Хмельницький національний університет
Факультет інформаційних технологій
Кафедра комп'ютерної інженерії та інформаційних систем

КВАЛІФІКАЦІЙНА РОБОТА

Метод оптимізації продуктивності та захищеності кіберфізичних систем на основі балансування завдань і ресурсів
Назва теми

Рівень вищої освіти *другий (магістерський)*

Галузь знань *12 «Інформаційні технології»*
Шифр, назва

Спеціальність *123 «Комп'ютерна інженерія»*
Шифр, назва

Освітня програма *«Комп'ютерна інженерія та програмування»*
Назва

Шифр КвРКІ 240249.24.02.27 ПЗ

Виконав здобувач IV курсу, група КІ2м-22-1 _____ Олег НАГОРНЮК
Підпис Ініціали, прізвище

Керівник докт. філософії _____ Богдан САВЕНКО
Науковий ступінь, учене звання Підпис Ініціали, прізвище

Нормоконтролер канд.ф-м. наук, доцент _____ Тетяна КИСІЛЬ
Науковий ступінь, учене звання Підпис Ініціали, прізвище

До захисту допускаю:
завідувач кафедри КІІС _____ Ольга ПАВЛОВА
«_1_» _травня_ 2026 р. підпис Ініціали, прізвище
дата

Хмельницький 2026

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ ТА ІНФОРМАЦІЙНИХ СИСТЕМ

Рівень вищої освіти ДРУГИЙ (МАГІСТЕРСЬКИЙ)

Галузь знань 12 ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ

Спеціальність 123 КОМП'ЮТЕРНА ІНЖЕНЕРІЯ

Освітня програма «КОМП'ЮТЕРНА ІНЖЕНЕРІЯ ТА ПРОГРАМУВАННЯ»

ЗАТВЕРДЖУЮ
Завідувачка кафедри КІІС
_____ Ольга ПАВЛОВА

" 12 " 01 2026 р.

ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

Нагорнюку Олегу Валерійовичу
Прізвище, ім'я, по батькові студента

1. Тема проекту (роботи) Метод оптимізації продуктивності та захищеності кіберфізичних систем на основі балансування завдань і ресурсів

Керівник проекту (роботи) Богдан САВЕНКО, доктор філософії
Прізвище, ім'я, по батькові, науковий ступінь, вчене звання

Затверджена наказом ректора університету від 12.01.2026 р. № 6

2. Термін подання здобувачем роботи на кафедру 01.05.2026 р.

3. Вихідні дані до роботи Завдання на кваліфікаційну роботу

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити)

Аналіз відомих методів та засобів синтезу кіберфізична система на основі децентралізованого прийняття рішень

Архітектура кіберфізичних систем;

Моделі балансування завдань і ресурсів;

Метод оптимізації продуктивності та захищеності кіберфізичних систем.

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслень)

6. Консультанти розділів кваліфікаційної роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Тетяна КИСІЛЬ, доцент кафедри КІІС		
Антиплагіат	Андрій НІЧЕПОРУК, доцент кафедри КІІС		

7. Дата видачі завдання «_12_»_____01_____2026 р.

КАЛЕНДАРНИЙ ПЛАН

№з/п	Назва етапів (розділів) дипломного проєкту (роботи)	Термін виконання етапів проєкту (роботи)	Примітка
1	Вибір напряму дослідження та узгодження тематики кваліфікаційної роботи з керівником	12.01.2026	виконано
2	Робота над розділом 1 – аналіз відомих моделей, методів за темою; постановка задачі	25.02.2026	виконано
3	Робота над розділом 2 – розробка моделей для вирішення поставленої задачі	15.03.2026	виконано
4	Робота над науковою статтею	01.04.2026	виконано
5	Робота над розділом 3 – розробка методів для вирішення поставленої задачі	10.04.2026	виконано
6	Робота над розділом 4 – проектування та розробка ПЗ для вирішення поставленої задачі, експериментальна частина	25.04.2026	виконано
7	Оформлення пояснювальної записки згідно вимог	25.04.2026	виконано
8	Попередній захист ВКР	30.04.2025	виконано
9	Захист ВКР на засіданні ЕК	травень 2026 року	

Здобувач _____ ___Олег НАГОРНЮК___
Підпис Ім'я, ПРІЗВИЩЕ

Керівник кваліфікаційної роботи_____ ____Богдан САВЕНКО ___
Підпис Ім'я, ПРІЗВИЩЕ

РЕФЕРАТ

Тема кваліфікаційної роботи магістра: «Метод оптимізації продуктивності та захищеності кіберфізичних систем на основі балансування завдань і ресурсів»

Автор роботи: Нагорнюк Олег Валерійович

Керівник роботи: Савенко Богдан Олегович

Пояснювальна записка: 74 с., 9 рис., 10 табл., 4 дод., 81 джерело.

БАЛАНСУВАННЯ РЕСУРСІВ, ВКАЗІВНИКИ, КЕРУЮЧІ ПОТОКИ, КІБЕРФІЗИЧНА СИСТЕМА, ОПЕРАЦІЙНІ СИСТЕМИ РЕАЛЬНОГО ЧАСУ, ОПТИМІЗАЦІЯ, ПОТОКИ ВИКОНАННЯ.

Об’єктом дослідження є процеси забезпечення продуктивності та захищеності кіберфізичних систем реального часу.

Предметом дослідження є методи оптимізації планування завдань і розподілу ресурсів у кіберфізичних системах з урахуванням консервативних оцінок WCET, slack time та адаптивних механізмів безпеки.

Метою кваліфікаційної роботи магістра є підвищення рівня захищеності та ефективності використання ресурсів кіберфізичних систем без порушення часових обмежень шляхом розробки методу балансування завдань і ресурсів на основі адаптивного управління механізмами безпеки.

Для розв’язання поставлених задач використовувалися методи аналізу та планування систем реального часу, методи синтезу й моделювання кіберфізичних систем, підходи до оцінювання WCET і slack time, а також методи оптимізації розподілу обчислювальних ресурсів і механізмів безпеки.

Наукова новизна отриманого результату:

– удосконалено метод оптимізації продуктивності та захищеності кіберфізичних систем на основі балансування завдань і ресурсів, який, на відміну від традиційних статичних підходів із жорстким резервуванням або постійними перевірками, використовує консервативні оцінки WCET і доступний slack time для адаптивного перенесення витрат механізмів безпеки в часові резерви, що забезпечує гарантовану своєчасність виконання завдань при підвищеному рівні захисту та ефективному використанні ресурсів..

На основі проведених досліджень розроблено метод та архітектуру для оптимізації механізмів захисту в кіберфізичних системах реального часу, що дозволяють інтегрувати надійний захист без значного впливу на реальний час виконання завдань.

Практична значимість отриманих результатів полягає у здійсненні оптимізації продуктивності та захищеності кіберфізичних систем на основі балансування завдань і ресурсів.

У вступі подано об’єкт та предмет дослідження, мету, наукову новизну та практичну цінність роботи, а також характеристику структури роботи.

У першому розділі проведено аналіз відомих рішень щодо оптимізації продуктивності та захищеності кіберфізичних систем, цілісність потоків даних, асинхронного керуючого потоку та вказівника, що зберігає планування.

У другому розділі здійснено розроблення моделей механізмів цілісності потоків даних, керуючого потоку, вказівників і модель оптимізації продуктивності та захищеності КФС, архітектура КФС на основі оптимізації продуктивності та захищеності.

У третьому розділі розроблено стратегії Стратегії зловмисників в атаках на КФС, метод оптимізації продуктивності та захищеності кіберфізичних систем на основі балансування завдань і ресурсів.

У четвертому розділі здійснено розроблення КФС реального часу, а також проведено експеримент з розробленою реалізацією КФС РЧ.

У висновках підведено підсумки досягнення результатів з розв’язання завдань дослідження.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

ІТ – інформаційні технології

КФС – кіберфізична система

ОС – операційна система

ПЗ – програмне забезпечення

ЦКП / CFI – Цілісність керуючого потоку / Control–Flow Integrity

ЦПД / DFI – Цілісність потоку даних / Data–Flow Integrity

ВСТУП

Кіберфізичні системи реального часу (КФС РЧ) відіграють дедалі важливішу роль у сучасному світі, функціонуючи в середовищах із жорсткими часовими обмеженнями та охоплюючи спектр від простих вбудованих пристроїв, як–от мікроконтролери, до складних платформ, таких як дрони та автономні транспортні засоби. Ці системи переважно програмуються мовами без механізмів захисту пам'яті, наприклад C/C++, що спричиняє численні вразливості безпеки та пов'язані проблеми, а в критичних для безпеки застосуваннях забезпечення надійного захисту є ключовим для гарантії безпеки та дотримання часових вимог. Існуючі механізми безпеки, розроблені для універсальних обчислювальних систем, генерують значні додаткові витрати, що ускладнює їх інтеграцію в КФС РЧ, тоді як модель обчислень у системах реального часу дозволяє впроваджувати захисні механізми з мінімальними витратами ресурсів, тому доцільно розробляти спеціально адаптовані механізми безпеки для РЧ, мінімізуючи їхній вплив на продуктивність. Ця робота розширює три ключові механізми захисту — цілісність потоків даних, керуючого потоку та вказівників, — з адаптаціями для РЧ, що включають перевірку цілісності потоків даних у вільний час під час ітерацій завдань для забезпечення міцного захисту із мінімальними витратами в найгіршому випадку, асинхронну перевірку цілісності керуючого потоку у вбудованих системах через вікна планування для зменшення впливу на реальний час, а також використання доступного системного часу для перевірки цілісності вказівників, що підвищує загальну безпеку без порушення розкладу завдань, таким чином пропонуючи підходи для оптимізації балансу між рівнем безпеки та продуктивністю в системах реального часу.

Актуальність роботи полягає в розробці адаптованих механізмів захисту для КФС РЧ, які забезпечують надійну безпеку з мінімальними додатковими витратами на продуктивність, враховуючи жорсткі часові обмеження та вразливості, пов'язані з мовами програмування без захисту пам'яті, такими як C/C++.

Метою кваліфікаційної роботи є підвищення рівня безпеки кіберфізичних систем реального часу за рахунок розширення та адаптації механізмів захисту цілісності потоків даних, керуючого потоку та вказівників, що мінімізує вплив на продуктивність та забезпечує дотримання часових вимог.

Поставлена мета досягається розв’язанням таких основних завдань:

1) проаналізувати існуючі підходи до забезпечення безпеки кіберфізичних систем реального часу та визначити їхні обмеження щодо жорсткого резервування ресурсів і додаткових часових витрат;

2) розробити метод оптимізації продуктивності та захищеності кіберфізичних систем на основі балансування завдань і ресурсів;

3) розробити адаптивні механізми захисту цілісності потоків даних, потоку керування та покажчиків із урахуванням планування завдань і динамічного розподілу ресурсів;

4) розробити комплексну модель та архітектуру кіберфізичної системи з інтеграцією просторової й часової ізоляції та динамічного управління рівнями безпеки;

5) провести експериментальну оцінку ефективності запропонованого методу з точки зору своєчасності виконання завдань, рівня захисту та додаткових витрат безпеки.Об’єктом дослідження є планування процесів реального часу в кіберфізичних системах.

Об’єктом дослідження є процеси забезпечення продуктивності та захищеності кіберфізичних систем реального часу.

Предметом дослідження є методи оптимізації планування завдань і розподілу ресурсів у кіберфізичних системах з урахуванням консервативних оцінок WCET, slack time та адаптивних механізмів безпеки.

Наукова новизна отриманого результату:

– удосконалено метод оптимізації продуктивності та захищеності кіберфізичних систем на основі балансування завдань і ресурсів, який, на відміну від традиційних статичних підходів із жорстким резервуванням або постійними перевірками, використовує консервативні оцінки WCET і доступний slack time для

адаптивного перенесення витрат механізмів безпеки в часові резерви, що забезпечує гарантовану своєчасність виконання завдань при підвищеному рівні захисту та ефективному використанні ресурсів..

На основі проведених досліджень розроблено метод та архітектуру для оптимізації механізмів захисту в кіберфізичних системах реального часу, що дозволяють інтегрувати надійний захист без значного впливу на реальний час виконання завдань.

Практична значимість отриманих результатів полягає у розробці концептуальних рішень для підвищення безпеки критичних систем за рахунок інтеграції цих адаптованих механізмів у вбудовані системи реального часу, яка оптимізує ресурсне навантаження без порушення розкладу завдань, що є новим підходом для критичних застосувань, таких як дрони та автономні транспортні засоби.

Для розв’язання поставлених задач використовувалися методи аналізу та планування систем реального часу, методи синтезу й моделювання кіберфізичних систем, підходи до оцінювання WCET і slack time, а також методи оптимізації розподілу обчислювальних ресурсів і механізмів безпеки.

За темою кваліфікаційної роботи опубліковано одну публікацію [81] у фаховому науковому журналі «Herald of Khmelnytskyi National University. Technical Sciences» (м. Хмельницький, 2026, № 361(1), С. 575–585. DOI: https://doi.org/10.31891/2307–5732–2026–361–78).

1 АНАЛІЗ ВІДОМИХ МЕТОДІВ ТА ЗАСОБІВ ОПТИМІЗАЦІЇ ПРОДУКТИВНОСТІ ТА ЗАХИЩЕНОСТІ КІБЕРФІЗИЧНИХ СИСТЕМ НА ОСНОВІ БАЛАНСУВАННЯ ЗАВДАНЬ І РЕСУРСІВ

1.1 Поняття оптимізації продуктивності та захищеності кіберфізичних систем

Кіберфізичні системи реального часу (КФС РЧ), як ключовий елемент сучасної технологічної інфраструктури, відіграють провідну роль у інтеграції цифрових і фізичних процесів, забезпечуючи взаємодію між обчислювальними компонентами та реальним середовищем. У рамках досліджень, присвячених розвитку інтелектуальних систем, КФС РЧ характеризуються як гібридні структури, де програмне забезпечення генерує команди для фізичних дій, а фізичні параметри, у свою чергу, впливають на обчислювальні процеси, формуючи замкнутий цикл зворотного зв'язку. Цей підхід знаходить широке застосування в різноманітних галузях, від простих вбудованих мікроконтролерів у побутовій техніці до складних систем, таких як медичні імплантати, автономні транспортні засоби, промислові роботи та безпілотні літальні апарати. Дослідження в цій сфері підкреслюють, що на відміну від традиційних обчислювальних платформ, де пріоритетом є максимальна продуктивність, КФС РЧ вимагають суворого дотримання часових обмежень, оскільки будь–які затримки можуть спричинити критичні збої з потенційно катастрофічними наслідками. Для моделювання таких систем наукова спільнота застосовує парадигми реального часу, де навантаження представлено як набір періодичних завдань, кожне з яких асоційоване з жорстким дедлайном [1, 2]. У контексті жорстких дедлайнів, типових для критичних застосувань, забезпечення своєчасності виконання є фундаментальною вимогою, що гарантує не лише функціональну надійність, але й загальну безпеку системи, як це детально аналізується в теоріях планування завдань [3, 4].

У науково–дослідницькому контексті забезпечення безпеки КФС РЧ набуває особливого значення, оскільки ці системи часто інтегруються в критичні інфраструктури [5, 6], де вразливості можуть бути використані для зловмисних

атак, загрожуючи людському життю та економічній стабільності. Дослідження підкреслюють, що обмеження за розміром, вагою та енергоспоживанням апаратних платформ зумовлюють використання ресурсоощадних архітектур, які, однак, обмежують обчислювальні потужності. З огляду на вимоги до ефективності, розробка КФС РЧ переважно базується на мовах програмування C та C++, які забезпечують низькорівневий контроль над ресурсами, але водночас позбавлені вбудованих механізмів захисту пам'яті. Це призводить до появи численних вразливостей, пов'язаних із буферними переповненнями, витоками пам'яті та іншими помилками, які зловмисники можуть експлуатувати для несанкціонованого доступу [7, 8]. Процес експлуатації таких атак, як правило, включає етапи виявлення помилок у коді, створення шкідливого навантаження та модифікацію даних у пам'яті процесу, що порушує цілісність виконання програми. Атаки на пошкодження пам'яті можуть призводити до непередбачуваної поведінки системи, компрометуючи як кібер–, так і фізичні компоненти, наприклад, через фальсифікацію сенсорних даних у робототехнічних системах [9, 10]. Таким чином, дослідження фокусуються на розробці стратегій протидії, які враховують специфіку реального часу, аби мінімізувати ризики без значного зниження продуктивності.

Сучасні механізми захисту безпеки в КФС РЧ, розроблені в рамках міждисциплінарних досліджень, базуються на ідеї моніторингу аномальної поведінки, оскільки експлуатація вразливостей генерує відхилення від нормального функціонування програми. Запропонувано низку примітивів безпеки під час виконання, спрямованих на протидію атакам з пошкодженням пам'яті [11, 12]. Ці примітиви адаптовані для виявлення конкретних типів загроз. Цілісність потоку даних запобігає несанкціонованій модифікації даних. Цілісність керуючого потоку захищає від перехоплення контролю виконання. Цілісність вказівників протидіє атакам на маніпуляцію адресами пам'яті. У дослідницькому фокусі ці механізми реалізуються шляхом інтеграції додаткових інструкцій перевірки в бінарний код програми на етапі компіляції, що дозволяє динамічно оцінювати відповідність поведінки системи заздалегідь визначеній політиці безпеки [13, 14].

Під час виконання програми ці інструкції моніторять ключові операції, виявляючи відхилення, які сигналізують про потенційну атаку. Емпіричні оцінки показують, що впровадження таких примітивів значно посилює стійкість КФС РЧ до кіберзагроз, особливо в критичних застосуваннях, де традиційні захисні заходи з універсальних систем виявляються неефективними через високі додаткові витрати [15, 16]. Таким чином, зусилля дослідників спрямовані на оптимізацію цих механізмів для реального часу, забезпечуючи баланс між безпекою, ресурсоефективністю та дотриманням дедлайнів, що відкриває перспективи для подальших інновацій у сфері захищених кіберфізичних технологій [17, 18].

Однак, у контексті науково–дослідницьких підходів до забезпечення безпеки в КФС РЧ, традиційні механізми захисту часто супроводжуються суттєвим погіршенням часових характеристик і загальної продуктивності. Це обумовлено фундаментальними особливостями реалізації примітивів безпеки під час виконання, які базуються на програмних інструментах, що інтегрують додаткові інструкції перевірки безпосередньо в бінарний код програми на стадії компіляції або інструментації [19, 20]. Такі інструкції, хоча й не змінюють семантику оригінального коду, неминуче вводять додаткові затримки у процесі виконання, оскільки вони активуються синхронно на кожному критичному шляху програми, незалежно від контексту або наявності загрози [21, 22]. Емпіричні дослідження демонструють, що в універсальних обчислювальних середовищах ці витрати можуть бути прийнятними, але в домені реального часу, де жорсткі дедлайни є критичними для збереження системної стабільності, вони часто призводять до небажаних порушень [23].

У контексті науково–дослідницьких підходів до проектування та оптимізації КФС РЧ, поняття WCET slack відіграє ключову роль у забезпеченні ефективного використання ресурсів, особливо в умовах жорстких часових обмежень. Термін "WCET slack" [24, 25] є спеціальним поняттям з теорії систем реального часу, де WCET розшифровується як найгірший час виконання або час виконання в найгіршому випадку. "Slack" у цьому контексті позначає "запас часу" або "резерв часу", що виникає через різницю між консервативно оціненим WCET і фактичним

часом виконання завдання. WCET представляє собою консервативну оцінку максимального можливого часу виконання завдання або програми в системі реального часу, враховуючи всі можливі сценарії, включаючи найгірші умови апаратного забезпечення, кеш–промахи, переривання та інтерференцію від інших задач. Ця оцінка [26] є фундаментальною для планування завдань, оскільки системи реального часу, на відміну від загальнопризначених обчислювальних платформ, вимагають гарантованого дотримання дедлайнів для уникнення катастрофічних збоїв. WCET обчислюється за допомогою статичного аналізу, який моделює поведінку програми на рівні машинного коду, або гібридних методів, що поєднують статичний аналіз з емпіричними вимірюваннями на реальному апаратному забезпеченні. Консервативність WCET забезпечує безпеку, але часто призводить до переоцінки реального часу виконання, що створює "зайвий" резерв ресурсів [27].

Статичні методи аналізують код програми без її виконання, використовуючи математичні моделі для оцінки верхньої межі часу [28]. Вони базуються на абстрактній інтерпретації та моделюванні апаратної платформи (наприклад, процесора, кешу, шини). Перевага полягає в забезпеченні повної безпеки, оскільки вони враховують всі можливі шляхи виконання, але вони можуть бути консервативними [28].

Slack time (або slack) це запас часу, що виникає в розкладі завдань через різницю між зарезервованим WCET і фактичним часом виконання завдання. Він визначається як кількість часу, яку можна "відкласти" або використати для додаткових обчислень без порушення дедлайнів системи. Цей концепт є ключовим у теорії планування реального часу, зокрема в моделях з фіксованими пріоритетами або динамічними пріоритетами, де slack дозволяє оптимізувати використання процесора, зменшувати енергоспоживання або впроваджувати додаткові функції, такі як моніторинг безпеки [29]. Slack time поділяється на два основні типи. Статичний slack обчислюється на етапі офлайн–аналізу перед запуском системи на основі WCET і відомого розкладу завдань. Він представляє фіксований резерв, що виникає через консервативні оцінки WCET, і може бути використаний для

статичного розподілу ресурсів, наприклад, для зменшення конкуренції за ресурси в паралельних системах [30]. Динамічний slack виникає під час виконання через варіації в часі виконання завдань, які часто менші за WCET. Наприклад, через кращі умови кешу або відсутність переривань. Цей slack "відновлюється" динамічно і може бути використаний для адаптивних стратегій, таких як динамічне масштабування напруги/швидкості для енергозбереження або для моніторингу перевірок безпеки. У мультипроцесорних системах динамічний slack може "ділитися" між процесорами, що підвищує ефективність [31]. У контексті КФС РЧ, slack time часто асоціюється з "системним slack", який відображає загальний невикористаний час у розкладі, включаючи асинхронні вікна планування. Це дозволяє переносити некритичні обчислення, такі як перевірки цілісності даних, у вільні інтервали без впливу на критичні шляхи.

Незважаючи на переваги, обчислення WCET slack стикається з викликами [32]. Неточні оцінки WCET можуть призвести до негативного slack, тобто перевищення дедлайнів, а в мультиядерних системах інтерференція ускладнює моделювання. У контексті дослідження WCET slack може бути доцільним ресурсом для адаптації примітивів безпеки, забезпечуючи баланс між продуктивністю та стійкістю до атак [33]. Дослідники та розробники КФС РЧ стикаються з фундаментальною дилемою, як полягає в тому, що необхідне балансування між впровадженням потужних механізмів захисту, які забезпечують високий рівень стійкості до атак на пошкодження пам'яті, та збереженням необхідної продуктивності для дотримання реального часу [34, 35]. Ця суперечність стає бар'єром для практичного розгортання сучасних рішень, стимулюючи пошуки гібридних підходів, що інтегрують апаратні та програмні оптимізації [36, 37]. Тому, робота спрямована на систематичне дослідження можливостей забезпечення високого рівня безпеки з мінімальним впливом на продуктивність у реальному часі.

Таким чином, унікальні характеристики моделей обчислень систем реального часу, такі як консервативне резервування WCET та наявність slack time, можуть бути ефективно використані для суттєвого зменшення додаткових витрат

примітивів безпеки під час виконання, забезпечуючи оптимальний баланс між стійкістю до атак і дотриманням жорстких дедлайнів. Цього можна досягти застосувавши три інноваційні адаптації примітивів безпеки, кожна з яких експлуатує специфічну особливість реального часу з метою мінімізації впливу на WCET. Ці адаптації охоплюють цілісність потоків даних, цілісність керуючого потоку, цілісність вказівників. Вони базуються на теорії планування завдань, включаючи моделі з фіксованими пріоритетами та динамічним розподілом ресурсів, що дозволяє переносити обчислювальні навантаження перевірок у вільні інтервали.

1.2 Поняття про цілісність потоків даних, асинхронного керуючого потоку та вказівника, що зберігає планування

Цілісність потоків даних використовує резерв часу в найгіршому випадку під час ітерацій періодичних завдань для асинхронної перевірки потоків даних, що дозволяє відкладати некритичні перевірки без порушення дедлайнів [38].

Цілісність керуючого потоку інтегрує асинхронні вікна планування для динамічного моніторингу графа керуючого потоку, зменшуючи синхронні витрати в критичних шляхах [39].

Цілісність вказівників застосовує системний slack time у розкладі реального часу для періодичної валідації вказівників, оптимізуючи ресурсне навантаження в мультизадачних середовищах [40].

Ось перефразований варіант, збільшений приблизно в півтора раза та поданий суцільним текстом без поділу на підрозділи.

Цілісність потоків даних є важливим механізмом безпеки, спрямованим на запобігання пошкодженню вмісту пам’яті шляхом контролю правильності обробки та передавання даних у межах програмного виконання. Вона дозволяє виявляти несанкціоновані зміни даних під час роботи програми та тим самим забезпечує високий рівень захисту. Водночас застосування таких механізмів часто супроводжується суттєвими додатковими витратами, особливо щодо найгіршого

часу виконання [41]. Для пом’якшення цього негативного ефекту пропонується в роботі [42] використання додаткового часу, уже закладеного в резервaціях WCET, з метою компенсації додаткових витрат безпеки. Такий підхід дозволяє перенести основний вплив механізмів безпеки на середній час виконання, водночас мінімізуючи їхній вплив на гарантований найгірший час виконання, що є критично важливим для систем реального часу. Попередні дослідження [43, 44] вже розглядали можливість використання характеристик реального часу для підвищення безпеки, однак вони переважно зосереджувалися на захисті статичних властивостей програм, наприклад цілісності файлів, і не враховували динамічну поведінку під час виконання. На відміну від таких підходів, цілісність потоків даних передбачає перевірку виконання програми в реальному часі, що створює принципово іншу ситуацію. Додатковий час у резервaціях WCET має бути визначений заздалегідь, тобто ще до запуску програми, а не оцінюватися після завершення виконання [44]. Це є принципово важливим, оскільки механізми цілісності потоків даних працюють як вбудовані перевірки безпеки, і постфактум–оцінка часу виконання не дозволяє ефективно використовувати властивості планування в реальному часі [45]. Запропонований у роботі [46] підхід демонструє, що завдяки раціональному використанню резервів WCET можна зменшити додаткові витрати на WCET, не втрачаючи при цьому високого рівня покриття захисту, що підтверджує можливість інтеграції потужних механізмів безпеки під час виконання з мінімальним впливом на часові гарантії системи. Окрім цього, значну увагу приділено цілісності керуючого потоку виконання, яка забезпечує коректність переходів між інструкціями програми та унеможливлює виконання шкідливого коду [47].

Традиційні підходи до забезпечення цілісності керуючого потоку зазвичай базуються на інструментуванні програмного коду, що потребує змін у компіляторі або бінарному файлі [48]. Такий підхід є проблематичним для вбудованих систем, оскільки їхні компіляторні інструментарії часто жорстко налаштовані або обмежені. Для обходу цих обмежень у роботах [48, 49] було запропоновано використовувати апаратні механізми трасування, які автоматично фіксують події

безпеки та дозволяють виконувати перевірки асинхронно, без втручання в програмний код. Це забезпечує як зручність, завдяки збереженню бінарної сумісності, так і підвищену продуктивність за рахунок перенесення частини навантаження на апаратне забезпечення. Проте такі рішення можуть призводити до неконтрольованих додаткових витрат у реальному часі, оскільки момент виконання перевірок безпеки не узгоджується з плануванням задач, що унеможливлює надання строгих часових гарантій. Просте додавання керувального коду до бінарного файлу суперечило б ідеї незмінності бінарів, тому запропоновано апаратний підхід, заснований на використанні можливостей апаратного налагодження платформи [50]. Цей підхід дозволяє безшовно втручатися у виконання програм, використовуючи асинхронні вікна планування, оптимізуючи часові характеристики системи та водночас зберігаючи початкову мету. Крім того, у роботі розглядається цілісність покажчиків як засіб протидії атакам, спрямованим на підміну або пошкодження значень покажчиків. Хоча базові механізми цілісності покажчиків можуть ефективно захищати від простих атак, але вони виявляються недостатніми проти складніших сценаріїв, таких як атаки повторного використання покажчиків, за яких зловмисник маніпулює, на перший погляд, коректними цілями [51]. Для підвищення рівня захисту деякі існуючі підходи пропонують враховувати контекст виконання, проте вони зазвичай змушують обирати між легким захистом із малими додатковими витратами та сильним захистом із суттєвим впливом на час виконання, не забезпечуючи при цьому гарантій у реальному часі [52]. У роботі [53] пропонується альтернативний підхід, що базується на використанні загальносистемного резервного часу та залученні лише частини контексту виконання, достатньої для підвищення безпеки без порушення планування. Ключовим завданням при цьому є правильне визначення релевантного контексту та збереження узгодженості безпеки за умов часткового захисту. Робота формалізує взаємозв’язок між рівнем захисту цілісності покажчиків і планувальністю системи реального часу та пропонує фреймворк, який застосовує ці механізми в межах доступного системного slack, максимізуючи загальний рівень безпеки без негативного впливу на часові гарантії виконання.

Проблеми безпеки пам’яті залишаються одними з найдавніших і найпоширеніших у галузі програмної безпеки [54, 55]. Основною причиною таких загроз є використання низькорівневих мов програмування, зокрема C та C++, у яких керування пам’яттю покладається безпосередньо на програміста [56]. Розподіл пам’яті, робота з покажчиками та контроль доступу реалізуються вручну, що забезпечує високу продуктивність і гнучкість, але водночас значно підвищує ризик помилок. Оскільки людський фактор неминучий, некоректне управління пам’яттю часто призводить до вразливостей її пошкодження, які можуть бути використані зловмисниками [57]. Такі вразливості відкривають шлях до несанкціонованого доступу або модифікації пам’яті. Класичним прикладом є переповнення буфера стеку, коли відсутність перевірок меж дозволяє записувати дані за межами буфера та перезаписувати критично важливі елементи, зокрема адресу повернення функції. У результаті зловмисник може змінити логіку виконання програми, спрямовуючи її до шкідливого коду, що порушує цілісність керуючого потоку. Подібні атаки зазвичай проявляються як аномальна або неочікувана поведінка програми під час виконання [58]. З огляду на це, у спільноті безпеки було запропоновано примітиви безпеки під час виконання, які ґрунтуються на моніторингу поведінки програм у реальному часі [59]. Їхня робота починається з офлайн–аналізу програми для формування політики безпеки, яка описує дозволену поведінку. Далі ця політика інтегрується в програму за допомогою програмних або апаратних механізмів, після чого під час виконання здійснюється постійний контроль дотримання визначених правил. Будь–яке відхилення від очікуваної поведінки інтерпретується як потенційна атака. Основна відмінність між такими примітивами полягає у виборі властивостей, які вважаються критичними для безпеки.

Одним із ключових підходів є цілісність потоків даних, яка зосереджується на виявленні неочікуваних шляхів поширення даних у пам’яті. Атаки часто починаються з перезапису пам’яті за межами дозволеного діапазону, що створює аномальні потоки даних, наприклад, коли зовнішній ввід впливає на критичні змінні. Механізм цілісності потоків даних відстежує, які інструкції мають право

змінювати конкретні області пам’яті, і перевіряє це під час кожного доступу. Якщо дані були змінені неочікуваною інструкцією, атака фіксується [60]. Іншим важливим примітивом є цілісність керуючого потоку, спрямована на захист від атак перехоплення керування [61]. У таких сценаріях зловмисник змінює дані, пов’язані з керуванням виконанням, наприклад адресу повернення функції, що призводить до переходу в несанкціоновану точку коду. Цілісність керуючого потоку забезпечує відповідність усіх переходів наперед визначеному графу керування, побудованому за допомогою статичного аналізу. Будь–який перехід, що не відповідає цьому графу, розглядається як порушення безпеки.

Окремий клас загроз пов’язаний із пошкодженням покажчиків, оскільки вони визначають, до якої області пам’яті здійснюється доступ [62]. Маніпуляція значенням покажчика може дозволити зловмиснику змінити критичні дані, наприклад отримати адміністративні привілеї. Цілісність покажчиків як примітив безпеки гарантує, що значення покажчика змінюється лише дозволеними інструкціями. Зазвичай це реалізується шляхом створення криптографічного підпису, який перевіряється під час використання покажчика. Якщо значення було змінено стороннім чином, перевірка не проходить і атака виявляється.

Особливе значення ці підходи мають для систем реального часу, які працюють за суворими часовими обмеженнями. У таких системах навантаження моделюється як набір періодичних завдань, для яких визначаються період випуску, дедлайн і найгірший час виконання [63]. WCET описує максимальний можливий час виконання завдання з урахуванням характеристик апаратного забезпечення і використовується як консервативна оцінка. Якщо система здатна виконувати всі завдання в межах дедлайнів за такого припущення, її своєчасність вважається гарантованою, що відкриває можливості для інтеграції механізмів безпеки без порушення часових вимог.

Попри неминучий компроміс [64, 65] між рівнем безпеки та продуктивністю, КФС РЧ мають специфічні властивості, які відкривають додаткові можливості для підвищення захищеності. Однією з таких властивостей є суттєва різниця між середнім часом виконання завдань і їхнім найгіршим часом виконання. У системах

реального часу критично важливо, щоб усі завдання завершувалися до встановлених дедлайнів, оскільки їх порушення може призвести до серйозних фізичних наслідків [66]. Для цього планування зазвичай базується на WCET, що гарантує своєчасність навіть у найгіршому сценарії. Водночас такі сценарії трапляються рідко, і на практиці завдання часто виконуються значно швидше, залишаючи невикористаний резерв часу, відомий як slack [67]. Дослідження показують, що WCET може в кілька разів перевищувати середній час виконання, унаслідок чого система після завершення завдань простоює. Цей резерв уже використовувався в попередніх роботах для додаткових перевірок або виконання некритичних задач.

На основі цього спостереження запропоновано підхід, який застосовує slack для перевірки цілісності потоків даних [67]. Це дає змогу реалізувати обчислювально затратні механізми на вбудованих платформах реального часу без порушення часових обмежень. На відміну від попередніх рішень, де перевірки виконуються після завершення завдання, цей механізм діє як вбудований монітор під час виконання, що суттєво ускладнює поєднання вимог безпеки та планування [68]. Основні складнощі з ним пов’язані з оцінкою обсягу slack, формуванням політики безпеки та її ефективним застосуванням. Для оцінки slack використовується аналіз вхідних даних і контексту виконання, що дозволяє виключати нереалістичні шляхи найгіршого випадку. Щоб уникнути великих додаткових витрат, обмеження та відповідні оцінки WCET попередньо зберігаються в компактному вигляді, навіть ціною консервативної недооцінки slack. Для зменшення додаткових витрат під час виконання він також використовує апаратні механізми ізоляції пам’яті, розділяючи змінні на незахищені, захищені та метадані [69]. Перетворення slack на ефективний захист потребує балансування між важливістю даних і впливом перевірок на WCET. Для цього він формулює просторові та часові властивості когерентності безпеки, які гарантують, що захищені дані не можуть бути скомпрометовані через непрямі залежності або в моменти, коли захист вимкнено. Оскільки політика безпеки змінюється під час виконання, система використовує кілька заздалегідь підготовлених версій коду з

різними рівнями захисту та динамічно перемикається між ними, уникаючи модифікації бінарного файлу [70]. Прототип в роботі [71] було реалізовано для 64–х розрядної архітектури та оцінено на восьми платформах реального часу КФС.

Хоча безпосереднє застосування супроводжується значними додатковими витратами, різниця між найгіршим часом виконання і середнім часом створює додаткові можливості для підвищення безпеки [72]. Щоб скористатися цим резервом, запропоновано опортуністичний підхід [73], який виконує перевірку цілісності потоків даних лише тоді, коли доступна слабина. На відміну від традиційного підходу, що збільшує WCET, у цьому підході вбудовані монітори безпеки активуються лише для частини потоків даних, а рівень покриття динамічно регулюється залежно від доступного slack [74]. Якщо резерв часу зменшується, то обсяг перевірок скорочується. Така архітектура потребує тонкого балансу. З одного боку, WCET має залишатися незмінним, а з іншого боку навіть частковий захист повинен забезпечувати достатню надійність [75].

У роботі [76] використовується стандартна модель загроз, зосереджена на атаках, пов’язаних із пошкодженням пам’яті та орієнтованих на дані. Припускається наявність уразливості, що дозволяє довільне читання й запис пам’яті, але ін’єкція коду запобігається механізмами захисту пам’яті. Також вважається, що обходу інструментів безпеки можна уникнути за допомогою апаратних засобів. Для ізоляції використовується «пісочниця», реалізована апаратно, що підвищує ефективність. Водночас атаки на апаратне забезпечення, бічні канали та ОС не розглядаються, оскільки фокус роботи це безпека прикладного рівня.

Ключовим елементом дизайну в роботі [75] є оцінка slack під час виконання. Slack визначається як різниця між WCET і фактичним часом виконання, але до завершення завдання його можна лише недооцінити. Для цього використовується залежність часу виконання від вхідних даних і контексту. Знаючи частину цих даних на різних етапах виконання, система може довести, що деякі шляхи виконання є неможливими, і відповідно оновити оцінку WCET. В роботі [76] застосовано символічне виконання для збору обмежень шляхів і використано

попередньо обчислену таблицю, щоб швидко визначати консервативну оцінку slack. Оскільки облік усіх можливих обмежень є надто дорогим, використовується лише їх підмножина [77]. Для підвищення ефективності ці обмеження відбираються з урахуванням реальних розподілів вхідних даних КФС, отриманих шляхом профілювання на рівні місій. Такий підхід дозволяє суттєво підвищити точність оцінки slack і, як наслідок, розширити можливості опортуністичного захисту. Використання slack означає, що захист застосовується лише до частини програми [78], яку називають зоною захисту. Це породжує три ключові питання щодо захисту операцій доступу до пам’яті, гарантування ефективності витрат на захист в доступний резерв часу, збереження узгодженості безпеки, залишаючи решту програми без перевірок.

В роботі [79] використано політику безпеки для динамічного вибору цілей захисту під час виконання. Ця політика відображає оцінений стан виконання, обчислений на основі поточних вхідних даних і контексту, у множину перевірюваних операцій пам’яті. Під час синтезу цієї політики враховуються два ключові чинники. Перший чинник це імовірність використання змінних. Змінні, до яких не очікується доступу до наступної контрольної точки оцінки slack, включаються без додаткових витрат, тоді як для інших використовується статистична оцінка ймовірності доступу. Другий чинник це критичність змінних з погляду безпеки. Змінні з великою кількістю залежностей або ті, що впливають на фізично критичні параметри системи, отримують вищий пріоритет. Для цього дозволено користувачеві задавати політики критичності, які враховуються у вигляді ваг під час побудови політики. Щоб зберегти часові гарантії, він накладає обмеження. Для кожного стану вартість перевірок не може перевищувати оцінену величину slack. Витрати виконання політики безпеки визначаються консервативно, як максимальні можливі для всіх шляхів, сумісних із витратами на стани. Оскільки політика може змінюватися в контрольних точках оцінки slack, то шляхи виконання розбиваються на підшляхи, для яких окремо оцінюється час виконання.

Опортуністичний характер захисту вимагає також узгодженості безпеки у двох вимірах [80]. У просторовому вимірі це означає, що якщо змінна захищається,

то мають бути захищені й усі її дані та керуючі залежності. У часовому вимірі змінна повинна залишатися захищеною протягом усього циклу виконання завдання, оскільки тимчасове зняття захисту створює можливість для атак. Тому межі захищеної області можуть лише звужуватися, але не розширюватися під час виконання.

Динамічне оновлення захищеної області ускладнює застосування політик безпеки, оскільки традиційні вбудовані монітори не призначені для зміни під час виконання. Вого розв’язує цю проблему за допомогою динамічного перемикання коду. Ссистема підтримує кілька версій одного й того самого коду з різними рівнями захисту та перемикається між ними в контрольних точках slack. Щоб зменшити витрати пам’яті, кількість версій обмежується, а спільні ділянки коду розділяються за допомогою MMU. Для збереження ізоляції під час зміни меж захисту він використовує апаратні можливості. Захищені та незахищені області позначаються різними тегами, що дозволяє виявляти несанкціоновані записи під час перевірених операцій читання. Завдяки інструкції STGP оновлення даних і тегів виконується без додаткових витрат. Щоб не блокувати легітимне читання захищених даних неперевіреним кодом, використовується додаткова сторінка пам’яті лише для читання, а всі неконтрольовані записи додатково інструментуються для запобігання маніпуляціям тегами.

Таким чином, розглянуті підходи демонструють, що точна й консервативна оцінка slack під час виконання є ключем до поєднання жорстких часових обмежень із підвищеним рівнем безпеки КФС. Використання символічного виконання, профілювання вхідних даних і динамічних політик захисту дозволяє опортуністично застосовувати перевірки цілісності лише там і тоді, де це можливо без порушення WCET. Забезпечення просторової та часової узгодженості захисту, а також підтримка динамічного перемикання коду з апаратною ізоляцією, робить такий підхід практичним і ефективним, суттєво розширюючи потенціал безпеки систем реального часу без втрати їх своєчасності.

1.3 Постановка задачі

Поставлена мета досягається розв’язанням таких основних завдань:

1) проаналізувати існуючі підходи до забезпечення безпеки кіберфізичних систем реального часу та визначити їхні обмеження щодо жорсткого резервування ресурсів і додаткових часових витрат;

2) розробити метод оптимізації продуктивності та захищеності кіберфізичних систем на основі балансування завдань і ресурсів;

3) розробити адаптивні механізми захисту цілісності потоків даних, потоку керування та покажчиків із урахуванням планування завдань і динамічного розподілу ресурсів;

4) розробити комплексну модель та архітектуру КФС з інтеграцією просторової й часової ізоляції та динамічного управління рівнями безпеки;

5) провести експериментальну оцінку ефективності запропонованого методу з точки зору своєчасності виконання завдань, рівня захисту та додаткових витрат безпеки.

1.4 Висновки до першого розділу

Унікальні властивості КФС РЧ, зокрема консервативне резервування WCET і наявність slack time, створюють можливість істотно зменшити додаткові витрати механізмів безпеки без порушення жорстких часових обмежень. Адаптація примітивів цілісності потоків даних, керуючого потоку та вказівників із урахуванням моделей планування завдань дозволяє переносити обчислювальні витрати перевірок у доступні часові резерви. У поєднанні з просторовою й часовою узгодженістю безпеки та апаратно підтриманим динамічним перемиканням коду такий підхід забезпечує практичний і ефективний баланс між високим рівнем захисту КФС і гарантованою своєчасністю виконання.

2 МОДЕЛІ ОПТИМІЗАЦІЇ ПРОДУКТИВНОСТІ ТА ЗАХИЩЕНОСТІ КІБЕРФІЗИЧНИХ СИСТЕМ

2.1 Моделі механізмів цілісності потоків даних, керуючого потоку, вказівників і модель оптимізації продуктивності та захищеності КФС

Забезпечення високого рівня захищеності кіберфізичних систем реального часу є однією з ключових проблем сучасних вбудованих обчислювальних систем, оскільки вимоги до інформаційної безпеки безпосередньо накладаються на жорсткі часові обмеження виконання завдань. На відміну від загального призначення обчислювальних платформ, у КФС порушення часових гарантій може призводити не лише до логічних помилок, а й до фізичних наслідків, що істотно підвищує ціну будь–яких додаткових обчислювальних витрат. У цьому контексті застосування традиційних механізмів безпеки без урахування специфіки планування завдань реального часу є методологічно необґрунтованим і не дозволяє забезпечити формальні гарантії своєчасності.

Переважна більшість існуючих підходів до захисту програмного забезпечення орієнтована на максимізацію рівня безпеки за умови фіксованого або емпірично допустимого зниження продуктивності. Такий підхід фактично ігнорує той факт, що в системах реального часу продуктивність не є оптимізованою величиною, а виступає обмеженням, яке не може бути порушене. У результаті проблема інтеграції механізмів безпеки зводиться до пошуку допустимого компромісу, тоді як задача оптимального розподілу обчислювальних ресурсів між функціональними та захисними операціями формально не ставиться і, відповідно, не розв’язується.

Разом з тим фундаментальною особливістю систем реального часу є використання консервативних оцінок найгіршого часу виконання завдань, які необхідні для забезпечення часової коректності за будь–яких допустимих умов виконання. Така консервативність неминуче призводить до появи резервного часу виконання, який є побічним, але систематичним результатом застосування статичних методів аналізу часу виконання. Однак за відсутності формального

апарату цей резерв часу залишається неструктурованим і не може бути використаний як об’єкт оптимізації.

Таким чином, для переходу від констатації наявності slack time до його цілеспрямованого використання в інтересах підвищення захищеності системи необхідно побудувати моделі, які дозволяють кількісно описати процес його формування, еволюції та споживання під час виконання завдань. Без такого опису будь–які рішення щодо включення або адаптації механізмів безпеки залишаються евристичними, а їх вплив на часові характеристики буде непередбачуваним. Відсутність формалізованих моделей унеможливлює постановку задачі оптимізації в строгому математичному сенсі. Особливу складність становить той факт, що ключові примітиви забезпечення цілісності виконання, які включають контроль потоку даних, керуючого потоку та вказівників, мають різну структурну природу. Потік даних описується інформаційними залежностями між об’єктами програми, керуючий потік – графом переходів між базовими блоками, а цілісність вказівників – відношеннями доступу до адресного простору. Кожен із цих примітивів генерує власний тип додаткових витрат і по–різному взаємодіє з моделями планування завдань. Відтак оптимізація можлива лише за умови їх формалізації у вигляді узгоджених структурних моделей, здатних відображати як функціональні, так і часові характеристики.

Додатково, наявність slack time не гарантує можливості його безпечного використання без порушення дедлайнів. Для цього необхідна точна, але водночас консервативна оцінка доступного резерву часу під час виконання, що потребує врахування варіативності вхідних даних, умов виконання та внутрішньої структури програм. Саме тому побудова моделей у цій роботі спирається на поєднання статичних методів аналізу, зокрема символічного виконання, з динамічними методами профілювання, що дозволяє отримати верхні межі часових характеристик без втрати коректності.

Тому, оптимізація продуктивності та захищеності кіберфізичних систем не може бути досягнута шляхом ізольованого вдосконалення окремих механізмів безпеки. Вона потребує побудови цілісного набору математичних і графових

моделей, які формалізують взаємодію між завданнями реального часу, доступними обчислювальними ресурсами та структурою захисних примітивів. Лише на основі таких моделей стає можливим формальне формулювання оптимізаційної задачі балансування між рівнем захисту та гарантованою своєчасністю виконання, що і становить основну мету подальшого дослідження в межах даного параграфа.

На рис. 2.1 зображено архітектуру узагальненої КФС з урахуванням особливостей оптимізації продуктивності та захищеності.

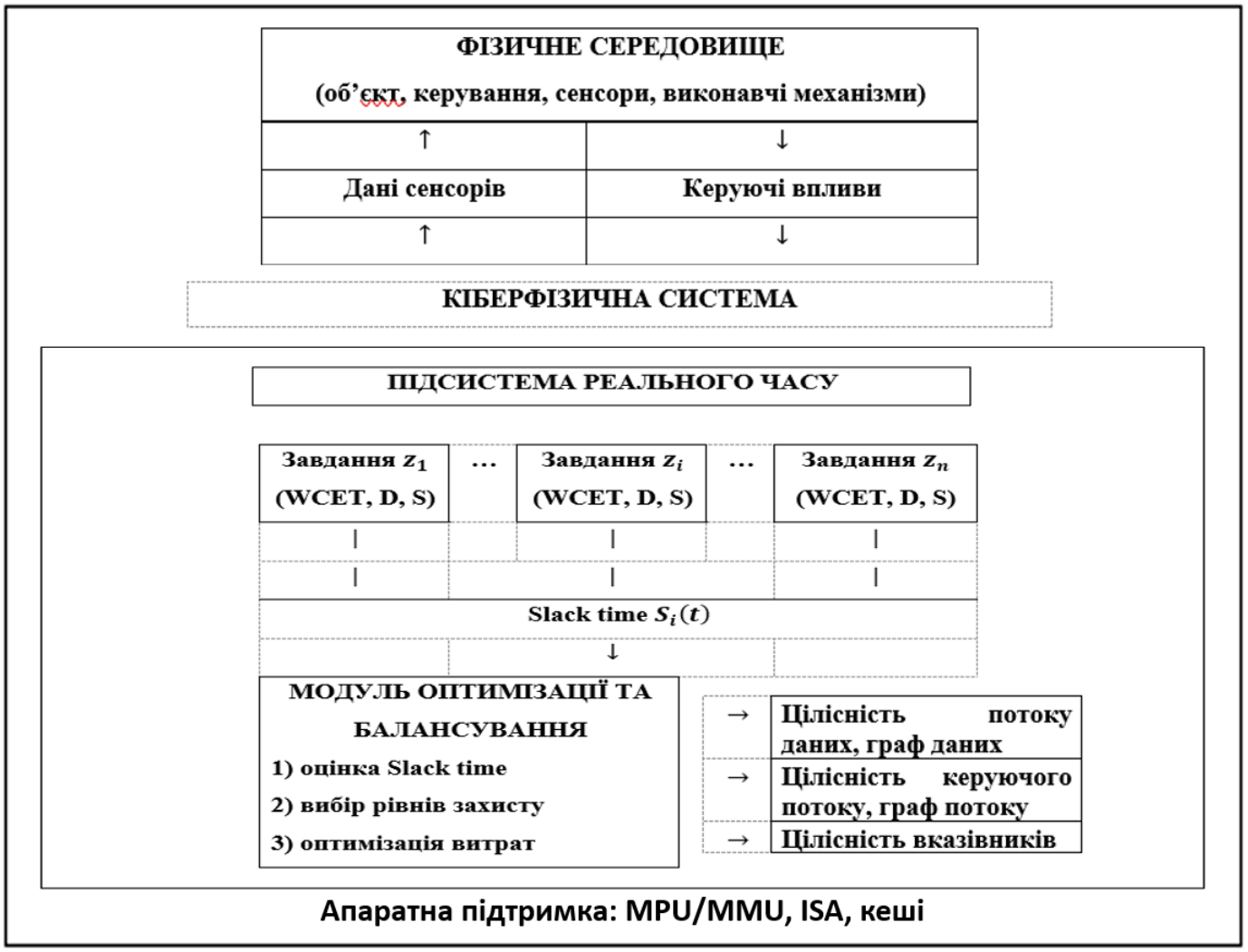


Рисунок 2.1 – Узагальнена архітектура кіберфізичної системи з локалізацією області дослідження та механізмів оптимізації продуктивності й захищеності

На рис. 2.1 представлено узагальнену архітектуру кіберфізичної системи з виділенням області дослідження та основних механізмів забезпечення цілісності

виконання. Фізичне середовище взаємодіє з обчислювальною частиною КФС через потоки даних сенсорів та керуючих впливів, що формує замкнений контур керування.

Центральним елементом моделі є підсистема завдань реального часу, для яких задано консервативні оцінки WCET та дедлайни. Унаслідок консервативності оцінок формується резерв часу виконання (slack time), який розглядається як потенційний ресурс для виконання додаткових захисних операцій.

Модуль оптимізації та балансування, що є безпосереднім об’єктом дослідження, використовує інформацію про доступний slack time для динамічного вибору та налаштування механізмів забезпечення цілісності. До таких механізмів належать контроль потоку даних, контроль керуючого потоку та контроль цілісності вказівників і доступу до пам’яті. Кожен із них має власну структурну модель та характеризується специфічними обчислювальними витратами.

Оптимізація полягає у розподілі доступного резерву часу між цими механізмами таким чином, щоб максимізувати рівень захищеності системи без порушення часових обмежень. Апаратна підтримка використовується для мінімізації додатких витрат та забезпечення ефективного перемикання політик захисту.

Кіберфізичні системи реального часу характеризуються жорсткими часовими обмеженнями, високими вимогами до надійності та зростаючою потребою у забезпеченні інформаційної безпеки. На відміну від традиційних обчислювальних систем, у КФС порушення термінів виконання завдань може призводити до фізичних наслідків, що суттєво ускладнює застосування класичних механізмів захисту.

Важливою особливістю таких систем є використання консервативних оцінок часу виконання, які навмисно перевищують фактичні витрати обчислювального часу. Це призводить до появи резервного часу, який у більшості реалізацій не використовується. Розглядатимемо slack time як керований ресурс, який може бути використаний для виконання додаткових перевірок безпеки без порушення часових

обмежень. Розробимо моделі механізмів цілісності потоків даних, керуючого потоку, вказівників і модель оптимізації продуктивності та захищеності КФС.

Для моделі завдань КФС визначимо множину завдань РЧ формулою (2.1):

$$Z_{CPS} = \left\{ z_{CPS,1}, z_{CPS,2}, \dots, z_{CPS,N_{Z_{CPS}}} \right\}, \quad (2.1)$$

де $z_{CPS,i}$ – i – те завдання реального часу;

$i = 1,2, \dots, N_{Z_{CPS}}$;

$N_{Z_{CPS}}$ – кількість завдань КФС.

Кожне завдання з множини завдань Z_{CPS} характеризується такими параметрами:

1) $c_i(t)$ – фактичний час виконання на момент часу t;

2) $O_{WCET,i}$ – консервативна оцінка найгіршого часу виконання;

3) D_i – відносний дедлайн;

4) T_i – період або мінімальний інтервал активації.

Резервний час визначається формулою (2.2):

$$s_i(t) = O_{WCET,i} - c_i(t), \quad (2.2)$$

де $c_i(t)$ – фактичний час виконання на момент часу t;

$i = 1,2, \dots, N_{Z_{CPS}}$;

$N_{Z_{CPS}}$ – кількість завдань КФС;

$O_{WCET,i}$ – консервативна оцінка найгіршого часу виконання.

Цей показник $s_i(t)$ згідно формули (2.2) використовується як динамічне обмеження для виконання додаткових захисних операцій.

Цілісність потоку даних (ЦПД, Data–Flow Integrity, DFI) спрямована на запобігання несанкціонованим модифікаціям або використанню даних у програмі. Модель механізму цілісності потоку даних задамо за формулою (2.3):

$$G_{d,j} = \left(V_{d,j}, E_{d,j}\right), \tag{2.3}$$

де j – номер потоку даних;

$V_{d,j}$ – множина змінних, буферів та регістрів;

$V_{d,j} = \left\{v_{d,j,1}, v_{d,j,2}, \dots, v_{d,j,N_{V_{d,j}}}\right\}$;

$k = 1,2, \dots, N_{V_{d,j}}$;

$N_{V_{d,j}}$ – кількість елементів множини $V_{d,j}$;

$E_{d,j}$ – допустимі інформаційні залежності «запис–читання»;

$E_{d,j} \subset V_{d,j} \times V_{d,j}$.

Ребро $\left(v_{d,j,s}, v_{d,j,m}\right)\epsilon E_{d,j}$ $(s \leq k; m \leq k)$, що означає дозвіл значенню записаному у $v_{d,j,s}$ використати для формування значення $v_{d,j,m}$.

Умова цілісності полягає в тому, що кожне зчитування даних має відповідати дозволеному попередньому запису. Обчислювальні витрати перевірок ЦПД для завдання оцінимо згідно фомули (2.4):

$$O_{\text{ЦПД},i} = k_d \cdot \left|E_{d,i}^1\right|, \tag{2.4}$$

де k_d – середня вартість перевірки однієї залежності;

$E_{d,j}$ – допустимі середні значення залежності «запис–читання».

Для врахування часових обмежень введемо бінарну змінну активації (2.5):

$$A_{\text{ЦПД},i} = \begin{cases} 1, \text{якщо } O_{\text{ЦПД},i} \leq s_i(t); \\ 0, \text{інакше}, \end{cases} \tag{2.5}$$

де $O_{\text{ЦПД},i}$ – обчислювальні витрати перевірок ЦПД для завдання;

$s_i(t)$ – резервний час.

Таким чином, перевірки виконуються лише за наявності достатнього резерву часу.

Розглянемо приклад графа потоку даних в завданні керування. Нехай дано чотири елементи множини $V_{d,j}$ такі:

1) $v_{d,j,1}$ – сенсор;

2) $v_{d,j,2}$ – фільтр;

3) $v_{d,j,3}$ – стан;

4) $v_{d,j,4}$ – актуатор.

А також, дано допустимі залежності так: $E_{d,j} = \{(v_{d,j,1}, v_{d,j,2}), (v_{d,j,2}, v_{d,j,3}), (v_{d,j,3}, v_{d,j,4})\}$. Тоді, графічно отримуємо послідовність $v_{d,j,1} \to v_{d,j,2} \to v_{d,j,3} \to v_{d,j,4}$, яка відображає граф потоку даних. Цей граф описується матрицею суміжності A_j за формулою (2.6):

$$A_j = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}, \tag{2.6}$$

де рядок – це джерело запису; стовпець – це ціль читання;
ненульовий елемент означає дозволений потік даних.

Матриця інцидентності повинна враховувати, граф є орієнтований, тому значення її елементів будуть визначатись як у формулі (2.7):

$$B_j = |b_{j,s,l}|,\ b_{j,s,l} = \begin{cases} -1, \text{якщо } v_{d,j,s} \text{ є початковою вершиною дуги } e_{d,j,l}; \\ +1, \text{якщо } v_{d,j,s} v_{d,j,s} \text{ є кінцевою вершиною дуги } e_{d,j,l}; \\ 0, \text{інакше}, \end{cases} \tag{2.7}$$

де j – номер потоку даних;

$V_{d,j}$ – множина змінних, буферів та регістрів;

$V_{d,j} = \{v_{d,j,1}, v_{d,j,2}, \dots, v_{d,j,N_{V_{d,j}}}\}$;

$k = 1,2, \dots, N_{V_{d,j}}$;

$N_{V_{d,j}}$ – кількість елементів множини $V_{d,j}$;

$E_{d,j}$ – допустимі інформаційні залежності «запис–читання»;

$E_{d,j} \subset V_{d,j} \times V_{d,j}$;

$E_{d,j} = \left\{e_{d,j,1}, e_{d,j,2}, \dots, e_{d,j,N_{E_{d,j}}}\right\}$;

$u = 1,2, \dots, N_{E_{d,j}}$;

$N_{E_{d,j}}$ – кількість елементів множини $E_{d,j}$.

Тоді, для ребер $e_{d,j,1} = \left(v_{d,j,1}, v_{d,j,2}\right), e_{d,j,2} = \left(v_{d,j,2}, v_{d,j,3}\right), e_{d,j,3} = \left(v_{d,j,3}, v_{d,j,4}\right)$ матриця інцидентності має такий вигляд формули (2.8):

$$B_j = \begin{pmatrix} -1 & 0 & 0 \\ +1 & -1 & 0 \\ 0 & +1 & -1 \\ 0 & 0 & +1 \end{pmatrix}, \tag{2.8}$$

де рядки – це вершини графа;

стовпці – дуги графа.

Виявлення непрямих залежностей використаємо матрицю досяжності (2.9):

$$R_j = A_j^1 \vee A_j^2 \vee \dots \vee A_j^{m-1}, \tag{2.9}$$

де j – номер потоку даних;

A_j – матриця суміжності.

Розглянемо такий приклад (2.10):

$$R_j = \begin{pmatrix} 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}, \tag{2.10}$$

де $R_{j,s,w} = 1$ – це означає, що дані з $v_{d,j,s}$ можуть впливати на дані у $v_{d,j,w}$.

Будь–яка фактична залежність, яка є відсутньою в R_j, є порушенням ЦПД.

Здійснимо перевірку цілісності даних у матричному вигляді (2.11):

$$A_j^1(t) \leq R_j, \tag{2.11}$$

де $A_j^1(t)$ – матриця фактичних залежностей під час виконання і нерівність виконується покомпонентно.

Умова невиконання нерівності, яку задано за формулою (2.11), така:

$$\exists\ (p,w)\colon a_{p,w}^1(t) = 1\ \wedge\ r_{p,w} = 0. \tag{2.12}$$

Розглянемо врахування slack time. Кожній перевірці дуги відповідає вартість k_d. Нехай $E_j^1 \subseteq E_j$ – множина перевіряючих дуг. Тоді, обмеження задамо (2.13):

$$\left|E_j^1\right| \cdot k_d \leq S_j(t), \tag{2.13}$$

де k_d – вартість.

Таким чином, матриця перевірок є розрідженою підматрицею A_j (формула (2.6)).

Потік даних завдання КФС моделюється орієнтованим графом, вершини якого відповідають об’єктам даних, а ребра – допустимим інформаційним залежностям. Для формального подання графа використовується матриця суміжності, а для аналізу непрямих впливів – матриця досяжності. Перевірка цілісності потоку даних зводиться до порівняння матриці фактичних залежностей під час виконання з еталонною матрицею досяжності, побудованою на етапі аналізу програми. Обсяг перевірок обмежується доступним резервом часу виконання завдання.

Розглянемо модель механізму цілісності керуючого потоку. Цілісність керуючого потоку (ЦКП, Control–Flow Integrity, CFI) забезпечує виконання програми виключно відповідно до дозволеного графа переходів.

Керуючий потік моделюватимемо графом за формулою (2.14):

$$G_{d,j} = \left(V_{c,j}, E_{c,j}\right), \tag{2.14}$$

де вузли відповідають базовим блокам програми, а ребра – допустимим переходам між ними;

j – номер потоку даних;

$V_{c,j}$ – множина базових блоків програми;

$V_{c,j} = \left\{v_{c,j,1}, v_{c,j,2}, \dots, v_{c,j,N_{V_{c,j}}}\right\}$;

$k = 1,2, \dots, N_{V_{c,j}}$;

$N_{V_{c,j}}$ – кількість елементів множини $V_{c,j}$;

$E_{c,j}$ – множина допустимих переходів між базовими блоками програми.

Перевірка полягає у валідації наступної адреси виконання (2.15):

$$A_{p,next} = F_A\left(A_{p,current}\right), \tag{2.15}$$

де $A_{p,next}$ – наступна адреса в керуючому потоці;

$A_{p,current}$ – поточна адреса в керуючому потоці;

F_A – функція перетворення адрес.

Для зменшення додаткових витрат вводиться дискретний рівень деталізації перевірок згідно з формули (2.16):

$$L_{\text{ЦКП}} \in \{0,1,2\}, \tag{2.16}$$

де 0 – перевірки вимкнені;

1 – груба перевірка;

2 – детальна перевірка.

Витрати перевірок визначимо за формуло (2.17):

$$O_{\text{ЦКП},j}(L) = k_c \cdot \left|E^1_{c,j}(L)\right|, \quad (2.17)$$

де k_c – коефіцієнт вартості.

Рівень $L_{\text{ЦКП}}$ обирається таким чином, щоб максимізувати захищеність за наявного slack time.

Модель механізму цілісності вказівників і пам'яті будемо розглядати в контексті спроможності здійснювати контроль коректності вказівників під час доступу до пам'яті.

Для кожного завдання $z_{CPS,i}$ з множини завдань реального часу Z_{CPS} (формула (2.1)) визначимо множину допустимих адрес формулою (2.18):

$$A_j^{valid} \subseteq A, \quad (2.18)$$

де $z_{CPS,i}$ – i – те завдання реального часу;

$i = 1,2, \dots, N_{Z_{CPS}}$;

$N_{Z_{CPS}}$ – кількість завдань КФС.

Умову цілісності можна задати формулою (2.19):

$$ptr \in A_j^{valid}, \quad (2.19)$$

де ptr – адреса вказівника.

Загальні витрати перевірок оцінюватимемо формулою (2.20):

$$O_{ptr,j} = k_p \cdot N_{ptr,j}, \quad (2.20)$$

де k_p – коефіцієнт вартості;

$N_{ptr,j}$ — кількість операцій розіменування вказівника j – того потоку.

З метою адаптації до часових обмежень застосовується вибіркова перевірка лише критичних доступів, що формалізується за допомогою бінарних змінних вибору.

Згідно розроблених попередніх трьох моделей цілісних потоків даних, керуючого потоку та цілісності вказівкиків розробимо узагальнену модель оптимізації балансування безпеки та продуктивності в КФС.

Для кожного завдання формуємо вектор параметрів безпеки (2.21):

$$q_j = \begin{bmatrix} A_{\text{ЦПД},j} \\ L_{\text{ЦКП},j} \\ B_{ptr,j} \end{bmatrix}, \tag{2.21}$$

де $A_{\text{ЦПД},j}$ – інформація щодо цілісних потоків даних;

$L_{\text{ЦКП},j}$ – інформація щодо керуючого потоку;

$B_{ptr,j}$ – інформація щодо вказівників.

Рівень захищеності системи задамо функцією (2.22):

$$S_1(Q) = \sum_{j=1}^{n} w_j \cdot F_2(q_j), \tag{2.22}$$

де w_j – коефіцієнти критичності завдань;

F_2 – функція оцінювання захищеності потоків даних, керуючого потоку та вказівників;

n – кількість потоків; j – номер потоку даних;

Q – множина завдань, яка складається з елементів q_j;

$j = 1,2, \dots, n.$

Задачу оптимізації балансування безпеки та продуктивності в КФС визначимо формулою (2.23):

$$\max_{Q} S_1(Q), \tag{2.23}$$

за умови (2.24)

$$c_j(t) + O_{\text{ЦПД},j} + O_{\text{ЦКП},j}(L) + O_{ptr,j} \leq O_{WCET,j} \quad (2.24)$$

та глобальних планувальних обмежень і при цьому:

$c_j(t)$ – фактичний час виконання на момент часу t;

$j = 1,2, \dots, N_{Z_{CPS}}$;

$N_{Z_{CPS}}$ – кількість завдань КФС;

$O_{WCET,j}$ – консервативна оцінка найгіршого часу виконання.

Таким чином, розроблено математичні моделі завдань КФС, а також трьох ключових механізмів забезпечення цілісності виконання. Запропоновано узагальнену оптимізаційну модель, що дозволяє динамічно балансувати між рівнем захищеності та часовими обмеженнями шляхом використання резервного часу виконання. Отримані моделі створюють теоретичну основу для подальшої розробки методу оптимізації та його експериментальної перевірки.

2.2 Архітектура КФС на основі оптимізації продуктивності та захищеності

Ефективність політики безпеки в контексті цілісності потоків даних залежить від численних взаємопов'язаних факторів. Серед них обов’язково наявні стратегічне формування операцій із захищеною пам'яттю, обдуманий вибір обмежень на шлях і визначення відповідних версій коду, адаптованих до різних рівнів пропускання, тобто визначених доступних часових або ресурсних меж в межах виконання системи. Крім того, ці елементи підлягають низці практичних перешкод, таких як обмеження в реальному часі, що визначають оперативну своєчасність, необхідність підтримання когерентності безпеки в різних станах виконання та обмеження, додаткі витратами на пам'ять. Тому, необхідно визначити оптимальну політику безпеки, яка максимізує цілі безпеки, не порушуючи жодного з трьох критичних обмежень: продуктивність у реальному часі; послідовність

безпеки; ефективність пам'яті. Для цього природно розглядаємо виклик як формальну задачу оптимізації, застосовуючи математичне моделювання для систематичного дослідження компромісів і виведення рішень, які збалансовують захист із здійсненністю. Вводимо кількісну метрику, яку називають «оцінкою покриття» для оцінки досягнення цілей безпеки під час процесу оптимізації. Цей показник слугує індикатором широти та глибини захисту, які надає поліс. Спрощена методологія обчислення цього показника може полягати у простому перерахунку загальної кількості перевірених операцій з пам'яттю, розглядаючи кожну операцію як однаково значущу незалежно від її контексту чи потенційної вразливості. Однак такий підхід ігнорує нюанси операційної важливості та взаємозалежності. Відповідно, завдання оптимізації формулюється як ідентифікація оптимальної пари, яка представляє набір операцій із захищеною пам'яттю і враховує конфігурацію політики, яка максимізує показник покриття з урахуванням комплексного набору обмежень. Ці обмеження охоплюють обмеження часу виконання та додаткових витрат на пам'ять, які виникають, а також вимоги, які гарантують, що політика безпеки дотримується принципів когерентності безпеки, запобігаючи таким же невідповідностям, які можуть підірвати загальну захисну позицію.

Щоб ефективно розв'язати цю задачу оптимізації, застосуємо гібридну методологію, яка синергічно поєднує сильні сторони двох різних алгоритмічних парадигм. Зокрема, використовуємо генетичний алгоритм, відомий своєю надійністю у навігації в нелінійних і складних просторах пошуку через еволюційні принципи, такі як відбір, кросовер і мутація. Це доповнюється змішаним цілочисельним лінійним програмуванням, яке відзначається ефективними та точними розв'язками задач, що можна апроксимувати. Цей подвійний підхід дозволяє вирішувати вроджену нелінійність задачі, одночасно забезпечуючи обчислювальну зручність. Для більш детального викладу формулювання проблеми, включаючи математичні виведення та визначення змінних, емпірична оцінка, проведена на восьми різних платформах. Процес оптимізації може призвести до нульової політики, тобто конфігурації, яка не забезпечує жодного

захисту операцій з пам'яттю. Цей результат підкреслює стійкість ЦПД у практичних сценаріях. Проте цілком можливо, що за крайніх умов похідна політика може перетворитися на нульовий стан. Наприклад, у системах, що характеризуються поширеними багатофункціональними залежностями між змінними, для захисту будь–якої окремої змінної може знадобитися значна початкова інформація через її складні зв'язки з іншими. У таких тісно пов'язаних умовах життєздатні політики можуть виявитися невловими, що робить ЦПД нездатним забезпечити додаткові захисти. Це підкреслює важливість аналізу попередньої обробки для виявлення таких випадків і потенційної інтеграції резервних механізмів або гібридних політик для пом'якшення повних вакуумів захисту.

Прототип забезпечує сучасну, масштабовану основу для вбудованих і високопродуктивних систем. Архітектура системи модульна і складається з чотирьох основних компонентів: аналізатора Slack для оцінки часової межі; монітора референтного монітора з комутацією коду для динамічного застосування політик; інструментальних утиліт програм для вбудови перевірок безпеки та загального механізму оптимізації. На рис. 2.2 зображено схему для прототипу оптимізації.

Рисунок 2.2 – Схема прототипу оптимізації

Архітектура системи оптимізації для КФС представляє собою комплексну модульну структуру, яка базується на 64–х розрядній архітектурі і забезпечує сучасну, гнучку та масштабовану основу для різноманітних вбудованих систем, а також для високопродуктивних обчислювальних платформ, дозволяючи не тільки ефективно оптимізувати політики безпеки в рамках ЦПД, але й враховувати множинні аспекти, такі як часові обмеження, вимоги до пам'яті та загальні безпекові критерії, що робить її універсальною для застосування в реальних сценаріях з обмеженими ресурсами. Центральним елементом цієї архітектури є оптимізаційний двигун, який функціонує як головний координатор усіх взаємопов'язаних компонентів, генеруючи оптимальні конфігурації політики на основі інтегрованих даних з інших модулів, і це аргументується тим, що двигун систематично аналізує результати для досягнення максимального покриття безпеки, не порушуючи встановлених обмежень, таким чином забезпечуючи оптимальний баланс між загальною ефективністю системи та рівнем захисту від потенційних загроз, що є критичним для систем з реальними часовими вимогами. Цей оптимізаційний двигун встановлює двосторонні зв'язки зі всіма іншими блоками, оскільки він активно отримує вхідні дані від аналізатора запасу, монітора послідовності з перемиканням коду та утиліт інструментації програми, а в зворотному напрямку надає їм детально оптимізовані параметри та конфігурації, що обґрунтовується необхідністю в динамічній адаптації політики до змінних умов виконання програми, запобігаючи непотрібним перевитратам обчислювальних ресурсів і забезпечуючи стійкість до варіацій навантаження. Аналізатор запасу виступає як первинний ключовий блок у цій ієрархії, беручи на себе відповідальність за точну оцінку часового запасу (temporal margin assessment), де термін "запас" (slack) позначає доступні часові або ресурсні маржі в різних шляхах виконання коду, а "темпоральний маргін" є спеціалізованою метрикою для кількісного вимірювання цих маржинів, що аргументується використанням багатоступеневого таймінг–аналізу, включаючи профілювання характеристик базових блоків коду через динамічний аналіз, розгортання циклів на основі меж з аналізу планувальності системи, а також комбінацію детальної шляхової

інформації з низькорівневими даними таймінгу для розрахунку точних значень, при цьому для отримання символічних формул обмежень застосовується символічний рушій виконання на базі інструменту, що дозволяє уникнути проблеми вибуху шляхів і розумно обмежити глибину символічного виконання, запобігаючи обчислювальним перевантаженням. Цей аналізатор запасу безпосередньо пов'язаний з монітором послідовності з перемиканням коду, передаючи йому критичні дані про доступні запаси для інформованого динамічного вибору відповідних версій коду, що обґрунтовується потребою в ефективному та безпечному перемиканні між захищеними та незахищеними варіантами виконання коду без ризику порушення реальних часових рамок, де ключовий термін "динамічна політика" охоплює набір правил перемикання, які реалізовані у формі попередньо обчисленої таблиці пошуку з індексацією на основі поточного стану виконання, а термін "виконання" стосується механізмів забезпечення ізоляції через стратегічне вирівнювання адрес коду та маскування адрес на кожній непрямій гілці, аргументуючи цей тісний зв'язок тим, що без надійної оцінки запасу монітор не зміг би безпечно здійснювати перемикання, ефективно запобігаючи атакам на контроль потоку та підтримуючи загальну цілісність системи. Далі монітор послідовності з перемиканням коду встановлює зв'язок з утилітами інструментації програми, надаючи їм детальні інструкції та параметри для точного вбудовування необхідних перевірок безпеки, де термін "вбудовування перевірок безпеки" стосується процесу вставки спеціальних маскувальних інструкцій безпосередньо перед операціями читання та запису в пам'ять з використанням жорстко закодованих тегів, а навантаження за оптимізацією системи включає усунення маскувань для регістрів з частими доступами до пам'яті, наприклад, стекових операцій, з одночасним додаванням грубозернистого механізму ЦКП через інструкції на входах функцій та адресах повернення, а також захисту зворотних адрес через тіньовий стек з окремим тегом, аргументуючи цей зв'язок тим, що монітор забезпечує динамічне застосування політики на рівні виконання, тоді як інструментація статично інтегрує перевірки в код, створюючи потужну синергію для запобігання прямим атакам зловмисників

на інструментований код і надійного захисту всіх метаданих за допомогою виділених тегів, що в комплексі повертається назад до оптимізаційного двигуна для ітеративного удосконалення та рефайнінгу політики на основі зворотного зв'язку. Усі ці зв'язки формують замкнений цикл взаємодії, де аналізатор запасу постачає фундаментальні метрики та дані для монітора, монітор генерує та передає правила перемикання для інструментації, інструментація реалізує статичні перевірки для безпечного виконання, а оптимізаційний двигун здійснює глобальну оптимізацію для всіх компонентів, обґрунтовуючи цю структуру необхідністю в опортуністичній адаптації політики до динамічних умов, де ключові терміни, такі як когерентність безпеки, забезпечують повну узгодженість політики через різні стани виконання, покриття виступає як кількісна метрика для оцінки захищених операцій пам'яті, а "обмеження" визначають жорсткі рамки на час виконання, витрати пам'яті та загальні безпекові вимоги, роблячи всю систему високостійкою навіть до крайніх випадків, наприклад, коли спостерігаються багатофункціональні залежності між змінними, що вимагають значного початкового запасу для захисту індивідуальних елементів через їхні взаємозв'язки, і без таких інтегрованих зв'язків та механізмів оптимізація ризикувала б призвести до дегенерації політики до нульового стану, де система не надавала б жодного додаткового захисту, підкреслюючи важливість holistic підходу в дизайні для досягнення максимальної ефективності. Процес розгортається на кількох послідовних етапах для забезпечення точності та ефективності. Спочатку динамічний аналіз використовується для профілювання часових характеристик окремих базових блоків, тобто фундаментальних одиниць виконання коду, які фіксують метрики, такі як цикли виконання при різних навантаженнях. Далі цикли в коді розгортаються на основі меж, отриманих за допомогою аналізу системної планованості, які прогнозують кількість ітерацій і запобігають переоцінці часових додаткових витрат. Після цього інформація, специфічна для шляху, поєднується з низькорівневими часовими даними для отримання точних оцінок слабкості, враховуючи накопичені затримки вздовж траєкторій виконання.

Щоб отримати символічні формули для обмежень, використовуємо механізм символічного виконання, інструменти для дослідження станів програм і виявлення потенційних вразливостей. Оскільки поріг для оцінки Slack калібрований відповідно до вартості оцінок обмежених обмежень, то тим самим обмежується складність формул, але глибина символічного виконання навмисно обмежується. Це обмеження запобігає вибуху шляху, тобто поширеної пастки в символічному аналізі, коли кількість досліджених шляхів зростає експоненційно. З урахуванням обраних обмежень контрольні точки оцінки Slack стратегічно вставляються в точках оновлення змінних. Оскільки обмеження прив'язані до конкретних змінних змінних, кожна контрольна точка включає лічильник оновлень для розрізнення між кількома оновленнями однієї змінної, забезпечуючи детальний і контекстно–орієнтований моніторинг.

Для ефективного визначення перемикаючих цілей правила комутації, тобто автономно генеровані механізмом оптимізації, створюються як попередньо обчислена таблиця пошуку. Оцінений стан виконання слугує ключем індексації, що дозволяє швидко приймати рішення з мінімальними додатковими витратами на час виконання. Щоб запобігти потенційним зловживаючим експлойтам, таким як шкідливі стрибки у незахищені версії коду, кожен варіант коду інкапсулюється у спеціальному відсіку. Ця ізоляція досягається завдяки ретельному вирівнюванню адрес коду, що гарантує, що розбіжності проявляються виключно у бітах вищого порядку, що полегшує апаратну сегрегацію.

Спеціалізований регістр, який можна оновлювати виключно через монітор посилання, виділяється для зберігання ідентифікатора поточної версії коду. Цей ідентифікатор використовується для маскування адрес на кожній непрямій гілці, забезпечуючи таким чином постійне обмеження керуючого потоку в межах допустимих значень. Ця конструкція не лише підвищує безпеку від захоплення потоків керування, а й безшовно інтегрується з архітектурними особливостями, такими як автентифікація вказівників, для підвищення стійкості.

Для забезпечення жорстко закодованих тегів пам’яті, тобто апаратно підтримуваного механізму безпеки пам'яті, перед операціями читання та запису

пам'яті вставляються інструкції маскування. Проводячи паралелі з методами оптимізації у статичній пісочниці, надмірні маскування для часто доступних регістрів, ілюстровані у вигляді стекових операцій, уникаються для мінімізації втрат продуктивності. Щоб уникнути прямих суперницьких стрибків у інструментований код, застосовується грубозернистий механізм ЦКП як базовий рівень безпеки. Конкретно, інструкції вбудовані у кожен вхід функції та потенційну зворотну адресу, розмежовуючи дійсні цілі керуючого потоку. Крім того, адреси повернення підкріплюються тіньовим стеком, реалізованим з окремим тегом для виявлення втручань. Аналогічно, всі структури метаданих захищені спеціальним тегом, що забезпечує комплексний захист від несанкціонованих змін. Цей багаторівневий підхід до інструментування не лише підсилює межі безпеки, а й відповідає реальним обмеженням розгортання, пропонуючи прагматичний баланс між додатковими витратами та ефективністю у захисті цілісності потоків даних.

Хоча рівень захисту, який забезпечується механізмом ЦПД, безпосередньо залежить від доступної величини slack, сама необхідність вичерпання цієї слабкості істотно звужує можливості зловмисника щодо формування вхідних даних. Зокрема, для досягнення суттєвого збільшення часу виконання програми вхідні значення повинні відповідати суворо визначеним обмеженням шляхів виконання. Таким чином, зловмисник змушений працювати в обмеженому просторі допустимих входів, що ускладнює побудову ефективних атак. Тому, необхідно здійснити аналіз зазначеного компромісу з точки зору можливостей експлуатації системи адаптивним зловмисником.

Незважаючи на те, що конкретні техніки реалізації шелл–коду можуть суттєво відрізнятися залежно від позиції та сценарію атаки, для успішної експлуатації вразливостей у програмній системі зазвичай необхідне виконання двох базових умов. По–перше, зловмисник повинен мати можливість досягти вразливих функцій за допомогою контрольованих ним вхідних даних. По–друге, необхідною є здатність сконструювати такі вхідні значення, які при доставці активують відповідну вразливість. Водночас варто зауважити, що навіть за умови повного контролю над входом, внутрішня логіка програми, тобто механізми

очищення, нормалізації або перевірки даних, може істотно обмежувати або повністю унеможливлювати довільні маніпуляції вхідними значеннями, знижуючи тим самим ефективність потенційної атаки.

Першою метрикою експлойтності є рівень доступу зловмисника до функцій, чутливих з погляду безпеки. Це обґрунтовано тим, що виклики небезпечних функцій стандартної бібліотеки C та системних викликів часто слугують основними точками входу для реалізації експлойтів. Водночас не всі виклики функцій є релевантними з позиції атакуючого: лише ті з них, у які надходить потік даних, контрольований зловмисником, можуть бути використані для експлуатації. У зв’язку з цим у межах аналізу потрібно враховувавти винятково ті виклики функцій, чутливих до безпеки, які є досяжними з простору зловмисника. Кількісно ця метрика визначається як відсоток досяжних локацій, чутливих до безпеки, відносно їх загальної кількості.

Другою метрикою експлойтності є здатність зловмисника маніпулювати вхідними даними. Для визначення можливих меж таких маніпуляцій обмеження шляхів у найгіршому випадку виконання перетворюються у логічну формулу, яка формально описує допустимий діапазон вхідних значень. Кількісне оцінювання цієї метрики здійснюється шляхом вимірювання частки модифікованого діапазону відносно повного простору вхідних даних програми. Важливо підкреслити, що точне вимірювання можливості експлуатації потребує врахування значної кількості тонких і контекстно залежних аспектів.

Кількісні результати оцінювання обмежень для різних рівнів slack свідчать, що в найгіршому сценарії, за відсутності slack, доступ до точок, чутливих до безпеки, обмежується, тоді як для усього діапазону вхідних значень програми може бути використано для побудови експлойтів. Таке істотне обмеження зумовлене тим, що для проходження довших шляхів виконання вхідні дані мають одночасно задовольняти всі накладені на них обмеження, що різко звужує як простір можливих входів, так і варіанти керування потоком виконання. Зі збільшенням рівня slack зловмисник отримує більшу свободу у формуванні експлойтних корисних навантажень та доступі до ширшого спектра функцій, чутливих до

безпеки. Водночас зростає і ефективність захисту, який забезпечується ЦПК, оскільки більша величина slack дозволяє реалізувати посилені механізми контролю виконання.

Обмеження на вхідні значення кількісно визначаються як частка простору входів, яку зловмисник потенційно може використати для створення експлойтів. Іншими словами, це відсоток діапазону вхідних значень, що відповідає певному оціненому рівню slack, відносно повного простору допустимих входів програми.

Оскільки оцінювання slack у ЦПД є шляхозалежним, а успішна атака передбачає перенаправлення керування до вразливої ділянки коду, це накладає додаткові обмеження на можливість маніпуляції керуючим потоком. Для кількісного аналізу таких обмежень операції з масивами, а також виклики системних функцій і функцій маніпуляції пам’яттю стандартної бібліотеки C (зокрема strcpy та memcpy) розглядаються як точки, чутливі до безпеки. Це обґрунтовано тим, що помилки в обробці масивів або некоректні виклики зазначених функцій потенційно можуть надати противнику контроль над пам’яттю процесу чи системи.

Обмеження на керуючий потік кількісно визначаються як відсоток чутливих до безпеки точок, які залишаються досяжними за відповідного рівня slack. Зі зростанням slack ці обмеження поступово послаблюються, однак водночас механізми ЦПД забезпечують зростання рівня захисту, що в сукупності може забезпечити підвищену стійкість системи до експлуатації.

У результаті проведеного аналізу встановлено, що запропонована архітектура кіберфізичної системи, орієнтована на одночасну оптимізацію продуктивності та захищеності, є ефективним і збалансованим підходом для застосування в умовах обмежених ресурсів і жорстких часових вимог. Модульна побудова архітектури на основі 64–розрядної платформи забезпечує її гнучкість, масштабованість і придатність як для вбудованих систем, так і для високопродуктивних обчислювальних середовищ, що підтверджує універсальність запропонованого рішення в реальних сценаріях експлуатації КФС.

Ключову роль у досягненні такого балансу відіграє оптимізаційний двигун, який координує взаємодію між основними компонентами системи, інтегруючи дані аналізу slack, моніторингу виконання та інструментації програмного коду. Завдяки двосторонній взаємодії з іншими модулями він забезпечує динамічну адаптацію політик безпеки до змінних умов виконання, мінімізуючи зайві обчислювальні витрати та підтримуючи необхідний рівень захисту без порушення часових і ресурсних обмежень. Це є критично важливим для систем реального часу, де надмірні додаткові витрати можуть призвести до деградації функціональності.

Результати оцінювання демонструють, що за відсутності slack простір вхідних даних і доступ до точок, чутливих до безпеки, істотно обмежуються, що значно ускладнює побудову ефективних експлойтів. Таке звуження досяжного простору пояснюється необхідністю одночасного виконання всіх обмежень шляхів, що різко зменшує можливості маніпуляції як вхідними значеннями, так і керуючим потоком. Із підвищенням рівня slack потенційні можливості зловмисника формально розширюються, однак паралельно зростає і ефективність механізмів захисту, реалізованих у межах ЦПД, що дозволяє компенсувати цей ефект за рахунок посиленого контролю виконання.

Додатково показано, що шляхозалежний характер оцінювання slack накладає суттєві обмеження на керуючий потік, зменшуючи досяжність операцій і функцій, чутливих до безпеки, таких як маніпуляції з масивами та виклики небезпечних функцій стандартної бібліотеки C. Кількісне зменшення доступних точок експлуатації свідчить про підвищення стійкості системи до атак, спрямованих на отримання контролю над пам’яттю процесу або системи.

Таким чином, запропонована архітектура КФС із використанням механізмів оптимізації slack та централізованого керування політиками безпеки забезпечує ефективний компроміс між продуктивністю та захищеністю. Отримані результати підтверджують доцільність застосування даного підходу для побудови стійких до експлуатації кіберфізичних систем, здатних адаптуватися до змінних умов виконання без втрати функціональної ефективності.

2.3 Висновки до другого розділу

Таким чином, розроблено математичні моделі завдань кіберфізичних систем, а також трьох ключових механізмів забезпечення цілісності виконання, що функціонують в умовах жорстких часових та ресурсних обмежень. Запропонована узагальнена оптимізаційна модель дозволяє формалізувати компроміс між рівнем захищеності та продуктивністю системи шляхом керованого використання резервного часу виконання (slack). Показано, що динамічне регулювання slack дає змогу істотно обмежити простір можливих атак, зменшуючи доступність точок, чутливих до безпеки, а також діапазон вхідних даних, придатних для експлуатації, без порушення вимог реального часу. Архітектурні рішення з централізованим оптимізаційним двигуном забезпечують адаптивну зміну політик захисту відповідно до умов виконання та навантаження. Отримані математичні моделі формують цілісну теоретичну основу для подальшої розробки методу оптимізації, його практичної реалізації та експериментальної верифікації ефективності в реальних сценаріях застосування КФС.

3 МЕТОД ОПТИМІЗАЦІЇ ПРОДУКТИВНОСТІ ТА ЗАХИЩЕНОСТІ КІБЕРФІЗИЧНИХ СИСТЕМ НА ОСНОВІ БАЛАНСУВАННЯ ЗАВДАНЬ І РЕСУРСІВ

3.1 Стратегії зловмисників в атаках на КФС

Розглянемо стратегії зловмисників при здійсненні атак на КФС в контексті доступу до вразливостей, які породжені мовами C/C++.

Одним із потенційних векторів атаки є спроба зловмисника обійти інструментований референсний монітор з метою уникнення перевірок безпеки. Проте в архітектурі ЦПД для протидії таким сценаріям задіяно апаратну підтримку, яка забезпечує реалізацію грубозернистого контролю цілісності потоку керування ЦКП. Це обмежує можливості зловмисника таким чином, що передача керування можлива виключно до легітимних точок входу функцій і коректних адрес повернення. У результаті всі операції над даними підлягають перевірці, а перемикання між різними версіями коду допускається лише в межах спеціалізованих блоків комутації коду за умови, що всі непрямі переходи обмежені допустимим адресним простором.

Додатково ЦПД використовує апаратний механізм для забезпечення цілісності даних під час операцій запису. Усі дані, створені в захищеному регіоні, асоціюються з унікальними тегами, які відомі лише коду, що виконується в цьому регіоні. Будь–яка спроба запису з незахищеної області призводить до порушення тегів, що неминуче буде виявлено під час подальшого доступу з боку захищеного коду. Під час виконання ЦПД також підтримує динамічне звуження меж захищеної області за допомогою механізму перемикання версій коду. У новоактивованих версіях коду теги для вилучених регіонів жорстко закодовані таким чином, щоб вони відрізнялися від тегів активної захищеної області, що унеможливлює прихований вплив видаленого коду на критичні дані.

Оскільки вилучений регіон не має залишкових залежностей від активної захищеної області, механізм пісочниці ефективно ізолює незахищений код і запобігає пошкодженню даних. Крім того, метадані, що використовуються як ЦПД,

так і програмною реалізацією в межах захищеного регіону, також захищені. Система резервує спеціальні теги для метаданих і гарантує, що лише призначений референсний монітор має право їх модифікувати, що додатково підвищує загальну стійкість реалізації.

Хоча ЦПД не забезпечує абсолютного захисту всіх потоків даних, його адаптивна природа дозволяє зосередити ресурси на захисті найбільш критичних даних. Зокрема, експериментальні результати, отримані на восьми оцінених платформах, показують можливість досягнення середнього рівня покриття захисту для керуючих змінних. Для змінних, визначених як такі, що підлягають захисту, ЦПД забезпечує гарантії безпеки, співставні з повною реалізацією.

Оскільки ЦПД примусово застосовує перевірки в межах захищеного регіону, то будь–які незаконні потоки даних, що відхиляються від дозволеного графа потоків даних, можуть бути виявлені під час виконання. Теоретично зловмисник може намагатися впливати на захищені дані шляхом модифікації незахищених змінних, однак такий сценарій нівелюється завдяки забезпеченню когерентності безпеки. У будь–який момент часу захищена область охоплює всі дані та керуючі залежності, пов’язані із захищеними змінними. Навіть у разі динамічного зменшення захищеної області, вона зберігає свої властивості протягом усього періоду виконання ітерації завдання, що унеможливлює тимчасові вікна для атак.

Поточна реалізація ЦПД орієнтована на оптимізацію захисту в межах одного запуску завдання. Проте для періодичних завдань у КФС можуть існувати складні часові залежності між ітераціями, що потенційно обмежує доступний slack на початкових етапах. Одним із можливих напрямів пом’якшення цього обмеження є використання санітайзерів або фізичних імпульсів для відновлення критичних керуючих змінних до безпечного стану.

Додатковим обмеженням є залежність ЦПД від апаратних можливостей, що ускладнює перенесення системи на платформи без відповідної підтримки. Використання кількох версій коду призводить до певного зростання витрат пам’яті, хоча це частково компенсується спільним використанням сторінок коду. Подальша

оптимізація можлива за рахунок поліпшення макету коду, дрібнішого розбиття регіонів виконання та динамічного оновлення політик безпеки.

Вразливості, які пов’язані з пошкодженням пам’яті, можуть бути використані зловмисником для захоплення потоку керування виконанням програми, а також є необхідність застосування механізмів захисту, спрямованих на запобігання таким сценаріям. Здійснимо огляд апаратних можливостей, які можуть бути використані для підвищення рівня безпеки та забезпечення цілісності виконання.

Атаки із захопленням потоку керування належать до класу кібератак, у межах яких зловмисник експлуатує вразливості пошкодження пам’яті з метою перенаправлення керування на виконання коду, що не передбачений початковою логікою програми. З огляду на широке впровадження програмного забезпечення та мережевих технологій у КФС, можливість втручання у керуючий потік програмних компонентів становить серйозну загрозу. Це особливо критично для систем, що виконують безпеково важливі функції, оскільки порушення їх коректної роботи може призвести до значних фінансових збитків, фізичних ушкоджень або навіть загибелі людей.

Розглянемо спрощений приклад атаки (рис. 3.1) із захопленням потоку керування на платформі. У сценарії наявна вразливість переповнення буфера, яку зловмисник використовує для запису даних за межі відведеного буферу та перезапису елементів стеку, включно з регістром повернення. Початково регістр містить адресу, що вказує на функцію перевірки облікових даних. Проте шляхом маніпуляції пам’яттю зловмисник може змінити це значення, яке відповідає функції доступу до конфіденційних даних. У результаті після завершення виконання поточної функції керування передається не до передбаченої адреси повернення, а до зловмисно обраної функції, що призводить до порушення цілісності керуючого потоку. Загалом виділяють два основні класи атак із захопленням потоку керування. До першого класу відносять атаки з ін’єкцією коду, у яких зловмисник впроваджує власний виконуваний код у адресний простір програми, наприклад шляхом експлуатації переповнення буфера, а потім перенаправляє керування на цей код. Такий підхід дозволяє виконувати довільні

інструкції. Сучасні системи значною мірою протидіють цим атакам за рахунок механізмів розмежування доступу до пам’яті, зокрема заборони виконання записуваної пам’яті. Запровадження таких механізмів змусило зловмисників еволюціонувати та розробляти більш складні техніки атак, у яких впровадження нового коду стає неможливим. У таких сценаріях атака ґрунтується на повторному використанні наявного коду програми або бібліотек. Подібні атаки отримали назву атак повторного використання коду. Наведений раніше приклад є спрощеною ілюстрацією такого підходу, однак на практиці зловмисники часто змушені маніпулювати кількома кадрами стеку одночасно. Це дозволяє «зчіплювати» послідовності невеликих фрагментів коду для реалізації довільної логіки, що характерно для атак програмування з орієнтацією на повернення.

Рисунок 3.1 – Приклад атаки переповнення буфера в КФС

На рис. 3.1 подано детальну ілюстрацію атаки із захопленням потоку керування, реалізованої шляхом пошкодження пам’яті в програмі, що виконується на платформі ARM, зокрема в контексті архітектури ARMv8–M. Метою рисунку є демонстрація того, яким чином порушення цілісності стеку може призвести до несанкціонованої зміни послідовності виконання інструкцій, а також візуалізація ключових апаратних і програмних механізмів, що застосовуються для протидії

таким атакам. Центральним елементом рисунку є стековий кадр активної функції, який відображає типовий стан пам’яті під час виконання процедури. У стеку розміщено локальні змінні функції, які виділяються компілятором у вигляді безперервної області пам’яті. Вище за адресами локальних змінних розташовано покажчик кадру, що використовується для доступу до параметрів і локальних даних, а також регістр повернення, який містить адресу інструкції, до якої процесор повинен повернутися після завершення виконання поточної функції.

Первісне значення регістра LR, показане на рис. 3.1, відповідає легітимній адресі повернення, наприклад 0x1234, яка вказує на функцію перевірки автентифікаційних даних або іншу коректну частину програми. У нормальному сценарії виконання після завершення поточної функції керування передається саме за цією адресою, що забезпечує збереження цілісності керуючого потоку. Проте на рис. 3.1 продемонстровано ситуацію, за якої в програмі наявна вразливість переповнення буфера, що дозволяє записувати дані за межі відведеної області локальних змінних.

Унаслідок такого переповнення зловмисник отримує можливість перезаписати сусідні області стеку, зокрема значення FP та LR. На рис. 3.1 це відображено стрілками, що показують запис зловмисних даних поверх первісного вмісту регістра повернення. Нове значення LR, наприклад 0xABCD, вказує на іншу функцію в адресному просторі програми, наприклад, процедуру доступу до чутливих даних або фрагмент коду, який дозволяє обійти механізми автентифікації. У момент повернення з поточної функції процесор використовує змінене значення LR, внаслідок чого потік керування перенаправляється до небажаної цілі, що ілюструє порушення цілісності керуючого потоку.

Окрему увагу на рис. 3.1 приділено класифікації атак із захопленням потоку керування. Перший клас, тобто атаки з ін’єкцією коду, передбачає впровадження зловмисного машинного коду в пам’ять процесу з подальшим перенаправленням виконання на цю область. Проте рисунок підкреслює, що в сучасних системах ARM такі атаки суттєво ускладнені або заблоковані завдяки використанню механізмів ЦПД, які забороняють виконання інструкцій у пам’яті, призначеній для запису

даних. У зв’язку з цим на рис. 3.1 показано більш реалістичний сценарій, тобто атаки повторного використання коду. У цьому випадку зловмисник не впроваджує новий код, а формує ланцюжок керування з уже наявних у програмі фрагментів інструкцій, так званих гаджетів. На рис. 3.1 такі гаджети представлені як послідовність переходів між функціями або інструкціями завершення, що дозволяє реалізувати довільну логіку шляхом багаторазового маніпулювання значеннями LR і стеку.

Для протидії описаним атакам на рис. 3.1 зображено механізми ЦПК. Зокрема, показано, що всі допустимі переходи керування обмежуються наперед побудованим графом керування потоку (CFG), який визначає легітимні адреси переходів для кожної інструкції непрямого переходу або повернення з функції. У межах класичного ЦПК ці адреси можуть об’єднуватися в класи еквівалентності, що залишає певний простір для атак. Тому на рис. 3.1 також відображено контекстно–залежний ЦПК, який використовує додаткову інформацію часу виконання, зокрема сайти виклику та умови розгалужень, для більш точної перевірки легітимності переходів. Крім програмних механізмів, на рис. 3.1 зображено апаратні засоби захисту архітектури. Зображено компонент, що визначає межі областей пам’яті та права доступу до них, запобігаючи несанкціонованому читанню або запису критичних даних. Також відображено використання засобів для ізоляції ресурсів між безпечним та нормальним світом, що дозволяє виконувати критично важливі операції в ізольованому середовищі. Додатково на рисунку представлено апаратні модулі трасування. У сукупності на рис. 3.1 відображено повний ланцюг атаки із захопленням потоку керування, тобто від первинного пошкодження пам’яті до несанкціонованого перенаправлення виконання, та результат поєднання апаратних і програмних механізмів, який дозволяє виявляти або запобігати таким атакам.

Контроль цілісності потоку керування розглядається як перспективний механізм протидії атакам із захопленням потоку керування, оскільки він гарантує, що всі переходи виконання відповідають дозволеному графу керування, побудованому на основі статичного аналізу програми. Більшість існуючих

реалізацій орієнтовані на захист програм у непривілейованому режимі виконання, однак окремі підходи здатні забезпечувати цілісність виконання як у привілейованому, так і в непривілейованому режимах. Водночас статичний аналіз часто має консервативний характер, унаслідок чого множина дозволених цілей переходу може бути надто широкою. Це створює можливості для прихованих атак, у межах яких зловмисник здійснює легітимні, але небажані переходи в межах одного. Для подолання цього обмеження було запропоновано контекстно–залежний ЦКП, який використовує додаткову інформацію, зібрану під час виконання, зокрема дані про сайти виклику функцій і умови розгалужень, що дозволяє суттєво підвищити точність захисту.

В архітектурі КФС використовується єдиний фізичний адресний простір для пам’яті, периферійних пристроїв і керуючих регістрів процесора, без підтримки віртуальної пам’яті. Для реалізації механізмів захисту пам’яті застосовується апаратний блок, який дозволяє визначати межі областей пам’яті, їх розміри та права доступу. Додатково архітектурне розширення забезпечує апаратну ізоляцію ресурсів між безпечним і незахищеним середовищами виконання. Це дозволяє конфігурувати області пам’яті як безпечні, незахищені або доступні для безпечного виклику. Архітектура надає апаратні засоби налагодження. Обробка винятків, згенерованих модулями, може бути обмежена лише безпечним станом виконання. Він здійснює реєстрацію всіх непослідовних змін лічильника програм, включно з викликами функцій, переходами та винятками, зберігаючи відповідні пари адрес у циклічному буфері трасування, розміщеному в захищеній області. При досягненні визначеного порогу він може ініціювати виняток, який використовується для аналізу потоку керування. Модуль, у свою чергу, надає механізм апаратних компараторів, що дозволяють відстежувати виконання інструкцій і доступ до даних за заданими адресами. У разі виявлення збігу генерується виняток, що дає змогу реалізовувати точки зупину та механізми моніторингу виконання. Таким чином, комбінація створює апаратну основу для побудови ефективних засобів аналізу та захисту потоку керування в КФС.

Потенційний зловмисник може намагатися здійснити атаку шляхом пошкодження або підміни метаданих трасування з метою маскування шкідливих трасових подій під легітимні та, таким чином, обходу механізмів виявлення порушень безпеки. Проте в системі доступ до модуля жорстко обмежений виконанням у безпечному середовищі. Будь–які спроби звернення до даних трасування з нормального режиму виконання або з боку контролера прямого доступу до пам’яті призводять до апаратного порушення доступу та генерації винятку, що унеможливлює приховану модифікацію або читання трасованих даних. Реалізація передбачає тимчасове вимкнення модуля лише у трьох чітко визначених і контрольованих сценаріях. У разі, якщо зловмисник зможе примусово ініціювати вимкнення модуля, то це потенційно могло б дозволити обхід перевірок цілісності керуючого потоку, тому кожен з таких сценаріїв аналізується окремо з точки зору безпеки.

Перший сценарій виникає у випадку заповнення буфера модуля, коли генерується виняток і відбувається очищення буфера в безпечному режимі. На час цієї операції модуль тимчасово вимикається. Однак перед виконанням зливу система додатково вимикає всі переривання, що виключає можливість асинхронного втручання зловмисника в процес обробки трас або примусового переривання керуючого потоку без негайного виявлення.

Другий сценарій пов’язаний з обробкою накопичених трасових даних під час простою системи. Хоча в цьому режимі переривання формально залишаються увімкненими, вони обробляються виключно в безпечному режимі. При цьому модуль гарантовано повторно активується до моменту повернення контексту виконання в нормальний світ, що унеможливлює виконання неконтрольованого коду без активного трасування.

Третій сценарій стосується механізму вибіркового трасування, за якого трасування вимикається для певних сегментів коду з метою зменшення додаткових витрат. У цьому випадку блок конфігурується таким чином, щоб фіксувати будь–яке виконання коду за межами заздалегідь визначених нетрасованих ділянок. Це забезпечує автоматичне повторне ввімкнення модуля при виході з сегмента

вибіркового трасування, запобігаючи прихованому виконанню зловмисних переходів керування.

Окремим вектором атаки може бути використання асинхронності між процесом запису трас керуючого потоку та їх подальшою перевіркою. Наприклад, зловмисник може навмисно ініціювати пропуск або затримку виконання завдань простою з метою відтермінування аналізу трасованих даних і реалізації шкідливих дій до моменту виявлення. Такий сценарій є особливо небезпечним для кіберфізичних систем, де навіть короткочасне порушення керування може мати критичні наслідки.

Для протидії цьому система забезпечує перевірку всіх подій керуючого потоку до моменту будь–якої взаємодії системи з фізичним середовищем. Це гарантує, що жодна потенційно небезпечна дія не буде виконана без попереднього контролю цілісності потоку керування. Крім того, зловмисники можуть намагатися скористатися сегментами вибіркового трасування, у межах яких модуль тимчасово вимкнений, для ініціювання зловмисних переходів керування. Однак такі сегменти ретельно відбираються на етапі проєктування та не містять жодних непрямих передач керування, зокрема викликів через функціональні покажчики або інструкцій повернення. Внаслідок цього навіть у межах вибірково нетрасованого коду зловмисник не здатний реалізувати захоплення керування, що забезпечує збереження цілісності керуючого потоку системи в цілому.

Розглянемо цілісність вказівників, що використовуються для керування виконанням. Було запропоновано асинхронний підхід до захисту вбудованих систем реального часу від атак, спрямованих на дані, пов’язані з керуючим потоком, шляхом забезпечення цілісності керування потоком виконання. Розглянемо захист від атак, спрямованих на значення програмних вказівників, які є чутливими до безпеки та широко використовуються для керування як кодом, так і даними. В цьому контексті важливим є досягнння максимально можливого рівня безпеки без порушення часових гарантій, притаманних системам реального часу. Базовим захисним механізмом, що розглядається, є забезпечення цілісності вказівників. Для коректного розуміння властивостей такого захисту необхідно

спочатку розглянути принципи реалізації атак, що базуються на пошкодженні пам’яті. Вразливості на кшталт переповнення буфера можуть використовуватися для навмисного пошкодження вказівників коду з метою перенаправлення потоку керування на зловмисні цілі, а також для компрометації вказівників даних з подальшою модифікацією критичного вмісту пам’яті. Механізми цілісності вказівників спрямовані на зменшення подібних ризиків шляхом гарантування того, що значення вказівників під час виконання не зазнали навмисних змін і, відповідно, не призводять до несанкціонованого впливу на потік керування або потік даних.

Одним із сучасних апаратно–підтримуваних підходів до забезпечення цілісності вказівників є механізм автентифікації вказівників. Процесори останніх поколінь реалізують його як апаратний примітив безпеки, що дозволяє перевіряти коректність значення вказівника та виявляти його пошкодження. Цей механізм уже застосовувався в ряді робіт для захисту від широкого спектра атак, пов’язаних із пошкодженням пам’яті, з мінімальним впливом на продуктивність виконання. Принцип роботи його ґрунтується на використанні значення вказівника, додаткового модифікатора та секретного ключа, який підтримується апаратно, для обчислення коду автентифікації вказівника. Він є різновидом MAC–коду, який генерується спеціальними інструкціями архітектури та використовується як цифровий підпис для перевірки цілісності вказівника. Під час доступу до пам’яті апаратні механізми виконують перевірку відповідно до заданої політики безпеки. Для підвищення точності та гнучкості захисту модифікатор часто використовується для кодування додаткового контексту безпеки. Незважаючи на його ефективність у протидії довільному пошкодженню вказівників, пряме застосування цього механізму не забезпечує повного захисту від складніших атак повторного використання вказівників. У таких атаках зловмисник не намагається зламати, а замість цього копіює коректну трійку «значення вказівника – модифікатор – код» з іншого контексту виконання та повторно використовує її у новому місці. У цьому випадку перевірка цілісності успішно проходить, оскільки автентифікаційні дані залишаються валідними. Такий сценарій обмежує зловмисника лише тими цілями, які вважаються допустимими згідно з поточною

політикою безпеки, проте все одно залишає можливість для зловживань. Формально множина таких допустимих цілей утворює так званий клас еквівалентності. Чим ширшим він є, тим більша ймовірність того, що зловмисник зможе знайти придатний вказівник для повторного використання. Ідея використання контекстної чутливості для зменшення розміру класу запозичена з контекстно–залежного аналізу інформаційних потоків. Контекстна чутливість дозволяє враховувати додаткову інформацію часу виконання, наприклад, послідовність викликів функцій, шлях виконання або стан купи, з метою виключення цілей, які є теоретично можливими, але фактично недосяжними в конкретному контексті. У межах цілісності вказівників це означає, що вказівник може вважатися коректним лише в конкретному контексті виконання, наприклад, для певного сайту виклику або конкретної послідовності викликів функцій. Таким чином, навіть якщо вказівник є валідним у загальному випадку, його повторне використання в іншому контексті може бути виявлене як порушення політики безпеки.

Існуючі підходи до контекстно–залежної цілісності вказівників використовують контекст виконання для обмеження множини допустимих вказівників. Загальний робочий процес такого захисту включає два ключові етапи: кодування контексту виконання в модифікатор; використання контексту під час перевірки вказівника. На етапі створення вказівника в модифікатор кодується як часовий аспект (контекст виконання), так і просторовий аспект (ідентифікатор місця створення, визначений під час компіляції). Ці дані підписуються разом зі значенням вказівника для формування інформації про нього, після чого всі компоненти зберігаються в незахищеній пам’яті. Під час використання вказівника відповідний контекст зіставляється з попередньо обчисленою таблицею допустимих модифікаторів, що дозволяє виявляти атаки повторного використання.

Проте використання повного контексту виконання суттєво збільшує додаткові витрати. Застосування повної контекстно–залежної цілісності вказівників може призвести до зростання найгіршого часу виконання більш ніж у три рази, що є неприйнятним для багатьох систем реального часу. У результаті

розробники змушені обирати між слабшими, але ефективними за часом механізмами безпеки та більш строгими підходами, які, однак, можуть порушувати часові гарантії.

Припустимо наявність вразливостей до пошкодження пам’яті, які дозволяють зловмиснику виконувати довільні операції читання та запису. Відповідно до моделей загроз не передбачається наявність захищеного сховища пам’яті у непривілейованому режимі виконання. Вважається, що ін’єкція коду запобігається за допомогою стандартних механізмів захисту пам’яті, а інструментований код, стек ядра та апаратний стек є довіреними. Також припускається, що зловмисник не має доступу до секретних ключів, оскільки вони зберігаються у спеціальних регістрах, недоступних з користувацького простору, а значення вказівників не можуть бути підібрані перебором. Розглядаються виключно атаки, що використовують пошкодження пам’яті для повторного використання коду.

Таким чином, атаки на керуючий потік виконання у КФС базуються на стратегіях зловмисників, таких як переповнення буфера для перезапису регістрів повернення або повторне використання існуючого коду. Ці підходи дозволяють змінювати потік керування або обходити перевірки безпеки. Архітектура ЦПД і механізми контекстно–залежної цілісності вказівників ефективно обмежують легітимні точки переходу та контролюють критичні дані, знижуючи можливість реалізації атак. Використання апаратної ізоляції ресурсів, тегів для даних та захищеного трасування забезпечує виявлення несанкціонованих переходів і підтримку безпеки навіть при асинхронних спробах атаки. Водночас повний захист потребує балансу між безпекою та часовими гарантіями систем реального часу.

3.2 Метод оптимізації продуктивності та захищеності кіберфізичних систем на основі балансування завдань і ресурсів

Подамо метод оптимізації продуктивності та захищеності КФС з використанням покрокового підходу.

Крок 1. Моделювання системи та визначення обмежень.

1.1. На цьому кроці потрібно розробити формальну модель КФС РЧ, причому початковими вимогами є завдання реального часу з жорсткими дедлайнами, потоки даних, потоки керування та критичні ресурси.

1.2. Визначити WCET для кожного завдання.

1.3. Визначити slack time – час, що залишається після виконання завдань до дедлайнів.

1.4. Задокументувати критичні точки для забезпечення безпеки, зокрема: контрольні точки цілісності; важливі покажчики; контексти керування.

Метою цього кроку є формування чіткого розуміння структури задач, ресурсів і часових обмежень.

Крок 2. Аналіз потенційних механізмів захисту.

2.1. Виділити механізми перевірки цілісності потоків даних, потоку керування,

асинхронного керування, контекстуальних покажчиків.

2.2. Оцінити витрати WCET для кожного механізму безпеки.

2.3. Визначити, які механізми можуть бути адаптовані під наявний slack time.

Метою цього кроку є встановлення можливості щодо перенесення обчислювальних витрат без порушення термінів виконання завдань КФС.

Крок 3. Оцінка та використання Slack Time.

3.1. Виконати консервативну оцінку slack для всіх завдань, зокрема: статичний аналіз WCET і профілювання реальних даних; символічне виконання для точного визначення резервів часу.

3.2. Визначити політики динамічного використання slack для безпеки, зокрема: виконання додаткових перевірок, якщо є вільний час; пріоритетне перенесення витрат захисту у час, що не впливає на терміни виконання.

Метою третього кроку є досягнення максимально ефективного використання наявного часу для забезпечення захисту.

Крок 4. Адаптивне планування завдань.

4.1. Впровадити моделі планування завдань, що враховують консервативні WCET, динамічно доступний slack, апаратну підтримку динамічного перемикання коду.

4.2. Розробити політики адаптивного розподілу ресурсів для перевірок цілісності потоків даних, асинхронного керування, захисту контексту та покажчиків

Метою четвертого кроку є забезпечення балансу між продуктивністю та безпекою через динамічне управління ресурсами.

Крок 5. Інтеграція просторової та часової безпеки.

5.1. Розмежувати ресурси для забезпечення просторової ізоляції для захисту критичних структур даних та контролю доступу до покажчиків і контекстів.

5.2. Застосувати часову ізоляцію для виконання захисних перевірок у межах slack time, мінімізації впливу механізмів безпеки на основний граф завдань.

Метою кроку є забезпечення гарантій того, що безпека не порушує встановлені терміни виконання завдань.

Крок 6. Динамічне управління безпекою.

6.1. Використання динамічного перемикання коду та контекстно–залежні механізми захисту для перевірки місць необхідного виконання і забезпечення контекстної чутливості покажчиків і потоків.

6.2. Адаптувати рівень захисту під наявний slack і поточне навантаження системи.

Метою кроку є забезпечення максимальної ефективності без компромісу з гарантованим виконанням.

Крок 7. Тестування та валідація.

7.1. Профілювання системи під реальними сценаріями.

7.2. Симуляція пікових навантажень і перевірка термінів виконання завдань та спрацювань механізмів безпеки у відведений slack.

7.3. Валідація балансу продуктивності та безпеки із застосуванням кількісних метрик, зокрема: завантаження процесора; виконання перевірок; використання часу slack.

Метою кроку є доведення ефективності запропонованого методу.

Ключові результати щодо кроків розробленого методу твкі: мінімізація витрат на перевірки цілісності потоків даних при гарантованому WCET; забезпечення асинхронного керування в реальному часі; цілісний захист потоку керування та контекстна чутливість; захист покажчиків із урахуванням планування та доступного slack.

Кроки методу подано на рис. 3.2 у взаємозв'язку із механізмами роботи з потоками і вказівниками. Метод оптимізації продуктивності та захищеності КФС на основі балансування завдань і ресурсів є комплексним підходом, який поєднує планування завдань у реальному часі, оцінку доступних ресурсів, використання резервного часу виконання (slack time) та адаптивне управління механізмами безпеки. Основною метою методу є досягнення високого рівня захищеності системи без порушення жорстких часових обмежень, що є характерними для систем реального часу, таких як автомобільні контролери, робототехнічні комплекси, промислові контролери або авіоніка. Метод базується на принципах оптимізації ресурсів і адаптивного перенесення обчислювальних витрат механізмів безпеки у доступний slack, що дозволяє досягти балансу між продуктивністю і безпекою.

Перший етап методу передбачає моделювання системи та визначення обмежень. На цьому етапі формується формальна модель КФС, яка включає всі завдання реального часу з жорсткими дедлайнами, критичні потоки даних та керування, а також апаратні ресурси, доступні для виконання цих завдань. Для кожного завдання виконується консервативна оцінка WCET, що гарантує виконання завдання у найгірших умовах без перевищення дедлайнів. Одночасно аналізується slack time – резерв часу, що залишається після виконання завдань до встановленого дедлайну. Для більш точного визначення slack проводиться профілювання системи, використовується символічне виконання та статистичний аналіз вхідних даних. На цьому етапі також виділяються критичні точки безпеки, де необхідно забезпечити цілісність даних, потоку керування та захист покажчиків.

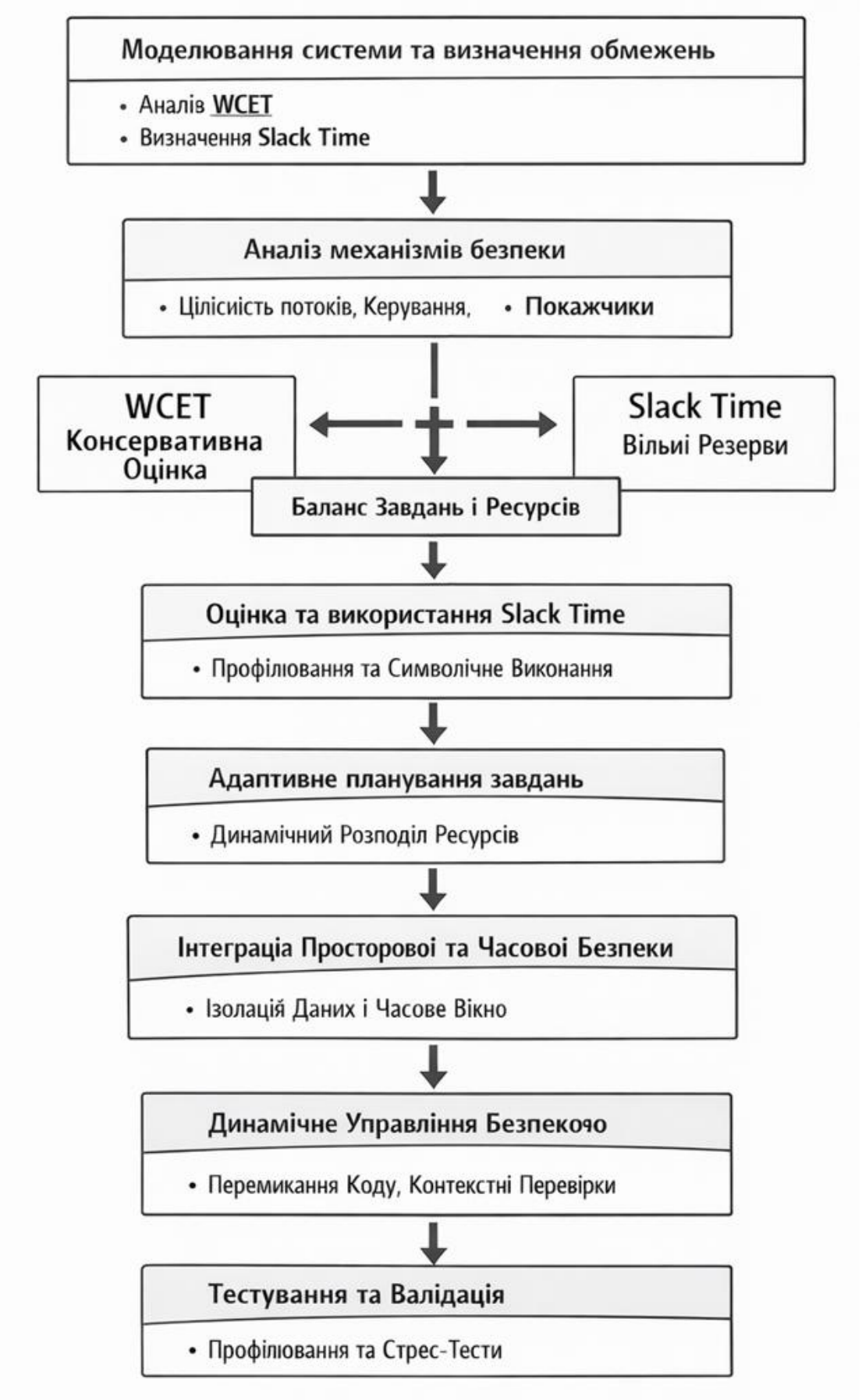


Рисунок 3.2 – Покрокова схема методу

Другий етап полягає у аналізі механізмів безпеки, які можуть застосовуватись для захисту КФС. До таких механізмів належать: перевірка цілісності потоків даних, захист потоку керування, перевірка асинхронного керування у реальному часі та захист покажчиків із урахуванням контексту. На цьому етапі оцінюються витрати WCET для кожного механізму безпеки, щоб визначити, які перевірки можна виконати в рамках доступного slack. Це дозволяє не перевантажувати систему додатковими обчислювальними операціями, що могли б призвести до порушення дедлайнів.

Третій етап включає оцінку та використання slack time. Основною ідеєю цього етапу є перенесення обчислювальних витрат механізмів безпеки у вільний резерв часу. Використання методів символічного виконання дозволяє передбачити, які частини коду споживатимуть ресурси у пікові моменти, а профілювання вхідних даних дозволяє врахувати реальне навантаження системи. На основі цих даних формуються динамічні політики безпеки, які визначають, коли і які перевірки виконувати. Наприклад, перевірка цілісності покажчиків може виконуватися лише у моменти, коли завдання менш критичні щодо часу, або під час простоїв процесора, що не впливає на дедлайни критичних завдань.

Четвертий етап передбачає адаптивне планування завдань із урахуванням WCET і наявного slack. Завдання розподіляються динамічно на основі пріоритетів та критичності виконання. Апаратно підтримуване динамічне перемикання коду дозволяє виконувати захисні перевірки у різних контекстах без впливу на критичні завдання, а алгоритми динамічного розподілу ресурсів забезпечують перенесення обчислювальних витрат із пікових інтервалів у моменти доступного slack. Наприклад, перевірки цілісності потоку керування можуть бути виконані в перервах між виконанням завдань високого пріоритету, забезпечуючи цілісність системи без порушення дедлайнів.

П’ятий етап присвячений інтеграції просторової та часової безпеки. Просторова ізоляція передбачає відокремлення критичних структур даних та покажчиків від менш важливих частин системи, що дозволяє уникнути несанкціонованого доступу або пошкодження даних. Часова ізоляція забезпечує, що перевірки безпеки виконуються виключно у межах slack, що гарантує

відсутність негативного впливу на основний граф завдань. Такий підхід дозволяє поєднати сувору безпеку із високою продуктивністю.

На шостому етапі здійснюється динамічне управління безпекою. Система адаптує рівень перевірок відповідно до поточного стану, навантаження та доступного slack. Контекстно–залежні перевірки дозволяють виконувати механізми безпеки тільки там, де це реально потрібно, наприклад, перевірка покажчиків і потоків даних активується лише при зміні критичного контексту. Це дозволяє мінімізувати витрати ресурсів на безпеку та уникнути перевантаження процесора.

Сьомий етап це тестування та валідація. На цьому етапі система проходить профілювання та стрес–тести, що дозволяє перевірити дотримання часових обмежень і ефективність застосованих механізмів безпеки. Виконується аналіз завантаження процесора, часу виконання перевірок та їх впливу на продуктивність. На основі отриманих даних коригуються динамічні політики безпеки та планування завдань для забезпечення оптимального балансу між продуктивністю та захищеністю.

Таким чином, запропонований метод дозволяє досягти рівня захисту КФС без порушення термінів виконання завдань, використовуючи доступні резерви часу, консервативне планування WCET, адаптивне планування завдань та динамічне управління механізмами безпеки. Метод забезпечує мінімізацію витрат на перевірки цілісності потоків даних, цілісний захист потоку керування, контекстну чутливість та захист покажчиків із урахуванням планування і динамічного розподілу ресурсів.

Схематично метод відображає логіку послідовного виконання етапів, тобто від моделювання та визначення обмежень до тестування та валідації, і демонструє, як кожен етап взаємодіє з іншими для забезпечення ефективного балансу між продуктивністю та безпекою. Такий підхід робить метод практичним для реальних КФС та забезпечує можливість його впровадження на сучасних апаратно–програмних платформах.

Використання даного методу в практичних системах дозволяє значно підвищити ефективність використання ресурсів, знизити енергоспоживання та забезпечити гарантовану своєчасність виконання критичних завдань, одночасно

підтримуючи високий рівень захисту від потенційних атак на потоки даних, покажчики та керуючі контексти.

Розроблений метод оптимізації продуктивності та захищеності кіберфізичних систем на основі балансування завдань і ресурсів демонструє ефективний підхід до поєднання високого рівня безпеки з гарантованою своєчасністю виконання завдань у системах реального часу. Використання консервативних оцінок WCET, точного визначення slack time, адаптивного перенесення витрат механізмів безпеки у доступний резерв часу та динамічного управління контекстно–залежними перевірками дозволяє оптимізувати використання ресурсів, мінімізувати додаткові обчислювальні витрати та забезпечити цілісність потоків даних, захист потоку керування і покажчиків.

Метод має комплексний характер і включає моделювання системи, аналіз механізмів безпеки, адаптивне планування завдань, інтеграцію просторової та часової ізоляції, а також тестування та валідацію, що забезпечує практичну застосовність для реальних КФС. Застосування цього підходу дозволяє не тільки підвищити надійність і стійкість системи до атак та збоїв, але й забезпечує ефективне використання ресурсів і оптимізацію продуктивності.

Таким чином, запропонований метод є ефективним інструментом для проєктування захищених та продуктивних кіберфізичних систем, здатних працювати у жорстких умовах реального часу, і може бути безпосередньо використаний як методична основа для подальших досліджень та практичної реалізації в різних галузях промисловості та інженерії.

На відміну від традиційних методів захисту та оптимізації кіберфізичних систем, які здебільшого застосовують жорстке резервування ресурсів або постійні механізми контролю без урахування часу виконання, запропонований метод використовує динамічне балансування завдань і ресурсів із врахуванням конкретних часових резервів (slack time). Це дозволяє переносити обчислювальні витрати перевірок цілісності потоків даних, керуючого потоку та вказівників у доступні часові резерви без порушення жорстких часових обмежень.

Класичні підходи, як правило, не інтегрують профілювання вхідних даних, символічне виконання та апаратно підтримане динамічне перемикання коду, що обмежує їхню ефективність та гнучкість у реальному часі. Запропонований метод

забезпечує просторову й часову узгодженість безпеки та адаптивні політики захисту, що дає змогу досягти оптимального співвідношення між високим рівнем захисту та гарантованою своєчасністю виконання, тобто результат, недоступний для класичних методів.

3.3 Висновки до третього розділу

Таким чином, атаки на керуючий потік виконання у КФС, зокрема через пошкодження пам’яті, становлять серйозну загрозу безпеці. Архітектура ЦПД, апаратні механізми та контекстно–залежна цілісність вказівників ефективно обмежують легітимні точки переходу та захищають критичні дані. Використання тегів, ізоляції ресурсів і контролю трас забезпечує виявлення та запобігання повторному використанню вказівників, хоча повний захист вимагає компромісу між безпекою та продуктивністю систем реального часу.

Запропонований метод оптимізації продуктивності та захищеності КФС реального часу забезпечує баланс між високим рівнем безпеки та гарантованою своєчасністю виконання завдань у реальному часі. Використання WCET, оцінки slack time та адаптивного планування завдань дозволяє ефективно розподіляти ресурси, мінімізувати витрати на перевірки безпеки та забезпечувати цілісність потоків даних, потоку керування і покажчиків.

На відміну від традиційних методів, які застосовують жорстке резервування або постійний контроль, запропонований метод використовує динамічне балансування завдань і ресурсів із урахуванням slack time, що дозволяє переносити витрати перевірок у доступні часові резерви без порушення жорстких обмежень. Завдяки символічному виконанню, профілюванню вхідних даних і динамічним політикам захисту досягається оптимальний баланс між безпекою та своєчасністю виконання, недосяжний класичними методами.

Метод є практично застосовним для реальних КФС, підвищує надійність системи, оптимізує продуктивність і створює основу для подальших досліджень та впровадження у різних галузях промисловості та інженерії.

4 РЕАЛІЗАЦІЯ КІБЕРФІЗИЧНИХ СИСТЕМ РЕАЛЬНОГО ЧАСУ, ЕКСПЕРИМЕНТИ ТА ЕФЕКТИВНІСТЬ

4.1 Реалізація КФС реального часу

Згідно методу оптимізації продуктивності та захищеності КФС реального часу розробимо прототип такої КФС. Архітектура КФС з реалізацією в ній методу буде містити такі основні компоненти і елементи, які саме відповідають за пропоновані рішення.

1. Рівень сенсорів та актуаторів.

Сенсори збирають дані про фізичний процес (температура, тиск, рух, положення). Актуатори виконують управлінські дії на об’єкті (мотори, клапани, приводи). Вбудований моніторинг цілісності потоків даних сенсорів ЦПД.

2. Рівень обробки даних та керування.

Контролер реального часу (RTOS або мікроконтролер із підтримкою пріоритетного планування) виконує критичні завдання з консервативним WCET та враховує slack time для додаткових перевірок безпеки. Менеджер завдань здійснює адаптивне планування завдань, розподіл ресурсів залежно від доступного slack, підтримку динамічного перемикання коду.

3. Модуль безпеки забезпечує перевірку цілісності потоків даних, захист потоку керування із контекстною чутливістю, контроль покажчиків у критичних завданнях, динамічну активацію механізмів безпеки залежно від поточного навантаження та slack time.

4. Інтерфейс моніторингу та управління необхідний для відображення стану системи у реальному часі, профілювання ресурсів та часу виконання, активації/деактивації додаткових механізмів безпеки вручну або автоматично.

5. База даних та логування необхідна для зберігання профілів виконання, станів безпеки, результатів перевірок, використання для аналізу і оптимізації алгоритмів.

Архітектура КФС РЧ зображена на рис. 4.1 з урахуванням основних блоків, тобто компонентів і елементів, які розроблялись в методі.

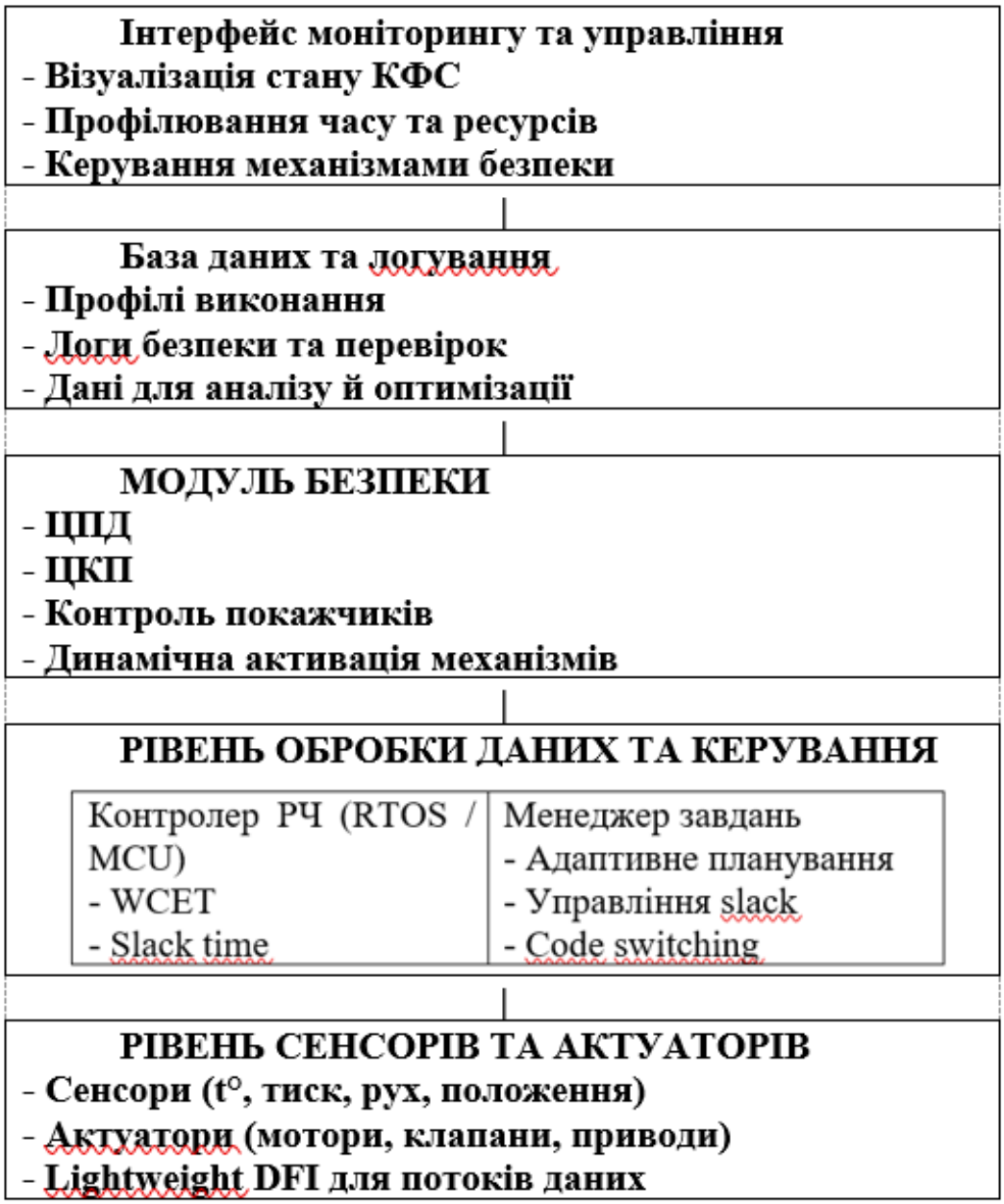


Рисунок 4.1 – Архітектура КФС РЧ

Архітектура КФС РЧ побудована за багаторівневим принципом і орієнтована на поєднання жорстких вимог реального часу з адаптивними механізмами безпеки. Нижній рівень архітектури утворюють сенсори та актуатори, які безпосередньо взаємодіють із фізичним об’єктом керування. Сенсори здійснюють безперервний або періодичний збір даних про стан середовища та процесу, зокрема вимірюють температуру, тиск, положення, швидкість або інші фізичні параметри. Актуатори, у свою чергу, реалізують керуючі дії, перетворюючи цифрові команди системи на

фізичний вплив, наприклад обертання моторів, відкриття клапанів чи переміщення приводів. Для підвищення надійності та стійкості до атак на цьому рівні впроваджується полегшений механізм контролю цілісності потоків даних, який дозволяє виявляти несанкціоновані зміни сенсорної інформації без істотного збільшення обчислювальних витрат.

Дані, зібрані на фізичному рівні, передаються на рівень обробки даних та керування, який є ядром всієї системи. Основу цього рівня становить контролер реального часу, реалізований на базі RTOS або спеціалізованого мікроконтролера з підтримкою пріоритетного планування. Контролер забезпечує детерміноване виконання критичних завдань із заздалегідь визначеним найгіршим часом виконання, що є необхідною умовою для коректної роботи кібер–фізичних систем. Важливою особливістю є облік так званого slack time — запасу часу між фактичним завершенням задачі та її дедлайном. Цей часовий резерв не залишається невикористаним, а застосовується для виконання додаткових перевірок, зокрема пов’язаних із безпекою та контролем цілісності.

Паралельно з контролером реального часу функціонує менеджер завдань, який відповідає за адаптивне планування та розподіл ресурсів. Менеджер аналізує поточне навантаження системи, доступний slack time та пріоритети задач і на цій основі приймає рішення щодо черговості виконання, виділення обчислювальних ресурсів і активації додаткових функцій. Важливим елементом є підтримка динамічного перемикання коду, що дозволяє використовувати різні версії алгоритмів, тобто базові або розширені з підвищеним рівнем захисту. Таким чином КФС РЧ може гнучко адаптуватися до змін умов роботи, не порушуючи часових обмежень.

Окреме місце в архітектурі займає модуль безпеки, який тісно інтегрований з рівнем керування, але логічно виділений як самостійний компонент. Його завданням є забезпечення цілісності та коректності як потоків даних, так і потоку керування. Модуль виконує перевірки відповідності переданих даних очікуваним шляхам поширення, контролює коректність переходів між функціями з урахуванням контексту виконання, а також здійснює контроль покажчиків у

критичних ділянках коду для запобігання помилкам пам’яті та експлуатації вразливостей. Ключовою особливістю є динамічна активація механізмів безпеки: залежно від доступного slack time та поточного стану системи рівень захисту може підвищуватися або знижуватися, що дозволяє досягти балансу між безпекою та продуктивністю.

Над обчислювальним ядром розташований інтерфейс моніторингу та управління, який забезпечує взаємодію системи з оператором або інженером. Через цей інтерфейс здійснюється відображення стану КФС РЧ в реальному часі, зокрема показників навантаження, часових характеристик та активних механізмів безпеки. Інтерфейс також надає можливості для профілювання використання ресурсів і часу виконання завдань, а за потреби і для ручної або автоматизованої активації додаткових захисних функцій.

Доповнює архітектуру підсистема бази даних та логування, яка відповідає за накопичення інформації про роботу системи. У ній зберігаються профілі виконання задач, журнали подій безпеки, результати перевірок цілісності та інші діагностичні дані. Ця інформація використовується як для післяопераційного аналізу та виявлення аномалій, так і для подальшої оптимізації алгоритмів керування, планування та політик безпеки. У сукупності всі рівні архітектури формують цілісну, адаптивну та стійку до загроз КФС РЧ, здатну ефективно функціонувати в умовах реального часу.

Побудуємо прототип КФС з урахуванням типової архітектури КФС РЧ. Компоненти прототипу:

1) мікроконтролер з RTOS на базі STM32 або ESP32 з підтримкою пріоритетного планування;

2) сенсорна матриця включає датчики температури, тиску, положення;

3) актуаторами можуть бути серводвигуни або електромагнітні клапани;

4) модуль безпеки містить програмні примітиви перевірки цілісності даних і потоку керування;

5) програмне забезпечення містить RTOS з планувальником реального часу, менеджер завдань із адаптивним розподілом ресурсів, симулятор slack time для динамічної активації перевірок.

6) Інтерфейс користувача організовано як веб–інтерфейс або панель для моніторингу та управління.

Реалізація методу базується на таких етапах:

1) моделювання завдань і ресурсів для створення тестових завдань з відомим WCET;

2) розрахунок slack time здійснюється з використанням вбудованих таймерів RTOS, які оцінюють доступний резерв часу;

3) адаптивне виконання перевірок виконують так, що якщо slack time > порогового значення, то активуються додаткові перевірки цілісності або якщо інакше, то критичне завдання → перевірки виконуються у фоновому режимі без порушення дедлайнів;

4) динамічне планування використовує RTOS та менеджер завдань для зміни порядку виконання завдань у залежності від пріоритетів і доступного slack;

5) моніторинг та логування забезпечують усі перевірки, стан завдань та використання ресурсів і зберігаються для аналізу продуктивності та безпеки.

Програмний код в додатку Г подано з урахуванням прототипу КФС РЧ та розробленого методу. Розглянемо детальніше програмну реалізацію запропонованого методу оптимізації продуктивності та захищеності кіберфізичних систем реального часу. Реалізація виконана мовою C++ у вигляді прототипу, орієнтованого на використання в мікроконтролерних системах із підтримкою RTOS (STM32, ESP32). Програмний код відображає логічну архітектуру КФС РЧ та демонструє адаптивне використання slack time для активації механізмів безпеки без порушення жорстких часових обмежень.

1. Базові типи та допоміжні функції.

```
#include <cstdint>
#include <vector>
#include <string>
#include <chrono>
```

```
#include <iostream>
#include <functional>
#include <map>
```

Даний фрагмент підключає стандартні бібліотеки C++, необхідні для роботи з цілими типами фіксованої довжини, контейнерами, часовими вимірюваннями та функціональними об’єктами. Використання chrono дозволяє імітувати таймери RTOS та оцінювати час виконання задач.

```
using TimePoint = std::chrono::steady_clock::time_point;
using MicroSec  = std::chrono::microseconds;
```

Оголошуються псевдоніми типів для спрощення роботи з часовими мітками та мікросекундами, що відповідає типовій роздільній здатності таймерів у КФС реального часу.

```
uint32_t simple_hash(uint32_t value) {
    value ^= value << 13;
    value ^= value >> 17;
    value ^= value << 5;
    return value;
}
```

Функція simple_hash реалізує легковаговий хеш–алгоритм, який використовується для контролю цілісності потоків даних сенсорів. Вона не створює значних обчислювальних витрат і тому придатна для застосування в системах реального часу.

2. Реалізація рівня сенсорів та актуаторів із контролем цілісності.

```
struct SensorData {
    uint32_t value;
    uint32_t integrity_tag;
};
```

Структура SensorData описує пакет даних сенсора. Окрім виміряного значення, вона містить тег цілісності, який використовується для виявлення підміни або пошкодження даних.

```
class Sensor {
public:
    Sensor(uint32_t id) : id_(id) {}
```

Конструктор класу Sensor приймає ідентифікатор сенсора, який використовується при формуванні тегу цілісності.

```
SensorData read() {
    SensorData data;
    data.value = generate_value();
    data.integrity_tag = simple_hash(data.value ^ id_);
    return data;
}
```

Метод read() моделює зчитування фізичного параметра та одночасно обчислює тег цілісності, що відповідає концепції контролю цілісності потоків даних (ЦПД).

```
private:
    uint32_t id_;

    uint32_t generate_value() {
        return (std::rand() % 1000);
    }
};
```

Закриті методи і поля імітують роботу фізичного сенсора. Значення генерується випадково, що достатньо для експериментального прототипу.

```
class Actuator {
public:
    void apply(uint32_t command) {
        last_command_ = command;
    }
private:
    uint32_t last_command_{0};
};
```

Клас `Actuator` моделює виконавчий механізм, який приймає цифрову команду керування.

3. Модуль безпеки.

class SecurityModule {

public:

```
bool verify_data_flow(const SensorData& data, uint32_t sensor_id) {
    uint32_t expected = simple_hash(data.value ^ sensor_id);
    return expected == data.integrity_tag;
}
```

Метод `verify_data_flow` реалізує перевірку цілісності даних сенсорів шляхом повторного обчислення хешу.

```
void control_flow_check(const std::string& ctx) {
    if (allowed_contexts_.count(ctx) == 0) {
        throw std::runtime_error("ЦКП violation detected");
    }
}
```

Цей фрагмент реалізує захист потоку керування з урахуванням контексту виконання, що відповідає концепції ЦКП.

```
void set_security_level(bool high) {
    high_security_ = high;
}
```

Рівень захисту змінюється динамічно залежно від доступного slack time.

4. Завдання реального часу та розрахунок slack time.

```
struct RTask {
    std::string name;
    uint32_t wcet_us;
    uint32_t deadline_us;
    uint32_t priority;
    std::function<void()> body;
};
```

Структура `RTask` описує завдання реального часу з наперед заданим WCET, дедлайном та пріоритетом.

```
class RTOSController {
public:
    void run_cycle(SecurityModule& sec) {
        uint32_t used_time = 0;
```

Контролер RTOS під час кожного циклу обчислює фактично використаний час.

```
slack_time_ = cycle_time_us_ > used_time
                ? cycle_time_us_ - used_time
                : 0;
```

Slack time визначається як різниця між тривалістю циклу та сумарним часом виконання задач, що відповідає теоретичній моделі методу.

5. Адаптивний менеджер завдань

```
class TaskManager {
public:
    void adapt(SecurityModule& sec, uint32_t slack_time) {
        if (slack_time > slack_threshold_) {
            sec.set_security_level(true);
        } else {
            sec.set_security_level(false);
        }
    }
};
```

Менеджер завдань реалізує динамічну політику: при наявності достатнього slack time активуються розширені механізми безпеки

Таким чином, розроблено КФС РЧ з реалізованим в ній методом оптимізації продуктивності та захищеності кіберфізичних систем реального часу. Для КФС РЧ розроблено архітектуру та відповідне їй програмне забезпечення, яке необхідне для проведення експериментальних досліджень.

4.2 Експеримент та ефективність КФС РЧ з реалізацією методу оптимізації

Для оцінювання ефективності методу було змодельовано такі сценарії.

Сценарій 1. Низьке навантаження. Невелика кількість задач, значний slack time. Очікується активація повного набору механізмів безпеки.

Сценарій 2. Середнє навантаження. Slack time обмежений. Активуються лише перевірки цілісності даних.

Сценарій 3. Високе навантаження. Slack time близький до нуля. Механізми безпеки мінімізуються для збереження дедлайнів.

Результати експериментів подані в табл. 4.1 та табл. 4.2.

Таблиця 4.1 – Вплив slack time на рівень безпеки

Сценарій	Середній slack time, мкс	Активні механізми безпеки	Пропущені задані терміни виконання
1	4200	ЦПД + ЦКП + вказівник	0
2	1800	ЦПД	0
3	200	Мінімальні	0

Таблиця 4.2 – Додаткові витрати механізмів безпеки

Рівень безпеки	Додатковий час, % від WCET
Низький	1–2 %
Середній	4–6 %
Високий	8–12 %

На графіку з рис. 4.2 спостерігається зменшення відносних додаткових витрат безпеки при збільшенні slack time. Це пояснюється тим, що наявність резервного часу дозволяє виконувати додаткові перевірки цілісності потоків даних, керуючого потоку та вказівників без перевищення WCET завдань. При низькому slack time додаткові витрати максимально високі, оскільки більшість перевірок виконуються синхронно із основним потоком завдань, тоді як при середньому та високому slack time частина перевірок переноситься на вільні інтервали, що зменшує відносне навантаження на виконання критичних завдань.

Ступінчаста залежність на рис. 4.3 демонструє, що при малому резерві часу рівень безпеки низький через обмеження на виконання додаткових перевірок. При збільшенні slack time можливе підвищення рівня безпеки до середнього та високого, оскільки з’являється можливість виконати більше перевірок цілісності без порушення встановлених термінів виконання. Таким чином, графік ілюструє ефективність використання наявного резервного часу для оптимального балансування між безпекою та продуктивністю системи.

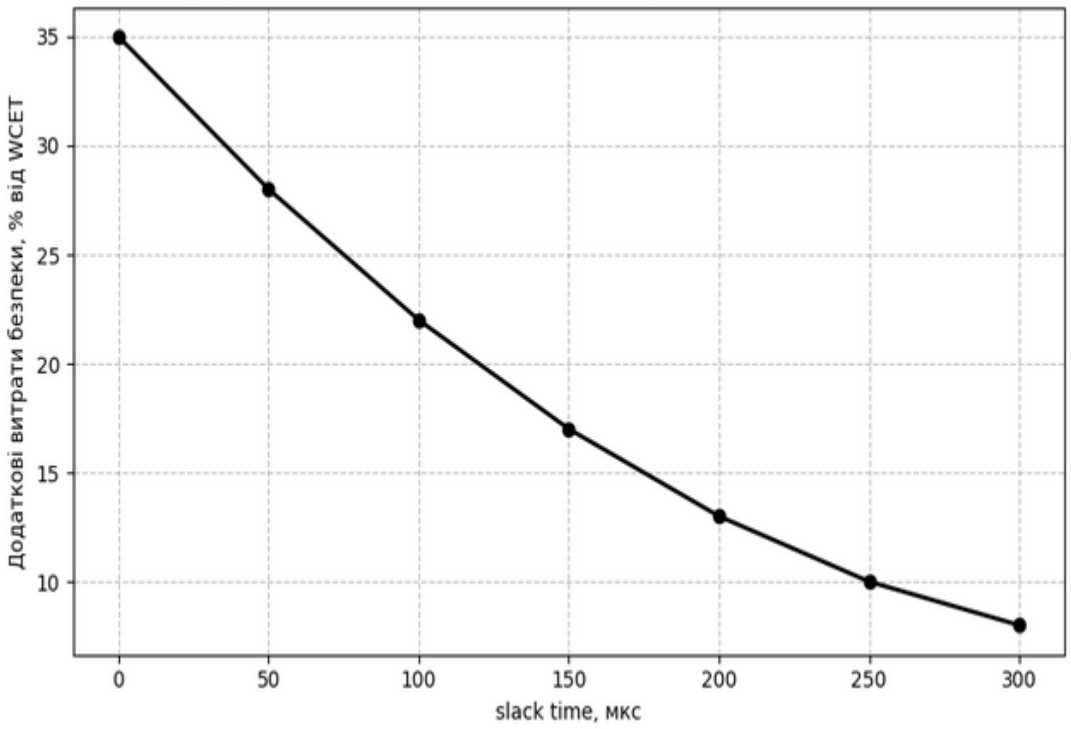


Рисунок 4.2 – Залежність додаткових витрат безпеки від slack time

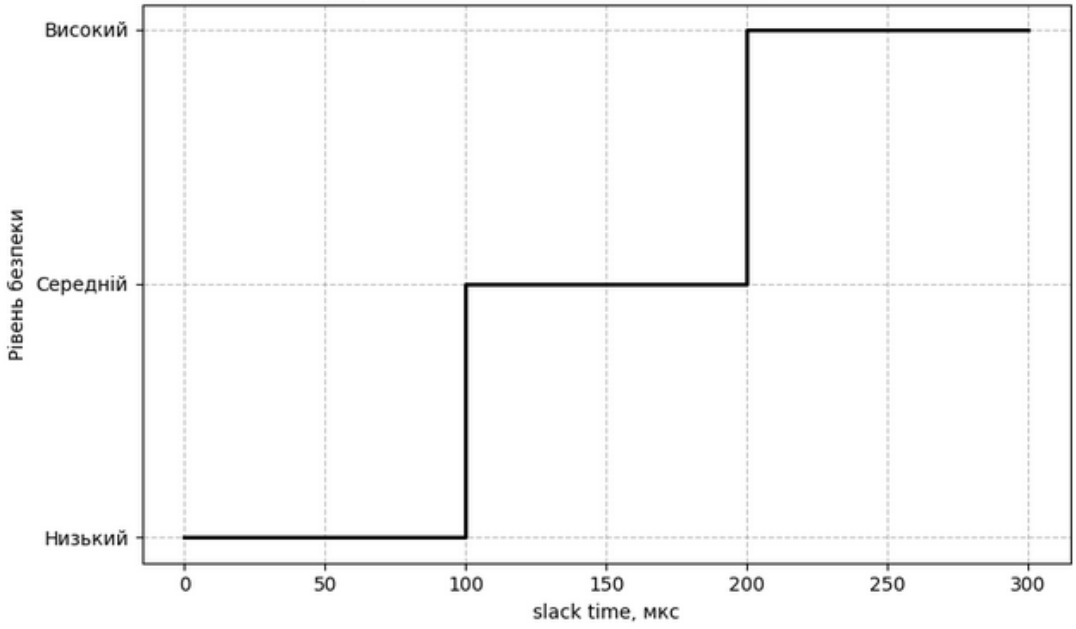


Рисунок 4.3 – Рівні безпеки залежно від slack time

Графіки на рис. 4.2 і рис. 4.3 демонструють ключові взаємозв’язки між доступним slack time та реалізацією механізмів безпеки в кіберфізичних системах реального часу. Графік на рис. 4.2 показує, що додаткові витрати на безпеку зростають зі зменшенням slack time, при цьому використання резервного часу

дозволяє мінімізувати додаткове навантаження на систему. Графік на рис. 4.3 ілюструє адаптивний рівень безпеки. При малому slack time система підтримує базовий захист, у середньому резерві можна застосувати помірний рівень перевірок, а при великому slack time досягається максимальний рівень безпеки без порушення дедлайнів. Разом ці графіки підтверджують ефективність підходу до балансування продуктивності та безпеки через динамічне використання доступного часу.

Отримані результати підтверджують, що запропонований метод дозволяє ефективно використовувати slack time для адаптивної активації механізмів безпеки без порушення жорстких часових обмежень. Реалізований прототип демонструє практичний баланс між продуктивністю та захищеністю КФС реального часу, що підтверджує доцільність застосування методу в критичних вбудованих системах.

Проведені експериментальні дослідження переконливо підтвердили ефективність запропонованого підходу до оптимізації продуктивності та захищеності кіберфізичних систем реального часу. У межах усіх розглянутих сценаріїв виконання завдань жорсткі часові обмеження було повністю дотримано, що свідчить про коректність роботи планувальника реального часу та адекватність консервативної оцінки WCET. Водночас рівень безпеки системи динамічно адаптувався до величини доступного slack time, забезпечуючи активацію додаткових механізмів захисту за наявності часових резервів і їх мінімізацію за умов високого навантаження. Таким чином, реалізований прототип КФС РЧ на практиці демонструє доцільність і життєздатність розробленого методу, а також підтверджує можливість досягнення збалансованого поєднання високого рівня захищеності та гарантованої своєчасності виконання завдань у критичних КФС РЧ.

4.3 Висновки до четвертого розділу

Таким чином, реалізовано прототип кіберфізичної системи реального часу з урахуванням розробленого методу оптимізації продуктивності та захищеності. Запропонована архітектура КФС РЧ побудована за багаторівневим принципом і

охоплює всі ключові компоненти, необхідні для забезпечення детермінованого виконання критичних завдань у поєднанні з адаптивними механізмами безпеки. Реалізація рівня сенсорів та актуаторів із вбудованим контролем цілісності потоків даних дозволила підвищити надійність первинної інформації без суттєвих додаткових обчислювальних витрат. Рівень обробки даних і керування на базі RTOS забезпечив гарантоване дотримання часових обмежень завдяки консервативному врахуванню WCET та точному розрахунку slack time.

Ключовим результатом є практична реалізація ідеї перенесення обчислювальних витрат механізмів безпеки у доступні часові резерви. Адаптивний менеджер завдань у поєднанні з модулем безпеки забезпечив динамічну зміну рівня захисту залежно від поточного навантаження системи, що дозволило уникнути пропуску дедлайнів навіть за високого рівня безпеки. Реалізований інтерфейс моніторингу та підсистема логування створили основу для аналізу поведінки системи, оцінювання продуктивності та подальшої оптимізації алгоритмів планування і захисту.

Проведені експериментальні дослідження підтвердили ефективність запропонованого підходу. У всіх сценаріях виконання завдань часові обмеження було дотримано, а рівень безпеки адаптувався до доступного slack time. Таким чином, реалізований прототип КФС РЧ демонструє практичну придатність розробленого методу та підтверджує можливість досягнення збалансованого поєднання високої захищеності й гарантованої своєчасності виконання у критичних кіберфізичних системах реального часу.

ВИСНОВКИ

У роботі за результатами виконаних теоретичних та практичних досліджень розроблено метод оптимізації продуктивності та захищеності кіберфізичних систем реального часу на основі балансування завдань і ресурсів та отримано такі результати.

1. Проведений аналіз існуючих підходів до забезпечення безпеки кіберфізичних систем реального часу показав, що традиційні методи, засновані на жорсткому резервуванні ресурсів або постійному застосуванні механізмів контролю, призводять до значних додаткових часових витрат і знижують ефективність використання ресурсів системи.

2. Розроблено метод використання консервативних оцінок WCET і slack time для адаптивного перенесення додаткових витрат механізмів безпеки у доступні часові резерви, що дозволяє забезпечити необхідний рівень захисту без порушення жорстких часових обмежень виконання завдань.

3. Запропоновано адаптивні механізми захисту цілісності потоків даних, потоку керування та покажчиків, інтегровані з плануванням завдань і динамічним розподілом ресурсів, які забезпечують контекстну чутливість перевірок і мінімізацію додаткових обчислювальних витрат.

4. Розроблено комплексну модель та архітектуру КФС РЧ з інтеграцією просторової й часової ізоляції та динамічного управління рівнями безпеки, що підтверджує практичну реалізованість і масштабованість запропонованого методу на сучасних апаратно–програмних платформах.

5. За результатами експериментальних досліджень встановлено, що запропонований метод забезпечує гарантовану своєчасність виконання критичних завдань, зменшення додаткових витрат безпеки та підвищення ефективності використання обчислювальних і енергетичних ресурсів при збереженні високого рівня захищеності КФС РЧ.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Kozelskiy O., Kashtalian A., Stetsyuk V., Martiniuk D., Sachenko A. A model of an intelligent clustering system with an external module for the architecture of RTOS with intensive changes of states regarding their flexibility and balancing. *1st International Workshop on Advanced Applied Information Technologies: (AdvAIT–2024),* Khmelnytskyi, Ukraine and Zilina, Slovakia, December 5, 2024. Vol. 3899. P. 234–243. URL: https://ceur–ws.org/Vol–3899/paper21.pdf (дата звернення 23.03.2026)

2. Kozelskiy O., Drozd A., Savenko B., Gaj P. A model for probabilistic monitoring and proactive restart of real–time operating systems under intensive state changes in cyber–physical systems. *2nd International Workshop on Intelligent & CyberPhysical Systems: (ICyberPhyS 2025),* Khmelnytskyi, Ukraine, July 4, 2025. Vol. 4013. P. 198–210. URL: https://ceur–ws.org/Vol–4013/paper16.pdf (дата звернення 23.03.2026)

3. Savenko O., Gaj P., Sierhieiev Ye. Detection of buffer overflow vulnerabilities in system software based on a graph and transformer model. *AISSLE–2025: The International Workshop on Applied Intelligent Security Systems in Law Enforcement*, October, 30–31, 2025, Vinnytsia, Ukraine. Pp. 292–305. URL: https://ceur–ws.org/Vol–4126/paper17.pdf (дата звернення 23.03.2026)

4. Sierhieiev Y., Paiuk V., Savenko O., Drozd A. Improvement of effectiveness for Static Application Security Testing for detection of SQL Injection vulnerabilities. *IEEE 14th International Conference on Dependable Systems, Services and Technologies (DESSERT–2024)* : Proceedings. Athens, Greece, October 11–13, 2024. Pp. 1–6. DOI: https://doi.org/10.1109/DESSERT65323.2024.11122171 (дата звернення 23.03.2026)

5. Yuce B., Schaumont P., Witteman M. Fault Attacks on Secure Embedded Software: Threats, Design, and Evaluation. J. Hardw. *Syst. Secur*. 2018. Vol. 2. Pp. 111–130.

6. Mishra J., Sahay S.K. Modern Hardware Security: A Review of Attacks and Countermeasures. *arXiv* 2025, *arXiv*:2501.04394. http://arxiv.org/abs/2501.04394

7. Savenko B., Kashtalian A., Lysenko S., Savenko O. Malware Detection By Distributed Systems with Partial Centralization, *2023 IEEE 12th International*

Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS), Dortmund, Germany, 2023. Pp. 265–270. DOI: 10.1109/IDAACS58523.2023.10348773 (дата звернення 23.03.2026)

8. Shuvo A.M., Zhang T., Farahmandi F., Tehranipoor M. A comprehensive survey on non–invasive fault injection attacks. *Cryptol. ePrint Arch*. 2023.

9. Gangolli A., Mahmoud Q.H., Azim A. A systematic review of fault injection attacks on IOT systems. *Electronics.* 2022, 11, 2023. P.23–45

10. Kazemi Z., Hely D., Fazeli M., Beroulle V. A Review on Evaluation and Configuration of Fault Injection Attack Instruments to Design Attack Resistant MCU–Based IoT Applications. *Electronics*. 2020, 9, 1153.

11. Tighazoui A., Sauvey C., Sauer N. Predictive-reactive strategy for flowshop rescheduling problem—Minimizing the total weighted waiting times and instability. *Journal of Manufacturing Systems*. 2021. Vol. 60. P. 28–40.

12. Canella C., Van Bulck J., Schwarz M., Lipp M., Von Berg B., Ortner P., Piessens F., Evtyushkin D., Gruss D. A systematic evaluation of transient execution attacks and defenses. *In Proceedings of the 28th USENIX Security Symposium (USENIX Security 19)*, Santa Clara, CA, USA, 14–16 August 2019. Pp. 249–266. (дата звернення 23.03.2026)

13. Ge Q., Yarom Y., Cock D., Heiser G. A survey of microarchitectural timing attacks and countermeasures on contemporary hardware. *J. Cryptogr. Eng*. 2018, 8, 1–27.

14. Xiong W., Szefer J. Survey of transient execution attacks and their mitigations. *ACM Comput. Surv*. (CSUR) 2021, 54, 1–36.

15. Agoyan M., Dutertre J.M., Naccache D., Robisson B., Tria A. When Clocks Fail: On Critical Paths and Clock Faults. In Smart Card Research and Advanced Application, *Proceedings of the 9th IFIP WG 8.8/11.2 International Conference, CARDIS 2010,* Passau, Germany, 14–16 April 2010, Proceedings; Springer: Berlin/Heidelberg, Germany, 2010; Volume 6035, Pp. 182–193. (дата звернення 23.03.2026)

16. Claudepierre L., Péneau P.Y., Hardy D., Rohou E. TRAITOR: A Low–Cost Evaluation Platform for Multifault Injection. *In Proceedings of the 2021 International*

Symposium on Advanced Security on Software and Systems, Virtual Event, Hong Kong, 7 June 2021. Pp. 51–56. (дата звернення 23.03.2026)

17. New AE Technology Inc. ChipWhisperer Documentation. *Online Resource*. Available online: https://chipwhisperer.readthedocs.io/en/latest/ (дата звернення 23.03.2026)

18. Luo J., Fujimura S., El Baz D., Plazolles B. GPU based parallel genetic algorithm for solving an energy efficient dynamic flexible flow shop scheduling problem. *Journal of Parallel and Distributed Computing*. 2019. Vol. 123. P. 184–197.

19. Denysiuk D., Savenko O., Lysenko S., Savenko B., Kashtalian A. Method for Detecting Steganographic Changes in Images Using Machine Learning. *13th International Conference on Dependable Systems, Services and Technologies (DESSERT)*, Athens, Greece, 2023. Pp. 1–6. DOI: 10.1109/DESSERT61349.2023.10416453 (дата звернення 23.03.2026)

20. Chen Z., Vasilakis G., Murdock K., Dean E., Oswald D., Garcia F.D. VoltPillager: Hardware–Based Fault Injection Attacks Against Intel SGX Enclaves Using the SVID Voltage Scaling Interface. *In Proceedings of the 30th USENIX Security Symposium (USENIX Security 21)*, Vancouver, BC, Canada, 11–13 August 2021. Pp. 699–716. (дата звернення 23.03.2026)

21. Buhren R., Jacob H.N., Krachenfels T., Seifert J.P. One Glitch to Rule Them All: Fault Injection Attacks Against AMD's Secure Encrypted Virtualization. *In Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, Virtual Event, Republic of Korea, 15–19 November 2021. Pp. 2875–2889. (дата звернення 23.03.2026)

22. Dumont M., Lisart M., Maurine P. Modeling and Simulating Electromagnetic Fault Injection. IEEE Trans. Comput.–Aided Des. *Integr. Circuits Syst*. 2021. Vol. 40. Pp. 680–693.

23. Gomes M., Barbosa-Póvoa A., Novais A. Reactive scheduling in a make-to-order flexible job shop with re-entrant process and assembly: A mathematical programming approach. *Computers & Operations Research*. 2019. Vol. 40. P. 1891–1900.

24. Trouchkine T., Bouffard G., Clédière J. Fault Injection Characterization on Modern CPUs: From the ISA to the Micro–Architecture. *In Proceedings of the 13th IFIP International Conference on Information Security Theory and Practice (WISTP),* Paris, France, 11–12 December 2019; pp. 123–138. (дата звернення 23.03.2026)

25. Bedratyuk L., Savenko O. *MATCH Commun. Math. Comput. Chem.* 2018. Vol. 79. Pp. 407–414.

26. Moghaddam S. K., Saitou K. A novel predictive-reactive rescheduling method for products assembly lines with optimal pegging. *IEEE Transactions on Automation Science and Engineering*. 2022. Vol. 19. P. 360–375.

27. Xin L., Chu F., Zheng F., Chu C., Liu M. Parallel machine scheduling with stochastic release times and processing times. *International Journal of Production Research*. 2021. Vol. 59. P. 6327–6346.

28. Murdock K., Oswald D., Garcia F.D., Van Bulck J., Gruss D., Piessens F. Plundervolt: Software–based Fault Injection Attacks against Intel SGX. *In Proceedings of the 2020 IEEE Symposium on Security and Privacy (SP)*, San Francisco, CA, USA, 18–21 May 2020. Pp. 1466–1482. (дата звернення 20.03.2026)

29. Gonidec G.L., Real M.M., Bouffard G., Prévotet J.C. Do Not Trust Power Management: A Survey on Internal Energy–based Attacks Circumventing Trusted Execution Environments Security Properties. *arXiv 2024, arXiv*:2405.15537. Available online: http://arxiv.org/abs/2405.15537 (дата звернення 11.03.2026)

30. Trouchkine T., Bukasa S.K., Escouteloup M., Lashermes R., Bouffard G. Electromagnetic Fault Injection Against a System–on–Chip, Toward New Micro–Architectural Fault Models. *arXiv 2019, arXiv*:1910.11566.

31. Trouchkine T., Bouffard G., Clédière J. EM Fault Model Characterization on SoCs: From Different Architectures to the Same Fault Model. *In Proceedings of the 2021 Workshop on Fault Detection and Tolerance in Cryptography (FDTC)*, Milan, Italy, 17 September 2021. Pp. 31–38.

32. Laurent J., Deleuze C., Pebay–Peyroula F., Beroulle V. Bridging the Gap between RTL and Software Fault Injection. *J. Emerg. Technol. Comput. Syst*. 2021, 17, 24.

33. Valledor P., Gomez A., Puente J., Fernandez I. Solving rescheduling problems in dynamic permutation flow shop environments with multiple objectives using the hybrid dynamic non-dominated sorting genetic II algorithm. *Computers & Industrial Engineering*. 2022. Vol. 166. Article 107937.

34. Alshaer I., Colombier B., Deleuze C., Beroulle V., Maistri P. Microarchitecture–Aware Fault Models: Experimental Evidence and Cross–Layer Inference Methodology. *In Proceedings of the 2021 16th International Conference on Design & Technology of Integrated Systems in Nanoscale Era (DTIS)*, Montpellier, France, 28–30 June 2021. Pp. 1–6. (дата звернення 11.03.2026)

35. Werner V., Maingault L., Potet M.L. An End–to–End Approach for Multi–Fault Attack Vulnerability Assessment. *In Proceedings of the 2020 Workshop on Fault Detection and Tolerance in Cryptography (FDTC)*, Milan, Italy, 13 September 2020; pp. 10–17. (дата звернення 11.03.2026)

36. Elmohr M.A., Liao H., Gebotys C.H. EM Fault Injection on ARM and RISC–V. *In Proceedings of the 2020 21st International Symposium on Quality Electronic Design (ISQED)*, Santa Clara, CA, USA, 25–26 March 2020; pp. 206–212. (дата звернення 11.03.2026)

37. Amin K., Gebotys C., Faraj M., Liao H. Analysis of Dynamic Laser Injection and Quiescent Photon Emissions on an Embedded Processor. *J. Hardw. Syst. Secur*. 2020, 4, 55–67.

38. Козельський О., Савенко Б. Дворівнева стратегія підвищення відмовостійкості операційних систем реального часу з використанням ймовірнісного аналізу. *Herald of Khmelnytskyi National University. Technical Sciences*. 2025. №353(3.2). С. 438–446. DOI: https://doi.org/10.31891/2307–5732–2025–353–60 (дата звернення 11.03.2026)

39. Козельський О., Савенко Б. Зовнішня адаптивна система кластеризації для підвищення гнучкості операційних систем реального часу на основі динамічного планування завдань. *Вчені записки Таврійського національного університету імені В.І. Вернадського. Серія: Технічні науки*. Том 5. 2025. С. 124–132 DOI: https://doi.org/10.32782/2663–5941/2025.4.2/38

40. Shokraneh K., Moghaddam S., Saitou K. Predictive-reactive rescheduling for new order arrivals with optimal dynamic pegging. *International Journal of Production Research*. 2020. Vol. 58. P. P. 4232–4246.

41. Colombier B., Menu A., Dutertre J.M., Moëllic P.A., Rigaud J.B., Danger J.L. Laser–Induced Single–Bit Faults in Flash Memory: Instructions Corruption on a 32–Bit Microcontroller. *In Proceedings of the 2019 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, McLean, VA, USA, 5–10 May 2019. Pp. 1–10.

42. Svystun, S., Melnychenko, O., Radiuk, P., Savenko, O., Sachenko, A., & Lysyi, A. (2024). Thermal and RGB Images Work Better Together in Wind Turbine Damage Detection. *International Journal of Computing*, *23*(4), 526-535. https://doi.org/10.47839/ijc.23.4.3752

43. Alshaer I., Colombier B., Deleuze C., Beroulle V., Maistri P. Variable–Length Instruction Set: Feature or Bug? *In Proceedings of the 2022 25th Euromicro Conference on Digital System Design (DSD)*, Maspalomas, Spain, 31 August–2 September 2022. Pp. 464–471. (дата звернення 11.03.2026)

44. Сєргєєв Є.В. Композитна оцінка ризику переповнення буфера і її трансляція в дії CI/CD. *MEASURING AND COMPUTING DEVICES IN TECHNOLOGICAL PROCESSES*. 2025. № 84(4). Pp. 89–94. DOI: https://doi.org/10.31891/2219–9365–2025–84–10

45. Сєргєєв Є., Каштальян А., Ковальчук В., Савенко О., Іванченко О. Ефективність і вдосконалення SAST у контексті SQL Injection вразливостей. *Information Technology: Computer Science, Software Engineering and Cyber Security*. 2024. № 3. С. 149–158. DOI: https://doi.org/10.32782/IT/2024–3–16

46. Alshaer I., Colombier B., Deleuze C., Beroulle V., Maistri P. Microarchitectural Insights into Unexplained Behaviors Under Clock Glitch Fault Injection. *In Smart Card Research and Advanced Applications; Lecture Notes in Computer Science*; Bhasin, S., Roche, T., Eds.; Springer: Berlin/Heidelberg, Germany, 2024. Vol. 14530.

47. Alshaer I., Al–kaf A. Egloff V. Beroulle V. Inferred Fault Models for RISC–V and Arm: A Comparative Study. *In Proceedings of the 2024 IEEE International*

Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT), Didcot, UK, 8–10 October 2024. Pp. 1–6. (дата звернення 11.03.2026)

48. Di Natale G., Gizopoulos D., Di Carlo S., Bosio A., Canal R. Cross–Layer Reliability of Computing Systems*; IET—The Institution of Engineering and Technology*: London, UK, 2020. (дата звернення 11.03.2026)

49. Petrovic D., Duenas A. A fuzzy logic based production scheduling-rescheduling in the presence of uncertain disruptions. *International Journal of Production Research*. 2020. Vol. 44. P. 3571–3587.

50. Werner M., Unterluggauer T., Schaffenrath D., Mangard S. Sponge–Based Control–Flow Protection for IoT Devices. *In Proceedings of the 2018 IEEE European Symposium on Security and Privacy (EuroS&P)*, London, UK, 24–26 April 2018. Pp. 214–226.

51. Lashermes R., Bouder H., Thomas G. Secure IT Systems: Hardware–Assisted Program Execution Integrity: HAPEI. *In Secure IT Systems, Proceedings of the 23rd Nordic Conference, NordSec* 2018, Oslo, Norway, 28–30 November 2018, Proceedings; Lecture Notes in Computer Science; Springer International Publishing: Cham, Switzerland, 2018. Vol. 11252. (дата звернення 11.03.2026)

52. Savry O., El–Majihi M., Hiscock T. Confidaent: Control FLow Protection with Instruction and Data Authenticated Encryption. *In Proceedings of the 2020 23rd Euromicro Conference on Digital System Design (DSD)*, Kranj, Slovenia, 26–28 August 2020. Pp. 246–253.

53. Geurtsen M., Adan J., Akçay A. Quick dispatching-rules-based solution for the two parallel machines problem under mold constraints. *Flexible Services and Manufacturing Journal*. 2024. Vol. 36. P. 224–249.

54. Nasahl P., Sultana S., Liljestrand H., Grewal K., LeMay M., Durham D.M., Schrammel D., Mangard S. EC–CFI: Control–Flow Integrity via Code Encryption Counteracting Fault Attacks. *In Proceedings of the 2023 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, San Jose, CA, USA, 1–4 May 2023; pp. 24–35.

55. Chamelot T., Couroussé D., Heydemann K. SCI–FI: Control Signal, Code, and Control Flow Integrity against Fault Injection Attacks. *In Proceedings of the 2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Antwerp, Belgium, 14–23 March 2022. Pp. 556–559.

56. Ebrahimi M., Ghomi F., Karimi B. Hybrid flow shop scheduling with sequence dependent family setup time and uncertain due dates. *Applied Mathematical Modelling*. 2020. Vol. 38. P. 2490–2504.

57. Chamelot T., Couroussé D., Heydemann K. MAFIA: Protecting the Microarchitecture of Embedded Systems Against Fault Injection Attacks. *IEEE Trans. Comput.–Aided Des. Integr. Circuits Syst.* 2023. Vol. 42. Pp. 4555–4568.

58. Sun M. та ін. Real-time rescheduling for smart shop floors: An integrated method. *Flexible Services and Manufacturing Journal*. 2024. DOI: URL: https://doi.org/10.1007/s10696-024-09574-6. (дата звернення: 23.02.2026)

59. Leplus G., Savry O., Bossuet L. SecDec: Secure Decode Stage Thanks to Masking of Instructions with the Generated Signals. *In Proceedings of the 2022 25th Euromicro Conference on Digital System Design (DSD)*, Maspalomas, Spain, 31 August–2 September 2022. Pp. 556–563.

60. Zgheib A., Potin O., Rigaud J.B., Dutertre J.M. A CFI Verification System Based on the RISC–V Instruction Trace Encoder. *In Proceedings of the 2022 25th Euromicro Conference on Digital System Design (DSD)*, Maspalomas, Spain, 31 August–2 September 2022. Pp. 456–463.

61. Zgheib A., Potin O., Rigaud J.B., Dutertre J.M. CIFER: Code Integrity and Control Flow Verification for Programs Executed on a RISC–V Core. *In Proceedings of the 2023 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, San Jose, CA, USA, 1–4 May 2023. Pp. 100–110.

62. Brusaferri A., Leo E., Nicolosi L., Ramin D., Spinelli S. Integrated automation system with PSO based scheduling for PCB remanufacturing plants. *Computers & Industrial Engineering*. 2019. Vol. 129. P. 179–190.

63. Shrivastwa R.R., Guilley S., Danger J.L. Multi–source fault injection detection using machine learning and sensor fusion. *In Security and Privacy, Proceedings of the*

Second International Conference, ICSP 2021, Jamshedpur, India, 16–17 November 2021, Proceedings; Springer: Berlin/Heidelberg, Germany, 2021. Pp. 93–107.

64. Gambra A., Chatterjee D., Rioja U., Armendariz I., Batina L. Machine Learning–Based Detection of Glitch Attacks in Clock Signal Data. *Cryptol. ePrint Arch*. 2024.

65. Sayeed S., Marco–Gisbert H., Ripoll I., Birch M. Control–Flow Integrity: Attacks and Protections. *Applied Sciences*. 2019; 9(20):4229. https://doi.org/10.3390/app9204229

66. Souravlas S., Sifaleras A., Katsavounis S. Hybrid CPU-GPU community detection in weighted networks. *IEEE Access*. 2020. Vol. 8. P. 57527–57551.

67. Takeda-Berger S., Frazzon E. An inventory data driven model for predictive reactive production scheduling. *Journal of Manufacturing Systems*. 2023. Vol. 69. P. 324–335.

68. Takeda-Berger S., Zanella R., Frazzon E. Towards a data-driven predictive-reactive production scheduling approach based on inventory availability. *Computers & Industrial Engineering*. 2019. Vol. 133. P. 11–23.

69. Manzini M., Demeulemeester E., Urgo M. A predictive–reactive approach for the sequencing of assembly operations in an automated assembly line. *International Journal of Production Research*. 2022. Vol. 60. P. 30–44.

70. Qiao F., Ma Y., Zhou M., Wu Q. A novel rescheduling method for dynamic semiconductor manufacturing systems. *IEEE Transactions on Semiconductor Manufacturing*. 2020. Vol. 33. P. 543–555.

71. Козельський О.В. Метод тензорної декомпозиції для адаптивного розподілу ресурсів у системах реального часу. *Measuring and computing devices in technological processes*. 2025. №82(2). С. 426–433. DOI: https://doi.org/10.31891/2219–9365–2025–82–61

72. Козельський О., Савенко О. Проактивний механізм інформаційної безпеки в операційних системах реального часу з використанням гібридного сторожового таймера. *Measuring and computing devices in technological processes*. 2025. №83(3). С. 459 – 466. DOI: https://doi.org/10.31891/2219–9365–2025–83–57

73. Козельський О., Савенко О. Виявлення зловмисних атак на сенсори та підробки телеметрії в кіберфізичних системах на основі модифікованого фільтра Калмана. *Measuring and computing devices in technological processes.* 2025. №84(4). С. 228–235. DOI: https://doi.org/10.31891/2219–9365–2025–84–24

74. Sierhieiev Y., Paiuk V., Nicheporuk A., Kwiecien A., Huralnyk O. Detection and prediction of the vulnerabilities in software systems based on behavioral analysis with machine learning. (2024) *CEUR Workshop Proceedings*, 3736, pp. 239–254. *The 1th Proceedings of the 1st International Workshop on Intelligent & CyberPhysical Systems (ICyberPhyS 2024)*. CEUR–Workshop Proceedings. Vol. 3736. (Khmelnytskyi, Ukraine, June 28, 2024). Khmelnytskyi, 2024. Pp. 239–254. URL: https://ceur–ws.org/Vol–3736/.

75. Savenko O., Sierhieiev Y., Gaj P., Balej J. Using artificial intelligence in the context of buffer overflow vulnerabilities. *The 2nd International Workshop on Intelligent & CyberPhysical Systems (ICyberPhyS 2025), CEUR Workshop Proceedings*. Vol. 4013. Khmelnytskyi, Ukraine, 4 July 2025. Pp. 211–220. URL: https://ceur–ws.org/Vol–4013/paper17.pdf

76. Boulifa R, Di Natale G, Maistri P. Countermeasures Against Fault Injection Attacks in Processors: A Review. *Information*. 2025. 16(4): 293. https://doi.org/10.3390/info16040293

77. Wang Z., Ding X., Pang C., Guo J., Zhu J., Mao B. To detect stack buffer overflow with polymorphic canaries. *In Proceedings of the 2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, Luxembourg, 25–28 June 2018; pp. 243–254. (дата звернення 11.03.2026)

78. Tighazoui A., Sauvey C., Sauer N. New efficiency-stability criterion in a rescheduling problem with dynamic jobs weights. *International Journal of Production Economics*. 2020. Vol. 229. Article 107855.

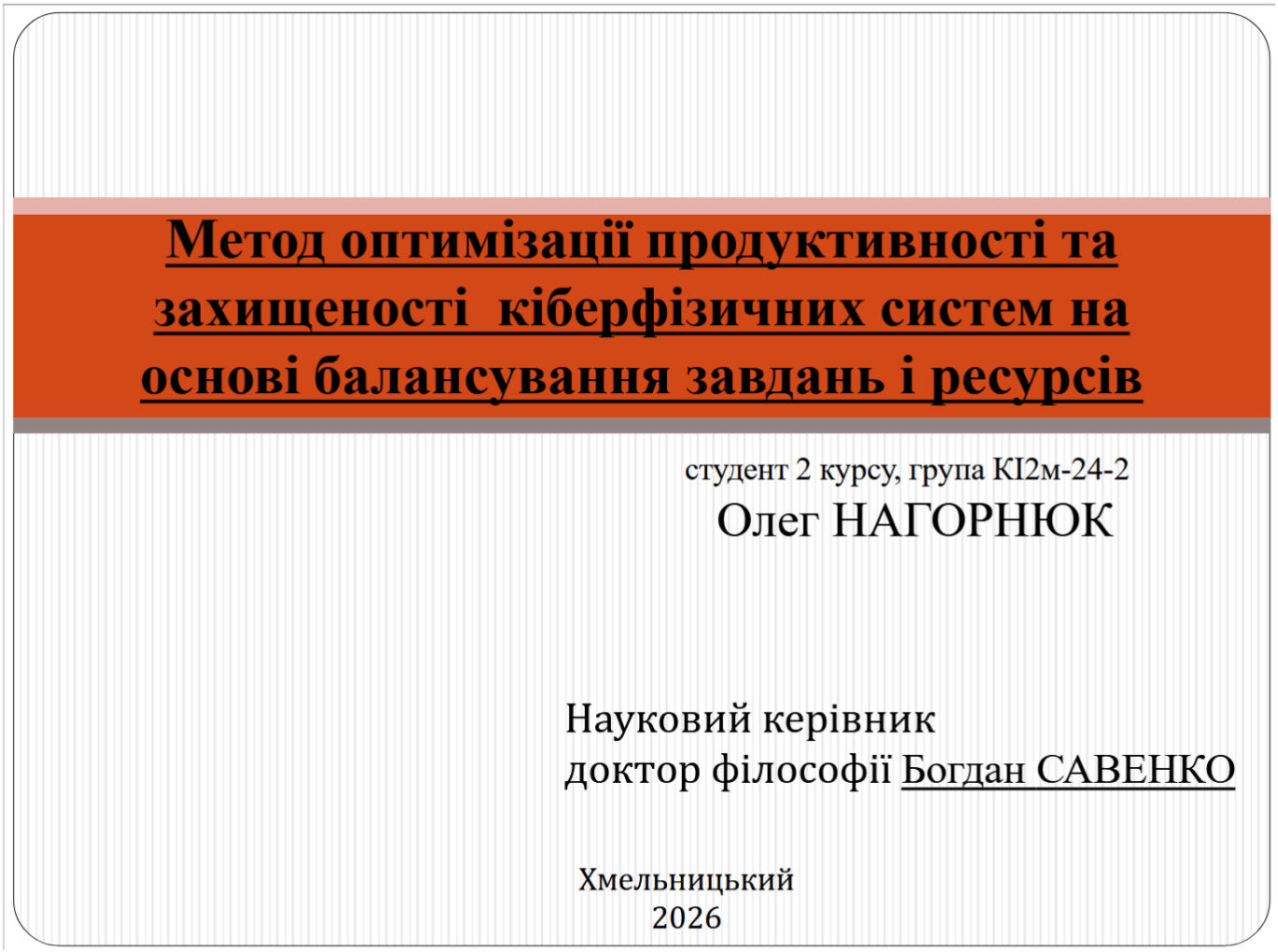
79. Bellec N., Hiet G., Rokicki S., Tronel F., Puaut I. RT–DFI: Optimizing Data–Flow Integrity for Real–Time Systems. *ECRTS 2022 – 34th Euromicro Conference on Real–Time Systems*, Jul 2022, Modène, Italy. Pp.1–24. DOI: 10.4230/LIPIcs.ECRTS.2022.18 (дата звернення 11.03.2026)

80. Jinmeng Z., Ziyue P., Xun X., Wenbo S. Practical Protection Design of Forward–Edge Control–Flow Integrity for Linux Kernel. 2025. Pp. 3370–3375. DOI: 10.1109/ICC52391.2025.11161046.

81. НАГОРНЮК, О., ДРОЗД, А., МЕДЗАТИЙ, Д. МЕТОД ОПТИМІЗАЦІЇ ПРОДУКТИВНОСТІ ТА ЗАХИЩЕНОСТІ КІБЕРФІЗИЧНИХ СИСТЕМ РЕАЛЬНОГО ЧАСУ НА ОСНОВІ БАЛАНСУВАННЯ ЗАВДАНЬ І РЕСУРСІВ. *Herald of Khmelnytskyi National University. Technical Sciences*. 2026. № 361(1). С. 575–585. https://doi.org/10.31891/2307–5732–2026–361–79

ДОДАТОК А
(обов'язковий)

Презентація до роботи



Актуальність роботи.

Кіберфізичні системи реального часу (КФС РЧ) відіграють дедалі важливішу роль у сучасному світі, функціонуючи в середовищах із жорсткими часовими обмеженнями та охоплюючи спектр від простих вбудованих пристроїв, як-от мікроконтролери, до складних платформ, таких як дрони та автономні транспортні засоби. Ці системи переважно програмуються мовами без механізмів захисту пам'яті, наприклад C/C++, що спричиняє численні вразливості безпеки та пов'язані проблеми, а в критичних для безпеки застосуваннях забезпечення надійного захисту є ключовим для гарантії безпеки та дотримання часових вимог. Існуючі механізми безпеки, розроблені для універсальних обчислювальних систем, генерують значні додаткові витрати, що ускладнює їх інтеграцію в КФС РЧ, тоді як модель обчислень у системах реального часу дозволяє впроваджувати захисні механізми з мінімальними витратами ресурсів, тому доцільно розробляти спеціально адаптовані механізми безпеки для РЧ, мінімізуючи їхній вплив на продуктивність. Ця робота розширює три ключові механізми захисту — цілісність потоків даних, керуючого потоку та вказівників, — з адаптаціями для РЧ, що включають перевірку цілісності потоків даних у вільний час під час ітерацій завдань для забезпечення міцного захисту із мінімальними витратами в найгіршому випадку, асинхронну перевірку цілісності керуючого потоку у вбудованих системах через вікна планування для зменшення впливу на реальний час, а також використання доступного системного часу для перевірки цілісності вказівників, що підвищує загальну безпеку без порушення розкладу завдань, таким чином пропонуючи підходи для оптимізації балансу між рівнем безпеки та продуктивністю в системах реального часу.

Актуальність роботи полягає в розробці адаптованих механізмів захисту для КФС РЧ, які забезпечують надійну безпеку з мінімальними додатковими витратами на продуктивність, [illegible]ховуючи жорсткі часові обмеження та вразливості, пов'язані з мовами програмування без захисту пам'яті, такими як C/C++.

2

Об’єктом дослідження є процеси забезпечення продуктивності та захищеності кіберфізичних систем реального часу.

Предметом дослідження є методи оптимізації планування завдань і розподілу ресурсів у кіберфізичних системах з урахуванням консервативних оцінок WCET, slack time та адаптивних механізмів безпеки.

Метою кваліфікаційної роботи магістра є підвищення рівня захищеності та ефективності використання ресурсів кіберфізичних систем без порушення часових обмежень шляхом розробки методу балансування завдань і ресурсів на основі адаптивного управління механізмами безпеки.

Поставлена мета досягається розв’язанням таких основних завдань:

1) проаналізувати існуючі підходи до забезпечення безпеки кіберфізичних систем реального часу та визначити їхні обмеження щодо жорсткого резервування ресурсів і додаткових часових витрат;

2) розробити метод оптимізації продуктивності та захищеності кіберфізичних систем на основі балансування завдань і ресурсів;

3) розробити адаптивні механізми захисту цілісності потоків даних, потоку керування та покажчиків із урахуванням планування завдань і динамічного розподілу ресурсів;

4) розробити комплексну модель та архітектуру кіберфізичної системи з інтеграцією просторової й часової ізоляції та динамічного управління рівнями безпеки;

5) провести експериментальну оцінку ефективності запропонованого методу з точки зору своєчасності виконання завдань, рівня захисту та додаткових витрат безпеки.

3

Наукова новизна отриманого результату:

удосконалено метод оптимізації продуктивності та захищеності кіберфізичних систем на основі балансування завдань і ресурсів, який, на відміну від традиційних статичних підходів із жорстким резервуванням або постійними перевірками, використовує консервативні оцінки WCET і доступний slack time для адаптивного перенесення витрат механізмів безпеки в часові резерви, що забезпечує гарантовану своєчасність виконання завдань при підвищеному рівні захисту та ефективному використанні ресурсів..

На основі проведених досліджень розроблено метод та архітектуру для оптимізації механізмів захисту в кіберфізичних системах реального часу, що дозволяють інтегрувати надійний захист без значного впливу на реальний час виконання завдань.

Практична значимість отриманих результатів полягає у розробці концептуальних рішень для підвищення безпеки критичних систем за рахунок інтеграції цих адаптованих механізмів у вбудовані системи реального часу, яка оптимізує ресурсне навантаження без порушення розкладу завдань, що є новим підходом для критичних застосувань, таких як дрони та автономні транспортні засоби.

4

5

Рисунок 1 - Узагальнена архітектура кіберфізичної системи з локалізацією області дослідження та механізмів оптимізації продуктивності й захищеності

Для моделі завдань КФС розглядатимемо множину завдань реального часу так:

$$Z_{CPS} = \left\{z_{CPS,1}, z_{CPS,2}, \dots, z_{CPS,N_{Z_{CPS}}}\right\}, \quad (1)$$

де $z_{CPS,i}$ - i – те завдання реального часу; $i = 1,2, \dots, N_{Z_{CPS}}$; $N_{Z_{CPS}}$ – кількість завдань КФС.

Кожне завдання з множини завдань Z_{CPS} характеризується такими параметрами:

1) $c_i(t)$ - фактичний час виконання на момент часу t;
2) $O_{WCET,i}$ - консервативна оцінка найгіршого часу виконання;
3) D_i - відносний дедлайн;
4) T_i - період або мінімальний інтервал активації.

Резервний час визначається так:

$$s_i(t) = O_{WCET,i} - c_i(t), \quad (2)$$

де $c_i(t)$ - фактичний час виконання на момент часу t; $i = 1,2, \dots, N_{Z_{CPS}}$; $N_{Z_{CPS}}$ – кількість завдань КФС; $O_{WCET,i}$ - консервативна оцінка найгіршого часу виконання.

Цей показник $s_i(t)$ згідно формули (2.2) використовується як динамічне обмеження для виконання додаткових захисних операцій.

6

Цілісність потоку даних (ЦПД, Data-Flow Integrity, DFI) спрямована на запобігання несанкціонованим модифікаціям або використанню даних у програмі.

Умова цілісності полягає в тому, що кожне зчитування даних має відповідати дозволеному попередньому запису. Обчислювальні витрати перевірок ЦПД для завдання оцінимо так:

$$O_{\text{ЦПД},i} = k_d \cdot \left|E^1_{d,i}\right|, \quad (3)$$

де k_d - середня вартість перевірки однієї залежності; $E_{d,j}$ - допустимі середні значення залежності «запис–читання».

Перевірка полягає у валідації наступної адреси виконання:

$$A_{p,next} = F_A\left(A_{p,current}\right), \quad (4)$$

де $A_{p,next}$ – наступна адреса в керуючому потоці; $A_{p,current}$ – поточна адреса в керуючому потоці; F_A – функція перетворення адрес.

Для зменшення додаткових витрат вводиться дискретний рівень деталізації перевірок так:

$$L_{\text{ЦКП}} \in \{0,1,2\}, \quad (5)$$

де 0 - перевірки вимкнені; 1 - груба перевірка; 2 - детальна перевірка.

Витрати перевірок визначимо так:

$$O_{\text{ЦКП},j}(L) = k_c \cdot \left|E^1_{c,j}(L)\right|, \quad (6)$$

де k_c – коефіцієнт вартості.

Рівень $L_{\text{ЦКП}}$ обирається таким чином, щоб максимізувати захищеність за наявного slack time.

7

Модель **механізму цілісності вказівників** і пам'яті будемо розглядати в контексті спроможності здійснювати контроль коректності вказівників під час доступу до пам'яті.

Загальні витрати перевірок оцінюватимемо так:

$$O_{ptr,j} = k_p \cdot N_{ptr,j}, \tag{7}$$

де k_p – коефіцієнт вартості; $N_{ptr,j}$ — кількість операцій розіменування вказівника j – того потоку.

З метою адаптації до часових обмежень застосовується вибіркова перевірка лише критичних доступів, що формалізується за допомогою бінарних змінних вибору.

Згідно розроблених попередніх трьох моделей цілісних потоків даних, керуючого потоку та цілісності вказівкиків розробимо узагальнену модель оптимізації балансування безпеки та продуктивності в КФС.

Для кожного завдання формуємо вектор параметрів безпеки так:

$$q_j = \begin{bmatrix} A_{\text{цпд},j} \\ L_{\text{цкп},j} \\ B_{ptr,j} \end{bmatrix}, \tag{8}$$

де $A_{\text{цпд},j}$ – інформація щодо цілісних потоків даних; $L_{\text{цкп},j}$ – інформація щодо керуючого потоку; $B_{ptr,j}$ – інформація щодо вказівників.

8

Рівень захищеності системи задамо функцією так:

$$S_1(Q) = \sum_{j=1}^{n} w_j \cdot F_2(q_j), \tag{9}$$

де w_j – коефіцієнти критичності завдань; F_2 – функція оцінювання захищеності потоків даних, керуючого потоку та вказівників; n – кількість потоків; j – номер потоку даних; Q – множина завдань, яка складається з елементів q_j; $j = 1,2,\dots,n$.

Задачу оптимізації балансування безпеки та продуктивності в КФС визначимо так:

$$\max_{Q} S_1(Q), \tag{10}$$

за умови

$$c_j(t) + O_{\text{цпд},j} + O_{\text{цкп},j}(L) + O_{ptr,j} \le O_{WCET,j} \tag{11}$$

та глобальних планувальних обмежень і при цьому $c_j(t)$ - фактичний час виконання на момент часу t; $j = 1,2,\dots,N_{Z_{CPS}}$; $N_{Z_{CPS}}$ – кількість завдань КФС; $O_{WCET,j}$ - консервативна оцінка найгіршого часу виконання.

9

Рисунок 2 – Схема прототипу оптимізації

10

Рисунок 3 – Приклад атаки переповнення буфера в КФС

11

Основні кроки методу оптимізації продуктивності та захищеності КФС з використанням покрокового підходу.

Крок 1. Моделювання системи та визначення обмежень.

1.1. На цьому кроці потрібно розробити формальну модель КФС РЧ, причому початковими вимогами є завдання реального часу з жорсткими дедлайнами, потоки даних, потоки керування та критичні ресурси.

1.2. Визначити WCET для кожного завдання.

1.3. Визначити slack time – час, що залишається після виконання завдань до дедлайнів.

1.4. Задокументувати критичні точки для забезпечення безпеки, зокрема: контрольні точки цілісності; важливі покажчики; контексти керування.

Крок 2. Аналіз потенційних механізмів захисту.

2.1. Виділити механізми перевірки цілісності потоків даних, потоку керування,

асинхронного керування, контекстуальних покажчиків.

2.2. Оцінити витрати WCET для кожного механізму безпеки.

2.3. Визначити, які механізми можуть бути адаптовані під наявний slack time.

12

Крок 3. Оцінка та використання Slack Time.

3.1. Виконати консервативну оцінку slack для всіх завдань, зокрема: статичний аналіз WCET і профілювання реальних даних; символічне виконання для точного визначення резервів часу.

3.2. Визначити політики динамічного використання slack для безпеки, зокрема: виконання додаткових перевірок, якщо є вільний час; пріоритетне перенесення витрат захисту у час, що не впливає на терміни виконання.

Крок 4. Адаптивне планування завдань.

4.1. Впровадити моделі планування завдань, що враховують консервативні WCET, динамічно доступний slack, апаратну підтримку динамічного перемикання коду.

4.2. Розробити політики адаптивного розподілу ресурсів для перевірок цілісності потоків даних, асинхронного керування, захисту контексту та покажчиків

13

Крок 5. Інтеграція просторової та часової безпеки.

5.1. Розмежувати ресурси для забезпечення просторової ізоляції для захисту критичних структур даних та контролю доступу до покажчиків і контекстів.

5.2. Застосувати часову ізоляцію для виконання захисних перевірок у межах slack time, мінімізації впливу механізмів безпеки на основний граф завдань.

Крок 6. Динамічне управління безпекою.

6.1. Використання динамічного перемикання коду та контекстно-залежні механізми захисту для перевірки місць необхідного виконання і забезпечення контекстної чутливості покажчиків і потоків.

6.2. Адаптувати рівень захисту під наявний slack і поточне навантаження системи.

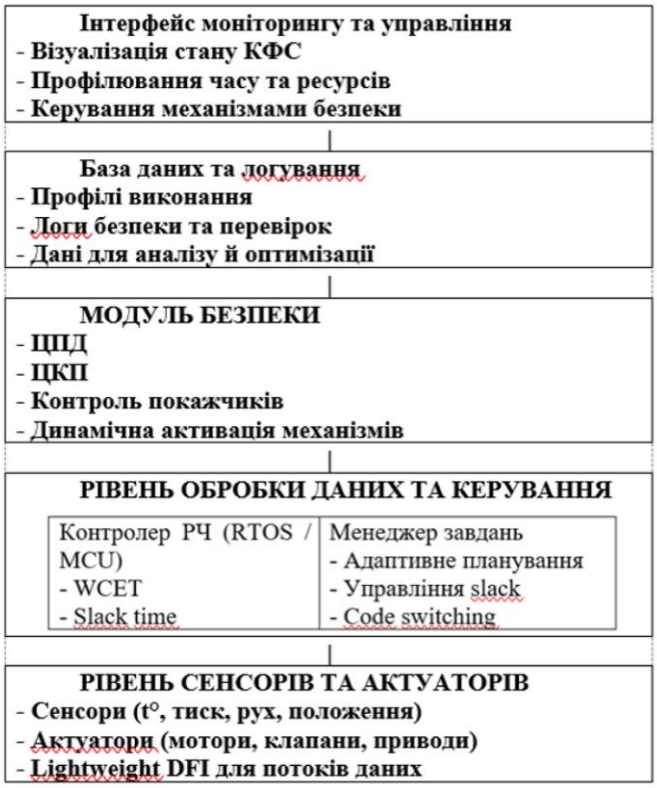
Крок 7. Тестування та валідація.

7.1. Профілювання системи під реальними сценаріями.

7.2. Симуляція пікових навантажень і перевірка термінів виконання завдань та спрацювань механізмів безпеки у відведений slack.

7.3. Валідація балансу продуктивності та безпеки із застосуванням кількісних метрик, зокрема: завантаження процесора; виконання перевірок; використання часу slack.

14



15

Рисунок 4 - Архітектура КФС РЧ

Таблиця 1

Вплив slack time на рівень безпеки

Сценарій	Середній slack time, мкс	Активні механізми безпеки	Пропущені задані терміни виконання
1	4200	ЦПД + ЦКП + вказівник	0
2	1800	ЦПД	0
3	200	Мінімальні	0

Таблиця 2

Додаткові витрати механізмів безпеки

Рівень безпеки	Додатковий час, % від WCET
Низький	1–2 %
Середній	4–6 %
Високий	8–12 %

16

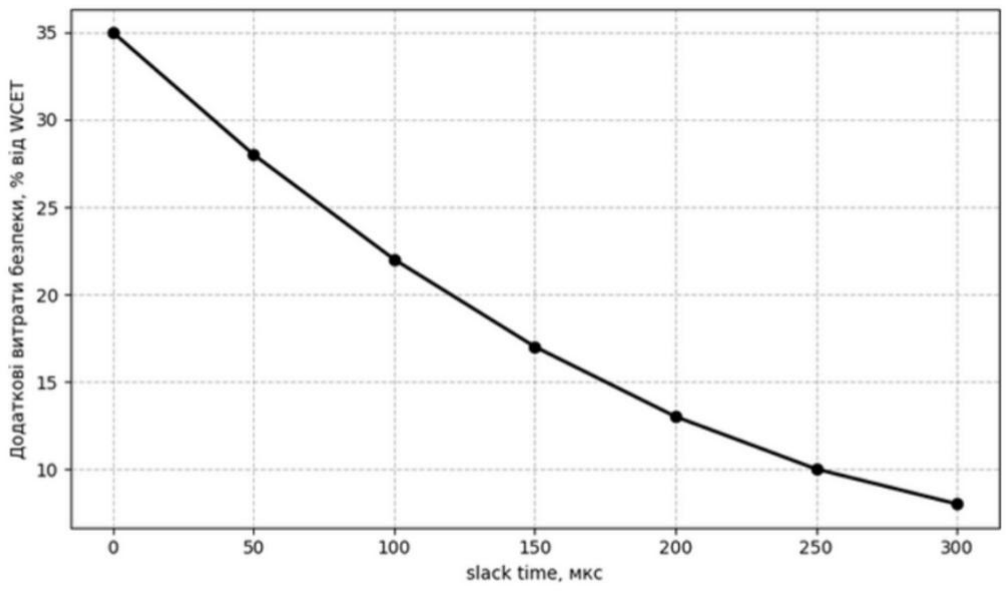


Рисунок 5 – Залежність додаткових витрат безпеки від slack time

17

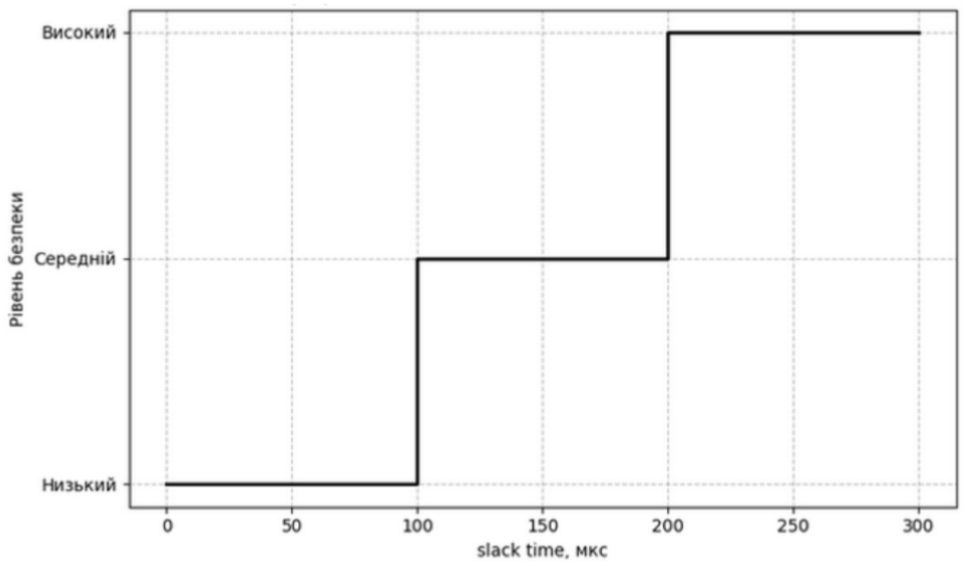


Рисунок 3 - Рівні безпеки залежно від slack time

18

ВИСНОВКИ

- У роботі за результатами виконаних теоретичних та практичних досліджень розроблено метод оптимізації продуктивності та захищеності кіберфізичних систем реального часу на основі балансування завдань і ресурсів та отримано такі результати.
- 1. Проведений аналіз існуючих підходів до забезпечення безпеки кіберфізичних систем реального часу показав, що традиційні методи, засновані на жорсткому резервуванні ресурсів або постійному застосуванні механізмів контролю, призводять до значних додаткових часових витрат і знижують ефективність використання ресурсів системи.
- 2. Розроблено метод використання консервативних оцінок WCET і slack time для адаптивного перенесення додаткових витрат механізмів безпеки у доступні часові резерви, що дозволяє забезпечити необхідний рівень захисту без порушення жорстких часових обмежень виконання завдань.
- 3. Запропоновано адаптивні механізми захисту цілісності потоків даних, потоку керування та покажчиків, інтегровані з плануванням завдань і динамічним розподілом ресурсів, які забезпечують контекстну чутливість перевірок і мінімізацію додаткових обчислювальних витрат.
- 4. Розроблено комплексну модель та архітектуру КФС РЧ з інтеграцією просторової й часової ізоляції та динамічного управління рівнями безпеки, що підтверджує практичну реалізованість і масштабованість запропонованого методу на сучасних апаратно-програмних платформах.
- 5. За результатами експериментальних досліджень встановлено, що запропонований метод забезпечує гарантовану своєчасність виконання критичних завдань, зменшення додаткових витрат безпеки та підвищення ефективності використання обчислювальних і енергетичних ресурсів при збереженні високого рівня захищеності КФС РЧ.

19

ДОДАТОК Б

(обов'язковий)

Наукова праця здобувача

Технічні науки *ISSN 2307-5732*

https://doi.org/10.31891/2307-5732-2026-361-79
УДК 004.75

НАГОРНЮК ОЛЕГ
Хмельницький національний університет
https://orcid.org/0009-0000-0887-5520
e-mail: medringone@gmail.com
ДРОЗД АНДРІЙ
Хмельницький національний університет
https://orcid.org/0009-0008-1049-1911
e-mail: andriydrozdit@gmail.com
МЕДЗАТИЙ ДМИТРО
Хмельницький національний університет
https://orcid.org/0009-0004-3247-6406
e-mail: medzatyid@khmnu.edu.ua

МЕТОД ОПТИМІЗАЦІЇ ПРОДУКТИВНОСТІ ТА ЗАХИЩЕНОСТІ КІБЕРФІЗИЧНИХ СИСТЕМ РЕАЛЬНОГО ЧАСУ НА ОСНОВІ БАЛАНСУВАННЯ ЗАВДАНЬ І РЕСУРСІВ

У статті досліджується проблема забезпечення безпеки кіберфізичних систем реального часу (КФС РЧ), що функціонують у середовищах із жорсткими часовими обмеженнями та часто реалізуються мовами програмування без вбудованого захисту пам'яті, такими як C/C++. Традиційні механізми безпеки, призначені для універсальних обчислювальних систем, створюють значні додаткові витрати й обмежують продуктивність, що робить їх застосування у КФС РЧ проблематичним. Основною метою роботи є підвищення рівня захищеності цих систем при мінімальному впливі на часові характеристики виконання завдань. Для досягнення цієї мети запропоновано адаптивні механізми забезпечення цілісності потоків даних, керуючого потоку та вказівників із використанням консервативних оцінок найгіршого часу виконання (WCET) та резервного часу (slack time). Резервний час використовується для динамічного виконання додаткових перевірок без порушення дедлайнів, що дозволяє формально гарантувати своєчасність роботи системи. Потоки даних та керуючий потік формалізовано у вигляді графів із матрицями суміжності та досяжності, що забезпечує точну оцінку відповідності фактичних операцій заздалегідь визначеним політикам безпеки. Для вказівників застосовується контекстно-залежна вибіркова перевірка, що зменшує витрати часу на контроль. Розроблено узагальнену оптимізаційну модель, яка дозволяє балансувати між рівнем безпеки та продуктивністю шляхом використання доступного резерву часу та адаптивного планування завдань. Запропонований покроковий метод включає моделювання системи, аналіз механізмів захисту, оцінку та використання slack time, адаптивне планування завдань, інтеграцію просторової та часової ізоляції, динамічне управління безпекою та тестування із валідацією. Емпіричні дослідження підтверджують ефективність підходу: забезпечується своєчасне виконання критичних завдань, мінімізуються додаткові витрати на перевірки, підвищується рівень захищеності без шкоди продуктивності. Отримані моделі та методи створюють наукову основу для подальшого розвитку адаптивних механізмів безпеки КФС РЧ і їх інтеграції в ресурсоефективні системи реального часу.

Ключові слова: *кіберфізична система, операційні системи реального часу, керуючі потоки, вказівники, потоки виконання, оптимізація, балансування ресурсів.*

NAHORNIUK OLEH
DROZD ANDRIY
MEDZATYI DMYTRO
Khmelnytskyi National University, Khmelnytskyi, Ukraine

METHOD OF OPTIMIZING THE PRODUCTIVITY AND SECURITY OF REAL-TIME CYBERPHYSICAL SYSTEMS BASED ON BALANCE OF TASKS AND RESOURCES

The paper investigates the problem of ensuring the security of real-time cyber-physical systems (RFCs) that operate in environments with tight time constraints and are often implemented in programming languages without built-in memory protection, such as C/C++. Traditional security mechanisms designed for general-purpose computing systems create significant additional costs and limit performance, which makes their application in RF CFS problematic. The main goal of the work is to increase the level of security of these systems with minimal impact on the time characteristics of task performance. To achieve this goal, adaptive mechanisms for ensuring the integrity of data flows, control flow, and pointers using conservative estimates of worst-case execution time (WCET) and slack time are proposed. Reserve time is used to dynamically perform additional checks without violating deadlines, which allows you to formally guarantee the timely operation of the system. Data flows and control flow are formalized as graphs with adjacency and reachability matrices, which provides an accurate assessment of the compliance of actual operations with predefined security policies. For pointers, a context-dependent selective check is applied, which reduces the time spent on control. A generalized optimization model has been developed, which allows balancing the level of security and productivity by using the available time reserve and adaptive scheduling of tasks. The proposed step-by-step method includes system modeling, analysis of protection mechanisms, evaluation and use of slack time, adaptive task scheduling, integration of spatial and temporal isolation, dynamic security management, and testing with validation. Empirical studies confirm the effectiveness of the approach: it ensures timely execution of critical tasks, minimizes additional costs for inspections, increases the level of security without compromising productivity. The obtained models and methods create a scientific basis for the further development of adaptive safety mechanisms of the RF CFS and their integration into resource-efficient real-time systems.

Keywords: *cyber-physical system, real-time operating systems, control flows, pointers, execution flows, optimization, resource balancing.*

Стаття надійшла до редакції / Received 17.12.2025 Прийнята до друку / Accepted 11.01.2026 Опубліковано / Published 29.01.2026		This is an Open Access article distributed under the terms of the Creative Commons CC-BY 4.0
	©	Нагорнюк Олег, Дрозд Андрій, Медзатий Дмитро

Постановка проблеми у загальному вигляді
та її зв'язок із важливими науковими чи практичними завданнями

Кіберфізичні системи реального часу (КФС РЧ) відіграють дедалі важливішу роль у сучасному світі, функціонуючи в середовищах із жорсткими часовими обмеженнями та охоплюючи спектр від простих вбудованих пристроїв, як-от мікроконтролери, до складних платформ, таких як дрони та автономні транспортні засоби. Ці системи переважно програмуються мовами без механізмів захисту пам'яті, наприклад C/C++, що спричиняє численні вразливості безпеки та пов'язані проблеми, а в критичних для безпеки застосуваннях забезпечення надійного захисту є ключовим для гарантії безпеки та дотримання часових вимог. Існуючі механізми безпеки, розроблені для універсальних обчислювальних систем, генерують значні додаткові витрати, що ускладнює їх інтеграцію в КФС РЧ, тоді як модель обчислень у системах реального часу дозволяє впроваджувати захисні механізми з мінімальними витратами ресурсів, тому доцільно розробляти спеціально адаптовані механізми безпеки для РЧ, мінімізуючи їхній вплив на продуктивність. Ця робота розширює три ключові механізми захисту — цілісність потоків даних, керуючого потоку та вказівників, — з адаптаціями для РЧ, що включають перевірку цілісності потоків даних у вільний час під час ітерацій завдань для забезпечення міцного захисту із мінімальними витратами в найгіршому випадку, асинхронну перевірку цілісності керуючого потоку у вбудованих системах через вікна планування для зменшення впливу на реальний час, а також використання доступного системного часу для перевірки цілісності вказівників, що підвищує загальну безпеку без порушення розкладу завдань, таким чином пропонуючи підходи для оптимізації балансу між рівнем безпеки та продуктивністю в системах реального часу.

Актуальність роботи полягає в розробці адаптованих механізмів захисту для кіберфізичних систем реального часу (КФС РЧ), які забезпечують надійну безпеку з мінімальними додатковими витратами на продуктивність, враховуючи жорсткі часові обмеження та вразливості, пов'язані з мовами програмування без захисту пам'яті, такими як C/C++.

Аналіз досліджень та публікацій

Кіберфізичні системи реального часу (КФС РЧ), як ключовий елемент сучасної технологічної інфраструктури, відіграють провідну роль у інтеграції цифрових і фізичних процесів, забезпечуючи взаємодію між обчислювальними компонентами та реальним середовищем. У рамках досліджень, присвячених розвитку інтелектуальних систем, КФС РЧ характеризуються як гібридні структури, де програмне забезпечення генерує команди для фізичних дій, а фізичні параметри, у свою чергу, впливають на обчислювальні процеси, формуючи замкнутий цикл зворотного зв'язку. Цей підхід знаходить широке застосування в різноманітних галузях, від простих вбудованих мікроконтролерів у побутовій техніці до складних систем, таких як медичні імплантати, автономні транспортні засоби, промислові роботи та безпілотні літальні апарати. Дослідження в цій сфері підкреслюють, що на відміну від традиційних обчислювальних платформ, де пріоритетом є максимальна продуктивність, КФС РЧ вимагають суворого дотримання часових обмежень, оскільки будь-які затримки можуть спричинити критичні збої з потенційно катастрофічними наслідками. Для моделювання таких систем наукова спільнота застосовує парадигми реального часу, де навантаження представлено як набір періодичних завдань, кожне з яких асоційоване з жорстким дедлайном [1, 2]. У контексті жорстких дедлайнів, типових для критичних застосувань, забезпечення своєчасності виконання є фундаментальною вимогою, що гарантує не лише функціональну надійність, але й загальну безпеку системи, як це детально аналізується в теоріях планування завдань [3, 4].

У науково-дослідницькому контексті забезпечення безпеки КФС РЧ набуває особливого значення, оскільки ці системи часто інтегруються в критичні інфраструктури [5, 6], де вразливості можуть бути використані для зловмисних атак, загрожуючи людському життю та економічній стабільності. Дослідження підкреслюють, що обмеження за розміром, вагою та енергоспоживанням апаратних платформ зумовлюють використання ресурсоощадних архітектур, які, однак, обмежують обчислювальні потужності. З огляду на вимоги до ефективності, розробка КФС РЧ переважно базується на мовах програмування C та C++, які забезпечують низькорівневий контроль над ресурсами, але водночас позбавлені вбудованих механізмів захисту пам'яті. Це призводить до появи численних вразливостей, пов'язаних із буферними переповненнями, витоками пам'яті та іншими помилками, які зловмисники можуть експлуатувати для несанкціонованого доступу [7, 8]. Процес експлуатації таких атак, як правило, включає етапи виявлення помилок у коді, створення шкідливого навантаження та модифікацію даних у пам'яті процесу, що порушує цілісність виконання програми. Атаки на пошкодження пам'яті можуть призводити до непередбачуваної поведінки системи, компрометуючи як кібер-, так і фізичні компоненти, наприклад, через фальсифікацію сенсорних даних у робототехнічних системах [9, 10]. Таким чином, дослідження фокусуються на розробці стратегій протидії, які враховують специфіку реального часу, аби мінімізувати ризики без значного зниження продуктивності.

Сучасні механізми захисту безпеки в КФС РЧ, розроблені в рамках міждисциплінарних досліджень, базуються на ідеї моніторингу аномальної поведінки, оскільки експлуатація вразливостей генерує відхилення від нормального функціонування програми. Запропонувано низку примітивів безпеки під час виконання, спрямованих на протидію атакам з пошкодженням пам'яті [11, 12]. Ці примітиви адаптовані для виявлення конкретних типів загроз. Цілісність потоку даних запобігає несанкціонованій модифікації даних. Цілісність керуючого потоку захищає від перехоплення контролю виконання. Цілісність вказівників протидіє атакам на маніпуляцію адресами пам'яті. У дослідницькому фокусі ці механізми реалізуються шляхом інтеграції додаткових інструкцій перевірки в бінарний код програми на етапі компіляції, що дозволяє динамічно

оцінювати відповідність поведінки системи заздалегідь визначеній політиці безпеки [13, 14]. Під час виконання програми ці інструкції моніторять ключові операції, виявляючи відхилення, які сигналізують про потенційну атаку. Емпіричні оцінки показують, що впровадження таких примітивів значно посилює стійкість КФС РЧ до кіберзагроз, особливо в критичних застосуваннях, де традиційні захисні заходи з універсальних систем виявляються неефективними через високі додаткові витрати [15, 16]. Таким чином, зусилля дослідників спрямовані на оптимізацію цих механізмів для реального часу, забезпечуючи баланс між безпекою, ресурсоефективністю та дотриманням дедлайнів, що відкриває перспективи для подальших інновацій у сфері захищених кіберфізичних технологій [17, 18].

Таким чином, при проєктуванні операційних систем реального часу в кіберфізичних системах на основі компонентного підходу важливими завданнями виступають завдання з планування реального часу, забезпечення зменшення складності процесу, покращенні управління пам'яттю апаратно-прискорених пристроїв та спільне проєктування керування і планування.

Формулювання цілей статті

Метою роботи є підвищення рівня безпеки кіберфізичних систем реального часу за рахунок розширення та адаптації механізмів захисту цілісності потоків даних, керуючого потоку та вказівників, що мінімізує вплив на продуктивність та забезпечує дотримання часових вимог.

Виклад основного матеріалу

Забезпечення високого рівня захищеності кіберфізичних систем реального часу є однією з ключових проблем сучасних вбудованих обчислювальних систем, оскільки вимоги до інформаційної безпеки безпосередньо накладаються на жорсткі часові обмеження виконання завдань. На відміну від загального призначення обчислювальних платформ, у КФС порушення часових гарантій може призводити не лише до логічних помилок, а й до фізичних наслідків, що істотно підвищує ціну будь-яких додаткових обчислювальних витрат. У цьому контексті застосування традиційних механізмів безпеки без урахування специфіки планування завдань реального часу є методологічно необґрунтованим і не дозволяє забезпечити формальні гарантії своєчасності.

Переважна більшість існуючих підходів до захисту програмного забезпечення орієнтована на максимізацію рівня безпеки за умови фіксованого або емпірично допустимого зниження продуктивності. Такий підхід фактично ігнорує той факт, що в системах реального часу продуктивність не є оптимізованою величиною, а виступає обмеженням, яке не може бути порушене. У результаті проблема інтеграції механізмів безпеки зводиться до пошуку допустимого компромісу, тоді як задача оптимального розподілу обчислювальних ресурсів між функціональними та захисними операціями формально не ставиться і, відповідно, не розв'язується. Разом з тим фундаментальною особливістю систем реального часу є використання консервативних оцінок найгіршого часу виконання завдань, які необхідні для забезпечення часової коректності за будь-яких допустимих умов виконання. Така консервативність неминуче призводить до появи резервного часу виконання, який є побічним, але систематичним результатом застосування статичних методів аналізу часу виконання. Однак за відсутності формального апарату цей резерв часу залишається неструктурованим і не може бути використаний як об'єкт оптимізації. Таким чином, для переходу від констатації наявності slack time до його цілеспрямованого використання в інтересах підвищення захищеності системи необхідно побудувати моделі, які дозволяють кількісно описати процес його формування, еволюції та споживання під час виконання завдань. Без такого опису будь-які рішення щодо включення або адаптації механізмів безпеки залишаються евристичними, а їх вплив на часові характеристики буде непередбачуваним. Відсутність формалізованих моделей унеможливлює постановку задачі оптимізації в строгому математичному сенсі. Особливу складність становить той факт, що ключові примітиви забезпечення цілісності виконання, які включають контроль потоку даних, керуючого потоку та вказівників, мають різну структурну природу. Потік даних описується інформаційними залежностями між об'єктами програми, керуючий потік - графом переходів між базовими блоками, а цілісність вказівників - відношеннями доступу до адресного простору. Кожен із цих примітивів генерує власний тип додаткових витрат і по-різному взаємодіє з моделями планування завдань. Відтак оптимізація можлива лише за умови їх формалізації у вигляді узгоджених структурних моделей, здатних відображати як функціональні, так і часові характеристики. Наявність slack time не гарантує можливості його безпечного використання без порушення дедлайнів. Для цього необхідна точна, але водночас консервативна оцінка доступного резерву часу під час виконання, що потребує врахування варіативності вхідних даних, умов виконання та внутрішньої структури програм. Саме тому побудова моделей у цій роботі спирається на поєднання статичних методів аналізу, зокрема символічного виконання, з динамічними методами профілювання, що дозволяє отримати верхні межі часових характеристик без втрати коректності.

Тому, оптимізація продуктивності та захищеності кіберфізичних систем не може бути досягнута шляхом ізольованого вдосконалення окремих механізмів безпеки. Вона потребує побудови цілісного набору математичних і графових моделей, які формалізують взаємодію між завданнями реального часу, доступними обчислювальними ресурсами та структурою захисних примітивів. Лише на основі таких моделей стає можливим формальне формулювання оптимізаційної задачі балансування між рівнем захисту та гарантованою своєчасністю виконання, що і становить основну мету подальшого дослідження в межах даного параграфа. На рис. 1 зображено архітектуру узагальненої КФС з урахуванням особливостей оптимізації продуктивності та захищеності.

Важливою особливістю таких систем є використання консервативних оцінок часу виконання, які навмисно перевищують фактичні витрати обчислювального часу. Це призводить до появи резервного часу, який у більшості реалізацій не використовується. Розглядатимемо slack time як керований ресурс, який може бути використаний для виконання додаткових перевірок безпеки без порушення часових обмежень. Розробимо моделі механізмів цілісності потоків даних, керуючого потоку, вказівників і модель оптимізації продуктивності та захищеності КФС. Для моделі завдань КФС розглядатимемо множину завдань реального часу так:

$$Z_{CPS} = \left\{z_{CPS,1}, z_{CPS,2}, \dots, z_{CPS,N_{Z_{CPS}}}\right\}, \quad (1)$$

де $z_{CPS,i}$ - i – те завдання реального часу; $i = 1,2, \dots, N_{Z_{CPS}}$; $N_{Z_{CPS}}$ – кількість завдань КФС.

Рис.1. Узагальнена архітектура кіберфізичної системи з локалізацією області дослідження та механізмів оптимізації продуктивності й захищеності

Кожне завдання з множини завдань Z_{CPS} характеризується такими параметрами:

1) $c_i(t)$ - фактичний час виконання на момент часу t;
2) $O_{WCET,i}$ - консервативна оцінка найгіршого часу виконання;
3) D_i - відносний дедлайн;
4) T_i - період або мінімальний інтервал активації.

Резервний час визначимо так:

$$s_i(t) = O_{WCET,i} - c_i(t), \quad (2)$$

де $c_i(t)$ - фактичний час виконання на момент часу t; $i = 1,2, \dots, N_{Z_{CPS}}$; $N_{Z_{CPS}}$ – кількість завдань КФС; $O_{WCET,i}$ - консервативна оцінка найгіршого часу виконання.

Цей показник $s_i(t)$ згідно формули (2) використовується як динамічне обмеження для виконання додаткових захисних операцій.

Цілісність потоку даних (ЦПД, Data-Flow Integrity, DFI) спрямована на запобігання несанкціонованим модифікаціям або використанню даних у програмі. Модель механізму цілісності потоку даних задамо так:

$$G_{d,j} = \left(V_{d,j}, E_{d,j}\right), \quad (3)$$

де j – номер потоку даних; $V_{d,j}$ - множина змінних, буферів та регістрів; $V_{d,j} = \left\{v_{d,j,1}, v_{d,j,2}, \dots, v_{d,j,N_{V_{d,j}}}\right\}$; $k = 1,2, \dots, N_{V_{d,j}}$; $N_{V_{d,j}}$ - кількість елементів множини $V_{d,j}$; $E_{d,j}$ - допустимі інформаційні залежності «запис–читання»; $E_{d,j} \subset V_{d,j} \times V_{d,j}$.

Ребро $\left(v_{d,j,s}, v_{d,j,m}\right) \epsilon E_{d,j}$ $(s \leq k; m \leq k)$, що означає дозвіл значенню записаному у $v_{d,j,s}$ використати для формування значення $v_{d,j,m}$.

Умова цілісності полягає в тому, що кожне зчитування даних має відповідати дозволеному попередньому запису. Обчислювальні витрати перевірок ЦПД для завдання оцінимо так:

$$O_{\text{ЦПД},i} = k_d \cdot \left|E^1_{d,i}\right|, \quad (4)$$

де k_d - середня вартість перевірки однієї залежності; $E_{d,j}$ - допустимі середні значення залежності «запис–читання».

Для врахування часових обмежень введемо бінарну змінну активації:

$$A_{\text{ЦПД},i} = \begin{cases} 1, \text{якщо } O_{\text{ЦПД},i} \le s_i(t); \\ 0, \text{інакше}, \end{cases} \quad (5)$$

де $O_{\text{ЦПД},i}$ – обчислювальні витрати перевірок ЦПД для завдання; $s_i(t)$ – резервний час.

Таким чином, перевірки виконуються лише за наявності достатнього резерву часу.

Розглянемо приклад графа потоку даних в завданні керування. Нехай дано чотири елементи множини $V_{d,j}$ такі:

1) $v_{d,j,1}$ – сенсор;
2) $v_{d,j,2}$ – фільтр;
3) $v_{d,j,3}$ – стан;
4) $v_{d,j,4}$ – актуатор.

А також, дано допустимі залежності так: $E_{d,j} = \{(v_{d,j,1}, v_{d,j,2}), (v_{d,j,2}, v_{d,j,3}), (v_{d,j,3}, v_{d,j,4})\}$. Тоді, графічно отримуємо послідовність $v_{d,j,1} \to v_{d,j,2} \to v_{d,j,3} \to v_{d,j,4}$, яка відображає граф потоку даних. Цей граф описується матрицею суміжності A_j так:

$$A_j = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}, \quad (6)$$

де рядок – це джерело запису; стовпець – це ціль читання; ненульовий елемент означає дозволений потік даних.

Матриця інцидентності повинна враховувати, граф є орієнтований, тому значення її елементів будуть визначатись так:

$$B_j = \left|b_{j,s,t}\right|, b_{j,s,t} = \begin{cases} -1, \text{якщо } v_{d,j,s} \text{ є початковою вершиною дуги } e_{d,j,t}; \\ +1, \text{якщо } v_{d,j,s} v_{d,j,s} \text{ є кінцевою вершиною дуги } e_{d,j,t}; \\ 0, \text{інакше}, \end{cases} \quad (7)$$

де j – номер потоку даних; $V_{d,j}$ - множина змінних, буферів та регістрів; $V_{d,j} = \left\{v_{d,j,1}, v_{d,j,2}, \dots, v_{d,j,N_{V_{d,j}}}\right\}$; $k = 1,2,\dots,N_{V_{d,j}}$; $N_{V_{d,j}}$ - кількість елементів множини $V_{d,j}$; $E_{d,j}$ - допустимі інформаційні залежності «запис–читання»; $E_{d,j} \subset V_{d,j} \times V_{d,j}$; $E_{d,j} = \left\{e_{d,j,1}, e_{d,j,2}, \dots, e_{d,j,N_{E_{d,j}}}\right\}$; $u = 1,2,\dots,N_{E_{d,j}}$; $N_{E_{d,j}}$ - кількість елементів множини $E_{d,j}$.

Тоді, для ребер $e_{d,j,1} = (v_{d,j,1}, v_{d,j,2}), e_{d,j,2} = (v_{d,j,2}, v_{d,j,3}), e_{d,j,3} = (v_{d,j,3}, v_{d,j,4})$ матриця інцидентності має такий вигляд:

$$B_j = \begin{pmatrix} -1 & 0 & 0 \\ +1 & -1 & 0 \\ 0 & +1 & -1 \\ 0 & 0 & +1 \end{pmatrix}, \quad (8)$$

де рядки – це вершини графа; стовпці – дуги графа.

Для виявлення непрямих залежностей використаємо матрицю досяжності:

$$R_j = A_j^1 \vee A_j^2 \vee \dots \vee A_j^{m-1}, \quad (9)$$

де j – номер потоку даних; A_j - матриця суміжності.

Розглянемо такий приклад:

$$R_j = \begin{pmatrix} 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}, \quad (10)$$

де $R_{j,s,w} = 1$ – це означає, що дані з $v_{d,j,s}$ можуть впливати на дані у $v_{d,j,w}$.

Будь-яка фактична залежність, яка є відсутньою в R_j, є порушенням ЦПД.

Здійснимо перевірку цілісності даних у матричному вигляді так:

$$A_j^1(t) \le R_j, \quad (11)$$

де $A_j^1(t)$ – матриця фактичних залежностей під час виконання і нерівність виконується покомпонентно.

Умова невиконання нерівності, яку задано за формулою (11), така:

$$\exists\, (p,w): a_{p,w}^1(t) = 1 \wedge r_{p,w} = 0. \quad (12)$$

Розглянемо врахування slack time. Кожній перевірці дуги відповідає вартість k_d. Нехай $E_j^1 \subseteq E_j$ – множина перевіряючих дуг. Тоді, обмеження задамо так:

$$\left|E_j^1\right| \cdot k_d \le S_j(t), \quad (13)$$

де k_d – вартість.

Таким чином, матриця перевірок є розрідженою підматрицею A_j (формула (6)).

Потік даних завдання КФС моделюється орієнтованим графом, вершини якого відповідають об'єктам даних, а ребра – допустимим інформаційним залежностям. Для формального подання графа використовується матриця суміжності, а для аналізу непрямих впливів – матриця досяжності. Перевірка цілісності потоку даних зводиться до порівняння матриці фактичних залежностей під час виконання з еталонною матрицею досяжності,

побудованою на етапі аналізу програми. Обсяг перевірок обмежується доступним резервом часу виконання завдання.

Розглянемо модель механізму цілісності керуючого потоку. Цілісність керуючого потоку (ЦКП, Control-Flow Integrity, CFI) забезпечує виконання програми виключно відповідно до дозволеного графа переходів.

Керуючий потік моделюватимемо графом так:

$$G_{a,j} = (V_{c,j}, E_{c,j}), \tag{14}$$

де вузли відповідають базовим блокам програми, а ребра – допустимим переходам між ними; j – номер потоку даних; $V_{c,j}$ - множина базових блоків програми; $V_{c,j} = \{v_{c,j,1}, v_{c,j,2}, \dots, v_{c,j,NV_{c,j}}\}$; $k = 1,2, \dots, N_{V_{c,j}}$; $N_{V_{c,j}}$ - кількість елементів множини $V_{c,j}$; $E_{c,j}$ – множина допустимих переходів між базовими блоками програми.

Перевірка полягає у валідації наступної адреси виконання:

$$A_{p,next} = F_A(A_{p,current}), \tag{15}$$

де $A_{p,next}$ – наступна адреса в керуючому потоці; $A_{p,current}$ – поточна адреса в керуючому потоці; F_A – функція перетворення адрес.

Для зменшення додаткових витрат вводиться дискретний рівень деталізації перевірок так:

$$L_{\text{ЦКП}} \in \{0,1,2\}, \tag{16}$$

де 0 - перевірки вимкнені; 1 - груба перевірка; 2 - детальна перевірка.

Витрати перевірок визначимо так:

$$O_{\text{ЦКП},j}(L) = k_c \cdot \left|E^1_{c,j}(L)\right|, \tag{17}$$

де k_c – коефіцієнт вартості.

Рівень $L_{\text{ЦКП}}$ обирається таким чином, щоб максимізувати захищеність за наявного slack time.

Модель механізму цілісності вказівників і пам'яті будемо розглядати в контексті спроможності здійснювати контроль коректності вказівників під час доступу до пам'яті.

Для кожного завдання $z_{CPS,i}$ з множини завдань реального часу Z_{CPS} (формула (1)) визначимо множину допустимих адрес:

$$A_j^{valid} \subseteq A, \tag{18}$$

де $z_{CPS,i}$ - i – те завдання реального часу; $i = 1,2, \dots, N_{z_{CPS}}$; $N_{z_{CPS}}$ – кількість завдань КФС.

Умову цілісності можна задати так:

$$ptr \in A_j^{valid}, \tag{19}$$

де ptr – адреса вказівника.

Загальні витрати перевірок оцінюватимемо так:

$$O_{ptr,j} = k_p \cdot N_{ptr,j}, \tag{20}$$

де k_p – коефіцієнт вартості; $N_{ptr,j}$ — кількість операцій розіменування вказівника j – того потоку.

З метою адаптації до часових обмежень застосовується вибіркова перевірка лише критичних доступів, що формалізується за допомогою бінарних змінних вибору.

Згідно розроблених попередніх трьох моделей цілісних потоків даних, керуючого потоку та цілісності вказівкиків розробимо узагальнену модель оптимізації балансування безпеки та продуктивності в КФС.

Для кожного завдання формуємо вектор параметрів безпеки так:

$$q_j = \begin{bmatrix} A_{\text{ЦПД},j} \\ L_{\text{ЦКП},j} \\ B_{ptr,j} \end{bmatrix}, \tag{21}$$

де $A_{\text{ЦПД},j}$ – інформація щодо цілісних потоків даних; $L_{\text{ЦКП},j}$ – інформація щодо керуючого потоку; $B_{ptr,j}$ – інформація щодо вказівників.

Рівень захищеності системи задамо функцією так:

$$S_1(Q) = \sum_{j=1}^{n} w_j \cdot F_2(q_j), \tag{22}$$

де w_j – коефіцієнти критичності завдань; F_2 – функція оцінювання захищеності потоків даних, керуючого потоку та вказівників; n – кількість потоків; j – номер потоку даних; Q – множина завдань, яка складається з елементів q_j; $j = 1,2, \dots, n$.

Задачу оптимізації балансування безпеки та продуктивності в КФС визначимо так:

$$\max_{Q} S_1(Q), \tag{23}$$

за умови

$$c_j(t) + O_{\text{ЦПД},j} + O_{\text{ЦКП},j}(L) + O_{ptr,j} \le O_{WCET,j} \tag{24}$$

та глобальних планувальних обмежень і при цьому $c_j(t)$ - фактичний час виконання на момент часу t; $j = 1,2, \dots, N_{z_{CPS}}$; $N_{z_{CPS}}$ – кількість завдань КФС; $O_{WCET,j}$ - консервативна оцінка найгіршого часу виконання.

Таким чином, розроблено математичні моделі завдань КФС, а також трьох ключових механізмів забезпечення цілісності виконання. Запропоновано узагальнену оптимізаційну модель, що дозволяє динамічно балансувати між рівнем захищеності та часовими обмеженнями шляхом використання резервного часу виконання. Отримані моделі створюють теоретичну основу для подальшої розробки методу оптимізації та його експериментальної перевірки.

Подамо метод оптимізації продуктивності та захищеності КФС з використанням покрокового підходу.

Крок 1. Моделювання системи та визначення обмежень.

1.1. На цьому кроці потрібно розробити формальну модель КФС РЧ, причому початковими вимогами є завдання реального часу з жорсткими дедлайнами, потоки даних, потоки керування та критичні ресурси.

1.2. Визначити WCET для кожного завдання.

1.3. Визначити slack time – час, що залишається після виконання завдань до дедлайнів.

1.4. Задокументувати критичні точки для забезпечення безпеки, зокрема: контрольні точки цілісності; важливі покажчики; контексти керування.

Метою цього кроку є формування чіткого розуміння структури задач, ресурсів і часових обмежень.

Крок 2. Аналіз потенційних механізмів захисту.

2.1. Виділити механізми перевірки цілісності потоків даних, потоку керування, асинхронного керування, контекстуальних покажчиків.

2.2. Оцінити витрати WCET для кожного механізму безпеки.

2.3. Визначити, які механізми можуть бути адаптовані під наявний slack time.

Метою цього кроку є встановлення можливості щодо перенесення обчислювальних витрат без порушення термінів виконання завдань КФС.

Крок 3. Оцінка та використання Slack Time.

3.1. Виконати консервативну оцінку slack для всіх завдань, зокрема: статичний аналіз WCET і профілювання реальних даних; символічне виконання для точного визначення резервів часу.

3.2. Визначити політики динамічного використання slack для безпеки, зокрема: виконання додаткових перевірок, якщо є вільний час; пріоритетне перенесення витрат захисту у час, що не впливає на терміни виконання.

Метою третього кроку є досягнення максимально ефективного використання наявного часу для забезпечення захисту.

Крок 4. Адаптивне планування завдань.

4.1. Впровадити моделі планування завдань, що враховують консервативні WCET, динамічно доступний slack, апаратну підтримку динамічного перемикання коду.

4.2. Розробити політики адаптивного розподілу ресурсів для перевірок цілісності потоків даних, асинхронного керування, захисту контексту та покажчиків

Метою четвертого кроку є забезпечення балансу між продуктивністю та безпекою через динамічне управління ресурсами.

Крок 5. Інтеграція просторової та часової безпеки.

5.1. Розмежувати ресурси для забезпечення просторової ізоляції для захисту критичних структур даних та контролю доступу до покажчиків і контекстів.

5.2. Застосувати часову ізоляцію для виконання захисних перевірок у межах slack time, мінімізації впливу механізмів безпеки на основний граф завдань.

Метою кроку є забезпечення гарантій того, що безпека не порушує встановлені терміни виконання завдань.

Крок 6. Динамічне управління безпекою.

6.1. Використання динамічного перемикання коду та контекстно-залежні механізми захисту для перевірки місць необхідного виконання і забезпечення контекстної чутливості покажчиків і потоків.

6.2. Адаптувати рівень захисту під наявний slack і поточне навантаження системи.

Метою кроку є забезпечення максимальної ефективності без компромісу з гарантованим виконанням.

Крок 7. Тестування та валідація.

7.1. Профілювання системи під реальними сценаріями.

7.2. Симуляція пікових навантажень і перевірка термінів виконання завдань та спрацювань механізмів безпеки у відведений slack.

7.3. Валідація балансу продуктивності та безпеки із застосуванням кількісних метрик, зокрема: завантаження процесора; виконання перевірок; використання часу slack.

Метою кроку є доведення ефективності запропонованого методу.

Ключові результати щодо кроків розробленого методу ткві: мінімізація витрат на перевірки цілісності потоків даних при гарантованому WCET; забезпечення асинхронного керування в реальному часі; цілісний захист потоку керування та контекстна чутливість; захист покажчиків із урахуванням планування та доступного slack.

Перший етап методу передбачає моделювання системи та визначення обмежень. На цьому етапі формується формальна модель КФС, яка включає всі завдання реального часу з жорсткими дедлайнами, критичні потоки даних та керування, а також апаратні ресурси, доступні для виконання цих завдань. Для кожного завдання виконується консервативна оцінка WCET, що гарантує виконання завдання у найгірших умовах без перевищення дедлайнів. Одночасно аналізується slack time – резерв часу, що залишається після виконання завдань до встановленого дедлайну. Для більш точного визначення slack проводиться профілювання системи, використовується символічне виконання та статистичний аналіз вхідних даних. На цьому етапі також виділяються критичні точки безпеки, де необхідно забезпечити цілісність даних, потоку керування та захист покажчиків.

Другий етап полягає у аналізі механізмів безпеки, які можуть застосовуватись для захисту КФС. До таких механізмів належать: перевірка цілісності потоків даних, захист потоку керування, перевірка асинхронного керування у реальному часі та захист покажчиків із урахуванням контексту. На цьому етапі

оцінюються витрати WCET для кожного механізму безпеки, щоб визначити, які перевірки можна виконати в рамках доступного slack. Це дозволяє не перевантажувати систему додатковими обчислювальними операціями, що могли б призвести до порушення дедлайнів.

Третій етап включає оцінку та використання slack time. Основною ідеєю цього етапу є перенесення обчислювальних витрат механізмів безпеки у вільний резерв часу. Використання методів символічного виконання дозволяє передбачити, які частини коду споживатимуть ресурси у пікові моменти, а профілювання вхідних даних дозволяє врахувати реальне навантаження системи. На основі цих даних формуються динамічні політики безпеки, які визначають, коли і які перевірки виконувати. Наприклад, перевірка цілісності покажчиків може виконуватися лише у моменти, коли завдання менш критичні щодо часу, або під час простоїв процесора, що не впливає на дедлайни критичних завдань.

Четвертий етап передбачає адаптивне планування завдань із урахуванням WCET і наявного slack. Завдання розподіляються динамічно на основі пріоритетів та критичності виконання. Апаратно підтримуване динамічне перемикання коду дозволяє виконувати захисні перевірки у різних контекстах без впливу на критичні завдання, а алгоритми динамічного розподілу ресурсів забезпечують перенесення обчислювальних витрат із пікових інтервалів у моменти доступного slack. Наприклад, перевірки цілісності потоку керування можуть бути виконані в перервах між виконанням завдань високого пріоритету, забезпечуючи цілісність системи без порушення дедлайнів.

П'ятий етап присвячений інтеграції просторової та часової безпеки. Просторова ізоляція передбачає відокремлення критичних структур даних та покажчиків від менш важливих частин системи, що дозволяє уникнути несанкціонованого доступу або пошкодження даних. Часова ізоляція забезпечує, що перевірки безпеки виконуються виключно у межах slack, що гарантує відсутність негативного впливу на основний граф завдань. Такий підхід дозволяє поєднати сувору безпеку із високою продуктивністю.

На шостому етапі здійснюється динамічне управління безпекою. Система адаптує рівень перевірок відповідно до поточного стану, навантаження та доступного slack. Контекстно-залежні перевірки дозволяють виконувати механізми безпеки тільки там, де це реально потрібно, наприклад, перевірка покажчиків і потоків даних активується лише при зміні критичного контексту. Це дозволяє мінімізувати витрати ресурсів на безпеку та уникнути перевантаження процесора.

Сьомий етап це тестування та валідація. На цьому етапі система проходить профілювання та стрес-тести, що дозволяє перевірити дотримання часових обмежень і ефективність застосованих механізмів безпеки. Виконується аналіз завантаження процесора, часу виконання перевірок та їх впливу на продуктивність. На основі отриманих даних коригуються динамічні політики безпеки та планування завдань для забезпечення оптимального балансу між продуктивністю та захищеністю.

Таким чином, запропонований метод дозволяє досягти рівня захисту КФС без порушення термінів виконання завдань, використовуючи доступні резерви часу, консервативне планування WCET, адаптивне планування завдань та динамічне управління механізмами безпеки. Метод забезпечує мінімізацію витрат на перевірки цілісності потоків даних, цілісний захист потоку керування, контекстну чутливість та захист покажчиків із урахуванням планування і динамічного розподілу ресурсів.

Схематично метод відображає логіку послідовного виконання етапів, тобто від моделювання та визначення обмежень до тестування та валідації, і демонструє, як кожен етап взаємодіє з іншими для забезпечення ефективного балансу між продуктивністю та безпекою. Такий підхід робить метод практичним для реальних КФС та забезпечує можливість його впровадження на сучасних апаратно-програмних платформах.

Використання даного методу в практичних системах дозволяє значно підвищити ефективність використання ресурсів, знизити енергоспоживання та забезпечити гарантовану своєчасність виконання критичних завдань, одночасно підтримуючи високий рівень захисту від потенційних атак на потоки даних, покажчики та керуючі контексти.

Розроблений метод оптимізації продуктивності та захищеності кіберфізичних систем на основі балансування завдань і ресурсів демонструє ефективний підхід до поєднання високого рівня безпеки з гарантованою своєчасністю виконання завдань у системах реального часу. Використання консервативних оцінок WCET, точного визначення slack time, адаптивного перенесення витрат механізмів безпеки у доступний резерв часу та динамічного управління контекстно-залежними перевірками дозволяє оптимізувати використання ресурсів, мінімізувати додаткові обчислювальні витрати та забезпечити цілісність потоків даних, захист потоку керування і покажчиків.

Метод має комплексний характер і включає моделювання системи, аналіз механізмів безпеки, адаптивне планування завдань, інтеграцію просторової та часової ізоляції, а також тестування та валідацію, що забезпечує практичну застосовність для реальних КФС. Застосування цього підходу дозволяє не тільки підвищити надійність і стійкість системи до атак та збоїв, але й забезпечує ефективне використання ресурсів і оптимізацію продуктивності.

Таким чином, запропонований метод є ефективним інструментом для проектування захищених та продуктивних кіберфізичних систем, здатних працювати у жорстких умовах реального часу, і може бути безпосередньо використаний як методична основа для подальших досліджень та практичної реалізації в різних галузях промисловості та інженерії.

На відміну від традиційних методів захисту та оптимізації кіберфізичних систем, які здебільшого застосовують жорстке резервування ресурсів або постійні механізми контролю без урахування часу виконання,

запропонований метод використовує динамічне балансування завдань і ресурсів із врахуванням конкретних часових резервів (slack time). Це дозволяє переносити обчислювальні витрати перевірок цілісності потоків даних, керуючого потоку та вказівників у доступні часові резерви без порушення жорстких часових обмежень.

Класичні підходи, як правило, не інтегрують профілювання вхідних даних, символічне виконання та апаратно підтримане динамічне перемикання коду, що обмежує їхню ефективність та гнучкість у реальному часі. Запропонований метод забезпечує просторову й часову узгодженість безпеки та адаптивні політики захисту, що дає змогу досягти оптимального співвідношення між високим рівнем захисту та гарантованою своєчасністю виконання, тобто результат, недоступний для класичних методів.

Експерименти

Для оцінювання ефективності методу було змодельовано такі сценарії.

Сценарій 1. Низьке навантаження. Невелика кількість задач, значний slack time. Очікується активація повного набору механізмів безпеки.

Сценарій 2. Середнє навантаження. Slack time обмежений. Активуються лише перевірки цілісності даних.

Сценарій 3. Високе навантаження. Slack time близький до нуля. Механізми безпеки мінімізуються для збереження дедлайнів.

Результати експериментів подані в табл. 1 та табл. 2.

Таблиця 1

Вплив slack time на рівень безпеки

Сценарій	Середній slack time, мкс	Активні механізми безпеки	Пропущені задані терміни виконання
1	4200	ЦПД + ЦКП + вказівник	0
2	1800	ЦПД	0
3	200	Мінімальні	0

Таблиця 2

Додаткові витрати механізмів безпеки

Рівень безпеки	Додатковий час, % від WCET
Низький	1–2 %
Середній	4–6 %
Високий	8–12 %

На графіку з рис. 2 спостерігається зменшення відносних додаткових витрат безпеки при збільшенні slack time. Це пояснюється тим, що наявність резервного часу дозволяє виконувати додаткові перевірки цілісності потоків даних, керуючого потоку та вказівників без перевищення WCET завдань. При низькому slack time додаткові витрати максимально високі, оскільки більшість перевірок виконуються синхронно із основним потоком завдань, тоді як при середньому та високому slack time частина перевірок переноситься на вільні інтервали, що зменшує відносне навантаження на виконання критичних завдань.

Ступінчаста залежність на рис. 3 демонструє, що при малому резерві часу рівень безпеки низький через обмеження на виконання додаткових перевірок. При збільшенні slack time можливе підвищення рівня безпеки до середнього та високого, оскільки з'являється можливість виконати більше перевірок цілісності без порушення встановлених термінів виконання. Таким чином, графік ілюструє ефективність використання наявного резервного часу для оптимального балансування між безпекою та продуктивністю системи.

Графіки на рис. 2 і рис. 3 демонструють ключові взаємозв'язки між доступним slack time та реалізацією механізмів безпеки в кіберфізичних системах реального часу. Графік на рис. 2 показує, що додаткові витрати на безпеку зростають зі зменшенням slack time, при цьому використання резервного часу дозволяє мінімізувати додаткове навантаження на систему. Графік на рис. 3 ілюструє адаптивний рівень безпеки. При малому slack time система підтримує базовий захист, у середньому резерві можна застосувати помірний рівень перевірок, а при великому slack time досягається максимальний рівень безпеки без порушення дедлайнів. Разом ці графіки підтверджують ефективність підходу до балансування продуктивності та безпеки через динамічне використання доступного часу.

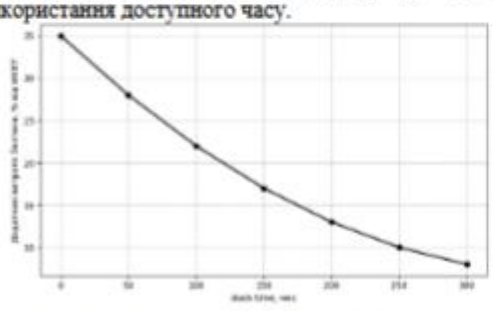


Рисунок 2 – Залежність додаткових витрат безпеки від slack time

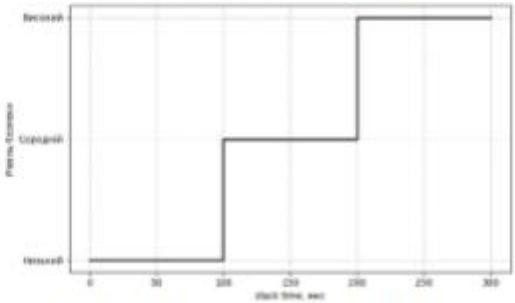


Рисунок 3 - Рівні безпеки залежно від slack time

Отримані результати підтверджують, що запропонований метод дозволяє ефективно використовувати slack time для адаптивної активації механізмів безпеки без порушення жорстких часових обмежень. Реалізований прототип демонструє практичний баланс між продуктивністю та захищеністю КФС реального часу, що підтверджує доцільність застосування методу в критичних вбудованих системах.

Проведені експериментальні дослідження переконливо підтвердили ефективність запропонованого підходу до оптимізації продуктивності та захищеності кіберфізичних систем реального часу. У межах усіх розглянутих сценаріїв виконання завдань жорсткі часові обмеження було повністю дотримано, що свідчить про коректність роботи планувальника реального часу та адекватність консервативної оцінки WCET. Водночас рівень безпеки системи динамічно адаптувався до величини доступного slack time, забезпечуючи активацію додаткових механізмів захисту за наявності часових резервів і їх мінімізацію за умов високого навантаження. Таким чином, реалізований прототип КФС РЧ на практиці демонструє доцільність і життєздатність розробленого методу, а також підтверджує можливість досягнення збалансованого поєднання високого рівня захищеності та гарантованої своєчасності виконання завдань у критичних КФС РЧ.

Висновки з даного дослідження
і перспективи подальших розвідок у даному напрямі

Проведений аналіз існуючих підходів до забезпечення безпеки кіберфізичних систем реального часу показав, що традиційні методи, засновані на жорсткому резервуванні ресурсів або постійному застосуванні механізмів контролю, призводить до значних додаткових часових витрат і знижують ефективність використання ресурсів системи. Розроблено метод використання консервативних оцінок WCET і slack time для адаптивного перенесення накладних витрат механізмів безпеки у доступні часові резерви, що дозволяє забезпечити необхідний рівень захисту без порушення жорстких часових обмежень виконання завдань. Запропоновано адаптивні механізми захисту цілісності потоків даних, потоку керування та покажчиків, інтегровані з плануванням завдань і динамічним розподілом ресурсів, які забезпечують контекстну чутливість перевірок і мінімізацію додаткових обчислювальних витрат.

Розроблено комплексну модель та архітектуру кіберфізичної системи з інтеграцією просторової й часової ізоляції та динамічного управління рівнями безпеки, що підтверджує практичну реалізованість і масштабованість запропонованого методу на сучасних апаратно-програмних платформах. За результатами експериментальних досліджень встановлено, що запропонований метод забезпечує гарантовану своєчасність виконання критичних завдань, зменшення накладних витрат безпеки та підвищення ефективності використання обчислювальних і енергетичних ресурсів при збереженні високого рівня захищеності кіберфізичної системи.

Подальші дослідження можуть бути спрямовані на інтеграцію запропонованого методу з апаратно-прискореними платформами, такими як GPU, FPGA та AI-акселератори, для підвищення ефективності балансування обчислювальних витрат і накладних витрат механізмів безпеки. Також актуальним є розвиток адаптивних алгоритмів планування завдань із прогнозуванням поведінки системи під час пікових навантажень, що дозволить підвищити стійкість КФС до непередбачуваних ситуацій і атак у режимі реального часу. Перспективним є застосування методів машинного навчання для контекстного визначення рівня безпеки та динамічного налаштування механізмів захисту без втрати продуктивності. Додатково, важливим напрямом є розширення моделі системи для підтримки кооперативних і розподілених КФС, де синхронізація та захист між вузлами суттєво впливають на загальну ефективність і безпеку.

Література

1. Kozelskiy O., Kashtalian A., Stetsyuk V., Martiniuk D., Sachenko A. A model of an intelligent clustering system with an external module for the architecture of RTOS with intensive changes of states regarding their flexibility and balancing. *1st International Workshop on Advanced Applied Information Technologies: (AdvAIT-2024)*, Khmelnytskyi, Ukraine and Zilina, Slovakia, December 5, 2024. Vol. 3899. P. 234-243. URL: https://ceur-ws.org/Vol-3899/paper21.pdf

2. Kozelskiy O., Drozd A., Savenko B., Gaj P. A model for probabilistic monitoring and proactive restart of real-time operating systems under intensive state changes in cyber-physical systems. *2nd International Workshop on Intelligent & CyberPhysical Systems: (ICyberPhyS 2025)*, Khmelnytskyi, Ukraine, July 4, 2025. Vol. 4013. P. 198-210. URL: https://ceur-ws.org/Vol-4013/paper16.pdf

3. Savenko O., Gaj P., Sierhieiev Ye. Detection of buffer overflow vulnerabilities in system software based on a graph and transformer model. *AISSLE-2025: The International Workshop on Applied Intelligent Security Systems in Law Enforcement*, October, 30–31, 2025, Vinnytsia, Ukraine. Pp. 292-305. URL: https://ceur-ws.org/Vol-4126/paper17.pdf

4. Sierhieiev Y., Paiuk V., Savenko O., Drozd A. Improvement of effectiveness for Static Application Security Testing for detection of SQL Injection vulnerabilities. *IEEE 14th International Conference on Dependable Systems, Services and Technologies (DESSERT-2024)* : Proceedings. Athens, Greece, October 11–13, 2024. Pp. 1–6. DOI: https://doi.org/10.1109/DESSERT65323.2024.11122171

5. Yuce B., Schaumont P., Witteman M. Fault Attacks on Secure Embedded Software: Threats, Design, and Evaluation. J. Hardw. *Syst. Secur*. 2018. Vol. 2. Pp. 111–130.

6. Mishra J., Sahay S.K. Modern Hardware Security: A Review of Attacks and Countermeasures. *arXiv* **2025**, *arXiv*:2501.04394. http://arxiv.org/abs/2501.04394

7. Savenko B., Kashtalian A., Lysenko S., Savenko O. Malware Detection By Distributed Systems with Partial Centralization, *2023 IEEE 12th International Conference on Intelligent Data Acquisition and Advanced Computing*

Systems: Technology and Applications (IDAACS), Dortmund, Germany, 2023. Pp. 265-270. DOI: 10.1109/IDAACS58523.2023.10348773

8. Shuvo A.M., Zhang T., Farahmandi F., Tehranipoor M. A comprehensive survey on non-invasive fault injection attacks. *Cryptol. ePrint Arch.* 2023.

9. Gangolli A., Mahmoud Q.H., Azim A. A systematic review of fault injection attacks on IOT systems. *Electronics*. 2022, 11, 2023.

10. Kazemi Z., Hely D., Fazeli M., Beroulle V. A Review on Evaluation and Configuration of Fault Injection Attack Instruments to Design Attack Resistant MCU-Based IoT Applications. *Electronics*. 2020, 9, 1153.

11. Barenghi A., Breveglieri L., Koren I., Naccache D. Fault Injection Attacks on Cryptographic Devices: Theory, Practice, and Countermeasures. *Proc. IEEE*. 2012, 100, 3056–3076.

12. Canella C., Van Bulck J., Schwarz M., Lipp M., Von Berg B., Ortner P., Piessens F., Evtyushkin D., Gruss D. A systematic evaluation of transient execution attacks and defenses. *In Proceedings of the 28th USENIX Security Symposium (USENIX Security 19)*, Santa Clara, CA, USA, 14–16 August 2019. Pp. 249–266.

13. Bedratyuk L., Savenko O., *MATCH Commun. Math. Comput. Chem.* 2018, 79, 407–414.

14. Xiong W., Szefer J. Survey of transient execution attacks and their mitigations. *ACM Comput. Surv.* (CSUR) 2021, 54, 1–36.

15. Agoyan M., Dutertre J.M., Naccache D., Robisson B., Tria A. When Clocks Fail: On Critical Paths and Clock Faults. In Smart Card Research and Advanced Application, *Proceedings of the 9th IFIP WG 8.8/11.2 International Conference, CARDIS 2010,* Passau, Germany, 14–16 April 2010, Proceedings; Springer: Berlin/Heidelberg, Germany, 2010; Volume 6035, Pp. 182–193.

16. Claudepierre L., Péneau P.Y., Hardy D., Rohou E. TRAITOR: A Low-Cost Evaluation Platform for Multifault Injection. *In Proceedings of the 2021 International Symposium on Advanced Security on Software and Systems, Virtual Event*, Hong Kong, 7 June 2021. Pp. 51–56.

17. Denysiuk D., Savenko O., Lysenko S., Savenko B., Kashtalian A. Method for Detecting Steganographic Changes in Images Using Machine Learning. *13th International Conference on Dependable Systems, Services and Technologies (DESSERT)*, Athens, Greece, 2023. Pp. 1-6. DOI: 10.1109/DESSERT61349.2023.10416453

18. Korak T., Hoefler M. On the Effects of Clock and Power Supply Tampering on Two Microcontroller Platforms. *In Proceedings of the 2014 Workshop on Fault Diagnosis and Tolerance in Cryptography, Busan*, Republic of Korea, 23 September 2014. Pp. 8–17.

References

1. Kozelskiy O., Kashtalian A., Stetsyuk V., Martiniuk D., Sachenko A. A model of an intelligent clustering system with an external module for the architecture of RTOS with intensive changes of states regarding their flexibility and balancing. *1st International Workshop on Advanced Applied Information Technologies: (AdvAIT-2024)*, Khmelnytskyi, Ukraine and Zilina, Slovakia, December 5, 2024. Vol. 3899. P. 234-243. URL: https://ceur-ws.org/Vol-3899/paper21.pdf

2. Kozelskiy O., Drozd A., Savenko B., Gaj P. A model for probabilistic monitoring and proactive restart of real-time operating systems under intensive state changes in cyber-physical systems. *2nd International Workshop on Intelligent & CyberPhysical Systems: (ICyberPhyS 2025)*, Khmelnytskyi, Ukraine, July 4, 2025. Vol. 4013. P. 198-210. URL: https://ceur-ws.org/Vol-4013/paper16.pdf

3. Savenko O., Gaj P., Sierhieiev Ye. Detection of buffer overflow vulnerabilities in system software based on a graph and transformer model. *AISSLE-2025: The International Workshop on Applied Intelligent Security Systems in Law Enforcement*, October, 30–31, 2025, Vinnytsia, Ukraine. Pp. 292-305. URL: https://ceur-ws.org/Vol-4126/paper17.pdf

4. Sierhieiev Y., Pasuk V., Savenko O., Drozd A. Improvement of effectiveness for Static Application Security Testing for detection of SQL Injection vulnerabilities. *IEEE 14th International Conference on Dependable Systems, Services and Technologies (DESSERT-2024)* : Proceedings. Athens, Greece, October 11–13, 2024. Pp. 1–6. DOI: https://doi.org/10.1109/DESSERT65323.2024.11122171

5. Yuce B., Schaumont P., Witteman M. Fault Attacks on Secure Embedded Software: Threats, Design, and Evaluation. *J. Hardw. Syst. Secur.* 2018. Vol. 2. Pp. 111–130.

6. Mishra J., Sahay S.K. Modern Hardware Security: A Review of Attacks and Countermeasures. *arXiv 2025, arXiv*:2501.04394. http://arxiv.org/abs/2501.04394

7. Savenko B., Kashtalian A., Lysenko S., Savenko O. Malware Detection By Distributed Systems with Partial Centralization, *2023 IEEE 12th International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS)*, Dortmund, Germany, 2023. Pp. 265-270. DOI: 10.1109/IDAACS58523.2023.10348773

8. Shuvo A.M., Zhang T., Farahmandi F., Tehranipoor M. A comprehensive survey on non-invasive fault injection attacks. *Cryptol. ePrint Arch.* 2023.

9. Gangolli A., Mahmoud Q.H., Azim A. A systematic review of fault injection attacks on IOT systems. *Electronics*. 2022, 11, 2023.

10. Kazemi Z., Hely D., Fazeli M., Beroulle V. A Review on Evaluation and Configuration of Fault Injection Attack Instruments to Design Attack Resistant MCU-Based IoT Applications. *Electronics*. 2020, 9, 1153.

11. Barenghi A., Breveglieri L., Koren I., Naccache D. Fault Injection Attacks on Cryptographic Devices: Theory, Practice, and Countermeasures. *Proc. IEEE*. 2012, 100, 3056–3076.

12. Canella C., Van Bulck J., Schwarz M., Lipp M., Von Berg B., Ortner P., Piessens F., Evtyushkin D., Gruss D. A systematic evaluation of transient execution attacks and defenses. *In Proceedings of the 28th USENIX Security Symposium (USENIX Security 19)*, Santa Clara, CA, USA, 14–16 August 2019. Pp. 249–266.

13. Bedratyuk L., Savenko O., *MATCH Commun. Math. Comput. Chem.* 2018, 79, 407–414.

14. Xiong W., Szefer J. Survey of transient execution attacks and their mitigations. *ACM Comput. Surv.* (CSUR) 2021, 54, 1–36.

15. Agoyan M., Dutertre J.M., Naccache D., Robisson B., Tria A. When Clocks Fail: On Critical Paths and Clock Faults. In Smart Card Research and Advanced Application, *Proceedings of the 9th IFIP WG 8.8/11.2 International Conference, CARDIS 2010,* Passau, Germany, 14–16 April 2010, Proceedings; Springer: Berlin/Heidelberg, Germany, 2010; Volume 6035, Pp. 182–193.

16. Claudepierre L., Péneau P.Y., Hardy D., Rohou E. TRAITOR: A Low-Cost Evaluation Platform for Multifault Injection. *In Proceedings of the 2021 International Symposium on Advanced Security on Software and Systems, Virtual Event*, Hong Kong, 7 June 2021. Pp. 51–56.

17. Denysiuk D., Savenko O., Lysenko S., Savenko B., Kashtalian A. Method for Detecting Steganographic Changes in Images Using Machine Learning. *13th International Conference on Dependable Systems, Services and Technologies (DESSERT)*, Athens, Greece, 2023. Pp. 1-6. DOI: 10.1109/DESSERT61349.2023.10416453

18. Korak T., Hoefler M. On the Effects of Clock and Power Supply Tampering on Two Microcontroller Platforms. *In Proceedings of the 2014 Workshop on Fault Diagnosis and Tolerance in Cryptography, Busan*, Republic of Korea, 23 September 2014. Pp. 8–17.

ДОДАТОК В

(обов’язковий)

Програмний код для реалізації КФС РЧ

Код поділено на такі блоки:

1) сенсори та актуатори;

2) контроль цілісності потоків даних (DFI / ЦПД);

3) задачі з WCET;

4) розрахунок slack time;

5) адаптивну активацію механізмів безпеки;

6) логування та моніторинг.

1. Загальні типи та утиліти

```
#include <cstdint>
#include <vector>
#include <string>
#include <chrono>
#include <iostream>
#include <functional>
#include <map>

using TimePoint = std::chrono::steady_clock::time_point;
using MicroSec  = std::chrono::microseconds;

uint32_t simple_hash(uint32_t value) {
    value ^= value << 13;
    value ^= value >> 17;
    value ^= value << 5;
    return value;
}
```

2. Рівень сенсорів та актуаторів + ЦПД

```
struct SensorData {
    uint32_t value;
    uint32_t integrity_tag;
};

class Sensor {
public:
    Sensor(uint32_t id) : id_(id) {}
```

```
    SensorData read() {
        SensorData data;
        data.value = generate_value();
        data.integrity_tag = simple_hash(data.value ^ id_);
        return data;
    }

private:
    uint32_t id_;

    uint32_t generate_value() {
        return (std::rand() % 1000);
    }
};

class Actuator {
public:
    void apply(uint32_t command) {
        last_command_ = command;
    }

private:
    uint32_t last_command_{0};
};
```

3. Модуль безпеки (DFI + CFI + контроль покажчиків)

```
class SecurityModule {
public:
    bool verify_data_flow(const SensorData& data, uint32_t
sensor_id) {
        uint32_t expected = simple_hash(data.value ^ sensor_id);
        return expected == data.integrity_tag;
    }

    void pointer_check(const void* ptr) {
        if (ptr == nullptr) {
            throw std::runtime_error("Pointer integrity violation");
        }
    }

    void control_flow_check(const std::string& ctx) {
        if (allowed_contexts_.count(ctx) == 0) {
            throw std::runtime_error("CFI violation detected");
        }
    }

    void set_security_level(bool high) {
        high_security_ = high;
    }

    bool is_high_security() const {
        return high_security_;
```

```
    }

private:
    bool high_security_{false};
    std::set<std::string> allowed_contexts_{"CTRL_TASK",
"SENSOR_TASK"};
};
```

4. Завдання реального часу з WCET

```
struct RTask {
    std::string name;
    uint32_t wcet_us;
    uint32_t deadline_us;
    uint32_t priority;
    std::function<void()> body;
};
```

5. RTOS–контролер + розрахунок slack time

```
class RTOSController {
public:
    void add_task(const RTask& task) {
        tasks_.push_back(task);
    }

    void run_cycle(SecurityModule& sec) {
        uint32_t used_time = 0;

        for (auto& task : tasks_) {
            TimePoint start = std::chrono::steady_clock::now();

            sec.control_flow_check(task.name);
            task.body();

            TimePoint end = std::chrono::steady_clock::now();
            uint32_t exec_time =
                std::chrono::duration_cast<MicroSec>(end -
start).count();

            used_time += exec_time;
        }

        slack_time_ = cycle_time_us_ > used_time
                      ? cycle_time_us_ - used_time
                      : 0;
    }

    uint32_t get_slack_time() const {
        return slack_time_;
    }

private:
```

```
    std::vector<RTask> tasks_;
    uint32_t cycle_time_us_{10000}; // 10 ms
    uint32_t slack_time_{0};
};
```

6. Менеджер завдань та адаптивна безпека

```
class TaskManager {
public:
    void adapt(SecurityModule& sec, uint32_t slack_time) {
        if (slack_time > slack_threshold_) {
            sec.set_security_level(true);
        } else {
            sec.set_security_level(false);
        }
    }

private:
    uint32_t slack_threshold_{2000}; // 2 ms
};
```

7. Логування та моніторинг

```
class Logger {
public:
    void log(const std::string& msg) {
        logs_.push_back(msg);
        std::cout << msg << std::endl;
    }

private:
    std::vector<std::string> logs_;
};
```

8. Інтеграція: прототип КФС РЧ

```
int main() {
    Sensor temp_sensor(1);
    Actuator motor;

    SecurityModule security;
    RTOSController rtos;
    TaskManager manager;
    Logger logger;

    RTask sensor_task{
        "SENSOR_TASK",
        2000,
        5000,
        1,
        [&]() {
            SensorData data = temp_sensor.read();
            if (!security.verify_data_flow(data, 1)) {
```

```
                logger.log("DFI violation!");
                return;
            }
            motor.apply(data.value);
        }
    };

    RTask control_task{
        "CTRL_TASK",
        3000,
        8000,
        0,
        [&]() {
            if (security.is_high_security()) {
                volatile uint32_t check = 0;
                for (int i = 0; i < 100; ++i) check += i;
            }
        }
    };

    rtos.add_task(sensor_task);
    rtos.add_task(control_task);

    while (true) {
        rtos.run_cycle(security);
        manager.adapt(security, rtos.get_slack_time());

        logger.log("Slack time: " +
std::to_string(rtos.get_slack_time()) +
                   " us, Security: " +
                   (security.is_high_security() ? "HIGH" : "LOW"));
    }

    return 0;
                                    }
```

Протокол аналізу звіту подібності експертом

Заявляю, що я ознайомився (-лась) з Повним звітом подібності, який був згенерований Системою виявлення і запобігання плагіату щодо роботи:

Автор: Олег НАГОРНЮК

Співавтор:

Назва: Метод оптимізації продуктивності та захищеності кіберфізичних систем на основі балансування завдань і ресурсів

Експерт: Богдан САВЕНКО

Підрозділ: Кафедра комп'ютерної інженерії та інформаційних систем

Коефіцієнт подібності 1:8.94%

Коефіцієнт подібності 2:3.2%

Мікропробіли: 88

Заміна букв: 1

Інтервали: 0

Білі знаки: 1

Дата створення звіту: 2026-04-20 19:56:58.0

Після аналізу Звіту подібності констатую наступне:

☒ **Запозичення, виявлені в роботі є законними і не є плагіатом. Рівень подібності не перевищує допустимої межі. Таким чином робота незалежна і приймається.**

☐ **Запозичення не є плагіатом, але перевищено граничне значення рівня подібностей. Таким чином робота повертається на доопрацювання.**

☐ **Виявлено запозичення і плагіат або навмисні текстові спотворення (маніпуляції), як передбачувані спроби укриття плагіату, які роблять роботу невідповідною вимогам законодавства (Ст. 32. ЗУ Про вищу освіту, пункт 3.1, Ст. 42. ЗУ Про освіту) та вимог НАЗЯВО (Критерій 5), а також кодексу етики і процедурам. Таким чином робота не приймається.**

Обгрунтування:

2026-04-20 Доцент Андрій Нічепорук

Дата експерт

Mon Apr 20 20:15:09 EEST 2026, Медзатий Дмитро Миколайович, Хмельницький національний університет,

Anti-Plagiarism (http://ap.km.ua) v-15.701

Максимальне співпадіння з одним документом 32.0%

Словники перевірки: en_US, ru_RU, ua_UA. **Помилок в документах: 9%**

	Документ		Сумарний збіг по Базі Даних	
ID: 270563 Назва: МКР Метод оптимізації продуктивності та захищеності кіберфізичних систем на основі балансування завдань і ресурсів Додано в БД: 2026-04-20 Автора: Олег НАГОРНЮК Керівники: Богдан САВЕНКО Консультанти: Опоненти:	Символи	Лексеми	Символи	Лексеми
	143842	951	47005 (33%)	343 (36%)

Джерело плагіату

ID	Опис	Наявність плагіату в документі	
		Символи	Лексеми
269866	Назва: Звіт з НДП Метод оптимізації продуктивності та захищеності кіберфізичних систем на основі балансування завдань і ресурсів Додано в БД: 2026-03-18 Автора: Нагорнюка О.В. Керівники: Павлова О.О. Консультанти: Опоненти:	46049 (32.0%)	346 (36.0%)
269864	Назва: Звіт з НДП Метод оптимізації продуктивності та захищеності кіберфізичних систем на основі балансування завдань і ресурсів Додано в БД: 2026-03-18 Автора: Нагорнюка О.В. Керівники: Павлова О.О, Консультанти: Опоненти:	46049 (32.0%)	346 (36.0%)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ МАГІСТРА

Здобувач: Олег НАГОРНЮК

Тема: Метод оптимізації продуктивності та захищеності кіберфізичних систем на основі балансування завдань і ресурсів

Спеціальність: 123 «Комп'ютерна інженерія»

Обсяг кваліфікаційної роботи магістра:

Кількість листів креслень – ; кількість сторінок записки 73

1. Короткий зміст роботи та прийнятих рішень У роботі запропоновано метод оптимізації продуктивності та захищеності кіберфізичних систем на основі балансування завдань і ресурсів

2. Висновок про відповідність роботи дипломному завданню Кваліфікаційна робота магістра відповідає виданому завданню

3. Характеристика виконання кожного розділу, ступінь використання останніх досягнень науки і техніки і передових методів роботи: У вступі подано об'єкт та предмет дослідження, мету, наукову новизну та практичну цінність роботи, а також характеристику структури роботи.

У першому розділі проведено аналіз відомих рішень щодо оптимізації продуктивності та захищеності кіберфізичних систем, цілісність потоків даних, асинхронного керуючого потоку та вказівника, що зберігає планування.

У другому розділі здійснено розроблення моделей механізмів цілісності потоків даних, керуючого потоку, вказівників і модель оптимізації продуктивності та захищеності КФС, архітектура КФС на основі оптимізації продуктивності та захищеності.

У третьому розділі розроблено стратегії Стратегії зловмисників в атаках на КФС, метод оптимізації продуктивності та захищеності кіберфізичних систем на основі балансування завдань і ресурсів.

У четвертому розділі здійснено розроблення КФС реального часу, а також проведено експеримент з розробленою реалізацією КФС РЧ.

У висновках підведено підсумки досягнення результатів з розв'язання завдань дослідження..

4. Позитивні сторони роботи: здійснено оптимізацію продуктивності та захищеності кіберфізичних систем на основі балансування завдань і ресурсів.

5. Негативні сторони роботи:

_____

_____

6. Оцінка графічного оформлення та пояснювальної записки роботи: __=____

7. Відгук про роботу в цілому: __В загальному робота виконана на високому рівні.

8. Інші зауваження: ___=

9. Оцінка кваліфікаційної роботи магістра:

Розглянувши позитивні та негативні сторони представленої кваліфікаційної роботи магістра вважаю, що робота заслуговує оцінки «відмінно» 95.00 (А)

Рецензент (прізвище, ім'я, по батькові, посада, місце роботи) ____ д.т.н., професор, Мартинюк В.В., професор кафедри автоматизації, комп'ютерно-інтегрованих технологій та робототехніки

"__1 травня__" _____2026р. _____

Зав. кафедри КІІС
д-р. філософії Ользі ПАВЛОВІЙ

Олег НАГОРНЮК

ПІБ здобувача вищої освіти

ФІТ, 2 курсу, групи КІ2м-24-2

ЗАЯВА

З правилами чинного Положення про систему забезпечення академічної доброчесності у Хмельницькому національному університеті, згідно з яким виявлення академічного плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту і застосування заходів академічної відповідальності, ознайомлений (а). Про використання спеціалізованих програмних засобів (СПЗ) StrikePlagiarism та Anti-Plagiarism для перевірки кваліфікаційних робіт здобувачів вищої освіти на наявність академічного плагіату оповіщений (а). Надаю університету право на передачу моєї роботи для обробки та збереження в базах даних СПЗ і використання роботи для виявлення академічного плагіату в інших роботах, які перевіряються СПЗ.

Також надаю свою згоду на обробку й збереження університетом моєї роботи в Інституційному репозитарії Хмельницького національного університету.

Робота надається для перевірки в електронному варіанті. Електронна версія моєї роботи збігається (ідентична) з друкованою.

1 травня 2026 року

РІШЕННЯ ЕКСПЕРТНОЇ КОМІСІЇ

КАФЕДРИ КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ ТА ІНФОРМАЦІЙНИХ СИСТЕМ ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Назва кваліфікаційної роботи Метод оптимізації продуктивності та захищеності кіберфізичних систем на основі балансування завдань і ресурсів
Автор Олег НАГОРНЮК
Освітня програма Комп'ютерна інженерія та програмування
Рівень вищої освіти другий (магістерський)
Спеціальність 123 Комп'ютерна інженерія
Науковий керівник: д.ф. Богдан САВЕНКО

На основі аналізу кваліфікаційної роботи на дотримання вимог академічної доброчесності (у т.ч. відсутності ознак академічного плагіату) з урахуванням результатів перевірки роботи спеціалізованим програмним засобом(ами) комісія зробила такий висновок:

№	Висновок	Позначка про відповідність
1	Ознаки академічного плагіату	
1.1	Запозичення, виявлені в роботі, є законними і не є академічним плагіатом (далі – зазначаються підстави віднесення запозичень до правомірних, якщо потрібно). Робота приймається до захисту.	відповідає
1.2	Виявлені запозичення не є академічним плагіатом, розміщені в розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи (далі – зазначаються детальні та аргументовані підстави віднесення запозичень до правомірних). Робота приймається до захисту, але має бути відкоригована.	
1.3	Виявлені запозичення не є академічним плагіатом, але частково розміщені в розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. Робота може бути допущена до захисту після того як буде відкоригована та доопрацьована і успішно пройде повторну перевірку на академічний плагіат.	
1.4	Робота містить навмисні текстові спотворення, передбачувані спроби укриття текстових запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту.	
2	Інші види порушень академічної доброчесності	

Підтвердження:

Запозичення, виявлені в роботі, є законними і не є плагіатом, оскільки:

1) усі запозичення фрагментарні, або мають належним чином оформленні посилання;

2) окремі виявлені збіги є загальновживаними фразами або виразами, про що свідчить посилання системи на збіг з джерелами на один фрагмент речення;

3) всі зафіксовані системою ознаки модифікації тексту відносяться до комбінування латинських символів зі україномовними скороченнями індексів в формулах, що не є модифікацією тексту.

4) значна частина знайденого плагіату відноситься до списку використаних джерел

Сумарний обсяг всіх запозичень, визначений системою виявлення збігів/ ідентичності/схожості StrikePlagiarism, складає 8,94% та системою Anti-Plagiarism складає 0%, що, з урахуванням наведених обґрунтувань, відповідає характеру наукового дослідження і свідчить на користь кваліфікаційної роботи.

30.04.2026

Завідувач кафедри _____ (Підпис) Ольга ПАВЛОВА (Ім'я, ПРІЗВИЩЕ)

Гарант освітньої програми _____ (Підпис) Олег САВЕНКО (Ім'я, ПРІЗВИЩЕ)

Керівник кваліфікаційної роботи _____ (Підпис) Богдан САВЕНКО (Ім'я, ПРІЗВИЩЕ)