

**ЧЕРКАСЬКИЙ ДЕРЖАВНИЙ ТЕХНОЛОГІЧНИЙ УНІВЕРСИТЕТ
МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

Кваліфікаційна наукова
праця на правах рукопису

Супруненко Ілля Олександрович

УДК 004.056.5

ДИСЕРТАЦІЯ

**МЕТОД АДАПТИВНОГО ЛОГУВАННЯ КОМП'ЮТЕРНИХ
СИСТЕМ ТА ПРОГРАМ**

123 – комп'ютерна інженерія

Подається на здобуття наукового ступеня доктора філософії.

Дисертація містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

_____І.О. Супруненко

Наукові керівники:

Нечипоренко Ольга Володимирівна,

кандидат технічних наук, доцент

Рудницький Сергій Володимирович,

кандидат технічних наук, доцент

АНОТАЦІЯ

Супруненко І.О. Метод адаптивного логування комп'ютерних систем та програм. – Кваліфікаційна наукова праця на правах рукопису.

Дисертація на здобуття наукового ступеня доктора філософії за спеціальністю 123 «Комп'ютерна інженерія». – Черкаський державний технологічний університет, Черкаси, 2026.

Дисертаційна робота присвячена вирішенню науково-технічного завдання з підвищення рівня спостережності та оперативності контролю роботи комп'ютерних систем та програм за рахунок розробки та впровадження методу адаптивного логування.

У першому розділі за результатами аналізу поточного рівня розвитку та місця комп'ютерів і програмного забезпечення в сучасному житті, а також деяких із найбільш гучних кіберінцидентів останніх років, була встановлена необхідність забезпечення належного рівня інформаційної безпеки розроблюваних систем та програм. Проведений аналіз показав, що деякі напрями кібербезпеки представлені більш повно (як у випадку широкого вибору та доступності антивірусного програмного забезпечення чи можливості використовувати технології віртуальних приватних мереж для конфіденційної роботи із віддалених пристроїв), в той час як аспект спостережності потребує глибшого дослідження. На основі проведеного аналізу сучасного стану програмних засобів забезпечення аспекту спостережності було сформульовано мету і завдання дисертаційного дослідження.

У другому розділі на основі формалізації функції логування базового методу, що орієнтований лише на критичність, було представлено базові функції логування та реініціалізації методу адаптивного логування. Відповідно до проведеного порівняння, керуючись міркуваннями створення архітектури, яка матиме змогу бути застосованою в більшій кількості платформ, середовищ та мов програмування, носієм для сигналу реініціалізації було обрано механізм міжпроцесових сигналів операційної системи Linux. Програмну архітектуру імплементації методу адаптивного логування запропоновано здійснювати з

використанням патернів програмування Фасад, що дозволить використовувати довільний спосіб безпосередньої взаємодії із системами введення-виведення операційних систем, написаних самостійно чи на основі існуючих перевірених рішень, та Синглтон, завдяки якому досягається єдиність та одночасність функціонування як процесу логування, так і процесу реініціалізації імплементації методу адаптивного логування.

В третьому розділі були розширені формальні основи моделей сигнатур логування та реініціалізації з подальшим включенням динамічного варіанту повідомлення, що був створений з використанням можливостей сучасних мов програмування до формування виконуваного коду безпосередньо в процесі роботи програми, що дала більшу гнучкість при роботі з компонентами системи, деталі роботи яких не є фіналізованими і можуть потребувати глибшої відладки вже будучи розгорнутими у відповідному середовищі розробки на віддалених потужностях. Для зменшення ризиків некоректного та зловмисного використання даної можливості було проведено дослідження різних підходів до аналізу поведінки виконуваного коду та обрано механізм аналізу абстрактних синтаксичних дерев вихідного коду за допомогою валідації через JSON-схеми. Обраний механізм валідації було додано до складових конфігураційного словника та передбачено можливість змінювати критерії перевірки не впливаючи на програмні компоненти системи. Оновлені моделі сигнатур методу адаптивного логування, доповнені динамічним варіантом повідомлення, були представлені у вигляді фіналізованої архітектури методу адаптивного логування.

В четвертому розділі, використовуючи можливості інфраструктури провайдера хмарних послуг Amazon Web Services, було продемонстровано приклад розгортання методу адаптивного логування та процес взаємодії з системою через адміністративний інтерфейс. Імплементація аспектів аутентифікації, які були винесені за межі формальних основ методу адаптивного логування, була здійснена з використанням можливостей програмного забезпечення Session Manager service Systems Manager. Процедура розгортання з використанням технології контейнеризації очікувано підтвердила простоту та ідентичність розгортання і застосування процедури реініціалізації як на

локальній машині розробника, так і на віддаленому сервері. Також було проведено обчислювальний експеримент для безпосереднього порівняння базового підходу до логування з методом адаптивного логування, результати якого підтвердили перевагу методу адаптивного логування над базовим, що спирається лише на критичність. В питанні підвищення рівня спостережності це було досягнуто за рахунок збільшення частки лог-записів після реініціалізації у відповідності до нових вимог, а підвищення оперативності контролю було досягнуто завдяки виключенню необхідності перезапуску системи для застосування оновленої конфігурації логування.

Наукова новизна отриманих результатів:

- вперше побудовано моделі сигнатур адаптивного логування за рахунок формалізації і вдосконалення сигнатур базової моделі логування, яка орієнтується лише на критичність, та створення можливості реініціалізації процесів, що забезпечило підвищення рівня точності логування;
- вперше побудовано метод адаптивного логування на основі логування повідомлень шляхом застосування моделей сигнатур та динамічного варіанту повідомлень, що забезпечило підвищення рівня спостережності та оперативності сигналювання про непередбачувані та критичні ситуації в роботі систем та програм;
- отримали подальший розвиток методи контролю роботи програм для хмарних обчислень за рахунок впровадження методу адаптивного логування з використанням хмарних сервісів.

Практичне значення отриманих результатів.

Практична цінність роботи полягає в доведенні розробленого підходу, до алгоритмів, функціональних схем і методики застосування, що в сукупності дозволяють отримати вищий рівень деталізації та інформативності стосовно процесів, які відбуваються під час виконання програмного коду. Запропоновані автором для використання модулі програмного забезпечення, були підібрані таким чином, щоб мати можливості для імплементації в якомога більшій кількості платформ, мов програмування та середовищ виконання програмного

коду. Для отриманого за результатами дослідження варіанту методу адаптивного логування з динамічними повідомленнями розроблено ескізний зразок модуля впровадження на інфраструктурі провайдера хмарних послуг Amazon Web Services, із врахуванням аспектів належного рівня безпеки при взаємодії користувачів з адміністративним рівнем доступу, а також простоту та прогнозованість переносу артефактів готової системи між різними середовищами розробки.

За результатами обчислювального експерименту при порівнянні з базовим алгоритмом логування, що спирається лише на критичність, було отримано збільшення інформативності результуючих лог-записів на 31% у відповідності до виконання задачі відлагодження конкретного компоненту системи. Водночас окрема можливість вказати рівень критичності, вище якого застосування фільтрації є небажаним, дозволило зберегти всі повідомлення про помилки із решти компонентів системи, що не перебували у фокусі уваги. Порівнюючи два аналогічних сценарії реініціалізації було продемонстровано, що зменшення кількості надлишкових повідомлень у випадку адаптивного методу склало 61.3%, що на 0.7% більше, ніж для базового варіанту. Також втрата клієнтських повідомлень, що склала 14% при реініціалізації базового варіанту, була цілком відсутня у випадку з адаптивним методом, оскільки запропонована архітектура більш придатна для сценарію адаптації до змін у вимогах логування при активній роботі системи.

Результати дисертаційного дослідження, а саме метод адаптивного логування з динамічним варіантом повідомлень та підсистемою валідації, були використані для контролю коректності розробки програмного забезпечення системи дистанційного управління наземним самохідним роботизованим комплексом виробництва ТОВ “МОРОЗ ТЕХ”.

Ключові слова: комп’ютерна система, інформаційна безпека, кіберзагрози, контроль, спостережність, моніторинг, логування, сигнатури, статичний аналіз коду

ABSTRACT

Suprunenko I.O. Adaptive logging method of computer systems and programs.
– Qualification scientific work in the form of a manuscript.

Dissertation for the degree of Doctor of Philosophy in specialty 123 "Computer Engineering". – Cherkasy State Technological University, Cherkasy, 2026.

The dissertation is dedicated to solving the scientific and technical problem of increasing the level of observability and efficiency of the control of operation of computer systems and programs through the development and implementation of the adaptive logging method.

In the first section, based on the analysis of the current level of development and the place of computers and software in modern life, as well as some of the most impactful cyber incidents of recent years, the necessity to ensure the proper level of information security of systems and programs, that are being developed, was established. The analysis showed that some areas of cybersecurity are represented more extensively (as in the case of a wide variety and availability of antivirus software or the ability to use virtual private network technologies for confidential work from remote devices), while the aspect of observability requires deeper research. Based on the analysis of the current state of software tools for ensuring the aspect of observability, the goal and objectives of the dissertation research were formulated.

In the second section, based on the formalization of the logging function of the basic method, focused only on severity, the basic logging and reinitialization functions of the adaptive logging method were presented. According to the comparison, in order to design an architecture that would be applicable in a larger number of platforms, environments and programming languages, the interprocess signaling mechanism of the Linux operating system was chosen as the carrier for the reinitialization signal. The software architecture for implementing the adaptive logging method is proposed to be developed using the Facade programming pattern, which would allow to use any method of interaction with the input-output systems of operating systems, written independently or based on existing tested solutions, and Singleton, which would allow for unified and simultaneous functioning of both the logging process and the

reinitialization procedure of the implementation of the adaptive logging method.

In the third section, the formal foundations of the logging and reinitialization signature models were expanded with the subsequent inclusion of a dynamic message variant, which was created using the capabilities of modern programming languages to generate executable code directly during the program operation, which provided greater flexibility when working with system components, the details of which are not yet finalized and may require deeper debugging while already deployed in the appropriate development environment on remote resources. To reduce the risks of incorrect and malicious use of this feature, a comparison of different approaches to analyzing the behavior of executable code was conducted and a mechanism for analyzing abstract syntax trees of the source code using validation via JSON schemas was selected. The selected validation mechanism was added to the components of the configuration dictionary and provided the ability to change the verification criteria without affecting the software components of the system. Updated signature models of the adaptive logging method, enhanced by the dynamic message variant, were presented as a finalized architecture of the adaptive logging method.

In the fourth section, using the capabilities of infrastructure of the cloud service provider Amazon Web Services, an example deployment of the adaptive logging method and the process of interacting with the system through the administrative interface was demonstrated. The implementation of authentication aspects that were extracted out of the formal foundations of the adaptive logging method was accomplished using the capabilities of the Session Manager software of the Systems Manager service. The deployment procedure using containerization technology, as expected, confirmed that both the deployment and invocation of the reinitialization procedure were simple and identical on the developer's local machine and on the remote server. A computational experiment was also conducted to compare the basic logging approach with the adaptive logging method directly, the results of which confirmed that the adaptive logging method is an improvement over the basic, based only on the severity, as it was shown to increase the level of observability, increasing the proportion of log entries after reinitialization in accordance with new requirements, and the efficiency of control, which was achieved by eliminating the need to restart the

system to apply the updated logging configuration.

Scientific novelty of the obtained results:

- for the first time, adaptive logging signature models were developed by formalizing and improving the signatures of the basic logging model, with an orientation only on severity, and creating the possibility of processes reinitialization, which ensured an increase in the level of logging accuracy;

- for the first time, an adaptive logging method was developed based on logging of messages utilizing new signature models and a dynamic message variant, which ensured an increase of the observability level and efficiency of signaling about unpredictable and critical situations in the operation of systems and programs;

- methods for controlling the operation of cloud-based programs were further developed by implementing the adaptive logging method using cloud services.

Practical significance of the results.

The practical value of the work lies in development of algorithms, functional schemes and application methodology, which altogether allow to obtain a higher level of detail and informativeness regarding the processes that occur during the execution of the program code. The software modules proposed by the author for use were developed in such a way so that it is possible to implement those in as many platforms, programming languages and execution environments as possible. For the adaptive logging method with dynamic message variant developed as the result of the study, an example implementation utilizing the infrastructure of the cloud service provider Amazon Web Services was developed, taking into account the aspects of the appropriate level of security during user interaction with the administrative level capabilities, as well as the simplicity and predictability of transferring artifacts of the finished system between different development environments.

According to the results of the computational experiment, when compared with the basic logging algorithm, which relies only on severity, an increase in the informativeness of the resulting log records by 31% was observed in accordance with the task of debugging a specific system component. At the same time, a separate capability to specify the severity level above which the application of filtering is undesirable allowed to preserve all error messages from other system components that

were not in the focus of attention. Comparing two similar reinitialization scenarios, it was demonstrated that the reduction in the number of redundant messages in the case of the adaptive method was 61.3%, which is 0.7% more than for the basic variant. Also, the loss of client messages, which amounted to 14% when reinitializing the basic variant, was completely absent in the case of the adaptive method, since the proposed architecture is more suitable for the scenario of adapting to changes in logging requirements during active system operation.

The results of the dissertation research, namely the adaptive logging method with a dynamic message variant and a validation subsystem, were used to control the correctness of the development of the software for the remote control system of a ground-based self-propelled robotic complex developed by MOROZ TECH LLC.

Keywords: computer system, information security, cyber threats, control, observability, monitoring, logging, signatures, static code analysis

Список публікацій здобувача:

1. Suprunenko I., Rudnytskyi V. On specifics of adaptive logging method implementation. Bulletin of Cherkasy State Technological University. 2024. Vol. 29, no. 1. P. 36–42. DOI: 10.62660/bcstu/1.2024.36
2. Супруненко І. О., Бабенко В. Г., Рудницький С. В. Метод адаптивного логування розподілених комп'ютерних систем. Технології розвитку безпілотних систем : кол. монографія / під ред. В. М. Рудницького. Черкаси : видавець Вовчок О. Ю., 2025. Т. 2. Безекіпажні системи. С. 135–156. ISBN: 978-617-8725-03-7. DOI: 10.5281/zenodo.19191122
3. Suprunenko I., Rudnytskyi V. Comparison of message passing systems in context of adaptive logging method. Visnyk of Kherson National Technical University. 2024. No. 2 (89). P. 228–234. DOI: 10.35546/kntu2078-4481.2024.2.32
4. Suprunenko I., Rudnytskyi V. Dynamic message variant in adaptive logging method. Modern Information Security. 2024. Vol. 59, no. 3. P. 94–99. DOI: 10.31673/2409-7292.2024.030010
5. Suprunenko I., Rudnytskyi V. Validation of dynamic message variant in adaptive logging method. Measuring and Computing Devices in Technological Processes. 2024. No. 4. P. 31–38. DOI: 10.31891/2219-9365-2024-80-5
6. Suprunenko I., Rudnytskyi V. Cloud based architecture for adaptive logging method implementation. Cybersecurity: Education, Science, Technique : Electron. Prof. Sci. J. 2025. Vol. 3, no. 27. P. 329–338. DOI: 10.28925/2663-4023.2025.27.724
7. Супруненко І. О. Адаптивний підхід до логування як новий вимір спостережності за прикладним програмним забезпеченням. Інформаційна безпека та комп'ютерні технології : тези доп. VII Міжнар. наук.-практ. конф. до 30-річчя кафедри кібербезпеки та програмного забезпечення, м. Кропивницький / М-во освіти і науки України, Центральнoукр. нац. техн. ун-т. Кропивницький : ЦНТУ, 2023. С. 45–46. URL: <https://kbpz.kntu.kr.ua/file/content/6621/2023-vii-mizhnarodna-naukovo-praktychnoi-konferentsiia-do-30-ty-richchia-kafedry-kiberbezpeky-ta-prohramnoho-zabezpechennia-informatsiina-bezpeka-ta-komp-yuterni-tekhnologii-.pdf>

8. Suprunenko I. Message passing systems in modern applications and their role in adaptive logging method. Інформаційні технології в освіті, науці й техніці (ІТОНТ-2024) : матеріали VII Міжнар. наук.-практ. конф. Черкаси, 2024. Р. 12–14. URL: https://knsa.chdtu.edu.ua/wp-content/uploads/2024/06/Conference-Proceedings-ITEST-2024_25_06.pdf

9. Suprunenko I., Rudnytskyi V. Dynamic source code processing approaches in context of adaptive logging method. Global Society in Formation of New Security System and World Order : матеріали III Міжнар. наук.-практ. інтернет- конф. Дніпро, 2024. С. 20–22. URL: <http://www.wayscience.com/wp-content/uploads/2024/07/Conference-Proceedings-July-4-5-2024.pdf>

ЗМІСТ

ВСТУП.....	14
РОЗДІЛ 1 АНАЛІЗ РОЗВИТКУ КОМП'ЮТЕРНИХ СИСТЕМ ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ СТВОРЕННЯ БЕЗПЕЧНИХ УМОВ ФУНКЦІОНУВАННЯ В ЦИФРОВОМУ ПРОСТОРИ.....	19
1.1 Розвиток обчислювальної техніки та програмного забезпечення, їх вплив на суспільство.....	19
1.2 Аналіз найбільш відомих кіберінцидентів та пов'язаних з ними загроз інформаційній безпеці: причини, масштаб та наслідки.....	22
1.3 Роль та місце кібербезпеки і відповідних програмних продуктів для створення більш безпечних умов існування в добу цифрових технологій.....	26
1.4 Мета і завдання дисертаційного дослідження.....	29
Висновки до розділу 1.....	30
РОЗДІЛ 2 МЕТОД АДАПТИВНОГО ЛОГУВАННЯ КОМП'ЮТЕРНИХ СИСТЕМ ТА ПРОГРАМ.....	31
2.1 Механізми забезпечення належного рівня спостережності в розподілених комп'ютерних системах на різних етапах розробки та функціонування програмного забезпечення.....	31
2.1.1 Застосування моніторингу для забезпечення належного рівня спостережності в комп'ютерних системах за заздалегідь визначеними характеристиками та метриками.....	34
2.1.2 Застосування логуювання для забезпечення належного рівня спостережності в комп'ютерних системах з налаштовуваним рівнем захищеності та деталізацією вихідних даних.....	35
2.2 Обґрунтування необхідності механізму адаптивності на різних етапах розробки та функціонування програмного забезпечення.....	37
2.3 Вимоги до архітектури та функціонування методу адаптивного логуювання.....	40
2.4 Формалізація базової функції логуювання та функції реініціалізації методу адаптивного логуювання.....	52

Висновки до розділу 2.....	56
РОЗДІЛ 3 МЕТОД АДАПТИВНОГО ЛОГУВАННЯ З ДИНАМІЧНИМ ВАРІАНТОМ ПОВІДОМЛЕНЬ.....	58
3.1 Обґрунтування доцільності використання механізму динамічних повідомлень при розробці комп'ютерних систем.....	58
3.2 Загальна структура механізму динамічних повідомлень та її порівняння з базовим варіантом, орієнтованим на текст.....	61
3.3 Формалізація модифікованих сигнатур методу адаптивного логування відповідно до архітектури динамічного варіанту повідомлень.....	65
3.4 Формалізація механізму роботи підсистеми валідації динамічних повідомлень ґрунтуючись на форматі JSON-schema.....	72
Висновки до розділу 3.....	83
РОЗДІЛ 4 ПРИКЛАДНА АРХІТЕКТУРА ФУНКЦІОНУВАННЯ МЕТОДУ АДАПТИВНОГО ЛОГУВАННЯ В ХМАРНИХ СЕРЕДОВИЩАХ.....	85
4.1 Опис необхідних складових для впровадження методу адаптивного логування в хмарному середовищі на прикладі веб-серверу.....	85
4.2 Стратегія імплементації методу адаптивного логування з використанням інфраструктури Amazon Web Services.....	89
4.3 Особливості програмної реалізації методу адаптивного логування.....	96
4.4 Особливості реалізації методу адаптивного логування з використанням хмарного провайдера Amazon Web Services.....	102
4.5 Приклад практичної переваги методу адаптивного логування над базовим підходом до логування.....	105
Висновки до розділу 4.....	108
ВИСНОВКИ.....	111
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	114
ДОДАТКИ.....	126

ВСТУП

Актуальність теми.

Комп'ютерні системи та програми є важливим компонентом сучасного світу. Переважна більшість населення планети щоденно в тому чи іншому вигляді користується результатами розробки програмного забезпечення, як от у випадку із кишеньковими пристроями для комунікації, настільними комп'ютерами для роботи чи вдома, в автомобілях, в громадських місцях, тощо. Для забезпечення величезної кількості споживачів належним рівнем інформаційних послуг, а також для підтримання задовільного рівня функціонування даних послуг, створюється все більше систем, а існуючі системи змушені масштабуватись. В свою чергу збільшення складності систем, кількості їх користувачів породжує нові виклики з точки зору інформаційної безпеки, які пов'язані з помилками в роботі, некоректностями роботи програмного забезпечення, виникнення загроз цілісності як для даних систем, так і для інформації всередині них. Одним із засобів вирішення цих проблем є постійний розвиток аспекту спостережності в реальному часі.

Значний внесок у дослідження аспекту спостережності в своїх роботах зробили науковці: R. E. Alsowaigh, R. Emsley, L. Ben-Shimol, E. Grolman, A. Elyashar, I. Maimon, P. Sambamurthy, B. Cantrill, F. Petrillo, S. Gholamian, C Majors, L. Fong-Jones, B. W. Weide, C. Fernando та ін.

Дослідження логування як частини аспекту спостережності можна знайти в працях W. Shang, P. L. Foalem, J. Bogatinovski та ін. Проте фокус подібних робіт зазвичай спрямований або на систематизацію існуючих методів, або на впровадження новітніх технологій (як от штучний інтелект чи машинне навчання) в процес роботи зі згенерованими даними.

Перспективним напрямком роботи з аспектом спостережності є дослідження процесу створення лог-записів саме в момент активної роботи системи. Такий підхід в свою чергу може потенційно дозволити зменшити кількість згенерованої інформації, зменшити непотрібне навантаження, а також знайти можливості для зміни фокусу підсистеми логування в процесі безпосередньої роботи.

Таким чином, можна стверджувати, що тема дисертаційного дослідження «Метод адаптивного логування комп'ютерних систем та програм» є актуальною.

Зв'язок роботи з науковими програмами, планами, темами.

Дисертаційна робота за своєю тематикою відповідає Закону України «Про перспективні напрямки наукових досліджень» від 2006 року зі змінами від 2024 року, підрозділу “Інформаційних та комунікаційних технологій”, а саме “Технологічні засоби та сервіси програмного інжинірингу”. Отримані результати дисертаційного дослідження використані при виконанні НДР в Черкаському державному технологічному університеті: «Інформаційна технологія психолінгвістичного аналізу тексту для стеганографічних систем» (ДР № 0123U102085), і «Дослідження шляхів розвитку потокового шифрування на основі криптографічного кодування» (ДР № 0121U114389), у яких автор брав участь як виконавець.

Мета і задачі дослідження. Основною метою дослідження є підвищення рівня спостережності та оперативності контролю роботи комп'ютерних систем та програм за рахунок розробки та впровадження методу адаптивного логування.

Для досягнення поставленої мети сформульовано і вирішено такі задачі:

1. Розробити моделі сигнатур адаптивного логування для забезпечення вищого рівня спостережності та можливості реініціалізації конфігурації.
2. Розробити метод адаптивного логування на основі моделей сигнатур та динамічного варіанту повідомлень.
3. Модифікувати метод адаптивного логування для реалізації з використання хмарної інфраструктури, що забезпечить можливість подальшого розвитку методів контролю роботи програмного забезпечення для хмарних обчислень.

Об'єкт дослідження – процеси контролю і логування.

Предмет дослідження – метод та засоби реалізації адаптивного логування в комп'ютерних системах та програмах.

Методи дослідження. У процесі розробки моделей сигнатур адаптивного логування використовувався аналітичний метод, узагальнення та формалізація,

що дозволило в подальшому вдосконалити сигнатури базової моделі логуювання, з орієнтацією лише на критичність, та створити можливості для реініціалізації процесів, що забезпечило підвищення рівня точності логуювання.

Для розробки методу адаптивного логуювання на основі моделей сигнатур та динамічного варіанту повідомлень використовувались методи формалізації та моделювання.

Модифікація методу адаптивного логуювання для реалізації з використання хмарної інфраструктури виконувалася з використанням методів моделювання та експерименту, в результаті чого було перевірено реальне застосування формальних основ методу в хмарному середовищі.

Наукова новизна одержаних результатів. У процесі вирішення поставлених задач автором одержано такі результати:

1) вперше побудовано моделі сигнатур адаптивного логуювання за рахунок формалізації і вдосконалення сигнатур базової моделі логуювання, що орієнтується лише на критичність, та створено можливості реініціалізації процесів, що забезпечило підвищення рівня спостережності;

2) вперше побудовано метод адаптивного логуювання на основі логуювання повідомлень шляхом застосування моделей сигнатур та динамічного варіанту повідомлень, що забезпечило підвищення рівня спостережності та оперативності сигналювання про непередбачувані та критичні ситуації в роботі систем та програм;

3) отримали подальший розвиток методи контролю роботи програм для хмарних обчислень за рахунок впровадження методу адаптивного логуювання з використанням хмарних сервісів.

Практичне значення отриманих результатів. Практична цінність роботи полягає в доведенні розробленого підходу, до алгоритмів, функціональних схем і методики застосування, що в сукупності дозволяють отримати вищий рівень деталізації та інформативності стосовно процесів, які відбуваються під час виконання програмного коду. Запропоновані автором для використання модулі програмного забезпечення, були підібрані таким чином, щоб мати можливості для імплементації в якомога більшій кількості платформ, мов програмування та середовищ виконання програмного коду. Для отриманого

за результатами дослідження варіанту методу адаптивного логування з динамічними повідомленнями розроблено ескізний зразок модуля впровадження на інфраструктурі провайдера хмарних послуг Amazon Web Services, із врахуванням аспектів належного рівня безпеки при взаємодії користувачів з адміністративним рівнем доступу, а також простоту та прогнозованість переносу артефактів готової системи між різними середовищами розробки.

За результатами обчислювального експерименту при порівнянні з базовим алгоритмом логування, що спирається лише на критичність, було отримано збільшення інформативності результуючих лог-записів на 31% у відповідності до виконання задачі відлагодження конкретного компоненту системи. Водночас окрема можливість вказати рівень критичності, вище якого застосування фільтрації є небажаним, дозволило зберегти всі повідомлення про помилки із решти компонентів системи, що не перебували у фокусі уваги. Порівнюючи два аналогічних сценарії реініціалізації було продемонстровано, що зменшення кількості надлишкових повідомлень у випадку адаптивного методу склало 61.3%, що на 0.7% більше, ніж для базового варіанту. Також втрата клієнтських повідомлень, що склала 14% при реініціалізації базового варіанту, була цілком відсутня у випадку з адаптивним методом, оскільки запропонована архітектура більш придатна для сценарію адаптації до змін у вимогах логування при активній роботі системи.

Результати дисертаційного дослідження, а саме метод адаптивного логування з динамічним варіантом повідомлень та підсистемою валідації, були використані для контролю коректності розробки програмного забезпечення системи дистанційного управління наземним самохідним роботизованим комплексом виробництва ТОВ “МОРОЗ ТЕХ”.

Особистий внесок здобувача. Всі нові результати дисертаційної роботи отримано автором самостійно. Роботи [7] і [8] виконані без співавторів. У наукових працях, опублікованих у співавторстві, з питань, що стосуються цього дослідження, автору належать: формалізація функції логування, що базується лише на рівні критичності, формалізація функцій логування та реініціалізації методу адаптивного логування, побудова конфігураційного словника [1, 2]; вибір

механізму міжпроцесових сигналів як засобу реініціалізації імплементації методу адаптивного логування [3]; розробка динамічного варіанту повідомлень [4]; розробка підсистеми валідації тіл динамічних функцій на основі схем [5]; моделювання архітектури розгортання методу адаптивного логування на хмарній інфраструктурі [6], аналіз алгоритмів обробки динамічного коду [9].

Апробація результатів дисертації. Результати дисертаційної роботи доповідалися й обговорювалися на VII Міжнародній науково-практичній конференції до 30-ти річчя кафедри кібербезпеки та програмного забезпечення (Кропивницький, 2023), VII Міжнародній науково-практичній конференції “Інформаційні технології в освіті, науці й техніці” ІТОНТ-2024 (Черкаси, 2024), III Міжнародній науково-практичній інтернет конференція “Global Society in Formation of New Security System and World Order” (Дніпро, 2024).

Публікації. Основні положення дисертації опубліковано у 9 друкованих працях, зокрема: у 5 статтях у фахових виданнях України, 1 колективній монографії та в матеріалах трьох міжнародних наукових конференцій.

Структура і обсяг дисертації. Дисертаційна робота складається зі вступу, чотирьох розділів, висновків, списку використаних джерел, і обов’язкового додатку. Загальний обсяг дисертації – 128 сторінок. Основний зміст викладений на 115 сторінках, у тому числі – 2 таблиці, 27 рисунків. Список використаних джерел містить 110 найменувань. Робота містить 2 додатки.

РОЗДІЛ 1 АНАЛІЗ РОЗВИТКУ КОМП'ЮТЕРНИХ СИСТЕМ ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ СТВОРЕННЯ БЕЗПЕЧНИХ УМОВ ФУНКЦІОНУВАННЯ В ЦИФРОВОМУ ПРОСТОРІ

1.1 Розвиток обчислювальної техніки та програмного забезпечення, їх вплив на суспільство

Комп'ютери, та програмні технології зокрема, є невід'ємною та значною частиною повсякденного життя. В історичному плані найперші спроби людської цивілізації автоматизувати та спростити задачі, пов'язані з обчисленням (*англ.* compute – “обчислювати”), були зроблені багато століть та тисячоліть тому. Так, для лічби 35 тисяч років до н.е. використовувались кістки із спеціальними засічками, тоді як 10 тисяч років до н.е. на Близькому Сході було вперше використано глиняні об'єкти для тієї ж самої задачі. Винайдення шестидесяткової системи числення, яку пізніше використовуватимуть у Вавилоні, Греції та Арабському світі, трапилось за 4 тисячі років до н.е., а вже через 1300 років перші рахівниці були винайдені в Шумері та Вавилоні. Деякі звичні речі, як от поняття про “нуль” досягали ріхних частинок планети з затримкою в століття, оскільки перше відоме застосування нуля датовано Вавилоном 300 років до н.е., а на території Китаю він з'явився аж у VIII столітті. Втім, попри значну нерівномірність того як саме термінологія та методологія, пов'язана із обчисленням, досягала різних куточків світу та як використовувалась, станом на сьогодні завдяки явищу глобалізації та всесвітньої мережі Інтернет обмін знаннями та інформацією відбувається швидше ніж коли-небудь раніше. Формування комп'ютерних технологій саме в тому вигляді, в якому ми знаємо їх сьогодні, значною мірою відбулося в XX столітті. Виклики Другої світової війни призвели до виникнення у 1943 році комп'ютера Colossus, призначеного для використання в зламуванні шифрів, в той час як в Гарварді збудований обчислювальний пристрій Mark I. В 1945 році для обчислення артилерійських балістичних таблиць в університеті Пенсильванії створюють

ENIAC (*англ.* Electronic Numerical Integrator and Computer), а напрацювання його головних інженерів Екхерта та Маучлі лягають в основу роботи математика Джона фон Неймана, який в результаті описує концепт комп'ютерної архітектури для програм, яка пізніше стає відомою як “архітектура фон Неймана”, що використовується в усіх сучасних комп'ютерах. Хоча перші комп'ютери є вузькоспеціалізовані та непридатні для використання широким загалом, створення Universal Automatic Computer (UNIVAC) Екхертом та Маучлі у 1951 році, а також вихід на комерційний ринок моделі IBM 650 у 1953 році, робить використання цієї технології все більш доступним. Поява таких моделей як Apple I у 1976 та Apple II у 1977 роках, а також комп'ютерної системи із відкритою архітектурою IBM PC у 1981 році (що, в свою чергу, дає поштовх для створення багатьох продуктів-клонів), ознаменувала остаточний перехід до широкої доступності комп'ютерів та програмних технологій для більш широкого кола споживачів. Врешті-решт, поява персональних комп'ютерів з графічним інтерфейсом, Apple Macintosh у 1984 році та Windows 1.0 у 1985, а також представлення Тімом Бернсом-Лі базових компонентів для функціонування World Wide Web у вигляді веб-сервера, веб клієнта, що використовують Hypertext Transfer Protocol (HTTP) та Hypertext Markup Language (HTML), створює такий ландшафт для комп'ютерних технологій, який ми можемо спостерігати сьогодні [10].

Згідно зі звітом Digital 2026 Global Overview Report [11], загальна кількість осіб, що користуються мережею Інтернет, становить 6.04 мільярди (73.2% від всього населення). Аудиторія соціальних мереж складає 5.66 мільярдів осіб, а 5.78 мільярдів людей використовують мобільні пристрої для доступу онлайн. Якщо порівняти із листопадом 2014 року, можна стверджувати, що кількість користувачів всесвітньої мережі подвоїлась менш ніж за 11 років, а в 1991 році користувачів було всього близько 5 мільйонів. Такий рівень проникнення в життя та суспільство очікувано впливає на взаємодію окремих людей та держави. Так, у Великій Британії спеціалізована система під назвою National Health Service digital тісно пов'язана із цифровою трансформацією медичних послуг, розробкою цифрових послуг, взаємодією із лікарями та технологічними

компаніями а також дотична до тестування штучного інтелекту в галузях патології, радіології та інших [12]. Технологічні досягнення в біотехнології змінюють галузі охорони здоров'я та медичних досліджень, що в результаті покращує діагностування хвороб, лікування та заходи для запобігання захворювань, в той час як розробки з корегування геному, персоналізована медицина та регенеративні терапії можуть допомогти продовжити тривалість життя та впоратись з комплексними проблемами стосовно людського здоров'я [13]. Процес навчання також зазнає змін пов'язаних із впровадженням цифрових підходів та засобів [14]: близько 81% студентів вищих навчальних закладів вважають, що цифрові інструменти для навчання (віртуальні класні кімнати та чати) покращують їх оцінки. Приблизно 96% викладачів схиляються до думки, що використання більш технологічних підходів при навчанні підвищують залученість студентів до процесу та результативність власне самого навчання, а зростання сегменту віртуального навчання становить приблизно 900% порівняно із 2000 роком.

Технологічний прогрес та впровадження великої кількості технологічних рішень в різні аспекти повсякденного життя призвів до виникнення нового напрямку під назвою “розумний дім” – напрям комп'ютерних технологій, що включає в себе використання різноманітних пристроїв з метою підвищення комфорту, безпеки, охорони та енергоефективності. Завдяки використанню систем з віддаленим керуванням (на основі телекомунікацій чи веб технологій) та спостереженням управління такими пристроями може здійснюватись навіть з спеціалізованих центрів підтримки [15]. І хоча такий підхід важко назвати новим, позаяк Р. Лютольф визначав його [16] як “поєднання різних сервісів всередині дому з використанням комунікаційних систем, завдяки якому досягається економічне, безпечне та безпечне функціонування з високим ступенем інтелектуальної функціональності та гнучкості” (визначення по суті створене під впливом саме технологій автоматизації та меншою мірою інтелектуальних технологій) ще в 20 сторіччі, саме досягнення останніх десятиліть дозволили наблизити використання даного концепту до реальності.

Варто також згадати доволі стрімкий прогрес в області штучного інтелекту: поява великих мовних моделей (*англ.* Large Language Models) привела до питання, чи можливо вважати їх певною формою свідомості. І хоча станом на сьогодні відповідь звучить як “ні”, швидкість їх розвитку, а також потенційна можливість створити для них відповідні сенсорні входи та запам’ятовувальні механізми, однозначність даної відповіді може бути під питанням [17]. Втім, вже зараз можна стверджувати, що ця технологія зайняла незаперечне місце у великій кількості галузей: науковці напряму фундаментальної фізики використовують штучний інтелект щоб збільшити точність вимірювальних інструментів та продуктивність експериментів; аналіз зображень для виокремлення певних характеристик та шаблонів з супутникових знімків за допомогою нейронних мереж дозволяє отримувати точніші географічні дані; в агрокультурній галузі з’являються нові можливості для діагностування захворювань у рослин, ефективного застосування хімікатів, збільшення врожайності з використанням фенотипування рослин.

1.2 Аналіз найбільш відомих кіберінцидентів та пов’язаних з ними загроз інформаційній безпеці: причини, масштаб та наслідки

Високий рівень впливу технологій та програмних засобів на життя людей як в приватному аспекті, так і в більш загальних, формує значні виклики. Доволі часто інформаційна інфраструктура та активи можуть стати ціллю для зловмисників чи постраждати від випадкових непередбачуваних ситуацій, наслідки яких вражають своїми масштабами. Так, одним із всесвітньо відомих інцидентів зловмисної атаки на державні інституції стала атака на National Health Service (NHS) в 2017 році під назвою WannaCry. Відповідно до дослідження було встановлено, що за результатами атаки було скасовано прийом 13500 пацієнтів, з яких щонайменше 139 були випадками для пацієнтів із можливим діагнозом “онкозахворювання” [18]. Загальні втрати впродовж часу, поки атака була активною, склали 19 мільйонів фунтів, а додаткові 73.5 мільйонів були витрачені згодом на відновлення функціонування систем та даних. Даний інцидент ще раз

підтвердив, що медична сфера є надзвичайно вразливою для кіберзлочинів. Сектор захисту галузі від кіберзагроз все ще залишається хронічно недоінвестованим. В результаті, для посилення аспекту цифрової безпеки NHS було інвестовано більше коштів з метою тимчасово підтримати систему, крім того було визнано необхідним паралельне збільшення бюджетів ІТ складової.

Ще одним помітним викликом для аспекту кібербезпеки стала пандемія COVID-19. Оскільки велика кількість людей були змушені змінити свій спосіб роботи на дистанційний, відповідно відбулись зміни в способах компрометації зі сторони зловмисників. Так у дослідженні [19] зазначається, що найчастішими типами атак в період з березня 2020 року по грудень 2021 були хакінг (17%), електронні листи-спам (13%), підроблені домени веб сайтів (9%) та скомпрометовані мобільні застосунки (8%). Також, за даними Identity Theft Resource Center, було відмічено різке зростання випадків компрометації: порівняно із 1175 випадками у 2018 році та 1108 у 2019, за 2020 та 2021 було зафіксовано 1872 та 1862 випадки відповідно. Навіть організації, пов'язані з охороною здоров'я, що спрямовували всі зусилля на боротьбу із світовою пандемією, самі ставали жертвами загроз інформаційних [20].

Швидке зростання впливу штучного інтелекту також має певний вплив на ландшафт кіберзлочинів: окрім використання цієї технології звичайними людьми, кіберзлочинці активно впроваджують нові способи та засоби для незаконного заволодіння даними, обману, тощо. Впровадження великих мовних моделей в повсякденне життя створило деякі нові види загроз: до вже існуючих атак соціальної інженерії додалися нові, орієнтовані на поширення неправдивих інструкцій стосовно можливості встановлення преміум моделей (тобто таких, що мають доступ до більшої кількості можливостей та довшу тривалість сесій, чи більшу кількість доступних запитань або тем, для користувача) з метою встановлення зловмисного програмного забезпечення на пристрій жертви чи отримання доступу до параметрів аутентифікації (*англ.* credentials); у випадку взаємодії безпосередньо із програмним забезпеченням моделі відмічено маніпуляції з інструкціями користувача для обходу заходів модерації контенту (наприклад заборону на створення чи пошук контенту на певні політичні,

соціальні, злочинні теми) та подальше використання системи для створення зловмисного чи оманливого контенту [21]. Існуючі механізми роботи штучного інтелекту дають зловмисникам нові можливості для пошуку жертв через використання підходу data mining (опрацювання великих об'ємів даних в пошуку інформації, закономірностей, тощо), а також для здійснення відомих типів атак, як от самозванство чи репутаційні атаки. В подальшому до жертв такого типу атак можуть застосовуватись так звані deepfakes, тобто змінені, високоякісні та реалістичні відео чи зображення, що використовуються для створення оманливих новин, фінансових схем чи для нанесення репутаційної шкоди у випадках пов'язаних із відомим особами, як от політики чи знаменитості [22].

Окремою проблемою використання великих мовних моделей є аспект так званих галюцинацій, тобто генерація відповідей, які видаються правдоподібними, втім не відповідають дійсності частково або повністю (інколи вони можуть навіть бути створеними із декількох реальних джерел інформації і представлені як джерело, використане для відповіді). Так у [23] описується процес пошуку інформації в галузі медицини з використанням моделі ChatGPT, на першому етапі якого відповіді алгоритму були допоміжними та глибокими, а уточнюючі питання стосовно загальної методології отримували доволі змістовні відповіді. Втім, після отримання дещо неочікуваної відповіді, при спробі дізнатись джерела даної інформації чат-бот видав декілька посилань, перше з яких стосувалось відомої публікації, яка не стосувалась вмісту отриманої відповіді, тоді як три інших було неможливо знайти за пошуком імені автора, назвою чи найменуванням журналу, а використання ідентифікаторів DOI вело на статті, що взагалі не стосувались теми. І такі проблеми не є одиничними, в даному дослідженні додатково наведено інформацію з інших робіт, відповідно до однієї з яких було досліджено, що із 178 посилань на джерела інформації згенерованих моделлю 28 не існувало взагалі, а за результатами аналізу іншої роботи серед 115 отриманих посилань лише 7% були точні та автентичні, а 47% та 46% були відповідно підробленими та автентичними, але неточними.

Високий рівень цифровізації, окрім очевидних переваг, також включає в себе приховані загрози, відповідно до яких компрометація нормальної роботи

інформаційних систем може виникнути внаслідок випадковості, а не навмисної атаки. Влітку 2024 року в роботі провайдера послуг кібербезпеки CrowdStrike стався збій, в результаті чого приблизно 8.5 мільйонів комп'ютерів з операційною системою Windows були тимчасово виведені з ладу, оскільки зачеплені пристрої переходили в режим “синього екрану смерті” (англ. Blue Screen of Death) [24]. Одним із основних програмних продуктів провайдера є так званий Falcon sensor, що встановлюється на клієнтські системи на рівні ядра та забезпечує сервіси для реагування на кібератаки, забезпечуючи нормальну роботу для користувачів: спостерігаючи за важливими системними діями, як от створення процесів та потоків, а також створення, модифікація та видалення файлів, дане програмне забезпечення може заблокувати підозрілі дії, що в результаті дає хороший рівень захисту від потенційних загроз. Втім, через організаційну проблему в процесі розробки та тестуванні продукту в один із готових для розповсюдження пакетів програми був доданий код, який невірно взаємодіяв з масивами в пам'яті та некоректно перевіряв межі елементів масиву, внаслідок чого в процесі виконання код сенсора старався адресувати неіснуючі елементи масиву. За результатами, глобальний вплив даного збою тривав лише декілька днів, однак економічний вплив від нього оцінюється в 5.4 мільярди доларів.

Доволі схожа ситуація, втім вже з явним зловмисним наміром, трапилась в 2020 році: завдяки використанню методів соціальної інженерії, кіберзлочинці зуміли від імені розробника компанії SolarWinds розмістити шкідливий програмний код в пакеті платформи Orion, який мав бути встановлений на комп'ютери жертв із наступним оновленням [25]. Після встановлення оновлення шкідливий код відкривав доступ зловмиснику до серверу жертви, після чого почав надсилати внутрішню інформацію, як от доменне ім'я Windows чи статуси та часові мітки безпекового продукту. Оскільки клієнти компанії SolarWinds включають в себе близько 400 компаній із списку Fortune 500, вплив даного інциденту важко переоцінити, позаяк його оцінювана тривалість до виявлення була кілька тижнів чи місяців, а орієнтовні оцінені збитки можуть сягати 100 мільярдів доларів.

1.3 Роль та місце кібербезпеки і відповідних програмних продуктів для створення більш безпечних умов існування в добу цифрових технологій

Популяризація комп'ютерних технологій та їх загальна доступність призвела як до виникнення все нових способів для зловмисників заволодіти даними жертви (мережеві хробаки, троянці, заплатки, використання недокументованих можливостей програм, тощо), так і до створення способів захисту. Одним із таких способів операційної системи Windows є Microsoft Defender Antivirus (попередньо відомий під назвою Windows Defender) [26]. Цей продукт є важливим компонентом захисту наступного рівня операційної системи Windows, що поєднує в собі можливості машинного навчання, аналітики великих даних, глибокого аналізу протидії загрозам та хмарної інфраструктури Microsoft для захисту пристроїв. До його функціоналу також входить виявлення аномалій, що дає додатковий рівень захисту від зловмисного програмного забезпечення, яке не підпадає під будь-який попередньо відомий шаблон поведінки, для чого такі події як створення процесів чи завантаження файлів з мережі Інтернет перебувають під його постійним контролем. В 2015 році з метою підвищення рівня захисту програмний рушій Defender-а змінив підхід із моделі, що базувалась на використанні статичних підписів, на технології, що здатні робити передбачення, як от машинне навчання та штучний інтелект. За даними дослідження [27], Windows Defender продемонстрував високі рівні виявлення для відомого шкідливого програмного забезпечення, втім дещо нижчі для вразливостей “нульового дня” (тобто таких, які існують в програмах без відома команд, відповідальних за розробку та підтримку), особливо в режимі офлайн. Також в режимі Endpoint Detection and Response було продемонстровано задовільний рівень протидії як відомим, так і вразливостям “нульового дня”. За необхідності додатково до можливостей Defender-а, що встановлений за замовчуванням, на операційну систему можна встановити й інші програмні продукти-антивіруси, такі як Avast, Avira, AVG, Eset Nod32, Panda, Malwarebytes [28].

Виклики, пов'язані з світовою епідемією коронавірусу COVID-19, призвели до того, що уряди багатьох держав вдавались до організації масових локдаунів, внаслідок чого роботодавці були вимушені переводити працівників на віддалений режим роботи та шукати способи продовжити функціонування бізнесу та організацій в нових умовах. Одним із варіантів такої роботи було використання технології Virtual Private Network (VPN), що встановлює зашифрований тунель між клієнтом (працівником, що виконує завдання з дому), провайдером послуг та цільовим ресурсом (сервер чи застосунок роботодавця, до якого необхідно мати захищений та контрольований доступ). В дослідженні [29] було проведено порівняння Інтернет трафіку з використанням відповідних сервісів приватних віртуальних мереж в перші місяці загальних локдаунів, базуючись як на окремих відомих портах, що використовуються конкретними протоколами (IPsec порт 500 чи 4500, OpenVPN порт 1194, L2TP порт 1701, PPTP порт 1723), так і на доменних іменах, які потенційно використовують VPN з'єднання. За результатами було відмічено зростання трафіку в робочі дні та години в березні 2020 року на більш ніж 200% порівняно з базовими замірами лютого того ж року. Важливо зазначити також, що зростання відмічене на вихідних є дещо меншим порівняно із робочими днями, а дані за квітень та травень (коли перші локдауни дещо ослабли) демонструють явне зменшення, що ще раз підкреслює, що основною причиною цього явища стали безпекові виклики пандемічного періоду.

Розвиток технологій штучного інтелекту дав поштовх не лише кіберзагрозам, а також різним напрямам інформаційної безпеки. Як вже було згадано раніше у випадку Windows Defender, перехід на рушій, що здатний адаптуватись до змін та вчитись в ландшафті загроз, на противагу попередньому підходу із статичними сигнатурами, був наступним кроком розвитку спроможностей програмного забезпечення. В роботі [30] описано особливу методологію впровадження під назвою OODA (*англ.* Observe, Orient, Decide, Act), що означає “Спостереження, Орієнтація, Рішення, Дія”, яку спеціалісти машинного навчання застосовують в контексті комерційних рішень з кібербезпеки. Перший етап характеризується спостереженням за різними типами

мережевого трафіку, обсяги якого достатньо значні, що не дає відповідній людині-спеціалісту опрацювати його за той же час, що це зробить алгоритм. Результатом опрацювання є виявлення аномальних патернів, що звуться “помилками”, та можуть включати в себе вимірювання кількості спроб аутентифікації, кількість запитів на програмний інтерфейс чи навіть біометричне розпізнавання. В межах другого кроку відбувається знаходження певних патернів, які здатні зрозуміти люди та алгоритм, що потенційно можуть потребувати дій для попередження загрози. В процесі кроку “Decide” розробники можуть імплементувати алгоритми, що будуть допомагати помічати аномальні патерни за мілісекунди та будуть постійно звітувати про свої результати через бази даних з інформацією про поточну ситуацію. І нарешті, протягом четвертого кроку написані алгоритми машинного навчання запускаються на короткі інтервали часу без втручання людини з метою відловити значну кількість індикаторів-помилки (в деяких випадках прогнозований рівень точності може сягати 99%), лишаючи людині ту частину помилок, яка пройшла непоміченою та може бути критичною для системи.

Високий рівень проникнення програмних продуктів в повсякденне життя, а також значний рівень вимог та очікувань користувачів змушують програмні системи швидко масштабуватись та зростати, щоб мати змогу задовольнити потреби десятків та сотень тисяч, а інколи мільйонів користувачів. Окрім типових міркувань, пов’язаних із захистом від кіберзагроз (як от щодо приватності комунікацій чи неможливістю спотворення вмісту інформації), додатковим аспектом, що потребує уваги, є спостережність. Відтак, в дослідженні [31] представлено особливо трирівневу структуру для опрацювання інформації про перебіг виконання операцій в системі, де в перших двох створюється онтологія активності програм з подальшим представленням її у вигляді уніфікованого “графу знань”, а в межах третьої зібрана інформація візуалізується та переформатовується в зручному представленні, що підвищує рівень спостережності (властивості системи, завдяки якій можливо фіксувати діяльність процесів та користувачів, використання ресурсів, а при виникненні інцидентів – однозначно ідентифікувати причетних користувачів та процеси з

метою забезпечення відповідальності за певні дії та унеможливлення майбутніх порушень політики безпеки) та ситуативної обізнаності. Відповідно до дослідження згаданого в [32], завдяки впровадженню технологій штучного інтелекту (Intent-Based Networking system) в сегмент мережевого менеджменту компанія Cisco змогла зменшити кількість відключень мереж на 74%, а також скоротити час відновлення на 56%.

Працюючи із аспектами спостережності важливим є не тільки процес опрацювання вже готових даних та метрик про роботу системи, а також генерація таких даних, покращений підхід до якої може зробити систему прозорішою та дати розробнику більше можливостей для пошуку несправностей та помилок.

1.4 Мета і завдання дисертаційного дослідження

На основі проведеного аналітичного огляду можливостей створення безпечних умов функціонування цифрового простору було встановлено, що отримати потрібний результат можна шляхом підвищення рівня спостережності та оперативності контролю функціонування комп'ютерних систем і програмного забезпечення. Сам процес підвищення рівня спостережності та оперативності контролю може базуватись на основі загальних практик ведення опису подій системи.

Враховуючи важливість етапу генерації метрик під час активної роботи системи, основною метою дисертаційного дослідження є підвищення рівня спостережності та оперативності контролю роботи комп'ютерних систем та програм за рахунок розробки та впровадження методу адаптивного логування.

Для досягнення поставленої мети необхідно вирішити наступні наукові завдання:

- розробити моделі сигнатур адаптивного логування;
- розробити метод адаптивного логування на основі моделей сигнатур та динамічного варіанту повідомлень;

- модифікувати метод адаптивного логування для реалізації з використанням хмарної інфраструктури, що забезпечить можливість подальшого розвитку методів контролю роботи програмного забезпечення для хмарних обчислень.

Висновки до розділу 1

В першому розділі розглядається актуальність наукового дослідження, визначається коло невирішених питань та на їх основі формулюється мета та завдання наукового дослідження.

1. Проведено аналіз розвитку і вдосконалення підходів до створення обчислювальних систем і програмного забезпечення. Виокремлено об'єктивні причини і наслідки історичного розвитку комп'ютерних систем та їх вплив на суспільство.
2. Проаналізовано причини, наслідки і масштаб деяких з найбільш відомих кіберінцидентів, та їх значення для розвитку інформаційної безпеки як однієї з умов існування цифрового простору.
3. Проаналізовано можливість створення безпечних умов функціонування цифрового простору з використанням досягнень галузі кібербезпеки та, зокрема, на основі підвищення рівня спостережності та оперативності контролю функціонування комп'ютерних систем і програмного забезпечення.
4. Сформульована мета і завдання дисертаційного дослідження.

РОЗДІЛ 2 МЕТОД АДАПТИВНОГО ЛОГУВАННЯ КОМП'ЮТЕРНИХ СИСТЕМ ТА ПРОГРАМ

2.1 Механізми забезпечення належного рівня спостережності в розподілених комп'ютерних системах на різних етапах розробки та функціонування програмного забезпечення

Розробка програмного забезпечення середніх та великих масштабів є складним та комплексним процесом. Успішна реалізація необхідного функціоналу на різних етапах розробки вимагає коректної організації процесу написання програм. Відповідно до поточного етапу циклу розробки вирізняють так звані “оточення” в яких відбувається написання, тестування чи відладка програмного продукту. Типовим підходом при розробці є використання 4х основних оточень: локальне (local) – знаходиться на персональному робочому комп'ютері розробника з повним доступом (в ідеальному варіанті) до всіх компонентів архітектури; дев (dev) – віддалене середовище, на якому відбувається попереднє розміщення та тестування функціоналу в умовах більш близьких до реальних; стейдж (stage) – віддалене оточення майже еквівалентне тому, на якому буде розгорнуто програмне забезпечення, готове до взаємодії з реальними користувачами, а тому представляє собою максимально близький до реальних умов тестовий майданчик для перевірки всієї системи; прод (prod) – власне фінальне середовище, на якому розміщений кінцевий продукт та наданий в повноцінне використання користувачам. При цьому важливе місце в функціонуванні кожного з них займає аспект спостережності, що дозволяє проводити відладку та пошук несправностей та неточностей в роботі програми.

На рисунку 2.1 зображена послідовність використання оточень розробки із демонстрацією життєвого циклу розроблюваних компонентів системи у відповідному середовищі та рух компоненту на наступні етапи, а також продемонстровано які класи користувачів очікувано мають змогу взаємодіяти із кожним з них.

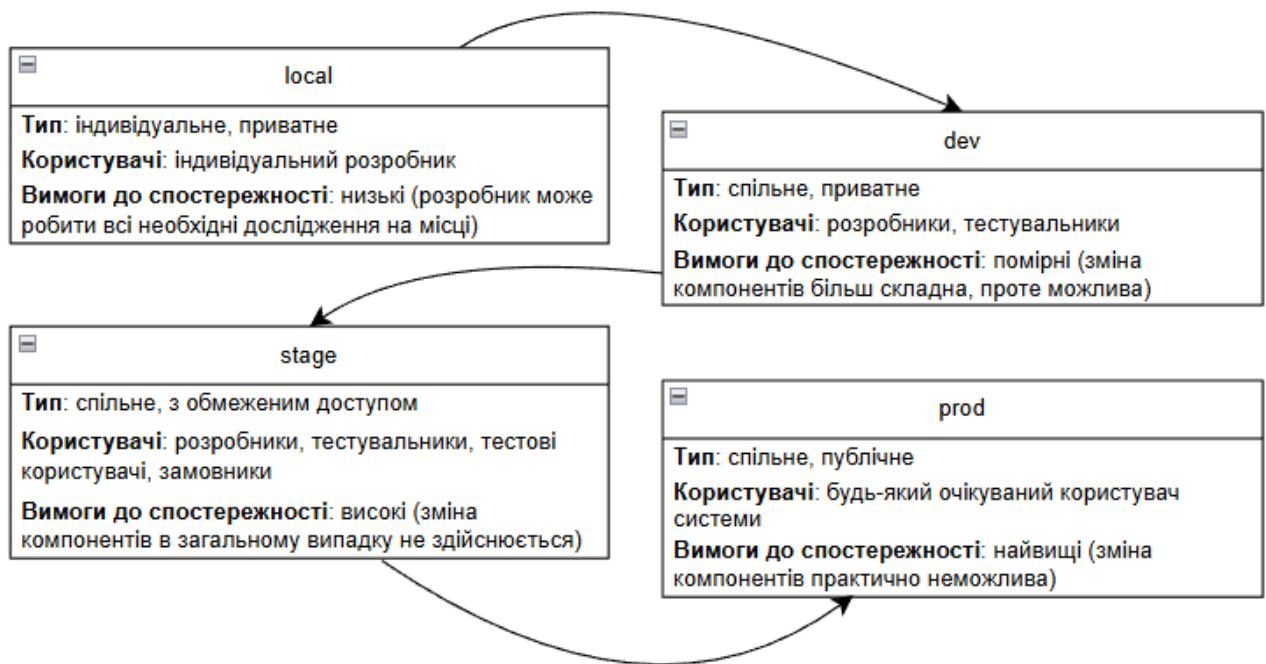


Рисунок 2.1 – Загальна схема взаємодії та функціонування середовищ розробки програмного забезпечення (результати отримані автором самостійно)

Направляючі стрілки, що поєднують схематичні пари, є очікуваним напрямом руху розроблюваного компоненту в процесі його реалізації. Важливо зазначити, що для кожного із середовищ характерний тип, що вказує в першу чергу на локалізацію вихідного коду та результуючих зібраних компонентів (які існують або на приватному комп'ютері розробника, або вже на віддалених спільних середовищах). Також важливим аспектом є рівень доступу:

- “приватний”: доступ мають виключно члени команди розробки програмного забезпечення, основний акцент уваги яких спрямований на початкові етапи розробки компонентів системи – власне створення компонентів та пошук в них найбільш критичних помилок чи невідповідностей очікуваному дизайну
- “з обмеженим доступом”: доступ мають безпосередньо учасники команди розробки, а також переглянути як система працюватиме при зверненнях від користувачів, близьких до реальних, мають змогу замовники та тестові користувачі
- “публічне” (для відповідних систем): доступ мають власне ті користувачі, для яких створюється система

З точки зору інформаційної безпеки, а також через складність та значні обсяги розроблюваних систем, для адекватної відповіді на сучасні виклики, окрім необхідності забезпечити три ключові аспекти кібербезпеки – цілісність, доступність та конфіденційність – важливо також приділяти достатню увагу аспекту спостережності, недостатній рівень якого може призвести до негативних наслідків. В грудні 1997го року в процесі розробки 64-процесорного мультипроцесора нового покоління, Sun Microsystems зіштовхнулись із доволі незвичною поведінкою, коли досліджуваний комп'ютер в певні проміжки часу починав виконувати незрозумілу роботу [33]. Корисне навантаження, яке демонстрували бенчмарки, було майже мінімальне, проте ядро виконувало якусь значну роботу. Після деякого часу система автовиправлялась та завершувала виконання тесту. І хоча у нового мультипроцесору були всі шанси поставити новий світовий рекорд, ці декілька втрачених хвилин могли бути вирішальними між першим та останнім місцем у змаганні.

Для дослідження даної проблеми були викликані найкращі інженери та розробники Sun Microsystems і робота йшла майже цілодобово в різних місцях земної кулі. Особливою проблемою також було те, що для дослідження системи зсередини необхідно було створювати спеціальні модулі ядра, впровадження яких для тестування певних гіпотез займало години. У випадку помилки, перезавантаження та переконфігурація займали порядку 90 хвилин. Через майже тиждень прискіпливих пошуків була виявлена проблема, що полягала в невірній конфігурації мережевого стеку комп'ютера, на якому виконувались тести: йому відводилась роль роутера. Зазвичай це не було би проблемою, проте якщо десь виходив з ладу сусідній роутер, тестовий комп'ютер починав підхоплювати його роль, зміщуючи свою активність на роботу з мережевим трафіком, що й зумовило виникнення епізодів з кількахвилинним просіданням продуктивності. Відсутність належного рівня спостережності розроблюваної системи вартувала багатьох днів роботи різних інженерів та могла бути фатальною для компанії.

В сучасних умовах ключовим є не лише якість вихідного продукту, а також швидкість та простота його розробки, неконфліктність внесення змін та коректив, а також простота пошуку проблем та їх усунення. Для досягнення

даної мети використовуються різні механізми забезпечення належного рівня спостережності, наприклад моніторинг та логування.

2.1.1 Застосування моніторингу для забезпечення належного рівня спостережності в комп'ютерних системах за заздалегідь визначеними характеристиками та метриками

Моніторингом називається процес збору даних та генерації звітів, що базуються на різних заздалегідь визначених метриках досліджуваної системи, щоб визначити її стан [34]. Практика моніторингу комп'ютерних систем існує вже доволі тривалий час, співмірний з часом існування таких систем. В ході цього процесу зібрані дані аналізуються з метою визначення чи система веде себе належним чином, а у випадку відхилень існують механізми повідомляти про помилки, аномалії та відмови. До прикладу, моніторинг доцільно використовувати для дослідження часу розгортання наступного релізу системи. Сформувавши певну вибірку із типових періодів скільки займали попередні релізи, кожен наступний можна перевіряти на відповідність та подавати попереджувальні сигнали, якщо спостерігається неочікуване відхилення.

Залежно від потреби, існують різні типи моніторингових програм, як от наприклад “диспетчер задач Windows” для персональних комп'ютерів [35]. З його допомогою користувач може бачити такі метрики як: поточні процеси операційної системи, їх стан, пріоритет, спожиті ресурси, тощо; метрики обладнання комп'ютера, в тому числі обсяг та утилізація оперативної пам'яті, використання та ядерність процесора, інформації про навантаження на мережу Інтернет чи WiFi, завантаженість постійних запам'ятовуючих пристроїв чи графічного процесора; споживання ресурсів конкретними програмами; перелік програм, що запускаються автоматично при запуску операційної системи; інформація про активних користувачів та активні служби, тощо. Висока інформативність та широке охоплення різних аспектів роботи системи дозволяє ефективно використовувати диспетчер задач для пошуку та виправлення несправностей.

Для більш ресурсоемних обчислювальних потужностей, таких як віртуальні чи фізичні сервери в хмарних середовищах, існують інші механізми для ведення моніторингу. Для прикладу в хмарній інфраструктурі Azure існує моніторингове рішення, що дозволяє збирати, аналізувати та реагувати на дані від хмарних (або навіть локальних) оточень, під назвою “Azure monitor” [36]. Його використання дозволяє максимізувати доступність та продуктивність застосунків та сервісів, а також дає більше розуміння стосовно деталей їх функціонування. Завдяки використанню платформи-орієнтованих метрик, згенерованих сервісами Azure, користувач має змогу отримати більшу видимість в питанні стабільності та продуктивності компонентів розгорнутої системи [37].

Доволі визначальною характеристикою моніторингу є заздалегідь визначена, можна навіть сказати типова, множина досліджуваних параметрів. Її використання дає змогу отримати більш загальну картину про стан роботи програмного забезпечення, проте при необхідності більшої деталізації доцільно використовувати інший механізм – логування.

2.1.2 Застосування логування для забезпечення належного рівня спостережності в комп'ютерних системах з налаштовуваним рівнем захищеності та деталізацією вихідних даних

Логуванням називається процес відслідковування подій, що трапляються в процесі використання програмного забезпечення [38]. Розробник програми додає в код спеціальні виклики логувальних функцій чи процедур щоб відслідкувати виникнення певної події. Зазвичай для опису такої події використовується описове повідомлення (з можливою інтерполяцією динамічних даних, тобто таких, що можуть змінюватись в різні моменти виконання). Лог-виклики також зазвичай містять рівень важливості або критичності (severity), що додатково описує виникнення події в процесі виконання програми та дає змогу відфільтровувати повідомлення відповідно до поточної задачі (проте дещо обмежено). Для позначення рівнів критичності використовуються впорядковані множини визначених констант, що дозволяє

відсортувати їх при необхідності. Так [39] визначає критичність як значення від 0 до 7 включно, де 7 це значення, що використовується для повідомлень відладки, а 0 має бути зарезервовано для найбільш критичних ситуацій, таких як відмова апаратних складових чи проблема з електроживленням. Іншим підходом є визначення текстових значень критичності, таких як “error” (зазвичай описується як серйозна проблема чи дуже серйозна помилка), “debug” (вказує на низько пріоритетні повідомлення, що використовуються для процесів відладки програмного забезпечення), “info” (типові для нормального функціонування системи повідомлення, що просто висвітлюють поточний стан виконання коду), “warn” (зазвичай значить підозрілу чи потенційно небезпечну ситуацію, зазвичай представляють відчутний інтерес для розробників), “fatal/critical” (позначають високо критичні помилки, які потребують негайних дій), “trace” (суміжний з рівнем “debug”, проте менш пріоритетний), “notice” (дещо схожий на рівень “info”), тощо [40].

Комбінація рівнів критичності та динамічна природа повідомлень дозволяє логуванню бути більш гнучким, аніж моніторинг, оскільки кожен аспект інформації, яку необхідно висвітлити, обирається розробником. Для ефективної роботи з цим способом забезпечення спостережності системи також є необхідним наявність механізму аналізу згенерованої інформації, її зберігання та видалення непотрібних чи застарілих файлів. Оскільки згенеровані файли можуть бути досить об’ємними та дещо розрізненими по своїй структурі, а вимоги при розробці системи можуть потребувати опрацювання логів в реальному часі, необхідною (проте доволі складною) є задача розробки ефективних та масштабованих засобів опрацювання згенерованої інформації [41]. Основна задача такого інструментарію полягає в тому, щоб спарсити (тобто зчитати та розібрати на змістовні частини) згенеровані файли, відслідкувати потенційні аномалії чи помилки, а також оцінити поведінку при нормальному виконанні програм. Складність даної задачі відрізняється в залежності від продукту, який потребує належного рівня спостережності, і тому рішення, як от мобільні системи чи застосунки для роботи з великими даними, вимагають більш ретельного підходу при роботі з логуванням. За своєю природою, чим довше

система перебуває у використанні, тим більше інформації вона генерує, а значить необхідно також передбачити що має відбуватись із непотрібними (застарілими) логами. До прикладу, для вирішення проблеми із стисненням, видалення або збереження в інших місцях накопичених лог-файлів системи Linux мають спеціальну утиліту під назвою `logrotate` [42]. Зазвичай вона виконується в рамках `cronjob`-и (тобто задачі, що відбувається в задані моменти часу протягом дня), проте за необхідності може працювати частіше (наприклад якщо розміри згенерованих файлів досягли певного розміру) чи одразу при передачі прапору командного рядка `--force`.

При належному підході до розміщення викликів функції логування, а також при використанні відповідних рівнів критичності, розробник отримує змогу більш гнучко спостерігати за процесом виконання програмного забезпечення та налаштовувати вивід (хоча й дещо обмежено) відповідно до своїх потреб. На відміну від моніторингу, додатково з'являється можливість самостійно визначати які саме характеристики процесу виконання (значення змінних, ідентифікатори процесів, файлових дескрипторів чи відкритих сокетів мережі Інтернет) існує необхідність відслідкувати та зафіксувати.

Проте один лише механізм вибору рівня критичності для виводу інформації про поточний процес виконання може не дати бажаної гнучкості, через що виникає потреба мати можливість адаптовуватись до вимог, що можуть змінюватись.

2.2 Обґрунтування необхідності механізму адаптивності на різних етапах розробки та функціонування програмного забезпечення

Для початку необхідно визначити що саме мається на увазі під “механізмом адаптивності”. В загальному це є комбінація програмних засобів, що дозволяє змінювати точність виведення інформації за допомогою лог-повідомлень. Проте, в такому ж ключі можна говорити й про стандартний механізм критичності. Типовим підходом до визначення бажаного рівня критичності є передача певної константи через змінні середовища [43] на тому

чи іншому середовищі виконання. Однак механізм інтерполяції (зчитування) таких змінних зазвичай працює на моменті запуску процесу і потребує перезапуску при модифікації. В межах даної роботи архітектура механізму адаптивності буде представлена з урахуванням необхідності не зупиняти поточний процес, однак все ж мати змогу змінити деталізацію та інформативність виводу.

Наступним важливим фактором, що вказує на необхідність існування даного механізму, є певні відмінності між згаданими середовищами виконання (local, dev, stage, prod). В локальному середовищі виконання коду розробник (в ідеалі) має повний контроль над усіма процесами, а тому може зупинитись і змінити вихідний код в будь-який момент. Тому справедливо стверджувати, що для локального середовища немає відчутної потреби в механізмі адаптивності. Однак для віддалених середовищ ситуація дещо інша: оскільки розміщення коду на віддаленому сервері зазвичай відбувається у вигляді так званих “артефактів” (тобто вихідних продуктів процесу лінування, компіляції, обфускації, мініфікації, тощо вихідного коду), побудова яких може займати доволі відчутний час, а розгортання (на відміну від локального середовища) може залежати від багатьох інших елементів, користь від здатності змінювати рівень деталізації лог-повідомлень куди більш значна. Втім, варто зазначити, що у випадку прод середовища, оскільки для нього досить важливі міркування безпеки та швидкодії, здатність швидко та ефективно отримувати деталізовану інформацію про стан системи перебуває в певному еквілібрії з міркуваннями забезпечення належного рівня закритості та стійкості системи. Тому в межах даної роботи доцільно вважати метод адаптивного логування як розробку, в першу чергу, для дев та стейдж середовищ. Так, для дев середовища, на якому цілком ймовірно розміщується не до кінця відтестований та перевірений програмний продукт, існує необхідність якомога швидше відловлювати помилки, які з якоїсь причини не були видимі локально. А для стейдж середовища, яке є максимальним наближенням до прод середовища, з дещо послабленими вимогами по безпеці та швидкодії, проте з куди більш обмеженим колом користувачів, рівновага між

безпекою та прозорістю системи може цілком бути зміщена в бік другого в процесі активної розробки функціоналу.

Наступним недоліком фільтрації логів виключно на базі рівнів критичності є те, що в досить складній системі може бути відчутна кількість модулів, що працюють одночасно та повідомляють деталі про процес свого виконання. Якщо несправність системи виявлена лише в певній частині програмного коду, увімкнення більш деталізованого рівня критичності автоматично активує його в усіх місцях. Відповідно, постає необхідність мати змогу більш точно вказувати на місце, чи групу, лог-повідомлень, інформація з яких нас цікавить на даному етапі. Типовим рішенням даної проблеми є використання механізмів аналізу (парсингу) згенерованої інформації (логів), проте це може відбуватись як з певною затримкою, так і взагалі постфактум в процесі перегляду доступних матеріалів для пошуку причини неполадок. Окрім всього іншого, велика кількість операцій введення/виведення може негативно впливати на швидкодію системи, тому бажане рішення виключало би сторонній непотрібний вплив при перемиканні конфігурації логування.

Останнім, але не менш проблематичним аспектом типового підходу, що базується на критичності, можна вважати ймовірну втрату активного стану виконання програми при зміні конфігурації. Як було згадано раніше, зазвичай налаштування механізмів логування відбувається через змінні середовища, що в свою чергу може вимагати перезапуску програми для застосування нових параметрів. Хоча цей процес може бути доволі швидким, більш важливим є те, що інформація, яка зберігається в оперативній пам'яті (змінні, екземпляри класів, інформація про відкриті сокети чи файли, тощо) може бути втрачена. Якщо помилковий стан виникає доволі рідко за неочевидних обставин, втрата активного стану програми може означати, що знайти першопричину просто не вдасться. Теоретично ймовірно можливо було би знайти рішення для цієї проблеми шляхом збереження інформації про активний стан системи перед її вимкненням та перезапуском, проте оскільки ця необхідність продиктована недосконалістю механізму, що не є критичним для безпосередньої логіки системи, а радше допоміжним щоб справлятися з помилками в процесі існування

коду, обмеження на необхідність внесення такого функціоналу в програму не можна вважати доцільним. Відповідно, бажане рішення мало би містити механізм зміни конфігураційних параметрів без необхідності перезапуску активний процес, що, в свою чергу, давало би можливість дослідити поточний помилковий стан в тому вигляді, в якому він був помічений, без необхідності намагатись його відтворити з нуля. Такий рівень гнучкості відіграє особливу роль для stage середовища, оскільки, як вже було згадано, воно максимально близько нагадує прод середовище, а значить може залежати від багатьох інфраструктурних елементів, зупинка та повторний запуск яких може займати певний час.

2.3 Вимоги до архітектури та функціонування методу адаптивного логування

Перш за все необхідно відзначити, що дизайн методу адаптивного логування передбачає його вбудовування безпосередньо в код тієї мови програмування, якою написана система. Це, в свою чергу, призводить до більшої гнучкості та позитивно впливає на швидкодію, аніж у випадку із стороннім сервісом чи процесом, проте також накладає певні обмеження: з метою зробити даний метод застосовним до якомога більшої кількості програмних рішень та систем, він має базуватись на примітивах, що з великою вірогідністю присутні в різних платформах, операційних системах, тощо. Так наприклад, якщо брати до уваги власне сам спосіб написання відповідного модуля, що імплементуватиме даний метод, вимогами до нього будуть: можливість використання в різних частинах системи та єдиність з точки зору зміну конфігурації - зміна параметрів (критичності) має бути одночасно відображена в усіх місцях, де використовується даний модуль. Окрім цього, необхідно мати можливість приховати певні деталі імплементации, в той же час роблячи інші публічними. Одним із можливих способів написання коду відповідно до таких вимог є використання патерну програмування “Синглтон” [44]. Відповідно до даного патерну в програмному коді може існувати виключно єдиний екземпляр класу, а

створення інших звичними методами, такими як використання конструктора, заборонено (конструктори позначаються модифікатором `private`) [45]. Для отримання чи створення інстанса синглтону існує спеціальний метод, що повністю приховує деталі імплементації, проте гарантує неможливість створення дублікату. Завдяки цьому можна бути впевненим, що будь-яка зміна проведена з екземпляром одночасно відбудеться в усіх місцях, які використовують даний клас, а це в свою чергу є важливою характеристикою, необхідною для створення імплементації методу адаптивного логування.

Також важливим є прикладний аспект логування, що стосується безпосередньо процесу виведення інформації про стан виконання програми. Типова взаємодія з комп'ютерними системами в цьому плані зазвичай передбачає взаємодію з стандартними потоками введення/виведення, роботу з локальною чи віддаленою файловими системами, мережевими сокетамі, тощо. Якщо для системи також важливий мінімальний вплив механізму логування на основний процес роботи, на особливу увагу заслуговують питання асинхронного (тобто неблокуючого) виведення та буферизації запитів (як на рівні програми, так і на рівні операційної системи, якщо необхідно). Надійна, перевірена та ефективна імплементація згаданих вимог є доволі трудомісткою задачею, що суттєво ускладнило б впровадження методу адаптивного логування в програмний продукт. Відтак більш доцільно виглядає альтернативна архітектура: для вирішення задачі роботи з безпосереднім введенням/виведенням краще за все взяти існуюче бібліотечне рішення, а за допомогою патерну програмування Фасад [46-47], поєднаного із вже згаданим Синглтоном, створити обгортку над бібліотечним функціоналом, розмістивши реалізацію необхідних функцій у відповідні методи екземпляру та приховавши деталі взаємодії з відносно низькорівневим програмним інтерфейсом бібліотеки логування.

Для досягнення належного рівня адаптивності необхідно додатково змінити загальну схему, за якою відбувається логування. Оскільки основною задачею адаптивного підходу є можливість адаптуватись до змін у вимогах стосовно логування, що в свою чергу означає включення або виключення певних

груп чи класів лог-повідомлень, необхідно мати змогу групувати та фільтрувати згенеровані записи за певною ознакою. Типовий підхід до логування визначає критичність та сам вміст повідомлення як дві необхідних складових, проте на основі лише них немає можливості досягти належного рівня адаптивності. Можливим вирішенням даної задачі було би впровадження єдиної форми для рядків лог-повідомлень, згідно з якою можна було би проводити групування, однак оскільки це безпосередньо впливатиме на вміст інформації про перебіг подій в системі, така зміна не може бути бажаною. Окрім цього, важливо передбачити можливість для повідомлення бути згрупованим по-різному в залежності від поточних вимог. Більш зручним та відповідним способом “маркування” є розширити перелік необхідних компонентів з пари до трьох, а саме: вміст повідомлення, рівень критичності та теги – список (без чітко визначеного порядку) рядків, що характеризують поточний виклик. Таким чином, введення третього аргументу дозволяє залишити без змін довільний формат тексту в повідомленнях, а також додає стандартизований підхід, що дозволяє групувати та відфільтровувати небажані виклики.

Втім, існують певні рівні критичності, для яких, відповідно до їх основного призначення, фільтрування може бути небажаним. Це, зокрема, такі рівні як “fatal/critical” або “error”, оскільки при їх виникненні в реальній системі необхідно якомога швидше прийняти відповідні дії. Рішення могло би представляти із себе використання особливого рядкового літералу-тегу, наприклад “*” (за аналогією із цим символом та його значенням при використанні в регулярних виразах для пошуку по тексті, а саме – пошук будь-якої кількості співпадінь відповідного символу чи групи [48], що при використанні у списку тегів означало би “цей виклик співпадає з будь-якою конфігурацією обраних тегів”. Проте, це вирішувало би проблему для середовища стейдж, де знання про серйозні помилки є більш значущим, ніж при розробці на дев середовищі, але в той самий час додавало би потенційно небажану інформацію в оточенні дев. Додатково до цього кожен такий виклик стає по суті “защитим” в ході процесів компіляції та збірки, і тому у вже зібраній системі завжди буде представлений і виведений без можливості це змінити.

Більш гнучким рішенням є введення додаткового елементу налаштувань, що дозволяв би вказати рівень критичності, згідно з яким всі повідомлення, що мають рівний чи вищий власний рівень критичності відносно вказаного, взагалі не потрапляють в алгоритм фільтрування. Додатково до більшої гнучкості цей спосіб має перевагу в тому, що налаштування відбувається без втручання у вихідний код програмного забезпечення, що також зменшує поверхню для помилки.

Налаштування бажаного рівня деталізації логування має здійснюватися за допомогою спеціального конфігураційного механізму. Основними вимогами до нього є здатність задати явне включення (тобто операцію фільтрування, що працюватиме за логікою “залишити лише ті виклики методу логування, для яких існує співпадіння між значеннями, вказаними в конфігурації, та значеннями, що характеризують виклик”) та явне виключення (логіка цієї операції радше представлятиме із себе “залишити такі виклики методу логування, у яких немає співпадінь із відповідними значеннями тегів з конфігурації”). Додатково до даних високорівневих типів операцій, необхідно мати змогу також застосовувати певні оператори комбінування конфігураційних тегів. До прикладу, доцільно мати змогу сказати, що згідно з поточними вимогами необхідно виводити інформацію про стан системи лише в межах тих повідомлень, що містять задану комбінацію тегів (тобто кожен виклик, якому дозволено вивести інформацію, має містити всі вказані елементи з конфігурації, щось на кшталт логічного оператора “and”), а також додатково до цього оператор, що дасть змогу вказати дозволені варіації елементів, з’єднаних попереднім оператором (тобто ті виклики, що мають співпадіння по будь-якому операнду даної операції, мають бути виведені, як в логічному “or”). Для зручності перший оператор називатиметься “and”, з відповідними сегментами, що називаються “and сегменти”, а другий – “or”, з відповідними сегментами, що називаються “or сегменти”. Представлена в такому вигляді конфігурація може бути відносно просто застосована в багатьох мовах програмування та платформах, оскільки її досить легко записати в текстовому форматі, так як значення тегів є просто

рядками, а операції можуть бути представлені довільними операторами, адже важлива лише суть їх впливу на операнди.

Додавання необхідних можливостей для адаптивності базується на конфігураційному механізмі, однак потребує впровадження спеціального засобу для взаємодії – механізму реініціалізації. Завдяки ньому відповідний користувач програмного забезпечення отримує змогу змінювати обрану комбінацію тегів та їх операторів з метою більш точного налаштування. Оскільки однією із важливих характеристик методу адаптивного логування є здатність підлаштовуватись під поточні вимоги щодо інформативності опису внутрішніх процесів системи не вимагаючи зупинки та перезапуску, а також зберігаючи поточні значення змінних, об'єктів, екземплярів класів, реініціалізація має відбуватись в межах системи, що перебуває в активному стані. Це, в свою чергу, обумовлює необхідність мати відповідний засіб для зворотного зв'язку з програмним забезпеченням (засіб пропагації змін) [3,8].

Типовим стороннім рішенням задачі обміну повідомленнями є використання сервісу на кшталт меседж-брокера як от RabbitMQ [49]. Меседж-брокером називають спеціалізований сервіс, що отримує та перенаправляє повідомлення за принципом, схожим до принципу роботи поштового відділення. В даному процесі є три основні складові частини (рис. 2.2): відправник (або ж “producer”), тобто такий учасник системи, що відправляє певну інформацію; черга, що представляє із себе величезний буфер для повідомлень; отримувач (або ж “consumer”), учасник системи, що опрацьовує повідомлення, отримані з черги від відправника.

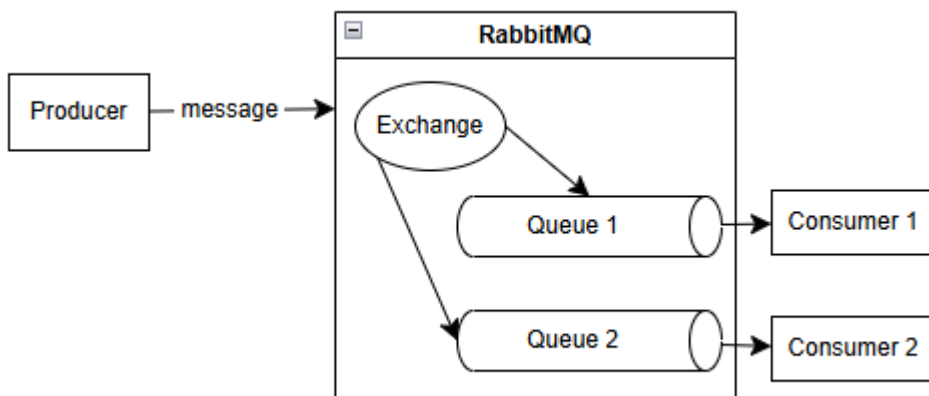


Рисунок 2.2 – Загальна схема роботи RabbitMQ (відповідно до [49])

У випадку методу адаптивного логування, відправником буде користувач з правами на реконфігурацію процесу логування, а споживачем – програмний код всередині активного процесу з імплементацією методу адаптивного логування, що по отриманню повідомлення запустить процес реініціалізації. Оскільки повідомлення в RabbitMQ можуть відносно довільну форму, нова конфігурація може бути відправлена у вигляді їх вмісту. В загальному вигляді такий підхід вирішує задачу реініціалізації, проте має свої недоліки. Перш за все, використання вже готового рішення, доволі гнучкого та надійного, накладає обмеження на системи які зможуть цим скористатись. Також, оскільки програмне забезпечення сервера меседж-брокера існуватиме за межами поточної системи, підвищується поверхня для помилки та вимоги до впровадження спеціалізованого коду роботи з чергами. Відповідно дане рішення навряд можна вважати бажаним в межах методу адаптивного логування.

Дещо менший вплив матиме альтернативна архітектура, що базується на можливостях системи управління базами даних PostgreSQL. При створенні підключення до серверу, клієнт має змогу скористатись командою “listen” [50], яка реєструє поточну сесію в якості слухача певного каналу, назва якого передається аргументом команди. В тому разі, якщо інша сесія з’єднання з базою даних розміщує повідомлення в даному каналі за допомогою команди “notify”, всі слухачі даного каналу отримають відповідну нотифікацію (рис. 2.3). На додачу до самого факту відправки повідомлення, “notify” також приймає другий параметр, який є рядком, та може бути використаний для передачі відповідної

інформації. За необхідності передані дані можуть бути додатково структуровані як використовуючи формат даних, що можна перетворити в рядок без втрати вмісту, так і використовуючи таблиці та інші елементи бази даних.

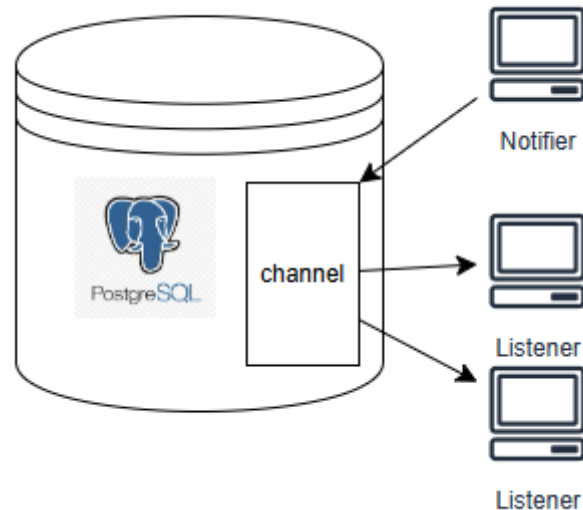


Рисунок 2.3 – Принцип роботи механізму Listen/Notify СУБД PostgreSQL (відповідно до [50])

В межах застосування для механізму реконфігурації методу адаптивного логування користувач з відповідними правами мав би приєднатись до серверу системи управління базою даних та передати повідомлення про зміну вимог до логування, розмістивши їх або в текстовому параметрі методу “notify”, або відповідно змінивши вміст інших частин бази даних. Порівняно з використанням меседж-брокера, такий підхід впливає на програмну систему меншою мірою, особливо якщо PostgreSQL вже використовується для роботи з базами даних, а також вимагає менше налаштувань для початку обміну повідомленнями. Втім, все ж розглядати цей підхід як основний в межах базової архітектури методу адаптивного логування ймовірно не є найбільш підходящим рішенням.

Завдяки використанню клієнт-серверної архітектури у веб-серверах, можливим вирішення даної задачі також є використання власне веб-сервера [51]. Програмний продукт з імплементацією методу адаптивного логування стає сервером, що очікує на специфічний запит від клієнта – користувача з правом змінювати конфігурацію для логування – та, отримавши відповідний пакет

даних, викликатиме процедуру реініціалізації. Подібно до рішення з меседж-брокером та з каналами комунікації системи управління базами даних PostgreSQL, наступний стан конфігураційного механізму цілком можливо передати як інформативну частину HTTP пакету. Важливо зазначити, що відповідно до безпекових міркувань та природи задачі логування видається логічним заборонити даному веб-серверу слухати на публічних інтерфейсах, потенційно доступних в мережі Інтернет, і навпаки слухати лише loopback інтерфейс localhost. Це, в свою чергу, дещо звужує можливості доступу для виклику методу реініціалізації, проте таке рішення все ще має місце, оскільки допоміжна природа аспекту спостережності комп'ютерних систем спонукає спиратись на рішення, що вимагають найменшої кількості змін та не призводять до небажаного зниження рівня безпеки. Порівняно з RabbitMQ чи PostgreSQL, такий підхід має більш широкі перспективи, оскільки існування програмного забезпечення по типу веб-сервера є доволі звичною практикою для багатьох різних середовищ, платформ та мов програмування, що позитивно впливає на загальні можливості імплементації методу адаптивного логування в таких системах (особливо для продуктів, що вже збудовані навколо задачі обслуговування веб запитів). Однак, навіть з такими перевагами й перспективами, даний підхід видається дещо надмірним, а отже необхідність знайти щось більш відповідне залишається актуальною.

При створенні програм, що функціонують в Unix-подібних системах, одним із базових способів для обміну повідомлення може слугувати Unix domain socket [52] (або local socket в Android-орієнтованих продуктах [53]) – канал для обміну даними між процесами, що виконуються в межах єдиної операційної системи (стандартний компонент POSIX [54] операційних систем). Його програмний інтерфейс дещо схожий з програмний інтерфейсом сокетів мережі Інтернет (рис. 2.4), проте замість використання мережевого протоколу вся комунікація здійснюється всередині ядра операційної системи.

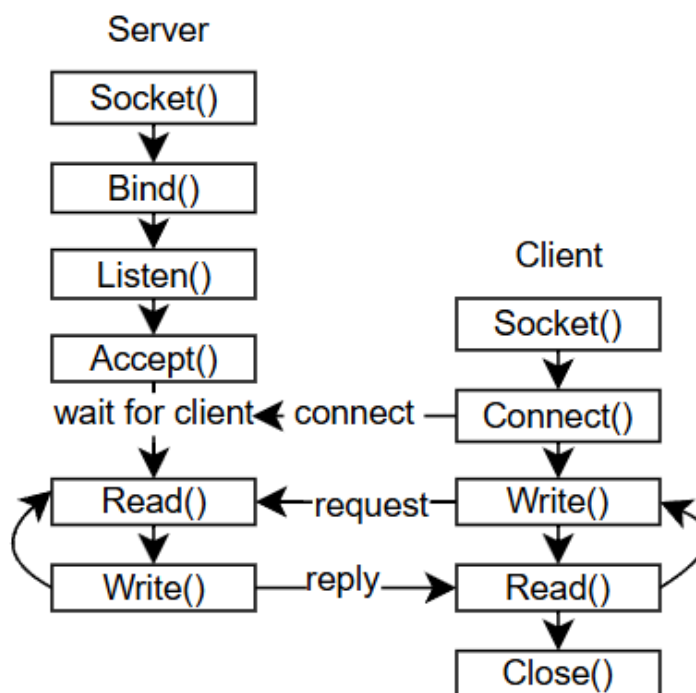


Рисунок 2.4 – Загальна схема функціонування Unix-сокетів (відповідно до [52])

Існують три типових простори імен, асоційованих з Unix domain sockets: `FILESYSTEM` (пов'язаний з файлом на файловій системі), `RESERVED` (по суті є підсистемою `FILESYSTEM`) та `ABSTRACT`, що існує незалежно від `FILESYSTEM`. Відповідно, якщо імплементація методу адаптивного логування ініціалізує сокет-сервер прив'язаний до певного файлу, сокет-клієнт, ініціалізований користувачем з правом змінювати конфігурацію механізму логування на цьому ж самому файлі, матиме змогу передати бажане повідомлення (вірогідно з новою конфігурацією тегів та рівня критичності). У порівнянні з попередніми варіантами рішення проблеми комунікації, Unix domain sockets представляють більш легке рішення, оскільки не вимагають встановлення сторонніх систем чи залежностей (за рахунок того, що є базовою функціональністю операційної системи) та одночасно зменшують ймовірність внесення нових небезпек, адже функціонують виключно в межах одного комп'ютера. Втім, якщо брати до уваги інші доступні механізми в Unix орієнтованих системах, існує рішення, що не потребуватиме додавання нових абстракцій до майже будь якого програмного продукту.

Ще одним базовим механізмом програмного забезпечення в операційних системах сімейства Unix є міжпроцесові сигнали [55]. Сигналом називається програмна нотифікація процесу про виникнення певної події, та генерується, коли виникає відповідна подія. Сигнал вважається доставленим, коли процес, якому він був відправлений, якось його опрацює (що може трапитись через певний відчутний проміжок часу після того, як сигнал був згенерований). Втім важливо відмітити, що опрацюванням не вважається виконання функцій стандартної відповіді чи стандартної помилки, а значить відповідальність в опрацюванні бажаного сигналу повністю лежить на розробникові.

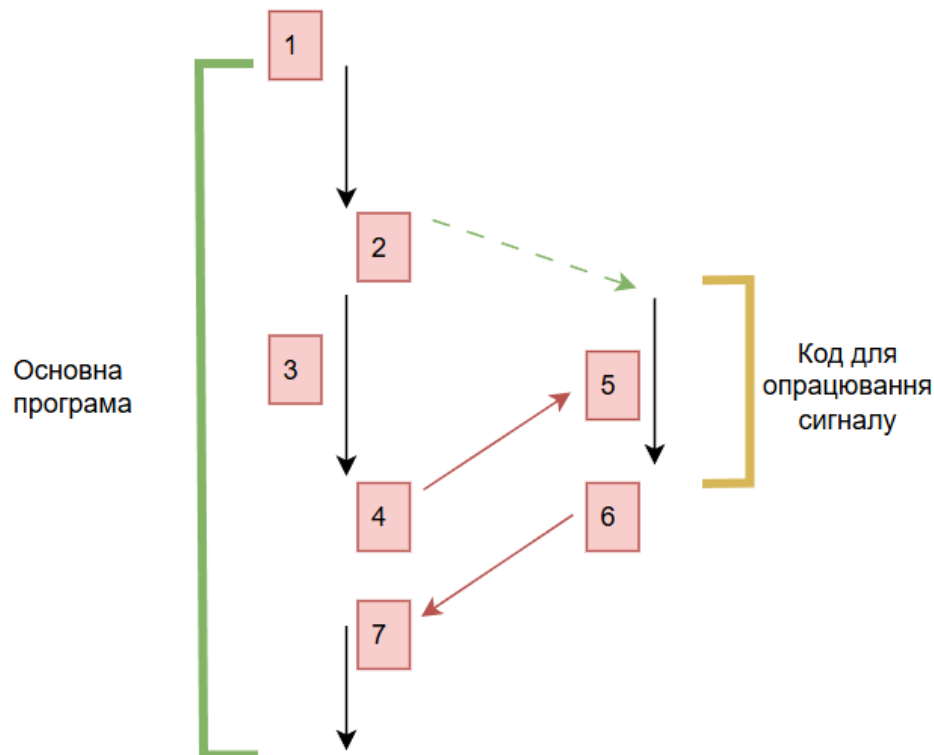


Рисунок 2.5 – Загальний механізм роботи міжпроцесових сигналів UNIX
(відповідно до [55])

Додавання та робота відповідного обробника міжпроцесового сигналу відбувається згідно з наступним алгоритмом:

1. Розпочинається виконання основної частини програми
2. Встановлюється відповідний обробник очікуваного міжпроцесового сигналу
3. Продовження виконання основної частини програми

4. Отримання повідомлення про виникнення сигналу
5. Початок виконання коду обробника
6. Завершення виконання коду обробника та повернення до виконання основної частини
7. Продовження виконання програми

Специфікація можливих значень сигналів визначає два особливих значення SIGUSR1 та SIGUSR2, призначених для міжпроцесової взаємодії [56], і які відповідно можуть бути використані для необхідних взаємодій з імплементацією методу адаптивного логування. Існує ключова відмінність від попередніх варіантів, яка полягає в тому, що сигнали не мають аргументу для передачі метаданих чи будь-якої додаткової інформації, як це було можливо з повідомленням меседж брокера чи HTTP пакетом веб-сервера. А тому для розміщення нової версії конфігурації необхідно використовувати додатковий метод. Доволі природним рішенням може слугувати використання конфігураційного файлу, вміст якого зчитує функція реініціалізації та виставляє відповідні фільтри. Відповідно, користувач системи для зміни поточного рівня критичності та застосованих тегів має змінити вміст файлу та відправити відповідний сигнал, який буде опрацьовано імплементацією методу адаптивного логування. Даний підхід є найменш вимогливим із всіх описаних, оскільки із значною вірогідністю присутній на великій кількості платформ та середовищ, а також містить найменшу поверхню для помилки та небезпеки, оскільки спирається на базову функціональність міжпроцесорної взаємодії операційної системи.

Останнім важливим аспектом, який необхідно згадати описуючи основні вимоги до методу адаптивного логування, є питання авторизації користувачів. Оскільки функціонал логування є, по суті, допоміжним, необхідно максимально убезпечити систему від небажаних наслідків, якщо злоумисник отримає доступ до керування механізмом реініціалізації (що може призвести як до втрати інформації про перебіг подій в цілому, і дасть злочинцю здатність робити будь-які дії, інформація про які ніде не збережеться, так і, наприклад, до можливого перевантаження системи через активацію занадто детального режиму

логування). Оскільки питання коректної, вивіреної та перевіреної системи авторизації користувачів є доволі складним програмним рішенням (аналогічно до безпосередньої імплементації низькорівневого механізму логуювання), включення вимоги по написанню такого функціоналу в межах імплементації методу адаптивного логуювання відчутно ускладнило б базові вимоги. Окрім цього, навіть якщо ресурси на реалізацію такого рішення існують, його використання все ще буде під питанням, оскільки існує значна кількість реалізованих та перевірених рішень в екосистемі відкритого коду. Відповідно, авторизація користувачів також цілком логічно виглядає як аспект, де доцільно використати існуюче рішення, що найкраще підходить до обраного механізму реініціалізації. У випадку з RabbitMQ авторизація здійснюється безпосередньо самим меседж-брокером [57], для доступу до каналів PostgreSQL необхідно авторизуватись використовуючи доступні методи аутентифікації на сервері бази даних [58], а у випадку з веб-сервером авторизація могла бути імплементована на рівні HTTP заголовків. Проте, як було показано, ці методи не зовсім підходять для базового опису методу адаптивного логуювання, що залишає в підсумку або domain sockets, або міжпроцесові сигнали. Оскільки обидва є реалізацією програмного функціоналу на рівні операційної системи, відповідне розмежування доступу в них вирішується в цілому спільним чином. Для віддалених середовищ досить типовим є використання протоколу Secure Shell [59], що являє собою протокол для безпечного віддаленого логіну, або реалізація інших безпечних сервісів, використовуючи мережу, яка може не бути безпечною. У випадку використання більш спеціалізованих рішень для розгортання комп'ютерних систем в хмарі, провайдери хмарних послуг зазвичай мають альтернативні способи авторизації та підключення до ресурсів користувача. У випадку з локальними системами звичні механізми авторизації та розділення доступу, як от використання користувачів операційних систем сімейства Unix, являють собою відносно задовільне рішення, оскільки компрометація доступу на цьому рівні свідчить про проблеми куди більші, ніж можливий негативний вплив на конфігурацію логуювання та систему, що логується. Отже, загалом можна стверджувати, що через складність реалізації, а також в умовах наявності

широкої вибірки існуючих рішень задачі авторизації при доступі як до локальних так і віддалених систем, найбільш доцільним рішенням для базової імплементації методу адаптивного логування є делегування авторизації на програмну чи апаратну інфраструктуру.

2.4 Формалізація базової функції логування та функції реініціалізації методу адаптивного логування

Відповідно до наведених вимог, наступним кроком є формалізація необхідних публічних функції інстансу Синглтона імплементації методу адаптивного логування [1,2,7]. Стандартний вигляд функції логування, що базується лише на критичності, можна представити наступним чином:

$$f_{log} = f(Sev, M), \quad (2.1)$$

де Sev – обраний рівень критичності, що характеризує поточний виклик, M – текстовий вміст лог-повідомлення. Відповідно до заданого бажаного рівня критичності, програмний код всередині даної функції приймає рішення стосовно того, чи матиме поточний виклик якісь видимі результати в підсистему введення/виведення.

Сигнатура методу, що використовується безпосередньо для виклику механізму логування (2.2), виглядає наступним чином:

$$f_{log\ adp} = f(Sev, M, T_{incl}), \quad (2.2)$$

де параметри Sev та M аналогічні тим, що представлені в формулі (2.1), а новий параметр T_{incl} – масив рядкових значень, що представляють собою теги для опису даного виклику.

Важливо акцентувати, що згідно з вимогою про використання існуючого рішення для безпосереднього здійснення логування, сигнатура не потребує параметрів, пов'язаних з деталізацією процесів взаємодії з системою введення/виведення чи взаємодії з мережею (при наявності конфігурації віддаленого зберігання лог-файлів). Це спрощує саму функцію, а також залишає

простір для розробників обирати саме те рішення, яке необхідне для конкретного проекту.

Як було згадано раніше, друга фундаментальна складова імплементації методу адаптивного логування представлена у вигляді механізму для реініціалізації поточної конфігурації логування. Відповідна функція інстансу Синглтона (2.3) має наступний вигляд:

$$f_{init} = f(Sev, Sev_{ign}, C), \quad (2.3)$$

де Sev – поточний обраний рівень критичності на рівні всієї системи, Sev_{ign} – рівень критичності, що визначає, що повідомлення з даним рівнем та вище не підпадають під механізм фільтрування за тегами, C – конфігураційний словник (2.4), що декларує поточні комбінації тегів для фільтрування.

$$C = T_{11}^{mod} \wedge T_{12}^{mod} \wedge \dots || T_{21}^{mod} \wedge T_{22}^{mod} \wedge \dots || \dots, \quad (2.4)$$

де T_{ij} – рядковий літерал, що являє собою ім'я тегу, mod – модифікатор, що застосований до поточного тегу (може бути “включити” чи “виключити”), \wedge - оператор поєднання тегів з функціоналом подібним до логічного “and”, $||$ - оператор поєднання “and”-сегментів з функціоналом подібним до логічного “or”.

Важливо також зазначити, що параметр Sev_{ign} не є обов'язковим, а радше представляє собою допоміжне налаштування, покликане виокремити певні виклики, для яких небажано застосовувати фільтрацію (в силу важливості чи небезпеки відповідного рівня критичності). В такому разі допускається використання null [60] значення, чи його аналогів в конкретній платформі чи мові програмування, як індикатор того, що будь-який виклик методу логування необхідно пропустити через процес фільтрації.

Такий вигляд функції реініціалізації з одного боку не накладає ніяких обмежень на конкретні оператори та символи, які має використати розробник при написанні власної імплементації, а з іншого – цілком задовольняє обране рішення для обміну повідомлень через сигнали процесів, оскільки всі елементи конфігурації можливо подати в серіалізованому [61] вигляді, а значить - записати

у конфігураційний файл та зчитати його, отримавши відповідний міжпроцесовий сигнал.

Схематичне зображення механізму виконання виклику логування в імплементації методу адаптивного логування виглядає наступним чином:

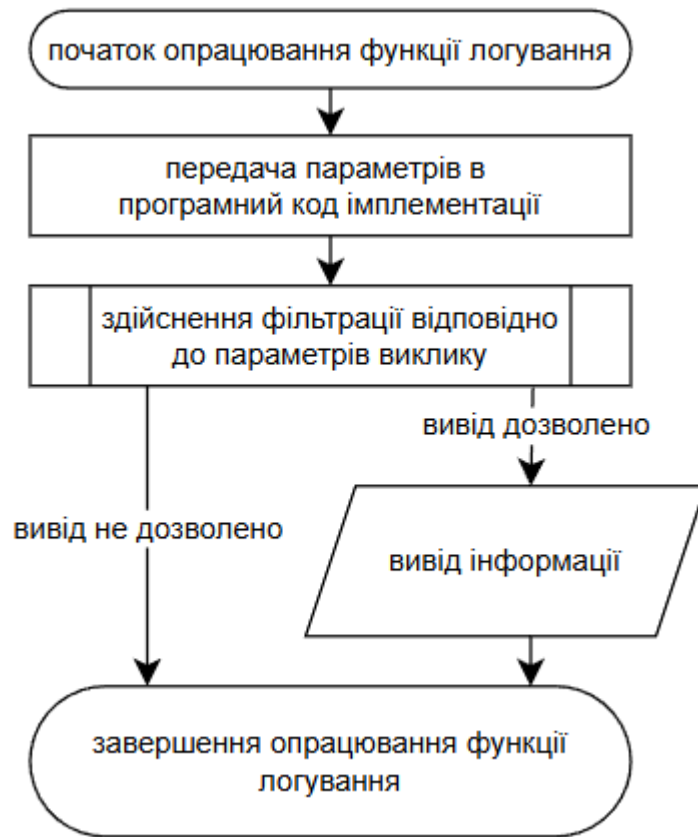


Рисунок 2.6 – Схема процесу виконання лог виклику методу адаптивного логування (результати отримані автором самостійно)

Процес здійснення фільтрації відповідно до параметрів виклику також є кілька ступеневим алгоритмом. При опрацюванні відповідного виклику спершу відбувається перевірка наявності параметру конфігурації Sev_{ign} , та якщо він присутній і рівень критичності поточного лог виклику є вищим чи рівним до нього – вивід повідомлення має відбутись незалежно від конфігурації “and” та “or” сегментів чи тегів, якими промаркований даний виклик. Інакше необхідно перевірити, чи є явне виключення хоча б одного із вказаних тегів лог-виклику, і якщо так – вивід не відбувається. Якщо в конфігураційному словнику S відсутні будь-які сегменти з явним включенням – вивід дозволяється. Втім, якщо це не так – вивід дозволено лише тоді, якщо існує щонайменше один “or” сегмент, всі

“and” елементи якого містяться в T_{incl} (оскільки ми вже пересвідчилися, що явного виключення немає). Схематичне зображення даного алгоритму представлено на рис 2.7:

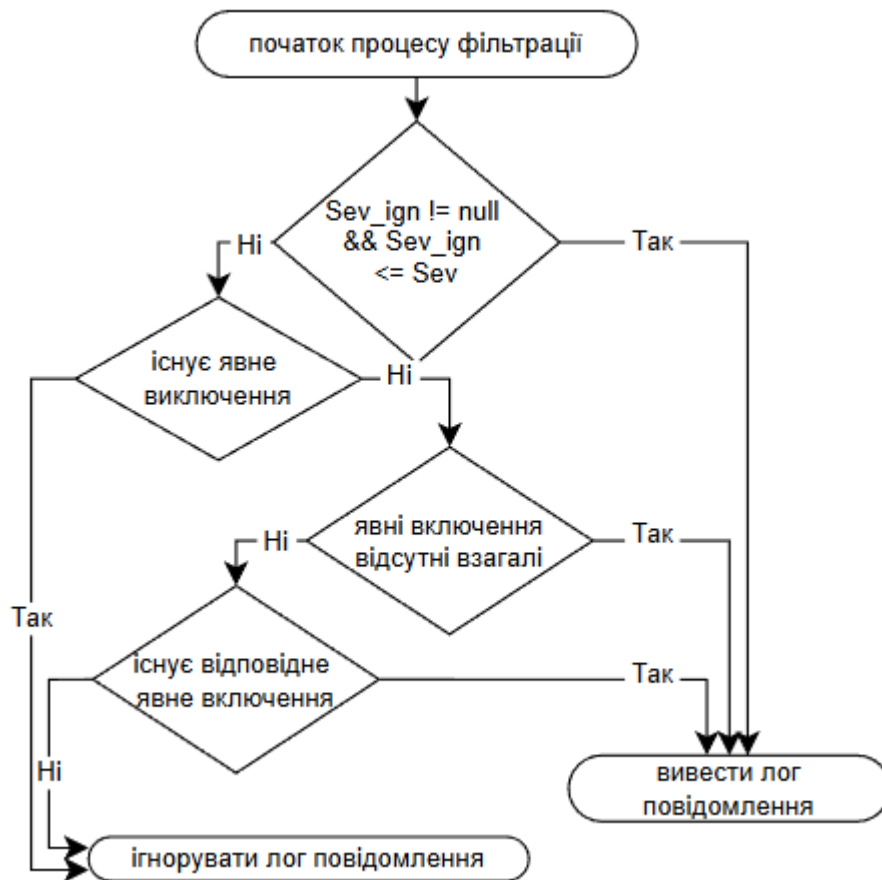


Рисунок 2.7 – алгоритм фільтрації лог виклику методу адаптивного логування (результати отримані автором самостійно)

Згідно з вимогою використання стороннього механізму авторизації користувачів, обидва методи формалізовані без урахування авторизаційних параметрів, таких як ролі чи дозволи відповідних користувачів. На поточному рівні формалізації доцільно вважати, що якщо користувач отримав доступ до обох описаних методів, всі необхідні кроки для його аутентифікації були виконані засобами програмної чи апаратної інфраструктури. Як і з стороннім низькорівневим механізмом логування, це накладає менше обмежень на імплементацію, та лишає ширші можливості для створення бажаного рішення (адже для деяких систем достатнім буде рівень авторизації користувачів як-от HTTP Basic Auth [62], тоді як для інших вимоги можуть бути достатньо

серйозними, щоб розглядати використання механізмів по типу Azure Active Directory [63]). Цілком очікуваним можна вважати варіант, при якому всі деталі імплементації механізму аутентифікації винесені на рівень нижче (операційна система, мережеве з'єднання, тощо), що залишає фокус при розробці та впровадженні методу адаптивного логування безпосередньо на покращенні якості контролю та рівня спостережності.

Висновки до розділу 2

За результатами розділу було вперше представлено формальну основу методу адаптивного логування інформаційних систем, що дозволяє забезпечити вищий рівень спостережності та налаштовуваності порівняно з типовим підходом до логування, що базується лише на критичності лог повідомлень, з наступними етапами:

- розглянуто типовий перелік середовищ розробки при створенні комп'ютерних систем, з їх відповідним очікуваним переліком користувачів, вимогами по рівню спостережності та можливістю до зміни використовуваних компонентів;
- досліджено особливості сучасних підходів до забезпечення необхідного рівня спостережності в інформаційних системах на основі практики моніторингу за заздалегідь визначеними метриками та очікуваними допустимими значеннями, а також практики логування із можливістю визначати та обирати необхідні аспекти роботи програм та їх деталізацію при відображенні;
- обґрунтовано необхідність та доцільність використання механізму адаптивного логування в середовищах dev та stage, представлені основні міркування стосовно задач, які має вирішувати імплементація з метод забезпечення більш високого рівня спостережності;
- були визначені та сформульовані основні вимоги до архітектури та функціонування механізму адаптивного логування, для виконання вимог, що стосуються єдиності існування програмного об'єкту з

необхідним функціоналом, було обрано патерн програмування Синглтон, а для приховання низькорівневих деталей прикладної реалізації вводу/виводу рекомендовано взяти патерн програмування Фасад; розглянуто декілька альтернативних реалізацій способу передачі повідомлень до активної програми та було обрано механізм міжпроцесової взаємодії як найбільш гнучкий та масштабований в контексті різних мов програмування та платформ виконання коду;

- формалізовано базові функції (логування та реініціалізації) з детальним вказанням їх складових та схематичним зображенням принципів їх роботи, описано спосіб задання та взаємодії “or” та “and” сегментів конфігураційного об’єкту при визначенні необхідних дій відповідно до перевірки списку лог-тегів для конкретного місця виклику лог-функції;
- матеріали розділу опубліковані в [1], [2], [3], [7] та [8].

РОЗДІЛ 3 МЕТОД АДАПТИВНОГО ЛОГУВАННЯ З ДИНАМІЧНИМ ВАРІАНТОМ ПОВІДОМЛЕНЬ

3.1 Обґрунтування доцільності використання механізму динамічних повідомлень при розробці комп'ютерних систем

Використання методу адаптивного логування в його поточному вигляді вже дає здатність більш гнучко підходити до питання пошуку проблем при розробці комп'ютерних систем: в залежності від середовища та даних, які в ньому опрацьовуються, розробник з належним рівнем доступу має змогу змінити деталізацію логування та акцентувати саме на тих частинах системи, де необхідний вищий рівень спостережності. Втім, найвищий ступінь результативності даного функціоналу досягається лише тоді, коли інформаційна система, потоки даних всередині неї, а також типові сценарії для її роботи вже були сформовані та опрацьовані, тобто перебувають або на завершальних етапах тестування в середовищі stage, або вже доступні реальним користувачам в середовищі prod. При повноцінному використанні системи стає максимально зрозуміло які є “вузькі місця” у відповідних компонентах, що потребують високого рівня спостережності, також стає зрозуміло які саме дані необхідно виокремлювати (до прикладу ідентифікатори процесів, запитів, інформацію про користувача, тощо), а що можна оминати, оскільки воно буде збережено на запам'ятовуючих пристроях, з можливістю переглянути пізніше. Типовим підходом для опису нормального процесу виконання програми є використання лог-викликів із рівнем критичності info, що (як було згадано раніше) означає “типові для нормального функціонування системи повідомлення, що просто висвітлюють поточний стан виконання коду”. До прикладу, при опрацюванні запиту до веб-сервера (рис. 3.1) доволі типовим є крок, при якому ідентифікуючі параметри веб-запиту (час звернення, інтернет адреса клієнта, шлях ресурсу за яким було звернення, відповідний метод для здійснення звернення, тощо) записуються в лог.

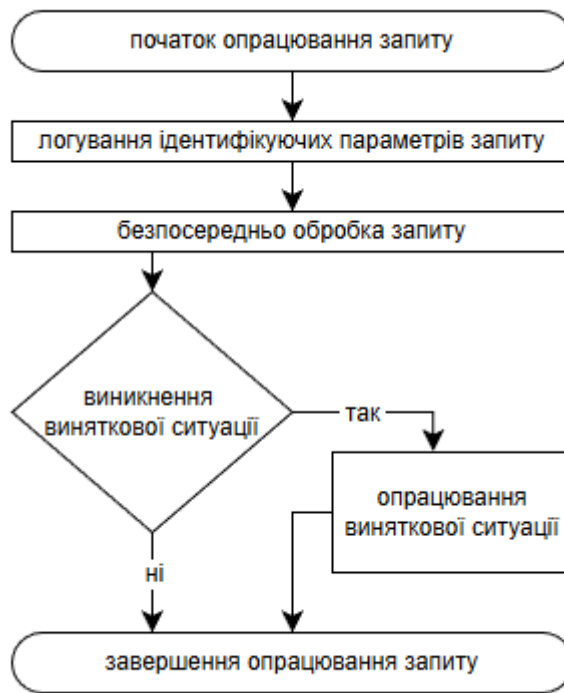


Рисунок 3.1 – Типовий сценарій опрацювання запиту до веб-сервера (результати отримані автором самостійно)

Опціонально, якщо є необхідність, додаткове логування рівня “info” може відбуватись і в кроці, де відбувається безпосередня обробка запиту. В такому випадку мова йтиме про довільну форму опису процесу виконання. У випадку ж із загальними ідентифікаторами для загально використовуваних рішень, як от реверс-проксі веб-сервер Nginx [64] існує доволі визначений та очікуваний формат, що включає в себе змінні \$remote_addr (IP адреса клієнта), \$time_local (дата здійснення запиту в локалізованому форматі із вказанням часового поясу), \$request (рядковий опис HTTP запиту, включно з методом запиту, шляхом до запитуваного ресурсу та деталями стосовно використаної версії HTTP), \$status (статус відповіді веб-сервера, де 2xx та 3xx вважаються успішними, а решта - помилками), \$body_bytes_sent (розмір тіла відповіді, з урахуванням стиснення) та \$http_user_agent (рядковий опис клієнта, зазвичай веб чи мобільного браузера, використаного для здійснення даного запиту). На рис. 3.2 зображений приклад логу для серверу Nginx.

```

2025/12/28 11:52:38 [notice] 1#1: start worker process 35
2025/12/28 11:52:38 [notice] 1#1: start worker process 36
172.17.0.1 - - [28/Dec/2025:11:52:44 +0000] "GET / HTTP/1.1" 200 9 "-" "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/131.0.0.0 Safari/537.36" "-"
172.17.0.1 - - [28/Dec/2025:11:52:50 +0000] "GET /about-us HTTP/1.1" 200 13 "-" "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/131.0.0.0 Safari/537.36" "-"
172.17.0.1 - - [28/Dec/2025:11:52:53 +0000] "GET /services HTTP/1.1" 200 13 "-" "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/131.0.0.0 Safari/537.36" "-"
172.17.0.1 - - [28/Dec/2025:11:52:57 +0000] "GET /contacts HTTP/1.1" 200 13 "-" "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/131.0.0.0 Safari/537.36" "-"
172.17.0.1 - - [28/Dec/2025:11:53:00 +0000] "GET /donate HTTP/1.1" 200 11 "-" "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/131.0.0.0 Safari/537.36" "-"
2025/12/28 11:53:03 [error] 29#29: *2 open() "/etc/nginx/html/members" failed (2: No such file or directory), client: 172.17.0.1, server: , request: "GET /members HTTP/1.1", host: "localhost:9123"
172.17.0.1 - - [28/Dec/2025:11:53:03 +0000] "GET /members HTTP/1.1" 404 555 "-" "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/131.0.0.0 Safari/537.36" "-"

```

Рисунок 3.2 – Термінал із лог-записами запущеного веб-серверу Nginx (результати отримані автором самостійно)

Проте, для середовищ dev та stage, в яких використання методу адаптивного логування є найбільш доцільним, такий рівень продуманості, вичерпності та інформативності може бути ще в процесі налаштування чи написання. В процесі ж відладки та, особливо, пошуку причин, що призводять до невірного функціонування, доволі часто неможливо відносно просто перемкнутись на необхідні локації лог повідомлень та побачити всю доцільну інформацію про проблему (що особливо відчутно для середовища dev, де відбувається перше реальне випробування написаного на віддалених потужностях). З огляду на можливу складність системи та її складових, пошук причин некоректної роботи може потребувати дослідження компонентів виконуваного коду, важливість яких могла бути неочевидною при написання поточної версії програмного забезпечення. При роботі, до прикладу, в контексті веб-сервера для повноцінної відладки процесу з'єднання клієнта із сервером розробник може потребувати доступу до різних частин як високорівневого HTTP-запиту – у вигляді тіла запиту для перевірки його на очікувану та коректну форму, чи заголовків для дослідження таких аспектів як коректне кодування, кешування, авторизація, підписи для запитів, що підтверджують автентичність пакету, тощо – так і внутрішніх аспектів роботи використовуваного веб-сервера, що може включати в себе дослідження екземплярів об'єкту з'єднання чи відповідних сесій. Аналогічно при роботі із базами даних, окрім типових параметрів, таких як ім'я користувача який використовується для авторизації, віддалена адреса бази даних чи порт, на якому очікується з'єднання, важливим

може бути дослідження максимального часу життя з'єднання, кількості перевикористовуваних з'єднань у випадку роботи через пул [65], деталі роботи SSL шифрування (сертифікат та деталі, що стосуються Certificate Authority, файл сертифікату, що використовується базою даних, приватний SSL ключ та його характеристики, права доступу, тощо), а у випадку виникнення помилок – додаткова інформація, наприклад чи пов'язаний факт виникнення непередбачуваних ситуації безпосередньо із мережевим з'єднанням, характеристиками та обмеженнями апаратного забезпечення сервера бази даних чи клієнта, логічними правилами, встановленими на рівні обмежень бази даних [66] або зв'язками по типу Foreign Key [67]. В свою чергу, для успішної відладки помилок бази даних доцільно мати інформацію про назву таблиці, колонок чи зв'язків, стосовно яких виникла проблема, а також про реальні значення даних, які призвели до некоректної роботи.

Відповідно до даної необхідності, стандартний підхід із заданням заздалегідь визначеного лог-повідомлення із відомим форматом та можливістю динамічно розміщувати параметри у визначених місцях, що будуть заповнюватись значеннями конкретних змінних вже в процесі виконання, потребує доопрацювання. В результаті цього доопрацювання розробник має отримати можливість, не змінюючи вихідний код програмного забезпечення, досліджувати різні властивості, змінні, екземпляри класів, тощо, що й буде досягатись за допомогою механізму динамічних повідомлень. Найбільша користь від оновленого підходу передбачається для середовища dev, а в результаті дослідження та відладки з його використанням більш змістовні та сформовані лог-повідомлення будуть з'являтись в середовищі stage та нарешті prod. Окремо має бути зазначено, що існування можливості задавати повідомлення із динамічним вмістом в середовищі prod є не виправдано небезпечним підходом і тому ця спроможність має бути явно вимкнена.

3.2 Загальна структура механізму динамічних повідомлень та її порівняння з базовим варіантом, орієнтованим на текст

Сигнатура для базового методу логування описується формулою (2.2).

Відповідно до неї, повідомлення *M* представлено рядком, який виводиться (логується) “як є”. Цього цілком достатньо, наприклад, для статичного опису місця виклику функції логування та навіть для використання змінних значень, що доступні в місці даного виклику (передати значення яких можна за рахунок приведення їх до рядкових чи співставних з такими типами даних), щоб дослідити їх вміст. І це дає ширші можливості стосовно відстеження деталей поведінки програмного продукту. Однак при цьому структура лишається незмінною, а для віддалених середовищ типу *dev* та *stage* ще й “закрита” необхідністю перезбирати результуючі артефакти. Для приклада можна знову звернутись до механізму логування веб-запитів до веб-сервера: якщо першочергова схема була розрахована на виведення лише деяких HTTP заголовків (припустімо *User-Agent* [68] для здійснення певних обчислень відповідно до того клієнту, яким було здійснено запит, або *Content-Type* [69] для відслідковування статистичної інформації стосовно вмісту певних запитів), для виведення чогось іншого необхідно змінити код додавши до нього звернення до іншого хедеру. А так як базовий метод адаптивного логування здатний лише вивести динамічне значення для конкретної написаної структури – це потребуватиме зміни в вихідному коді та перезапуск процесу розміщення результуючих компонентів для відповідного середовища. Звісно, теоретично також можливий варіант продовжувати користуватись все тим самим механізмом статично визначених інтерполяцій у викликах базового методу логування та передавати якомога більше деталей про процес виконання, детально описуючи велику кількість елементів виконуваної програми, проте це загалом є протилежною ідеологією порівняно з тією, для якої створювались базові засади методу: механізм логування має адаптовуватись під можливі зміни у вимогах до деталізації виводу, а не впливати на процес написання коду через власну недосконалість. З технічної точки зору таке рішення теж викликає певні питання, оскільки в загальному випадку для довільної структури процесу виконання неможливо сказати, які самі елементи можливо вивести в читабельному форматі, так як для цього необхідно мати розуміння щонайменше

щодо публічних деталі контракту даної структури [70-71], що не завжди є можливим при написанні програмного рішення, і більш доцільно було би мати змогу інспектувати реальні значення, з якими працює процес безпосередньо в ході роботи. Відповідно виникає необхідність змінити (чи швидше розширити) структуру повідомлення *M* щоб врахувати такі можливості. Варто зазначити, що подальший опис розширення для методу адаптивного логування накладає дещо більше обмежень на базову архітектуру, оскільки можливості, необхідні для реалізації механізму динамічних повідомлень вимагають спеціальних технологій.

Деякі високорівневі мови програмування дозволяють генерувати вихідний код та виконувати його вже в процесі виконання початкового вихідного коду. На цьому моменті важливо згадати також те, що загальноприйняте ставлення до механізмів програмного коду, що дозволяють виконання динамічної логіки, є радше негативним, втім в межах задачі адаптивного логування, а також беручи до уваги явне обмеження на ті оточення, де цей варіант повідомлень буде мати сенс та буде застосовуватись (включно з його явною заборонаю в *prod* середовищі), ця спроможність все ще може представляти певну користь. Для інтерпретованої об'єктно-орієнтованої мови програмування високого рівня із суворою динамічною типізацією Python [72] механізмом для виконання динамічного коду є функція “*exec*” [73], що працює за рахунок передачі бажаного коду в рядковому представленні першим аргументом та опціональними аргументами-словниками глобальних та локальних змінних, що будуть використані при виконанні коду. У випадку із динамічною об'єктно-орієнтованою прототипною скриптовою мовою програмування з динамічною типізацією та частковою підтримкою імперативної та функціональної парадигм Javascript [74] відповідним підходом може бути створення функцій за допомогою використання конструктора “*Function()*” [75]. Сигнатура його використання вказує на те, що по аналогії з “*exec*” ми також передаємо рядкове представлення тіла функції, проте робота з доступними аргументами дещо інша. Насамперед важливо те, створення таких функцій відбувається виключно в глобальному скоупі (себто середовищі виконуваного коду разом із доступними змінними), а

всі необхідні параметри мають передаватись через аргументи конструктора. В результаті розробник отримує в користування значення-функцію, яку має змогу викликати передавши необхідні параметри (рис. 3.3). У випадку із застосуванням динамічного варіанту повідомлень в контексті роботи обробника HTTP роута ця здатність цілком дала би можливість звернутись до інших частин запиту, включно з HTTP заголовками, які не розглядались при написанні оригінальної версії програмного забезпечення.

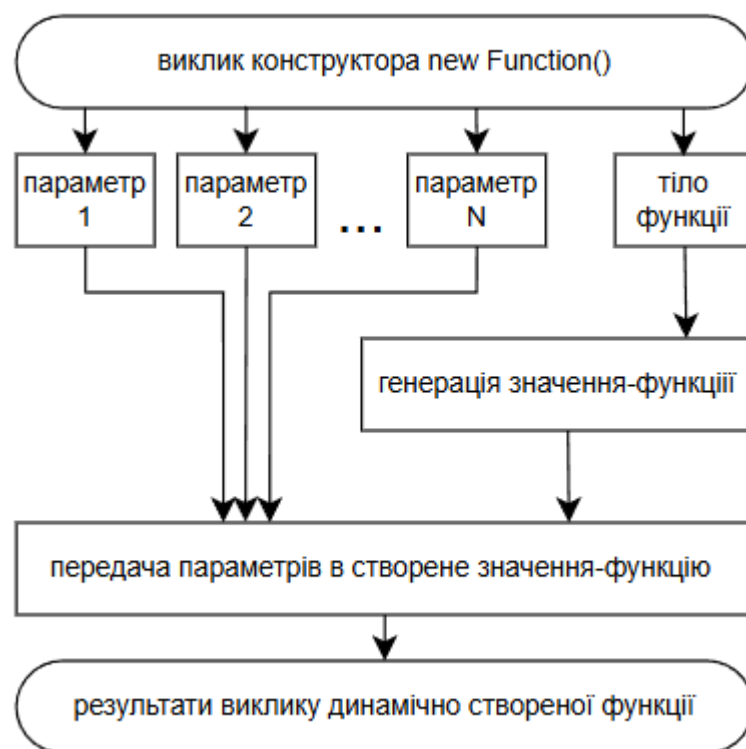


Рисунок 3.3 – Схема роботи конструктора `new Function()` в Javascript (відповідно до [75])

Подальше дослідження стосовно розробки динамічного варіанту повідомлень, а також дотичного до нього функціоналу включно із вмістом подальших розділів буде ґрунтуватись саме на можливостях Javascript.

Отже, цільовим підходом при реалізації архітектури повідомлень із динамічним вмістом є використання можливостей мов програмування для динамічної генерації виконуваного коду та його виконання з певними параметрами. З точки зору механізму адаптивного логування це вимагатиме доповнити сигнатуру методу (2.2) – повідомлення *M* тепер має існувати в двох

варіантах: базовому та динамічному, здатному згенерувати відповідний функціонал та виконати його із певними параметрами. В свою чергу, конфігураційний словник (2.4) також потребуватиме змін, оскільки саме там має бути джерело для рядкових представлень тіл функцій (адже лише цей варіант дозволить змінити поведінку не впливаючи безпосередньо на код та не призводячи до його перезбірки). Важливо також акцентувати на тому, що хоча ми вже обмежені використанням мови Javascript, подальші архітектурні рішення все ще мають створюватись із міркувань гнучкості та простоти задання щонайменше у питанні серіалізації конфігурації.

3.3 Формалізація модифікованих сигнатур методу адаптивного логування відповідно до архітектури динамічного варіанту повідомлень

Як було згадано раніше, наступним кроком є модифікувати сигнатури відповідних методів адаптивного логування щоб уможливити використання двох типів повідомлень [4]. З урахуванням цього, відповідні зміни до функції логування Синглтону виглядають наступним чином:

$$f_{log\ adp} = f(Sev, M_{DU}, T_{incl}) \quad (3.1)$$

Індекс DU поруч з параметром повідомлення у новому визначенні позначає концепт, взятий із теорії алгебраїчних типів даних [76], під назвою “сума”, а більш конкретно – його імплементацію, взятую із суперсета над мовою програмування Javascript під назвою Typescript [77], “discriminated union”. Формула (3.2) демонструє загальну структуру цього типу даних:

$$\begin{cases} variant: "1", & \dots \\ variant: "2", & \dots \\ \dots & \\ variant: "N", & \dots \end{cases} \quad (3.2)$$

Для зображеної комбінації із N варіантів справедливо наступне: загалом неможливо сказати чи співставні між собою дві випадково взятих варіанти (чи мають вони однакові поля з однаковими типами змінних), проте абсолютно

точно можна сказати, що значення поля “variant” однозначно класифікує який саме варіант записаний у значення. Назва поля не обов’язково має бути саме такою, проте має зберігатись основний принцип – існування спільної для всіх варіантів властивості, значення якої беззаперечно дає змогу описати решту очікуваних властивостей. Назва такої властивості “discriminator” (і відповідно від неї ж назва “discriminated union”).

У випадку адаптивного методу логування із динамічними повідомленнями модифікація повідомлення M із використанням цього типу даних виглядає наступним чином:

$$M_{DU} = \begin{cases} \begin{cases} type: "static" \\ msg: string \end{cases} \\ \begin{cases} type: "dynamic" \\ params: \{ key; value; \} \\ bodyId: string \end{cases} \end{cases} \quad (3.3)$$

Властивістю-дискримінатором вважатимемо властивість *type* з двома можливими значеннями: “static” та “dynamic”. Частина юніона, позначена значенням “static”, є новим позначенням попереднього варіанту лише з текстом лог-повідомлення. Варто зазначити, що можливість до інтерполяції змінних в процесі виконання не відображена в формальному позначенні, оскільки для цього варіанту повідомлень це не вагомою архітектурною складовою, а важливий лише вміст, який був сформований. Друга частина, з відповідним значенням дискримінатора “dynamic” описує динамічний варіант лог-повідомлень.

Відповідно до властивостей конструктора функцій `new Function()`, створені значення-функції матимуть доступ до глобальної області видимості змінних та до тих параметрів, що були передані безпосередньо на початкових позиціях виклику конструктора. В межах роботи динамічного варіанту повідомлень пропонується щонайменше наголосити на тому, що очікувана область видимості для динамічного коду має бути чітко визначена та контрольована. Для цього використовується елемент *params* в другому варіанті M_{DU} , який, в свою чергу,

представляє собою структуру із рядковими ключами та довільними значеннями, реалізовану через Javascript об'єкт [78], що по суті є реалізацією для розміщення даних типу “ключ-значення”. Дані об'єкти є дуже гнучкими компонентами, які можна створити або використавши літерал “{}”, або застосувавши конструктор “new Object()”, в подальшому звертаючись до відповідних значень за допомогою синтаксису “object.propertyName” чи “object[‘propertyName’]”. Завдяки можливості динамічно додавати бажані ключі та значення, цей параметр має змогу вмістити всі необхідні елементи для коректного опрацювання виклику динамічно згенерованої лог-функції. Завдяки цьому параметру виклики лог-функції з динамічним варіантом повідомлення будуть більш зрозумілими та володітимуть більш очікуваною поведінкою

Наступним кроком є визначення операцій, які необхідно виконати над словником переданих змінних (власне мова про тіло динамічної функції), і оскільки виклики методу логування є частиною коду, що потрапляє в такі процеси як лінування, компіляція та (в широкому розумінні) збірка, місце виклику не може ні в якій формі включати в себе тіло функції (хоча навіть це вже однозначно дало би більше спроможностей порівняно із базовим варіантом повідомлень лише з рядком), адже тоді не буде змоги змінювати його без зміни вихідних артефактів для роботи системи. Тому третім параметром динамічного варіанту є *bodyId*, що являє собою унікальний ідентифікатор тіла функції, яке буде використано для збірки та запуску динамічного коду для логування. Як результат унікальності цього ідентифікатора, розробник отримує здатність стабільно посилатись на певне місце в коді після вибору достатньо унікального значення, і через це – змінювати вихідний код функції, відповідно до поставленої задачі, з високим рівнем гнучкості. Важливо відмітити, що словник параметрів *params* формально теж має ту саму проблему, що й запис тіла функції безпосередньо в місці виклику лог-методу – передавши визначені аргументи ми втрачаємо можливість звертатись до чогось іншого (не враховуючи сторонні механізми роботи зі змінними та їх областями видимості по типу механізму спливання в Javascript [79]), проте з метою встановлення прозорого та однозначного зв'язку між значеннями, яких набувають змінні в процесі роботи

системи, цей параметр приведений саме в такому вигляді. Щоб вирішити проблему неможливості змінити перелік аргументів в словнику після процесу збірки коду можливим рішенням було би скористатись динамічною природою Javascript об'єктів (що не накладає обмеження на те які саме ключі та значення можуть бути додані) та додати відразу всі необхідні значення, інстанси та інші компоненти процесу виконання, які можуть становити інтерес в процесі відладки та пошуку проблем в системі. На відміну від рішення із заданням вмісту динамічних функцій ззовні, таке рішення не може бути застосоване для аргументів, що використовуються при виклику такої функції, і тому це обмеження є вимушеним.

Наступним кроком є формалізація оновленого конфігураційного словника (2.4) відповідно до необхідності мати можливість знайти відповідне тіло динамічної функції за заданим *bodyId*. Для повноцінного опису динамічного варіанту повідомлень також необхідно мати особливий параметр, завдяки якому можна попередити використання доволі небезпечного механізму збірки коду в процесі виконання програми в prod середовищі.

$$C = \left\{ \begin{array}{c} T_{11}^{mod} \wedge T_{12}^{mod} \wedge \dots \parallel T_{21}^{mod} \wedge T_{22}^{mod} \wedge \dots \parallel \dots \\ isProd \\ M_{dyn} \end{array} \right. \quad (3.4)$$

Параметр *isProd* є булевим аргументом конфігурації (тобто з можливими значеннями true або false) і має бути рівним true у випадку якщо розробник використовує метод адаптивного логування в prod середовищі. В такому випадку будь який виклик лог-функції з динамічним варіантом повідомлення має бути проігнорований (наприклад через задання вмісту лог-повідомлення рівному порожньому рядку). Вся необхідна інформація для встановлення взаємозв'язку між ідентифікаторами тіл функцій та відповідними тілами буде розміщена в *M_{dyn}* та матиме наступний вигляд:

$$M_{dyn} = Map < string, string > \quad (3.5)$$

Map в даному випадку є структурою подібною до “хеш-мапи” [80], що дозволяє знайти тіло функції за відповідним ідентифікатором (ідентифікаторами

є ключі мапи, а тіла функцій – значеннями). Додавання даного параметру в сигнатуру нової функції реініціалізації є максимально доцільним, оскільки робота саме цього методу дозволяє змінювати бажану поведінку відносно різних рівнів критичності чи тегів лог-повідомлень при виконанні програмного коду, а отже цілком підходить для перевизначення вмістів тіл функції динамічного варіанту повідомлень. Проте, як вже було згадано, важлива особливість полягає в тому, що використання цього конфігураційного параметру передбачає використання такої структури для представлення M_{dyn} , що піддається простій серіалізації та десеріалізації, оскільки згадані методи передачі повідомлень – такі як зчитування конфігураційного файлу при отриманні відповідного міжпроцесового сигналу, передача повідомлення через Unix сокет чи використання механізму каналів системи управління базою даних PostgreSQL – однозначно здатні працювати з текстовими даними, а значить будуть спроможні передати оновлений мапінг ідентифікаторів до рядків виконуваного коду, щоб змінити очікувану поведінку лог-виклику. На рис. 3.4 зображено більш детальний схематичний алгоритм взаємодії з двома можливими варіаціями лог-повідомлень.

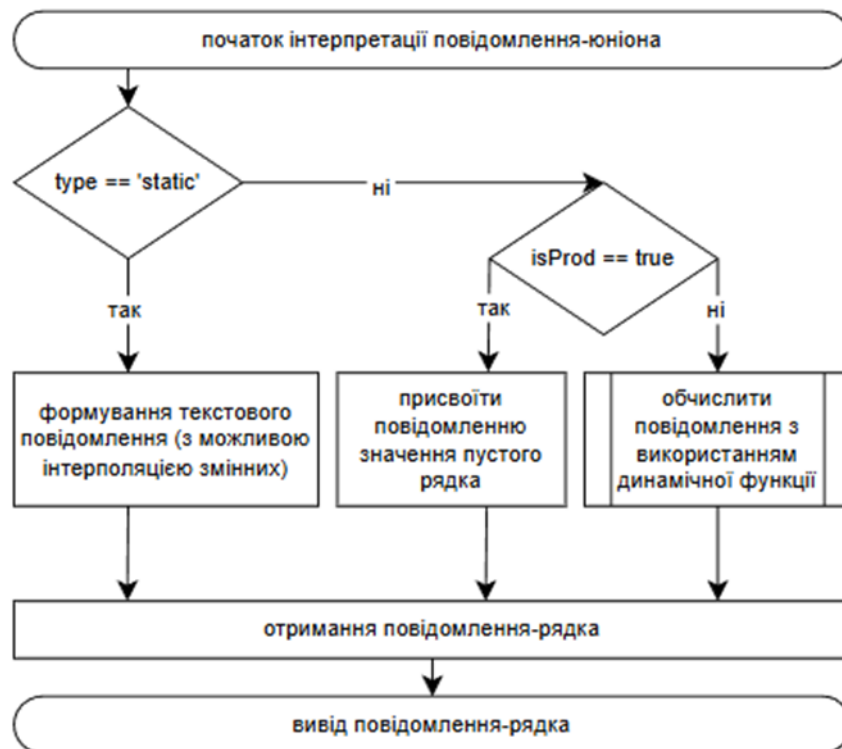


Рисунок 3.4 – Визначення повідомлення-рядка відповідно до варіанту повідомлення (результати отримані автором самостійно)

Обчислення повідомлення з використанням динамічної функції є окремим підпроцесом та буде деталізовано після обговорення деяких аспектів роботи динамічних тіл та хеш-мапи M_{dyn} , і оскільки тепер дві основні складові визначення динамічного варіанту повідомлень переважно визначені (оновлена структура повідомлення M_{DU} в функції адаптивного логування та параметр-мапа для функції реініціалізації), важливо звернути увагу на певні обмеження та вузькі місця. Перш за все, здійснюючи пошук конкретного ідентифікатора ймовірним є варіант, що не існує відповідного тіла функції, для якого цей ідентифікатор є ключем. Такий випадок однозначно є некоректним з точки зору цілісності програмного коду, адже всі ідентифікатори заздалегідь відомі і не можуть бути обчисленими чи згенерованими динамічно, а значить повинні мати однозначне співпадіння. Втім, оскільки логування в цілому, та метод адаптивного логування зокрема, є допоміжним функціоналом при написанні комп'ютерних систем (особливо зважаючи на виключну орієнтацію на середовища dev та stage), критичність та небезпеку такого неспівпадіння не можна назвати високими. Відтак, імплементації повинні розраховувати, що при настанні такої ситуації система щонайменше без видимих ознак продовжує виконувати свою основну функцію, а щонайбільше – повідомляє розробника про появу такого неспівпадіння, все так само не впливаючи на основний процес виконання. Оскільки динамічний варіант повідомлень не є доцільним для використання в середовищі prod, а в середовищах dev та stage дану помилку можна відслідкувати відносно просто та в межах тестування написаного функціоналу, такий “тихий” підхід (на противагу генерації помилки відносно високого рівня критичності) є цілком задовільним.

Наступним важливим аспектом роботи з кодом, згенерованим “на льоту” є кооректне опрацювання виняткових чи помилкових ситуацій. Загалом, в процесі звичайного виконання коду потенційні проблематичні місця обрамляються у відповідні синтаксичні структури для перевірки, такі як `try...catch` [81] або ж опрацьовують спеціальні результуючі значення, що повертає функція чи метод. У випадку із динамічним кодом, що використовується в межах динамічного варіанту лог-повідомлень, потенційно будь-яке місце може згенерувати

помилкову ситуацію (наприклад, через некоректно сформований код, який існує у вигляді звичайного тексту згідно з (3.5) або через будь-яку іншу логічну помилку), а значить механізм адаптивного логування має забезпечити коректну роботу в таких випадках і виключити необхідність розробнику по-особливому ставитись до різних варіантів виклику функції логування – базового чи динамічного. У випадку виникнення таких виняткових ситуацій, механізм роботи може цілком співпадати із реакцією на помилку, що виникає при неспівпадінні вказаного ідентифікатора тіла функції лог-виклику із списком всіх доступних через конфігураційний об'єкт, і може бути проігнорована. Очікувана схема роботи зображена на рис. 3.5.

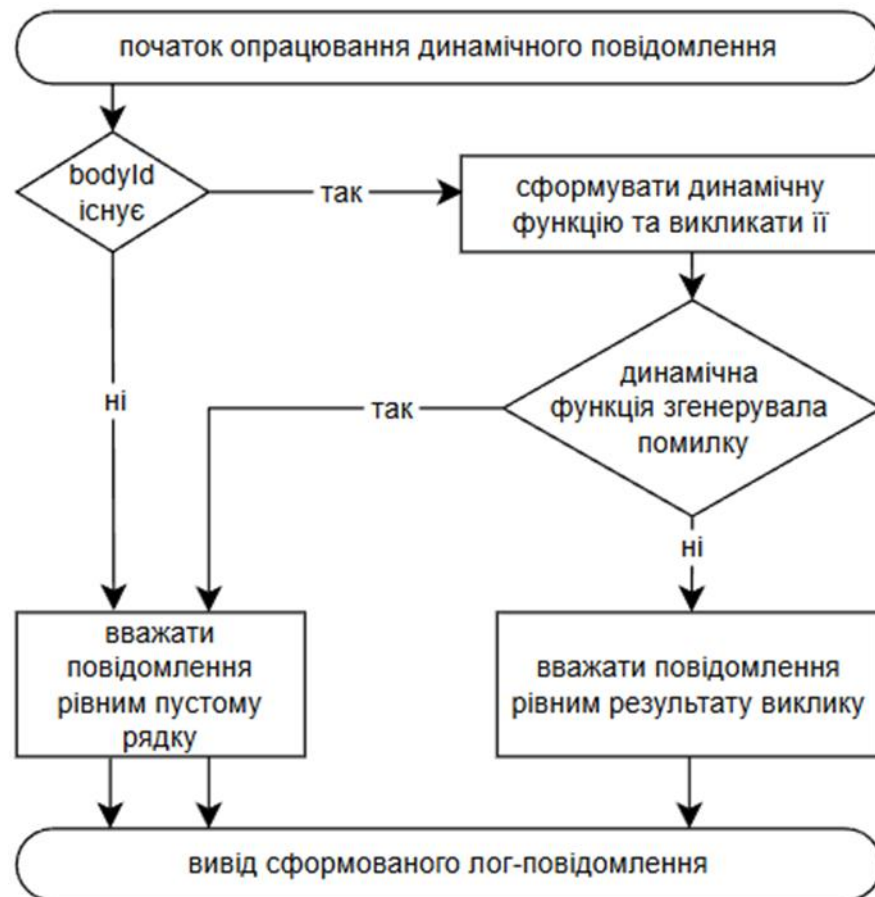


Рисунок 3.5 – Процес обчислення повідомлення з використанням динамічної функції (результати отримані автором самостійно)

Описаний підхід дозволяє в загальному випадку прирівняти опрацювання динамічного варіанту до статичного, оскільки обидва в результаті отримують на виході рядок-повідомлення, що підтверджує зовнішню подібність обох варіантів

оновленого повідомлення M_{DU} та дозволяє працювати з їх результатами в однаковий спосіб.

Ще одним обмеженням є те, що відповідно до необхідності мати доступ до механізмів, що дозволяють формування та виконання коду вже в процесі роботи основного коду, вибір можливих платформ та середовищ, в яких можливо реалізувати оновлений метод адаптивного логування, дещо звужується (і використання мов із пре-скомпільованими компонентами скоріше за все не є можливим). Втім, для таких випадків базовий метод все ще здатний забезпечити покращений рівень спостережності порівняно із підходом до логування, що базується виключно на механізмі критичності.

Важливо зазначити, що хоча “тихе” ігнорування помилок, які виникають при роботі з тілами функцій динамічного варіанту лог повідомлень, частково вирішує проблему із ненадійністю чи несправністю написаного коду, існує інша проблема: написаний таким чином код може звертатись до сутностей та працювати з механізмами, які мають здатність безпосередньо впливати на звичайну роботу системи (як от запис в файли, з’єднання з базою даних, робота з мережевими сокетам, тощо) і в загальному є нетиповими для задачі логування. Тому доцільним було би мати механізм, що дозволяв би контролювати та перевіряти дії та поведінку коду описаного в межах динамічних тіл функцій.

3.4 Формалізація механізму роботи підсистеми валідації динамічних повідомлень ґрунтуючись на форматі JSON-schema

Загалом завдання дослідження поведінкових особливостей коду є доволі типовим та розповсюдженим для таких аспектів роботи комп’ютерних систем як, наприклад, взаємодія із шкідливим програмним забезпеченням. При роботі з програмним кодом, щодо якого існують підстави вважати, що він створений зловмисниками та має на меті отримати несанкціонований доступ, знищити чи спотворити інформацію, застосовуються різні механізми. Типовим та відносно простим є використання технік статичного аналізу програм та модулів [82], що включають в себе зняття хеш-зліпків з файлів бінарного коду, використання імен

файлів рядків (в тому числі IP адрес, веб-доменів, тощо) для ідентифікації вже відомих вірусів та зловмисного програмного забезпечення. Також застосовується підхід, при якому здійснюється перетворення бінарного коду назад у вихідний код (із певними можливими спотвореннями) з метою подальшого поведінкового аналізу. Альтернативним підходом є динамічний аналіз, що передбачає спостереження за поведінкою підозрілої програми в закритому та ізольованому (інколи віртуалізованому) середовищі, що має на меті попередити її розповсюдження та дати більше деталей для аналітиків та осіб, відповідальних за опрацювання звернень, пов'язаних з даним інцидентом. При цьому також застосовуються різні підходи для ізоляції, як на рівні запуску повноцінних гостьових операційних систем в межах основної (так звані “віртуальні машини” та відповідне програмне забезпечення для їх роботи, як от VirtualBox), так і менш радикальне обмеження лише тих ресурсів чи файлів, до яких має доступ підозрілий програмний продукт [83]. Перевагою динамічного аналізу є можливість більш точно та ретельно дослідити, які саме дії, доступ до яких ресурсів та об'єктів здійснює підозрілий код.

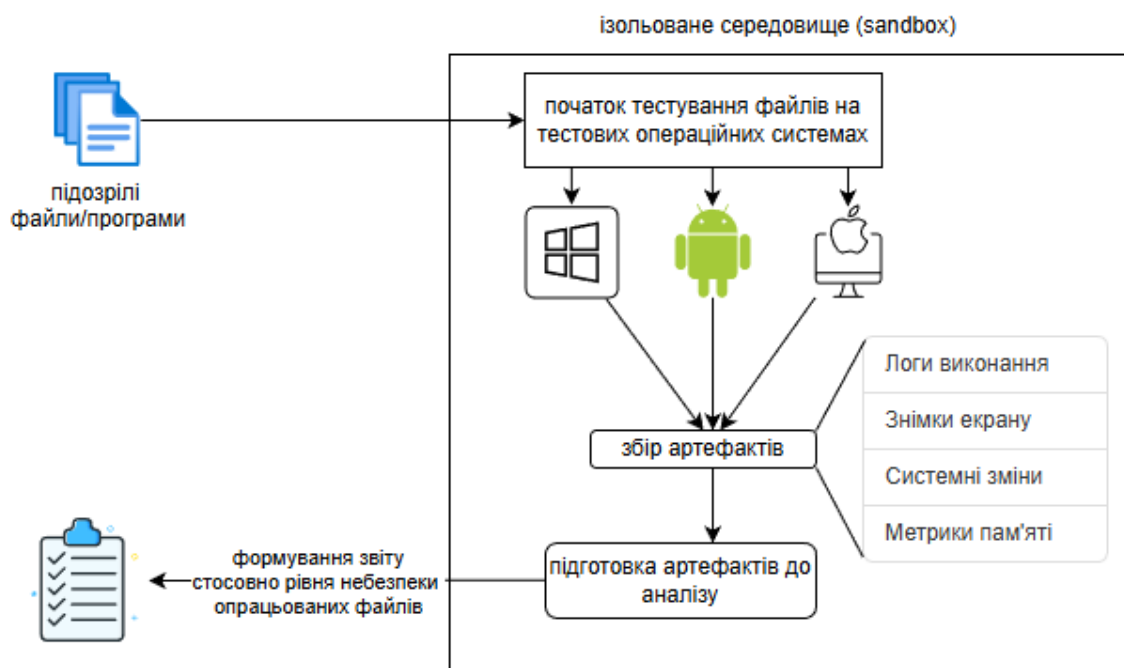


Рисунок 3.6 – Приклад функціонування ізольованого середовища для динамічного аналізу поведінки програми (відповідно до [83])

Завдяки такого роду дослідженню можливо отримати велику кількість змістовних метрик (вплив на системні параметри, споживання обчислювальних ресурсів, зовнішні ознаки виконання чи навіть візуальний перебіг роботи програми), пов'язаних із роботою та поведінкою файлів чи програмного коду, і в результаті отримати більш чітке уявлення стосовно рівня небезпеки. Проте, в межах задачі контролю та аналізу коду в динамічному варіанті повідомлень методу адаптивного логування використання динамічного аналізу коду на основі віртуальних машин чи обмеження доступу до ресурсів є доволі складним для впровадження: в першу чергу через миттєвий та невиннований ріст рівня вимог до середовищ та платформ, що імплементують метод, а також через неможливість в загальному випадку опрацьовувати кожен виклик двічі – на ізольованій системі та ще раз на реальній, адже для цього необхідно вміти зберігати та передавати відповідний контекст виконання з усіма активними значеннями в це саме контрольоване середовище. І взагалі, хоча природа задач подібна, вимоги до точності та ізоляції суттєво відрізняються, адже за рахунок розмежування доступу здійсненого засобами за межами методу адаптивного логування, рівень довіри до тіл динамічних функцій є більш високим, аніж у випадку підозрілого програмного забезпечення. Тому більш доцільним видається адаптація підходів статичного аналізу для перевірки та попередження небажаної поведінки динамічних повідомлень.

Втім, далеко не всі методи статичного аналізу згадані раніше можуть бути застосовані. Так, наприклад, зняття хеш-зліпків із навіть не скомпільованого, а людино-зрозумілого вихідного коду в загальному не дасть результатів, оскільки на противагу проблемі з розповсюдженням одного й того самого файлу, що містить шкідливе програмне забезпечення, на велику кількість окремих комп'ютерів (і як наслідок можливість його пошуку по унікальному зліпку), тіла функцій динамічного варіанту повідомлень є доволі унікальними та в загальному навряд чи будуть перевикористані в багатьох місцях. Аналогічно і з пошуком певних рядкових значень, повторення чи взагалі існування яких не дає зрозуміти майже нічого, але, що головне – має доволі мало індикаторів, що дозволяє говорити про певну поведінку коду, який досліджується. Втім підхід із

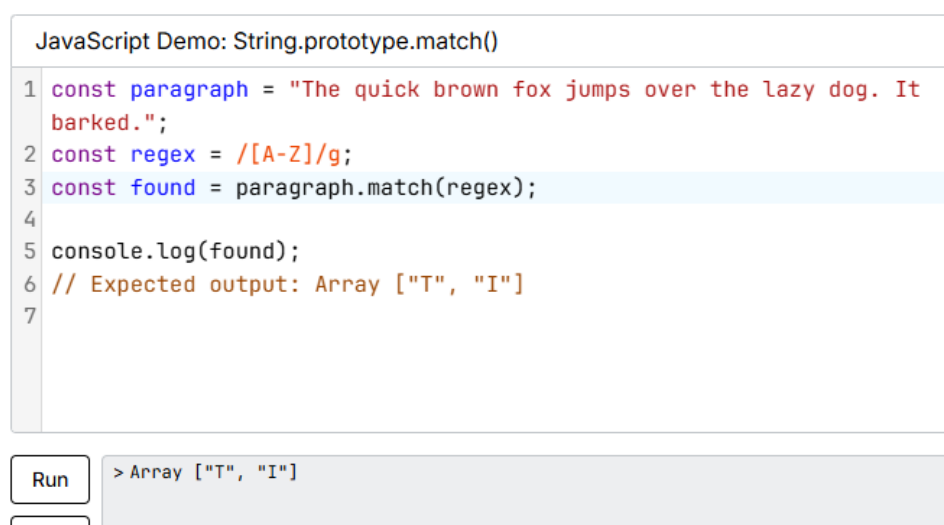
дослідженням вихідного коду (який присутній у вигляді конфігураційного параметру M_{dyn}) може бути більш доцільним та результативним.

Найпростішим способом дослідження є перегляд динамічного коду “в ручному режимі”, тобто процес, при якому розробник вивчає та досліджує сам виконуваний код, його умови та граничні випадки. За рахунок розуміння, сформованого в результаті такого дослідження, можна в подальшому приймати рішення стосовно того, наскільки даний код підходить для задачі. Проте доволі значним недоліком такого підходу є по-перше, що кожна зміна коду динамічного тіла функції вимагатиме повторного перегляду, що займатиме певний час, а по-друге – доволі відчутна залежність від здібностей та рівня підготовки розробника, який опрацьовує відповідний уривок програмного рішення. З урахуванням того, що в фінальному представленні у вигляді лог-методу Синглтона використання динамічного варіанту повідомлень є не особливо складною задачею навіть для розробників з невисоким рівнем компетенції, ручне опрацювання в загальному випадку є неоптимальним та навіть інколи не особливо надійним способом статичного аналізу. Відповідно, більш бажаним було би мати якийсь спосіб автоматизованої оцінки чи перевірки. Це, в свою чергу, значно би звузило можливості стосовно того який саме код може міститись в тілах динамічних функцій (адже написання програмного рішення, здатного оцінити характеристики іншого програмного рішення, є складною, чи навіть в окремих випадках неможливою, задачею, на противагу чому аналіз, що ґрунтується на розумовій діяльності людини, є більш гнучким), проте доволі очікувано отримати інструментарій, який значно швидше справляється з повторними перевітками. При цьому, якщо перевірку дозволених операцій чи дій можливо описати з використанням базових критеріїв (використання лише певних операторів, заборона роботи із зовнішніми системами, тощо), отриманого функціоналу могло би бути цілком достатньо в межах динамічного варіанту повідомлень методу адаптивного логування.

Для аналізу вихідного коду можна застосувати, наприклад, механізм регулярних виразів [48]. Регулярними виразами називають шаблони та спеціалізовану мову, збудовану навколо цих шаблонів, що виникли протягом

років роботи з різними задачами (як от опрацювання прозових творів, звітів, розмітки мережі Інтернет, тощо) для єдиного узагальненого рішення проблеми з опрацюванням довільного тексту. Використовуючи механізми, як от пошук початку та кінця рядку, робота з класами символів (латинські, кириличні, чисельні, тощо), здатність ігнорувати регістр символів в слові, знаходити початок та кінець слова чи навіть знаходити співпадіння, базуючись на вже знайдених раніше, розробник має можливість знаходити бажані чи небажані структури та патерни поведінки, що в свою чергу могло би вирішити проблему перевірки тіл динамічних функцій. Враховуючи той факт, що середовище виконання Javascript містить доволі потужний інструментарій для роботи з регулярними виразами доступний завдяки інструментам глобального оточення, як от конструктор RegExp [84] чи стандартні методи класу String “match” та “matchAll”, використання такого підходу було би доволі зручним та простим. На рисунку 3.7 представлений типовий сценарій пошуку з використанням регулярних виразів в мові програмування Javascript: в заданому реченні знайдені всі літери, що написані символами латинського алфавіту у верхньому регістрі. В результаті знайдені обидві літери, які позначають початок кожного з двох речень в рядку, що опрацьовується.

Try it



The screenshot shows a web-based JavaScript demo environment. At the top, the title is "JavaScript Demo: String.prototype.match()". Below the title is a code editor with the following code:

```

1 const paragraph = "The quick brown fox jumps over the lazy dog. It
  barked.";
2 const regex = /[A-Z]/g;
3 const found = paragraph.match(regex);
4
5 console.log(found);
6 // Expected output: Array ["T", "I"]
7

```

Below the code editor is a "Run" button. To the right of the button, the console output is displayed: "> Array ["T", "I"]".

Рисунок 3.7 – Приклад роботи методу “match” класу String в мові програмування Javascript (відповідно до [84])

Проте, попри зручність та переваги доступності механізму RegExp, його застосування для задачі перевірки та контролю вмісту тіл динамічних функцій є дещо обмеженим. В першу чергу, через величезну кількість можливих варіантів та комбінацій, які можуть бути використані при написанні коду (так наприклад лише способів задання змінних в мові Javascript є щонайменше 3). Додатково варто зазначити, що хоча формат подання формально підходить під опрацювання механізмами та засобами регулярних виразів, результуюча структура суттєво відрізняється від, наприклад, прозових творів чи матеріалів довільної розповідної форми. Рівно як з форматами з чітко структурованим поданням, як от Hypertext Markup Language (HTML) чи Extensible Markup Language (XML), вихідний код Javascript описує структуру більш вкладену “вглиб”, аніж “вшир”. Зважаючи на це, для вирішення задачі перевірки та контролю вмісту тіл динамічних функцій більш доцільно було би використати представлення, що близько (якщо не повністю) відтворює цю вкладеність.

Одним із етапів аналізу та виконання вихідного коду на Javascript є перетворення оригінального тексту в проміжний формат, що має назву Abstract Syntax Tree [85] та є спеціалізованою структурою даних, в якій було видалено всю надлишкову інформацію з вихідного коду для подальшого використання компілятором в процесі лексичного аналізу та парсингу, та яка має передбачувану будову, що дозволяє опрацьовувати її програмними засобами на зразок патерну програмування Visitor [86]. Схема приведення вихідного коду до AST наведена на рис. 3.8.

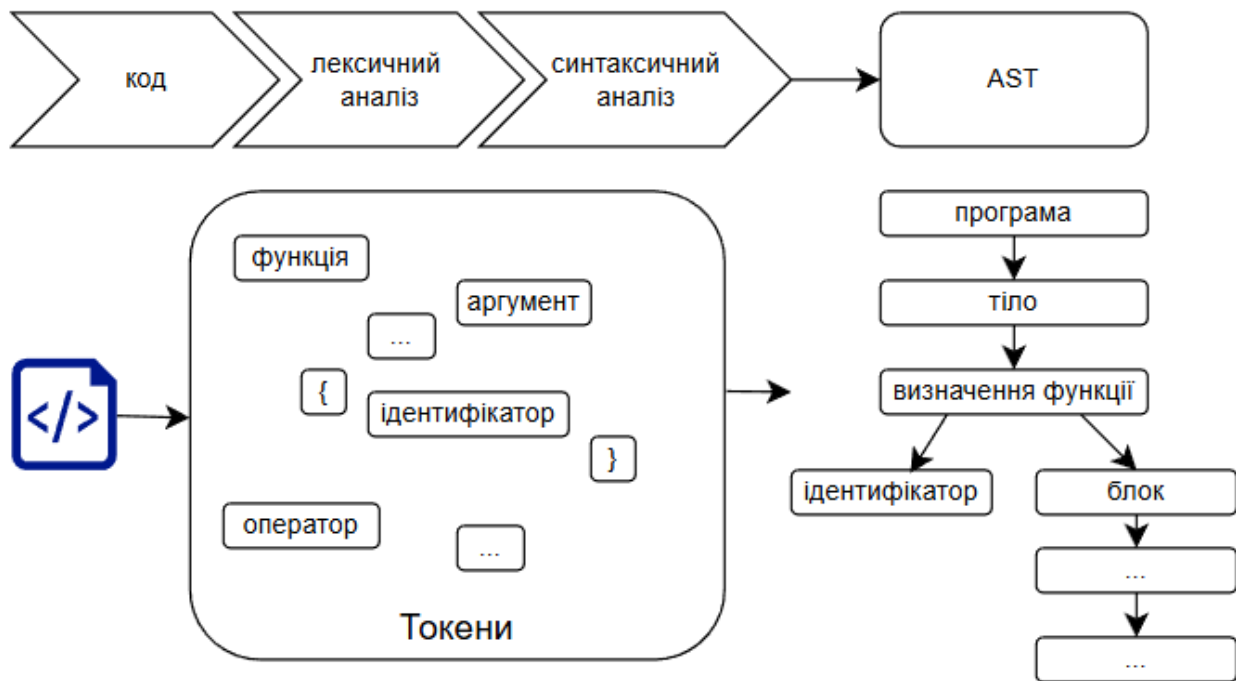


Рисунок 3.8 – Схема перетворення вихідного коду в абстрактне синтаксичне дерево (відповідно до [85])

Відповідно до даного процесу, файл вихідного коду спершу має пройти процес токенизації (тобто розбиття на найменші змістовні частинки в термінах певної мови програмування – токени), що відбувається завдяки виконанню лексичного аналізу, і потім згенеровані компоненти поєднуються в ієрархічну структуру за результатом процесу синтаксичного аналізу. Порівняно із варіантом опрацювання через регулярні вирази, даний підхід більш коректно та влучно описує справжню форму перетвореного програмного коду, а також значно зменшує необхідність розробляти та опрацьовувати доволі складний та нетривіальний синтаксис Javascript в “ручному режимі”, адже це є однією із задач алгоритмів для створення абстрактних синтаксичних дерев. Повертаючись до прикладу з роботою веб-сервера та типовою необхідністю дослідження HTTP заголовків, що пов'язані із запитом, код для виконання такої операції може виглядати, наприклад, як “req.headers”, що позначає доступ до властивості “headers” об’єкту “req”. Відповідне синтаксичне дерево матиме наступний вигляд:

```

1  {
2      "type": "Program",
3      "body": [
4          {
5              "type": "ExpressionStatement",
6              "expression": {
7                  "type": "MemberExpression",
8                  "object": {
9                      "type": "Identifier",
10                     "name": "req"
11                 },
12                 "property": {
13                     "type": "Identifier",
14                     "name": "headers"
15                 }
16             }
17         }
18     ]
19 }

```

Рисунок 3.9 – Абстрактне синтаксичне дерево для програмного коду, який зчитує заголовки клієнтського HTTP запиту (результати отримані автором самостійно)

Код опрацьований таким чином є більш зручним для подальшого програмного аналізу та пошуку небажаних чи очікуваних структур. Програма, перетворена в дане представлення, описується у вигляді ієрархічної структури “вузлів” [87] різного типу, як от `FunctionBody`, `WithStatement`, `FunctionDeclaration` та інші. Завдяки цьому розробник має змогу однозначно відсікти небажані дії, як от додавання оператора для дебагінгу через `DebuggerStatement` (оскільки тіла динамічних функцій для логування не є доцільним місцем додавання даної програмної структури в першу чергу через те, що її використання вимагає особливого розгортання програмного продукту, яке потребує відповідних приготувань для застосування в середовищі `stage` чи `dev`) чи оператора присвоєння в межах `AssignmentExpression` (адже зміна стану змінних програми може бути небажаною операцією в межах задачі спостереження з використанням лог-повідомлень).

Опрацювання цієї ієрархічної структури цілком можливе через написання вузькоспеціалізованого коду, що перебиратиме структуру вузол за вузлом, проте можливо не є найоптимальнішим варіантом. Альтернативним та більш зручним

варіантом було би використання готових методів аналізу Javascript структур із широкими можливостями для дослідження та перевірки їх складових. Для задач такого класу існує особливий вид програмних компонентів, що носять назву валідатори (англ. “validate” – “перевіряти”). Принцип їх роботи в загальному наступний: відповідний компонент (функція валідації) отримує об’єкт з даними, який необхідно перевірити, та схематичне подання очікуваних полів та їх типів даних (імплементация різниться в залежності від бібліотеки) і видає результат у вигляді булевого значення, яке в разі правдивості підтверджує, що переданий об’єкт з даним відповідає поданій схемі. Одним із таких програмних компонентів є Another Javascript Validator [88], чи скорочено Ajv. Для формування схематичного подання очікуваної структури використовується формат Javascript Object Notation, скорочено JSON [89], що побудований з використанням типів даних мови програмування Javascript (об’єкти, масиви, рядки, числа, булеві значення та особливе значення null), а також надбудову над ним під назвою JSON Schema [90]. Для опису очікуваної форми даних в JSON Schema можна вказати будь-який із 6 фундаментальних типів формату JSON та використати відповідні ключові слова (keywords), щоб чіткіше окреслити очікувану поведінку та вміст відповідного поля. Так, наприклад, для числових значень існує можливість задати мінімальне допустиме значення (minimum), мінімально допустиме значення “не включно” (exclusiveMinimum), чи є дане число цілим, чи є воно множником іншого, тощо. Для рядкових набір ключових слів включає в себе обмеження на мінімальну довжину (minLength), максимальну довжину (maxLength) та можливість задання регулярного виразу (pattern), якому має відповідати значення поля, а для об’єктів можна визначити перелік очікуваних полів (properties), дозвіл чи заборону додавання полів, які не вказані явно (additionalProperties), обов’язкові поля (required), та інші. Завдяки доволі широкому набору інструментів, валідація на базі JSON Schema з використанням Ajv є достатньо зручним способом для перевірки абстрактних синтаксичних дерев, сформованих з динамічних тіл функцій, що використовуються в повідомленнях з динамічним вмістом. Відповідна схема, що перевіряє структуру дерева, проілюстрованого на рисунку 3.9 виглядає наступним чином:


```

49 const schema = {
50   type: 'object', required: ['body', 'type'],
51   properties: {
52     type: { type: 'string', enum: ['Program'] },
53     body: {
54       type: 'array', maxItems: 1, minItems: 1,
55       items: {
56         type: 'object', required: ['expression', 'type'],
57         properties: {
58           type: { type: 'string', enum: ['ExpressionStatement'] },
59           expression: {
60             type: 'object', required: ['object', 'property', 'type'],
61             properties: {
62               type: { type: 'string', enum: ['MemberExpression'] },
63               object: {
64                 type: 'object', required: ['type'],
65                 properties: { type: { type: 'string', enum: ['Identifier'] } }
66               },
67               property: {
68                 type: 'object', required: ['type'],
69                 properties: { type: { type: 'string', enum: ['Identifier'] } }
70               }
71             }
72           }
73         }
74       }
75     }
76   }
77 };

```

Рисунок 3.10 – Структура JSON Schema для перевірки коду, що здійснює взяття властивості із об’єкта (результати отримані автором самотійно)

Як наслідок валідації тіла динамічної функції очікувано може виникнути помилка, вірніше процес валідації поверне результат, що сигналізуватиме про невідповідність переданих даних очікуваній схемі [91]. З огляду на такий розвиток подій, доцільним виглядає варіант дій зі сторони розробника, як і у випадку неспівпадіння унікального ідентифікатора тіла динамічної функції із переліком усіх можливих, а саме – “тихе” ігнорування. Мотивація здебільшого аналогічна як і до цього: через доповнюючу природу задачі логування було би недоцільно генерувати помилки, що можуть мати небажаний вплив на процес виконання основного коду, а також оскільки два середовища для яких існує сенс застосовувати динамічний варіант повідомлень є в широкому розумінні тестовими, неспівпадіння з очікуваною поведінкою (відсутність будь-якого логування) будуть видимі в короткотерміновій перспективі.

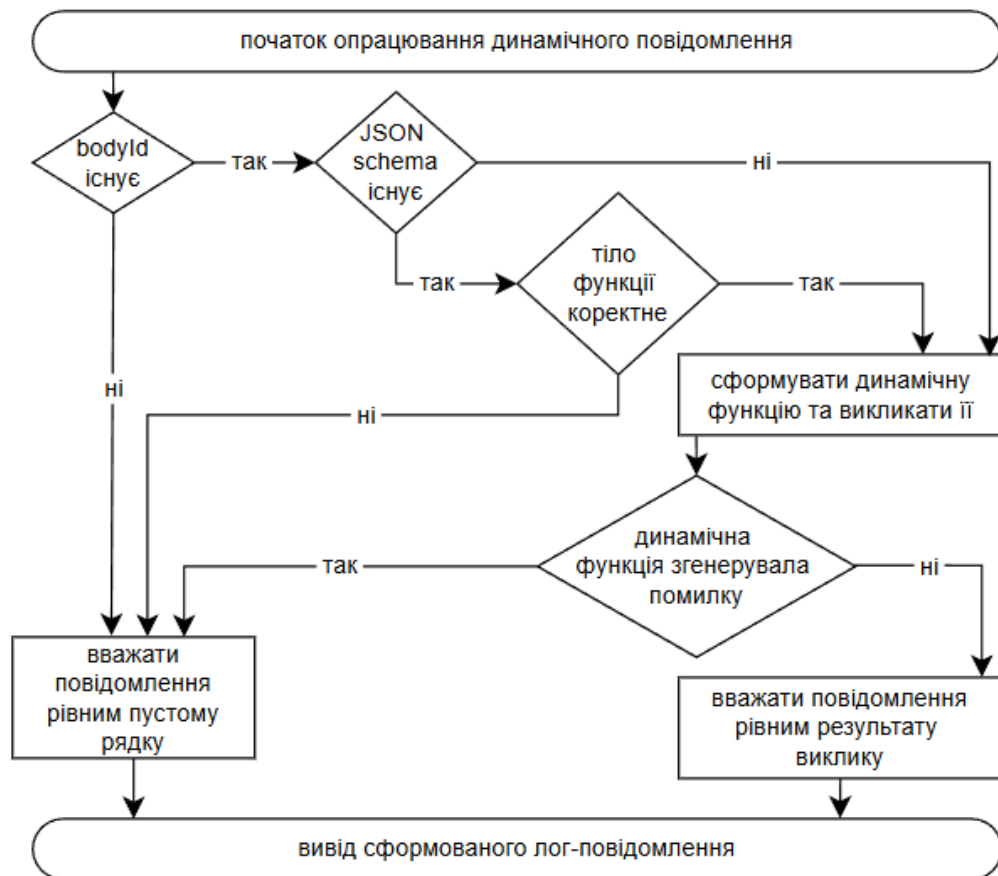


Рисунок 3.11 – Оновлений процес обчислення повідомлення використанням динамічної функції (результати отримані автором самостійно)

На рис. 3.11 наведено удосконалений алгоритм, завдяки якому відбувається обчислення результуючого лог-повідомлення з використанням як динамічного варіанту повідомлення, так і валідації за допомогою JSON-схем та AJV (що є надбудовою над алгоритмом на рис. 3.5). Згідно з новим процесом виконання, одразу після знаходження відповідного рядкового тіла динамічної функції, відбувається пошук схеми, якій він має відповідати, і у разі знаходження – відбувається процес перевірки. Якщо перевірка не пройдена (з причин порушення очікуваної логіки чи виникнення неочікуваних виняткових чи помилкових ситуацій) – повідомлення стає пустим рядком (що відповідає ідеї “тихого” ігнорування). Інакше – як і в попередньому варіанті результат роботи динамічної функції стає лог-повідомленням.

Інтеграція процедури валідації тіл динамічних функцій [5] [9] також здійснюється за аналогією до (3.5) і виглядає наступним чином: ідентифікація

конкретної функції для перевірки здійснюється за рахунок використання того самого унікального ідентифікатора, що і в M_{dyn} , а спосіб встановлення зв'язків між ним та відповідною JSON Schema встановлюється за допомогою хеш-мапи:

$$M_{dyn\ schm} = Map < string, O_{JSON-schema} > \quad (3.6),$$

де ключ є ідентифікатором, а $O_{JSON-schema}$ – об'єктом, подібним до зображеного на рис. 3.10. Завдяки тому, що формат JSON Schema піддається серіалізації, новий компонент налаштувань може бути доданий в конфігураційний об'єкт C:

$$C = \begin{cases} T_{11}^{mod} \wedge T_{12}^{mod} \wedge \dots || T_{21}^{mod} \wedge T_{22}^{mod} \wedge \dots || \dots \\ isProd \\ M_{dyn} \\ M_{dyn\ schm} \end{cases} \quad (3.7)$$

Оновлений таким чином конфігураційний словник все ще залишає можливість для взаємодії із функцією реініціалізації через базові методи операційних систем, як от читання файлів, проте також містить вичерпну кількість компонентів, необхідних для конструювання тіл динамічних функцій та їх валідації.

Висновки до розділу 3

За результатами розділу формальне визначення базового методу адаптивного логування було доповнено новим видом динамічних повідомлень, що дозволяють розробнику більш гнучко підходити до питання дослідження процесу виконання системи, аніж тегування та рівні критичності представлені у розділі 2. Робота відбувалась в такі етапи:

- було встановлено, що при розробці комп'ютерних систем в середовищах dev та stage, оскільки програмний продукт ще недостатньо відпрацьований та відтестований, інколи доволі важко розраховувати на точність та вичерпність існуючих лог-викликів;

- запропоновано впровадити динамічний варіант повідомлень, що базується на можливостях високорівневих мов програмування до динамічного формування та виконання коду, що в свою чергу вплинуло на функцію логування, де тепер повідомлення M може бути представлене як базовий варіант або динамічний, що дозволяє змінити обчислення не змінюючи вихідний код програмного компоненту, та конфігураційний словник, який тепер залежить від 4-х елементів;
- новий вид повідомлення формалізований з використанням алгебраїчного типу даних discriminated union, а взаємозв'язок між місцями лог-викликів з динамічним варіантом повідомлень та відповідними динамічними тілами встановлений за допомогою введення параметру M_{dyn} в формулі конфігураційного словника; параметри для динамічних повідомлень пропонується передавати явно через словник *params* динамічного варіанту повідомлення;
- додатково запропонований механізм статичного аналізу тіл динамічних функцій за допомогою абстрактних синтаксичних дерев виконуваного коду Javascript та JSON-схем, що дає змогу певною мірою обмежити функціонал динамічного формування коду при виконанні програми;
- матеріали розділу опубліковані в [4], [5] та [9].

РОЗДІЛ 4 ПРИКЛАДНА АРХІТЕКТУРА ФУНКЦІОНУВАННЯ МЕТОДУ АДАПТИВНОГО ЛОГУВАННЯ В ХМАРНИХ СЕРЕДОВИЩАХ

4.1 Опис необхідних складових для впровадження методу адаптивного логування в хмарному середовищі на прикладі веб-серверу

Метод адаптивного логування має дві основні переваги порівняно із базовим алгоритмом логування, що спирається лише на критичність лог-повідомлень: здатність до зміщення акценту в бік певної конкретної системи, компоненту чи методу, який необхідно дослідити в процесі відладки, та отримання деталізованого опису процесу його роботи як наслідок, а також здатність зберігати ті елементи стану виконання програмного коду, які існують лише в процесі роботи системи та не залишаються ні в якому вигляді на постійних запам'ятовуючих пристроях. Втім, щоб сповна розкрити весь потенціал цих підходів існують певні обмеження та вимоги до систем, для яких застосування даного методу буде найбільш доцільним.

По-перше, це мають бути багатокomпонентні системи, оскільки простота та прямолінійність системи та, вірогідно, задачі, для якої вона створена, дещо розмивають доцільність впровадження більш складного алгоритму логування, аніж базовий, адже якщо в процесі роботи алгоритм проходить очікуваний та відносно постійний набір кроків (звісно з урахуванням логічного розгалуження), деталізація цілком може бути досягнута за рахунок опрацювання помилкових ситуацій з використанням рівня критичності `error`, тоді як у випадку узагальненого недеталізованого виводу використовувати рівень `info` та рівень `debug` коли необхідно бачити більшу кількість нюансів (втім очікувано що адаптуватись до зміни вимог логування в процесі виконання все ще буде неможливо). З іншого ж боку, якщо робота програми спирається на багато компонентів, виконання яких відбувається паралельно (до прикладу – різні ендпоінти веб-сервера, опрацювання та видача відповідей на які виконується в широкому розумінні незалежно один від одного), то навіть за умови виконання кожного з них послідовно (у випадку веб-сервера процес опрацювання веб-

запиту виконується крок за кроком за допомогою відповідних функцій-хендлерів, від парсингу HTTP пакетів, даних сесії, ідентифікаторів користувача та власне вмісту запиту, до формування тіла відповіді на запит) здатність викоремлювати вузькоспеціалізовані місця (ендпоінти) та вимикати деталізацію журналювання решти може відчутно спростити відладку.

По-друге, чим довше “час життя” відповідного процесу, тим більшою є актуальність механізму зміни конфігурації без втрати стану. Застосування даної можливості для так званих “run to completion” алгоритмів є дещо недоцільним як з точки зору потенційно короткого часу життя процесу (що більш наочно зрозуміло для випадків таких як парсери файлів та текстів різних форматів, утиліти для маніпуляцій з медіафайлами, криптографічні алгоритми, та в загальному будь-яка інша задача, інтенсивна та цілеспрямована робота в межах якої відбувається безпосередньо від початку виконання до його повного завершення). Навіть суто з технічної точки зору виконання алгоритмів такого типу може не мати зручного місця для виклику методу реініціалізації, а наявність операції зчитування з файлової системи наступного варіанту конфігураційного об’єкту може бути взагалі неможливою або небажаною, якщо важливим є аспект швидкодії системи. З іншого боку, якщо час життя процесу доволі довгий, як от у веб-серверів, середовищ роботи з графічним інтерфейсом користувача, платформ для інтерактивного моделювання, тощо, здатність проводити відладку без втрати деталей, що характеризують поточний екземпляр активної програми, має однозначні переваги, а за рахунок функціонування таких систем у форматі, при якому існує постійно запущений базовий процес та при потребі створюються допоміжні, що виконують специфічні функції, технічна можливість додати додаткову логіку, що відповідатиме за реініціалізацію синглтону методу адаптивного логування є більш реалізовуваною.

В цілому, механізм роботи веб-сервера підходить до обох вимог, оскільки його ендпоінти представляють собою паралелізовані обчислювальні модулі, кожен з яких може бути доцільно логувати та досліджувати по-своєму, опрацювання веб-запиту може відбуватись складними багатокроковими алгоритмами, а час існування та роботи програми-сервера є відносно тривалим,

так як щоб задовольнити вимоги доступності відповідне програмне забезпечення повинно бути весь час активними та “слухати” на певному TCP порту операційної системи. В свою чергу, так як веб-сервери працюють в межах операційних систем та мають доступ як до файлової системи, що робить можливим створення редагування, читання, збереження файлу на файловій системі, в який буде записано конфігураційний об’єкт *C* в форматі, який імплементація методу адаптивного логування буде здатна зчитати, так і до механізму міжпроцесових сигналів, використання методу реініціалізації після отримання відповідного сигналу та зчитування оновленої конфігурації має всі технічні можливості для впровадження.

Звичайний процес роботи веб-серверів передбачає наявність зовнішніх клієнтів, що приєднуються використовуючи мережу Інтернет. При цьому спосіб приєднання може бути як публічний, коли будь-яка людина із мережевим з’єднанням має змогу перейти на веб-ресурс, так і приватний, коли для безпосереднього доступу необхідно пройти механізм аутентифікації та авторизації. Втім в межах описаного функціоналу методу адаптивного логування визначення публічного та приватного доступу, які також мають місце, дещо відрізняються (рис. 4.1).

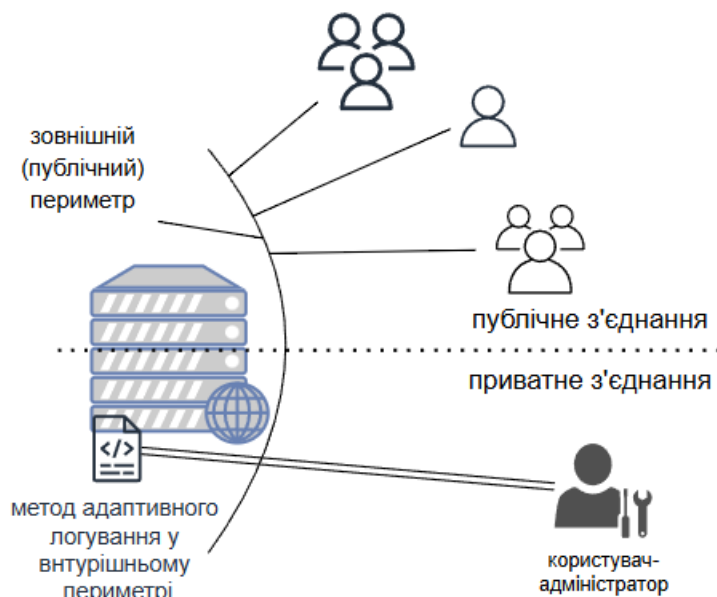


Рисунок 4.1 – Два типи доступу до веб-серверу із імплементацією методу адаптивного логування (результати отримані автором самостійно)

Як можна побачити, звичайні користувачі та групи користувачів мають доступ лише до зовнішнього (публічного) периметру системи, оскільки саме через нього очікувано буде здійснюватись основний набір операцій. З іншого ж боку доступ до деталей імплементації методу адаптивного логування дозволено лише користувачам з адміністративними повноваженнями із використанням спеціалізованого з'єднання безпосередньо з внутрішнім периметром (тобто глибше, ніж зовнішня публічна частина сервісу), що дозволить їм здійснити перевизначення конфігурації роботи Синглтону адаптивного логування та запустити процедуру реініціалізації.

Розміщення веб-серверів в кінцевому середовищі, де до них можливий доступ користувачів, може відбуватись або на власних потужностях – on premise setup [92] - індивідуального розробника (чи компанії-розробника) або з використанням відповідної послуги, яку надають провайдери хмарних послуг [93]. Обидві моделі мають власні переваги та недоліки, перелік частини яких наведено у таблиці 4.1.

Таблиця 4.1 – Порівняння підходів до розміщення програмного забезпечення

Розміщення на власних потужностях	Використання послуг хмарного провайдера
+ Програмне забезпечення встановлюється на фізичні сервери розробника (компанії-розробника)	- програмне забезпечення встановлюється на сервери провайдера
- відчутні початкові витрати для закупівлі складових частин інфраструктури	+ типовий підхід при використанні – “плати за те, що використовуєш”
- масштабування потребує ручного налаштування	+ масштабування ресурсів зазвичай легко налаштовуване засобами, що надані провайдером

- розробник (компанія-розробник) самостійно відповідають за безпеку та відповідність нормам закону	+ надавач хмарних послуг зазвичай пропонує готові рішення пов'язані із безпекою та відповідністю закону (з сертифікованим підтвердженням, тощо)
- у випадку виникнення неочікуваних ситуацій розробник (компанія-розробник) будуть вимушені шукати спеціалістів самостійно	+ провайдер хмарних послуг зазвичай винаймає команду спеціалістів для взаємодії та налаштування інфраструктури

Оскільки крім можливості встановлювати програмне забезпечення на сервери, які можна контролювати фізично, використання власних потужностей при розгортанні рішення по типу веб-сервера не має інших переваг в порівняння з використанням послуг відповідного хмарного провайдера, а також враховуючи той факт, що частина функціоналу, що стосується аутентифікації та авторизації при отриманні доступу користувачами з можливостями адміністратора, була заздалегідь винесена за межі формальних основ методу адаптивного логування, більш доцільним видається використання розгортання системи на потужностях хмарного провайдера.

4.2 Стратегія імплементації методу адаптивного логування з використанням інфраструктури Amazon Web Services

При виборі хмарного провайдера для впровадження імплементації методу адаптивного логування важливо звернути увагу на наявність необхідних компонентів-сервісів, як от обчислювальні потужності (по суті, сервери), сервіс для роботи зі сховищем (бажано методами файлової системи) та безпековий компонент. Бажаним, проте не критичним фактором, є використання провайдера, для якого відносно просто знайти необхідні матеріали (документацію), які будуть використані при плануванні та розгортанні системи. Три найбільш відомі

на даний момент компанії, що надають такі послуги, є Amazon Web Services (AWS) [94], Microsoft Azure та Google Cloud. Оскільки архітектура адаптивного методу логування розроблена таким чином, щоб бути сумісною із якомога більшою кількістю систем та платформ, можна стверджувати, що всі три провайдери мають необхідні сервіси для реалізації необхідного функціоналу. Так сервісом для виконання обчислень може слугувати Elastic Compute Cloud (EC2) у AWS, Azure Virtual Machines у Azure та Compute Engine у Google Cloud, для роботи зі сховищем можна використати сервіси Elastic Block Storage, Azure Disk Storage чи Persistent Disk, а для роботи з безпековими аспектами дозволяють працювати сервіси AWS Identity and Access Management (IAM), Entra ID (до того - Azure Active Directory) та Cloud Identity and Access Management відповідно. Отже, оскільки з технічної точки зору будь-який із представлених варіантів має бути цілком достатнім для реалізації необхідного функціоналу, вибір надавача послуг був здійснений з огляду на досвід використання автором роботи. Відтак, провайдером хмарних послуг в межах даної роботи було обрано Amazon Web Services [6].

В межах своєї діяльності AWS надає в користування клієнтам обчислювальні ресурси, мережі, мережеві ресурси, такі як публічні IP адреси, доменні імена, та інші. Базовим елементом створення програмного рішення у даного провайдера є Virtual Private Cloud (VPC), що за своєю суттю є виділеною частинкою хмарного середовища AWS, яке клієнт отримує у використання [95]. Базовими складовими частинами хмари є так звані регіони, що являють собою географічні локації в різних куточках планети. В межах окремого регіону існують зони доступності (англ. “availability zone”) - спеціально виділені дата центри в конкретном регіоні, з такими характеристиками як: задубльована інфраструктура енергоживлення з джерелами безперебійного живлення та, можливо, потужностями місцевої генерації, задубльованими мережевими потужностями, ізольовані точки відмови, тощо. При розробці архітектури для хмарної системи, розробник має змогу вибрати регіон для створення власного інстансу VPC, в межах якого розміщує інфраструктурні елементи, такі як інтернет шлюз (англ. internet gateway), таблиці маршрутизації (англ. routing

tables), компоненти перетворення мережевих адрес (англ. network area translation), а також підмережі, що можуть бути публічними чи приватними, та розміщуються в одній чи декількох зонах доступності (якщо існує необхідність мати вищий рівень безвідмовності програмного забезпечення), і зрештою розмістити необхідні для програмного рішення ресурси. Одним із таких ресурсів, який вже було згадано раніше, є сервіс під назвою Elastic Compute Cloud (EC2). Даний сервіс є одним із фундаментальних та найбільш використовуваних в лінійці доступних можливостей Amazon Web Services та по суті являє собою віртуальні потужності в хмарі, доступ до яких можна отримати “за вимогою” [96]. Основними можливостями EC2 є здатність взяти віртуальну машину в тимчасове використання, зберігати дані застосунку на віртуальних пристроях Elastic Block Storage, балансувати навантаження між інстансами віртуальних серверів з використанням Elastic Load Balancing, масштабування архітектури із використанням Auto Scaling Groups, та інші.

При створенні інстансу EC2 є можливість вибору операційної системи, що буде автоматично додана на виділений сервер, і цей вибір відбувається за рахунок використання образів віртуальних машин Амазон (англ. Amazon Machine Images), що по суті є певним “рецептом” завдяки якому на інстанс встановлюється та запускається необхідне програмне забезпечення, включно з інформацією про необхідні блокові пристрої (елементи сервісу Elastic Block Service), які необхідно підключити до новоствореного серверу. Для даної роботи було обрано Amazon Linux 2023 оскільки ця операційна система підтримується хмарним провайдером та є похідною від Linux, відповідно до чого цілком можливо використовувати як механізми введення/виведення для роботи з файлами, так і міжпроцесові сигнали для методу реініціалізації методу адаптивного логування. Відповідно, веб-сервер разом із імплементацією методу адаптивного логування буде розміщений на EC2 інстансі та матиме публічний доступ до HTTP ендпоінтів із мережі Інтернет. Загальна схема цього, з урахуванням необхідних елементів AWS інфраструктури, наведена на рис. 4.2.

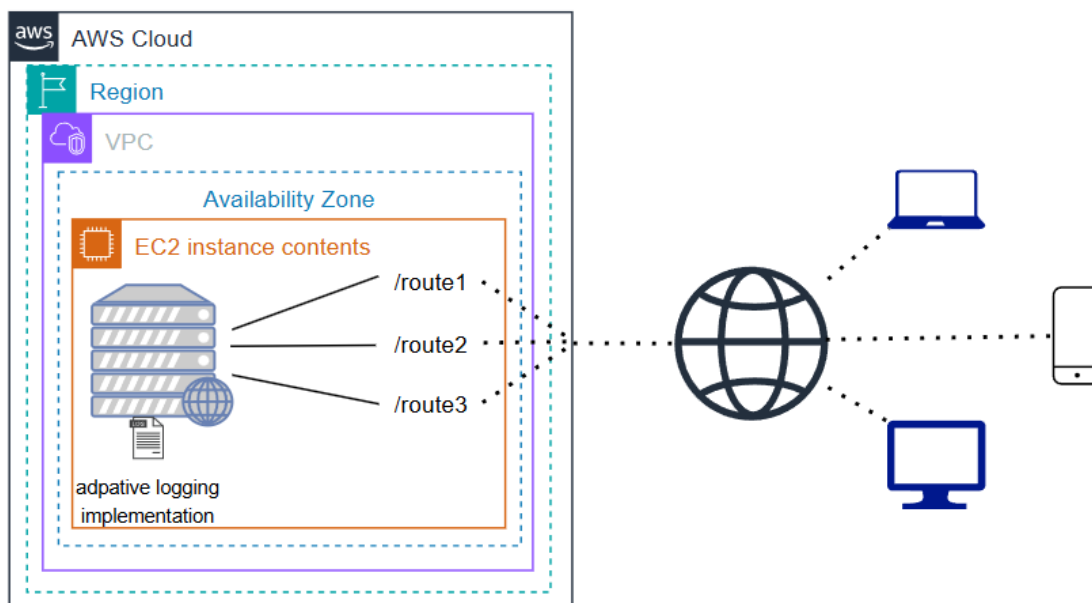


Рисунок 4.2 – EC2 інстанс з веб-сервером та імплементацією методу адаптивного логування - публічна сторона (результати отримані автором самостійно)

Типовим підходом при роботі в хмарі Amazon є використання сервісу Elastic Load Balancing [97] для розподілу навантаження між декількома серверами, перенаправляючи запити до відповідних комп'ютерів з метою досягнення очікуваної контрольованості та швидкодії, а також для контролю стану серверів системи. Втім, в межах розробки архітектури для прикладного розгортання методу адаптивного логування даний сервіс не є необхідним, оскільки інстанс веб-сервера буде лише один і обслуговування його роботи при довготривалому виконанні завдань не є пріоритетом. Тому пропонується використовувати прямий доступ до EC2 інстансів зі сторони публічних клієнтів. Для реалізації цього необхідно коректно налаштувати security groups [98], що по суті є фаєрволом із можливістю збереження стану, який дозволяє задати перелік правил для вхідного та вихідного трафіку, відповідно до яких відбувається перевірка мережевих запитів. Для демонстрації можливостей механізму адаптивного логування пропонується взяти узагальнений вигляд публічної

сторони веб-серверу та зробити його доступним на загальновідомих портах 80 та 443 будь-кому в мережі Інтернет.

Відповідно до рис. 4.1, окрім публічного каналу зв'язку, доступного із мережі Інтернет для широкого кола користувачів, додатково має існувати можливість більш захищеного доступу для користувачів з адміністративними правами та спроможністю реініціалізовувати конфігурацію методу адаптивного логування. Типовим рішенням для створення з'єднання із віддаленою машиною є Secure Shell [99] – популярне програмне рішення в площині інформаційної безпеки, що працює за рахунок автоматичного шифрування даних, що виходять в мережу Інтернет, та автоматичного розшифрування при досягненні очікуваного отримувача (з мінімальним впливом на власне користувачів, які можуть навіть не підозрювати що їх дані шифруються при передачі). SSh працює за принципом клієнт-серверної архітектури, де програма-сервер встановлюється на систему, доступ до якої необхідно регламентувати та контролювати, а програма-клієнт – на систему, якій необхідно створити захищене з'єднання із сервером. Механізми цього протоколу допомагають забезпечити аутентифікацію для взаємодіючих сторін, обриваючи з'єднання якщо підтвердження особи не було надано або було надано невірне. Також, як було згадано, інформація шифрується при передачі мережею та а цілісність переданих даних контролюється, попереджуючи можливість того, що невідома третя сторона їх модифікувала чи підробила. В Amazon Web Services існує можливість використання публічного ключа із пари SSh ключів для автоматичного встановлення з'єднання із відповідним інстансом, що є частиною сервісу EC2 та називається Key Pair [100]. Втім її використання має певні недоліки. Перш за все для підключення з використанням клієнт-серверної архітектури має існувати доступний з мережі Інтернет порт, на якому відбудеться з'єднання. Цей порт не обов'язково має бути загальнодоступний і трафік до нього може фільтруватись, наприклад, забороняючи доступ із всіх IP адрес, крім визначених, проте навіть так – збільшує поверхню для потенційної атаки (Denial of service чи подібне). Також за замовчуванням у пари SSh ключів не існує поняття “терміну життя”, а тому вже додана пара може використовуватись необмежену кількість разів.

Нарешті, окрім можливостей аутентифікації, шифрування та перевірки цілісності, які дає протокол Secure Shell, інколи існує необхідність переглядати історію сесій користувачів чи налаштовувати доступ більш точно. В межах хмари AWS існує сервіс, що дає більш підходящі спроможності для віддаленого з'єднання із EC2 інстансами.

AWS Systems Manager є сервісом для менеджменту та автоматизації операційних задач із багатьма хмарними сервісами провайдера, виступаючи при цьому централізованим хабом для контролю та налаштування інфраструктури та зменшуючи при цьому кількість операційних ресурсів, необхідних для роботи [101]. Використання даного сервісу, а саме його функціоналу під назвою Run Command, дозволяє адміністраторам віддалено виконувати команди на виділених інстансах не вимагаючи встановлення SSH чи Remote Desktop Protocol (RDP) з'єднання. Це, в свою чергу, підвищує безпеку та операційну швидкість системи, зменшуючи поверхню для атаки, а також дозволяє управляти багатьма інстансами одночасно. Іншою можливістю Systems Manager-а є утиліта під назвою Session Manager [102], що дозволяє управляти інстансами в Elastic Compute Cloud інтерактивно через вікно веб-браузера або через інтерфейс командного рядка. Як і у випадку з Run Command, відсутня необхідність відкривати порти чи обмінюватись SSH ключами, і для встановлення з'єднання на віддалену машину необхідно встановити Session Manager Agent – спеціальне програмне забезпечення, що опрацьовує запити від сервісу Systems Manager та робить відповідні зміни на інстансі, що, у випадку використання Amazon Linux 2023 AMI, є базовим компонентом превстановленим за замовчуванням. Після під'єднання адміністратор отримує доступ до інтерфейсу, що загалом дуже подібний до звичайного інтерфейсу командного рядка (термінал на Linux-орієнтованих системах) та набору стандартних утиліт для роботи з файлами чи процесами, відповідно до чого існує можливість оновити конфігурацію для імплементації методу адаптивного логування і подати сигнал для запуску функції реініціалізації.

Одним із фундаментальних архітектурних рішень методу адаптивного логування є упущення деталей стосовно задачі аутентифікації та авторизації з

метою спрощення власне самого методу та використання більш зрілих рішень, зокрема із арсеналу провайдерів хмарних послуг. У випадку із звичайним SSh, розділення доступу може відбуватись, наприклад, за допомогою конфігураційних параметрів `AllowUsers` та `AllowGroups` файлу `/etc/ssh/sshd_config` [103], в значенні яких можна вказати перелік користувачів чи груп (або текстових патернів), відповідно до яких буде вирішено, чи може користувач розпочати сеанс віддаленого доступу. Session Manager в свою чергу працює базуючись на системі менеджменту користувачів, що існує окремо від конкретних сервісів на рівні всієї хмари під назвою Identity and Access Management, скорочено IAM [104]. Завдяки цьому можна як задавати під яким саме користувачем операційної системи буде відбуватись з'єднання через Session Manager, що в загальному рівноцінно заходам безпеки SSh, але також можливо розмежовувати доступ на рівні власне сутності IAM користувача без необхідності вносити зміни безпосередньо в записи в операційній системі інстанса, до якого має відбуватись підключення. Відповідно, розмежування та контроль доступу для адміністраторів цілком може бути реалізований за допомогою Session Manager-а.

Відповідна архітектура матиме наступний вигляд:

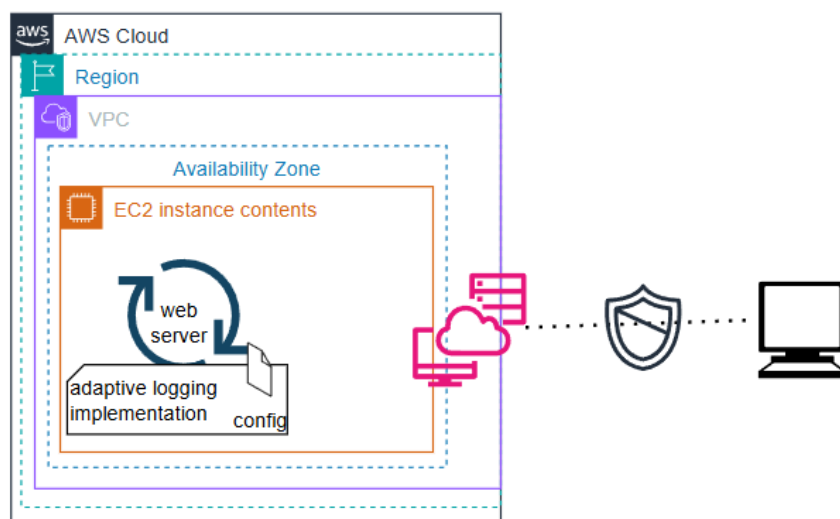


Рисунок 4.3 – EC2 інстанс з веб-сервером та імплементацією методу адаптивного логування - приватна сторона (результати отримані автором самостійно)

Далі пропонується розглянути програмні компоненти реалізації необхідні для успішного розгортання системи на АМІ.

4.3 Особливості програмної реалізації методу адаптивного логування

При реалізації архітектури динамічного варіанту повідомлень з метою використання можливостей виконання програмного коду, що згенерований під час виконання програми, було обрано мову програмування Javascript. Важливо зазначити, що історично дана мова програмування була розроблена в першу чергу для клієнтського середовища веб-браузера [105] і довгий час була обмежена виключно середовищем на комп'ютері клієнта, що хотів відвідати веб-сайт, а арсенал можливостей зводився, наприклад, до маніпуляцій з характеристиками вікна веб-браузера, таких як висота, ширина, позиція або звернення до властивостей браузера через окрему Browser Object Model (BOM). Втім у 2009 році Раян Дал (англ. Ryan Dahl) представив свою розробку під назвою NodeJS [106], що представляло собою комбінацію із середовища виконання Javascript від Google під назвою V8, програмну реалізацію механізму циклу подій та програмний механізм для взаємодії з низькорівневим інтерфейсом введення/виведення, що дало можливість перенести мову з браузерів на сервери та почати створювати програмні продукти з її використанням, в тому числі і веб-сервери. На відміну від деяких інших мов програмування, що використовуються для написання серверів, Javascript в межах платформи NodeJS одразу ж по запуску програми створює інстанс циклу подій, який керує всім процесом виконання опрацьовуваного коду (на противагу підходу, де для кожної окремої операції, як от підключення клієнта, виділяється окремий потік операційної системи). В межах цієї роботи цикл подій опрацьовує події із “черги подій” та за необхідності виносить виконання задач (таких як робота із вводом-виводом чи мережею, виконання ресурсоємних операцій, тощо) в заздалегідь виділений пул потоків, додаючи спеціальний програмний код під назвою колбек (від англійського “call back”), який буде виконаний по

завершенню операції щоб повідомити про її результат. Загальну схему роботи циклу подій можна побачити на рис. 4.4.

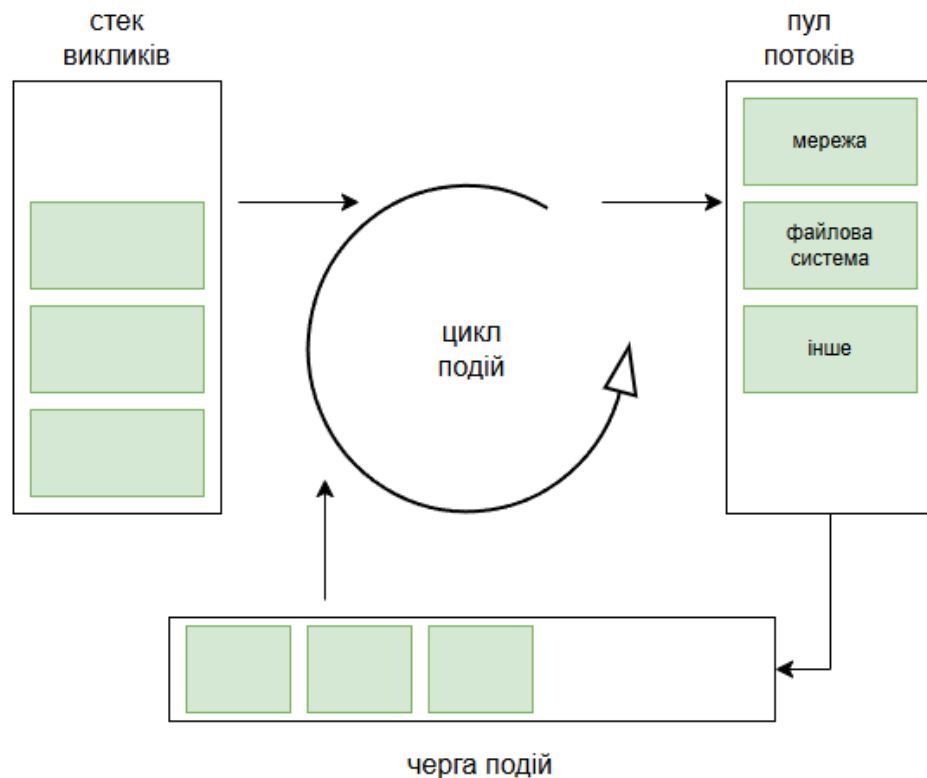


Рисунок 4.4 – Узагальнена схема роботи циклу подій платформи NodeJS (відповідно до [106])

Завдяки цьому досягається використання меншої кількості ресурсів порівняно із підходом з відокремленими потоками на кожну задачу, а також імплементація методу адаптивного логування отримує технічну можливість звернутись до циклу подій та змінити конфігурацію Синглтона логування не зупиняючи процес операційної системи, в межах якого відбувається виконання, та зберегти активний стан пам'яті. Через це в межах даної роботи платформою для написання веб-сервера пропонується взяти NodeJS.

Набір стандартних модулів (англ. core modules) платформи дозволяє писати програми по типу веб-серверів користуючись доволі низькорівневим (порівняно з використанням спеціалізованих бібліотек), проте зручним набором інструментів. Втім такий підхід важко було би назвати оптимальним і для потреб прикладної реалізації в межах поточної роботи було би зручніше взяти рішення, що має вищий ступінь готовності. В межах екосистеми Javascript існує публічний

репозиторій готових рішень, під назвою npm [107]. Завдяки цьому найбільшому репозиторію програмних рішень в світі розробники програмного забезпечення з відкритим вихідним кодом мають змогу ділитись власними та використовувати бібліотеки, написані іншими. Однією з таких бібліотек є ExpressJS [108]: мінімалістичний та гнучкий фреймворк для створення веб-застосунків із зручним набором компонентів. Основний принцип його роботи базується на ідеї проміжних компонентів програмного забезпечення (англ. *middleware*), що дозволяють доволі просто та гнучко додавати необхідні HTTP ендпоінти та писати обробку запитів (в тому числі із додаванням методу адаптивного логування). Саме тому для функціоналу веб-серверу більш доцільно було би використати дану бібліотеку. Для частини, що буде пов'язана із процесом реініціалізації синглтону адаптивного логування, стандартні модулі, а саме модуль взаємодії з файловою системою “fs” та глобальна змінна “process”, дозволяють написати необхідний функціонал, що буде очікувати на відповідний міжпроцесовий сигнал (наприклад, SIGUSR2), зчитуватиме та перевірятиме оновлену конфігурацію з файлу та застосовуватиме її.

Для імплементації синглтону методу адаптивного логування пропонується використати клас, сигнатуру якого можна побачити на рис. 4.5

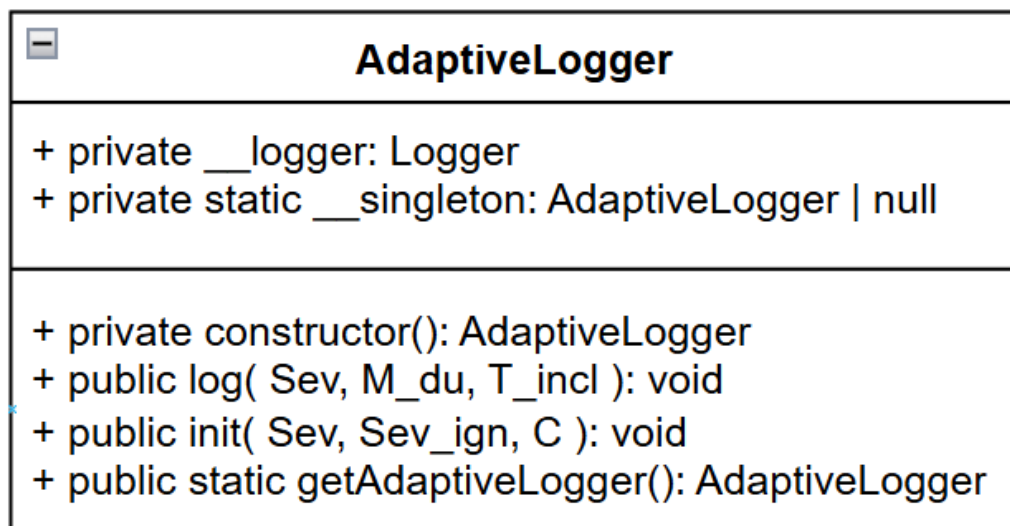


Рисунок 4.5 – Сигнатура класу-імплементації методу адаптивного логування (результати отримані автором самостійно)

Відповідно до оновлених формальних визначень із розділу 3, функція `log` є імплементацією (3.1), позначеною модифікатором `public` так як це загальнодоступний метод синглтону, та із “порожнім” результуючим значенням, оскільки результат логування можна буде побачити в загальному потоці виводу. Аналогічно функція `init` є імплементацією (2.3) з конфігураційним словником (3.7). Пусте результуюче значення в цьому випадку є більш повним описом того, що відбудеться при використанні методу, оскільки пророблені дії не будуть видимі ззовні, а лише змінять внутрішній стан синглтону. Решта методів та членів класу є допоміжними, оскільки не мають відповідної формалізації. Приватний член класу `__logger` є компонентом, що додає можливості реального логування, адже саме він відповідає за формування відповідних викликів, щоб повідомлення з’явилося в потоці виводу системи. Як було згадано раніше, додавання цієї логіки в межах опису методу адаптивного логування є комплексною задачею, і за можливості бажано взяти існуючу імплементацію. Так, для середовища Javascript та менеджера пакетів `npm` таким рішенням може слугувати `winston` [109] – бібліотека для логування, що розроблена з метою бути простим та універсальним інструментом з можливістю підтримки багатьох транспортів (тобто кінцевих точок для виводу логів, від стандартного виводу до віддалених мережевих дисків).

Наступним важливим аспектом є використання приватного конструктора для екземплярів класу. Це означає, що зовнішні користувачі не зможуть створити екземпляри `AdaptiveLogger` самотійно і єдиний спосіб отримати об’єкт з таким інтерфейсом це використати статичний метод `getAdaptiveLogger`. Маючи цю “єдину точку входу” код класу має змогу відслідковувати всі виклики до функціоналу створення синглтону та або віддавати існуючий, або один єдиний раз створити новий. Для зберігання початкового стану, або вже створеного інстансу доцільно використовувати приватний статичний член класу `__singleton`. Важливо відмітити, що так само як для індикації, що в (2.3) розробник не зацікавлений в параметрі найменш важливого рівня критичності, починаючи з якого можна ігнорувати механізм тегування, ми можемо використовувати спеціальне значення `null` в Javascript, аналогічно це саме значення

використовується як індикатор пустоти (тобто відсутності попередньо створеного) значення поля `__singleton`.

Щоб встановити NodeJS на операційну систему можна використати підхід з використанням прескомпільованих бінарних файлів, що існують у вільному доступі для всіх популярних систем (Windows, Linux та MacOS). Втім, такий підхід має певні нюанси, пов'язані із природою роботи 4х середовищ розміщення системи на різному етапі готовності: так для середовища `local` розміщення відбувається на персональному комп'ютері розробника і є доволі гнучким та з можливістю легко вплинути та виправити неточності, що можуть виникнути в процесі встановлення, тоді як для середовищ `dev`, `stage` та `prod` встановлення буде відбуватись на віддалених серверах, де рівень гнучкості вже відчутно менший. І хоча для методу адаптивного логування в цілому не є необхідним будь-яка взаємодія із середовищами `local` та `prod`, архітектура розгортання навіть на одному беззаперечно впливатиме на решту і тому уніфікація підходу є очікуваним та вірним вибором. Щоб зменшити необхідну кількість зусиль для розгортання системи пропонується використати підхід контейнеризації на базі технології під назвою Docker [110]. По своїй суті Docker є платформою, що дозволяє зібрати, відправити та запустити програмний продукт практично на будь-якій операційній системі, від якої вимагається лише надати узагальнений базовий функціонал для самої технології, тоді які всі деталі роботи того чи іншого застосунку приховані. Завдяки цьому різниця між середовищами, локальними чи віддаленими, стає значно меншою і вирішити значну частину проблем, пов'язаних із розгортанням програмного забезпечення, можна лише раз, в майбутньому перевикористовуючи готові компоненти. За своєю природою технологія дуже схожа на віртуалізацію (та віртуальні машини), втім є більш легковісною, що в свою чергу спрощує та пришвидшує використання та розгортання (детальніше порівняння двох технологій зображено на рис. 4.6).

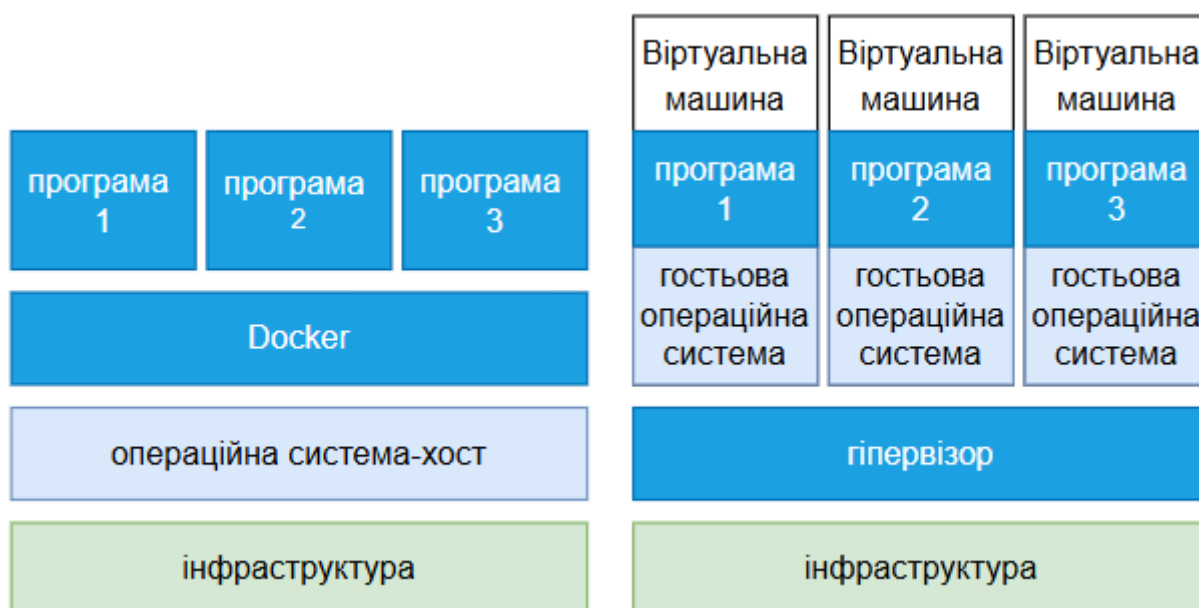


Рисунок 4.6 – Порівняння контейнерів та віртуальних машин (відповідно до [110])

Хоча технологія віртуалізації дає більший рівень ізоляції за рахунок використання повноцінної операційної системи, перевикористання більшої кількості можливостей операційної системи-хоста дозволяє в результаті мати більш ресурсоефективні артефакти. Це в свою чергу дає змогу описувати бажану конфігурацію контейнера за допомогою спеціалізованого файлу Dockerfile, що по суті являє собою інструкцію та одночасно перелік операцій, які необхідно здійснити, щоб відтворити середовище розгортання програмного продукту. З цих інструкцій збирається певний шаблон, з якого потім буде створено реальний інстанс для запуску в процесі виконання, під назвою image. Саме завдяки цим шаблонам всі 4 середовища для розгортання веб-серверу із імплементацією методу адаптивного логування можуть перевикористати вже готове рішення із більш раннього етапу розробки, тим самим зменшивши площу для помилки при створенні компонентів, які могли виникати, якщо би інфраструктура потребувала різних процесів для запуску. Додатково до цього можна бути впевненим, що робота із механізмом міжпроцесових сигналів, яка необхідна для механізму реініціалізації, буде однаковою та всі неточності будуть усунуті на початкових етапах.

Одним із аспектів, який потребуватиме коректив, є взаємодія із файловою системою (хоча це пройде абсолютно непомітно для Javascript коду і стосуватиметься лише процесу роботи адміністратора). Оскільки контейнери певною мірою ізолюють середовище виконання від системи-хоста, доступ до файлів теж певною мірою ізолюваний: за замовчуванням зміни зроблені всередині контейнера невидимі зовні. Втім, для реініціалізації та застосування нової конфігурації користувач-адміністратор має оновити вміст файлу з конфігураційним об'єктом. Рішенням цього є механізм прокидування файлів за допомогою volume-ів: при створенні контейнеру розробник може вказати який файл із системи-хоста (та в якому режимі, наприклад “лише читання”, щоб запобігти небажаному запису із середини ізолюваного середовища) розмістити під яким шляхом в контейнері. І тоді, змінивши його вміст через звичні засоби роботи з файлами (vi, vim, nano, тощо), веб-сервер зможе підхопити ці зміни із середини.

4.4 Особливості реалізації методу адаптивного логування з використанням хмарного провайдера Amazon Web Services

Процедуру розгортання доцільно розпочати із створення інстансу EC2. Для цього в панелі адміністратора (під назвою AWS console) існує спеціалізована сторінка, що дозволяє вибрати різні частини майбутнього серверу. Користувач має змогу вказати ім'я, вибрати необхідний Amazon Machine Image (як було вказано раніше, в межах даної роботи буде використано Amazon Linux 2023, що можна побачити на рис. 4.7), вибрати тип інстанса, що впливатиме на кількість виділеного процесорного часу та обсяг доступної оперативної пам'яті, обрати обсяг пам'яті запам'ятовуючого пристрою, тощо. Оскільки в переважній більшості ці параметри не впливають на можливість демонстрації принципів роботи методу адаптивного логування, їх можна взяти за замовчуванням. Втім, один компонент початкових налаштувань - instance profile - необхідно встановити в спеціальне значення, що дозволить використовувати Session Manager для управління сервером. За своєю суттю дана властивість визначає

дозволи, які матиме результуючий сервер при взаємодії з іншими елементами хмари AWS. Для того, щоб дозволити використання підключень через сервіс менеджера сесій необхідно вказати IAM роль, що володітиме стандартним дозволом під назвою “AmazonSSMManagedEC2InstanceDefaultPolicy”.

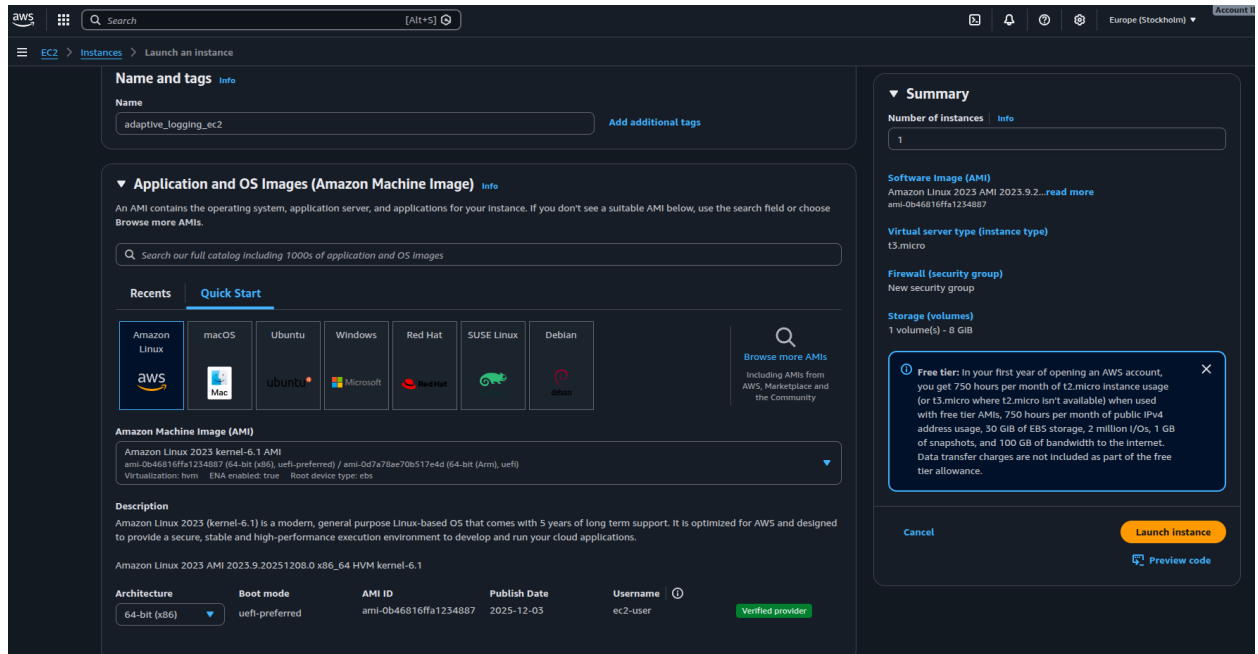


Рисунок 4.7 – Початкові параметри створення EC2 інстанса, включно з АМІ (результати отримані автором самостійно)

Наступним кроком є налаштування програмного фаєрволу під назвою security groups, щоб дозволити приєднання із мережі Інтернет до веб-серверу. При створенні даної групи важливо вказати два вхідних правила для HTTP та HTTPS портів 80 та 443 відповідно, що дозволяють трафік з будь-яких IP адрес (рис. 4.8).

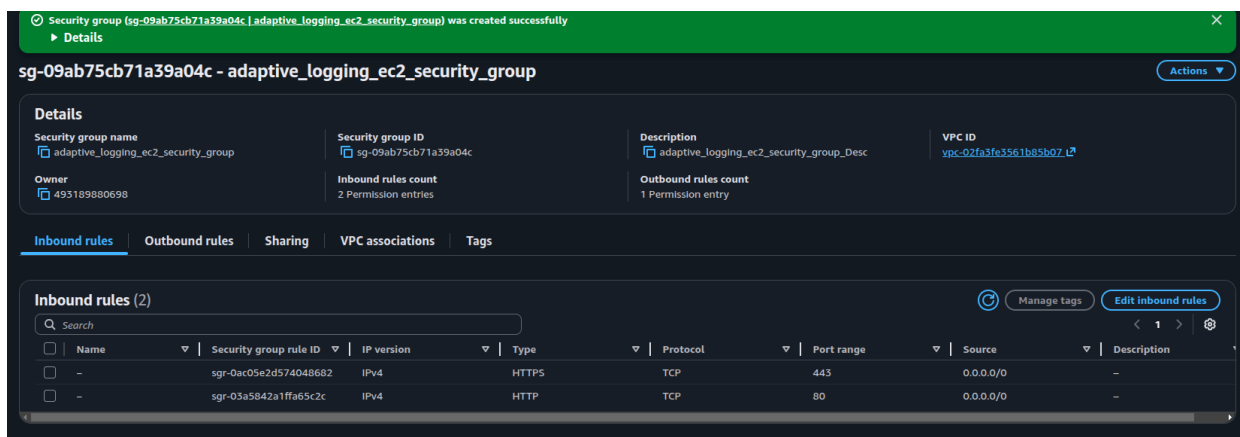


Рисунок 4.8 – Створення security group (результати отримані автором самостійно)

Далі дану групу необхідно приєднати до нового інстансу через відповідну властивість на екрані створення і ініціалізація може вважатись завершеною. Після декількох хвилин необхідних для запуску програмного забезпечення користувач отримує змогу через спеціальний варіант меню “Session manager”, який можна побачити натиснувши кнопку “Connect” в панелі керування сервером, приєднатись до терміналу операційної системи. Подальше встановлення необхідних пакетів (таких як Docker та git) є доволі буденним процесом тому не потребує ілюстрації. Після завантаження та розгортання коду із веб-сервером, розробник може спробувати отримати доступ із мережі Інтернет на 80 порту і якщо налаштування пройшло успішно – отримає очікувану відповідь (наприклад у вікні браузера). Для демонстрації можливості переключення рівня критичності та тегів без перезавантаження активного процесу було піднято два ендпоінти “/auth” та “/profile”, що емулюють реальний функціонал та логування процесу виконання. Кожен із них містить лог-виклики рівнів “debug” та “info”, та додатково відповідний ендпоінт містить тег, що відповідає його шляху без слеша (“/auth” -> “auth”). Для зчитування логів, що генерує веб-сервер, необхідно підключитись до контейнера командою “docker logs -f <container name>”. Початкова конфігурація синглтону адаптивного логування задана так, що виведення відбудеться для будь-якого лог-виклику із рівнем критичності вище чи рівний “debug”, а перевірка оновленого конфігураційного словника відбувається шляхом переходу на обидва ендпоінти в веб-браузері. Відповідно, на рис. 4.9 перші 6 рядків після повідомлення про початок роботи веб-сервера (“Example app listening on port 80”) відповідають повному набору всіх лог-викликів для обох ендпоінтів. Для реініціалізації конфігураційного словника необхідно змінити вміст відповідного файлу та відправити потрібний міжпроцесовий сигнал USR1, що відбувається з використанням команди “kill -s 10 1” та означає “відправити міжпроцесовий сигнал номер 10 процесу з ідентифікатором 1”. На рис. 4.9 цьому моменту відповідає перший рядок “adaptive logging init”.


```

[root@ip-172-31-47-233 logger_web_server_test]# docker logs -f logger_web_server-test-web_server-1
adaptive logger init
Example app listening on port 80
{"T_incl":["auth"],"level":"debug","message":"/auth started step 1","timestamp":"2026-01-04T16:33:33.736Z"}
{"T_incl":["auth"],"level":"debug","message":"/auth started step 2","timestamp":"2026-01-04T16:33:33.757Z"}
{"T_incl":["auth"],"level":"info","message":"/auth about to respond","timestamp":"2026-01-04T16:33:33.779Z"}
{"T_incl":["profile"],"level":"debug","message":"/profile started step 1","timestamp":"2026-01-04T16:33:42.162Z"}
{"T_incl":["profile"],"level":"debug","message":"/profile started step 2","timestamp":"2026-01-04T16:33:42.183Z"}
{"T_incl":["profile"],"level":"info","message":"/profile about to respond","timestamp":"2026-01-04T16:33:42.203Z"}
adaptive logger init
{"T_incl":["auth"],"level":"info","message":"/auth about to respond","timestamp":"2026-01-04T16:34:22.920Z"}
{"T_incl":["profile"],"level":"info","message":"/profile about to respond","timestamp":"2026-01-04T16:34:26.399Z"}
adaptive logger init
{"T_incl":["auth"],"level":"info","message":"/auth about to respond","timestamp":"2026-01-04T16:35:09.704Z"}

```

Рисунок 4.9 – Термінал-консоль Session Manager-а з виводом лог-повідомлень досліджуваного веб-сервера (результати отримані автором самостійно)

Як можна побачити, при переході на обидва ендпоінти ми спостерігаємо суттєво менше виводу. Наступна реініціалізація включає в себе вказання лише тегу “auth” як такого, що має бути обов’язково присутній в параметрах лог-виклику. Після застосування оновлення перехід на ендпоінт “/profile” взагалі не генерує жодних лог-повідомлень.

4.5 Приклад практичної переваги методу адаптивного логування над базовим підходом до логування

Для демонстрації практичної переваги при використанні методу адаптивного логування у порівнянні з базовим підходом, що орієнтований лише на критичність, пропонується розглянути використання обох в контексті веб-серверу. Для цього створено екземпляр веб-серверу з базовим функціоналом, що складається із трьох ендпоінтів: “/auth”, “/profile” та “/dashboard”. Логіка роботи даних ендпоінтів написана таким чином, що при опрацюванні запиту можливо використати як імплементацію базового алгоритму логування, так і методу адаптивного логування. При нормальному сценарії опрацювання запиту кожен ендпоінт генерує лог-повідомлення двох рівнів критичності “debug” та “info”, та додатково позначає відповідні виклики власним унікальним тегом, що рівний назві ендпоінта без початкового слеша. Оскільки виняткові ситуації це невід’ємна частина функціонування будь-якої системи, додано фактор

випадковості у відповідності до якого будь-який ендпоінт із вірогідністю 5% може згенерувати помилку, яку веб-сервер здатний відловити та коректно опрацювати. В реальному житті програмне забезпечення веб-серверів має характерний період ініціалізації, під час якого встановлюється з'єднання із базою даних, перевірка доступності необхідних сервісів, реєстрація в мережі, зчитування налаштувань, тощо. Для його емуляції в тестовому середовищі використовується штучна затримка в 3 секунди. Аналогічним чином емулюється період деініціалізації, що відповідає процесу закриття активних з'єднань, очистки ресурсів, тощо при вимкненні веб-сервера, та штучно чекає 1 секунду. Для емуляції користувацького трафіку на ендпоінти системи щосекунди генерується 50 запитів (по одному від унікального віртуального користувача з випадковим вибором ендпоінту) протягом 15 секунд, з додатковим обмеженням, що після запиту користувач очікує 1 секунду перед здійсненням наступного. Таким чином за 15 секунд система отримує сумарно 750 запитів від користувачів, що є певним наближенням відносно того, як би система діяла під помірним користувацьким навантаженням.

Основним сценарієм, який необхідно дослідити, є зміна деталізації (та фокусу, якщо можливо) компоненту логування в процесі активної роботи системи. На початку роботи рівень критичності встановлений в значення “info”, але в певний момент відтворюючи той випадок, ніби адміністратор системи хоче побачити більш деталізовану інформацію про певний компонент системи, ендпоінт “/auth”, відбувається перемикання. У випадку базового підходу до логування перемикання відбувається за рахунок перезавантаження серверу із оновленою конфігурацією, а у випадку адаптивного методу логування – відбувається перезавантаження лише конфігурації синглтону. Щоб мінімізувати вплив людського фактору на точність результатів, процес запуску системи, створення штучного навантаження, перемикання рівня критичності, вимкнення веб-серверу та агрегації лог-записів системи працюватиме автоматично через скрипт. Пропонується розглянути наступні сценарії:

1. базовий підхід до логування, рівень критичності весь час “info”
2. базовий підхід до логування, рівень критичності весь час “debug”

3. базовий підхід до логування, рівень критичності перемикається з “info” на “debug”
4. метод адаптивного логування, рівень критичності перемикається з “info” на “debug”
5. метод адаптивного логування, рівень критичності перемикається з “info” на “debug”, додається фільтрація по ендпоінту “/auth”

Результати роботи при різних сценаріях наведені в таблиці 4.2.

Таблиця 4.2 – Результати обчислювального експерименту

№ сценарію	H_{200}	H_{500}	D_{fail}	L	L_{auth}	L_{err}
1 (base, Sev = info, no reinit)	712	38	0	1462	496	38
2 (base, Sev = debug, no reinit)	707	43	0	3578	1116	43
3 (base, Sev = info -> debug)	522	28	200	1408	424	24
(adp, Sev = info -> debug)	713	37	0	2615	973	37
(adp, Sev = info -> debug, filter “auth”)	717	33	0	1382	941	33

H_{200} – кількість успішних запитів, H_{500} – кількість помилкових запитів, які можна опрацювати, D_{fail} – кількість помилкових запитів, які неможливо опрацювати, L – загальна кількість згенерованих лог-повідомлень, L_{auth} – кількість лог повідомлень ендпоінту “/auth”, L_{err} – кількість лог-повідомлень про помилкові запити, які можна опрацювати.

Одразу можна відзначити, що сценарій 3 є єдиним, який провокує появу помилок, які неможливо відловити. Це пов’язано із тим, що на період перезапуску веб-сервера ніякий запит клієнта не може бути опрацьований. Додатково можна побачити, що 4 лог-записи про помилкові ситуації були втрачені, оскільки реально помилкових ситуацій було 28 (тобто було втрачено 14% помилкових запитів). Використання адаптивного методу не вимагає перезапуску і тому проблема із неопрацьованими запитами через перезапуск по факту зникає.

Відповідно до випадкової природи вибору ендпоінту віртуальним користувачем, ймовірність випадання одного з трьох ендпоінтів близька до

третини. В цьому можна пересвідчитись, якщо порахувати яку частку складають логи про ендпоінт “/auth” від загальних в сценаріях 1 – 4, що складає 34%, 31%, 30% та 37% відповідно. Однак у сценарії 5 ми бачимо частку рівну 68%, що означає, що для нашої задачі “відлагодити ендпоінт /auth” у даному сценарії необхідно відкинути на 31% менше “зайвих” лог-записів (тобто таких, що не стосуються нашого досліджуваного ендпоінту). Ще важливо зазначити, що в сценарії 5 не відбулось втрати лог-записів про помилкові ситуації, що робить такий варіант більш інформативним, порівняно із сценарієм 3, коли така втрата була.

Також попарно порівнюючи сценарії 2 і 3 та 2 і 5 (сценарій 2 доцільно охарактеризувати як найбільш деталізований опис роботи системи) можна побачити, що у випадку сценарію 3 необхідно опрацювати на 60.6% менше рядків порівняно із сценарієм 2 (однак є втрачені запити), а у випадку сценарію 5 необхідно опрацювати на 61.3% менше (і втрачені запити відсутні).

Висновки до розділу 4

За підсумками роботи в даному розділі формальні основи методу адаптивного логування в його формі з динамічним варіантом повідомлення було розгорнуто з використанням реальних обчислювальних потужностей на прикладі веб-серверу, що дало змогу перевірити як коректність формалізації, так і можливість реалізації інфраструктурних механізмів, таких як авторизація, за допомогою компонентів навмисно винесених за межі формальних визначень методу. Робота велась відповідно до наступних етапів:

- було окреслено основні вимоги до систем для яких має зміст використання методу (а саме багатокomпонентність та тривалість існування основного процесу), відповідно до чого було запропоновано використати веб-сервер для створення прикладної реалізації та наведено загальну архітектуру застосування із виокремленням двох частин, публічної та приватної; з точки зору фізичних ресурсів була розглянута можливість розгортання на власних ресурсах розробника чи компанії

розробника у порівнянні з використанням послуг провайдерів хмарних обчислювальних потужностей, і як наслідок запропоновано використання рішень, орієнтованих на хмарну архітектуру;

- при узагальненому порівнянні трьох найбільших провайдерів хмарних послуг Microsoft Azure, Google Cloud та Amazon Web Services за умови того, що технічно всі три надають достатньо спроможностей для імплементації методу, через попередній досвід використання автором даної платформи було обрано AWS; в межах хмари AWS описано основні частини Virtual Private Cloud, виокремлені та описані деталі функціонування віртуальних серверів EC2 та їх компонентів, наприклад security groups, що необхідні для розгортання веб-сервера; в межах задачі реалізації обмеженого доступу для користувачів з адміністративними повноваженнями запропоновано використати сервіс Session Manager та сервіс Identity and Access Management для більш зручного налаштування відповідних дозволів;
- як платформу для написання власне програмного забезпечення запропоновано взяти NodeJS (за рахунок того, що робота над адаптивним методом із динамічним варіантом повідомлення вже використовувала Javascript), продемонстровано як базовий компонент роботи платформи (event loop) дозволяє за потреби виконати процедуру реініціалізації; для написання базового функціоналу веб-серверу пропонується скористатись екосистемою програмного забезпечення з відкритим вихідним кодом та використати бібліотеку ExpressJS; представлено та описано основні методи та поля програмної реалізації синглтону методу адаптивного логування з їх сигнатурами та очікуваним алгоритмом роботи; для простоти розгортання з огляду на декілька пов'язаних середовищ запуску програмного коду додатково використана технологія контейнеризації Docker;
- описані кроки практично застосовані в межах користувацького інтерфейсу хмарного провайдера AWS із фіксацією основних моментів: створення EC2 інстансу, налаштування доступу з мережі Інтернет та

взаємодія із інтерфейсом командного рядку через можливості Session Manager-a; на прикладі двох ендпоінтів веб-сервера продемонстровано, як метод адаптивного логування за допомогою механізму міжпроцесових сигналів дозволяє без перезапуску процесу та втрати стану оперативної пам'яті змінити як рівень критичності, що використовується при логування, так і перелік тегів, відповідно до яких відбувається фільтрація при виводі;

- відповідно до результатів обчислювального експерименту було зафіксовано підвищення інформативності зібраного масиву лог-повідомлень на 31% при дослідженні компоненту системи з тегом-ідентифікатором “/auth” (при повному збереженні всіх записів про помилкові події в інших компонентах системи), а також відмічено вирішення проблеми недоступності системи в процесі реініціалізації, яка спостерігалась при використанні базового підходу до логування та спричиняла втрату 14% запитів;
- матеріали розділу опубліковані в [6].

ВИСНОВКИ

В дисертаційній роботі вирішено важливу науково-технічну задачу, яка полягає в підвищенні рівня спостережності та оперативності контролю роботи комп'ютерних систем та програм за рахунок розробки та впровадження методу адаптивного логування.

1. Відповідно до проведеного аналізу та формалізації базового підходу до логування, що базується лише на критичності, були розроблені моделі сигнатур адаптивного логування, що складаються із функції логування з ширшим переліком параметрів порівняно з базовим підходом до логування, а також із функції реініціалізації, яка за допомогою параметру конфігураційного словника дає можливість більш точно та гнучко налаштувати результуючі лог-повідомлення; архітектуру програмної реалізації імплементації методу адаптивного логування пропонується будувати на основі патернів програмування Синглтон та Фасад; за результатом порівняння різних методів передачі повідомлень в комп'ютерних системах було обрано механізм міжпроцесових сигналів як найбільш гнучкий та широкодоступний спосіб, скористатись яким можна в багатьох мовах програмування, платформах та середовищах.
2. Вперше побудовано метод адаптивного логування на основі логування повідомлень шляхом застосування моделей сигнатур та динамічного варіанту повідомлень. Додавання динамічного варіанту повідомлень, що базується на можливості сучасних мов програмування формувати виконуваний код вже в процесі роботи програми, зумовило більш гнучкий спосіб взаємодії з програмними компонентами, аспекти роботи яких не фіналізовані та можуть потребувати різного рівня деталізації в процесі розробки та відладки. Дані результати забезпечили підвищення рівня спостережності та оперативності сигналювання про непередбачувані та критичні ситуації в роботі систем та програм. Для попередження використання небажаних синтаксичних структур, операторів чи звернення до непередбачуваних елементів процесу

виконання моделі сигнатур адаптивного логування були додатково доповнені підсистемою валідації тіл динамічних функцій на основі аналізу абстрактних синтаксичних дерев вихідного коду з використанням JSON-схем.

3. Отримали подальший розвиток методи контролю роботи програм для хмарних обчислень за рахунок впровадження методу адаптивного логування з використанням хмарних сервісів. Для застосування методу адаптивного логування в реальних умовах було створено модель та було розгорнуто імплементацію на потужностях провайдера хмарних послуг Amazon Web Services з використанням інфраструктури обчислювальних можливостей Elastic Compute Cloud та програмного забезпечення Session Manager сервісу Systems Manager, що дозволило підтвердити можливість реалізації аспектів аутентифікації в процесі адміністративного доступу за допомогою можливостей інфраструктури.
4. Практична цінність роботи полягає в доведенні розробленого підходу, до алгоритмів, функціональних схем і методики застосування, що в сукупності дозволяють отримати вищий рівень деталізації та інформативності стосовно процесів, які відбуваються під час виконання програмного коду. Запропоновані автором для використання модулі програмного забезпечення, були підібрані таким чином, щоб мати можливості для імплементації в якомога більшій кількості платформ, мов програмування та середовищ виконання програмного коду. Для отриманого за результатами дослідження варіанту методу адаптивного логування з динамічними повідомленнями розроблено ескізний зразок модуля впровадження на інфраструктурі провайдера хмарних послуг Amazon Web Services, із врахуванням аспектів належного рівня безпеки при взаємодії користувачів з адміністративним рівнем доступу, а також простоту та прогнозованість переносу артефактів готової системи між різними середовищами розробки. За результатами обчислювального експерименту при порівнянні з базовим алгоритмом логування, що спирається лише на критичність, було отримано збільшення

інформативності результуючих лог-записів на 31% у відповідності до виконання задачі відлагодження конкретного компоненту системи. Водночас окрема можливість вказати рівень критичності, вище якого застосування фільтрації є небажаним, дозволило зберегти всі повідомлення про помилки із решти компонентів системи, що не перебували у фокусі уваги. Порівнюючи два аналогічних сценарії реініціалізації було продемонстровано, що зменшення кількості надлишкових повідомлень у випадку адаптивного методу склало 61.3%, що на 0.7% більше, ніж для базового варіанту. Також втрата клієнтських повідомлень, що склала 14% при реініціалізації базового варіанту, була цілком відсутня у випадку з адаптивним методом, оскільки запропонована архітектура більш придатна для сценарію адаптації до змін у вимогах логування при активній роботі системи. Результати дисертаційного дослідження, а саме метод адаптивного логування з динамічним варіантом повідомлень та підсистемою валідації, були використані для контролю коректності розробки програмного забезпечення системи дистанційного управління наземним самохідним роботизованим комплексом виробництва ТОВ “МОРОЗ ТЕХ”.

Список використаних джерел

1. Suprunenko I., Rudnytskyi V. On specifics of adaptive logging method implementation. *Bulletin of Cherkasy State Technological University*. 2024. Vol. 29, no. 1. P. 36–42. DOI: 10.62660/bcstu/1.2024.36
2. Супруненко І. О., Бабенко В. Г., Рудницький С. В. Метод адаптивного логування розподілених комп'ютерних систем. *Технології розвитку безпілотних систем : кол. монографія / під ред. В. М. Рудницького. Черкаси : видавець Вовчок О. Ю., 2025. Т. 2. Безекіпажні системи. С. 135–156. ISBN: 978-617-8725-03-7. DOI: 10.5281/zenodo.19191122*
3. Suprunenko I., Rudnytskyi V. Comparison of message passing systems in context of adaptive logging method. *Visnyk of Kherson National Technical University*. 2024. No. 2 (89). DOI: 10.35546/kntu2078-4481.2024.2.32
4. Suprunenko I., Rudnytskyi V. Dynamic message variant in adaptive logging method. *Modern Information Security*. 2024. Vol. 59, no. 3. P. 94–99. DOI: 10.31673/2409-7292.2024.030010
5. Suprunenko I., Rudnytskyi V. Validation of dynamic message variant in adaptive logging method. *Measuring and Computing Devices in Technological Processes*. 2024. No. 4. P. 31–38. DOI: 10.31891/2219-9365-2024-80-5
6. Suprunenko I., Rudnytskyi V. Cloud based architecture for adaptive logging method implementation. *Cybersecurity: Education, Science, Technique : Electron. Prof. Sci. J.* 2025. Vol. 3, no. 27. P. 329–338. DOI: 10.28925/2663-4023.2025.27.724
7. Супруненко І. О. Адаптивний підхід до логування як новий вимір спостережності за прикладним програмним забезпеченням. *Інформаційна безпека та комп'ютерні технології : тези доп. VII Міжнар. наук.-практ. конф. до 30-річчя кафедри кібербезпеки та програмного забезпечення, м. Кропивницький / М-во освіти і науки України, Центральн. нац. техн. ун-т. Кропивницький : ЦНТУ, 2023. С. 45–46. URL: <https://kbpz.kntu.kr.ua/file/content/6621/2023-vii-mizhnarodna-naukovo-praktychnoi-konferentsiia-do-30-ty-richchia-kafedry-kiberbezpeky-ta-prohramnoho-zabezpechennia-informatsiina-bezpeka-ta-komp-yuterni-tekhnohii-.pdf>*

8. Suprunenko I. Message passing systems in modern applications and their role in adaptive logging method. Інформаційні технології в освіті, науці й техніці (ІТОНТ-2024) : матеріали VII Міжнар. наук.-практ. конф. Черкаси, 2024. С. 12–14. URL: https://knsa.chdtu.edu.ua/wp-content/uploads/2024/06/Conference-Proceedings-ITEST-2024_25_06.pdf

9. Suprunenko I., Rudnytskyi V. Dynamic source code processing approaches in context of adaptive logging method. Global Society in Formation of New Security System and World Order : матеріали III Міжнар. наук.-практ. інтернет- конф. Дніпро, 2024. С. 20–22. URL: <http://www.wayscience.com/wp-content/uploads/2024/07/Conference-Proceedings-July-4-5-2024.pdf>

10. Swedin E. G., Ferro D. L. Computers: The life story of a technology. Baltimore : Johns Hopkins University Press, 2007. ISBN: 0801887747, 9780801887741.

11. Digital 2026 Global Overview Report. DataReportal – Global Digital Insights. URL: <https://datareportal.com/reports/digital-2026-global-overview-report> (дата звернення: 18.06.2025).

12. Ala A., Yin J. L., Cussen C. A snapshot of digital transformation in hepatology and rare liver disease related health care: A UK perspective. Mayo Clinic Proceedings: Digital Health. 2023. Vol. 1, iss. 4. P. 451–454. DOI: 10.1016/j.mcpdig.2023.08.003

13. Barnes R. M. Bioinformatics for geneticists: A bioinformatics primer for the analysis of genetic data. John Wiley & Sons, 2007. ISBN: 0470059176, 9780470059173.

14. Almufarreh A., Arshad M. Promising emerging technologies for teaching and learning: Recent developments and future challenges. Sustainability. 2023. Vol. 15, issl. 8. P. 6917. DOI: 10.3390/su15086917

15. Alam M. R., Reaz M. B. I., Ali M. A. M. A review of smart homes – past, present, and future. IEEE Transactions on Systems, Man, and Cybernetics. Part C (Applications and Reviews). 2012. Vol. 42, iss. 6. P. 1190–1203. DOI: 10.1109/TSMCC.2012.2189204

16. Lutolf R. Smart Home concept and the integration of energy meters into a home based system. *Proceedings of the 7th International Conference on Metering Apparatus and Tariffs for Electricity Supply*. 1992. P. 277–278.
17. Artificial intelligence in science and society: The vision of USERN / T. Dorigo et al. *IEEE Access*. 2025. Vol. 13. P. 15993–16054, DOI: 10.1109/ACCESS.2025.3529357
18. Ghafur S., Kristensen S., Honeyford K. A retrospective impact analysis of the WannaCry cyberattack on the NHS. *NPJ Digital Medicine*. 2019. Vol. 2, P. 98. DOI: 10.1038/s41746-019-0161-6
19. A deeper look into cybersecurity issues in the wake of Covid-19: A survey / M. Alawida, A. E. Abiodun, O. I. Abiodun, M. Al-Rajab. *Journal of King Saud University - Computer and Information Sciences*. 2022. Vol. 34, iss. 10, part A. P. 8176–8206. DOI: 10.1016/j.jksuci.2022.08.003
20. Hasham S., Joshi S., Mikkelsen D. *Financial crime and fraud in the age of cybersecurity*. McKinsey & Company, 2019.
21. The future of cybercrime: AI and emerging technologies are creating a cybercrime tsunami / P. Treleaven, J. Barnett, D. Brown et al. 2023. DOI: 10.2139/ssrn.4507244
22. Mary A., Edison A. Deep fake detection using deep learning techniques: A literature review. 2023 *International Conference on Control, Communication and Computing (ICCC)*. Thiruvananthapuram, India, 2023. P. 1–6. DOI: 10.1109/ICCC57789.2023.10164881
23. Emsley R. ChatGPT: these are not hallucinations – they’re fabrications and falsifications. *Schizophrenia*. 2023. Iss. 9. P. 52. DOI: 10.1038/s41537-023-00379-4
24. Alsowaigh R. E. Crowd strike causes global Microsoft outage: A case study. *JISCR*. 2025. Vol. 8, iss. 1. P. 63–76. DOI: 10.26735/QHDD4798
25. Martínez J., Durán J. M. Software supply chain attacks, a threat to global cybersecurity: SolarWinds’ case study. *International Journal of Safety and Security Engineering*. 2021. Vol. 11, iss. 5. P. 537–545. DOI: 10.18280/ijss.110505

26. Microsoft Defender Antivirus in Windows overview - Microsoft Defender for Endpoint. Microsoft Learn. URL: <https://learn.microsoft.com/en-us/defender-endpoint/microsoft-defender-antivirus-windows> (дата звернення: 14.03.2025).

27. Johnson M. W. Testing the efficacy of Windows Defender Endpoint Security Control using BAS technology : Masters thesis. Dublin : National College of Ireland, 2024.

28. Drakulić U., Mujčić E. A comparative performance analysis of various antivirus software. Advanced Technologies, Systems, and Applications VIII : Proceedings of the International Symposium on Innovative and Interdisciplinary Applications of Advanced Technologies (IAT 2023). Cham : Springer, 2023. Vol. 644. DOI: 10.1007/978-3-031-43056-5_3

29. The lockdown effect: Implications of the COVID-19 pandemic on Internet traffic / A. Feldmann, O. Gasser, F. Lichtblau et al. Proceedings of the ACM Internet Measurement Conference (IMC '20). Association for Computing Machinery. New York, USA, 2020. P. 1–18. DOI: 10.1145/3419394.3423658

30. Michael K., Abbas R., Roussos G. AI in cybersecurity: The paradox. IEEE Transactions on Technology and Society. 2023. Vol. 4, iss. 2. P. 104–109. DOI: 10.1109/TTS.2023.3280109

31. Observability and incident response in managed serverless environments using ontology-based log monitoring / L. Ben-Shimol, E. Grolman, A. Elyashar et al. IEEE Transactions on Cloud Computing. 2025. Vol. 13, iss. 04. P. 1161–1176. DOI: 10.1109/TCC.2025.3598060

32. Sambamurthy P. Advancing systems observability through artificial intelligence: A comprehensive analysis. International Research Journal of Modernization in Engineering Technology and Science. 2024. Vol. 6, iss. 7. DOI: 10.56726/IRJMETs60598

33. Cantrill B. Hidden in plain sight: Improvements in the observability of software can help you diagnose your most crippling performance problems. ACM Queue. 2006. Vol. 4, no. 1. P. 26–36. DOI: 10.1145/1117389.1117401

34. Observability vs monitoring - difference between data-based processes. Amazon Web Services (AWS). URL: <https://aws.amazon.com/compare/the-difference-between-monitoring-and-observability/> (дата звернення: 19.04.2025).
35. Task Manager. Microsoft Learn. URL: <https://learn.microsoft.com/en-us/shows/inside/task-manager> (дата звернення: 19.04.2025).
36. Azure Monitor overview. Microsoft Learn. URL: <https://learn.microsoft.com/en-us/azure/azure-monitor/fundamentals/overview> (дата звернення: 19.04.2025).
37. Metrics in Azure Monitor. Microsoft Learn. URL: <https://learn.microsoft.com/en-us/azure/azure-monitor/metrics/data-platform-metrics> (дата звернення: 19.04.2025).
38. Logging HOWTO – Python 3.13.4 documentation. Python programming language documentation. URL: <https://docs.python.org/3/howto/logging.html> (дата звернення: 19.04.2025).
39. RFC 5424: The Syslog protocol. RFC. URL: <https://www.rfc-editor.org/rfc/rfc5424.html#page-36> (дата звернення: 20.04.2025).
40. Mendes E., Petrillo F. Log severity levels matter: A multivocal mapping. ArXiv. 2021. DOI: 10.48550/arXiv.2109.01192
41. Gholamian S., Ward P. A. S. A comprehensive survey of logging in software: From logging statements automation to log mining and analysis. Transactions on Software Engineering. Second Revision. ArXiv. 2022. DOI: 10.48550/arXiv.2110.12489
42. logrotate(8) - Linux man page. Linux die.net. URL: <https://linux.die.net/man/8/logrotate> (дата звернення: 20.04.2025).
43. Krasovec B., Príca T. Secure usage of containers in the HPC environment. Nordic e-Infrastructure Tomorrow. Cham : Springer, 2024. Iss. 2398. DOI: 10.1007/978-3-031-86240-3_7
44. Design patterns: elements of reusable object-oriented software / E. Gamma, R. Helm, R. Johnson, J. Vlissides. Addison-Wesley Professional Computing Series. 1995. ISBN: 0-201-63361-2.

45. Design patterns: Singleton. Microsoft Learn. URL: <https://learn.microsoft.com/en-us/shows/visual-studio-toolbox/design-patterns-singleton> (дата звернення: 20.04.2025).
46. The facade pattern. Pro JavaScript Design Patterns. Apress, 2008. DOI: 10.1007/978-1-4302-0496-1_10
47. Design patterns: Adapter and façade. Microsoft Learn. URL: <https://learn.microsoft.com/en-us/shows/visual-studio-toolbox/design-patterns-adapterfaade> (дата звернення: 20.04.2025).
48. Friedl J. Mastering regular expressions: A nutshell handbook. O'Reilly Media, Inc., 2006. ISBN: 0596002890, 9780596002893.
49. Dossot D. RabbitMQ essentials. Packt Publishing Ltd., 2014. ISBN: 1783983205.
50. PostgreSQL: Documentation: 17: LISTEN. PostgreSQL. URL: <https://www.postgresql.org/docs/current/sql-listen.html> (дата звернення: 26.04.2025).
51. Web server vs application server - difference between technology servers. Amazon Web Services (AWS). URL: <https://aws.amazon.com/compare/the-difference-between-web-server-and-application-server/> (дата звернення: 20.04.2025).
52. Jiang X., Mu D., Zhang H. Unix domain sockets applied in Android malware should not be ignored. Information. 2018. Vol. 9 (3). P. 54. DOI: 10.3390/info9030054
53. Cinar O. Pro Android C++ with the NDK. Berkeley, CA : Apress, 2012. DOI: 10.1007/978-1-4302-4828-6
54. Walli S. R. The POSIX family of standards. StandardView. 1995. Vol. 3, iss. 1. P. 11–17. DOI: 10.1145/210308.210315
55. Robbins K. A., Robbins S. UNIX systems programming: Communication, concurrency, and threads. Prentice Hall Professional, 2003. ISBN: 0130424110, 9780130424112.
56. Taylor D. Work the shell - dealing with signals. Linux Journal. 2010. Vol. 2010, iss. 196.

57. Authentication, authorisation, access control. RabbitMQ. URL: <https://www.rabbitmq.com/docs/access-control> (дата звернення 26.04.2025).
58. PostgreSQL: Documentation: 17: 20.3. Authentication methods. PostgreSQL. URL: <https://www.postgresql.org/docs/current/auth-methods.html> (дата звернення: 26.04.2025).
59. RFC 4251: The secure shell (SSH) protocol architecture. RFC. URL: <https://www.rfc-editor.org/rfc/rfc4251> (дата звернення: 26.04.2025).
60. Does return null matter? / S. Kimura, K. Hotta, Y. Higo et al. Software Evolution Week - IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering (CSMR-WCRE), Antwerpen, Belgium. 2014. P. 244–253. DOI: 10.1109/CSMR-WCRE.2014.6747176
61. Sud K., Erdogmus P., Kadry S. Introduction to data science and machine learning. BoD – Books on Demand, 2020. ISBN: 1838803335, 9781838803339.
62. Siriwardena P. HTTP basic/digest authentication. Advanced API Security. Berkeley, CA : Apress, 2014. DOI: 10.1007/978-1-4302-6817-8_3
63. Chandra J. V., Challa N., Pasupuletti S. K. Authentication and authorization mechanism for cloud security. International Journal of Engineering and Advanced Technology (IJEAT). 2019. Vol. 8, iss. 6. DOI: 10.35940/ijeat.F8473.088619
64. DeJonghe D. NGINX cookbook. O'Reilly Media, Inc., 2020. 206 p. ISBN: 149207845X, 9781492078456.
65. SQL Server Connection Pooling - ADO.NET. Microsoft Learn. URL: <https://learn.microsoft.com/en-us/dotnet/framework/data/adonet/sql-server-connection-pooling> (дата звернення: 01.06.2025).
66. Mancas C. On the paramount importance of database constraints. Journal of Information Technology & Software Engineering. 2015. Vol. 05, iss. 03. DOI: 10.4172/2165-7866.1000e125
67. Cleve A., Hainaut J. -L. What do foreign keys actually mean? Proceedings of the 2012 19th Working Conference on Reverse Engineering. Kingston, ON, Canada, 2012. P. 299–307. DOI: 10.1109/WCRE.2012.39

68. La V. H., Fuentes R., Cavalli A. R. Network monitoring using MMT: An application based on the user-agent field in HTTP headers. Proceedings of the IEEE 30th International Conference on Advanced Information Networking and Applications (AINA). Crans-Montana, Switzerland, 2016. P. 147–154. DOI: 10.1109/AINA.2016.41
69. Lavrenovs A., Visky G. Investigating HTTP response headers for the classification of devices on the Internet. Proceedings of the 7th IEEE Workshop on Advances in Information, Electronic and Electrical Engineering (AIEEE). Liepaja, Latvia, 2019. P. 1–6. DOI: 10.1109/AIEEE48629.2019.8977115
70. Rangisetti A. K. A brief introduction to OOP in Python and Solidity. Hands-On Object-Oriented Programming. Berkeley, CA. : Apress, 2024. DOI: 10.1007/979-8-8688-0524-0_10
71. MSDN magazine: Cutting edge - give your classes a software contract. Microsoft Learn. URL: <https://learn.microsoft.com/en-gb/archive/msdn-magazine/2011/april/msdn-magazine-cutting-edge-give-your-classes-a-software-contract> (дата звернення: 01.06.2025).
72. The Python language reference. Python documentation. URL: <https://docs.python.org/3/reference/index.html> (дата звернення: 02.06.2025).
73. Built-in functions. Python documentation. URL: <https://docs.python.org/3/library/functions.html#exec> (дата звернення: 02.06.2025).
74. ECMAScript® 2026 language specification. Draft ECMA-262. 2025. URL: <https://tc39.es/ecma262/multipage/> (дата звернення: 02.06.2025).
75. Kereki F. Mastering JavaScript functional programming: Write clean, robust, and maintainable web and server code using functional JavaScript and TypeScript. Packt Publishing Ltd., 2023. 614 p. ISBN: 1804610410, 9781804610411.
76. Algebraic data type. HaskellWiki. URL: https://wiki.haskell.org/Algebraic_data_type (дата звернення: 03.06.2025).
77. Rehman B. A blend of intersection types and union types : Abstract of thesis for the degree of Doctor of Philosophy at The University of Hong Kong. 2023.

78. Bae S. JavaScript data structures and algorithms: An introduction to understanding and implementing Core Data Structure and Algorithm Fundamentals. Apress Access Books, 2019. ISBN: 1484239881, 9781484239889.
79. Herman D. Effective JavaScript. Ch. 12: Understand variable hoisting. Effective Software Development Series. Pearson Education, 2012. ISBN 978-0-321-81218-6.
80. Formal specification and verification of JDK's identity hash map implementation / M. De Boer, S. De Gouw, J. Klamroth et al. Formal Aspects of Computing. 2023. Vol. 35 (3), article 18. P. 26. DOI: 10.1145/3594729
81. Rees-Hill J. A. Error handling approaches in programming languages. Honors Papers. 2022. Vol. 855.
82. Maran K. A. R., Abdullah N. A. Cuckoo Sandbox vs Virus Total: Categorical analysis between sandboxes. Applied Information Technology and Computer Science. 2023. Vol. 4, no. 2. P. 30–45. URL: <https://publisher.uthm.edu.my/periodicals/index.php/aitcs/article/view/11931>
83. Debas E., Alhumam N., Riad K. Unveiling the dynamic landscape of malware sandboxing: A comprehensive review. 2023. (Preprints). DOI: 10.20944/preprints202312.1009.v1
84. Groner L., Manricks G. JavaScript regular expressions. Series: Community experience distilled. Packt Publishing, 2015. ISBN: 1783282266, 9781783282265.
85. Kumar P. M. A., Pugazhendi E., Kavitha D. Cross-user level de-duplication using distributive soft links. Proceedings of the IEEE International Conference on Power, Control, Signals and Instrumentation Engineering (ICPCSI). 2017. DOI: 10.1109/ICPCSI.2017.8392172
86. Bree D. C. C., Cinnéide M. Ó. The energy cost of the visitor pattern. Proceedings of the IEEE International Conference on Software Maintenance and Evolution (ICSME). 2022. P. 317–328. DOI: 10.1109/ICSME55016.2022.00036
87. estree/es5.md at master. GitHub. URL: <https://github.com/estree/estree/blob/master/es5.md#node-objects> (дата звернення: 10.06.2025).
88. Ajv JSON schema validator. Ajv JSON schema validator. URL: <https://ajv.js.org/> (дата звернення: 10.06.2025).

89. JSON: Data model, query languages and schema specification / P. Bourhis, J. L. Reutter, F. Suárez, V. Domagoj. Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems. 2017. DOI: 10.48550/arXiv.1701.02221
90. Foundations of JSON schema / F. Pezoa, J. L. Reutter, F. Suarez et al. Proceedings of the 25th International Conference on World Wide Web (WWW '16). 2016. P. 263–273. DOI: 10.1145/2872427.2883029
91. Getting started.. Ajv JSON schema validator. URL: <https://ajv.js.org/guide/getting-started.html#basic-data-validation> (дата звернення: 11.06.2025).
92. Systematic analysis of on-premise and cloud services / S. Younus, K. Kumar, I. A. Kandhro et al. International Journal of Cloud Computing. 2024. Iss. 13. P. 214–242. DOI: 10.1504/IJCC.2024.10063641
93. A survey of profit optimization techniques for cloud providers / P. Cong, G. Xu, T. Wei, K. Li. ACM Computing. Surveys (CSUR). 2020. Vol. 53 (2), article no. 26. P. 1–35. DOI: 10.1145/3376917
94. A review on AWS - Cloud computing technology / N. Kewate, A. Raut, Dubekar M. et al. International Journal for Research in Applied Science & Engineering Technology (IJRASET). 2022. Vol. 10, iss. 1. DOI: 10.22214/ijraset.2022.39802
95. Soni M., Altfatter Z., Mukherjee A. Practical AWS networking: Build and manage complex networks using services such as Amazon VPC, Elastic Load Balancing, Direct Connect, and Amazon Route 53. Packt Publishing, 2018. ISBN: 9781788479028.
96. Saini R., Behl R. An introduction to AWS-EC2 (Elastic Compute Cloud). Proceedings of the 2020 International Conference on Research in Management & Technovation. ACSIS. 2020. Vol. 24. P. 99–102. DOI: 10.15439/2020KM4
97. Advanced deployment strategies for elastic load balancing in AWS: A comprehensive study on multi-tier architecture optimization / S. Sharma, R. P. Mahapatra, D. Seth, M. Kapil. Proceedings of the International Conference on Communication, Computing and Energy Efficient Technologies (I3CEET). Gautam Buddha Nagar, India, 2024. P. 285–289. DOI: 10.1109/I3CEET61722.2024.10993779

98. Shields D. AWS security. Simon and Schuster, 2022. P. 312. ISBN: 1638351163, 9781638351160.
99. Barrett D. J., Silverman R. E., Byrnes R. G. SSH, the secure shell: The definitive guide. O'Reilly Media, Inc., 2005. ISBN: 9780596008956, 0596008953.
100. Ryan M., Lucifredi F. AWS system administration: Best practices for sysadmins in the Amazon cloud. O'Reilly Media, 2018. ISBN: 9781449342562, 1449342566.
101. Ramirez H. Automated change management and instance control in AWS: A workflow utilizing AWS systems manager, SNS, and role-based access. International Journal of AI, BigData, Computational and Management Studies. 2024. Vol. 5, no. 1. P. 1–16. DOI: 10.63282/3050-9416.IJAIBDCMS-V5I1P101
102. AWS systems manager session manager. Amazon AWS Documentation. URL: <https://docs.aws.amazon.com/systems-manager/latest/userguide/session-manager.html> (дата звернення: 15.06.2025).
103. sshd_config(5) - Linux man page. Linux Documentation. URL: https://linux.die.net/man/5/sshd_config (дата звернення: 15.06.2025).
104. Talluri S., Makani S. T. Managing Identity and Access Management (IAM) in Amazon Web Services (AWS). Journal of Artificial Intelligence & Cloud Computing. 2023. Vol. 2, no. 1. P. 1–5. DOI: 10.47363/JAICC/2023(2)147
105. Keith J., Sambells J. A brief history of JavaScript. Berkeley, CA : Apress, 2010. DOI: 10.1007/978-1-4302-3390-9_1
106. Teixeira P. Professional node.js: Building Javascript based scalable software. ITPro collection. Programmer to programmer. John Wiley & Sons, 2012. ISBN: 1118240561, 9781118240564.
107. About npm. npm Docs. URL: <https://docs.npmjs.com/about-npm> (дата звернення: 15.06.2025).
108. Archer R. ExpressJS: Web app development with node.js framework. CreateSpace Independent Publishing Platform, 2015. 70 p. ISBN: 1519791607, 9781519791603.
109. winston. NPM. URL: <https://www.npmjs.com/package/winston> (дата звернення: 16.06.2025).

110. Miell I., Sayers A. Docker in practice. Second Ed. Simon and Schuster, 2019. 384 p. ISBN: 1638356300, 9781638356301.

ДОДАТОК А

Список публікацій, в яких опубліковані основні наукові результати дисертації

1. Suprunenko I., Rudnytskyi V. On specifics of adaptive logging method implementation. Bulletin of Cherkasy State Technological University. 2024. Vol. 29, no. 1. P. 36–42. DOI: 10.62660/bcstu/1.2024.36
2. Супруненко І. О., Бабенко В. Г., Рудницький С. В. Метод адаптивного логування розподілених комп'ютерних систем. Технології розвитку безпілотних систем : кол. монографія / під ред. В. М. Рудницького. Черкаси : видавець Вовчок О. Ю., 2025. Т. 2. Безекіпажні системи. С. 135–156. ISBN: 978-617-8725-03-7. DOI: 10.5281/zenodo.19191122
3. Suprunenko I., Rudnytskyi V. Comparison of message passing systems in context of adaptive logging method. Visnyk of Kherson National Technical University. 2024. No. 2 (89). DOI: 10.35546/kntu2078-4481.2024.2.32
4. Suprunenko I., Rudnytskyi V. Dynamic message variant in adaptive logging method. Modern Information Security. 2024. Vol. 59, no. 3. P. 94–99. DOI: 10.31673/2409-7292.2024.030010
5. Suprunenko I., Rudnytskyi V. Validation of dynamic message variant in adaptive logging method. Measuring and Computing Devices in Technological Processes. 2024. No. 4. P. 31–38. DOI: 10.31891/2219-9365-2024-80-5
6. Suprunenko I., Rudnytskyi V. Cloud based architecture for adaptive logging method implementation. Cybersecurity: Education, Science, Technique : Electron. Prof. Sci. J. 2025. Vol. 3, no. 27. P. 329–338. DOI: 10.28925/2663-4023.2025.27.724
7. Супруненко І. О. Адаптивний підхід до логування як новий вимір спостережності за прикладним програмним забезпеченням. Інформаційна безпека та комп'ютерні технології : тези доп. VII Міжнар. наук.-практ. конф. до 30-річчя кафедри кібербезпеки та програмного забезпечення, м. Кропивницький / М-во освіти і науки України, Центральн. нац. техн. ун-т. Кропивницький : ЦНТУ, 2023. С. 45–46. URL: <https://kbpz.kntu.kr.ua/file/content/6621/2023-vii-mizhnarodna-naukovo-praktychnoi-konferentsiia-do-30-ty-richchia-kafedry->

kiberbezpeky-ta-prohramnoho-zabezpechennia-informatsiina-bezpeka-ta-komp-yuterni-tekhnologii-.pdf

8. Suprunenko I. Message passing systems in modern applications and their role in adaptive logging method. Інформаційні технології в освіті, науці й техніці (ІТОНТ-2024) : матеріали VII Міжнар. наук.-практ. конф. Черкаси, 2024. С. 12–14. URL: https://knsa.chdtu.edu.ua/wp-content/uploads/2024/06/Conference-Proceedings-ITEST-2024_25_06.pdf

9. Suprunenko I., Rudnytskyi V. Dynamic source code processing approaches in context of adaptive logging method. Global Society in Formation of New Security System and World Order : матеріали III Міжнар. наук.-практ. інтернет- конф. Дніпро, 2024. С. 20–22. URL: <http://www.wayscience.com/wp-content/uploads/2024/07/Conference-Proceedings-July-4-5-2024.pdf>

ДОДАТОК Б**Документи про впровадження результатів дисертаційної роботи****Довідка**

**про впровадження результатів наукових досліджень
Супруненко Іллі Олександровича**

Наукові результати Супруненко Іллі Олександровича, отримані в процесі дисертаційного дослідження, а саме метод адаптивного логування з динамічним варіантом повідомлень та підсистемою валідації, використаний для контролю коректності розробки програмного забезпечення системи дистанційного управління наземним самохідним роботизованим комплексом.

Програмне забезпечення реалізації методу адаптивного логування встановлено на автоматизованих робочих місцях розробників програмного забезпечення роботизованого комплексу.

Директор ТОВ "МОРОЗ ТЕХ"



Роман МОРОЗ