Форма № 24

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій
(повне найменування факультету)

Кафедра програмних засобів
(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)
бакалавр
(ступінь вищої освіти)

на тему ДОСЛІДЖЕННЯ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ АЛГОРИТМІВ ОБЧИСЛЕНЬ ДЛЯ ВИЯВЛЕННЯ КОМП'ЮТЕРНИХ АТАК СПЕКУЛЯТИВНИМ ВИКОНАННЯМ КОДУ
RESEARCH AND SOFTWARE IMPLEMENTATION OF COMPUTATIONAL ALGORITHMS FOR DETECTING SPECULATIVE EXECUTION ATTACKS

Виконав(ла): студент(ка) 4 курсу, групи КНТ-212
Спеціальності 122 Комп'ютерні науки
(код і найменування спеціальності)

Освітня програма (спеціалізація)
Комп'ютерні науки

РЄЗНІКОВ Д.М.
(ПРІЗВИЩЕ та ініціали)
Керівник ЗАЙКО Т.А.
(ПРІЗВИЩЕ та ініціали)
Рецензент СКРУПСЬКИЙ С.Ю.
(ПРІЗВИЩЕ та ініціали)

2026

Форма № 25

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
(повне найменування закладу вищої освіти)

Факультет КНТ
Кафедра програмних засобів
Ступінь вищої освіти бакалавр
Спеціальність 122 Комп'ютерні науки
(код і найменування)
Освітня програма (спеціалізація) Комп'ютерні науки
(назва освітньої програми (спеціалізації)

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
_____Сергій СУББОТІН
"____" _____2026 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

РЄЗНІКОВА Дмитра Максимовича
(ПРІЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Дослідження та програмна реалізація алгоритмів обчислень для виявлення комп'ютерних атак спекулятивним виконанням коду. Research and Software Implementation of Computational Algorithms for Detecting Speculative Execution Attacks
керівник проєкту (роботи) к.т.н., доцент, ЗАЙКО Тетяна Анатоліївна,
(науковий ступінь, вчене звання, ПРІЗВИЩЕ, ім'я, по батькові)
затверджені наказом закладу вищої освіти від "07" квітня 2026 року № 139
2. Строк подання студентом проєкту (роботи) 10 червня 2026 року
3. Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області. 2. Матеріали та методи. 3. Опис програми. 4. Експлуатація та тестування програми.

5. Перелік графічного матеріалу (з з точним зазначенням обов'язкових креслень, кількість слайдів, плакатів)
Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРІЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	ЗАЙКО Т.А., доцент		
Нормоконтроль	КАЛІНІНА М.В., асистент		

7. Дата видачі завдання « 20 » квітня 2026 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	1 тиждень	Розділ 1
3	Вибір мови програмування та інших технологій	2 тиждень	Розділ 2
	розробки.		
4	Розробка архітектури програми.	2 тиждень	Розділ 3
5	Розробка програми.	3-4 тижні	Розділ 3
6	Тестування та експериментальне дослідження	5 тиждень	Розділ 4
	програмного забезпечення.		
7	Оформлення пояснювальної записки та документів	6 тиждень	Додатки
	до неї.		
8	Нормоконтроль та рецензування.	7 тиждень	
9	Захист роботи.	8 тиждень	

Студент(ка) _____ Дмитро РЄЗНІКОВ
(підпис) (Ім'я ПРІЗВИЩЕ)

Керівник проєкту (роботи) _____ Тетяна ЗАЙКО
(підпис) (Ім'я ПРІЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра: 80 с., 1 табл., 21 рис., 2 дод., 22 джерела.

СПЕКУЛЯТИВНЕ ВИКОНАННЯ КОДУ, ПОБІЧНІ КАНАЛИ, АНАЛІЗ ЧАСОВИХ ХАРАКТЕРИСТИК, SPECTRE, MELTDOWN.

Об’єкт дослідження – процес аналізу часових характеристик доступу до пам’яті та виявлення аномальної активності в обчислювальних системах.

Предмет дослідження – методи та програмні засоби виявлення потенційних атак спекулятивного виконання коду на основі статистичного аналізу часових характеристик доступу до пам’яті.

Мета роботи – підвищення швидкості виявлення потенційно аномальної активності в обчислювальних системах шляхом програмної реалізації алгоритмів аналізу часових характеристик доступу до пам’яті.

Матеріали, методи та технічні засоби: мова програмування Python, операційна система Linux, середовище розробки Visual Studio Code та бібліотеки мови Python psutil, statistics і matplotlib, персональний комп’ютер з процесором Intel Core 2 Duo під управлінням операційної системи Microsoft Windows.

Результати. Створено програмне забезпечення для аналізу часових характеристик доступу до пам’яті та виявлення потенційно аномальної активності.

Висновки. Розроблено програмне забезпечення для виявлення непрямих ознак аномальної поведінки системи на основі аналізу часових характеристик доступу до пам’яті.

Галузь використання – моніторинг комп’ютерних систем та виявлення потенційно аномальної активності, пов’язаної з атаками спекулятивного виконання коду в навчально-дослідницьких проєктах.

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelor: 80 pages, 1 tables, 21 figures, 2 appendixes, 22 sources.

SPECULATIVE CODE EXECUTION, SIDE CHANNELS, TIMING ANALYSIS, SPECTRE, MELTDOWN.

Object of study is a the process of analyzing memory access timing characteristics and detecting anomalous activity in computing systems.

Subject of study are methods and software tools for detecting potential speculative execution attacks based on the statistical analysis of memory access timing characteristics.

The purpose of the work is increasing the speed of detecting potentially anomalous activity in computing systems through the software implementation of algorithms for analyzing memory access timing characteristics.

Materials, methods, and tools: Python programming language, Linux operating system, Visual Studio Code development environment, Python libraries psutil, statistics and matplotlib, and a personal computer with an Intel Core 2 Duo processor running the Microsoft Windows operating system.

Results. Software for analyzing timing characteristics of memory access and detecting potentially anomalous activity has been developed.

Conclusions Software for detecting indirect signs of anomalous system behavior based on the analysis of timing characteristics of memory access has been developed.

The field of use is a monitoring of computer systems and detection of potentially anomalous activity related to speculative code execution attacks in educational and research projects.

ЗМІСТ

С.

Перелік скорочень та умовних познак 8

Вступ 9

1 Аналіз предметної області 11

1.1 Особливості атак спекулятивного виконання коду 11

1.2 Аналіз існуючих методів виявлення спекулятивних атак 14

1.3 Методи аналізу часових характеристик та поведінки пам’яті 17

1.4 Аналіз програмного забезпечення моніторингу та виявлення атак спекулятивного виконання 20

1.5 Висновки за розділом 1 22

2 Матеріали та методи 24

2.1 Вибір мови програмування 24

2.2 Вибір середовища розробки програмного забезпечення 25

2.3 Вибір операційної системи 27

2.4 Вибір засобів моніторингу системи 28

2.5 Вибір методів аналізу часових характеристик 29

2.6 Вибір методів статистичної обробки результатів 30

2.7 Висновки до розділу 2 31

3 Опис програми 32

3.1 Структура програмного забезпечення 32

3.2 Функціонування програмного забезпечення для виявлення

потенційних атак спекулятивного виконання коду 34
3.3 Проєктування інтерфейсу програмного забезпечення 40
4 Експлуатація та тестування програми 47
4.1 Призначення програми 47
4.2 Умови виконання програми 49
4.3 Виконання і тестування програми 51
4.4 Висновки за розділом 4 52
Висновки 54
Перелік джерел посилання 56
Додаток А Текст програми 58
Додаток Б Слайди презентації 74

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

CPU	– Central Processing Unit;
PMC	– Performance Monitoring Counters;
VS Code	– Visual Studio Code.

ВСТУП

Сучасний етап розвитку інформаційних технологій характеризується постійним зростанням обчислювальної потужності комп’ютерних систем, збільшенням кількості оброблюваних даних та високими вимогами до швидкодії програмного забезпечення. Для забезпечення максимальної продуктивності сучасні процесори використовують складні механізми оптимізації виконання команд, серед яких особливе місце займає спекулятивне виконання коду.

Даний механізм дозволяє процесору передбачати можливі гілки виконання програми та виконувати інструкції наперед, ще до остаточного підтвердження правильності переходу.

Попри значне підвищення продуктивності обчислень, використання спекулятивного виконання призвело до появи нового класу критичних вразливостей, пов’язаних із витоком конфіденційної інформації через побічні канали.

Найбільш відомими прикладами таких атак стали Spectre, Meltdown, Foreshadow, ZombieLoad та інші модифікації атак класу transient execution attacks. Їх особливість полягає у тому, що навіть після скасування спекулятивно виконаних інструкцій сліди їх роботи можуть залишатися у кеш-пам’яті процесора, що дозволяє зловмиснику отримувати доступ до захищених даних.

Поява подібних атак продемонструвала, що традиційні механізми програмного та апаратного захисту не забезпечують повної безпеки сучасних комп’ютерних систем.

Особливо небезпечними такі атаки є для хмарних сервісів, серверних платформ, систем віртуалізації та багатокористувацьких середовищ, де витік навіть незначного обсягу інформації може призвести до компрометації критичних даних користувачів або всієї системи в цілому.

У зв’язку з цим особливої актуальності набуває задача виявлення атак спекулятивного виконання коду та створення програмних засобів, здатних аналізувати поведінку системи, виявляти аномальні часові характеристики доступу до пам’яті та фіксувати ознаки використання побічних каналів. Важливим аспектом є не лише дослідження принципів роботи таких атак, але й практична реалізація алгоритмів їх виявлення із використанням сучасних засобів програмування.

Актуальність теми дипломної роботи обумовлена необхідністю підвищення рівня інформаційної безпеки сучасних комп’ютерних систем в умовах постійного розвитку методів апаратних атак та недостатньої ефективності традиційних механізмів захисту.

Дослідження алгоритмів виявлення атак спекулятивного виконання дозволяє сформувати підходи до створення систем моніторингу та аналізу небезпечної активності на рівні процесорної архітектури [1]-[2].

Для досягнення поставленої мети необхідно вирішити такі завдання:

– проаналізувати архітектурні особливості сучасних процесорів та механізми спекулятивного виконання коду;

– дослідити принципи реалізації атак класу Spectre та Meltdown;

– виконати аналіз існуючих методів виявлення атак побічних каналів;

– розробити алгоритми аналізу часових характеристик доступу до пам’яті;

– реалізувати програмний засіб для виявлення ознак атак спекулятивного виконання;

– провести експериментальне дослідження ефективності розробленого програмного забезпечення.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Особливості атак спекулятивного виконання коду

Особливістю сучасних комп’ютерних систем є використання механізмів оптимізації виконання програмного коду. Зростання обсягів обчислень, складності програмного забезпечення та вимог до швидкодії змусило виробників процесорів застосовувати технології, спрямовані на підвищення продуктивності без суттєвого збільшення тактової частоти процесора [1]-[3].

Для цього сучасні процесори використовують конвеєризацію інструкцій, багаторівневу кеш-пам’ять, механізми прогнозування переходів та спекулятивне виконання команд. У звичайному режимі роботи подібні механізми дозволяють значно прискорити виконання програм та ефективніше використовувати ресурси обчислювальної системи.

Спекулятивне виконання являє собою механізм, при якому процесор виконує інструкції наперед, ще до остаточного визначення правильності обраної гілки виконання програми. Якщо прогноз процесора буде правильним, то система отримає приріст продуктивності, оскільки частина команд вже виконана. У випадку помилки, результати таких обчислень скасовуються, а процесор повертається до коректної гілки виконання [3]-[4].

Довго вважалося, що скасування результатів спекулятивного виконання повністю усуває можливі наслідки помилкових операцій. Але подальші дослідження показали, що навіть після скасування виконаних інструкцій, певні зміни внутрішнього стану процесора можуть залишатися. Зокрема, це стосується кеш-пам’яті, буферів процесора та часових характеристик доступу до даних [4].

Саме на використанні таких побічних ефектів і базуються атаки спекулятивного виконання коду. Їх основна ідея полягає у тому, щоб примусити процесор тимчасово виконати операції доступу до даних, які зазвичай недоступні для користувача або процесу. Після цього зловмисник

аналізує непрямі ознаки роботи системи та отримує можливість відновити значення певних конфіденційних даних.

Особливістю таких атак є те, що вони безпосередньо не порушують класичні механізми контролю доступу. Операційна система може працювати коректно, а механізми розмежування прав – не фіксувати порушень. А витік інформації відбувається через аналіз внутрішніх процесів роботи процесора, які не були розраховані на використання як каналу передачі даних.

Найбільш відомими атаками даного класу стали Spectre та Meltdown, інформація про які була оприлюднена у 2018 році. Їх поява викликала значний резонанс у сфері кібербезпеки, оскільки вразливими виявилися мільйони комп’ютерних систем по всьому світу.

Атака Spectre базується на маніпулюванні механізмом прогнозування переходів. Зловмисник створює умови, при яких процесор починає спекулятивно виконувати небезпечну ділянку коду. Хоча після виявлення помилки результати виконання скасовуються, зміни у кеш-пам’яті залишаються доступними для подальшого аналізу [5].

Meltdown використовує іншу особливість роботи процесора. У цьому випадку процесор тимчасово виконує операції читання привілейованої пам’яті ще до завершення перевірки прав доступу. Навіть короткочасне виконання такої операції дозволяє залишити сліди у кеш-пам’яті, які можуть бути використані для відновлення інформації [6].

Після виявлення Spectre та Meltdown було знайдено ще значну кількість подібних атак: Foreshadow, ZombieLoad, RIDL, Fallout та інші. Більшість із них використовують схожий принцип – аналіз побічних ефектів роботи процесора після тимчасового виконання інструкцій [7].

Важливою особливістю атак спекулятивного виконання є складність їх виявлення. На відміну від традиційних мережевих атак або шкідливого програмного забезпечення, вони можуть не залишати очевидних слідів у системі: відсутні зміни файлів, підозріла мережева активність або критичні помилки операційної системи.

Ще однією проблемою є те, що атаки можуть маскуватися під звичайні обчислювальні процеси: багаторазові звернення до пам’яті, вимірювання часу виконання операцій та інші. Це може виглядати як нормальна робота прикладного програмного забезпечення і це суттєво ускладнює задачу автоматичного виявлення подібної активності.

Для реалізації атак спекулятивного виконання часто використовуються методи аналізу часових характеристик доступу до кеш-пам’яті. Найбільш відомими є Flush+Reload, Prime+Probe та Flush+Flush. Усі вони базуються на вимірюванні часу доступу до певних областей пам’яті та подальшому аналізі отриманих результатів [7]-[8].

Наприклад, якщо дані вже знаходяться у кеш-пам’яті, доступ до них виконується значно швидше, ніж у випадку звернення до оперативної пам’яті. Проаналізувавши різницю у часі доступу, зловмисник може визначати, які саме дані використовувалися процесором під час спекулятивного виконання.

Проблема атак спекулятивного виконання є особливо актуальною для хмарних обчислень та систем віртуалізації. У подібних середовищах декілька користувачів можуть використовувати один фізичний процесор, що створює потенційну можливість витоку даних між ізольованими віртуальними середовищами.

Повністю усунути ризик подібних атак досить складно, оскільки вони безпосередньо пов’язані з архітектурними особливостями сучасних процесорів. Саме тому важливого значення набувають методи програмного виявлення атак, аналізу поведінки системи та моніторингу аномальної активності.

Таким чином, атаки спекулятивного виконання коду являють собою один із найбільш небезпечних сучасних класів апаратно-програмних вразливостей, що використовують побічні ефекти роботи процесора для отримання конфіденційної інформації. Їх складність, прихований характер та можливість обходу традиційних механізмів захисту обумовлюють

необхідність розробки нових алгоритмів та програмних засобів виявлення подібних атак.

1.2 Аналіз існуючих методів виявлення спекулятивних атак

Атаки типу спекулятивного виконання коду представляють серйозну проблему для сучасних комп'ютерних систем. Після оприлюднення атак Spectre та Meltdown, значна увага дослідників була зосереджена не лише на пошуку способів усунення вразливостей, але і на розробці методів їх виявлення. Складність виявлення атак такого типу полягає у тому, що подібні атаки працюють на рівні внутрішніх механізмів процесора та практично не залишають класичних ознак компрометації системи [9]-[10].

На відміну від традиційного шкідливого програмного забезпечення, атаки спекулятивного виконання не обов'язково створюють нові файли, змінюють системні параметри або генерують помітний мережевий трафік. У багатьох випадках процес, який виконує атаку, може виглядати як звичайна програма [11]. Саме тому традиційні антивірусні засоби або системи виявлення вторгнень не завжди здатні виявляти подібну активність системи.

Одним із перших напрямків протидії таким атакам стали апаратні та програмні оновлення процесорів і операційних систем. Виробники почали впроваджувати механізми ізоляції пам'яті, обмеження спекулятивного виконання та додаткові перевірки під час прогнозування переходів. Але такі рішення не можуть повністю усунути проблему та часто впливають на продуктивність системи [9]-[11].

Тому особливого значення набули методи програмного виявлення атак. Основна ідея яких полягає в аналізі поведінки системи та пошуку непрямих ознак аномальної активності.

Одним із найбільш поширених підходів є аналіз часових характеристик доступу до пам'яті. Більшість атак спекулятивного виконання використовують різницю між швидкістю доступу до кешованих і

некешованих даних. Якщо певні області пам’яті багаторазово перевіряються з дуже високою точністю вимірювання часу, це може свідчити про спробу реалізації cache side-channel attack.

Такий підхід не дозволяє точно визначити сам факт атаки, але дає можливість виявляти підозрілу поведінку процесів. Наприклад, аномально велика кількість звернень до кеш-пам’яті або постійне використання високоточних таймерів можуть розглядатися як потенційно небезпечна активність.

Ще одним методом протидії є використання апаратних лічильників продуктивності процесора – Performance Monitoring Counters. Сучасні процесори здатні збирати статистику щодо кількості промахів кешу, передбачень переходів, спекулятивно виконаних інструкцій та інших внутрішніх подій системи.

Аналіз таких показників дозволяє виявляти нетипову поведінку програм. Наприклад, під час виконання атак спекулятивного виконання часто спостерігається збільшення кількості cache miss, а також значна кількість коротких циклічних операцій доступу до пам’яті.

Сильною стороною такого підходу є можливість отримання інформації саме від процесора, що дозволяє виконувати моніторинг системи у режимі реального часу. Але існує проблема великої кількості службових подій, серед яких складно виділити саме ознаки атаки. Крім того, деякі легітимні програми можуть створювати схожу активність, що призводить до хибних спрацювань.

Окрему групу складають методи поведінкового аналізу системи, де оцінюється не окрема подія, а загальна модель роботи процесу або системи. Аналізуються частота звернень до пам’яті, структура виконання циклів, використання інструкцій синхронізації, характер взаємодії з кеш-пам’яттю та інші параметри [12].

Подібні методи часто використовують елементи статистичного аналізу або машинного навчання. Система накопичує інформацію про

нормальну поведінку програм, після чого порівнює її з поточними характеристиками роботи процесу. Якщо виявляються значні відхилення, формується повідомлення про можливу атаку.

Перевагою поведінкового аналізу є здатність виявляти нові або модифіковані варіанти атак, для яких ще не існує готових сигнатур. Однак реалізація таких систем є досить складною, оскільки необхідно правильно визначити межу між нормальною та аномальною поведінкою.

Також для виявлення такого роду атак використовуються методи статичного аналізу програмного коду. В цих методах виконується перевірка наявності потенційно небезпечних конструкцій, що зв’язані із вимірюванням часу доступу до пам’яті, частими зверненнями до кешу або використанням інструкцій, характерних для cache attacks [13].

Недоліком статичного аналізу є те, що він не враховує реальну поведінку програми під час виконання. Крім того, сучасні атаки можуть навмисно маскуватися або змінювати структуру коду для уникнення подібних перевірок.

У деяких дослідженнях пропонуються комбіновані методи виявлення, які поєднують декілька підходів одночасно. Наприклад, аналіз апаратних лічильників може використовуватися разом із поведінковим моніторингом або статистичною обробкою часових характеристик. Такий підхід дозволяє підвищити точність виявлення та зменшити кількість хибних спрацювань.

Але попри значну кількість досліджень у цій сфері, універсального методу виявлення атак спекулятивного виконання на сьогодні не існує. Основна причина полягає у великій різноманітності способів реалізації атак та залежності їх поведінки від особливостей конкретної архітектури процесора [9]-[12].

Крім того, частина існуючих методів потребує значних обчислювальних ресурсів або може негативно впливати на продуктивність системи.

Саме тому одним із актуальних напрямків досліджень залишається розробка ефективних алгоритмів програмного моніторингу, які забезпечують достатню точність виявлення при мінімальному навантаженні на систему.

Таким чином, аналіз існуючих методів показує, що найбільш перспективними напрямками є дослідження часових характеристик доступу до пам’яті, аналіз поведінки кеш-пам’яті та використання апаратних показників роботи процесора. Саме ці підходи можуть бути використані як основа для створення програмної системи виявлення атак спекулятивного виконання коду.

1.3 Методи аналізу часових характеристик та поведінки пам’яті

Головною особливістю атак спекулятивного виконання є те, що вони використовують не прямий доступ до інформації, а побічні ефекти роботи комп’ютерної системи. Джерелом витоку даних стає кеш-пам’ять процесора та різниця у швидкості доступу до різних областей пам’яті. Саме тому значна частина сучасних методів виявлення подібних атак базується на аналізі часових характеристик виконання операцій.

У звичайному режимі роботи доступ до даних у кеш-пам’яті відбувається значно швидше, ніж звернення до оперативної пам’яті. Якщо необхідні дані вже знаходяться у кеші, то процесор отримує їх практично миттєво. У випадку cache miss виникає необхідність звернення до оперативної пам’яті, що потребує більшого часу. Для користувача ця різниця майже непомітна, але спеціальні програмні засоби здатні фіксувати подібні зміни з дуже високою точністю [11].

Саме на цьому принципі базується значна кількість cache side-channel attacks. Зловмисник виконує серію вимірювань часу доступу до певних ділянок пам’яті та аналізує отримані результати. Якщо доступ виконується швидко, це може свідчити про те, що відповідні дані вже були використані процесором та знаходяться у кеш-пам’яті.

Для проведення подібного аналізу використовуються високоточні таймери та спеціальні інструкції процесора. Одним із найбільш відомих механізмів є використання інструкції RDTSC, яка дозволяє отримати значення внутрішнього лічильника тактів процесора. Порівнюючи часові мітки до та після виконання операції доступу до пам’яті, можна визначити тривалість виконання конкретної команди.

У більшості випадків одиничне вимірювання не дає достатньо точної інформації через вплив фонових процесів, планувальника операційної системи та інших факторів. Саме тому зазвичай використовується багаторазове повторення операцій із подальшою статистичною обробкою результатів [12].

Одним із найпростіших підходів є обчислення середнього часу доступу до пам’яті. Якщо отримане значення суттєво відрізняється від типових показників роботи системи, це може свідчити про аномальну активність. Але середнього значення часто недостатньо, тому що окремі вимірювання можуть мати значні відхилення.

Тому додатково використовуються медіанні значення, дисперсія та аналіз розподілу результатів. Наприклад, під час реалізації атак типу Flush+Reload часові характеристики доступу до кешованих даних зазвичай формують окрему групу значень, яка добре відокремлюється від доступу до оперативної пам’яті.

Ще одним важливим напрямком є аналіз поведінки кеш-пам’яті процесора. Сучасні процесори мають багаторівневу структуру кешу, а механізми керування пам’яттю працюють автоматично та приховано від користувача. Активне використання кешу залишає характерні ознаки, які можуть бути зафіксовані програмними методами моніторингу.

Для цього використовуються апаратні лічильники продуктивності процесора. Вони дозволяють отримувати інформацію про кількість cache miss, cache hit, виконаних інструкцій, помилкових передбачень переходів та інших внутрішніх подій системи.

Під час нормальної роботи програм подібні показники зазвичай змінюються відносно плавно. А атаки спекулятивного виконання часто супроводжуються великою кількістю циклічних звернень до пам’яті та нетиповою активністю кешу, що може використовуватися як ознака потенційної атаки.

Окрему увагу приділяють аналізу поведінки процесів. У багатьох випадках шкідливі програми виконують значну кількість повторюваних операцій із дуже короткими інтервалами часу. Наприклад, для отримання стабільного результату атака може багаторазово повторювати однакову послідовність вимірювань [14].

Така поведінка не завжди характерна для звичайних програм користувача, тому її можна використовувати як додатковий критерій для виявлення підозрілої активності. Для цього аналізуються частота звернень до пам’яті, структура циклів, кількість системних викликів та інтенсивність використання процесора

У сучасних системах також використовуються методи статистичного аналізу та класифікації даних. Після накопичення результатів вимірювань система може автоматично визначати аномальні значення та формувати повідомлення про можливу атаку.

У деяких випадках застосовуються порогові алгоритми. Наприклад, якщо кількість cache miss або частота вимірювання часу перевищує встановлене значення, процес визначається як потенційно небезпечний. Перевагою такого підходу є простота реалізації, однак вибір порогових значень потребує експериментального дослідження та залежить від особливостей конкретної системи.

Більш складні системи використовують елементи машинного навчання. У цьому випадку формується набір характеристик роботи процесу, після чого алгоритм виконує класифікацію поведінки як нормальної або аномальної. Подібний підхід дозволяє виявляти навіть нові модифікації атак,

однак потребує значної кількості навчальних даних та додаткових обчислювальних ресурсів [12]-[14].

На практиці найбільш ефективними виявляються комбіновані методи аналізу. Поєднання часових характеристик доступу до пам’яті, статистичного аналізу та моніторингу апаратних подій дозволяє підвищити точність виявлення та зменшити кількість хибних спрацювань.

Таким чином, аналіз часових характеристик та поведінки пам’яті є одним із найбільш перспективних напрямків виявлення атак спекулятивного виконання коду. Використання програмних методів моніторингу дозволяє виявляти непрямі ознаки аномальної активності та створює основу для розробки ефективних систем захисту сучасних комп’ютерних систем [14].

1.4 Аналіз програмного забезпечення моніторингу та виявлення атак спекулятивного виконання

Для виявлення атак спекулятивного виконання коду використовуються різні програмні засоби моніторингу системи. Частина з них призначена для аналізу апаратних подій процесора, інші – для моніторингу поведінки процесів, аналізу кеш-пам’яті або дослідження часових характеристик виконання операцій.

Більшість засобів не є вузькоспеціалізованими рішеннями саме для виявлення атак Spectre або Meltdown. Вони використовуються як універсальні інструменти моніторингу продуктивності та діагностики системи, однак можуть застосовуватися для виявлення непрямих ознак атак через побічні канали.

Проведений аналіз показує, що найбільш перспективними для створення системи виявлення атак є засоби, які дозволяють працювати з апаратними лічильниками продуктивності процесора та аналізувати часові характеристики доступу до пам’яті (таблиця 1.1).

Таблиця 1.1 – Програмне забезпечення виявлення спекулятивних атак

Програмний засіб	Основне призначення	Можливості для виявлення атак	Переваги	Недоліки
1	2	3	4	5
Valgrind	Динамічний аналіз програм та роботи пам’яті	Аналіз звернень до пам’яті та поведінки процесів	Детальний контроль виконання програми	Значне зниження швидкодії системи
Cachegrind	Аналіз використання кеш-пам’яті	Дослідження cache hit/cache miss	Детальна статистика кеш-пам’яті	Працює переважно у тестовому режимі
Intel VTune Profiler	Профілювання та аналіз продуктивності	Аналіз апаратних подій, дослідження навантаження процесора	Висока деталізація результатів	Закритий комерційний продукт
Wireshark	Аналіз мережевого трафіку	Непряме виявлення аномальної активності процесів	Зручний інтерфейс, поширеність	Не аналізує апаратні події процесора

Продовження таблиці 1.1

1	2	3	4	5
Python + psutil	Програмний моніторинг процесів та ресурсів	Аналіз поведінки процесів, навантаження CPU та пам’яті	Простота програмної реалізації	Обмежений доступ до низькорівневих подій
Intel PIN	Інструмент динамічного аналізу виконання програм	Відстеження поведінки інструкцій та доступу до пам’яті	Глибокий аналіз роботи програм	Складність реалізації та налаштування

Аналіз існуючого програмного забезпечення показує, що для ефективного виявлення атак спекулятивного виконання доцільно використовувати комбінований підхід, який поєднує моніторинг апаратних подій процесора, аналіз поведінки пам’яті та програмну обробку часових характеристик системи.

1.5 Висновки за розділом 1

В першому розділі дипломної роботи було проведено аналіз предметної області, пов’язаної з атаками спекулятивного виконання коду та методами їх виявлення у сучасних комп’ютерних системах.

Розглянуто особливості атак Spectre, Meltdown та інших модифікацій атак через побічні канали. Визначено, що їх основною особливістю є використання непрямих ознак роботи процесора, зокрема часових характеристик доступу до пам’яті та змін стану кеш-пам’яті. На відміну від традиційних атак, подібні механізми складно виявляються стандартними

засобами захисту, оскільки не залишають явних слідів компрометації системи.

Проведений аналіз існуючих методів виявлення атак показав, що найбільш ефективними є підходи, пов’язані з аналізом поведінки процесів, дослідженням часових характеристик виконання операцій та моніторингом апаратних подій процесора. Особливу увагу приділено використанню апаратних лічильників продуктивності та методам статистичної обробки результатів вимірювань.

Також було виконано аналіз програмного забезпечення, яке може використовуватися для моніторингу системи та виявлення ознак атак спекулятивного виконання. Встановлено, що найбільш перспективними є засоби, які забезпечують доступ до апаратних подій процесора та дозволяють реалізовувати власні алгоритми аналізу поведінки пам’яті й процесів.

За результатами проведеного аналізу можна зробити висновок, що задача виявлення атак спекулятивного виконання залишається актуальною та потребує подальших досліджень в розробці програмних алгоритмів моніторингу часових характеристик доступу до пам’яті та аналізу аномальної поведінки системи.

Отримані результати є основою для подальшої розробки алгоритму виявлення атак спекулятивного виконання коду та створення програмного засобу моніторингу, що буде розглянуто у наступних розділах дипломної кваліфікаційї роботи.

2 МАТЕРІАЛИ ТА МЕТОДИ

Основною проблемою сучасних комп’ютерних систем є складність виявлення атак, які використовують побічні ефекти роботи процесора. На відміну від традиційного шкідливого програмного забезпечення, атаки спекулятивного виконання не завжди залишають очевидні ознаки своєї присутності в системі. У більшості випадків вони використовують особливості роботи кеш-пам’яті, механізмів прогнозування переходів та часових характеристик виконання операцій.

У зв’язку з цим виникає необхідність створення програмних засобів, здатних виконувати моніторинг поведінки системи та виявляти ознаки аномальної активності. Особливу увагу доцільно приділити аналізу часових характеристик доступу до пам’яті та дослідженню параметрів роботи процесора під час виконання підозрілих операцій.

Основною задачею даної роботи є розробка алгоритму програмного виявлення атак спекулятивного виконання коду на основі аналізу поведінки пам’яті та часових характеристик виконання операцій доступу до даних.

2.1 Вибір мови програмування

Одним з важливих етапів розробки програмної системи виявлення атак спекулятивного виконання є вибір мови програмування. Від цього залежить швидкість розробки програмного забезпечення, можливість роботи із системними ресурсами, доступ до засобів аналізу даних та зручність реалізації алгоритмів моніторингу [15].

Для виконання завдань дипломної кваліфікаційної роботи було обрано мову програмування Python. Основною причиною такого вибору є поєднання простоти розробки, широких можливостей для обробки даних та наявності великої кількості готових бібліотек для системного програмування.

На відміну від низькорівневих мов програмування, Python дозволяє швидше реалізовувати алгоритми аналізу та тестувати різні підходи до виявлення аномальної поведінки системи. Це особливо важливо під час експериментального дослідження, коли необхідно багаторазово змінювати параметри алгоритмів та виконувати обробку великої кількості результатів вимірювань.

Ще однією перевагою Python є підтримка роботи із системними утилітами операційної системи Linux. Це дозволяє отримувати інформацію про процеси, використання пам’яті, навантаження процесора та інші параметри системи без необхідності розробки складних низькорівневих модулів [16].

Для реалізації програмного засобу також важливе значення має наявність бібліотек статистичної обробки даних. Аналіз часових характеристик доступу до пам’яті передбачає виконання великої кількості вимірювань та подальшу математичну обробку отриманих результатів. Python забезпечує зручні засоби для виконання таких задач.

Крім того, мова Python активно використовується у сфері кібербезпеки, автоматизації аналізу даних та створення систем моніторингу. Це спрощує інтеграцію програмного забезпечення з існуючими інструментами аналізу та дозволяє використовувати готові бібліотеки для роботи із системними подіями.

2.2 Вибір середовища розробки програмного забезпечення

Для розробки програмного засобу було обрано середовище Visual Studio Code. Дане середовище є одним із найбільш поширених інструментів для розробки програмного забезпечення мовою Python та підтримує роботу в операційних системах Linux, Windows і macOS.

Однією з основних причин вибору Visual Studio Code є його універсальність та підтримка великої кількості розширень для

програмування, налагодження та аналізу коду. Середовище дозволяє зручно працювати з Python-проєктами, виконувати запуск програм, переглядати результати роботи та швидко вносити зміни в код [17]-[20].

Під час реалізації програмного забезпечення використовувались вбудовані засоби підсвічування синтаксису, автоматичного форматування коду та налагодження програм. Це значно спрощує процес розробки та дозволяє швидше виявляти помилки під час тестування окремих модулів системи.

Ще однією перевагою Visual Studio Code є підтримка інтегрованого термінала. Завдяки цьому запуск програм, робота із системними утилітами Linux та використання інструментів моніторингу можуть виконуватись безпосередньо у середовищі розробки.

Для роботи з бібліотеками Python та встановлення додаткових пакетів використовувався менеджер пакетів pip. Це дозволило швидко підключити необхідні модулі для статистичної обробки даних, моніторингу системи та побудови графіків [21].

Також середовище підтримує роботу з віртуальними середовищами Python, що дозволяє ізолювати необхідні бібліотеки проєкту та уникати конфліктів між різними версіями пакетів [20]-[22].

Під час розробки програми використовувались такі основні бібліотеки:

– psutil – для моніторингу процесів та системних ресурсів;

– statistics – для статистичної обробки результатів;

– matplotlib – для побудови графіків;

– time – для вимірювання часових характеристик;

– subprocess – для взаємодії із системними утилітами.

Таким чином, використання Visual Studio Code забезпечує зручність розробки, тестування та налагодження програмного забезпечення, а також дозволило організувати роботу з системними інструментами моніторингу у єдиному середовищі.

2.3 Вибір операційної системи

Для реалізації програмного забезпечення було обрано операційну систему Linux.

Однією з основних причин такого вибору є відкритість системи та можливість отримання доступу до механізмів моніторингу апаратних подій процесора. У Linux доступні інструменти аналізу продуктивності, які дозволяють працювати з апаратними лічильниками процесора та отримувати інформацію про поведінку кеш-пам’яті [16].

У середовищі Linux також реалізована підтримка системного інтерфейсу perf_event_open, що використовується для взаємодії з Performance Monitoring Counters. Ці механізми є важливими під час аналізу атак спекулятивного виконання.

Ще однією перевагою Linux є можливість гнучкого керування процесами та системними ресурсами. Операційна система дозволяє виконувати моніторинг процесів у режимі реального часу, отримувати статистику використання пам’яті та аналізувати навантаження на процессор [17].

Важливим фактором є і те, що значна частина сучасних досліджень у сфері side-channel attacks виконується саме у Linux-середовищі. Це дозволяє використовувати існуючі приклади реалізації атак, інструменти тестування та методики аналізу [18].

Крім того, Linux не потребує використання платного програмного забезпечення та забезпечує стабільну роботу навіть під час виконання великої кількості експериментальних вимірювань.

Таким чином, використання Linux є доцільним для реалізації системи моніторингу та дослідження атак спекулятивного виконання коду.

2.4 Вибір засобів моніторингу системи

Для аналізу поведінки процесора та дослідження часових характеристик доступу до пам’яті дуже важливо використовувати спеціалізовані засоби моніторингу системи [16].

В даній роботі основним інструментом моніторингу було обрано Linux perf. Він дозволяє отримувати інформацію про апаратні події процесора, зокрема кількість cache miss, cache hit, branch miss та інші характеристики роботи системи.

Перевагою Linux perf є можливість роботи без внесення змін в ядро операційної системи або архітектуру процесора. Цей інструмент входить до складу більшості Linux-дистрибутивів та підтримує роботу з сучасними процесорами Intel і AMD [15].

Для аналізу процесів та використання системних ресурсів також використовується бібліотека psutil. Вона дозволяє отримувати інформацію про навантаження процесора, використання оперативної пам’яті, активні процеси та кількість потоків виконання.

Використання psutil спрощує реалізацію програмного моніторингу та дозволяє автоматизувати збір статистичних даних під час роботи системи.

Для візуалізації результатів дослідження використовується бібліотека matplotlib. Побудова графіків дозволяє наочно аналізувати зміну часових характеристик та поведінку системи під час виконання експериментів.

Отже, обрані засоби моніторингу забезпечують можливість збору необхідних даних для реалізації алгоритму виявлення атак спекулятивного виконання [16].

2.5 Вибір методів аналізу часових характеристик

Більшість спекулятивних атак через побічні канали пов’язані зі зміною часу доступу до даних, тому саме аналіз часових характеристик дозволяє виявляти нетипову поведінку системи [12].

В даній роботі використовується підхід, заснований на багаторазовому вимірюванні часу виконання операцій доступу до пам’яті з подальшим аналізом отриманих результатів [14].

Окреме вимірювання не завжди є достатньо інформативним, тому що на роботу системи впливають різні фонові процеси, навантаження процесора, особливості планування потоків та інші фактори. Тому аналіз виконується не для одного значення, а для серії послідовних вимірювань.

Після збору даних проводиться їх статистична обробка. Для цього визначаються: середні значення часу доступу, медіанні показники, мінімальні та максимальні значення, а також величина стандартного відхилення. Окремо аналізується характер розподілу отриманих результатів [19].

Такий підхід дозволяє відокремлювати випадкові коливання від стабільних аномальних змін у роботі системи. Наприклад, якщо певний процес регулярно демонструє нетипові часові характеристики доступу до пам’яті, це може свідчити про спробу реалізації атаки через побічний канал.

Ще однією перевагою багаторазових вимірювань є зменшення кількості хибних спрацювань. У реальних умовах робота комп’ютерної системи ніколи не є повністю стабільною, тому поодинокі відхилення можуть виникати навіть під час нормального функціонування програмного забезпечення. Використання серії вимірювань дозволяє отримати більш надійні результати та підвищити точність аналізу.

Крім того, статистична обробка даних дає можливість порівнювати поведінку системи у звичайному режимі роботи та під час виконання тестових сценаріїв, пов’язаних із навантаженням кеш-пам’яті або інтенсивними зверненнями до пам’яті [16].

Таким чином, аналіз часових характеристик доступу до пам’яті є одним із найбільш доцільних методів виявлення аномальної активності, характерної для атак спекулятивного виконання коду.

2.6 Вибір методів статистичної обробки результатів

Під час виконання вимірювань часу доступу до пам’яті отримані результати можуть помітно відрізнятися навіть в межах одного експерименту. Це пов’язано з тим, що операційна система постійно виконує фонові задачі, розподіляє ресурси між процесами та потоками, а навантаження на процесор у різні моменти часу може змінюватися [16]. Тому окремі значення іноді виходять нестабільними або випадковими.

Саме тому для аналізу результатів недостатньо одного вимірювання. У роботі використовується підхід, заснований на серії послідовних вимірювань із подальшою статистичною обробкою отриманих даних [17].

Основним показником аналізу є середній час доступу до пам’яті. Він дозволяє отримати загальне уявлення про поведінку системи та зменшити вплив випадкових короткочасних відхилень. Проте лише середнього значення недостатньо, оскільки окремі аномальні результати можуть суттєво впливати на кінцевий показник.

Тому додатково використовується медіанне значення, яке є більш стійким до поодиноких різких змін. Окрім цього, визначаються мінімальні та максимальні значення часу доступу, а також стандартне відхилення [19].

Важливим є аналіз стандартного відхилення. Якщо система працює у звичайному режимі, часові характеристики зазвичай змінюються в невеликих межах. А ось під час аномальної активності або спроб реалізації атаки розкид результатів може ставати значно більшим.

Після статистичної обробки результати порівнюються з типовими показниками роботи системи. Якщо отримані значення стабільно виходять за встановлені межі то процес розглядається як потенційно підозрілий.

2.7 Висновки до розділу 2

В другому розділі роботи було обґрунтовано вибір програмних та технічних засобів, що використовуються для реалізації системи виявлення атак спекулятивного виконання коду.

Для розробки програмного забезпечення обрано мову програмування Python, яка забезпечує зручність реалізації алгоритмів моніторингу та статистичної обробки даних.

У якості операційної системи обрано Linux, що дозволяє працювати з апаратними лічильниками процесора та системними механізмами моніторингу.

Також визначено основні методи аналізу часових характеристик доступу до пам’яті та підходи до статистичної обробки результатів вимірювань.

Отримані результати створюють основу для подальшої програмної реалізації алгоритму виявлення атак спекулятивного виконання коду та проведення експериментального дослідження ефективності роботи системи.

3 ОПИС ПРОГРАМИ

3.1 Структура програмного забезпечення

Структура програмного забезпечення побудована за модульним принципом.

Кожен модуль відповідає за виконання окремої задачі, що спрощує тестування програми та дозволяє незалежно змінювати окремі частини системи без необхідності повного переписування коду.

Загалом програмне забезпечення складається з таких основних модулів:

– модуль збору часових характеристик;

– модуль моніторингу системи;

– модуль статистичної обробки результатів;

– модуль аналізу аномальної активності;

– модуль формування звітів та візуалізації результатів.

Основним елементом системи є модуль збору часових характеристик. Його задача полягає у виконанні багаторазових вимірювань часу доступу до пам’яті та збереженні отриманих результатів для подальшого аналізу.

Під час роботи модуль виконує серію послідовних операцій доступу до визначених ділянок пам’яті та фіксує тривалість виконання кожної операції. Для цього використовуються системні засоби вимірювання часу з високою точністю.

Отримані результати передаються до модуля статистичної обробки. На цьому етапі виконуються обчислення середніх значень, медіани, стандартного відхилення та інших параметрів, необхідних для оцінювання стабільності роботи системи.

Окремо реалізовано модуль моніторингу системи. Його основна задача – отримання інформації про поточний стан процесів, навантаження на процесор та використання оперативної пам’яті.

Під час виконання програми модуль моніторингу періодично збирає інформацію про активні процеси та системні ресурси. Це дозволяє враховувати вплив стороннього навантаження на результати вимірювань та зменшувати кількість хибних спрацювань.

Для аналізу аномальної активності використовується окремий модуль, який порівнює отримані результати з типовими характеристиками роботи системи. Якщо часові параметри виходять за встановлені межі або система демонструє нестандартну поведінку, формується повідомлення про потенційно небезпечну активність.

У роботі використовується простий пороговий підхід до виявлення аномалій. Це дозволяє зменшити складність програмної реалізації та забезпечує достатню швидкість роботи системи під час виконання моніторингу у реальному часі.

Для зручності аналізу результатів у програмі передбачено модуль візуалізації даних. Він використовується для побудови графіків зміни часових характеристик та відображення статистичних параметрів роботи системи.

Взаємодія між модулями виконується послідовно. Спочатку система збирає часові характеристики та інформацію про роботу процесів, після чого виконується статистична обробка даних і аналіз отриманих результатів. На завершальному етапі формується звіт про роботу системи та виявлені аномалії.

Перевагою такої структури є можливість подальшого вдосконалення програми. За необхідності до системи можуть бути додані нові модулі аналізу або змінені методи виявлення аномальної активності без суттєвої зміни загальної архітектури програмного забезпечення.

Таким чином, розроблена структура програми забезпечує можливість збору та аналізу часових характеристик доступу до пам’яті, моніторингу системних параметрів та виявлення потенційних ознак атак спекулятивного виконання коду (рис. 3.1).

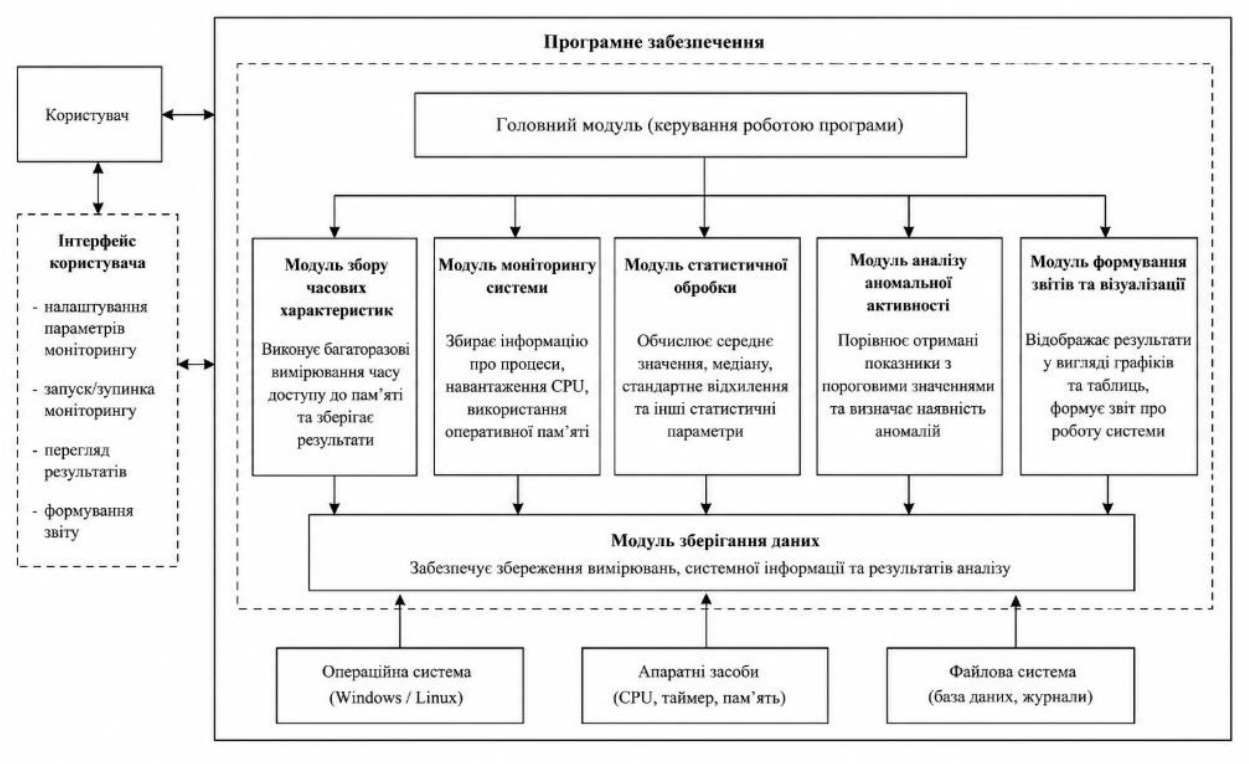

Рисунок 3.1 – Структурна схема програмного забезпечення

3.2 Функціонування програмного забезпечення для виявлення потенційних атак спекулятивного виконання коду

Програмне забезпечення призначене для пошуку непрямих ознак, які можуть супроводжувати атаки спекулятивного виконання коду.

Основна увага в програмі приділяється аналізу часових характеристик доступу до пам'яті, оскільки саме зміна часу доступу до даних часто використовується в атаках через побічні канали.

Програма працює у режимі користувацького застосунку. Це означає, що для її запуску не потрібно змінювати ядро операційної системи, встановлювати спеціальні драйвери або вносити зміни до апаратної частини комп'ютера. Такий підхід був обраний свідомо, оскільки метою роботи є створення програмного засобу, який можна використати для експериментального аналізу поведінки системи на звичайному персональному комп'ютері.

На рисунку 3.2 наведено функціональну схему програмного забезпечення.

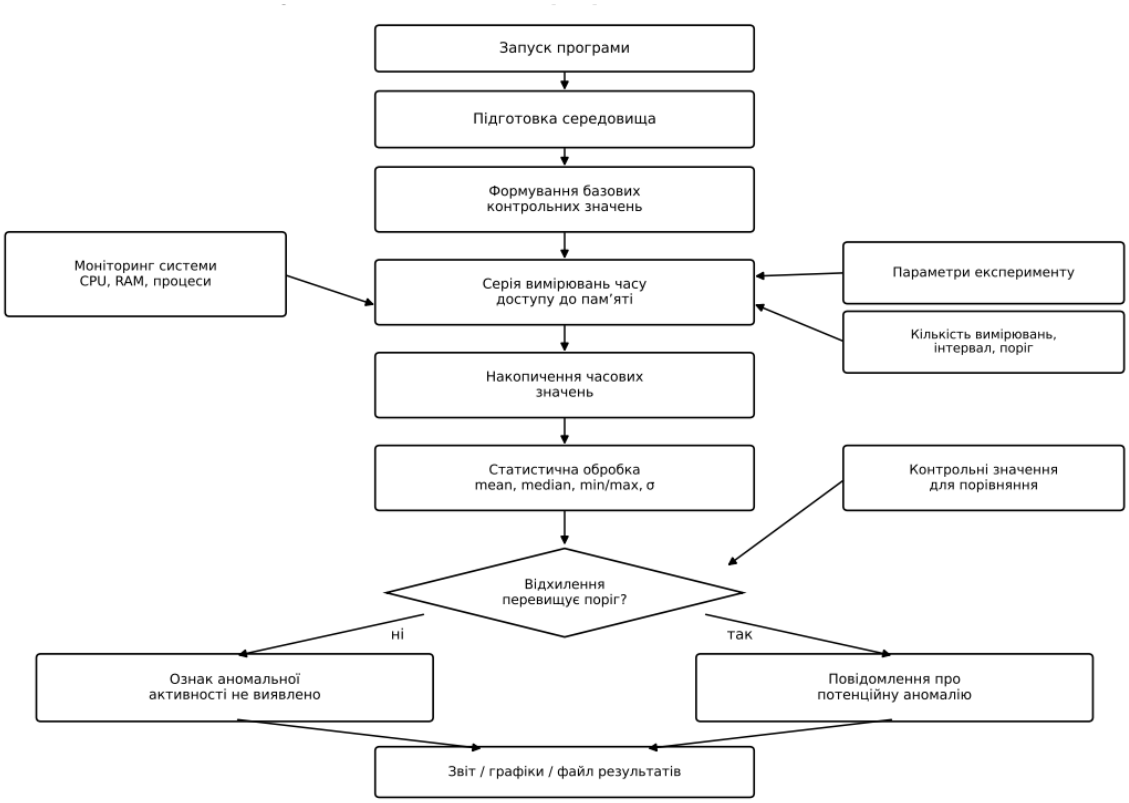

Рисунок 3.2 – Функціональна схема програмного забезпечення

Функціонування програмного забезпечення можна поділити на кілька послідовних етапів (рис.3.2.):

– підготовка середовища;

– отримання базових значень;

– проведення вимірювань;

– моніторинг стану системи;

– статистична обробка результатів;

– виявлення відхилень;

– формування підсумкового результату.

На першому етапі виконується підготовка програмного середовища. Після запуску програма перевіряє наявність необхідних бібліотек, ініціалізує основні змінні та створює структури для збереження результатів вимірювань.

До таких структур належать:

– масиви часових значень;

– змінні для збереження середнього часу доступу;

– медіани мінімального та максимального значення, стандартного відхилення.

Окремо на цьому етапі задаються параметри експерименту:

– кількість вимірювань у серії;

– кількість повторень серій;

– інтервал між вимірюваннями;

– порогове значення, за яким надалі буде оцінюватися наявність аномальної поведінки.

Ці параметри важливі, оскільки занадто мала кількість вимірювань може дати випадковий результат, а занадто велика – збільшити час роботи програми без суттєвого покращення точності.

Другим етапом є отримання базових, або контрольних, значень. Перед основним аналізом програма повинна визначити, які часові характеристики є типовими для конкретної системи у нормальному режимі роботи. Для цього виконується серія вимірювань без створення додаткового навантаження та без запуска тестових сценаріїв, що імітують підозрілу активність.

Отримані контрольні значення використовуються як основа для подальшого порівняння. Це важливо, оскільки різні комп’ютери можуть мати різні показники часу доступу до пам’яті. На результат впливають модель процесора, обсяг кеш-пам’яті, частота роботи процесора, стан операційної системи та поточне навантаження. Тому використання одного універсального порогу для всіх систем було б некоректним.

Після формування базових значень програма переходить до основного циклу вимірювань. На цьому етапі багаторазово виконується операція доступу до визначеної ділянки пам’яті. Перед зверненням до пам’яті фіксується початковий час, після завершення операції – кінцевий час. Різниця між цими значеннями записується як час виконання конкретного доступу.

Один результат вимірювання не використовується як самостійна підстава для висновку. Це пов’язано з тим, що операційна система постійно виконує фонові процеси, перемикає потоки, обслуговує системні задачі та розподіляє ресурси між активними програмами. Через це окреме значення може бути випадково завищеним або заниженим. Тому програма накопичує не одне значення, а цілу серію результатів.

Під час накопичення даних кожне вимірювання зберігається у загальному списку часових характеристик. Надалі цей список використовується для статистичної обробки. Такий підхід дозволяє оцінювати не окрему випадкову подію, а загальну поведінку системи протягом певного проміжку часу.

Паралельно з вимірюванням часу доступу до пам’яті виконується моніторинг загального стану системи. Програма фіксує поточне навантаження на процесор, використання оперативної пам’яті та кількість активних процесів. Ці дані не є прямою ознакою атаки, але вони допомагають пояснити можливі відхилення в результатах вимірювань.

Наприклад, якщо під час експерименту різко зростає завантаження процесора, то збільшення часу доступу до пам’яті може бути пов’язане не з атакою, а зі звичайним навантаженням системи. Саме тому інформація про стан системи використовується як допоміжна. Вона дозволяє обережніше інтерпретувати результати та не робити висновок про аномалію лише на основі одного показника.

Після завершення серії вимірювань програма переходить до статистичної обробки результатів. Спочатку обчислюється середнє значення часу доступу до пам’яті. Воно показує загальний рівень часових характеристик за весь період вимірювання. Якщо середнє значення суттєво відрізняється від контрольного, це може свідчити про зміну поведінки системи.

Далі визначається медіанне значення. Медіана використовується для того, щоб зменшити вплив окремих випадкових відхилень. Якщо в серії

вимірювань є кілька дуже великих або дуже малих значень, вони можуть помітно вплинути на середнє значення, але значно менше впливають на медіану.

Окремо обчислюються мінімальне та максимальне значення часу доступу. Ці показники дають можливість оцінити межі, в яких змінювалися результати під час виконання експерименту. Велика різниця між мінімальним і максимальним значенням може свідчити про нестабільність вимірювань.

Важливим параметром є стандартне відхилення. Саме воно дозволяє оцінити, наскільки сильно окремі результати відрізняються від середнього значення. Якщо стандартне відхилення невелике, то результати вимірювань є відносно стабільними. Якщо ж воно значно зростає, це може бути ознакою нестабільної або нетипової поведінки системи.

На наступному етапі виконується порівняння поточних результатів із контрольними значеннями. Програма аналізує не тільки середній час доступу, а й розкид результатів. Це важливо, оскільки під час атак через побічні канали змінюватися може не лише середній час, а й характер розподілу часових значень.

Логіка прийняття рішення в програмі побудована за пороговим принципом. Якщо середній час доступу або стандартне відхилення перевищують контрольні значення на певний коефіцієнт, програма фіксує потенційну аномалію. При цьому повідомлення формується саме як попередження про можливі ознаки аномальної активності, а не як остаточне твердження про наявність конкретної атаки.

Такий підхід є більш коректним для даної роботи, оскільки програма аналізує непрямі ознаки. Вона не має доступу до внутрішньої мікроархітектури процесора на рівні, який дозволив би однозначно підтвердити виконання конкретної атаки. Програма визначає, чи відрізняється поведінка системи від нормального режиму за часовими параметрами доступу до пам’яті.

Після завершення аналізу програма формує результат роботи. Якщо відхилення не перевищують встановлені межі, система повідомляє, що суттєвих ознак аномальної активності не виявлено. Якщо ж один або кілька показників виходять за межі допустимих значень, формується повідомлення про потенційно підозрілу поведінку.

Результати можуть бути подані у текстовому вигляді: кількість вимірювань, середній час доступу, медіана, мінімальне та максимальне значення, стандартне відхилення, а також висновок програми. За потреби ці дані можуть зберігатися у файлі для подальшого аналізу або порівняння з іншими експериментами.

Окремим елементом роботи програми є візуалізація результатів. Побудова графіків дозволяє наочно побачити, як змінювалися часові характеристики під час виконання вимірювань. Це особливо корисно під час експериментального дослідження, оскільки на графіку легше помітити різкі стрибки, нестабільні ділянки або відхилення від звичайної поведінки системи.

У загальному вигляді роботу програми можна описати так: спочатку система визначає нормальні часові параметри, потім виконує основний цикл вимірювань, після цього обробляє отримані дані та порівнює їх із базовими значеннями. Якщо поведінка системи залишається стабільною, програма не формує попередження. Якщо ж спостерігаються стійкі відхилення, система повідомляє про можливу аномалію.

Перевагою такого підходу є його простота та зрозумілість. Програма не потребує складного налаштування, а її логіку можна легко перевірити під час тестування. Разом із тим вона дозволяє дослідити важливу для цієї роботи проблему – зміну часових характеристик доступу до пам’яті як можливу ознаку атак спекулятивного виконання.

Водночас необхідно враховувати обмеження запропонованого підходу. Програма не може гарантувати точне виявлення всіх атак цього класу. Її задача полягає у фіксації потенційно підозрілих відхилень, які

потребують подальшого аналізу. Саме тому результати роботи програмного забезпечення слід розглядати як допоміжний інструмент моніторингу, а не як повноцінну систему захисту промислового рівня.

Таким чином, функціонування розробленого програмного забезпечення базується на поєднанні трьох основних складових: вимірювання часу доступу до пам’яті, статистичної обробки результатів та порівняння отриманих значень із контрольними показниками. Така логіка відповідає поставленій меті дипломної роботи і дозволяє програмно виявляти непрямі ознаки аномальної поведінки, характерної для атак спекулятивного виконання коду.

3.3 Проєктування інтерфейсу програмного забезпечення

Після завершення аналізу було прийнято рішення використовувати консольний інтерфейс з підтримкою графічної візуалізації результатів. Саме цей варіант дозволяє забезпечити достатню інформативність системи без зайвого ускладнення структури програмного забезпечення.

Під час проєктування інтерфейсу було визначено декілька основних задач, які він повинен забезпечувати:

– запуск процесу аналізу;

– відображення параметрів експерименту;

– виведення поточного стану вимірювань;

– відображення статистичних характеристик;

– повідомлення про виявлені аномалії;

– побудову графіків результатів;

– збереження результатів аналізу.

Структурно інтерфейс програмного забезпечення складається з декількох логічних частин:

– інформаційної області;

– області моніторингу;

– області статистичного аналізу;

– області повідомлень системи;

– області виведення підсумкових результатів.

Після запуску програми користувач отримує початкову інформацію про стан системи та параметри експерименту. На цьому етапі інтерфейс відображає:

– кількість запланованих вимірювань;

– інтервал між операціями;

– встановлене порогове значення;

– поточне завантаження процесора;

– обсяг використаної оперативної пам’яті;

– стан запуску модулів системи (рис. 3.3).

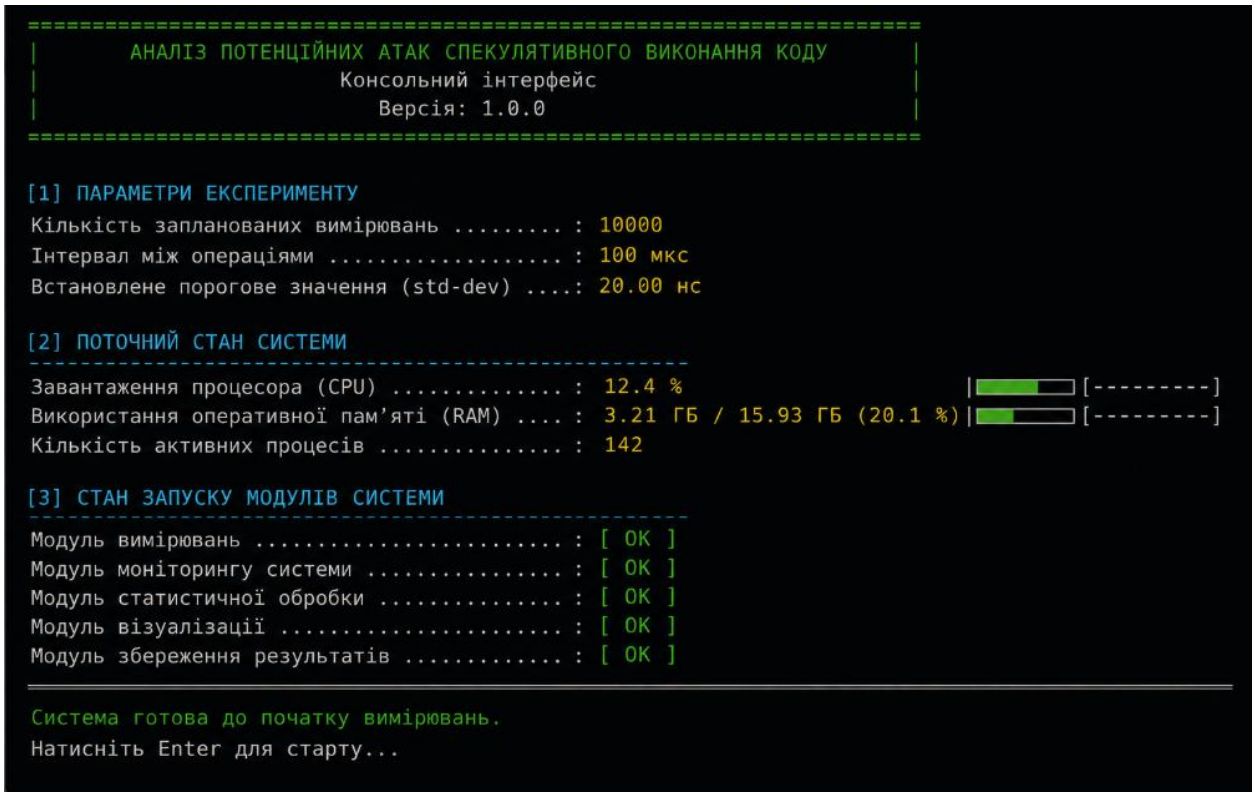

Рисунок 3.3 – Консольний інтерфейс вікна запуску програмного забезпечення

Такий підхід дозволяє ще до початку вимірювань оцінити, чи готова система до проведення експерименту.

Після завершення ініціалізації програма переходить у режим збору часових характеристик. У процесі виконання вимірювань інтерфейс в режимі реального часу відображає хід роботи системи.

Користувач може бачити:

– номер поточного вимірювання;

– поточне значення часу доступу до пам’яті;

– середнє значення для поточної серії;

– зміну статистичних параметрів;

– стан накопичення результатів (рис. 3.4).

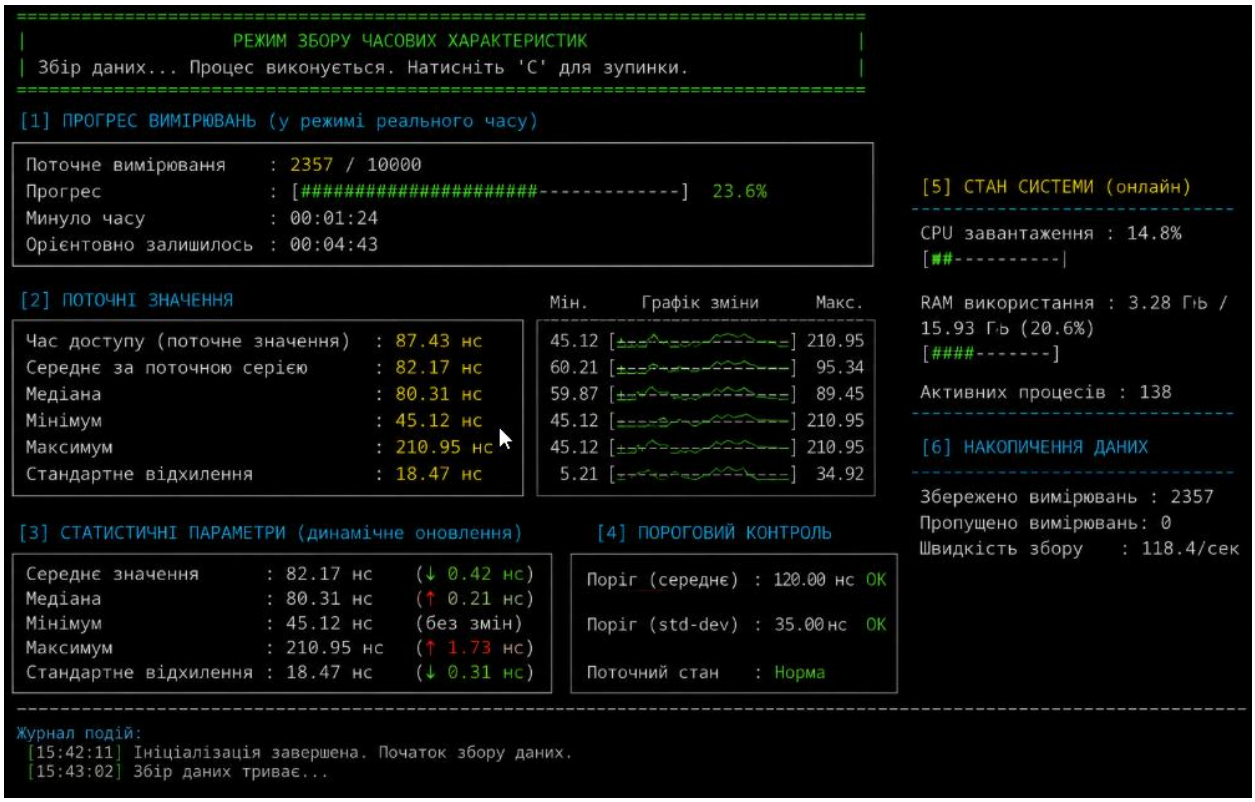

Рисунок 3.4 – Робоча панель для моніторингу даних

Відображення проміжних результатів є важливим, оскільки дозволяє контролювати стабільність роботи програми без необхідності очікувати завершення всього циклу вимірювань.

Окрему увагу під час проєктування було приділено системі повідомлень. У процесі роботи програма може виводити:

– інформаційні повідомлення;

– службові повідомлення;

– попередження;

– повідомлення про потенційну аномалію.

Інформаційні повідомлення використовуються для відображення поточного етапу роботи системи. Наприклад, програма повідомляє про початок вимірювань, завершення накопичення результатів або запуск статистичної обробки (рис. 3.5).

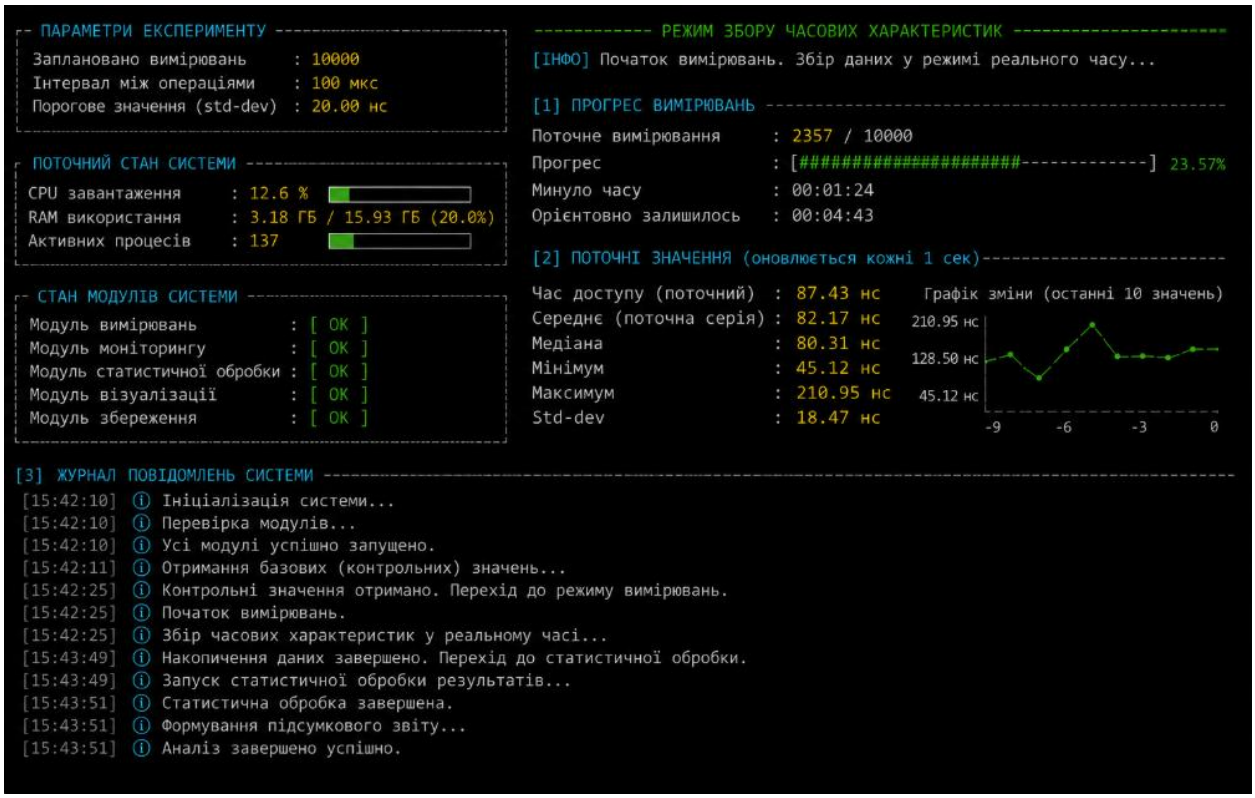

Рисунок 3.5 – Аналіз спекулятивних атак: панель моніторингу

Службові повідомлення призначені для відображення технічної інформації, необхідної під час тестування програмного забезпечення. До таких повідомлень належать результати перевірки бібліотек, параметри запуску та внутрішні значення системи (рис. 3.6).

Попередження формуються у випадку нестабільної роботи системи. Наприклад, якщо під час виконання вимірювань різко збільшується

навантаження процесора або кількість активних процесів, програма повідомляє про можливий вплив сторонніх факторів на результати експерименту.

Рисунок 3.6 – Аналіз системи в режимі реального часу

Повідомлення про потенційну аномалію формується у випадку, коли статистичні параметри стабільно перевищують контрольні значення. При цьому інтерфейс не повідомляє про наявність конкретної атаки, а лише вказує на виявлення нетипової поведінки системи (рис. 3.7).

Для зручності аналізу результати роботи програми можуть виводитись не тільки у текстовому вигляді, а й у формі графіків. Саме графічне представлення дозволяє найбільш наочно оцінити поведінку часових характеристик під час виконання вимірювань.

Під час проєктування візуальної частини системи було вирішено використовувати прості графіки без надлишкових елементів оформлення. Основна увага приділяється саме інформативності результатів.

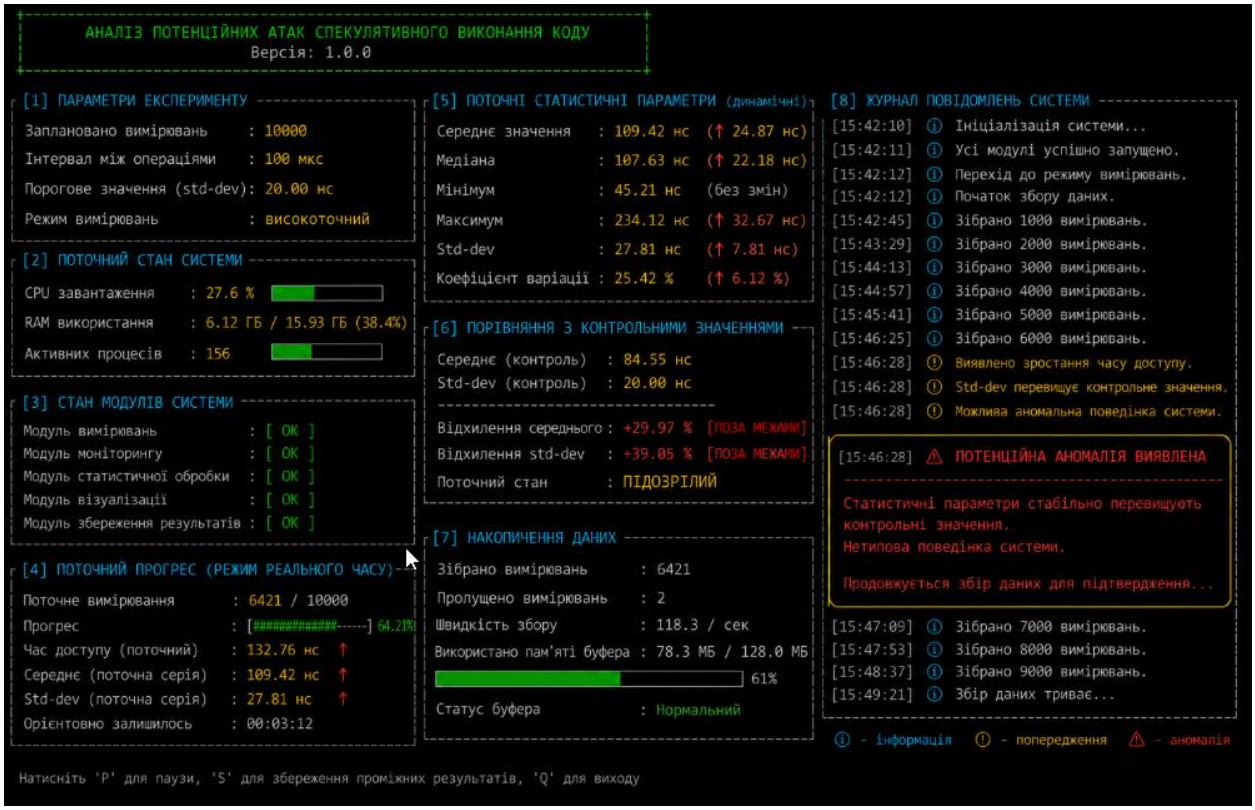

Рисунок 3.7 – Аналіз потенційних атак спекуляції

На графіках відображаються:

– зміна часу доступу до пам’яті;

– розподіл часових значень;

– середній рівень часових характеристик;

– порогові значення;

– аномальні ділянки вимірювань.

Побудова графіків виконується автоматично після завершення серії вимірювань. За необхідності користувач може зберегти результати у файл для подальшого аналізу або порівняння з іншими експериментами.

Під час проєктування інтерфейсу також враховувалася можливість подальшого розвитку програмного забезпечення. Структура взаємодії між модулями була реалізована таким чином, щоб у майбутньому можна було додати:

– графічний desktop-інтерфейс;

– вебпанель моніторингу;

– автоматичне логування результатів;

– систему журналювання подій;

– розширену систему налаштувань.

Важливим фактором стала і мінімізація навантаження інтерфейсу на систему. Оскільки програма виконує велику кількість вимірювань часу доступу до пам’яті, надмірно складний інтерфейс міг би сам впливати на результати аналізу. Саме тому було обрано максимально простий та легкий режим взаємодії з користувачем.

Таким чином, під час проєктування інтерфейсу програмного забезпечення основна увага приділялася забезпеченню інформативності, простоти використання та мінімального впливу інтерфейсу на результати вимірювань.

Реалізований підхід дозволяє ефективно контролювати процес аналізу часових характеристик доступу до пам’яті та забезпечує зручне представлення результатів експериментального дослідження.

4 ЕКСПЛУАТАЦІЯ ТА ТЕСТУВАННЯ ПРОГРАМИ

4.1 Призначення програми

Розроблене програмне забезпечення призначене для аналізу часових характеристик доступу до пам’яті та виявлення непрямих ознак потенційно аномальної активності, яка може бути пов’язана з використанням механізмів спекулятивного виконання коду у сучасних обчислювальних системах.

Основною задачею програми є автоматизований збір показників часу виконання операцій доступу до пам’яті, подальша статистична обробка отриманих значень та визначення випадків відхилення від характерної поведінки системи.

Програмний засіб не виконує безпосереднього підтвердження факту реалізації атаки типу Spectre, Meltdown або їх модифікацій, а використовується як інструмент моніторингу та виявлення потенційно підозрілих змін у поведінці обчислювального середовища.

Програма орієнтована на виконання дослідницьких та навчальних задач у сфері інформаційної безпеки, пов’язаних із дослідженням побічних каналів витоку інформації та аналізом особливостей роботи сучасних процесорних архітектур.

Використання програмного забезпечення дозволяє отримувати експериментальні дані щодо зміни часових параметрів доступу до пам’яті та виконувати їх подальший аналіз.

Під час роботи програма здійснює послідовне виконання етапів моніторингу: ініціалізацію середовища (рис. 4.1), запуск збору показників (рис. 4.2), обчислення статистичних характеристик та формування підсумкового результату аналізу (рис. 4.3). Отримані дані можуть використовуватися для оцінювання стабільності роботи системи та проведення експериментальних досліджень у сфері виявлення атак побічних каналів.

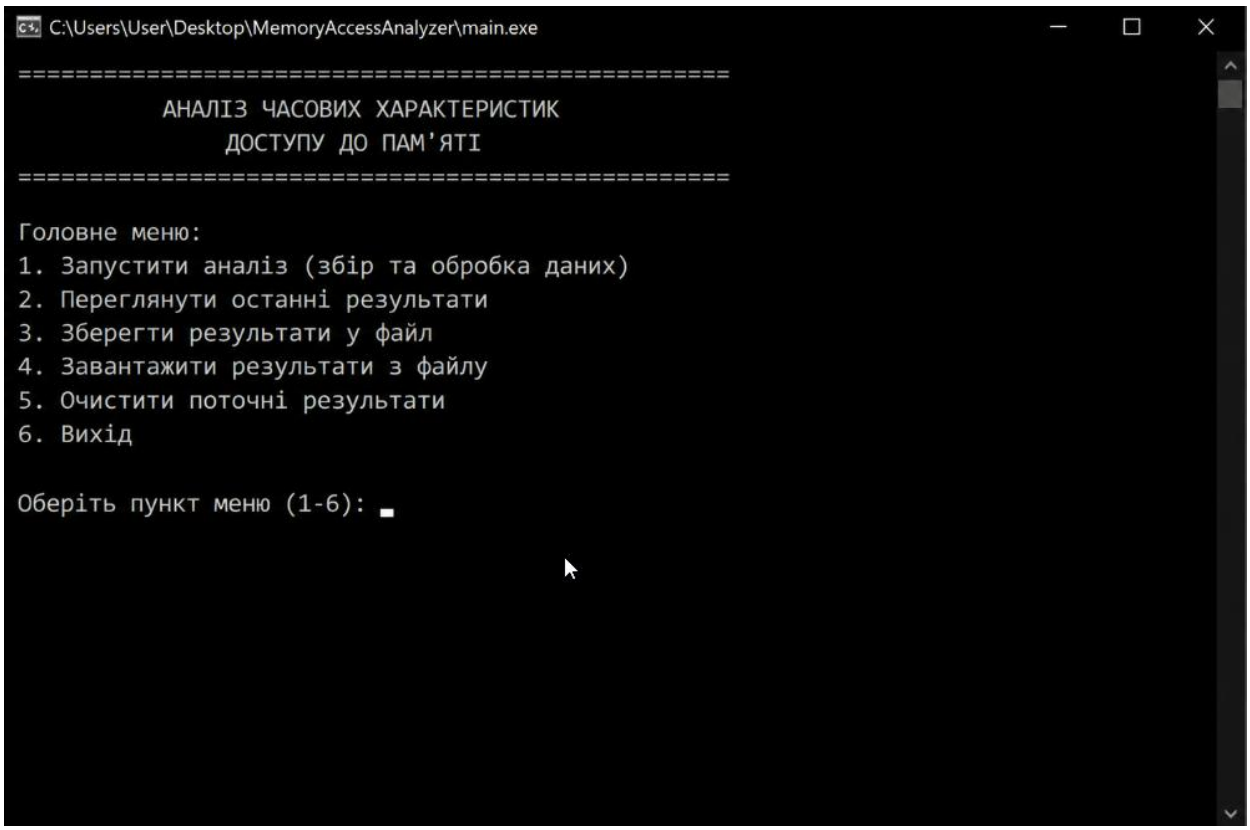

Рисунок 4.1 – Головне вікно програми після запуску

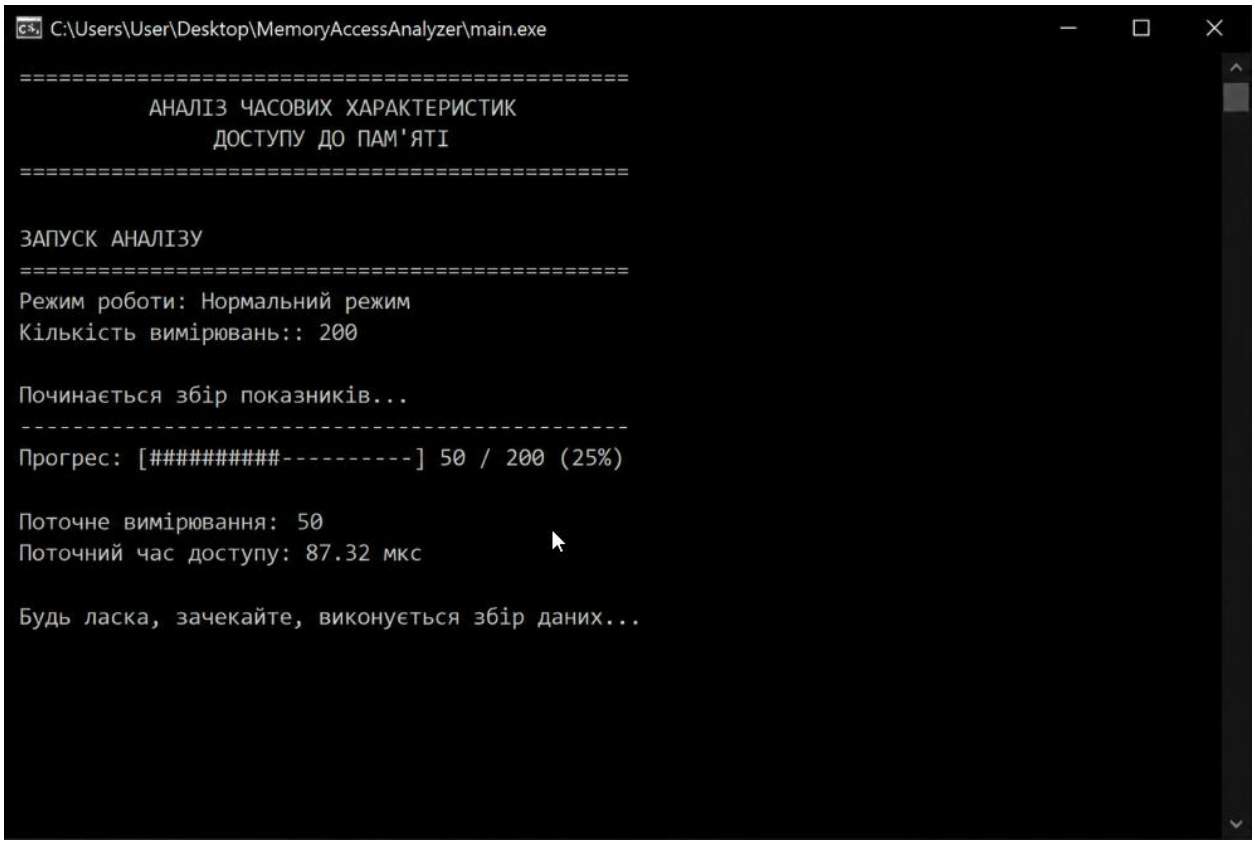

Рисунок 4.2 – Запуск збору показників

На рисунку 4.3 наведено формування підсумкового результату аналізу.

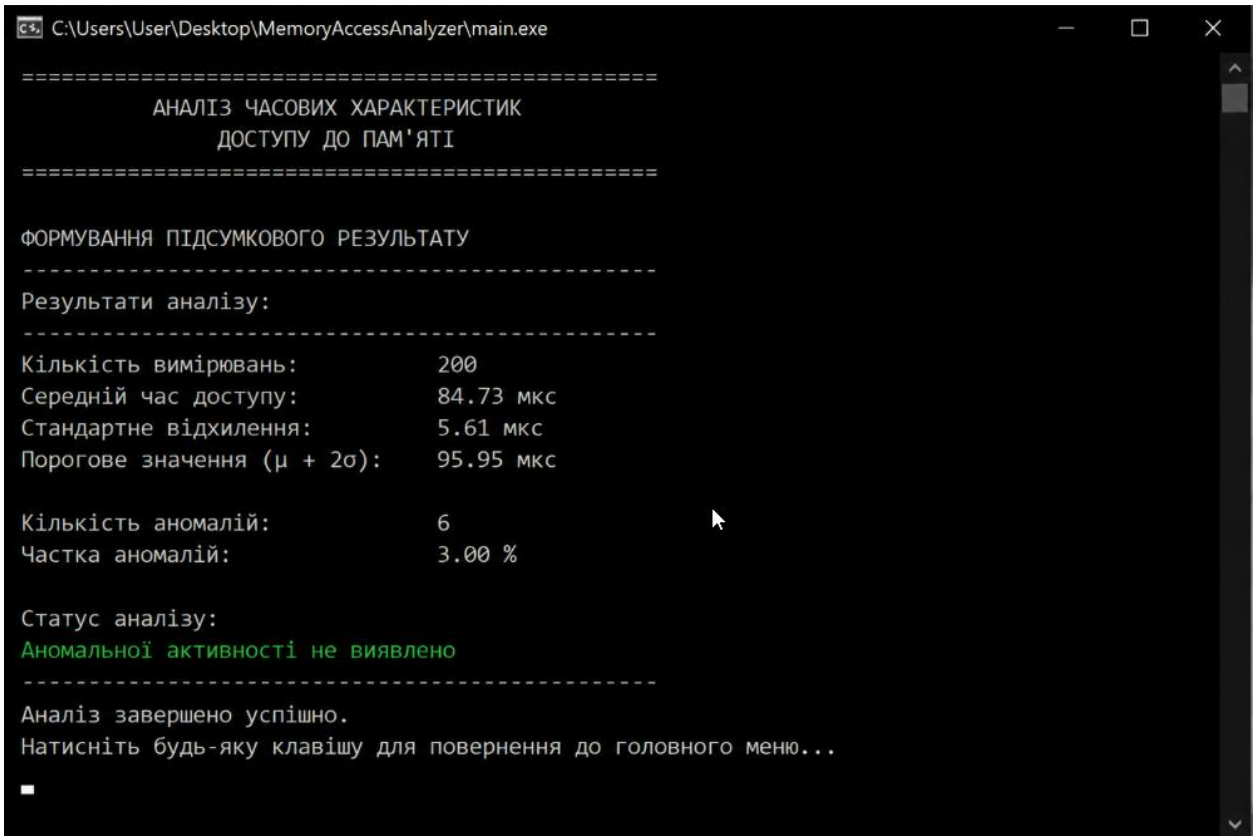

Рисунок 4.3 – Формування підсумкового результату аналізу

4.2 Умови виконання програми

Розроблене програмне забезпечення призначене для виконання на персональному комп’ютері під керуванням сучасної операційної системи та не потребує використання спеціалізованих апаратних засобів. Робота програми здійснюється у середовищі виконання мови програмування Python із використанням додаткових бібліотек для збору системних показників, статистичної обробки результатів та візуалізації отриманих даних.

Для коректного функціонування програмного забезпечення необхідна наявність встановленого інтерпретатора Python версії не нижче 3.10. Розробка та перевірка працездатності програми виконувались із використанням середовища розробки Visual Studio Code, що забезпечує

можливість запуску, налагодження та аналізу результатів роботи програмного коду.

Програмний засіб використовує стандартні програмні бібліотеки та зовнішні модулі, необхідні для отримання інформації про характеристики роботи системи та виконання статистичних розрахунків. До складу програмного оточення входять такі бібліотеки:

– psutil – використовується для отримання інформації про параметри роботи процесів та системних ресурсів;

– statistics – застосовується для обчислення статистичних показників та аналізу вибірки отриманих значень;

– matplotlib – використовується для побудови графічного представлення результатів аналізу.

Для коректного запуску програми необхідно забезпечити виконання таких умов:

– наявність встановленого інтерпретатора Python;

– встановлення необхідних бібліотек програмного середовища;

– наявність доступу до системних ресурсів для виконання вимірювань;

– достатній обсяг оперативної пам’яті для накопичення результатів аналізу.

Підготовка програмного середовища до роботи виконується шляхом встановлення необхідних залежностей за допомогою менеджера пакетів Python. Після завершення встановлення користувач отримує можливість запускати програму без додаткового налаштування системи.

Розроблене програмне забезпечення не потребує постійного підключення до мережі Інтернет, оскільки всі операції збору показників, статистичного аналізу та побудови результатів виконуються локально на обчислювальному пристрої.

Рекомендовані технічні характеристики комп’ютера для роботи програми:

– процесор із тактовою частотою не нижче 2,0 ГГц;

– оперативна пам’ять обсягом не менше 4 Гб;

– не менше 500 Мб вільного місця на накопичувачі;

– операційна система сімейства Linux або Microsoft Windows.

Використання наведених параметрів забезпечує стабільне виконання процедури моніторингу та дозволяє отримувати результати аналізу без помітного впливу на роботу операційної системи.

4.3 Виконання і тестування програми

Після завершення розробки програмного забезпечення було проведено його експериментальну перевірку з метою оцінювання коректності роботи алгоритму збору часових характеристик доступу до пам’яті та перевірки можливості виявлення аномальних відхилень у роботі системи.

На відміну від класичного функціонального тестування, у даній роботі основна увага приділялась дослідженню поведінкових характеристик системи та аналізу статистичних параметрів отриманих вимірювань.

Перед запуском програми виконувалась підготовка робочого середовища. На комп’ютері було встановлено операційну систему Linux, середовище виконання Python та необхідні програмні бібліотеки. Для роботи програмного забезпечення використовувалися бібліотеки `psutil`, `statistics`, `matplotlib`, а також стандартні модулі `time` і `subprocess`, які забезпечують збір системної інформації та обробку результатів.

Після відкриття каталогу проєкту запуск програми здійснювався із використанням термінала операційної системи. Для запуску використовувалася команда:

```
python main.py
```

Після запуску програма виконувала початкову ініціалізацію модулів, перевіряла доступність необхідних системних ресурсів та переходила до етапу збору показників (рис.).

На першому етапі виконання відбувався запуск механізму моніторингу та починався послідовний збір часових характеристик доступу до пам’яті. У процесі роботи програма відображала службові повідомлення у консольному вікні та інформувала користувача про поточний стан виконання.

Після накопичення необхідної кількості вимірювань виконувався етап аналізу отриманих результатів. На цьому етапі здійснювалася обробка сформованої вибірки та визначення наявності потенційно аномальної активності.

У разі завершення аналізу користувачу виводилося підсумкове повідомлення із результатами роботи програми. Якщо під час виконання не було виявлено ознак нетипової поведінки системи, програма повідомляла про відсутність аномальної активності. У випадку виявлення відхилень формувалося повідомлення про потенційно підозрілий характер роботи системи.

Для перевірки працездатності було виконано декілька послідовних запусків програми. Під час тестування перевірялася коректність запуску, стабільність збору показників, відсутність помилок виконання та формування підсумкового результату.

4.4 Висновки за розділом 4

В четвертому розділі виконано практичну перевірку розробленого програмного забезпечення та досліджено особливості його роботи під час аналізу часових характеристик доступу до пам’яті.

Визначено умови виконання програми, описано порядок її запуску та проведено експериментальне тестування у різних режимах роботи системи.

Під час тестування програмного забезпечення було підтверджено можливість фіксації стабільних відхилень часових характеристик від контрольних значень у випадках підвищеного навантаження на систему та виконання сценаріїв, пов’язаних із інтенсивним доступом до пам’яті.

Таким чином, поставлені для даного розділу задачі виконані, а реалізоване програмне забезпечення підтвердило можливість використання запропонованого підходу для дослідження часових характеристик доступу до пам’яті під час аналізу проявів атак спекулятивного виконання коду.

ВИСНОВКИ

В результаті виконання дипломної кваліфікаційної роботи було розглянуто особливості атак спекулятивного виконання коду та реалізовано програмне забезпечення для виявлення непрямих ознак аномальної поведінки системи на основі аналізу часових характеристик доступу до пам’яті.

Під час виконання роботи було проаналізовано принципи функціонування атак класу Spectre та Meltdown, особливості використання побічних каналів витоку інформації, а також сучасні підходи до виявлення аномальної активності в комп’ютерних системах. Окрему увагу приділено аналізу часових характеристик доступу до пам’яті.

В процесі виконання роботи було встановлено, що одним з найбільш характерних непрямих проявів атак спекулятивного виконання є зміна часових параметрів доступу до пам’яті та нестабільність роботи кеш-пам’яті процесора. Саме тому для реалізації програмного засобу було обрано підхід, заснований на багаторазовому вимірюванні часу виконання операцій та подальшому статистичному аналізі отриманих результатів.

У роботі обґрунтовано вибір мови програмування Python, операційної системи Linux та засобів моніторингу системи. Використання Python дозволило спростити реалізацію алгоритмів аналізу та статистичної обробки даних, а Linux забезпечив можливість роботи із системними засобами моніторингу та виконання експериментальних досліджень.

В межах дипломної роботи було спроєктовано структуру програмного забезпечення та реалізовано основні функціональні модулі системи: модуль збору часових характеристик, модуль моніторингу системи, модуль статистичної обробки результатів, модуль аналізу аномальної активності та модуль візуалізації даних.

Розроблене програмне забезпечення забезпечує:

– виконання багаторазових вимірювань часу доступу до пам’яті;

– накопичення та статистичну обробку результатів;

– аналіз середніх значень та стандартного відхилення;

– порівняння отриманих параметрів із контрольними значеннями;

– виявлення непрямих ознак потенційно аномальної поведінки системи;

– відображення результатів аналізу у текстовому та графічному вигляді.

Отримані результати показали, що використання статистичного аналізу часових характеристик дозволяє виявляти непрямі ознаки нетипової поведінки системи без необхідності модифікації апаратної частини комп’ютера або ядра операційної системи.

Розроблений програмний засіб не є повноцінною системою захисту від атак спекулятивного виконання коду та не виконує прямого визначення конкретних експлойтів Spectre або Meltdown. Основною задачею системи є виявлення аномальних відхилень, які можуть свідчити про потенційно небезпечну активність та потребують додаткового аналізу.

Практичне значення отриманих результатів полягає у можливості використання розробленого програмного забезпечення для експериментального дослідження часових характеристик доступу до пам’яті, тестування алгоритмів аналізу аномалій та навчальних задач у сфері кібербезпеки.

Поставлену мету дипломної роботи можна вважати досягнутою, а розроблене програмне забезпечення – придатним для проведення досліджень у сфері виявлення непрямих ознак атак спекулятивного виконання коду.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Гнатюк С. О., Сидоренко В. М. Аналіз сучасних кіберзагроз апаратного рівня. Сучасний захист інформації. 2022. № 1. С. 34–42.

2. Капустян В. О., Гавриленко С. Ю. Аналіз сучасних атак через побічні канали витоку інформації. Захист інформації. 2021. Т. 23, № 2. С. 85–94.

3. Stallings W. Computer Organization and Architecture : Designing for Performance. 11th ed. London : Pearson, 2022. 896 p.

4. Мельник А. О. Архітектура комп’ютерних систем : навч. посіб. Львів : Магнолія-2006, 2019. 432 с.

5. Kocher P., Horn J., Fogh A., Genkin D., Gruss D., Haas W., Hamburg M., Lipp M., Mangard S., Prescher T., Schwarz M., Yarom Y. Spectre Attacks: Exploiting Speculative Execution. 2019 IEEE Symposium on Security and Privacy (SP). San Francisco, 2019. P. 1–19.

6. Lipp M., Schwarz M., Gruss D., Prescher T., Haas W., Fogh A., Horn J., Mangard S., Kocher P., Genkin D., Yarom Y., Hamburg M. Meltdown: Reading Kernel Memory from User Space. 27th USENIX Security Symposium. Baltimore, 2018. P. 973–990.

7. Yarom Y., Falkner K. FLUSH+RELOAD: A High Resolution, Low Noise, L3 Cache Side-Channel Attack. 23rd USENIX Security Symposium. San Diego, 2014. P. 719–732.

8. Gruss D., Maurice C., Wagner K., Mangard S. Flush+Flush: A Fast and Stealthy Cache Attack. Detection of Intrusions and Malware, and Vulnerability Assessment. Cham : Springer, 2016. P. 279–299.

9. Гнатюк С. О., Сидоренко В. М. Аналіз сучасних кіберзагроз апаратного рівня. Сучасний захист інформації. 2022. № 1. С. 34–42.

10. Kocher P., Horn J., Fogh A., Genkin D., Gruss D., Haas W., Hamburg M., Lipp M., Mangard S., Prescher T., Schwarz M., Yarom Y. Spectre Attacks:

Exploiting Speculative Execution. 2019 IEEE Symposium on Security and Privacy (SP). San Francisco, 2019. P. 1–19.

11. Yarom Y., Falkner K. FLUSH+RELOAD: A High Resolution, Low Noise, L3 Cache Side-Channel Attack. 23rd USENIX Security Symposium. San Diego, 2014. P. 719–732.

12. Грищук Р. В. Теоретичні основи кібербезпеки : навч. посіб. Житомир : ЖНАЕУ, 2020. 407 с.

13. Корченко О. Г. Системи захисту інформації : монографія. Київ : НАУ, 2015. 520 с. Goehr L. The concept of opera. *The Oxford handbook of opera* / ed. by H. M. Greenwald. Oxford, 2014. P. 92–136.

14. Дудикевич В. Б. Захист інформації в комп’ютерних системах : навч. посіб. Львів : Вид-во Львівської політехніки, 2016. 360 с.

15. Глинський Я. М. Python для студентів та школярів : навч. посіб. Львів : СПД Глинський, 2021. 276 с.

16. Мельник А. О., Кравець П. О., Наконечний Ю. М. Системне програмування та операційні системи : навч. посіб. Львів : Вид-во Львівської політехніки, 2018. 416 с.

17. Гнатюк С. О., Юдін О. К. Основи кібербезпеки та захисту інформації : навч. посіб. Київ : НАУ, 2021. 304 с.

18. Таненбаум Е., Бос Г. Сучасні операційні системи. Київ : Видавнича група BHV, 2020. 1120 с.

19. Кобзар А. І. Прикладна математична статистика з використанням Excel і STATISTICA : навч. посіб. Київ : Центр учбової літератури, 2019. 328 с.

20. Microsoft. Visual Studio Code Documentation. URL: https://code.visualstudio.com/docs (date of access: 08.05.2026).

21. Python Software Foundation. pip documentation. URL: https://pip.pypa.io/en/stable/ (date of access: 08.05.2026).

22. Лозинський В. Т., Шека В. І. Python для інженерів та науковців : навч. посіб. Львів : Новий Світ – 2000, 2021. 312 с.

ДОДАТОК А

Текст програми

А.1 Текст файлу main.cpp

from __future__ import annotations

```
import argparse
import csv
import os
import platform
import statistics
import sys
import time
from dataclasses import dataclass
from datetime import datetime
from pathlib import Path
from typing import List, Optional


try:
import psutil
except ImportError:
psutil = None


try:
import matplotlib.pyplot as plt
except ImportError:
plt = None


@dataclass
class ExperimentConfig:
measurements: int = 10_000
baseline_runs: int = 2_000
interval_us: int = 100
threshold_factor: float = 1.35
buffer_size_mb: int = 64
output_dir: Path = Path("results")


@dataclass
```

```
class StatisticsResult:
count: int
mean_ns: float
median_ns: float
min_ns: int
max_ns: int
stdev_ns: float


@dataclass
class SystemSnapshot:
cpu_percent: float
ram_used_gb: float
ram_total_gb: float
ram_percent: float
process_count: int


@dataclass
class AnalysisResult:
mean_deviation_percent: float
stdev_deviation_percent: float
anomaly_detected: bool
conclusion: str


def now_str() -> str:
return datetime.now().strftime("%H:%M:%S")


def log_info(message: str) -> None:
print(f"[{now_str()}] [INFO] {message}")


def log_service(message: str) -> None:
print(f"[{now_str()}] [SERVICE] {message}")


def log_warning(message: str) -> None:
print(f"[{now_str()}] [WARNING] {message}")
```

```
def clear_screen() -> None:
os.system("cls" if os.name == "nt" else "clear")


class SystemMonitor:
"""Модуль моніторингу стану системи."""

def get_snapshot(self) -> SystemSnapshot:
if psutil is None:
return SystemSnapshot(0.0, 0.0, 0.0, 0.0, 0)

cpu = psutil.cpu_percent(interval=0.05)
mem = psutil.virtual_memory()

return SystemSnapshot(
cpu_percent=cpu,
ram_used_gb=mem.used / (1024 ** 3),
ram_total_gb=mem.total / (1024 ** 3),
ram_percent=mem.percent,
process_count=len(psutil.pids()),
)


class MemoryTimingMeter:
"""
Модуль вимірювання часу доступу до пам'яті.

Програма не містить експлойтів і не читає захищену пам'ять.
Вимірюється лише час доступу до власного буфера bytearray.
"""

def __init__(self, buffer_size_mb: int) -> None:
self.buffer_size = buffer_size_mb * 1024 * 1024
self.buffer = bytearray(self.buffer_size)
self.index = 0
self.step = 4096

for i in range(0, self.buffer_size, self.step):
self.buffer[i] = i % 256
```

```
def memory_access(self) -> int:
value = self.buffer[self.index]
self.index = (self.index + self.step) % self.buffer_size
return value

def measure_once(self) -> int:
start = time.perf_counter_ns()
_ = self.memory_access()
end = time.perf_counter_ns()
return end - start

def measure_series(
self,
count: int,
interval_us: int,
show_progress: bool = True
) -> List[int]:
results: List[int] = []
interval_sec = interval_us / 1_000_000

for i in range(1, count + 1):
results.append(self.measure_once())

if show_progress and (
i == 1 or i % max(1, count // 20) == 0 or i == count
):
percent = i / count * 100
current = results[-1]
mean_value = statistics.mean(results)

sys.stdout.write(
f"\rПоточне вимірювання: {i}/{count} | "
f"Прогрес: {percent:6.2f}% | "
f"Поточне: {current:8d} нс | "
f"Середнє: {mean_value:8.2f} нс"
)
sys.stdout.flush()

if interval_sec > 0:
time.sleep(interval_sec)
```

```
if show_progress:
print()

return results


class StatisticsAnalyzer:
"""Модуль статистичної обробки результатів."""

@staticmethod
def calculate(values: List[int]) -> StatisticsResult:
if not values:
raise ValueError("Список вимірювань порожній.")

stdev_value = statistics.stdev(values) if len(values) > 1 else 0.0

return StatisticsResult(
count=len(values),
mean_ns=statistics.mean(values),
median_ns=statistics.median(values),
min_ns=min(values),
max_ns=max(values),
stdev_ns=stdev_value,
)


class AnomalyDetector:
"""Модуль порогового аналізу аномалій."""

def __init__(self, threshold_factor: float) -> None:
self.threshold_factor = threshold_factor

def analyze(
self,
baseline: StatisticsResult,
current: StatisticsResult,
snapshot: SystemSnapshot,
) -> AnalysisResult:
mean_limit = baseline.mean_ns * self.threshold_factor
stdev_limit = baseline.stdev_ns * self.threshold_factor
```

```
mean_deviation = self._percent_change(
current.mean_ns,
baseline.mean_ns
)
stdev_deviation = self._percent_change(
current.stdev_ns,
baseline.stdev_ns
)

mean_exceeded = current.mean_ns > mean_limit
stdev_exceeded = (
current.stdev_ns > stdev_limit
if baseline.stdev_ns > 0
else False
)

anomaly = mean_exceeded or stdev_exceeded

if anomaly:
conclusion = (
"Виявлено потенційно аномальну поведінку системи. "
"Статистичні параметри перевищують контрольні значення. "
"Це не є підтвердженням конкретної атаки, але потребує "
"додаткового аналізу."
)
else:
conclusion = (
"Суттєвих ознак аномальної активності не виявлено. "
"Поточні значення залишаються в межах контрольних параметрів."
)

if snapshot.cpu_percent > 80:
conclusion += (
" Увага: під час вимірювань зафіксовано високе "
"навантаження CPU, що могло вплинути на результати."
)

return AnalysisResult(
mean_deviation_percent=mean_deviation,
stdev_deviation_percent=stdev_deviation,
anomaly_detected=anomaly,
```

```
conclusion=conclusion,
)

@staticmethod
def _percent_change(current: float, baseline: float) -> float:
if baseline == 0:
return 0.0
return (current - baseline) / baseline * 100


class ResultReporter:
"""Модуль збереження звіту, CSV та графіка."""

def __init__(self, output_dir: Path) -> None:
self.output_dir = output_dir
self.output_dir.mkdir(parents=True, exist_ok=True)

def save_csv(self, values: List[int], filename: str) -> Path:
path = self.output_dir / filename

with path.open("w", newline="", encoding="utf-8") as f:
writer = csv.writer(f, delimiter=";")
writer.writerow(["index", "access_time_ns"])

for i, value in enumerate(values, start=1):
writer.writerow([i, value])

return path

def save_report(
self,
baseline: StatisticsResult,
current: StatisticsResult,
snapshot: SystemSnapshot,
analysis: AnalysisResult,
) -> Path:
path = self.output_dir / "analysis_report.txt"

with path.open("w", encoding="utf-8") as f:
f.write("ЗВІТ ПРО РЕЗУЛЬТАТИ АНАЛІЗУ\n")
f.write("=" * 60 + "\n\n")
```

```
f.write("1. Контрольні значення\n")
self._write_stats(f, baseline)

f.write("\n2. Поточні значення\n")
self._write_stats(f, current)

f.write("\n3. Стан системи\n")
f.write(f"CPU: {snapshot.cpu_percent:.2f}%\n")
f.write(
f"RAM: {snapshot.ram_used_gb:.2f} ГБ / "
f"{snapshot.ram_total_gb:.2f} ГБ "
f"({snapshot.ram_percent:.2f}%)\n"
)
f.write(f"Активних процесів: {snapshot.process_count}\n")

f.write("\n4. Аналіз відхилень\n")
f.write(
f"Відхилення середнього: "
f"{analysis.mean_deviation_percent:.2f}%\n"
)
f.write(
f"Відхилення std-dev: "
f"{analysis.stdev_deviation_percent:.2f}%\n"
)

f.write("\n5. Висновок\n")
f.write(analysis.conclusion + "\n")

return path

@staticmethod
def _write_stats(file_obj, stats: StatisticsResult) -> None:
file_obj.write(f"Кількість вимірювань: {stats.count}\n")
file_obj.write(f"Середнє значення: {stats.mean_ns:.2f} нс\n")
file_obj.write(f"Медіана: {stats.median_ns:.2f} нс\n")
file_obj.write(f"Мінімум: {stats.min_ns} нс\n")
file_obj.write(f"Максимум: {stats.max_ns} нс\n")
file_obj.write(f"Стандартне відхилення: {stats.stdev_ns:.2f} нс\n")

def plot_results(
```

```
self,
current_values: List[int],
baseline_stats: StatisticsResult,
current_stats: StatisticsResult,
) -> Optional[Path]:
if plt is None:
log_warning("matplotlib не встановлено. Графік не буде побудовано.")
return None

path = self.output_dir / "timing_plot.png"

plt.figure(figsize=(12, 6))
plt.plot(current_values, linewidth=0.8, label="Поточні вимірювання")
plt.axhline(
baseline_stats.mean_ns,
linestyle="--",
linewidth=1,
label="Середнє контрольне значення",
)
plt.axhline(
current_stats.mean_ns,
linestyle=":",
linewidth=1,
label="Середнє поточне значення",
)
plt.title("Часові характеристики доступу до пам'яті")
plt.xlabel("Номер вимірювання")
plt.ylabel("Час доступу, нс")
plt.legend()
plt.grid(True, linewidth=0.3)
plt.tight_layout()
plt.savefig(path, dpi=150)
plt.close()

return path


class ConsoleInterface:
"""Консольний інтерфейс програми."""

@staticmethod
```

```
def print_header() -> None:
clear_screen()
print("=" * 78)
print(" АНАЛІЗ ПОТЕНЦІЙНИХ АТАК СПЕКУЛЯТИВНОГО
ВИКОНАННЯ КОДУ")
print(" Консольний інтерфейс | Версія 1.0.0")
print("=" * 78)

@staticmethod
def print_environment(config: ExperimentConfig) -> None:
print("\n[1] ПАРАМЕТРИ ЕКСПЕРИМЕНТУ")
print(f"Кількість запланованих вимірювань : {config.measurements}")
print(f"Кількість контрольних вимірювань : {config.baseline_runs}")
print(f"Інтервал між операціями : {config.interval_us} мкс")
print(f"Пороговий коефіцієнт : {config.threshold_factor}")
print(f"Розмір буфера пам'яті : {config.buffer_size_mb} МБ")

print("\n[2] СЕРЕДОВИЩЕ ВИКОНАННЯ")
print(f"Операційна система : {platform.system()} {platform.release()}")
print(f"Платформа : {platform.machine()}")
print(f"Python : {platform.python_version()}")
print(f"psutil : {'OK' if psutil is not None else 'не встановлено'}")
print(f"matplotlib : {'OK' if plt is not None else 'не встановлено'}")

@staticmethod
def print_snapshot(snapshot: SystemSnapshot) -> None:
print("\n[3] ПОТОЧНИЙ СТАН СИСТЕМИ")
print(f"CPU завантаження : {snapshot.cpu_percent:.2f}%")
print(
f"RAM використання : "
f"{snapshot.ram_used_gb:.2f} ГБ / "
f"{snapshot.ram_total_gb:.2f} ГБ "
f"({snapshot.ram_percent:.2f}%)"
)
print(f"Активних процесів : {snapshot.process_count}")

@staticmethod
def print_stats(title: str, stats: StatisticsResult) -> None:
print(f"\n{title}")
print(f"Кількість вимірювань : {stats.count}")
print(f"Середнє значення : {stats.mean_ns:.2f} нс")
```

```
print(f"Медіана : {stats.median_ns:.2f} нс")
print(f"Мінімум : {stats.min_ns} нс")
print(f"Максимум : {stats.max_ns} нс")
print(f"Стандартне відхилення : {stats.stdev_ns:.2f} нс")

@staticmethod
def print_analysis(analysis: AnalysisResult) -> None:
print("\n[6] РЕЗУЛЬТАТ АНАЛІЗУ")
print(f"Відхилення середнього : {analysis.mean_deviation_percent:.2f}%")
print(f"Відхилення std-dev : {analysis.stdev_deviation_percent:.2f}%")

if analysis.anomaly_detected:
print("\n[!] ПОТЕНЦІЙНА АНОМАЛІЯ ВИЯВЛЕНА")
else:
print("\n[OK] Суттєвих ознак аномалії не виявлено")

print(analysis.conclusion)


class SpeculativeExecutionMonitor:
"""Основний клас програмного забезпечення."""

def __init__(self, config: ExperimentConfig) -> None:
self.config = config
self.monitor = SystemMonitor()
self.meter = MemoryTimingMeter(config.buffer_size_mb)
self.analyzer = StatisticsAnalyzer()
self.detector = AnomalyDetector(config.threshold_factor)
self.reporter = ResultReporter(config.output_dir)
self.ui = ConsoleInterface()

def run(self) -> None:
self.ui.print_header()
self.ui.print_environment(self.config)

snapshot_start = self.monitor.get_snapshot()
self.ui.print_snapshot(snapshot_start)

log_service("Ініціалізація модулів завершена.")
log_info("Початок формування базових контрольних значень.")
```

```
baseline_values = self.meter.measure_series(
count=self.config.baseline_runs,
interval_us=self.config.interval_us,
show_progress=True,
)
baseline_stats = self.analyzer.calculate(baseline_values)
self.ui.print_stats("[4] КОНТРОЛЬНІ ЗНАЧЕННЯ", baseline_stats)

log_info("Перехід до основної серії вимірювань.")
current_values = self.meter.measure_series(
count=self.config.measurements,
interval_us=self.config.interval_us,
show_progress=True,
)

snapshot_end = self.monitor.get_snapshot()
current_stats = self.analyzer.calculate(current_values)

self.ui.print_snapshot(snapshot_end)
self.ui.print_stats("[5] ПОТОЧНІ СТАТИСТИЧНІ ПАРАМЕТРИ",
current_stats)

analysis = self.detector.analyze(
baseline=baseline_stats,
current=current_stats,
snapshot=snapshot_end,
)

self.ui.print_analysis(analysis)

log_info("Збереження результатів.")
baseline_csv = self.reporter.save_csv(
baseline_values,
"baseline_values.csv"
)
current_csv = self.reporter.save_csv(
current_values,
"current_values.csv"
)
report_file = self.reporter.save_report(
baseline=baseline_stats,
```

```
current=current_stats,
snapshot=snapshot_end,
analysis=analysis,
)
plot_file = self.reporter.plot_results(
current_values=current_values,
baseline_stats=baseline_stats,
current_stats=current_stats,
)

print("\n[7] ФАЙЛИ РЕЗУЛЬТАТІВ")
print(f"Контрольні вимірювання : {baseline_csv}")
print(f"Поточні вимірювання : {current_csv}")
print(f"Текстовий звіт : {report_file}")

if plot_file is not None:
print(f"Графік : {plot_file}")

log_info("Аналіз завершено.")


def parse_args() -> ExperimentConfig:
parser = argparse.ArgumentParser(
description="Аналіз непрямих ознак атак спекулятивного виконання коду."
)

parser.add_argument(
"--measurements",
type=int,
default=10_000,
help="Кількість вимірювань в основній серії.",
)
parser.add_argument(
"--baseline",
type=int,
default=2_000,
help="Кількість контрольних вимірювань.",
)
parser.add_argument(
"--interval-us",
type=int,
```

```
default=100,
help="Інтервал між вимірюваннями у мікросекундах.",
)
parser.add_argument(
"--threshold",
type=float,
default=1.35,
help="Пороговий коефіцієнт для аналізу.",
)
parser.add_argument(
"--buffer-mb",
type=int,
default=64,
help="Розмір буфера пам'яті у мегабайтах.",
)
parser.add_argument(
"--output",
type=str,
default="results",
help="Каталог для збереження результатів.",
)

args = parser.parse_args()

if args.measurements <= 0:
raise ValueError("Кількість вимірювань повинна бути більшою за нуль.")
if args.baseline <= 0:
raise ValueError("Кількість контрольних вимірювань повинна бути більшою
за нуль.")
if args.interval_us < 0:
raise ValueError("Інтервал не може бути від'ємним.")
if args.threshold <= 1.0:
raise ValueError("Пороговий коефіцієнт повинен бути більшим за 1.0.")
if args.buffer_mb <= 0:
raise ValueError("Розмір буфера повинен бути більшим за нуль.")

return ExperimentConfig(
measurements=args.measurements,
baseline_runs=args.baseline,
interval_us=args.interval_us,
threshold_factor=args.threshold,
```

```
buffer_size_mb=args.buffer_mb,
output_dir=Path(args.output),
)


def main() -> None:
try:
config = parse_args()
app = SpeculativeExecutionMonitor(config)
app.run()
except KeyboardInterrupt:
print("\nРоботу програми перервано користувачем.")
except Exception as exc:
print(f"\nПомилка виконання програми: {exc}", file=sys.stderr)
sys.exit(1)


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

ДОДАТОК Б

Слайди презентації

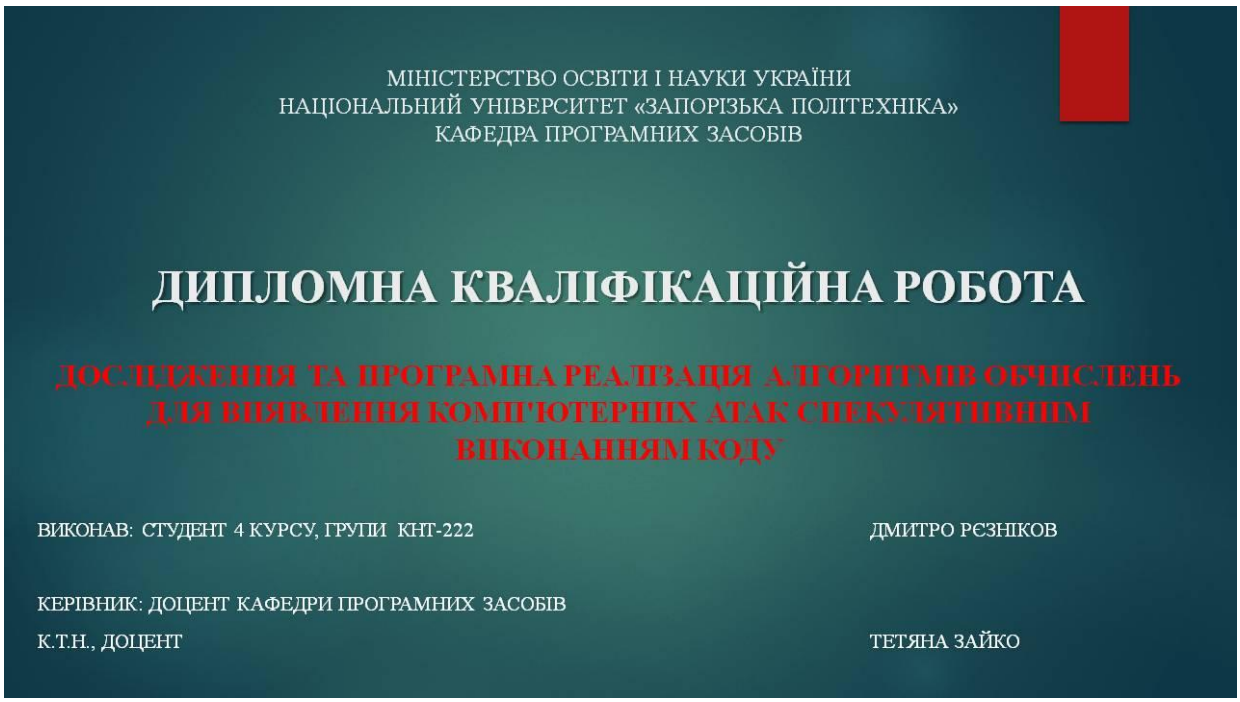

Рисунок Б.1 – Слайд 1

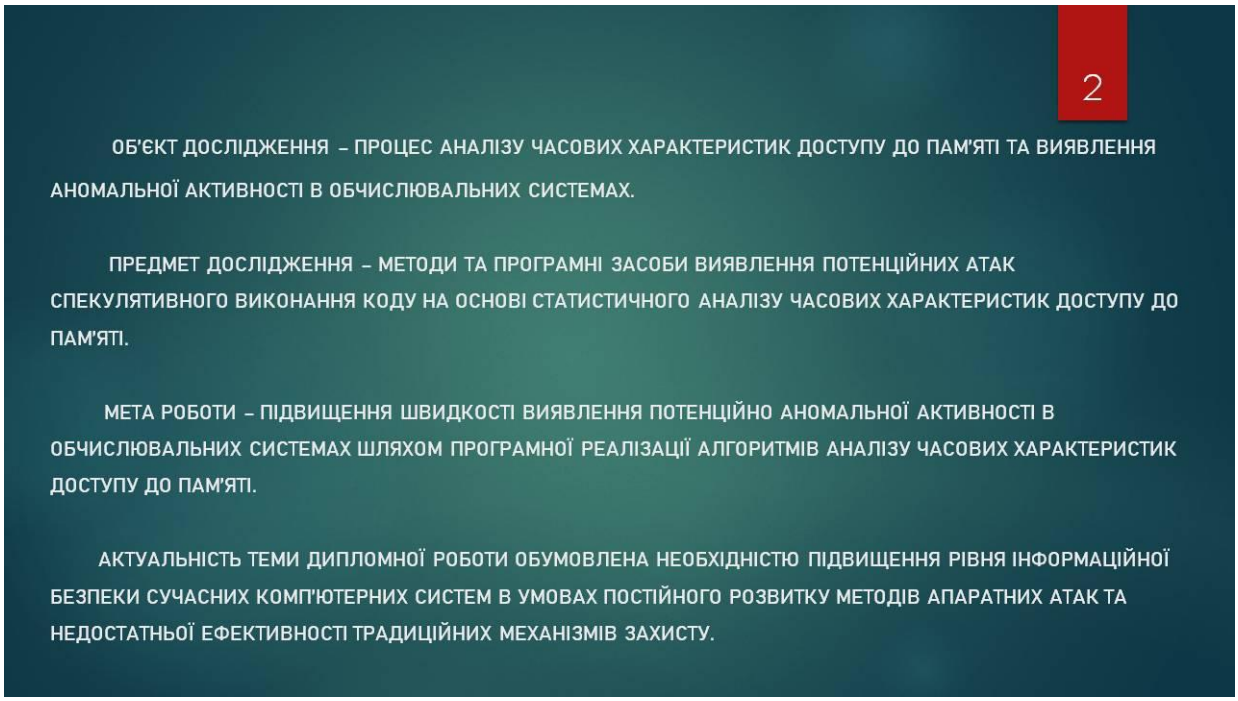

Рисунок Б.2 – Слайд 2

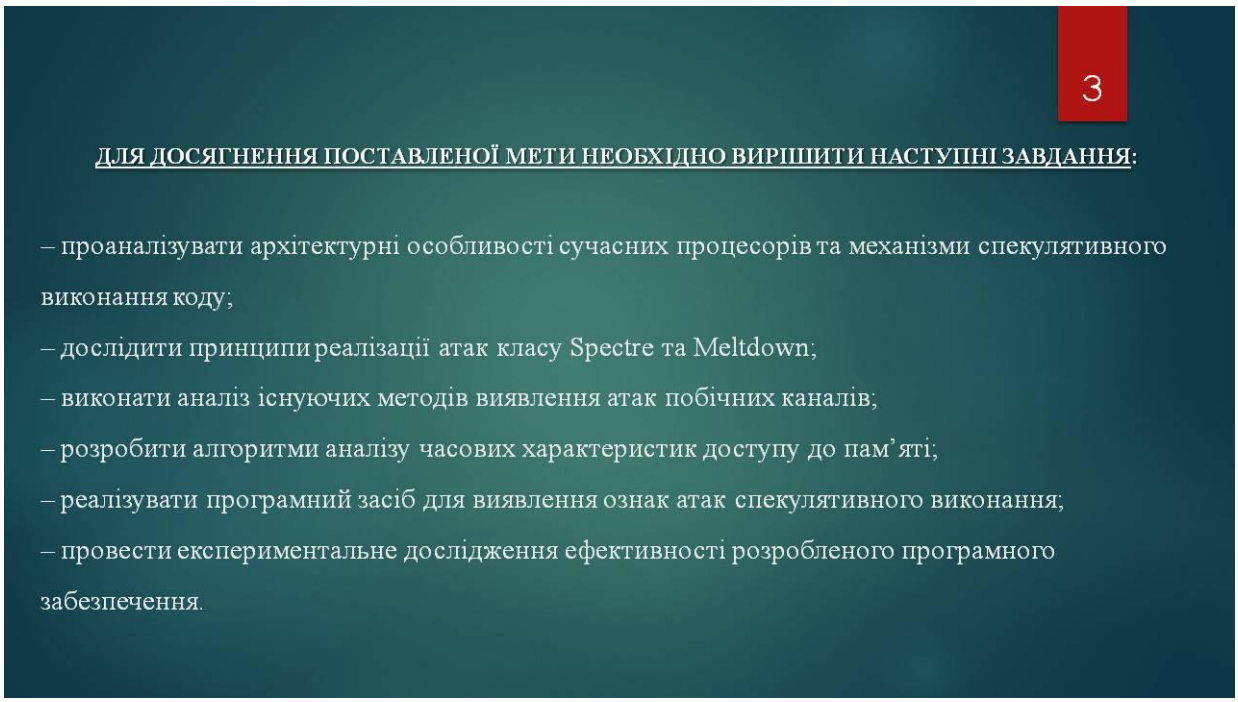

Рисунок Б.3 – Слайд 3

Рисунок Б.4 – Слайд 4

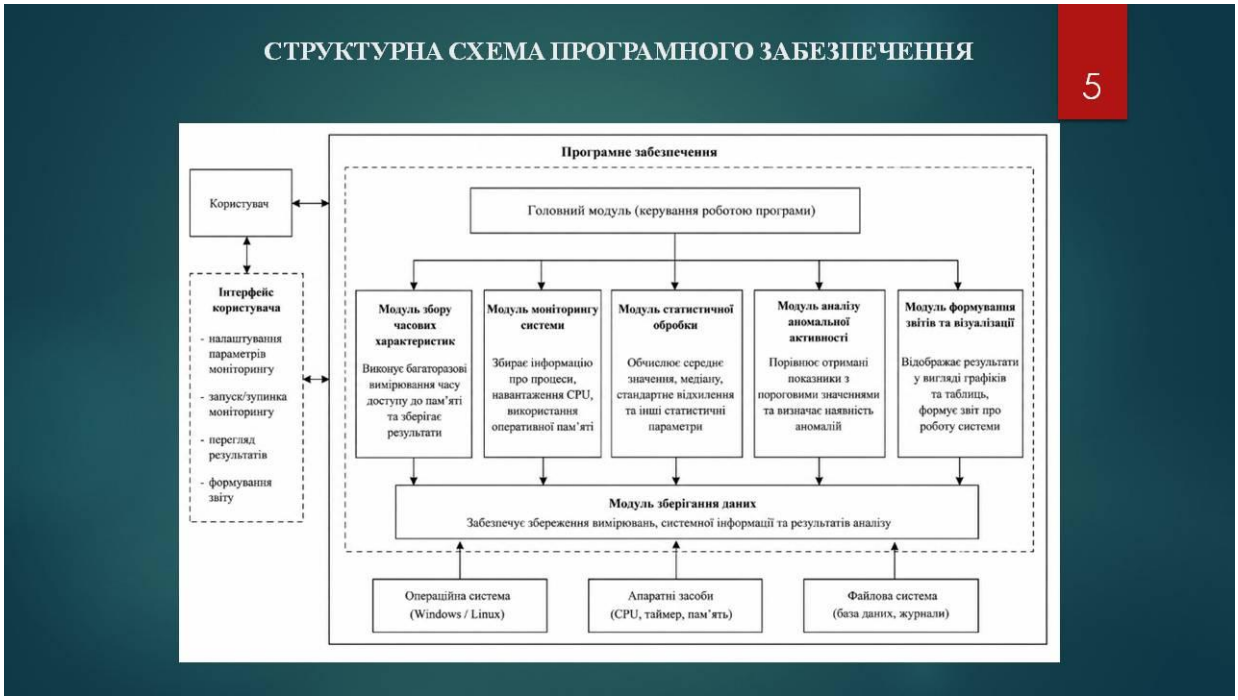

Рисунок Б.5 – Слайд 5

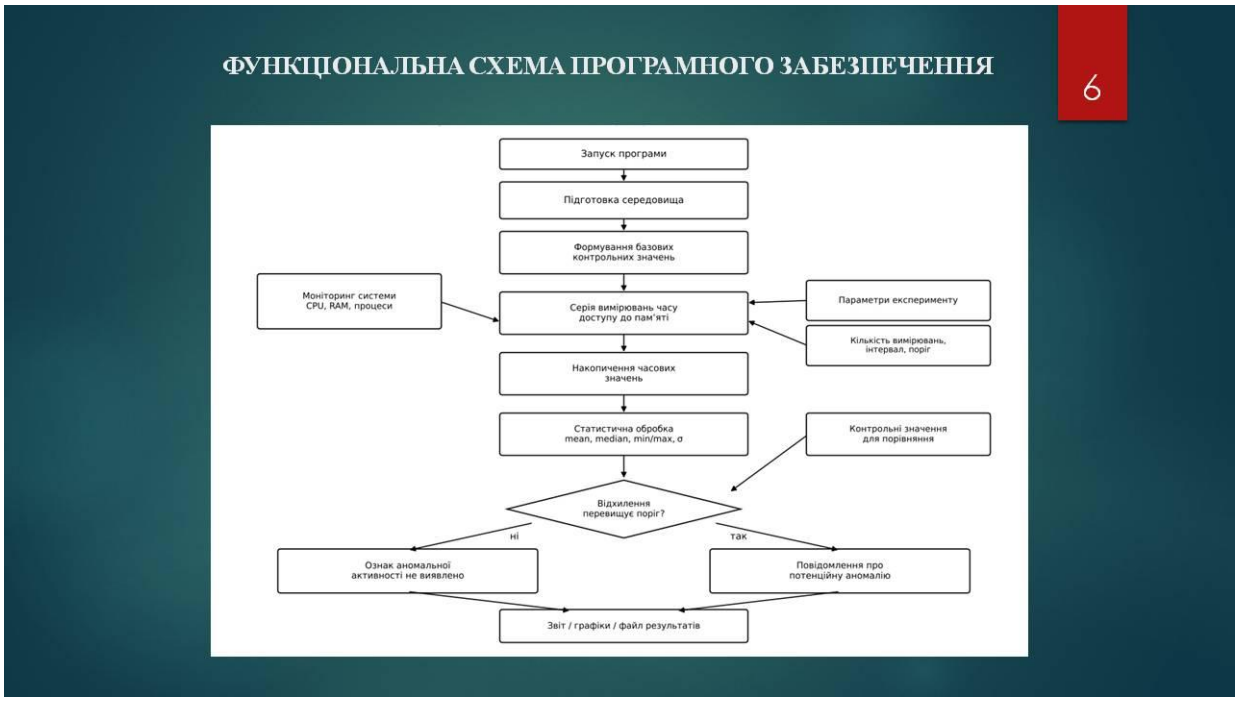

Рисунок Б.6 – Слайд 6

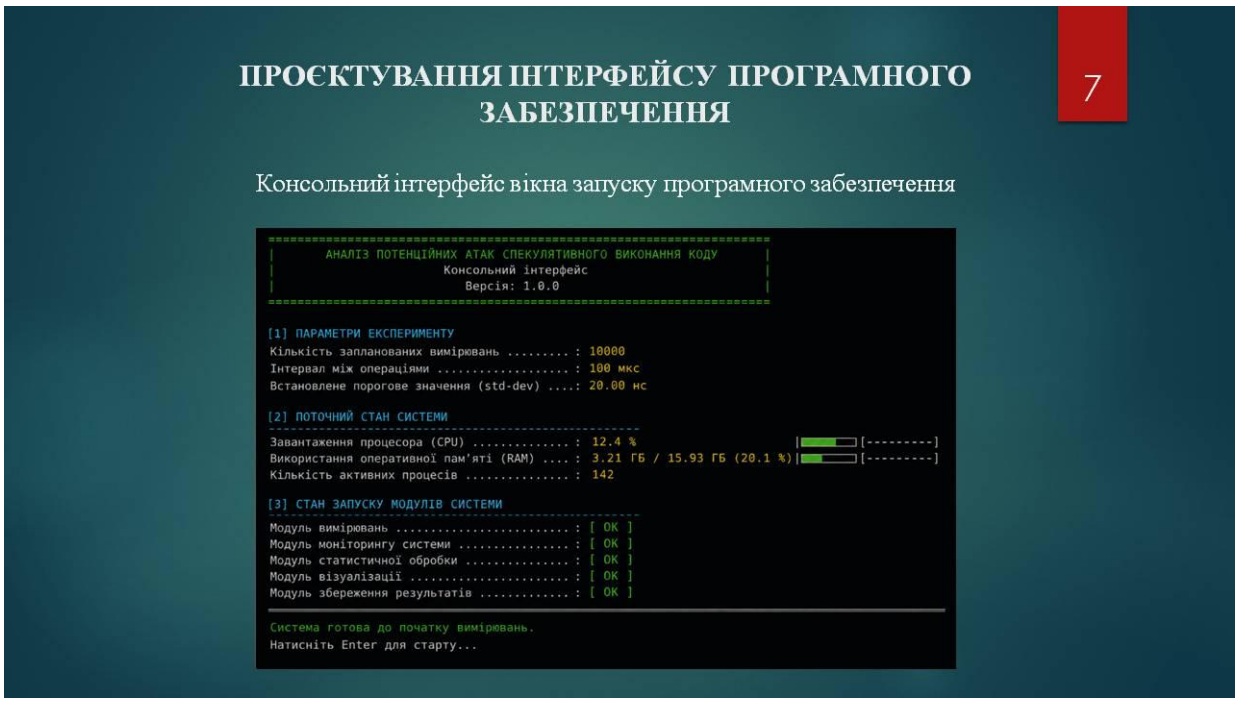

Рисунок Б.7 – Слайд 7

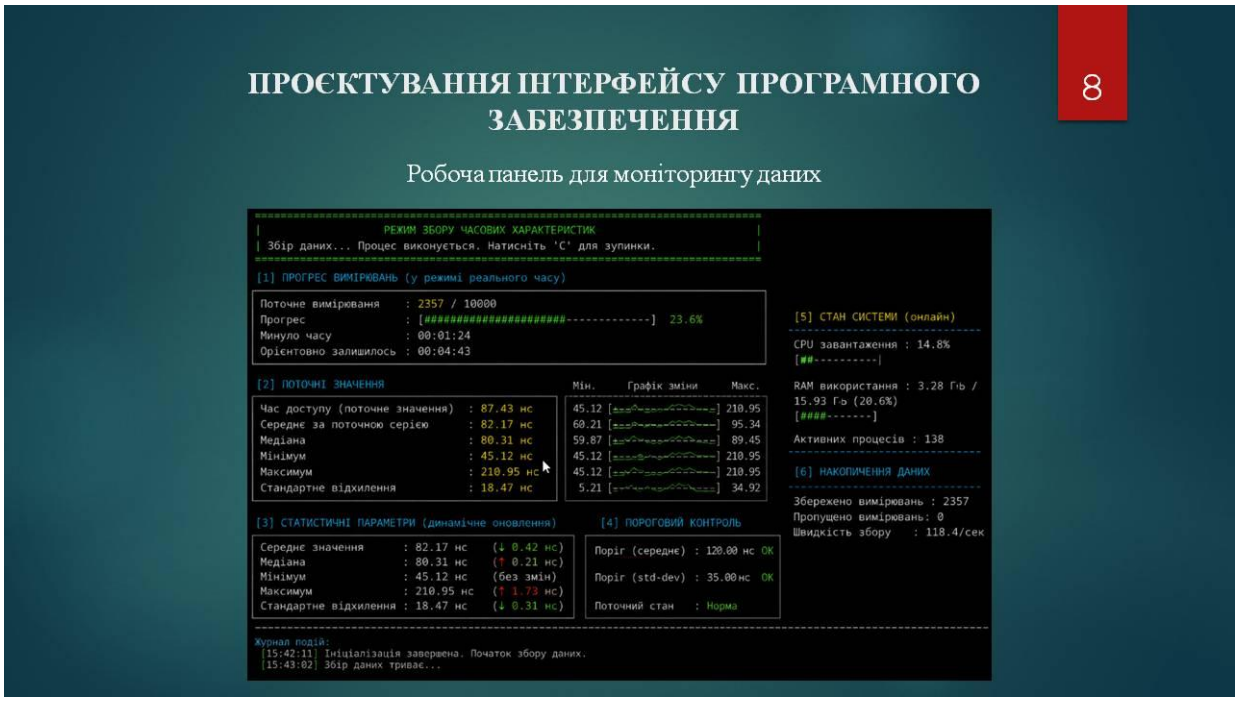

Рисунок Б.8 – Слайд 8

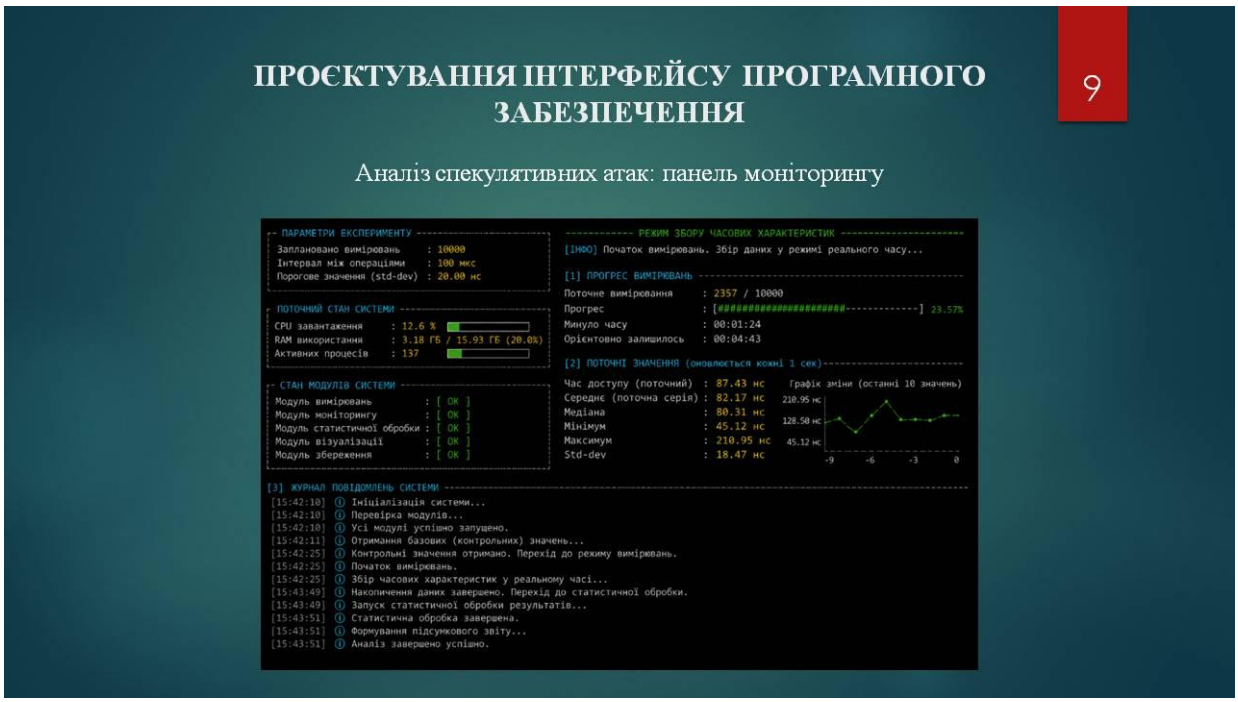

Рисунок Б.9 – Слайд 9

Рисунок Б.10 – Слайд 10

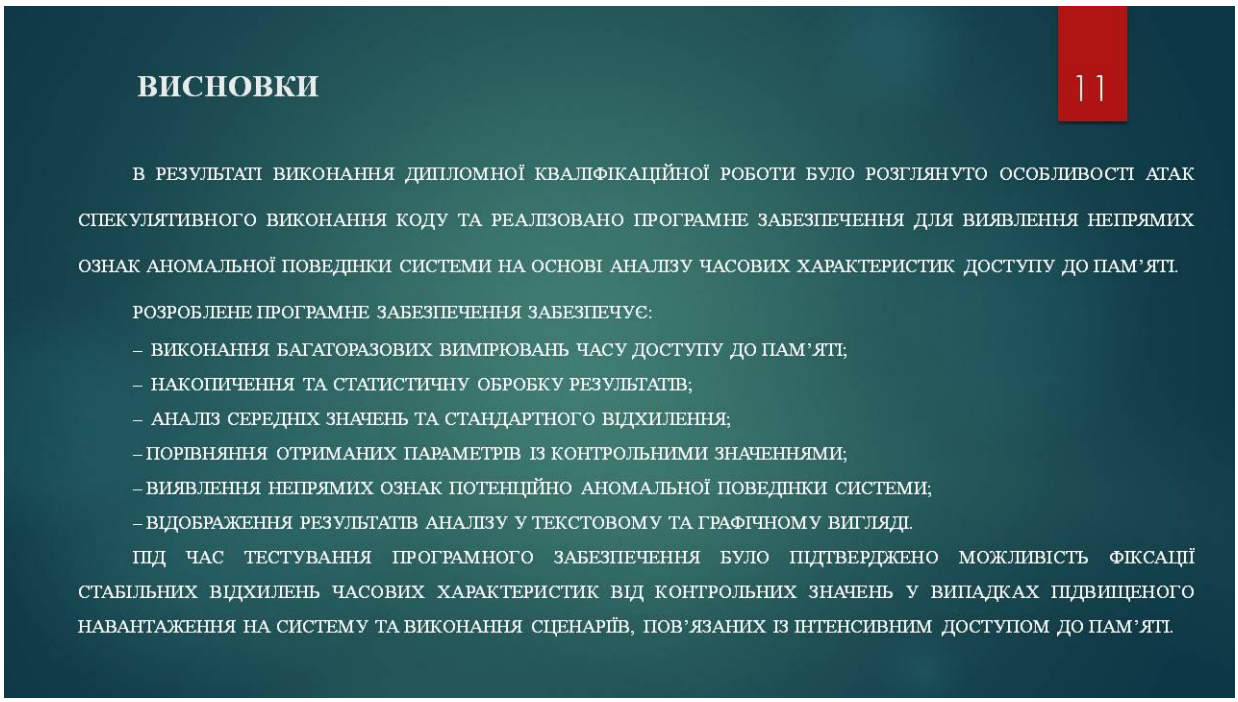

Рисунок Б.11 – Слайд 11