова бази даних, яка використовується для додавання, видалення, зміни та запитувати дані в реляційній базі даних. Поки система використовує базу даних, з більшою частиною вона взаємодіє бази даних за допомогою операторів SQL.

Суть вразливості – це виконання довільного запиту до бази даних. Запит може бути будь-яким: на читання, запис, модифікацію та видалення будь-яких записів. Крім того, за певних обставин можна дістатися і до читання/запису локальних файлів або навіть до виконання коду. Все залежить від цілей, які переслідує зловмисник, від системи, що використовується, і того, як вона налаштована. Розглянемо типи SQL ін'єкцій.

• Класична SQL Injection – проста та легка в експлуатації. Дозволяє зловмисникові атакувати базу даних (БД) і одразу бачити результат атаки. Останнім часом трапляється нечасто.

• Error-based SQL Injection – трохи складніший і витратний за часом тип атаки, що дозволяє, на основі помилок СУБД, що виводяться, отримати інформацію про всю БД і дані, що зберігаються в ній. Експлуатується, якщо забули відключити виведення помилок.

• Boolean-based SQL Injection – одна зі «сліпих» ін'єкцій. Суть атаки зводиться до додавання спеціального підзапиту у вразливий параметр, який БД відповідатиме або «True», або, несподівано, «False». Атака не дозволяє відразу вивести всі дані БД «на екран» зловмиснику, але дозволяє, перебираючи параметри раз за разом, отримати вміст БД, хоча для

цього буде потрібно тимчасовий відрізок порівнянний зі вмістом БД.

• Time-based SQL Injection – наступна зі «сліпих» ін'єкцій. У цьому випадку зловмисник додає підзапит, що призводить до уповільнення або паузи роботи БД за певних умов. Таким чином, атакуючий, порівнюючи час відповіді на «True» і на «False» запити, символ за символом може отримати весь вміст БД, але часу піде на це більше, ніж у разі експлуатації Boolean-based атаки.

• Out-of-band SQL Injection – рідкісний тип. Атака може бути успішною лише за певних обставин, наприклад, якщо сервер БД може генерувати DNS- або HTTP-запити, що зустрічається нечасто. Також, як і Blind SQL, дозволяє посимвольно збирати інформацію про дані, що зберігаються там.

Виявлено, що основною причиною вразливостей SQL Injection є те, що розробники під час написання коду використовують метод конкатенації рядків для побудови SQL-запитів, які передаються до бази даних. В результаті, зловмисники можуть змінювати SQL-запит, вводячи ключові слова SQL або спеціальні символи. Внаслідок виконання такого запиту система зазнає атаки. Фундаментальною причиною є надмірна довіра до даних, введених користувачами, без належної фільтрації введення та без здійснення раціональної верифікації на стороні сервера, що дозволяє зловмиснику досягати своїх цілей, таких як крадіжка конфіденційної інформації системи або отримання контролю над сервером. На рис. 1 показано основні принципи та процеси атаки SQL Injection.

SQL injection є поширеним типом веб-вразливості, що дозволяє зловмиснику обійти механізми автентифікації та авторизації застосунку шляхом створення шкідливих SQL-запитів для отримання конфіденційних даних або виконання незаконних операцій з базою даних. У звичайному методі дані, як правило, вводяться користувачем і певним чином передаються до Backend-бази даних для відповідних запитів, оновлень та видалень. Якщо метод не належним чином фільтрує або не ухиляється від цих введених користувачем даних, зловмисник може вставити шкідливі SQL-інструкції у введення, що призводить до виконання методом неочікуваних запитів або дій, таких як видалення таблиць, вставка даних або витік конфіденційної інформації.

Розглянемо приклад ін'єкції SQL-коду у веб-застосунку. Багато методів приймають введення користувача і передають їх у попередньо визначений запит, який потім передається до бази даних для виконання. Якщо розробники не налаштують код додатка належним чином для захисту від неочікуваного введення даних користувачами, зміни структури бази даних, пошкодження даних або розкриття приватної та конфіденційної інформації можуть статися ненавмисно.

Наприклад, розглянемо сторінку входу CGI-методу, яка очікує ім'я користувача та відповідний пароль. Коли облікові дані вводяться, вони вставляються у шаблон запиту, такий як:

```
select * from mysql.user
where username=' «. $uid. « ' and
password=password(' «. $pwd. « ');
```

Замість дійсного імені користувача, зловмисник задає змінну $uid як рядок: ' or 1=1; --, що змушує CGI-скрипт створити наступний SQL-запит до бази даних:

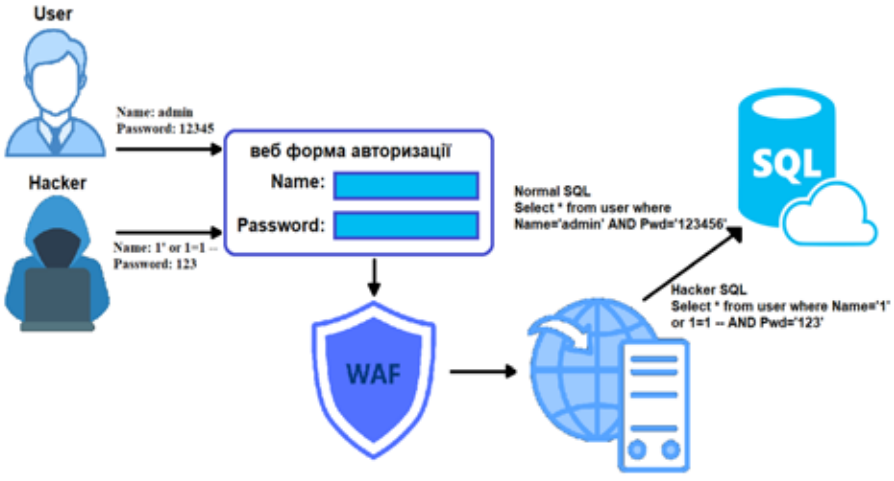


**Рис. 1. Схема основних принципів та процесів атаки SQL Injection**

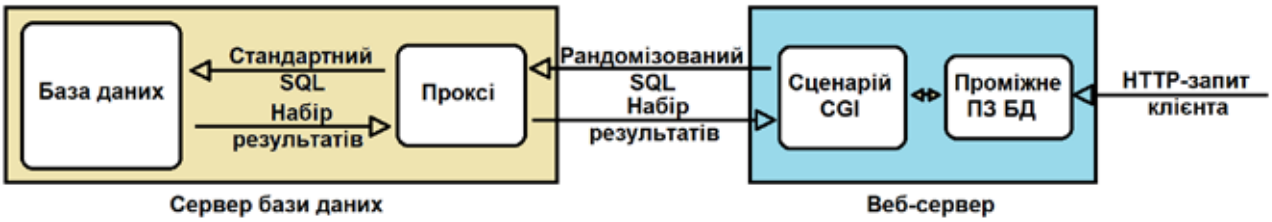


**Рис. 2. Базові принципи проникнення SQL ін'єкції**

```
select * from mysql.user
where username='' or 1=1; --' and
password=password('_any_text_');
```

Зауважимо, що одинарні лапки врівноважують лапки у попередньо визначеному запиті, а подвійне тире коментує решту SQL-запиту. Таким чином, значення паролю не має значення і може бути встановлено будь-яким рядком. Набір результатів запиту міститиме принаймні один запис, оскільки умова where є істинною. Якщо метод визначає дійсного користувача, перевіряючи, чи є набір результатів непорожнім, зловмисник може обійти перевірку безпеки.

**Метод покращення SAST для захисту від SQL-ін'єкцій**

Проаналізувавши дослідження науковців, було вирішено, як можна покращити SAST, щоб запобігти SQL Injection. Тому, запропоновано наступні дії.

1. Розробка та впровадження більш складних і спеціалізованих правил для виявлення SQL Injection. Ці правила повинні враховувати різні способи формування SQL-запитів, включаючи використання параметризованих запитів, ORM (Object-Relational Mapping) та інших засобів роботи з базами даних. Реалізувати це можна за допомогою виявлення шаблонів рядків, тобто SAST-інструменти повинні мати можливість виявляти місця в коді, де введення користувача об'єднується з SQL-запитами через конкатенацію рядків. Це можуть бути рядки, що містять ключові слова SQL, або змінні, які безпосередньо впливають на структуру запиту. Також треба запровадити аналіз використання ORM – це означає, що ORM-інструменти спрощують взаємодію з базами даних, але можуть також бути джерелом вразливостей, якщо використовуються неправильно. SAST-інструменти повинні мати правила для перевірки правильності використання ORM, включаючи перевірку наявності параметризації запитів та уникнення динамічних запитів.

2. Інтеграція з іншими інструментами безпеки. Інтеграція SAST-інструментів з іншими засобами безпеки, такими як динамічні аналізатори (DAST) та засоби безперервної інтеграції та доставки (CI/CD). Це дозволить отримати більш повну картину безпеки додатків і знизити ризик пропуску вразливостей. Реалізація вимагає синхронізацію з DAST, результати динамічного аналізу можуть бути використані для уточнення правил статичного аналізу. Наприклад, виявлені DAST-інструментом вразливості можуть вказати на проблемні місця в коді, які варто перевірити SAST-інструментом. Інтеграція з CI/CD дозволить автоматично сканувати код на кожному етапі розробки, що забезпечує постійну перевірку безпеки. SAST-інструменти можуть бути інтегровані в конвеєр CI/CD для регулярного аналізу коду при кожному коміті або зборці.

3. Автоматичне виправлення вразливостей вимагає розробку функціональності для автоматичного створення виправлень для виявлених вразливостей. Ці виправлення можуть включати рефакторинг коду для використання параметризованих запитів замість конкатенації рядків або впровадження належного очищення введених даних. Результатом буде автоматичне заміщення SQL-запитів: SAST-інструменти можуть автоматично знаходити місця в коді, де використовуються небезпечні SQL-запити, і пропонувати зміни для їх заміни на параметризовані запити. Це включає зміну синтаксису запиту та додавання відповідних параметрів. Також плюсом буде рефакторинг коду: Інструмент може автоматично генерувати патчі для коду, що виправляють вразливості, та надсилати їх розробникам для перевірки і впровадження. Це значно зменшує час, необхідний для усунення вразливостей.

4. Контекстуальний аналіз. Впровадження можливостей контекстуального аналізу, які дозволяють SAST-інструментам краще розуміти контекст, в якому використовуються SQL-запити. Це допоможе виявляти більш складні та приховані вразливості. Наприклад аналіз шляхів передачі даних, тобто SAST-інструменти можуть відстежувати шлях даних від точки введення до точки використання в SQL-запитах. Це допомагає виявити місця, де дані можуть бути змінені або неправильно оброблені. Також інструмент

повинен аналізувати контекст виконання SQL-запитів, включаючи умови, цикли та інші конструкції, які можуть впливати на безпеку запиту. Це дозволяє виявляти вразливості, які можуть бути приховані у складних логічних конструкціях.

**Експеримент**

Для оцінки ефективності нового методу статичного аналізу безпеки (SAST) у виявленні SQL Injection був проведений експеримент, використовуючи дані реальних комітів, що сприяли виникненню вразливостей.

1. Підготовка даних.

Було використано набір даних Vulnerability Contributing Commits (VCCs), наданий Iannone та ін., зібраний з проєктів GitHub, пов'язаних із вразливостями в базі даних CVE. Ми обрали VCCs за такими критеріями:

1. Мова програмування: проєкти, реалізовані на C або C++, оскільки ці мови дозволяють низькорівневі операції, що можуть призвести до вразливостей.

2. Призначений тип вразливості: VCCs, пов'язані з CVE, які мають призначений тип вразливості (CWE), зокрема SQL Injection.

3. Можливість компіляції: VCCs з проєктів, які можуть бути успішно скомпільовані, щоб забезпечити коректну роботу SAST-інструментів.

Схему процесу підготовки даних зображено на рис. 3.

2. Виконання експерименту.

Було обрано безкоштовні, активно підтримувані SAST-інструменти, які можуть працювати через командний рядок (CLI). Вибір інструментів базувався на попередніх дослідженнях та списку інструментів з NIST's Software Assurance Metrics And Tool Evaluation (SAMATE). Ми використовували найновіші стабільні версії інструментів, зокрема CodeChecker v6.24.1, CodeQL v1.15, Flawfinder v2.0.19 та інші, на автоматизованій платформі з використанням Docker-контейнерів для ізоляції середовища.

Попередження, згенеровані SAST-інструментами, були згруповані за типами вразливостей, зокрема за SQL Injection, шляхом мапінгу ідентифікаторів CWE відповідно до офіційної документації інструментів та стандартів CWE.

3. Аналіз та результати.

Було виміряно кількість VCCs, на яких SAST-інструменти могли надати попередження про SQL Injection. Результати показали, що новий метод SAST виявив вразливості у 78% обраних комітів. Цей показник було отримано шляхом співвідношення кількості виявлених вразливостей до загальної кількості VCCs, які відповідали нашим критеріям.

Аналіз продемонстрував, що пріоритизація на основі попереджень SAST дозволила

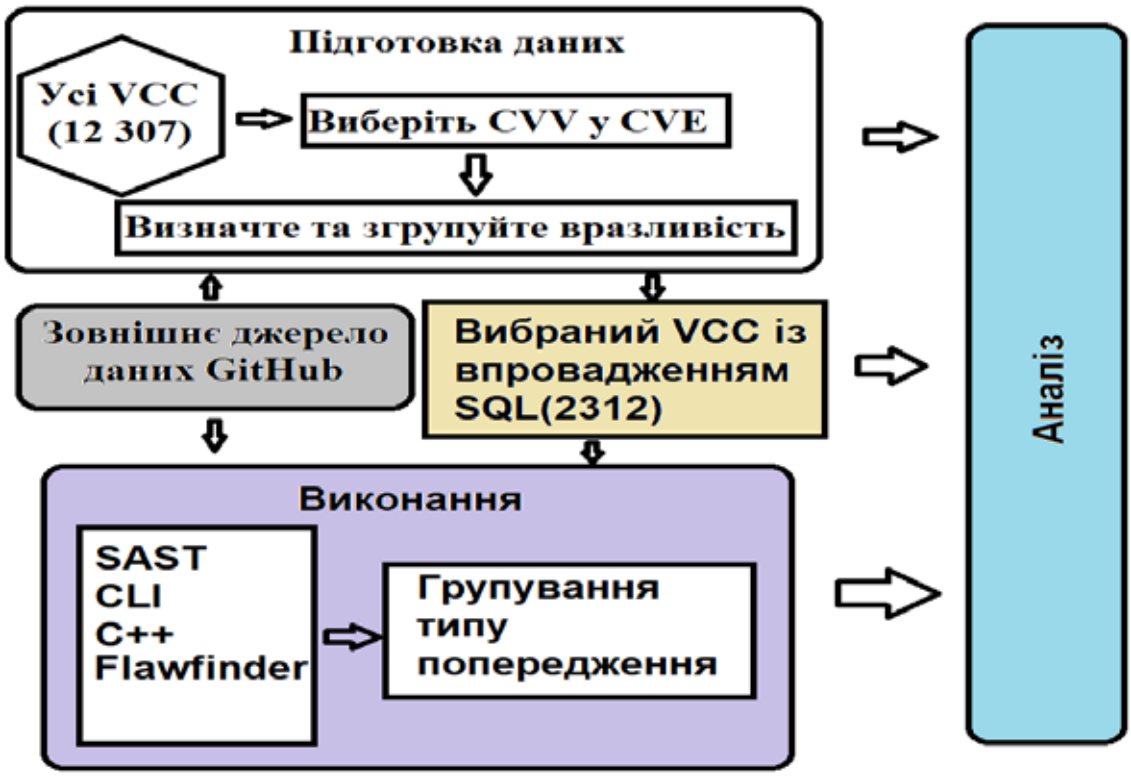


**Рис. 3. Схема процесів при експерименті**

рев'юерам виявити на 45% більше вразливих функцій при обмежених затратах зусиль. Це було досягнуто шляхом порівняння ефективності виявлення вразливостей з використанням та без використання пріоритизації.

Середній час обробки кожного коміту склав 12 хвилин, що було виміряно під час експерименту за допомогою нашої автоматизованої платформи. Це відповідає допустимому часу для автоматизованих тестів під час код-рев'ю, забезпечуючи інтеграцію процесу аналізу в робочий цикл розробки.

**Висновки і перспективи подальших досліджень.** У цьому дослідженні було розглянуто метод статичного аналізу безпеки (SAST) з особливим акцентом на виявлення вразливостей типу SQL Injection. Було проведено експеримент із використанням удосконаленого методу SAST, який включає розробку більш складних і спеціалізованих правил для виявлення SQL Injection та інтеграцію з іншими інструментами безпеки. Результати експерименту показали, що удосконалений метод SAST здатен виявляти вразливості типу SQL Injection у 78% випадків. Це свідчить про значне покращення порівняно з традиційними підходами, оскільки він дозволяє більш ефективно ідентифікувати специфічні типи вразливостей, підвищуючи загальний рівень захисту від кіберзагроз. Крім того, використання цього методу SAST дозволяє пріоритизувати перевірку змін у коді, що сприяє виявленню на 45% більше вразливих функцій при обмежених ресурсах для перевірки. Це підвищує ефективність процесу ревізії коду та забезпечує більш надійний захист програмного забезпечення.

Щодо часу обробки, середній час аналізу для кожного коміту склав 12 хвилин, що відповідає прийнятному часу для автоматизованих тестів під час процесу ревізії коду та демонструє, що удосконалений метод SAST є ефективним з точки зору продуктивності. Для підвищення безпеки програмного забезпечення було запропоновано кілька удосконалень методу SAST, включаючи розробку більш складних правил для виявлення SQL Injection, інтеграцію з іншими інструментами безпеки, автоматичне виправлення вразливостей та впровадження контекстуального аналізу.

Отже, результати нашого дослідження демонструють значний потенціал удосконаленого методу SAST у підвищенні ефективності виявлення вразливостей типу SQL Injection та забезпеченні більш надійного захисту програмного забезпечення. Майбутні дослідження можуть бути спрямовані на подальше вдосконалення інструментів SAST та їх інтеграцію з іншими методами забезпечення кібербезпеки.

**ЛІТЕРАТУРА:**

1. Cavelty M. D. The Politics of Cyber-Security. New York, 2024. P. 224. DOI: org/10.4324/9781003497080 (дата звернення: 19.08.2024).

2. Luo C., Li P., Meng W. T. TChecker: Precise Static Inter-Procedural Analysis for Detecting Taint-Style Vulnerabilities in PHP Applications. *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security USA.* 07 November, 2022. P. 2175–2188. DOI: 10.1145/3548606.3559391.

3. Wang Y., Wang D., Zhao W., Liu Y. Detecting SQL Vulnerability Attack Based on the Dynamic and Static Analysis Technology. *IEEE 39th Annual Computer Software and Applications Conference*. 2015. P. 604–607. DOI: 10.1109/COMPSAC.2015.277.

4. Charoenwet W., Thongtanunam P., Pham V., & Treude, C. An Empirical Study of Static Analysis Tools for Secure Code Review. *In Proceedings of ACM SIGSOFT International Symposium on Software Testing and Analysis. ACM.* 2024. New York P. 13. DOI: 10.48550/arXiv.2407.12241.

5. Do L. N. Q., Wright J. R., Ali K. Why Do Software Developers Use Static Analysis Tools? A User-Centered Study of Developer Needs and Motivations. *IEEE Transactions on Software Engineering*. 2022. Vol. 48, № 3, P. 835–847. DOI: 10.1109/TSE.2020.3004525.

6. Yuan Ye, Yuliang Lu, Kailong Zhu, Hui Huang, Lu Yu and Jiazhen Zhao. A Static Detection Method for SQL Injection Vulnerability Based on Program Transformation. *Applied Sciences.* 2023. Vol. 13, № 21. DOI: 10.3390/app132111763.

7. NVD. National vulnerability database. 2023. URL: https://nvd.nist.gov/ (дата звернення: 19.08.2024).

8. Chen Z., Cao J. VMCTE: visualization-based malware classification using transfer and ensemble learning. *Computers, Materials & Continua.* 2023. № 75(2), P. 4445–4465. DOI:10.32604/cmc.2023.038639.

9. NIST. National institute of standards and technology. URL: https://www.nist.gov/(дата звернення: 19.08.2024).

10. Vassallo C., Panichella S., Palomba F., Proksch S., Zaidman A., Gall H. C., Context is king: The developer perspective on the usage of static analysis tools. *IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER).* 2018. P. 38–49. DOI: 10.1109/SANER.2018.8330195.

11. Smith M., Naiakshina A, Danilova A., Gerlitz E. On conducting security
developer studies with cs students: Examining a password-storage study with cs students, freelancers, and company developers. *Proceedings of the Conference on Human Factors in Computing Systems, Association for Computing Machinery.* 2020. P. 1–12. DOI: 10.1145/3290605.3300370.

12. Mehrpour S., LaToza T. D. Can static analysis tools find more defects? *Empir Software Eng*. 2023. Vol.28. № 5. DOI:10.1007/s10664-022-10234.

13. Shen S., Kolluri A., Dong Z., Saxena P., Roychoudhury A. Localizing vulnerabilities statistically from one exploit. *In Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security*. 2021. P. 537–549. DOI:10.1145/3433210.3437528.

14. Tufano R., Dabić O., Mastropaolo A., Ciniselli M. and Bavota G. Code review automation: strengths and weaknesses of the state of the art. IEEE Transactions on Software Engineering, 2023. P. 1–16. DOI: 10.1109/TSE.2023.3348172.

15. Esposito M., Falaschi V., Falessi D. An Extensive Comparison of Static Application Security Testing Tools.2024. DOI: 10.13140/RG.2.2.12326.54085.

16. Esposito M., Moreschini S., Lenarduzzi V., Hästbacka D., Falessi D. Can we trust the default vulnerabilities severity? *In IEEE 23rd International Working Conference on Source Code Analysis and Manipulation (SCAM).* 2023. P. 265–270. DOI: 10.1109/SCAM59687.2023.00037.

17. Lysenko S, Lysenko S., Bobrovnikova K., Kharchenko V., Savenko O. IoT multi-vector cyberattack detection based on machine learning algorithms: traffic features analysis, experiments, and efficiency. *Algorithms.* 2022. Vol 15. № 7. P. 239. DOI: 10.3390/a15070239

18. Website of Our Study. Static Application Security Testing (SAST) Tools for Smart Contracts: How Far Are We? 2024. URL: https://sites.google.com/view/sc-sast-study-fse2024/home (Accessed on 29/07/2024).

19. Azman M., Marhusin M. F., Sulaiman R. Machine Learning – Based Technique to Detect SQL Injection Attack. *Journal of Computer Science.* 2021. № 17. P. 296–303. DOI:10.3844/jcssp.2021.296.303.

20. Medeiros P. I., Fonseca J., Neves N., Correia M., Vieira M. Benchmarking Static Analysis Tools for Web Security. in *IEEE Transactions on Reliability*. 2018. V. 67. № 3. P. 1159–1175. DOI: 10.1109/TR.2018.2839339.

21. Charoenwet W., Thongtanunam P., Pham V. T., Treude C. An Empirical Study of Static Analysis Tools for Secure Code Review. *In Proceedings of ACM SIGSOFT International Symposium on Software Testing and Analysis.* 2024. P. 13 DOI: 10.48550/arXiv.2407.12241.

22. Savenko B., Lysenko S., Bobrovnikova K., Savenko O., Markowsky G. Detection DNS tunneling botnets. *11th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS)*, 22 September, 2021. P. 64–69. DOI: 10.1109/IDAACS53288.2021.9661022

**REFERENCES:**

1. Dunn Cavelty, M. (2024). The Politics of Cyber-Security (1st ed.). New York, Routledge https://doi.org/10.4324/9781003497080 [in English].

2. Luo, C., Li, P., Meng, W. T. (2022, November). TChecker: Precise Static Inter-Procedural Analysis for Detecting Taint-Style Vulnerabilities in PHP Applications. *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security USA,* 2175–2188 https://doi.org/10.1145/3548606.3559391

3. Wang, Y., Wang, D., Zhao, W., Liu, Y. (2015, September). Detecting SQL Vulnerability Attack Based on the Dynamic and Static Analysis Technology. *IEEE 39th Annual Computer Software and Applications Conference*, 604–607 https://doi.org/10.1109/COMPSAC.2015.277

4. Charoenwet, W., Thongtanunam, P., Pham, V. & Treude, C. (2024). An Empirical Study of Static Analysis Tools for Secure Code Review. *In Proceedings of ACM SIGSOFT International Symposium on Software Testing and Analysis. ACM, New York, NY, USA, 13.* https://doi.org/10.48550/arXiv.2407.12241

5. Do L. N. Q., Wright J. R., Ali K. (2022, March). Why Do Software Developers Use Static Analysis Tools? A User-Centered Study of Developer Needs and Motivations. *IEEE Transactions on Software Engineering*, 48, (3), 835-847. https://doi.org/10.1109/TSE.2020.3004525

6. Yuan, Ye, Yuliang Lu, Kailong Zhu, Hui Huang, Lu Yu, and Jiazhen Zhao. (2023). «A Static Detection Method for SQL Injection Vulnerability Based on Program Transformation» *Applied Sciences* 13, (21): 11763. https://doi.org/10.3390/app132111763

7. NVD. (2023). National vulnerability database https://nvd.nist.gov/ [in English].

8. Chen, Z., Cao, J. (2023). VMCTE: visualization-based malware classification using transfer and ensemble learning. *Computers, Materials & Continua, 75(2),* 4445–4465. https://doi.org/10.32604/cmc.2023.038639

9. NIST. (2023). National institute of standards and technology. https://www.nist.gov/ [in English].

10. Vassallo, C., Panichella, S., Palomba, F., Proksch, S., Zaidman, A., Gall, H. C.(2018). Context is king: The developer perspective on the usage of static analysis tools. *IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER),* 38–49 https://doi.org/10.1007/S10664-019-09750-5/TABLES/9

11. Smith, M., Naiakshina, A, Danilova, A., Gerlitz, E. (2020). On conducting security developer studies with cs students: Examining a password-storage study with cs students, freelancers, and company developers. *Proceedings of the Conference on Human Factors in Computing Systems, Association for Computing Machinery.* pp. 1–12. https://doi.org/10.1145/3290605.3300370

12. Mehrpour, S., LaToza, T. D. (2023). Can static analysis tools find more defects? *Empir Software Eng* 28 (5). https://doi.org/10.1007/s10664-022-10234

13. Shen, S., Kolluri, A., Dong, Z., Saxena, P., Roychoudhury, A. (2021). Localizing vulnerabilities statistically from one exploit. *In Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security*. 9781450382878 pp. 537–549. https://doi.org/10.1145/3433210.3437528

14. Tufano, R., Dabić, O., Mastropaolo, A., Ciniselli, M., and Bavota, G. (2023). Code review automation: strengths and weaknesses of the state of the art. IEEE Transactions on Software Engineering, 1–16. https://doi.org/10.1109/TSE.2023.3348172

15. Esposito, M., Falaschi, V., Falessi, D. (2024). An Extensive Comparison of Static Application Security Testing Tools. https://doi.org/10.13140/RG.2.2.12326.54085

16. Esposito, M., Moreschini, S., Lenarduzzi, V., Hästbacka, D., Falessi, D. (2023). Can we trust the default vulnerabilities severity? *In IEEE 23rd International Working Conference on Source Code Analysis and Manipulation (SCAM),* 265–270. https://doi.org/10.1109/SCAM59687.2023.00037

17. Lysenko, S., Bobrovnikova, K., Kharchenko, V. & Savenko, O. (2022). IoT multi-vector cyberattack detection based on machine learning algorithms: traffic features analysis, experiments, and efficiency. *Algorithms*, *15*(7), 239. https://doi.org/10.3390/a15070239

18. Website of Our Study. (2024). Static Application Security Testing (SAST) Tools for Smart Contracts: How Far Are We? https://sites.google.com/view/sc-sast-study-fse2024/ home (Accessed on 29/07/2024).

19. Azman, M., Marhusin, M. F., Sulaiman, R. (2021). Machine Learning-Based Technique to Detect SQL Injection Attack. *Journal of Computer Science. 17,* 296–303. https://doi.org/10.3844/jcssp.2021.296.303

20. Medeiros, P. I., Fonseca, J., Neves, N., Correia, M., Vieira, M. (2018, September). Benchmarking Static Analysis Tools for Web Security. in *IEEE Transactions on Reliability*, V. 67, (3), 1159–1175. https://doi.org/10.1109/TR.2018.2839339

21. Charoenwet, W., Thongtanunam, P., Pham, V. T., Treude, C. (2024). An Empirical Study of Static Analysis Tools for Secure Code Review. *In Proceedings of ACM SIGSOFT International Symposium on Software Testing and Analysis,* 13 https://doi.org/10.48550/arXiv.2407.12241

22. Savenko, O., Sachenko, A., Lysenko, S., Markowsky, G. & Vasylkiv, N. (2020). Botnet detection approach based on the distributed systems. *International Journal of Computing*, *19*(2), 190–198. https://doi.org/10.47839/ijc.19.2.1761