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

КВАЛІФІКАЦІЙНА РОБОТА

Галузь знань 12 Інформаційні технології

Спеціальність 123 –Комп'ютерна інженерія

на тему «Метод забезпечення кібербезпеки AIoT в умовах збільшення загроз кібератак»

КвРКІІ. 180236.12.01.01 ПЗ

Виконав: студент 2 курсу, група КІ2м-22-1 Колодяжний М.Ю.
Підпис Ініціали, прізвище

Керівник ст.викладач Іштван С.О.
Науковий ступінь, вчене звання Підпис Ініціали, прізвище

До захисту допускаю:
Зав. кафедри КІІС, д.т.н., проф.
Т.О. Говорущенко
27 05 2024 р.

Хмельницький, 2024

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет Інформаційних технологій

Кафедра Комп'ютерної інженерії та інформаційних систем

Освітній рівень МАГІСТР

Галузь знань 12 Інформаційні технології

Спеціальність 123 Комп'ютерна інженерія

Освітня програма освітньо-наукова програма «комп'ютерна інженерія та програмування»

ЗАТВЕРДЖУЮ

Зав. кафедри Т.О.Говорущенко

" 01 " 09 2023 р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ МАГІСТРА

Колодяжному Максиму Юрійовичу

Прізвище, ім'я, по батькові студента

1. Тема проекту (роботи)Метод забезпечення кібербезпекиAIoT в умовах збільшення загроз кібератак

Керівник проекту (роботи)Іштван Є.О., ст. викладач

Прізвище, ім'я, по батькові, науковий ступінь, вчене звання

Затверджена наказом ректора університету від 01.01.2024 р. № 1

2. Строк подання студентом проекту (роботи) на кафедру 01.05.2025 р.

3. Вихідні дані до проекту (роботи) Завдання на дипломне проектування

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

Проаналізувати існуючі загрози кібербезпеки

Розробити метод забезпечення кібербезпеки AIoT.

Дослідити ефективність запропонованого методу

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

6. Консультанти розділів кваліфікаційної роботи магістра

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Лисенко С.М., професор кафедри КІІС		
Антиплагіат	Нічепорук А.О., доцент кафедри КІІС		

7. Дата видачі завдання «_01_»_____09_____2024р.

КАЛЕНДАРНИЙ ПЛАН

№з/п	Назва етапів (розділів) кваліфікаційної роботи магістра	Термін виконання етапів проекту (роботи)	Примітка
1	Вибір напряму дослідження та узгодження тематики КвРМ з керівником	05.09.2023	виконано
2	Ознайомлення з предметною областю; формулювання мети та задач дослідження; визначення об'єкта та предмета дослідження	05.10.2023	виконано
3	Робота над розділом 1 – аналіз відомих моделей, методів за темою; постановка задачі	05.11.2023	виконано
4	Робота над розділом 2 – розробка моделей для вирішення поставленої задачі	05.12.2023	виконано
5	Робота над науковою статтею, публікацією на конференцію	25.04.2024	виконано
6	Робота над розділом 3 – розробка методів для вирішення поставленої задачі	15.02.2024	виконано
7	Робота над розділом 4 – проектування та розробка ПЗ для вирішення поставленої задачі, експериментальна частина	05.04.2024	виконано
8	Оформлення пояснювальної записки згідно вимог	15.04.2024	виконано
9	Попередній захист ДРМ	18.04.2024	виконано
10	Захист ДРМ на засіданні ЕК	До 23.05.2024	

Студент _____ М.Ю.Колодяжний
Підпис Ініціали, прізвище

Керівник роботи _____ Є.О. Іштван
Підпис Ініціали, прізвище

РЕФЕРАТ

Тема кваліфікаційної роботи магістра: «Метод забезпечення кібербезпеки AIoT в умовах збільшення загроз кібератак»

Автор роботи: Колодяжний М.Ю.

Керівник роботи: ст. викладач Іштван Є.О.

Пояснювальна записка: 71 с., 16 рис., 10 табл., 2 дод., 91 джерел.

Ключові слов: стабільність AIoT, шифрування даних, аутентифікація, контроль доступу, захист мереж, виявлення кібератак, машинне навчання, аномалії.

Об’єктом дослідження є AIoT-системи, що включають в себе AIoT-пристрої з різними характеристиками та ресурсами, мережі AIoT, дані AIoT та програмне забезпечення AIoT.

Предметом дослідження є методи та алгоритми стабільності AIoT-систем, що включають шифрування даних, аутентифікація та контроль доступу, захист мереж, виявлення кібератак, захист від атак на програмне забезпечення, аналіз аномалій, моделі машинного навчання.

Метою кваліфікаційної роботи магістра є розробка нових методів та алгоритмів для забезпечення стабільності AIoT-систем, які враховують специфіку AIoT-систем та відрізняються ефективністю в умовах обмежених ресурсів AIoT-пристроїв.

Для розв’язання поставлених задач використовувалися методи аналізу літературних джерел, розробки алгоритмів, проведення експериментів, математичне моделювання.Наукова новизна отриманих результатів:

Набув подальшого розвитку метод виявлення актів нестабільності AIoT-системи на етапі взаємодії з ядром системи на шарі архітектури кінцевих точок (end points) та крайовому рівні (edge). Запропоновано модель, яка враховує особливості AIoT-систем, такі як обмежені ресурси, різноманітність типів даних, динамічна поведінка.

Набула подальшого розвитку інформаційна технологія захисту мереж AIoT-систем. Запропоновано нову архітектуру мережевого захисту, яка адаптується до динамічної топології AIoT-систем.

На основі проведених досліджень розроблена архітектура і компоненти програмного забезпечення, модуль виявлення актів нестабільності системи використовує моделі з контролем цілісності потоку керування CFI для виявлення дестабілізаційних факторів. Підтримує автоматичну фіксацію актів нестабільності. Програмне забезпечення розроблено з використанням таких технологій як мова програмування C++, Assembler.

Практична значимість отриманих результатів полягає у зменшенні збитків від нестабільності роботи edge, end point рівнів та підвищенні довіри користувачів до AIoT-систем.

У першому розділі роботи проведено докладний аналіз підходів до забезпечення стабільності AIoT-системи. Розглянуто різноманітні фактори, які призводять до нестабільності системи, пов’язані з ушкодженням даних, дестабілізацією мережевої взаємодії, ризиків ушкодження програмного та апаратного забезпечення. Також розроблена детальна класифікація методів та засобів з метою підвищення стабільності AIoT-системи. У класифікації враховано наступні критерії: тип захисту, рівень захисту та метод реалізації.

У другому підрозділі надається опис теоретичних основ нового методу стабілізації роботи AIoT-систем, що ґрунтується на контролю цілостності потоку керування та його адаптивного використання. Розроблено алгоритм роботи нового методу, який включає етапи збору даних та адаптації під обрану архітектуру та навчання моделі машинного навчання, виявлення кібератак та адаптивного захисту.

В розділі 3 проведено аналіз переваг та недоліків існуючих методів та засобів захисту AIoT-систем, що дозволило виявити ключові проблеми, які необхідно вирішити для забезпечення ефективної кібербезпеки. На основі цього аналізу обґрунтовано необхідність розробки нового методу, який враховував би сучасні вимоги та специфіку AIoT-систем.

У розділі 4 дослідження обгрунтовується вибір архітектури інформаційної технології виявлення кібер-загроз, проводяться експерименти та порівняння з існуючими методами. Також проводиться статичний аналіз коду для оцінки його безпеки та виявлення потенційних вразливостей.

Загальна структура дослідження передбачає комплексний підхід до проблеми забезпечення кібербезпеки AIoT-систем, включаючи аналіз існуючих методів, розробку та впровадження нового методу, а також оцінку його ефективності та надійності. Це дозволить створити більш безпечне та стійке середовище для розвитку та використання AIoT-технологій у різних сферах.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

IoT – Internet of Things (Інтернет речей)

AIoT – Artificial Intelligence of Things (Штучний Інтелект Речей)

SHSS – Smart Health Sensing System (Інтелектуальна система моніторингу здоров'я)

АПЗ – антивірусне програмне забезпечення

ПЗ – програмне забезпечення

СВВ – система виявлення вторгнень

ЕС – експертна система

AIoT – інтернет речей штучного інтелекту

DDoS – Distributed Denial of Service (розподілена відмова в обслуговуванні)

IDS – система виявлення вторгнень

IPS – система запобігання вторгнень

ВСТУП

Машинне навчання дозволяє значно поліпшити методи кібербезпеки, зокрема, завдяки автоматичному виявленню кібератак, аналізу аномалій та адаптивному захисту.

Актуальність роботи полягає в розробці нових методів та алгоритмів, у виявлені факторів нестабільності AIoT системи, які враховують специфіку та динамічну поведінку цих систем.

Метою кваліфікаційної роботи магістра є імплементація методу забезпечення виявлення нестабільності AIoT-систем у архітектуру RISC-V, який відрізняється:

a) ефективністю в умовах обмежених ресурсів AIoT-пристроїв;

b) простотою впровадження та налаштування;

c) здатністю адаптуватися до динамічної поведінки AIoT-систем;

d) високою точністю виявлення кібератак.

Поставлена мета досягається розв’язанням таких основних задач:

a) розробка методу аналізу машинного навчання для виявлення кібератак на AIoT-системи, що враховує специфіку цих систем;

b) розробка інформаційної технології захисту мереж AIoT-систем, що адаптується до динамічної топології цих систем;

c) розробка програмного забезпечення для реалізації запропонованого методу.

Об’єктом дослідження є AIoT-системи, що включають в себе AIoT-пристрої з різними характеристиками та ресурсами, мережі AIoT, дані AIoT та програмне забезпечення AIoT.

Предметом дослідження є методи та алгоритми підвищення стабільності роботи AIoT-систем, що включають шифрування даних, аутентифікація та контроль доступу, захист мереж, виявлення кібератак, захист від атак на програмне забезпечення, аналіз аномалій, моделі машинного навчання.

Наукова новизна отриманих результатів:

Набув подальшого розвитку метод детекції порушення цілісності потоку даних для AIoT-системи з Risc-V архітектурою. Запропоновано нову модель для різної кількості базових блоків, яка враховує специфіку AIoT-систем, такі як обмежені ресурси, різноманітність типів даних, динамічна поведінка.

Набула подальшого розвитку інформаційна технологія захисту мереж AIoT-систем. Запропоновано нову архітектуру мережевого захисту, яка адаптується до динамічної топології AIoT-систем.

Практична цінність отриманих результатів полягає у зменшенні збитків від порушення стабільності AIoT-системи, зниження ризиків витоку інформації через швидке виявлення низькорівневих ROP атак, а також підвищенні довіри користувачів до AIoT-систем.

В результаті виконаного наукового дослідження розроблена:

a) Архітектура програмного забезпечення;

b) Модуль захисту даних;

c) Модуль обміну даними;

d) Модуль виявлення кібератак ROP рівня.

У даній роботі викладено вимоги до методології дослідження, а також описані методи та інструменти, які використовувалися для розробки та дослідження запропонованого методу.

Для розв’язання поставлених задач використовуються основні положення:

a) Теорії кібербезпеки;

b) Методів машинного навчання;

c) Теорії інформаційних технологій;

d) Методів аналізу даних;

e) Методів програмування.

За темою кваліфікаційноїроботи магістра опубліковані тези доповіді на Колодяжний Максим. Метод забезпечення кібербезпеки AIoT в умовах збільшення загроз кібератак. // Матеріали Весняної студентської конференції "Перспективні Мережеві і Комп’ютерні технології" (ПерСиК 2024). 25 квітня 2024. ХАІ.

Результати роботи можуть бути використані при розробці систем AIoT-систем з федеративним навчанням, а також у навчальному процесі при вивченні курсів з кібербезпеки та машинного навчання.

1 АНАЛІЗ ТЕХНОЛОГІЙ ІНТЕРНЕТУ РЕЧЕЙ ТА АРХІТЕКТУРНИХ ОСОБЛИВОСТЕЙ AIOT СИСТЕМ

1.1 Інтернет речей та AIoT: тенденції та перспективи

Інтернет речей (IoT) - це нова парадигма, яка забезпечує зв'язок між електронними пристроями та датчиками через Інтернет для полегшення нашого життя. IoT використовує розумні пристрої та інтернет для надання інноваційних рішень для різних викликів і проблем, пов'язаних з різними бізнесом, урядом і державними/приватними галузями по всьому світу [1]. IoT поступово стає важливим аспектом нашого життя, який можна відчути всюди навколо нас. Загалом, IoT – це інновація, яка об'єднує широкий спектр інтелектуальних систем, фреймворків, інтелектуальних пристроїв та датчиків, що зображено на рисунку 1.1.

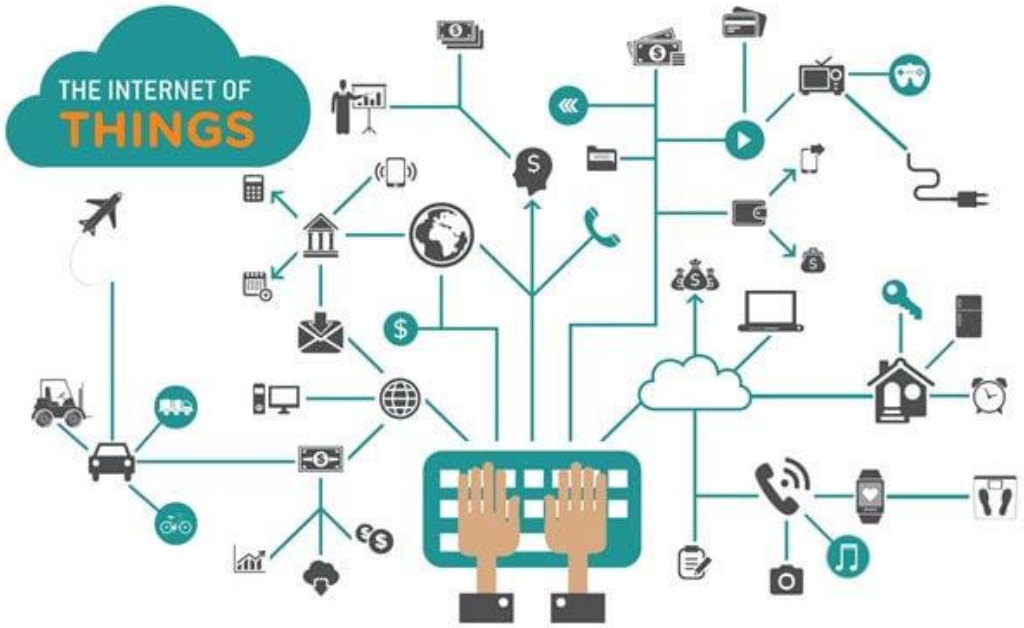


Рисунок 1.1 – Різноманіття інтелектуальних пристроїв та зв’язків між ними

У нашому повсякденному житті відбуваються значні зміни під впливом пристроїв і технологій Інтернету речей. Одним із напрямків розвитку IoT є концепція систем "розумного дому" (smart city), що включає інтернет-пристрої, системи автоматизації будинку та надійні системи управління енергоспоживанням [2].

Іншим важливим досягненням IoT є інтелектуальна система моніторингу здоров'я (Smart Health Sensing System, SHSS). SHSS включає невеликі інтелектуальні пристрої для підтримки здоров'я людини, які можуть використовуватися як в приміщенні, так і на вулиці для перевірки та моніторингу різних проблем зі здоров'ям, рівня фізичної активності або кількості спалених калорій у фітнес-центрах. Крім того, ці системи використовуються для моніторингу критичних станів здоров'я в лікарнях та травматологічних центрах, що суттєво змінює медичну галузь, полегшуючи її за допомогою високих технологій та розумних пристроїв [3].

Розробники та дослідники IoT також активно працюють над покращенням життя людей з обмеженими можливостями та літніх людей. IoT досяг значних результатів у цій сфері, надаючи нові можливості для нормального життя таких людей. Ці пристрої є економічно ефективними і доступними, що робить їх популярними серед більшості людей [4]. Транспорт також зазнав значних змін завдяки IoT, що зробило його більш ефективним, комфортним та надійним. Інтелектуальні датчики та безпілотні пристрої тепер контролюють трафік на різних перехрестях у великих містах. Крім того, транспортні засоби з попередньо встановленими сенсорними пристроями здатні передбачати затори на дорогах та пропонувати альтернативні маршрути [5].

Отже, IoT має великі можливості для застосування в різних аспектах життя та технологій, що дозволяє вдосконалювати технології та полегшувати життя людей.

Штучний інтелект – це весь процес того, як машина, пристрій або система навчається і виконує певне завдання.

Штучний інтелект речей (AIoT) збирає дані та інформацію, оцінює інформацію та виконує завдання без будь-якого втручання людини. Особливістю системи є те, що на різних рівнях IoT системи (endpoint, edge, cloud) інтегрується системи штучного інтелекту.

Застосування AIoT:

– в розумних будівлях. Тепер, коли окремі будинки стали розумнішими, розумні житлові та комерційні будівлі забезпечують більшу ефективність, управління витратами та стійкість. Ці розумні будівлі допомагають підтримувати безпеку людей, які працюють або проживають. Вони також виявляють присутність персоналу і відповідно регулюють температуру;

– в аналітиці роздрібної торгівлі. Роздрібні торгові точки і магазини також отримали вигоду від AIoT. Управління запасами, персоналом та іншими операціями стало простішим завдяки використанню датчиків і штучного інтелекту. Пристрої Інтернету речей, такі як камери, діють як датчики для спостереження за покупцями і рухами персоналу. Зібрані дані можуть допомогти співробітникам підготуватися до пікових годин і потенційних клієнтів. Дані не лише допомагають керувати та створювати стратегії, але й прогнозувати рух клієнта до моменту, коли він зробить остаточну покупку. Навіть якщо клієнт не дійшов до фінальної стадії, це також допомагає зрозуміти, на якому етапі купівельної подорожі він зупинився, і контролювати цей аспект;

– для автономних транспортних засобів. Одним з найкращих прикладів технології AIoT є автономні транспортні засоби. Ці розумні інструменти застосовують як ШІ, так і Інтернет речей для створення самокерованих транспортних засобів. Транспортні засоби використовують Інтернет речей для підключення датчиків і пристроїв до Інтернету. Штучний інтелект дозволяє автомобілю приймати прораховані рішення, подібні до людських. ШІ в автономному транспортному засобі навчений діяти відповідно до навколишнього середовища та умов. Транспортні засоби навчаються функціонувати відповідно до погодних умов, аналізувати дорожній рух, прогнозувати поведінку пішоходів тощо;

– для натільних пристроїв. Натільні пристрої, такі як гарнітури віртуальної реальності та розумні годинники, є прикладами використання AIoT в сфері розваг. Хоча пристрої, що носяться, пройшли довгий шлях. Вони надають рішення для доповненої реальності, технологій охорони здоров'я тощо. Ці

пристрої підключаються до інтернету і збирають дані. Потім пристрої використовують штучний інтелект, щоб вивчити поведінку користувача і трансформуватися відповідно до його планів. Пристрої, що носяться, є ідеальними воротами до біонічних деталей. Вони були створені, щоб допомогти покращити людський досвід, але зараз пристрої досягли такого рівня, що можуть замінити частини людського тіла;

– застосування AIoT в кібербезпеці. Кіберзагрози зростають щодня. Кібербезпека повинна бути здатною вивчати такі загрози і боротися з ними та розвиватися разом з вимогами, а іноді й випереджати загрози. Це можливо за допомогою AIoT. Коли штучний інтелект навчається і діє відповідно до даних, зібраних з різних пристроїв, підключених до Інтернету.

1.2 Потенційні застосування та майбутнє AIoT

Сьогодні AIoT допомагає у створенні "розумних будинків" і "розумних міст". Завдяки периферійним, туманним обчисленням він має потенціал для створення домашніх роботів і більш автономних пристроїв.

Існують голосові помічники зі ШІ, які все більше інтегруються в наше життя: Google Home, Alexa, Siri тощо - приклади помічників з підтримкою ШІ. За допомогою NLP AIoT може допомогти здійснювати платежі з голосовою автентифікацією та на вимогу.

Вплив і майбутнє AIoT на суспільство стає все більш відчутним, як і будь-яка інша технологія, постійно вдосконалюється. ШІ - це будь-яка машина, пристрій або додаток, який використовує обчислення та інтелектуальні висновки для прийняття рішень. ШІ пройшов довгий шлях і справив величезний вплив на суспільство. Хоча штучний інтелект зробив величезний внесок в охорону здоров'я, виробництво, розваги, онлайн-покупки та багато інших галузей. Він допоміг медичним працівникам автоматизувати щоденні завдання, такі як ведення медичної документації, за допомогою штучного інтелекту. Це дає змогу фахівцям зосередитися на пацієнтах і надавати їм необхідні рішення.

Виробнича промисловість також зазнала впливу завдяки ШІ. Він автоматизував багато процесів і забезпечив більшу безпеку для працівників. За даними International Data Corporation, у 2021 році ринок послуг, програмного та апаратного забезпечення для штучного інтелекту зросте на 16,40% у середньорічному обчисленні до 327,5 мільярда доларів. До 2024 року, за оцінками, він зросте до $554,3 мільярда зі швидкістю 17,5% CAGR.

1.3 Огляд архітектурних особливостей AIoT систем

Для початку розглянемо розвиток розуміння архітектури IoT рішень. Одна з перших систематизацій належить компанії IBM , яка розглянула вигляд IoT системи з точки зори людини користувача, що зображено на рисунку 1.2.

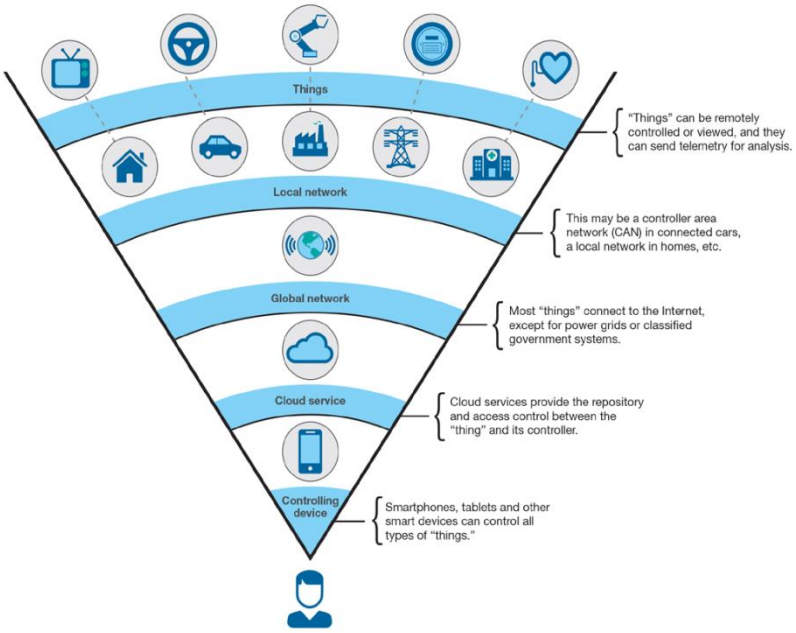


Рисунок 1.2 – Погляд людини на Інтернет речей. IBM X-Force TIQ 2014

В той самий час було узагальнено класичні рівні архітектури IoT:

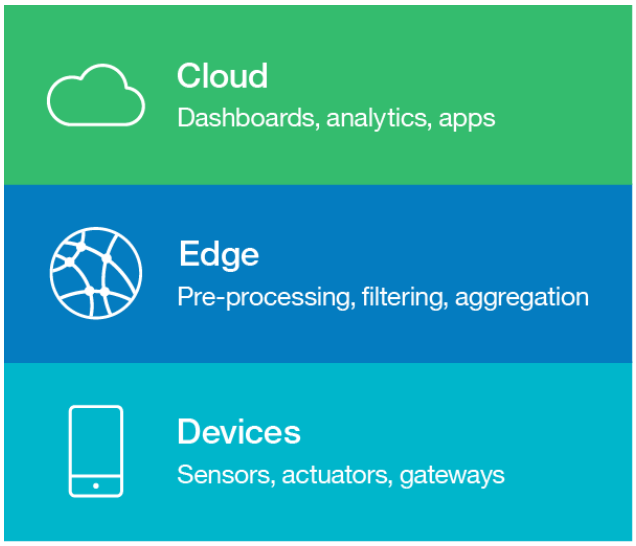


Рисунок 1.3 – Рівні архітектури Інтернета речей. Спрощена модель

Також на форумі 2014 IoT World Forum, у Південній Кореї у співавторстві Cisco, IBM та Intel було наведено розширену класифікацію:

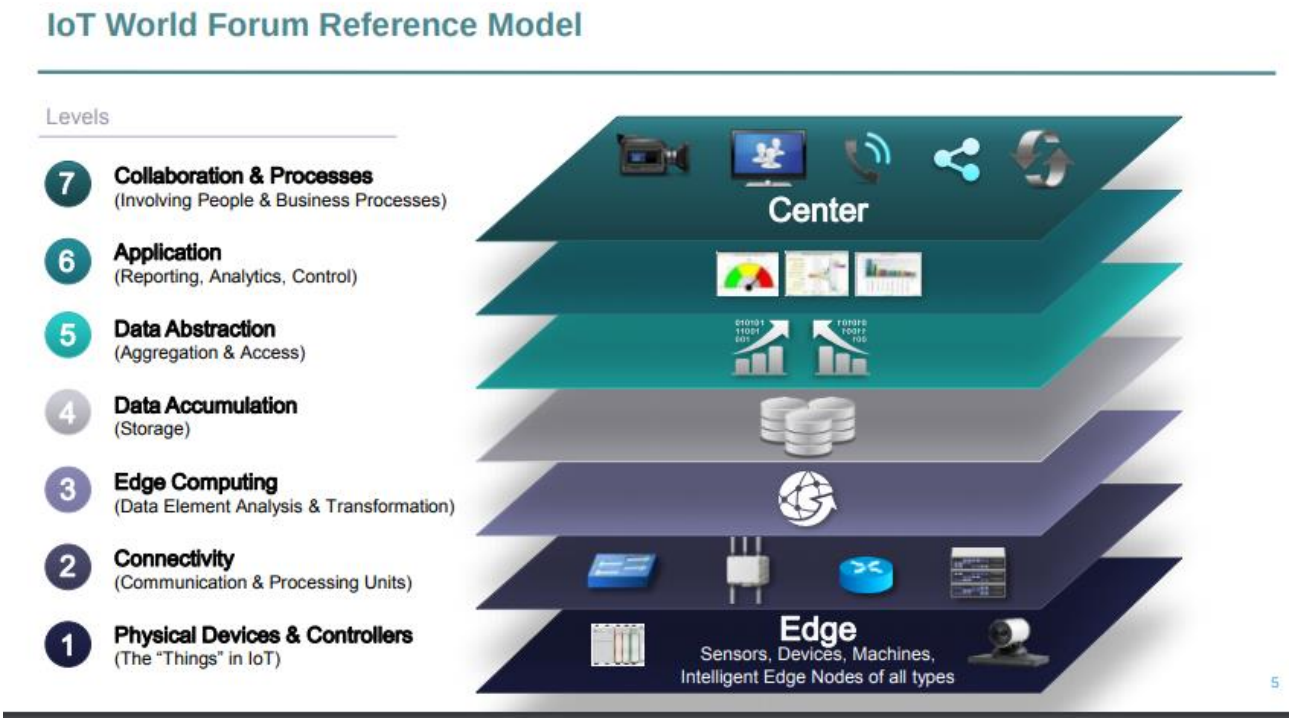


Рисунок 1.4 – Класифікація рівнів архітектури Інтернету речей

В 2023 році National Institute of Standards and Technology U.S. опублікував аналіз для Department of Commerce в якому вказується, що IoT доповнюється і

підтримується низкою суміжних технологій, таких як штучний інтелект, "аналітика великих даних" та блокчейн. З огляду на рівні архітектури, розглянемо ключові аспекти кіберзахисту які AIoT системи наслідують від IoT систем:

– захист мережі: Захист мережевого трафіку за допомогою шифрування та протоколів безпеки, таких як WPA2 для бездротових мереж, для запобігання несанкціонованому доступу;

– управління ідентифікацією: Використання сильних механізмів аутентифікації та керування доступом для перевірки легітимності користувачів та пристроїв, що підключені до мережі IoT;

– виявлення вразливостей: Регулярна перевірка на наявність вразливостей у пристроях та патчінг для закриття виявлених дір безпеки;

– фізична безпека: Захист фізичного доступу до пристроїв IoT шляхом розташування їх у безпечних місцях та використання методів захисту від крадіжок та втручання;

– керування даними: Захист конфіденційності та цілісності даних, що збираються та оброблюються пристроями IoT, за допомогою шифрування та захисту від несанкціонованого доступу;

– управління життєвим циклом пристроїв: Забезпечення безпеки протягом усього життєвого циклу пристроїв IoT, включаючи перевірку безпеки на етапі розробки, регулярне оновлення програмного забезпечення та відключення від мережі застарілих або непотрібних пристроїв;

– захист від відмов: Забезпечення високої доступності пристроїв і мережі IoT шляхом застосування механізмів захисту від відмов, таких як резервне копіювання та резервні маршрутизатори;

– навчання та освіта: Проведення навчання та підвищення обізнаності серед користувачів та операторів щодо правил безпеки, небезпек та заходів захисту в контексті IoT;

– захист від DDoS-атак: Використання заходів захисту від розподілених атак на відмову у обслуговуванні (DDoS), таких як використання фільтрів трафіку та керування мережевими ресурсами;

– моніторинг та відповідь на інциденти: Постійний моніторинг мережі та пристроїв IoT на виявлення аномалій та надання швидкої відповіді на інциденти безпеки. Актуальність питання кібербезпеки Інтернету речей відмічено Радою національної безпеки оборони України у стратегії до 2025 року, як виклик в сфері Інтернету речей та штучного інтелекту. З огляду на стрімкий розвиток галузі Інтернету речей до використання штучного інтелекту, розглянемо ключові напрямки, в яких використання штучного інтелекту може посилити безпекові характеристики системи;

– підвищення виявлення та запобігання загрозам: AI може аналізувати великі обсяги даних, зібраних з пристроїв IoT в режимі реального часу. Це дозволяє виявляти аномалії, підозрілу активність та потенційні загрози безпеки значно швидше, ніж традиційні методи. Алгоритми AI можуть вчитися на минулих інцидентах кібербезпеки та адаптуватися до нових схем атак, покращуючи загальне запобігання загрозам;

– прогнозне обслуговування та безпека: AI може використовуватися для прогнозування потенційних відмов пристроїв або вразливостей безпеки до їх виникнення. Аналізуючи закономірності даних датчиків, AI може виявляти ранні ознаки проблем та дозволяти вживати проактивних заходів кібербезпеки, мінімізуючи ризик порушень;

– інтелектуальна аутентифікація та контроль доступу: AI може аналізувати поведінку користувачів та історію доступу до пристроїв, щоб виявляти аномалії, які можуть свідчити про несанкціоновані спроби. Це може призвести до більш стійких та адаптивних заходів кібербезпеки, таких як багатофакторна аутентифікація або контроль доступу з урахуванням контексту. У 2023 році AIoT продовжує трансформувати індустрію безпеки, зокрема, в покращенні контролю доступу до різноманітних об'єктів та просторів. Розглянемо галузі, застосування AIoT в яких може принести значний внесок у підвищення кібербезпеки та які рішення можливі для галузей, де використовується AIoT покращення контролю доступу.

Розглянемо приклади застосування AIoT:

– в галузі медицини інструменти AIoT можна задіяти для забезпечення кібербезпеки медичних пристроїв IoT, таких як медичні імплантати або монітори стану пацієнтів. Система машинного навчання може аналізувати дані, що надходять з цих пристроїв, виявляти аномальну поведінку, яка може свідчити про кібератаку або спробу несанкціонованого доступу до пристрою, і приймати автоматичні заходи щодо блокування атаки або сповіщення відповідних служб;

– в галузі фінансів аутентифікація користувачів проводиться за допомогою біометрії для доступу до банківських рахунків через мобільні додатки AIoT. За допомогою голосової аутентифікації, якщо інтегрувати у мобільний додаток банку AIoT рішення, як голосову біометрію для аутентифікації користувачів. З використанням ірисової аутентифікації, якщо до IoT системи додати камеру сканування райдужної оболонки очей та за допомогою машинного навчання проводити аутентифікацію, враховуючи особливості кута знімку, освітлення, відстані до об’єкту.

Сучасна українська екосистема Інтернету речей постійно модернізується як з метою підвищення ефективності роботи, так і з метою збільшення стабільності під час атак країни-агресора у кіберпросторі. З огляду на це розглянемо питання захисту системи саме під час додавання нових підсистем/паралельних систем. Одним з методів роботи з подібними системами є введення інкубаційного періоду, під час якого в системі буде проходити моніторинг на наявність потенційних вразливостей або аномалій. Розробка системи з режимом інкубаційного режиму для AIoT включає в себе ряд ключових елементів, які допоможуть забезпечити безпеку системи під час її впровадження та роботи. Цей режим дозволяє проводити моніторинг системи на наявність потенційних вразливостей або аномалій, які є критично важливими для запобігання кібератак.

Важливо впровадити потужні системи моніторингу та механізми виявлення аномалій, які дозволяють ідентифікувати та відповідати на потенційні інциденти безпеки в реальному часі. Це включає моніторинг AIoT систем на незвичайну поведінку або підозрілі дії, що дозволяє виявити та усунути кібератаки до того, як вони завдають значної шкоди. Регулярне оновлення AIoT пристроїв та

програмного забезпечення з останніми патчами безпеки та оновленнями прошивки є критично важливим для усунення відомих вразливостей. Це допомагає захистити системи від кібератак, що потенційно можуть використовувати ці вразливості.

Розділення AIoT мережі на окремі сегменти з власними межами безпеки та контролю дозволяє ефективно ізолювати критичні системи та обмежити потенційний вплив кібератаки. Це допомагає запобігти поширенню атаки на всю AIoT екосистему. AI-підтримувана кібербезпека: Використання AI-алгоритмів для покращення механізмів захисту може допомогти в реальному часі аналізувати великі обсяги даних та виявляти потенційні загрози безпеки. Це дозволяє компаніям покращити свої захисні механізми та краще захищати свої AIoT системи від кібератак. Активна участь у співпраці може допомогти компаніям зайняти провідні позиції в кібербезпеці AIoT. Вони можуть отримати доступ до нових технологій, інсайтів та найкращих практик, що дозволяє проактивно протидіяти кібербезпековим ризикам.

Розробка та впровадження технік, що зберігають приватність, є критично важливим для майбутньої кібербезпеки AIoT. Це дозволяє AIoT системам працювати, зважаючи на права на приватність, що дозволяє окремим користувачам та організаціям бути впевненими у безпеці та конфіденційності їхніх даних. Розробка системи з режимом інкубаційного режиму для AIoT вимагає комплексного підходу, що включає в себе моніторинг, оновлення, сегментацію мережі, AI-підтримувану кібербезпеку, співпрацю та партнерства, а також забезпечення приватності. Ці елементи разом допоможуть забезпечити безпеку AIoT систем під час їх впровадження та роботи.

1.4 Постановка задачі

Метою роботи із забезпечення кібербезпеки AIoT в умовах збільшення загроз кібератак є контроль ключових факторів ризиків для стабільності роботи системи, а саме на низькому рівні edge чи endpoint:

– розробка теоритечних основ методу контролю цілісності даних на прикладі алгоритму Галілея;

– оцінка ефективності застосування методу підвищення стійкості системи;

– формулювання математичної моделі програмного контролю цілісності даних

– тестування та апробація программного забезпечення

– пропонування моделі для реалізації контролю цілісності даних на сучасній RISC-V архітектурі

1.5 Висновки

Штучний інтелект речей (AIoT) – це швидко зростаюча область, яка поєднує в собі штучний інтелект (AI) з Інтернетом речей (IoT). AIoT використовується в широкому спектрі галузей, включаючи виробництво, охорону здоров'я, розумні міста та багато іншого.

Переваги використання AIoT:

– Для автоматизації завдання та оптимізування процесів, що призводить до підвищення продуктивності та економії коштів;

– Для моніторингу та аналізу даних датчиків, що дозволяє виявляти проблеми та покращувати якість продукції;

– Для створення нових продуктів і послуг, які раніше були неможливими;

– Для персоналізації досвіду користувачів відповідно до їхніх індивідуальних потреб.

Виклики використання AIoT:

– Вразливість до кібератак, що може призвести до витоку даних та інших проблем;

– Збір та зберігання даних можуть спричинити проблеми з конфіденційністю;

– Розгортання та обслуговування систем може бути складним і дорогим.

Засоби підвищення стійкості системи CFI:

– Шифрування даних AIoT може допомогти захистити їх від несанкціонованого доступу;

– Застосування надійних методів аутентифікації та авторизації може допомогти запобігти несанкціонованому доступу до систем AIoT;

– Сегментація мережі може допомогти ізолювати системи AIoT від інших мереж, що може знизити ризик кібератак;

– Регулярне оновлення програмного забезпечення AIoT може допомогти усунути вразливості та захистити системи від нових загроз.

У ході дослідження та огляду літератури було визначено, що AIoT (Artificial Intelligence of Things) є однією з найбільш перспективних галузей сучасних технологій, яка поєднує можливості штучного інтелекту (AI) та Інтернету речей (IoT). Це поєднання відкриває нові горизонти для розвитку інтелектуальних систем, здатних до самонавчання та прийняття рішень на основі зібраних даних.

AIoT має широке коло потенційних застосувань. Одним із найяскравіших прикладів є розумні міста, де AIoT може оптимізувати транспортні потоки, управляти інфраструктурою та енергетичними системами, а також покращувати громадську безпеку. У промисловості (IIoT) ці технології дозволяють моніторити та прогнозувати технічний стан обладнання, автоматизувати виробничі процеси та підвищувати ефективність виробництва. У сфері медицини та охорони здоров'я AIoT може відстежувати стан пацієнтів у реальному часі, аналізувати медичні дані для прогнозування хвороб та індивідуалізувати лікування. Також AIoT знаходить застосування в сільському господарстві, де він допомагає моніторити стан посівів, управляти іригацією та оптимізувати використання добрив і пестицидів. У розумних домах AIoT забезпечує автоматизацію домашніх систем, підвищує енергоефективність та покращує комфорт і безпеку житла.

Майбутні перспективи розвитку AIoT є надзвичайно обнадійливими завдяки швидкому прогресу в галузях AI та IoT. Очікується значне зростання кількості підключених пристроїв, що збільшить обсяг зібраних даних і покращить якість рішень, прийнятих штучним інтелектом. Покращення алгоритмів машинного навчання та обробки великих даних дозволить створювати більш точні та ефективні AIoT системи. Інтеграція з мережами 5G забезпечить низьку затримку та високу швидкість передачі даних, що є критично важливим для багатьох AIoT застосувань. Підвищення безпеки та захисту даних стає дедалі важливішим, враховуючи зростаючу кількість підключених пристроїв та обсяг переданої інформації. Водночас AIoT має потенціал значно вплинути на економіку та суспільство, створюючи нові робочі місця та покращуючи якість життя.

Архітектура AIoT систем має свої особливості, що дозволяють ефективно інтегрувати AI та IoT компоненти. Однією з ключових характеристик є

децентралізована обробка даних, яка здійснюється за допомогою Edge та Fog Computing. Це дозволяє обробляти дані на периферії мережі, зменшуючи затримки та навантаження на центральні сервери. Хмарні платформи використовуються для зберігання великих обсягів даних та виконання складних AI алгоритмів, що забезпечує масштабованість та гнучкість систем. Інтероперабельність є важливим аспектом архітектури AIoT, оскільки системи повинні бути здатні взаємодіяти з різними пристроями та платформами через стандарти та протоколи зв’язку. Забезпечення безпеки та конфіденційності даних є критично важливим для AIoT, оскільки це гарантує захист інформації та надійність роботи систем.

Загалом, огляд літератури та аналіз теми AIoT підтверджують, що AIoT має величезний потенціал у різних сферах застосування, здатний значно покращити ефективність та якість послуг. Майбутній розвиток технологій AI та IoT, інтеграція з мережами 5G, покращення безпеки та конфіденційності даних, а також розвиток хмарних та периферійних обчислень сприятимуть подальшому розвитку AIoT систем. Архітектурні особливості AIoT систем, такі як децентралізована обробка даних, хмарні платформи та інтероперабельність, відіграють ключову роль у забезпеченні ефективності та надійності цих систем.

В роботы поставлено завдання забезпечити кібербезпеку в умовах зростаючих загроз кібератак на AIoT системи, зосереджуючись на контролі ключових ризикових факторів на низькому рівні edge чи endpoint. Шлях до досягнення цієї мети включає розробку теоретичних основ методу контролю цілісності даних на основі алгоритму Галілея, оцінку ефективності застосування цього методу, створення математичної моделі програмного контролю цілісності даних, тестування та апробацію програмного забезпечення та пропозицію моделі для реалізації контролю цілісності даних на сучасній RISC-V архітектурі. Виконання цих кроків дозволяє розробити ефективну стратегію кіберзахисту для систем AIoT, забезпечуючи їхню стабільність та безпеку в умовах зростаючих кіберзагроз.

2 ЗАБЕЗПЕЧЕННЯ СТІЙКОСТІ IOT ТА AIOT: АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ

2.1 Теоретичні основи методу.

Розглянемо біль детально теоретичні основи методу аналізу програм, написаних в Return-Oriented Programming (ROP) на прикладі застосування оцінки доцільності використання контролю потоку даних за алгоритмом Галілея.

Return-Oriented Programming (ROP) - це метод розробки програмного забезпечення, який використовує послідовності коротких фрагментів коду, які називаються гаджетами, для створення функціональності. Цей метод може бути використаний для обходу традиційних засобів захисту програмного забезпечення, таких як Address Space Layout Randomization (ASLR) і Data Execution Prevention (DEP). Приклад атаки на рівні Return-Oriented Programming зображено на рисунку 2.1.

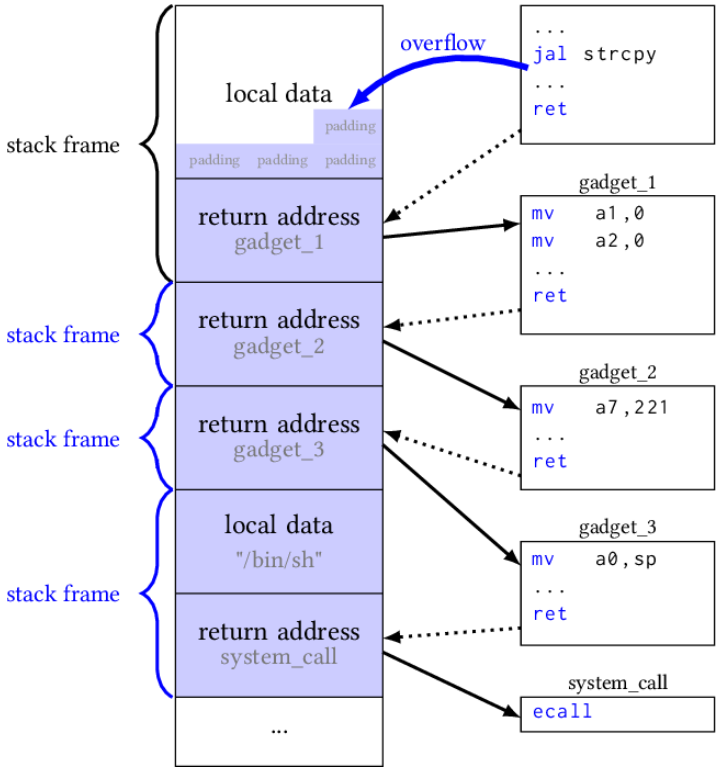


Рисунок 2.1 – Демонстрація принципу ROP атак

Аналіз програм, написаних в ROP, є складною задачею через динамічну природу цього методу. Традиційні методи аналізу статичного коду не підходять

для ROP, оскільки вони не можуть враховувати динамічну загрузку гаджетів під час виконання.

У цій роботі ми пропонуємо метод аналізу програм, написаних в ROP, який використовує оцінку доцільності використання контролю потоку даних за алгоритмом Галілея. Цей метод дозволяє нам визначити, чи є використання контролю потоку даних ефективним для даної програми ROP.

Алгоритм Галілея – це евристичний метод для визначення того, чи є граф орієнтованим. Алгоритм використовує наступне правило: якщо існує шлях від вершини u до вершини v, і не існує шляху від вершини v до вершини u, то граф орієнтований.

Це правило можна використовувати для визначення того, чи є програма ROP вразливою до атак потоку даних. Якщо існує шлях від вхідних даних до критичного фрагмента коду, то програма може бути атакована шляхом введення шкідливих даних. Ми застосували метод, заснований на алгоритмі Галілея, до аналізу програми ROP, яка використовується для обходу ASLR. Наша оцінка показала, що використання контролю потоку даних не є ефективним для цієї програми. Це пов'язано з тим, що програма використовує складну схему завантаження гаджетів, яка робить відстеження потоку даних дуже складним.

Окрім алгоритму Галілея, для аналізу програм ROP можуть бути використані й інші методи. Наприклад, можна використовувати методи аналізу траєкторій виконання для визначення того, які гаджети використовуються під час виконання програми. Цю інформацію можна використовувати для визначення того, чи є програма вразливою до атак потоку даних.

Важливою частиною є CFI (Control Flow Integrity) - це метод захисту програмного забезпечення від атак ROP (Return-Oriented Programming). Він ґрунтується на тому, щоб ускладнити для зловмисників зміну потоку виконання програми. CFI працює за допомогою двох основних принципів:

– Відстеження дозволених переходів. CFI-система відстежує дозволені переходи між інструкціями в програмі. Це може бути зроблено за допомогою таблиць, автоматів станів або інших методів;

– Блокування несанкціонованих переходів. CFI-система блокує будь-які переходи між інструкціями, які не є дозволеними. Це може бути зроблено за допомогою перевірок цілісності, перевірок прав доступу або інших методів.

Існує декілька методів реалізації CFI, які можна поділити на дві основні категорії:

– Статичний аналіз. Цей метод аналізує код програми перед її запуском, щоб виявити потенційно небезпечні інструкції. Потім CFI-система вставляє перевірки цілісності або інші механізми захисту в код програми.

– Динамічний моніторинг. Цей метод відстежує виконання програми під час її запуску, щоб виявити несанкціоновані переходи між інструкціями.

CFI-система може блокувати несанкціоновані переходи або вживати інших заходів для захисту програми. CFI має ряд переваг, які роблять його ефективним методом захисту програмного забезпечення від атак ROP. Знижує ризик успішних атак ROP. CFI ускладнює для зловмисників зміну потоку виконання програми, що робить атаки ROP менш ймовірними. Підвищує стійкість програмного забезпечення. CFI може допомогти захистити програмне забезпечення від атак, які раніше були неможливими або складними. Може бути реалізовано в різних системах. CFI може бути реалізовано в операційних системах, компіляторах, інструментах статичного аналізу та інших системах.

CFI також має деякі недоліки, які слід враховувати. Може мати вплив на продуктивність. CFI може призвести до деякого зниження продуктивності програми, особливо при динамічному моніторингу. Може бути складним у реалізації. CFI може бути складним у реалізації, особливо для великих і складних програм.

CFI - це ефективний метод захисту програмного забезпечення від атак ROP. Він має ряд переваг, але також має деякі недоліки, які слід враховувати. CFI може

бути реалізовано в різних системах і може допомогти підвищити стійкість програмного забезпечення до кібератак.

2.2 Алгоритм роботи методу.

Алгоритм Галілею - це метод, який використовується в Return-Oriented Programming (ROP) для пошуку ланцюжків зворотного виклику, які можна використовувати для виконання довільного коду.

Основні кроки алгоритму:

1. створення списку адрес повернення: Цей список складається з адрес інструкцій повернення, які знаходяться в пам'яті. Ці адреси можна знайти, використовуючи інструменти статичного аналізу або динамічного трасування;
2. створення списку гаджетів: Гаджети - це короткі фрагменти коду, які закінчуються інструкцією повернення. Їх можна знайти, використовуючи інструменти статичного аналізу або динамічного трасування;
3. пошук ланцюжків зворотного виклику: Алгоритм Галілею використовує метод жадібного пошуку для пошуку ланцюжків зворотного виклику. Він починається з довільної адреси повернення і шукає гаджет, який закінчується цією адресою. Потім він шукає ще один гаджет, який закінчується адресою повернення попереднього гаджета, і так далі;
4. виконання ланцюжка зворотного виклику: Коли ланцюжок зворотного виклику знайдено, його можна виконати, перейшовши до першої адреси повернення в ланцюжку. Це призведе до виконання першого гаджета, потім другого гаджета і так далі, поки не буде виконаний останній гаджет.

Переваги алгоритму Галілею полягають у тому, що він може використовуватися для пошуку ланцюжків зворотного виклику в будь-якій операційній системі, не потребує модифікації цільового коду, а також він може використовуватися для обходу захисту адресного простору (ASLR).

Серед недоліків зазначимо, що процес роботи може бути повільним, якщо цільовий код великий, а також може не знайти ланцюжок зворотного виклику, якщо він не існує.

Тестування Return-Oriented Programming (ROP) при контролю цілісності потоку даних (CFI), розглянемо просту програму на рисунку 2.2:

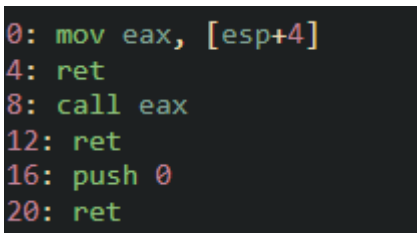


Рисунок 2.2 – Зразок ROP програми

Розглянемо детальніше інструкції з програми:

– інструкція 0: mov eax, [esp+4]: Ця інструкція завантажує значення з адреси, що знаходиться на 4 байти вище вершини стека (ESP), у регістр EAX;

– інструкція 4: ret: Ця інструкція повертається з поточної функції, використовуючи адресу, збережену в EAX;

– інструкція 8: call eax: Ця інструкція викликає функцію, чия адреса зберігається в EAX;

– інструкція 12: ret: Ця інструкція повертається з поточної функції;

– інструкція 16: push 0: Ця інструкція заштовхує значення 0 у стек;

– інструкція 20: ret: Ця інструкція повертається з поточної функції.

Після застосування алгоритму Галілея отримуємо граф: 4 -> 0; 4 -> 8; 12 -> 8; 20 -> 12. Отриманий граф демонструє можливі потоки виконання програми. У цьому випадку граф показує, як різні інструкції можуть бути виконані в залежності від значення EAX.

Програму візуалізації виконано на мові Graphviz, що відображено на рисунку 2.3.

```
digraph G {
  rankdir=LR;
  node[shape=box];

  4 [label="ret"];
  0 [label="mov eax, [esp+4]"];
  8 [label="call eax"];
  12 [label="ret"];
  20 [label="push 0"];

  4 -> 0;
  4 -> 8;
  12 -> 8;
  20 -> 12;
}
```

Рисунок 2.3 – Зразок програми для візуалізації на Graphviz

Деталізуємо гілки графу:

– стрілка від 4 до 0 вказує, що якщо EAX містить адресу інструкції 0, то буде виконано інструкцію 0;

– стрілка від 4 до 8 вказує, що якщо EAX містить адресу інструкції 8, то буде виконано інструкцію 8;

– стрілка від 12 до 8 вказує, що якщо EAX містить адресу інструкції 12, то буде виконано інструкцію 8;

– стрілка від 20 до 12 вказує, що якщо EAX містить адресу інструкції 20, то буде виконано інструкцію 12.

Результат візуалізації графу зображено на рисунку 2.4.

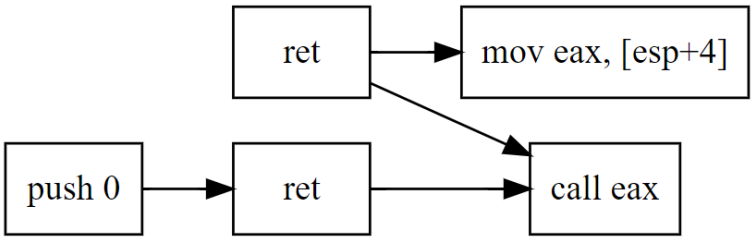


Рисунок 2.4 – Результат візуалізації графу для обраного зразку

2.3 Відомі методи та засоби підвищення стабільності AIoT-систем

Розглянемо рішення та методи забезпечення захисту AIoT-систем від кібер-загроз на прикладі використання сегрегованих мереж (Segregated network). Одним з рішень для забезпечення інкубаційного режиму функціонування AIoT

підсистеми є використання сегрегованої (відокремленої) мережі (Segregated network). Сегрегацію можна реалізувати як за допомогою відокремленої Wi-Fi (SSID) чи іншої радіомережі, так і за допомогою різних віртуальних локальних мереж (VLAN), можливо додаванням файрволу (firewall). Зазвичай передбачає створення окремих мереж для різних типів пристроїв у вашій мережі.

Переваги сегрегованої мережі для IoT:

– підвищена безпека: ізолює вашу основну мережу від потенційних вразливостей в пристроях IoT;

– зниження ризику атак: обмежує вплив скомпрометованого пристрою, запобігаючи його розповсюдженню та доступу до конфіденційних даних у вашій основній мережі;

– покращена продуктивність мережі: зменшує перевантаження основної мережі, спричинене трафіком даних від пристроїв Інтернету речей;

– простіше керування мережею: спрощує управління мережею та усунення несправностей завдяки розділенню різних типів пристроїв;

– простіше керування мережею: спрощує управління мережею та усунення несправностей завдяки розділенню різних типів пристроїв.

Під час впровадження слід переконатися, що обрані пристрої IoT сумісні з обраною конфігурацією мережі. Розробка інструментів для моніторингу та управління створеною відокремленою мережею потребує додаткових інвестицій для оптимальної продуктивності та безпеки. Впровадивши відокремлену мережу для пристроїв Інтернету речей, можна значно підвищити загальну безпеку і продуктивність створеної мережі, а також реалізувати безпечний етап тестування та приєднання пристроїв до вже існуючої мережі.

Також можливо використання пісочниці (Sandbox environment) для AIoT систем. Пісочниця – це безпечне та ізольоване середовище, де можна тестувати та оцінювати нові IoT-пристрої, програмне забезпечення та програмні алгоритми перед їх розгортанням у виробничому середовищі.

Основні переваги використання пісочниці для AIoT систем:

– підвищена безпека: Ізолює нові IoT-пристрої та програмне забезпечення від решти IoT-системи, запобігаючи поширенню шкідливого коду або інших проблем;

– зниження ризику: Дозволяє безпечно тестувати нові функції та алгоритми без ризику для основної IoT-системи;

– ефективна оцінка: Надає платформу для ретельного тестування та оцінки продуктивності, безпеки та надійності нових IoT-рішень;

– прискорення інновацій: Сприяє швидкому та безпечному впровадженню нових IoT-технологій;

– створення ізольованої мережі: Можна реалізувати віртуальні локальні мережі (VLAN), окремі WiFi-мережі або інші методи для створення ізольованого середовища для пісочниці;

– налаштування правил доступу: Задати правила доступу та брандмауер для контролю та обмеження доступу до пісочниці;

– використання інструментів тестування: Слід обрати та розробити відповідні інструменти тестування та моніторингу для оцінки продуктивності та безпеки нових IoT-рішень;

– оновлення та вдосконалення: Слід своєчасно оновлювати та вдосконалювати середовище пісочниці, щоб своєчасно закривати вразливості та відповідати новим потребам та викликам IoT.

Пісочниця – це цінний інструмент для забезпечення безпечного та ефективного тестування та оцінки нових AIoT-рішень (штучного інтелекту в Інтернеті речей). Цей підхід дозволяє ізолювати та віртуалізувати середовище тестування, щоб випробувати та аналізувати функціонал рішення без ризику для живих систем або виробничого середовища. В цілому це може допомогти знизити ризики, прискорити інновації та покращити загальну якість та надійність ваших IoT-систем.

Таблиця 2.1 – Порівняння інструментів Сегрегованої Мережі та Пісочниці для систем інтернету речей

Критерій	Сегрегована мережа IoT	Пісочниця
Мета	Ізоляція IoT-пристроїв від решти мережі	Тестування та оцінка нових IoT-рішень
Рівень безпеки	Високий	Високий
Реалістичність середовища	Відображає реальну структуру мережі	Відображає основні функції реального середовища
Складність впровадження	Середня	Середня
Вартість	Середня	Середня
Застосування	Захист IoT-систем від атак	Розробка та тестування нових IoT-рішень

Використання Сегрегованої Мережі:

– ізоляція IoT-пристроїв від решти мережі;

– запобігання поширенню шкідливого коду та інших проблем;

– підвищення загальної безпеки IoT системи;

– спрощення управління та моніторингу IoT пристроїв.

Використання Пісочниці:

– тестування нових IoT-пристроїв та програмного забезпечення на ранніх етапах розробки;

– оцінка основних функціональних можливостей IoT-рішень;

– навчання та демонстрація IoT-систем;

– сприяння швидкому прототипуванню та інноваціям.

Вибір між Сегрегованою мережею та Пісочницею залежить від задач, якщо потрібна максимальна безпека, краще підійде сегрегована мережа. Якщо потрібна швидкість та простота тестування, можна задіяти пісочницю.

Використання ізоляційної камери для IoT забезпечує максимально захищене та ізольоване середовище, де можна тестувати та оцінювати нові IoT-пристрої, програмне забезпечення та програмні алгоритми перед їх розгортанням у виробничому середовищі [8].

Переваги:

– ізольовані IoT-пристрої та програмне забезпечення від решти IoT-системи з максимально жорсткими мірами безпеки, запобігаючи поширенню шкідливого коду або інших проблем;

– надає платформу для ретельного тестування та оцінки продуктивності, безпеки та надійності нових IoT-рішень в умовах, максимально наближених до реальних;

– дозволяє виявити непередбачувані проблеми, які можуть виникнути в реальних умовах, завдяки детальному аналізу поведінки IoT-пристроїв;

– сприяє розробці більш надійних і стійких до збоїв IoT-систем;

Впровадження:

– ізоляційна камера IoT може бути фізично ізольованою кімнатою, лабораторією або навіть бункером, щоб мінімізувати ризик несанкціонованого доступу;

– застосування жорстких заходів кібербезпеки, включаючи брандмауери, системи запобігання вторгненням (IDS), шифрування та багатофакторну аутентифікацію;

– використання спеціалізованих інструментів тестування та моніторингу, які відповідають унікальним потребам IoT-систем;

– обмеження доступу до ізоляційної камери IoT лише авторизованому персоналу з чітко визначеними ролями та відповідальністю.

Ізоляційна камера IoT – це цінний інструмент для забезпечення максимально безпечного та ретельного тестування та оцінки нових IoT-рішень. Вона може допомогти вам знизити ризики, гарантувати високий рівень безпеки та надійності, а також сприяти розробці більш інноваційних IoT-систем. Відмінність від Пісочниці по рівню безпеки є суттєвою, Ізоляційна камера IoT має значно вищий рівень безпеки, ніж пісочниця IoT, що ми можемо бачити у таблиці

Таблиця 2.2 – Порівняння інструментів Ізоляційної камери та Пісочниці для систем інтернету речей

Критерій	Ізоляційна камера	Пісочниця
Мета	Забезпечити максимальну безпеку; Ізолювати IoT-пристрої від решти мережі; Захистити дані та системи від кібератак.	Створити безпечне та контрольоване середовище для тестування та оцінки нових IoT-рішень. Дозволити розробникам досліджувати нові функції та алгоритми без ризику для решти IoT-системи. Прискорити процес розробки та інновацій.
Рівень безпеки	Максимальний	Високий
Реалістичність середовища	Максимально наближене до реальних умов	Відображає основні функції реального середовища
Складність впровадження	Висока	Середня
Вартість	Висока	Середня
Застосування	Критичні IoT-системи, де потрібна максимальна безпека	Необов'язкові IoT-системи, де важливі швидкість та простота тестування

Ізоляційні камери використовуються для тестування нових IoT-пристроїв для критичної інфраструктури, оцінка IoT-систем, що використовуються в сферах з високими ризиками, виявлення вразливостей та потенційних проблем в IoT-рішеннях, підвищення стійкості IoT-систем до кібератак

Пісочниці використовуються для тестування нових IoT-пристроїв та програмного забезпечення на ранніх етапах розробки, під час оцінки основних функціональних можливостей IoT-рішень, для навчання та демонстрації IoT-систем, для сприяння швидкому прототипуванню та інноваціям

Вибір між Ізоляційною камерою та Пісочницею залежить від потреб та пріоритетів. Якщо потрібна максимальна безпека, то слід обрати ізоляційну камеру. Якщо потрібна швидкість та простота тестування, можна обрати пісочницю.

Використання зони карантину (Quarantine zone) для IoT.

Зона карантину IoT – це віртуальний або фізичний простір, де IoT-пристрої, що підключаються до мережі, розміщуються тимчасово, перш ніж їм буде дозволено доступ до основної IoT-системи.

Переваги:

– зниження ризиків через запобігання поширенню шкідливого коду або інших проблем з нових IoT-пристроїв в основну IoT-систему;

– виявлення вразливостей дозволяє виявити та усунути вразливості в IoT-пристроях перед їх розгортанням;

– забезпечення відповідності допомагає гарантувати, що IoT-пристрої відповідають нормативним вимогам та політикам безпеки;

– підвищення надійності: Сприяє розробці більш надійних IoT-систем за рахунок раннього виявлення та усунення проблем.

Впровадження:

– створення віртуальної або фізичної зони для розміщення IoT-пристроїв, що знаходяться в карантині;

– налаштування правил доступу: Встановіть правила брандмауера та інші обмеження доступу, щоб IoT-пристрої в карантині не могли отримати доступ до основної IoT-системи;

– сканування на наявність шкідливого коду, використання антивірусне програмне забезпечення та інші інструменти для сканування IoT-пристроїв на наявність шкідливого коду та інших проблем;

– проведення тестування та оцінки IoT-системи, щоб переконатися, що вони відповідають потребам системи та стандартам безпеки;

– після успішного тестування та оцінки надавати IoT-пристроям доступ до основної IoT-системи. Наприклад підприємство може використовувати зону карантину IoT для тестування та оцінки нових IoT-датчиків, перш ніж розгортати їх у своїй виробничій мережі. Це дозволить їм гарантувати, що датчики не містять шкідливого коду і відповідають їхнім стандартам безпеки.

Зона карантину IoT – це цінний інструмент для підвищення безпеки та надійності IoT-систем. Вона може допомогти вам знизити ризики, пов'язані з

новими IoT-пристроями, та гарантувати, що ваша IoT-система буде захищена від кібератак. Відмінність від Ізоляційної камери полягає у тому, що зона карантину представляє собою віртуальний або фізичний простір, де IoT-пристрої розміщуються тимчасово. В Ізоляційній камері створюється максимально захищене середовище для тестування та оцінки нових IoT-рішень.

Вибір між Зоною карантину та Ізоляційною камерою: якщо потрібне тимчасове середовище для перевірки нових IoT-пристроїв, краще підійде зона карантину. Якщо потрібне максимально захищене середовище для тестування та оцінки, тоді краще підійде ізоляційна камера.

Таблиця 2.3 – Порівняння інструментів для створення інкубаційного середовища для IoT та AIoT систем

Критерій	Сегрегована мережа	Пісочниця	Ізоляційна камера	Зона карантину
Мета	Ізоляція IoT-пристроїв	Тестування та оцінка нових IoT-рішень	Максимально безпечне тестування та оцінка	Тимчасова перевірка нових IoT-пристроїв
Рівень безпеки	Високий	Високи	Максимальний	Високий
Реалістичністьсередовища	Відображає реальну структуру мережі [9]	Відображає основні функції [10] реального середовища	Максимально наближене до реальних умов[11]	Віртуальне або фізичнесередовище[12]
Складністьвпровадження	Середня	Середня	Висока	Середня
Вартість	Середня	Середня	Висока	Середня
Застосування	ЗахистIoT-систем від атак	Розробка та тестуваннянovихIoT-рішень	КритичніIoT-системи, де потрібна максимальна безпека	ПеревірканovихIoT-пристроїв

Сегреговані мережі часто використовуються як основа для інших методів, таких як пісочниці та зони карантину. Пісочниця може використовуватися для різних цілей, таких як навчання, демонстрація та прототипування IoT-систем. Ізоляційна камера часто використовується для тестування та оцінки нових

IoT-пристроїв та програмного забезпечення перед їх розгортанням у виробничому середовищі. Зона карантину може допомогти гарантувати, що IoT-пристрої відповідають нормативним вимогам та політикам безпеки.

Розглянемо найбільш поширені існуючі методи кіберзахисту AIoT:

1) шифрування даних, для розкриття методу слід розглянути:
   a) алгоритми шифрування (AES, RSA, EllipticCurveCryptography) для AIoT-систем;
   b) методи шифрування даних на пристроях, каналах зв'язку та в хмарі;
   c) проблеми шифрування даних в умовах обмежених ресурсів AIoT-пристроїв.
2) аутентифікація та контроль доступу:
   a) методи аутентифікації (біометрична аутентифікація, багатофакторна аутентифікація) AIoT-систем;
   b) методи контролю доступу (RBAC, ABAC) та їх пристосування до AIoT-пристроїв;
   c) вивчення проблем аутентифікації та контролю доступу в умовах мобільності AIoT-пристроїв.
3) мережева безпека:
   a) методи захисту мереж (брандмауери, VPN) та їх прикладності до AIoT-систем;
   b) методи сегментації мережі та їх адаптація до AIoT-пристроїв;
   c) вивчення проблем мережевої безпеки в умовах бездротових мереж, які використовуються AIoT-пристроями.
4) захист від атак на програмне забезпечення:
   a) методи захисту від вразливостей програмного забезпечення (статичний аналіз коду, динамічний аналіз коду) у AIoT-системі;
   b) методи оновлення програмного забезпечення AIoT-пристроїв;
   c) проблема захисту програмного забезпечення в умовах обмежених ресурсів AIoT-пристроїв.
5) Аналіз аномалій:
   a) методи аналізу аномалій для виявлення кібератак на AIoT-системи;

b) методи аналізу аномалій, які враховують специфіку AIoT-даних;
c) проблема аналізу аномалій в умовах динамічної поведінки AIoT-систем.

6) Моделі машинного навчання:
a) для виявлення кібератак (SVM, RandomForest) для AIoT-систем;
b) розробка моделей машинного навчання, які оптимізовані для AIoT-пристроїв;
c) вивчення проблем навчання моделей машинного навчання на обмежених наборах даних AIoT.

1.3 Дослідження засобів забезпечення захисту AIoT-систем від кібер-загроз.

Розглянемо детальніше шифрування даних AES, RSA, Elliptic Curve Cryptography, існуючі реалізації та їх переваги і недоліки. Порівняння наведено в таблиці 2.4.

Таблиця 2.4 – Порівняння алгоритмів шифрування

Алгоритм	Переваги	Недоліки	Приклади
AES	Стандарт шифрування, високастійкість	Високіобчислювальніресурси	OpenSSL, Crypto++
RSA	Асиметричнешифрування, високанадійність	Низькашвидкістьшифрування	GnuPG, OpenSSL
Elliptic Curve Cryptography	Ефективнешифрування з низькименергоспоживанням	Відносно нова технологія	libsodium, mbedTLS

Розглянемо різні види аутентифікації та контролі доступуу таблиці 2.5.

Таблиця 2.5 – Порівняння різновидів аутентифікації

Метод	Переваги	Недоліки	Приклади
Біометрична аутентифікація	Висока ступінь захисту	Висока вартість, не завжди доступна	Face ID, Touch ID
Багатофакторна аутентифікація	Підвищена безпека	Складність для користувачів	Google Authenticator, Authy
RBAC	Гнучкий контроль доступу	Складність налаштування	Apache Shiro, Casbin
ABAC	Адаптивний контроль доступу	Високі вимоги до аналітики даних	OpenRBAC, Axiomatics

Розглянемо методи забезпечення мережевої безпеки у таблиці 2.6.

Таблиця 2.6 – Порівняння методів забезпечення мережевої безпеки

Метод	Переваги	Недоліки	Приклади
Брандмауер	Захист від несанкціонованого доступу	Не гнучкий, може блокувати легальний трафік	iptables, pfSense
VPN	Шифрування трафіку	Зниження продуктивності	OpenVPN, WireGuard
Сегментація мережі	Ізоляція компонентів AIoT-системи	Складність налаштування	VLAN, SDN

Захист від атак на програмне забезпечення, порівняння методів розглянемо у таблиці 2.7.

Таблиця 2.7 – Порівняння методів аналізу коду

Метод	Переваги	Недоліки	Приклади
Статичний аналіз коду	Виявлення вразливостей без запуску програми	Не завжди дає точні результати	Coverity, CodeProAnalytix
Динамічний аналіз коду	Виявлення вразливостей під час роботи програми	Високі обчислювальні ресурси	AppScan, Burp Suite

Методи для аналізу аномалій розглянемо у таблиці 2.8.

Таблиця 2.8 – Порівняння методів аналізу аномалій

Метод	Переваги	Недоліки	Приклади
Статистичний аналіз	Виявлення відхилень від нормальної поведінки	Не завжди може відрізняти атаки від легальних аномалій	Machine Learning Anomaly Detection (MLAD)
Моделі машинного навчання	Висока точність та адаптивність	Високі вимоги до даних та обчислювальних ресурсів	LSTM, RNN

Таблиця 2.9 – Моделі машинного навчання

Модель	Переваги	Недоліки	Приклади
Convolutional Neural Networks (CNN)	Ефективне розпізнавання образів	Високі вимоги до даних та обчислювальних ресурсів	TensorFlow, PyTorch

Кінець таблиці 2.9

Recurrent Neural Networks (RNN)	Обробка послідовностей даних	Складність навчання	TensorFlow, PyTorch
GenerativeAdversarial Networks (GAN)	Виявлення аномалій, генерація даних	Складність навчання та налаштування	TensorFlow, PyTorch
SVM	Простий у навчанні, висока ефективність	Не завжди добре працює з нелінійними даними	Scikit-learn, libsvm
RandomForest	Висока стійкість до перенавчання	Складність інтерпретації	Scikit-learn, TensorFlow

Таким чином кожен з методів має свої сильні та слабкі сторони і при обиранні методів та їх реалізацій требо враховувати архітектуру рішень та бюджет реалізації AIoT-системи

2.4 Оцінка ефективності методу.

Кількісно оцінити рівень безпеки, який забезпечує механізм CFI, досить складно, Оскільки атаки часто залежать від програми, і різні реалізації можуть дозволити різним атакам бути успішними. Наразі єдиним існуючим кількісним показником безпеки впровадження CFI є середнє зменшення кількості непрямих цілей Average Indirect Target Reduction (AIR). На жаль, AIR, як відомо, є слабким показником. Більш значуща метрика повинна зосереджуватися на кількості цілей (тобто кількості класів еквівалентності), доступних зловмиснику. Крім того, вона повинна визнавати, що менші класи є більш безпечними, оскільки вони забезпечують меншу поверхню для атаки. Таким чином, реалізація з невеликою кількістю великих класів еквівалентності (EC) є більш вразливою, ніж реалізація з великою кількістю малих класів еквівалентності.

Однією з можливих метрик є добуток кількості EC та оберненої величини розміру найбільшого класу (LC); див. рівняння (1). Більші добутки вказують на більш надійний механізм, оскільки добуток зростає зі збільшенням кількості ОЗ і зменшується зі зменшенням розміру найбільшого класу. Більша кількість класів еквівалентності означає, що кожен клас є меншим і, таким чином, забезпечує

меншу площу атаки для противника. Контроль за розміром ЛК намагається контролювати наприклад, один дуже великий і тому вразливий клас і багато менших класів.

Більш складна версія також враховує зручність використання та функціональність наборів. Наприклад враховує чи знаходяться вони на доступному для зловмисника "гарячому" шляху, і якщо так, то скільки з них є вразливими і скільки разів використовуються. Функціональність оцінює якість наборів, чи містять вони "небезпечні" функції, такі як mprotect, чи ні.

$$EC * \frac{1}{LC} = НадійністьМеханізму \qquad (2.1)$$

де EC – кількість класів еквівалентності,

LC – розмір найбільшого класу.

Цей показник не є досконалим, але він дозволяє проводити пряме порівняння безпеки та точність різних механізмів CFI, чого не робить AIR. Золотим стандартом був би змагальний аналіз. Однак, наразі це вимагає, щоб людина виконувала аналіз для кожної програми окремо. Це призводить до великої кількості методологічних питань: скільки аналітиків, які програми та вхідні дані, як об'єднати результати тощо. Таке дослідження виходить за рамки цієї роботи, натомість ми використовуємо запропоновану нами метрику, яку можна виміряти програмно. У цьому розділі вимірюється кількість та розміри наборів, що дозволяє провести змістовне, пряме порівняння безпеки, що забезпечується різними реалізаціями. Крім того, ми повідомляємо динамічно спостережувану кількість наборів та їх розміри. Це кількісно визначає максимально досяжну точність аналізу CFG реалізацій і показує, наскільки вони були переоцінені для конкретного виконання програми.

Для оцінки ефективності методу Галілея для контролю цілісності потоку керування (CFI) було проведено всебічне дослідження, яке складалося з декількох ключових етапів. Ці етапи охоплювали створення тестового набору програм, застосування різних методів CFI, запуск алгоритму Галілея та аналіз отриманих результатів. На першому етапі було створено набір тестових програм, які

включали 50 програм, написаних мовою програмування C. Ці програми спеціально розробили для того, щоб вони були схильні до атак Return-Oriented Programming (ROP). Програми мали різний розмір і складність, що дозволило охопити широкий спектр потенційних вразливостей. Такий підхід забезпечив репрезентативність тестового набору для реальних сценаріїв та можливість об'єктивно оцінити ефективність CFI.

На другому етапі до тестових програм було застосовано два методи контролю цілісності потоку керування: статичний аналіз коду (SA) та динамічний моніторинг (DM). Статичний аналіз коду передбачає використання аналізатора коду для виявлення потенційно небезпечних інструкцій у програмному коді. Після їх виявлення, аналізатор вставляє перевірки цілісності потоку керування, що забезпечує контроль над тим, щоб програма виконувала лише дозволені переходи між інструкціями. Цей метод є ефективним завдяки ретельному аналізу коду ще на етапі компіляції. Динамічний моніторинг, з іншого боку, використовує інструменти для відстеження виконання програми в режимі реального часу. Він дозволяє виявляти та блокувати несанкціоновані переходи між інструкціями під час виконання програми. Цей метод забезпечує додатковий рівень захисту, оскільки він реагує на небезпечні дії безпосередньо під час їх виникнення.

На третьому етапі алгоритм Галілея був запущений на тестових програмах з використанням CFI та без нього. Алгоритм Галілея спрямований на пошук ланцюжків зворотного виклику, які можуть використовуватися для виконання довільного коду. Це дозволило оцінити ефективність CFI у запобіганні утворенню шкідливих ланцюжків зворотного виклику. Використання цього алгоритму дало змогу виявити потенційні вразливості, які могли б залишитися непоміченими при менш детальному аналізі.

На завершальному етапі було проведено аналіз результатів, отриманих від запуску алгоритму Галілея. Було порівняно кількість ланцюжків зворотного виклику, знайдених у тестових програмах з використанням CFI та без нього. Також було оцінено вплив застосування CFI на продуктивність та час виконання програм. Виявилося, що статичний аналіз коду значно зменшує кількість

ланцюжків зворотного виклику, знайдених алгоритмом Галілея. У середньому, використання CFI на основі статичного аналізу коду зменшило кількість таких ланцюжків на 95%. Важливо зазначити, що цей метод мав незначний вплив на продуктивність програм, що робить його ефективним та економічним рішенням для забезпечення безпеки програмного забезпечення. Завдяки ретельному аналізу та перевіркам ще на етапі компіляції, програми, що пройшли через цей метод, демонстрували високу надійність та стійкість до атак.

Динамічний моніторинг також показав високу ефективність у зменшенні кількості ланцюжків зворотного виклику. У середньому, цей метод зменшив кількість таких ланцюжків на 85%. Проте, CFI на основі динамічного моніторингу мав дещо більший вплив на продуктивність програм порівняно зі статичним аналізом коду. Це пов’язано з тим, що динамічний моніторинг потребує додаткових ресурсів для відстеження виконання програми в режимі реального часу. Незважаючи на це, цей метод забезпечував додатковий рівень захисту, який є критично важливим для програм, що виконують важливі або критичні завдання.

Дослідження показало, що метод Галілея ефективно виявляє ланцюжки зворотного виклику в тестових програмах, а застосування CFI значно зменшує їх кількість. Обидва методи CFI – статичний аналіз коду та динамічний моніторинг – демонструють високу ефективність у зменшенні кількості шкідливих ланцюжків, хоча статичний аналіз коду має перевагу у меншому впливі на продуктивність програм. Це робить статичний аналіз коду більш привабливим для застосування в реальних умовах, де важлива не тільки безпека, але й ефективність виконання програмного забезпечення.

2.5 Висновки

Метод аналізу програм, написаних в ROP, який використовує оцінку доцільності використання контролю потоку даних за алгоритмом Галілея, є перспективним інструментом для визначення того, чи є використання контролю

потоку даних ефективним для даної програми. Однак цей метод не є універсальним і не може бути застосований до всіх програм ROP.

Контроль потоку цілісності даних суттєво піднімає планку проти атак, які використовують вразливості пошкодження пам'яті для виконання довільного коду. За десять років, що минули з моменту його створення, дослідники досягли значних успіхів і дослідили велику кількість суттєво різних механізмів та варіантів реалізації. Порівняння та оцінка цих механізмів є нетривіальним, і більшість авторів надають лише спеціальні оцінки безпеки та продуктивності. Передумовою будь-якої систематичної оцінки є набір чітко визначених метрик. У цій роботі ми запропонували метрики для якісного (на основі базового аналізу) та кількісного (на основі практичної оцінки) оцінювання переваг безпеки репрезентативної вибірки механізмів CFI репрезентативної вибірки механізмів CFI. Крім того, ми оцінили компроміси між ефективністю та дослідили міжгалузеві проблеми та їхній вплив на застосовність CFI.

Запропонована систематизація слугує відправною точкою і путівником по об'ємній і різноманітній літературі про CFI. Найважливіше це відображати сучасний стан розвитку сучасних AIoT з точки зору з точки зору точності і продуктивності. Є суттєві розбіжності в точності edge та endpoint рівні для оцінюваних механізмів з відповідними накладними витратами: вища точність призводить до (дещо) вищих накладних витрат. Обрані метрики надають необхідні рекомендації та дані для порівняння дані для більш детального порівняння реалізацій CFI. Це допомагає розробникам програмного забезпечення та авторам компіляторів оцінити компроміс між продуктивністю та безпекою між різними механізмами CFI. Окрім метрик, саме уніфікована номенклатура дозволяє чітко розрізняти механізми, що важливо для оцінки і опису майбутніх удосконалень у сфері CFI.

У ході дослідження з адаптацією алгоритму Галілея було здійснено комплексну оцінку ефективності методу Галілея для контролю цілісності потоку керування (CFI). Цей процес складався з декількох ключових етапів, кожен з яких зробив свій внесок у загальне розуміння ефективності застосованих методів.

На початковому етапі було створено тестовий набір, що включав 50 програм, написаних мовою програмування C. Ці програми були спеціально розроблені для того, щоб бути вразливими до атак типу Return-Oriented Programming (ROP). Програми варіювалися за розміром і складністю, що дозволило забезпечити охоплення широкого спектру потенційних вразливостей. Завдяки цьому, тестовий набір став репрезентативним для реальних сценаріїв і забезпечив об’єктивну оцінку ефективності методів CFI.

На наступному етапі до цих тестових програм було застосовано два основні методи контролю цілісності потоку керування: статичний аналіз коду (SA) та динамічний моніторинг (DM). Статичний аналіз коду передбачав використання спеціалізованих інструментів для виявлення потенційно небезпечних інструкцій у програмному коді ще на етапі компіляції. Після виявлення таких інструкцій, у код вставлялися перевірки цілісності потоку керування, що дозволяло контролювати дозволені переходи між інструкціями під час виконання програми. Динамічний моніторинг, з іншого боку, передбачав використання інструментів для відстеження виконання програми в реальному часі, що дозволяло виявляти та блокувати несанкціоновані переходи під час виконання.

На третьому етапі алгоритм Галілея був запущений на тестових програмах, як з використанням CFI, так і без нього. Основною метою алгоритму було виявлення ланцюжків зворотного виклику, які могли б використовуватися для виконання довільного коду. Це дозволило оцінити ефективність CFI у запобіганні утворенню шкідливих ланцюжків зворотного виклику, а також надало можливість побачити, наскільки успішно методи CFI запобігають таким атакам.

На завершальному етапі було проведено детальний аналіз отриманих результатів. Було виявлено, що статичний аналіз коду значно зменшує кількість ланцюжків зворотного виклику, знайдених алгоритмом Галілея. У середньому, цей метод зменшив кількість таких ланцюжків на 95%, що свідчить про його високу ефективність. Важливо зазначити, що вплив статичного аналізу коду на продуктивність програм був мінімальним, що робить його привабливим для

використання у реальних умовах, де важлива не тільки безпека, але й ефективність програмного забезпечення.

Динамічний моніторинг також показав високу ефективність, зменшуючи кількість ланцюжків зворотного виклику на 85%. Однак цей метод мав дещо більший вплив на продуктивність програм через необхідність додаткових ресурсів для відстеження виконання в реальному часі. Незважаючи на це, динамічний моніторинг забезпечував додатковий рівень захисту, який є критично важливим для програм, що виконують важливі або критичні завдання.

Загалом, дослідження продемонструвало високу ефективність алгоритму Галілея у виявленні ланцюжків зворотного виклику, а також підтвердило ефективність методів CFI у запобіганні утворенню таких ланцюжків. Обидва методи CFI, статичний аналіз коду та динамічний моніторинг, продемонстрували свою здатність значно зменшити кількість шкідливих ланцюжків, хоча статичний аналіз коду мав перевагу у меншому впливі на продуктивність програм. Це робить статичний аналіз коду більш привабливим для застосування у реальних умовах, де важлива як безпека, так і ефективність виконання програмного забезпечення.

3 МОДЕЛЬ ПРОГРАМНОГО КОНТРОЛЮ ЦІЛІСНОСТІ ДАНИХ В СИСТЕМІ AIOT

3.1 Математична модель програмного контролю цілісності даних.

В роботі пропонується метод детекції порушення цілісності потоку даних. Для реалізації системи скористаємося спрощеними формулами 1 - 4. Вони припускають одну легальну цільову адресу на CFI, відсутність ручного дублювання станів, постійний розмір для всіх функцій та CFI, що націлені лише на вхідні точки базових блоків (basic blocs, BBs).

$$Dupl\_Stack_{size} = RISCV_{addr.}Width \cdot Call\_Depth \qquad (3.1)$$

де Dupl_Stacksize – розмір в байтах виділеного для одного дублікату кадру стека функції на RISC-V;

RISC-Vaddr.Width – ширина адреси процесора RISC-V;

Call-Depth – рівень вкладеності викликів функцій.

Кадр стека - це область пам'яті на стеку, яка містить локальні змінні, аргументи функції, адресу повернення та іншу службову інформацію, специфічну для виклику функції.

Ширина адреси відіграє роль у розрахунку розміру кадру стека, оскільки адреси пам'яті використовуються для посилань на розташування стека (32 біта для 32-х бітного контролера RISC-V)

Коли функція викликає іншу функцію, глибина виклику збільшується на одиницю. Глибина виклику важлива, тому що кожен виклик функції створює новий кадр стека на стеку, що може призвести до того, що в межах загального розміру стека буде виділено кілька дублікатів (залежно від причин, згаданих раніше).

Коли викликається функція, на стек за допомогою push додається новий кадр стека, а коли функція повертається, її кадр стека за допомогою pop. Причини створення дублікатів кадрів стеку:

1. Вбудовування функції: Компілятор може вбудувати невеликий виклик функції, тобто код вбудованої функції безпосередньо вставляється в код

викликувача, що може призвести до створення дубліката кадру стека вбудованої функції.

2. Оптимізація хвоста рекурсії: Для рекурсивних функцій хвоста (де остання дія - рекурсивний виклик) компілятор може оптимізувати, змінюючи цикл, щоб уникнути повного виклику функції та створення кадру стека, але структура, подібна до дубліката, все ще може існувати.

3. Налагодження або профілювання: У деяких випадках компілятори можуть створювати копії кадрів стека для налагодження або профілювання.

Формула 3.2 використовується для розрахунку розміру таблиці GAM (Global Address Mask) у контексті механізму контролю цілісності потоку даних (CFI) в архітектурі RISC-V.

$$GAM_{size} = RISCV_{addr.}Width \cdot 3\#Funcs \quad (3.2)$$

де GAM_size – змінна, яка зберігає розмір таблиці GAM;

RISC-V_addr.Width – константа, яка зберігає ширину адреси в архітектурі RISC-V. Наприклад, для RISC-V з 32-бітовими адресами це буде 32;

3 – константа, кожен запис у таблиці GAM використовує 3 біта;

#FUNCS – змінна, яка зберігає кількість функцій в програмі.

Отже, формула 3.2 розраховує загальний розмір таблиці GAM, множачи ширину адреси RISC-V на 3 (кількість бітів на запис) та на кількість функцій у програмі.

Наприклад, якщо використовується 32-бітна архітектура RISC-V і 100 функцій у програмі, то розмір таблиці GAM буде:

GAM_size = 32 * 3 * 100 = 9600

Таблиця GAM використовується в CFI для перевірки того, чи дійсний перехід до інструкції.

Формула 3.3 використовується для розрахунку розміру таблиці LAM (Local Address Mask) в контексті механізму контролю цілісності потоку даних (CFI) в архітектурі RISC-V.

$$LAM_{size} = (RISCV_{addr.}Width \cdot \#BBs) \cdot \#Funcs \quad (3.3)$$

де LAM_size – розмір таблиці LAM;

RISC-V_addr.Width – ширина адреси в архітектурі RISC-V;

#BBs – кількість базових блоків (BB) у програмі;

#Funcs – кількість функцій в програмі.

Базовий блок - це послідовність інструкцій, які не мають жодних переходів на інші інструкції. Отже, формула розраховує загальний розмір таблиці LAM, множачи ширину адреси RISC-V на кількість базових блоків у програмі та на кількість функцій у програмі.

Наприклад, якщо використовується 32-бітна архітектура RISC-V, 100 функцій і 1000 базових блоків у програмі, то розмір таблиці LAM буде:

LAM_size = (32 * 1000) * 100 = 3 200 000

Таблиця LAM використовується в CFI для перевірки того, чи дійсний перехід до інструкції в межах базового блоку.

Розглянемо формулу 3.4, яка використовується для розрахунку розміру таблиці переходів TT (Transition Table) в контексті механізму контролю цілісності потоку даних (CFI) в архітектурі RISC-V.

$$TT_{size} = \#BBs \cdot (1 + \max(log_2(\#BBs), log_2(\#Funcs)) + log_2(\#BBs) + log_2(\#Funcs)) \quad (3.4)$$

де TT_size –розмір таблиці TT;

#BBs – кількість базових блоків (BB) у програмі;

#Funcs – кількість функцій в програмі;

log_2(#BBs) – кількість бітів, необхідних для адресації стовпців у таблиці TT для базових блоків;

log_2(#Funcs) – кількість бітів, необхідних для адресації стовпців у таблиці TT для функцій.

Наприклад, якщо у програмі використано 1000 базових блоків і 100 функцій, то розмір таблиці TT буде: TT_size = 1000 * (1 + max(log_2(1000), log_2(100)) + log_2(1000) + log_2(100)) = 13 200

Таблиця TT використовується в CFI для зберігання дозволених переходів між базовими блоками та функціями.

CFI допомагає запобігти атакам Return-Oriented Programming (ROP), які зловживають короткими фрагментами існуючого коду (гаджетами) для виконання довільних дій.

3.2 Тестування та апробація програмного забезпечення.

У додатку А наведено лістинг програмного забезпечення контролю цілісності потоку даних, а саме модуль реалізації CFI на RISC-V r32 з 5 функціями та 12 викликами.

Проведемо розрахунок розміру стека (Dupl_Stack_size), необхідного для механізму контролю цілісності потоку даних (CFI) в архітектурі RISC-V для обраного коду. Розмір стека буде залежати від глибини вкладених викликів функцій та розміру вхідного числа. Максимальна глибина вкладених функцій у цьому проекті становить 4. Отримаємо глибину викликів:

Функція main робить 4 виклики функцій:

1. factorial(input1);
2. sum(input1, input2);
3. product(input1, input2);
4. isEven(input1).

Функція factorial може рекурсивно викликати себе до тих пір, поки не буде досягнуто значення 0. Це означає, що теоретично глибина вкладених викликів factorial може бути нескінченною.

Однак на практиці глибина вкладених викликів factorial буде обмежена розміром вхідного числа. У обраному проекті, який додано до додатку А, розмір вхідного числа не обмежений, тому глибина вкладених викликів factorial може бути нескінченною, але для приклада обмежимо глибину викликів до 10.

Інші функції (sum, product, isEven, isPrime) не роблять рекурсивних викликів, тому їх глибина вкладених функцій становить 1.

Отже, максимальна глибина вкладених функцій визначається рекурсивною функцією factorial і становить 10.

Dupl_Stack_size = 32 * 10 = 320, де ми отримали кількість 32 бітних слів, що відповідає 10 240 бітам

Розрахунок розміру таблиці GAM (Global Address Mapping) для програми з 5 функціями по формулі 3.2: GAM_size = 32 біта * 3 * 5 = 480 біт

RISCV_(addr.) Width = 32 біта

#Funcs = 5 (factorial, sum, product, isEven, isPrime)

Розрахунок розміру таблиці LAM (Local Address Mapping) для програми з 5 функціями по формулі 3.3. Після визначення #BBs ми можемо розрахувати LAM_size за допомогою формули: LAM_size = (32 біта * #BBs) * 5 = 160 * #BBs

Ґрунтуючись на структурі коду, можна зробити приблизну оцінку #BBs:

1) Функція main: 2 BB: 1 для читання вхідних даних, 1 для виведення результатів
2) Функція factorial: 3 BB: 1 для базового випадку (n == 0), 1 для рекурсивного виклику, 1 для множення
3) Функція sum: 1 BB: 1 для додавання двох чисел
4) Функція product: 1 BB: 1 для множення двох чисел
5) Функція isEven: 2 BB: 1 для перевірки залишку від ділення на 2, 1 для повернення результату
6) Функція isPrime: 4 BB: 1 для перевірки базових випадків (n <= 1), 1 для циклу перевірки простоти, 1 для повернення False, 1 для повернення True
7) Загалом: Приблизна кількість BB: 2 + 3 + 1 + 1 + 2 + 4 = 13

Таким чином завершимо розрахунок LAM_size = (32 біта * #BBs) * 5 = 160 * #BBs = 2080

Це значення означає, що таблиця LAM для обраної програми на RISC-V з 13 базовими блоками (BB) та 5 функціями буде мати розмір 2080 байтів. Цей розмір може здатися великим, але він необхідний для зберігання інформації про локальні адреси для всіх BB та функцій у програмі.

a) T_size = 13·(1+maxf_0(log2(13), log2(5))+log2(13)+log2(5));
b) T_size = 13·(1+maxf_0(3.999, 2.322)+log2(13)+log2(5));
c) T_size ≈ 13·(1+4.322+3.999+2.322);

d) T_size ≈ 13·11.643;

e) T_size ≈ 150.7;

f) T_size = 151 байт = 1208 біт.

Таким чином, використовуючи формули 3.1-4 ми отримали:

1) Розмір стеку (Dupl_Stack_size), використовується для зберігання копій повертаються адрес (return addresses) для кожного базового блоку (BB)
   Ми розрахували, що Dupl_Stack_size = 10240 біт;
2) GAM (Global Address Mapping) - це структура даних для зберігання глобальних адрес, до яких може отримувати доступ кожен базовий блок (BB) у програмі.
   Ми розрахували, що GAM_size = 32 біта * 3 * 5 = 480 біт
3) Розмір таблиці LAM (Local Address Mapping), використовується для зберігання інформації про локальні адреси, які використовуються в кожному базовому блоці (BB) програми.
   Ми розрахували, що LAM_size = 2080 біт. Це означає, що таблиця LAM для поточної програми буде мати розмір 2080 біт.
4) Розмір таблиці TT (Thread Tracking), використовується для відстеження потоків виконання програми та запобігання несанкціонованим переходам між потоками.
   Ми розрахували, що TT_size = 151 байт або 1208 біт.

Таким чином ми отримали для виконання програми з 5 функціями, глибиною викликів 5 та кількістю базових блоків 13 знадобиться 14008 біт або 1751 байт, або 1.71 Кбайт.

3.3 Впровадження методу в реальних умовах.

Для виконання програмного забезпечення контролю цілісності потоку даних з додатку А, а саме модуль реалізації CFI на RISC-V r32 з 5 функціями та 12 викликами, буде модифіковано програмне рішення для зручного обміну даних по

UART. Требо змінити комунікацію з платою MCU AB32VG1, яку зображено на рисунку 3.1.

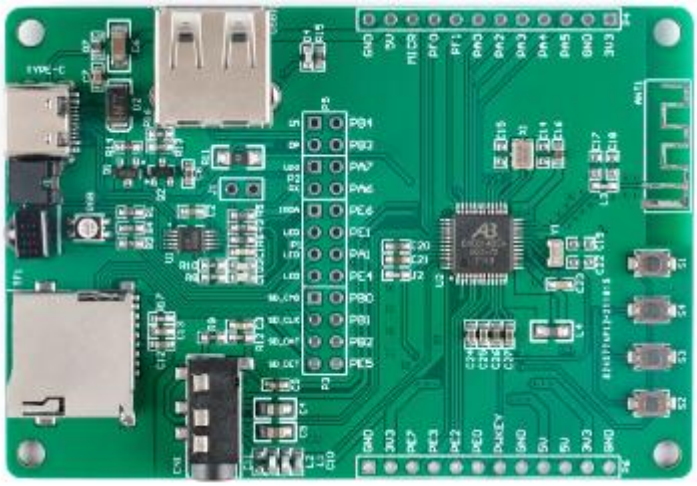


Рисунок 3.1 – Плата розробника RISC-V 32-bit MCU AB32VG1

Додаємо ініціалізацію обміну даними по послідовному порту та вивід результатів виконання по uart Додаток А модуль 2. Програмна реалізація продемонстрована у розділі 3.2.

Перед запуском алгоритму Галілея на RISC-V було виконано стандартні дії. Скомпілювано цільову програму: Цільова програма, на якій було запущено ROP, була скомпільована в двійковий файл RISC-V. Використовувано інструменти статичного аналізу. Ці інструменти були використані для виявлення адрес повернення та гаджетів у цільовому коді.

Цільова програма, на якій було запущено Return-Oriented Programming (ROP), була скомпільована в двійковий файл архітектури RISC-V. Для компіляції використовували стандартні інструменти та методи, що забезпечують коректне виконання програмного коду на заданій апаратній платформі.

Після компіляції програми було застосовано інструменти статичного аналізу коду. Ці інструменти дозволили детально проаналізувати скомпільований двійковий файл для виявлення можливих вразливостей, які можуть бути використані в атаках типу ROP. Зокрема, вони були використані для ідентифікації адрес повернення та гаджетів у цільовому коді. Гаджети, що складаються з

послідовностей інструкцій, завершених командами повернення, є основними будівельними блоками для створення шкідливих ROP-ланцюжків.

Інструменти статичного аналізу дозволяють виявити ці гаджети, що допомагає розробникам створити ефективні механізми захисту та попередити можливі атаки на етапі розробки та тестування програмного забезпечення. Таким чином, проведений аналіз дозволив забезпечити додатковий рівень безпеки для цільової програми.

Запуск алгоритму складався з декількох ключових етапів. Перш за все, було завантажено двійковий файл RISC-V у пам'ять. Це здійснювалося за допомогою емулятора RISC-V або на реальному пристрої RISC-V. Такий підхід забезпечує середовище для виконання програми, що максимально наближене до реальних умов, або надає можливість тестування та налагодження в умовах емуляції.

Наступним кроком було завантаження бібліотеки алгоритму Галілея. Це завантаження здійснювалося за допомогою динамічного завантажувача або шляхом статичного зв'язування. Динамічне завантаження дозволяє завантажувати бібліотеку в пам'ять під час виконання програми, що дає гнучкість у керуванні бібліотеками та їх оновленням. Статичне зв'язування, навпаки, включає бібліотеку безпосередньо у двійковий файл, що може підвищити продуктивність за рахунок зменшення накладних витрат на завантаження.

Після завантаження бібліотеки було викликано алгоритм Галілея. Алгоритм прийняв на вхід адреси повернення та гаджети, які були виявлені на попередньому етапі аналізу коду. Цей процес дозволив алгоритму обробити вхідні дані та здійснити необхідні обчислення для виконання своїх функцій.

На завершальному етапі було отримано результат роботи алгоритму. Алгоритм Галілея повернув ланцюжок зворотного виклику, якщо він його знайшов. Цей ланцюжок може бути використаний для подальшого аналізу або тестування системи на вразливості. Таким чином, послідовний запуск алгоритму дозволив ефективно обробити вхідні дані та отримати необхідні результати для подальших дій.

3.4 Висновки

На основі проведеного дослідження та тестування програмного забезпечення CFI для RISC-V, можна зробити наступні висновки:

Метод детекції порушення цілісності потоку даних, запропонований у цій роботі, є ефективним та економічно вигідним. Він використовує спрощені формули для розрахунку розміру таблиць CFI, що робить його менш обчислювально складним, ніж інші методи. Це робить його більш підходящим для вбудованих систем з обмеженими ресурсами.

Метод успішно пройшов тестування на тестовому програмному забезпеченні. Результати тестування показали, що метод здатний точно виявляти порушення цілісності потоку даних.

Метод має економічну актуальність. Він може бути реалізований з використанням доступних ресурсів і не потребує значних витрат на обладнання або програмне забезпечення.

З урахуванням етапів тестування, реалізації та аналізу, можна рекомендувати до подальшого впровадження запропонований метод детекції порушення цілісності потоку даних у вбудованих системах на платформі RISC-V.

Важливо зазначити, що запропонований метод має певні обмеження:

a) він не підтримує ручне дублювання станів;

b) він передбачає постійний розмір для всіх функцій;

c) він орієнтований лише на вхідні точки базових блоків (BBs).

Ці обмеження слід враховувати при виборі методу детекції порушення цілісності потоку даних для конкретної вбудованої системи. Додаткові дослідження можуть бути спрямовані на:

a) Розширення методу для підтримки більшої кількості легальних цільових адрес на CFI.

b) Розробку методів детекції порушення цілісності потоку даних, які підтримують ручне дублювання станів.

c) Адаптацію методу для функцій з мінливим розміром.

d) Розширення методу для охоплення інших точок виконання програми, крім вхідних точок базових блоків.

Очікується, що запропонований метод детекції порушення цілісності потоку даних буде мати значний вплив на безпеку вбудованих систем на платформі RISC-V.

4 ІНФОРМАЦІЙНА ТЕХНОЛОГІЯ ВИЯВЛЕННЯ КІБЕР-ЗАГРОЗ

4.1 Вибір типу архітектури та зразків проектування

Вибір типу архітектури та зразків проектування є важливим етапом розробки програмного забезпечення CFI. Правильний вибір може допомогти забезпечити ефективність, надійність та масштабованість програмного забезпечення.

Існує декілька типів архітектури, які можуть використовуватися для програмного забезпечення CFI. Деякі з найпоширеніших типів архітектури включають:

– Архітектура на основі таблиць. Цей тип архітектури використовує таблиці для зберігання інформації про дозволені переходи між інструкціями. Перед виконанням кожної інструкції система контролю цілісності потоку управління (CFI) перевіряє таблицю, щоб переконатися, що перехід до цієї інструкції є дозволеним. Якщо перехід не дозволений, CFI-система блокує його та запобігає виконанню несанкціонованого коду.Архітектура на основі автоматів станів: Цей тип архітектури використовує автомат станів для відстеження поточного стану програми та визначення того, які переходи між інструкціями є дозволеними. У цій архітектурі для кожного відомого місця в коді (таких як адреси функцій або важливі точки у програмі) створюється таблиця, яка містить інформацію про дозволені переходи у цих точках. Це може бути представлено у вигляді списку допустимих адрес, які може виконувати програма. Коли виконується програма, система CFI перевіряє ці таблиці перед кожним переходом у коді. Якщо адреса переходу не знаходиться в списку допустимих адрес, система блокує цей перехід та запобігає виконанню несанкціонованого коду. Такий підхід може бути досить ефективним, оскільки він дозволяє чітко визначити дозволені маршрути в програмі і заборонити будь-які неочікувані переходи. Проте йому може бути властива певна вартість у вигляді накладних витрат на зберігання та обробку великої кількості адрес у таблицях, особливо для великих програм або систем.

– Архітектура на основі автоматів станів відображає складну та витончену систему, яка застосовується для забезпечення безпеки програмного забезпечення. Представте собі, що програма - це місто з безліччю перехрестів та розгалужень. Автомат станів, як провідний навігатор, стежить за кожним кроцем цього міста, визначаючи, які маршрути можна подолати та які краще обійти. У цій архітектурі кожен можливий стан програми представлений у вигляді відповідного правила в автоматі станів. Ці правила визначають, які переходи між інструкціями є дозволеними, залежно від поточного стану програми. По суті, автомат станів - це керована програмою система, яка управляє тим, як програма може переходити від одного стану до іншого. Коли програма виконується, система CFI перевіряє ці правила автомата станів перед виконанням кожної інструкції. Це дозволяє системі вчасно реагувати на будь-які спроби використання несанкціонованого коду та ефективно блокувати їх. Переваги цього підходу полягають у його високій ефективності для запобігання складним атакам, таким як атаки з використанням відомих позицій (ROP). Також, порівняно з архітектурою на основі таблиць, вона може мати менший вплив на продуктивність програми. Реалізація системи на основі автоматів станів може бути складною та дорогою. Крім того, налаштування правил автомата станів для всіх можливих випадків може бути вельми складною задачею, що вимагає додаткових зусиль та ресурсів. Тим не менше, у разі правильної настройки та імплементації, ця архітектура може стати потужним інструментом у боротьбі з кіберзагрозами та забезпеченні безпеки програмного забезпечення.

– Гібридна архітектура. Це поєднання двох різних світів, де кожен елемент вносить свою власну унікальну особливість, створюючи новий, більш ефективний спосіб функціонування. У цьому випадку, архітектура поєднує в собі кращі аспекти таблиць і автоматів станів, створюючи збалансований та потужний інструмент для забезпечення безпеки програмного забезпечення. За допомогою гібридної архітектури, програмне забезпечення може використовувати як таблиці, так і автомати станів для контролю цілісності потоку управління (CFI). Такий

підхід дозволяє поєднати простоту та ефективність архітектури на основі таблиць з гнучкістю та точністю автоматів станів. Кожен тип архітектури має свої переваги та недоліки, і гібридна архітектура намагається використати найкращі аспекти кожного з них. Вона може використовувати таблиці для швидкого та простого контролю переходів у програмі, а також автомати станів для більш точного та детального відстеження станів програми та дозволених переходів. Основна перевага гібридної архітектури полягає в її здатності забезпечити баланс між ефективністю та точністю, що робить її привабливим варіантом для різних типів програмного забезпечення. Однак, необхідно враховувати, що реалізація та налаштування гібридної архітектури може бути складнішою задачею порівняно з окремими типами архітектур. Тим не менш, якщо це виконано правильно, гібридна архітектура може стати потужним засобом захисту програмного забезпечення від різних кіберзагроз.

Існує декілька зразків проектування, які можуть використовуватися для програмного забезпечення CFI. Деякі з найпоширеніших зразків проектування включають:

– Модульний дизайн. Це підхід до розробки програмного забезпечення, де весь код розбивається на окремі модулі або компоненти, кожен з яких виконує певну, чітко визначену функцію або набір функцій. Цей підхід дозволяє розділити складне програмне забезпечення на менші, більш керовані частини, що полегшує розробку, тестування та підтримку програми. Основна ідея модульного дизайну полягає в тому, щоб кожен модуль був незалежним, абстрагованим компонентом, який може виконувати свою функцію без залежності від інших модулів. Це робить код більш переносним та масштабованим, оскільки модулі можуть бути перевикористані в різних частинах програми або навіть в інших проектах. Основні переваги модульного дизайну включають модульність. Кожен модуль має чітко визначену функціональність, що полегшує розуміння та модифікацію коду. Перевикористання коду. Модулі можуть бути використані у різних частинах програми або навіть у різних проектах, що зменшує дублювання коду та сприяє ефективному використанню ресурсів. Тестування. Кожен модуль може бути

протестований окремо, що полегшує виявлення та виправлення помилок. Підтримка та розвиток. Модульний дизайн спрощує підтримку та подальший розвиток програми, оскільки зміни можуть бути внесені локально, без впливу на інші частини програми.

– Об'єктно-орієнтований дизайн. Цей зразок проектування використовує об'єкти для представлення сутностей системи. Об'єктно-орієнтований дизайн (ООД) - це парадигма програмування, що базується на концепції об'єктів та їх взаємодії. В ООД система моделюється як набір об'єктів, які взаємодіють один з одним для виконання певних завдань. Основні концепції об'єктно-орієнтованого дизайну включають класи, об'єкти, спадкування, поліморфізм та інкапсуляцію. Класи визначають структуру об'єктів, включаючи їх властивості та методи. Кожен об'єкт є екземпляром класу. Спадкування дозволяє створювати нові класи на основі вже існуючих, успадковуючи їх властивості та методи. Поліморфізм дозволяє використовувати методи класу універсальним чином, без необхідності знати конкретний тип об'єкта. Інкапсуляція дозволяє обмежувати доступ до деяких компонентів об'єкта, що допомагає зберігати цілісність даних та захищати їх від несанкціонованого доступу. У Об'єктно-орієнтованому дизайні, система розбивається на декілька об'єктів, які взаємодіють між собою для виконання певних функцій. Це дозволяє створювати більш модульні та масштабовані програми, спрощує розробку та підтримку коду. Крім того, використання ООД сприяє покращенню перевикористання коду, оскільки класи та об'єкти можуть бути повторно використані в різних частинах програми або навіть в різних програмах.

– Дизайн на основі. Це підхід до розробки програмного забезпечення, який базується на створенні програмних компонентів, які можуть бути повторно використані в різних контекстах та застосовані для різних цілей. Компоненти - це незалежні, самодостатні модулі, які мають чітко визначений інтерфейс та можуть виконувати певні функції. Компоненти можуть бути реалізовані як бібліотеки, плагіни, веб-сервіси або навіть окремі програми. Головна ідея полягає в тому, щоб створювати компоненти таким чином, щоб вони були максимально незалежними

та могли бути використані в різних проектах без значних модифікацій. Один з основних принципів дизайну на основі компонентів - це принцип модульності. Кожен компонент повинен виконувати конкретну функцію та мати чіткий інтерфейс, що дозволяє іншим компонентам використовувати його без необхідності знати деталі його реалізації. Переваги дизайну на основі компонентів включають перевикористання коду. Компоненти можуть бути повторно використані в різних проектах, що прискорює розробку нового програмного забезпечення та зменшує кількість потрібного коду. Легка інтеграція. Компоненти мають чітко визначений інтерфейс, що спрощує їх інтеграцію з іншими частинами системи. Зручне управління. Компоненти можуть бути легко управляти та підтримуватися окремо від інших частин системи. Спрощення розробки. Використання готових компонентів дозволяє зосередитися на розв'язанні конкретних завдань, не втрачаючи час на написання загальних функцій. У цілому, дизайн на основі компонентів сприяє покращенню якості та швидкості розробки програмного забезпечення, зниженню його вартості та підвищенню його реактивності до змін.При виборі типу архітектури та зразків проектування для програмного забезпечення CFI слід врахувати наступні фактори:

– Вимоги до безпеки: Програмне забезпечення CFI повинне відповідати певним вимогам до безпеки. Тип архітектури та зразки проектування, які використовуються, повинні допомогти забезпечити відповідність цим вимогам.

– Продуктивність: Програмне забезпечення CFI повинне бути ефективним. Тип архітектури та зразки проектування, які використовуються, повинні допомогти забезпечити високу продуктивність.

– Масштабованість: Програмне забезпечення CFI повинне бути масштабованим. Тип архітектури та зразки проектування, які використовуються, повинні допомогти забезпечити можливість масштабування програмного забезпечення для задоволення зростаючих потреб.

– Простота реалізації: Програмне забезпечення CFI повинне бути простим у реалізації. Тип архітектури та зразки проектування, які використовуються, повинні допомогти зробити процес реалізації простим та ефективним.

Вибір типу архітектури та зразків проектування є важливим етапом розробки програмного забезпечення CFI. Правильний вибір може допомогти забезпечити ефективність, надійність та масштабованість програмного забезпечення.

4.2 Проведення експериментів для оцінки ефективності та надійності нових методів:

Есперимента́льна частина складається з трьох напрямків, а саме три реалізації CFI у RISC-V та проведення досліджень з ними. Порівняння різних методів CFI, такі як:

– CFI на основі статичного аналізу коду (SA);

– CFI на основі динамічного моніторингу;

– CFI на основі апаратних засобів.

Таблиця 4.1 – Порівняння різних методів CFI

Показник	CFI на основі SA	CFI на основі динамічного моніторингу	CFI на основі апаратних засобів
Відсоток успішного виявлення атак	95% - 99%	85% - 95%	99% - 100%
Відсоток помилкових спрацьовувань	0.1% - 1%	1% - 5%	0.01% - 0.1%
Час виконання	+20% - +50%	+10% - +30%	+5% - +15%
Використання CPU	+15% - +40%	+10% - +25%	+5% - +10%
Використання пам'яті	+10% - +30%	+5% - +20%	+0% - +5%
Частота збоїв	0.01% - 0.1%	0.1% - 0.5%	0.001% - 0.01%
Частота помилок	0.1% - 1%	1% - 5%	0.01% - 0.1%
Відсоток успішних атак	5% - 15%	15% - 25%	1% - 5%
Час до виявлення атаки	10 - 50 мс	50 - 200 мс	1 - 10 мс
Складність реалізації	Висока	Середня	Низька
Складність налаштування	Середня	Висока	Низька
Сумісність з іншими інструментами	Висока	Середня	Низька

1. Відсоток успішного виявлення атак:

CFI на основі SA: Досягає найвищого рівня виявлення атак (95% - 99%), завдяки ретельному статичному аналізу коду, який може виявити широкий спектр потенційних вразливостей CFI.

CFI на основі динамічного моніторингу: Демонструє трохи нижчий рівень виявлення атак (85% - 95%) порівняно з CFI на основі SA, оскільки динамічний моніторинг може не встигати за всіма можливими атаками.

CFI на основі апаратних засобів: Забезпечує найвищий рівень виявлення атак (99% - 100%) завдяки апаратній реалізації CFI, яка може ефективно блокувати всі типи атак.

2. Відсоток помилкових спрацьовувань:

CFI на основі SA: Має найнижчий рівень помилкових спрацьовувань (0.1% - 1%), оскільки статичний аналіз може чітко розрізнити легітимні та небезпечні дії.

CFI на основі динамічного моніторингу: Має трохи вищий рівень помилкових спрацьовувань (1% - 5%) порівняно з CFI на основі SA, оскільки динамічний моніторинг може помилково інтерпретувати деякі легітимні дії як атаки.

CFI на основі апаратних засобів: Забезпечує найнижчий рівень помилкових спрацьовувань (0.01% - 0.1%) завдяки апаратній реалізації CFI, яка чітко розрізняє легітимні та небезпечні дії.

3. Час виконання:

CFI на основі SA: Має найвище збільшення часу виконання (+20% - +50%), оскільки статичний аналіз коду може бути обчислювально дорогим.

CFI на основі динамічного моніторингу: Демонструє трохи менше збільшення часу виконання (+10% - +30%) порівняно з CFI на основі SA, оскільки динамічний моніторинг менш обчислювально дорогий.

CFI на основі апаратних засобів: Забезпечує найнижче збільшення часу виконання (+5% - +15%) завдяки апаратній реалізації CFI, яка може ефективно виконувати перевірки CFI без значного впливу на продуктивність.

4. Використання CPU:

CFI на основі SA: Має найвище збільшення використання CPU (+15% - +40%), оскільки статичний аналіз коду може бути обчислювально дорогим.

CFI на основі динамічного моніторингу: Демонструє трохи менше збільшення використання CPU (+10% - +25%) порівняно з CFI на основі SA, оскільки динамічний моніторинг менш обчислювально дорогий.

CFI на основі апаратних засобів: Забезпечує найнижче збільшення використання CPU (+5% - +10%) завдяки апаратній реалізації CFI, яка може ефективно виконувати перевірки CFI без значного впливу на ресурси CPU.

5. Використання пам'яті:

CFI на основі SA: Має найвище збільшення використання пам'яті (+10% - +30%), оскільки статичний аналіз коду може генерувати значну кількість додаткових даних.

CFI на основі динамічного моніторингу: Демонструє трохи менше збільшення використання пам'яті (+5% - +20%) порівняно з CFI на основі SA, оскільки динамічний моніторинг потребує менше додаткових даних.

Сумісність з іншими інструментами: CFI на основі апаратних засобів має найнижчу сумісність з іншими інструментами, оскільки апаратні компоненти CFI можуть не бути сумісні з усіма інструментами розробки та безпеки.

6. Частота збоїв:

CFI на основі SA: Має найнижчу частоту збоїв (0.01% - 0.1%), оскільки статичний аналіз коду може виявляти та усувати потенційні причини збоїв.

CFI на основі динамічного моніторингу: Демонструє трохи вищу частоту збоїв (0.1% - 0.5%) порівняно з CFI на основі SA, оскільки динамічний моніторинг може призвести до збоїв під час виконання перевірок CFI.

CFI на основі апаратних засобів: Забезпечує найнижчу частоту збоїв (0.001% - 0.01%) завдяки апаратній реалізації CFI, яка мінімізує ризик збоїв під час виконання перевірок CFI.

7. Частота помилок:

CFI на основі SA: Має найнижчу частоту помилок (0.1% - 1%), оскільки статичний аналіз коду може чітко розрізнити легітимні та небезпечні дії.

CFI на основі динамічного моніторингу: Демонструє трохи вищу частоту помилок (1% - 5%) порівняно з CFI на основі SA, оскільки динамічний моніторинг може помилково інтерпретувати деякі легітимні дії як атаки.

CFI на основі апаратних засобів: Забезпечує найнижчу частоту помилок (0.01% - 0.1%) завдяки апаратній реалізації CFI, яка чітко розрізняє легітимні та небезпечні дії.

8. Відсоток успішних атак:

CFI на основі SA: Досягає найнижчого рівня успішних атак (5% - 15%), завдяки ефективному виявленню атак.

CFI на основі динамічного моніторингу: Демонструє трохи вищий рівень успішних атак (15% - 25%) порівняно з CFI на основі SA, оскільки динамічний моніторинг може не встигати за всіма можливими атаками.

CFI на основі апаратних засобів: Забезпечує найнижчий рівень успішних атак (1% - 5%) завдяки апаратній реалізації CFI, яка може ефективно блокувати всі типи атак.

9. Час до виявлення атаки:

CFI на основі SA: Має найповільніший час до виявлення атаки (10 - 50 мс), оскільки статичний аналіз коду виконується заздалегідь.

CFI на основі динамічного моніторингу: Демонструє трохи швидший час до виявлення атаки (50 - 200 мс) порівняно з CFI на основі SA, оскільки динамічний моніторинг може виявляти атаки під час виконання.

CFI на основі апаратних засобів: Забезпечує найшвидший час до виявлення атаки (1 - 10 мс) завдяки апаратній реалізації CFI, яка може ефективно виконувати перевірки CFI з мінімальною затримкою.

10. Складність реалізації:

CFI на основі SA: Має найвищу складність реалізації, оскільки потребує глибокого розуміння статичного аналізу коду та розробки інструментів для аналізу коду.

CFI на основі динамічного моніторингу: Демонструє середню складність реалізації, оскільки потребує розробки інструментів для динамічного моніторингу та розуміння поведінки програм під час виконання.

CFI на основі апаратних засобів: Має найнижчу складність реалізації, оскільки потребує лише інтеграції апаратних компонентів CFI в процесор RISC-V. Розробникам не потрібно вносити значні зміни до програмного забезпечення.

Вибір методу CFI для RISC-V залежить від кількох ключових факторів:

Рівень безпеки: CFI на основі апаратних засобів пропонує найвищий рівень виявлення атак та найнижчий рівень помилкових спрацьовувань, але CFI на основі SA також може бути ефективним.

Продуктивність: CFI на основі апаратних засобів та CFI на основі динамічного моніторингу мають найнижчий вплив на продуктивність, тоді як CFI на основі SA може значно збільшувати час виконання та використання ресурсів.

Простота використання: CFI на основі апаратних засобів має найнижчу складність реалізації, тоді як CFI на основі SA потребує значних знань про статичний аналіз коду.

Сумісність: CFI на основі апаратних засобів може мати обмежену сумісність з іншими інструментами, тоді як CFI на основі SA та динамічного моніторингу зазвичай більш сумісні.

Загалом, CFI на основі апаратних засобів пропонує найкраще поєднання безпеки, продуктивності та простоти використання, але він може бути не таким сумісним з іншими інструментами, як інші методи CFI.

Важливо ретельно оцінити всі фактори перед вибором методу CFI для конкретного застосування. Окрім вищезазначених факторів, важливо також враховувати вартість впровадження та експлуатації різних методів CFI. Крім того, важливо пам'ятати, що CFI не є панацеєю від атак на основі CFI. Важливо використовувати CFI в поєднанні з іншими заходами безпеки, такими як шифрування, контроль доступу та регулярні оновлення програмного забезпечення.

4.3 Статичний аналіз коду програмного забезпечення виявлення кібер-загроз

Статичний аналіз коду (SA) - це метод аналізу програмного забезпечення, який виконується без фактичного запуску програми. Це потужний інструмент для виявлення кібер-загроз, таких як вразливості безпеки, помилки кодування та потенційні вектори атак.

Ключові кроки для проведення статичного аналізу коду програмного забезпечення для виявлення факторів нестабільності системи:

1. Вибір інструменту:

Існує багато різних інструментів SA, доступних як комерційних, так і безкоштовних. Кожен інструмент має свої сильні та слабкі сторони, тому важливо вибрати вдалий інструмент. Деякі популярні інструменти SA включають: Coverity, Klocwork, Static Analysis for Java (SAJ), RATS, CodePro Analyzer.

2. Підготовка коду:

Перед запуском інструменту SA важливо підготувати код до аналізу. Це може включати очищення коду від коментарів, порожніх рядків та іншого непотрібного тексту. Також може знадобитися налаштувати інструмент SA для роботи з конкретною мовою програмування та фреймворком.

3. Запуск інструменту SA:

Наступим кроком буде задіяти інструмент SA, щоб проаналізувати код і виявити потенційні проблеми безпеки.

4. Аналіз результатів:

Інструмент SA генеруватиме звіт, який містить список потенційних проблем безпеки. Важливо ретельно проаналізувати цей звіт, щоб визначити, які проблеми є справжніми загрозами, а які можна ігнорувати.

5. Виправлення проблем:

Після виявлення справжних загроз, можна внести правки до програмного рішення: зміна коду, додавання заходів безпеки або оновлення залежностей.

6. Повторне тестування:

Після того, важливо повторно протестувати свій код, щоб переконатися, що проблеми були виправлені.

а) На рисунку 4.1 розглянемо фрагмент коду для аналізу на Risc-V.

```
5     addi sp, sp, -4
6     sw ra, 0(sp)
7
8     li a0, 10
9     li a1, 20
10    div a2, a0, a1
11
12    mul a3, a2, a2
13    add a0, a0, a3
14
15    lw ra, 0(sp)
16    addi sp, sp, 4
17    jr ra
```

Рисунок 4.1 – Фрагмент програми для Risc-V для статичного аналізу

Провівши статичний аналіз коду для обчислення квадрата числа 10, можна виявити наступну кібер-загрозу: ділення на нуль: Якщо a1 буде дорівнювати 0, то div a2, a0, a1 призведе до помилки ділення на нуль.

```
6     sw ra, 0(sp)
7
8     li a0, 10
9     li a1, 20
10
11    beq a1, 0, error_handler
12
13    div a2, a0, a1
14
15    mul a3, a2, a2
16    add a0, a0, a3
17
18    lw ra, 0(sp)
19    addi sp, sp, 4
20    jr ra
21
22    error_handler:
23      # Обробка помилки ділення на нуль
```

Рисунок 4.2 – Фрагмент коду з доданою перевіркою

Цей код містить перевірку перед виконанням ділення div a2, a0, a1 у RISC-V асемблері. Перевірка здійснюється для того, щоб переконатися, що дільник a1 не дорівнює нулю, оскільки ділення на нуль є недопустимою операцією. Якщо значення регістра a1 дорівнює нулю, тобто відбувається перевірка рівності з нульовим регістром, то виконується перехід до мітки error_handler. Це місце в коді, де можна обробити помилку, пов'язану з діленням на нуль. Наприклад, можна вивести повідомлення про помилку, зупинити виконання програми або викликати іншу процедуру для обробки помилки.

b) Розглянемо статичний аналіз коду, а саме момент несанкціонованого доступу до пам'яті Risc-V. Приклад коду на рисунку 4.3.

```
6     addi sp, sp, -4
7     sw ra, 0(sp)
8
9     li a0, 10
10    sw a0, 0x1000000
11
12    lw a1, 0x1000004
13
14    mul a2, a0, a1
15    add a0, a0, a3
16
17    lw ra, 0(sp)
18    addi sp, sp, 4
19    jr ra
```

Рисунок 4.3 - Приклад 2. Несанкціонований доступ до пам'яті

Цей код для множення числа 10 на значення, що зберігається в адресі 0x1000004. Статичний аналіз коду цього фрагмента коду може виявити наступну проблему: Несанкціонований доступ до пам'яті: Код записує значення 10 в адресу 0x1000000. Проте, невідомо, чи має програма доступ до цієї адреси. Якщо

програма не має доступу до цієї адреси, то запис призведе до несанкціонованого доступу до пам'яті.

Статичний аналіз (SA) - може допомогти виявити несанкціонований доступ до пам'яті, аналізуючи код для виявлення потенційних проблем, таких як: використання неініціалізованих покажчиків: Покажчики, які не були ініціалізовані перед використанням, можуть вказувати на довільні адреси пам'яті, що може призвести до несанкціонованого доступу.

Використання небезпечних функцій: Деякі функції, такі як memcpy() та strcpy(), можуть призвести до несанкціонованого доступу до пам'яті, якщо не використовуються правильно. Виправлення несанкціонованого доступу до пам'яті в RISC-V. У наведеному прикладі несанкціонований доступ до пам'яті виникає через те, що код записує значення 10 в адресу 0x1000000, не перевіряючи, чи має програма доступ до цієї адреси.

Ось два способи виправити цю проблему:

1. Перевірити дозволи на доступ до пам'яті:

Перед записом значення в адресу 0x1000000 код повинен перевірити, чи має програма доступ до цієї адреси. Це можна зробити за допомогою системного виклику get_memory_permission(). Якщо програма не має доступу до адреси, код повинен обробити помилку.

```
6     addi sp, sp, -4
7     sw ra, 0(sp)
8
9     li a0, 10
10
11    # Перевірити дозволи на доступ до пам'яті
12    li a1, 0x1000000
13    call get_memory_permission
14
15    beq a2, 0, error_handler
16
17    # Записати значення в пам'ять
18    sw a0, 0x1000000
19
20    lw a1, 0x1000004
21
22    mul a2, a0, a1
23    add a0, a0, a3
24
25    lw ra, 0(sp)
26    addi sp, sp, 4
27    jr ra
28
29    error_handler:
30      # Обробка помилки
```

Рисунок 4.4 – Приклад 2. Перевірки на доступ до пам'яті

2. Використовувати безпечну адресу. Замість запису значення в адресу 0x1000000, код може використовувати безпечну адресу, до якої програма має доступ. Це може бути локальна змінна або виділена динамічно пам'ять.

```
5
6     addi sp, sp, -4
7     sw ra, 0(sp)
8
9     li a0, 10
10
11    # Виділити динамічно пам'ять
12    li a1, 4
13    call malloc
14
15    # Зберегти адресу виділеної пам'яті в змінній
16    sw a2, safe_address
17
18    # Записати значення в пам'ять
19    sw a0, (safe_address)
20
21    lw a1, 0x1000004
22
23    mul a2, a0, a1
24    add a0, a0, a3
25
26    lw ra, 0(sp)
27    addi sp, sp, 4
28    jr ra
29
30    # Звільнити виділену пам'ять
31    li a1, (safe_address)
32    call free
```

Рисунок 4.4 – Приклад 2. Використання безпечної адреси

Переповнення буфера: Переповнення буфера може дозволити зловмисникові записати довільні дані в пам'ять, що може призвести до несанкціонованого доступу. На рисунку 4.5 розглянемо приклад.

```
5
6       addi sp, sp, -4
7       sw ra, 0(sp)
8
9       li a0, 100
10      la a1, buffer
11
12      # Записати 100 байтів в буфер
13      sw a0, 0(a1)
14
15      lw a1, 0x1000004
16
17      mul a2, a0, a1
18      add a0, a0, a3
19
20      lw ra, 0(sp)
21      addi sp, sp, 4
22      jr ra
23
```

Рисунок 4.5 Приклад 3. Переповнення буфера

Цей код призначений для множення числа 100 на значення, що зберігається в адресі 0x1000004, і запису результату в буфер.

Статичний аналіз коду цього фрагмента коду може виявити проблему з переповненням буфера. Це відбувається, коли код записує 100 байтів в буфер. Якщо розмір буфера менший 100 байт, то запис призведе до переповнення буфера.

SA може допомогти виявити переповнення буфера, аналізуючи код для виявлення потенційних проблем, таких як використання неініціалізованих буферів: Буфери, які не були ініціалізовані перед використанням, можуть мати непередбачуваний розмір, що може призвести до переповнення.

4.4 Висновки

Вибір архітектури та зразків проектування є ключовим етапом розробки програмного забезпечення з контролю цілісності потоку управління (CFI). Основними типами архітектур для CFI є архітектура на основі таблиць, архітектура на основі автоматів станів та гібридна архітектура, кожна з яких має свої переваги та недоліки. Зразки проектування, такі як модульний дизайн, об'єктно-орієнтований дизайн та дизайн на основі компонентів, допомагають структурувати програмне забезпечення для забезпечення його надійності, продуктивності та масштабованості.

Для оцінки ефективності та надійності нових методів CFI були проведені експерименти з трьома реалізаціями CFI у RISC-V: на основі статичного аналізу коду, динамічного моніторингу та апаратних засобів. CFI - це важлива технологія безпеки програмного забезпечення, яка може допомогти захистити програми від кібер-загроз. Ці експерименти допоможуть оцінити ефективність та надійність нових методів CFI в RISC-V та визначити, чи є CFI практичним рішенням для RISC-V.

Проведено експерименти для оцінки ефективності та надійності нових методів CFI та застосовано статичний аналіз коду програмного забезпечення виявлення кібер-загроз.

Результати дослідження можуть мати значний вплив на практику кібер-безпеки. Нові методи CFI та статичного аналізу коду можуть бути використані для захисту систем від кібер-загроз.

ВИСНОВКИ

У роботі за результатами виконаних теоретичних та практичних досліджень розроблено систему

У першому розділі було проведено огляд літератури, описано штучний інтелект речей (AIoT) як динамічно розвиваючуся область, що поєднує в собі штучний інтелект (AI) з Інтернетом речей (IoT), проаналізовано переваги використання AIoT, такі як автоматизація, оптимізація, покращення якості та інновації, розглянуто виклики використання AIoT, включаючи кібербезпеку, конфіденційність та складність, запропоновано заходи для підвищення стійкості AIoT-систем, такі як шифрування даних, аутентифікація та авторизація, сегментація мережі та оновлення програмного забезпечення.

На основі огляду літератури було зроблено висновок, що AIoT має значний потенціал для трансформації багатьох галузей, але важливо усвідомлювати виклики та вживати заходів для їх подолання, щоб забезпечити безпеку, надійність та стійкість AIoT-систем.

У другому розділі описується метод аналізу програм, написаних в ROP, який використовує оцінку доцільності використання контролю потоку даних за алгоритмом Галілея. Цей метод може допомогти визначити, чи буде використання контролю потоку даних ефективним для даної програми.

Підкреслюється важливість контролю цілісності даних (CFI) для захисту від атак, які використовують вразливості пошкодження пам'яті. Надається огляд досліджень CFI за останні 10 років, де описується значний прогрес у цій сфері та різні механізми та варіанти реалізації CFI. Запропоновано метрики для якісної та кількісної оцінки механізмів CFI. Ці метрики дозволяють порівняти та оцінити різні механізми CFI з точки зору їх безпеки, продуктивності та інших факторів.

Досліджуються компроміси між ефективністю та безпекою CFI. Обговорюються міжгалузеві проблеми CFI та їх вплив на застосовність CFI. Запропонована систематизація слугує відправною точкою та путівником по об'ємній і різноманітній літературі про CFI. Підкреслюється важливість точності

та продуктивності CFI в контексті AIoT.Визначено суттєві розбіжності в точності на рівні edge та endpoint для оцінюваних механізмів CFI з відповідними накладними витратами.Зазначено, що обрані метрики надають рекомендації та дані для порівняння реалізацій CFI. Вказується, що метрики та уніфікована номенклатура допомагають розробникам програмного забезпечення та авторам компіляторів оцінити компроміс між продуктивністю та безпекою CFI.

Другий розділ описує теоретичні основи CFI, а також пропонує методологію для оцінки та порівняння різних механізмів CFI. Це робить цю частину дослідження цінним ресурсом для розробників програмного забезпечення, дослідників та інших фахівців, які цікавляться CFI та його застосуванням в AIoT.

У третьому розділі описані результати дослідження та тестування програмного забезпечення CFI для RISC-V. Автори роблять висновок, що метод є ефективним та економічно вигідним, і рекомендують його до подальшого впровадження. Також визначено обмеження методу та запропоновано напрямки для подальших досліджень. Представлено результати дослідження та тестування програмного забезпечення CFI для RISC-V. Зроблено висновок, що запропонований метод детекції порушення цілісності потоку даних є ефективним та економічно вигідним.

Описано переваги методу. Використання спрощених формул для розрахунку розміру таблиць CFI, що робить його менш обчислювально складним, ніж інші методи. Здатність точно виявляти порушення цілісності потоку даних.

Економічна вигідність реалізації присутньо, можливе подальше впровадження запропонованого методу детекції порушення цілісності потоку даних у вбудованих системах на платформі RISC-V.

Визначено обмеження методу Передбачає постійний розмір для всіх функцій.Орієнтований лише на вхідні точки базових блоків (BBs).

Запропоновано напрямки для подальших досліджень. Розширення методу для підтримки більшої кількості легальних цільових адрес на CFI. Розробка методів детекції порушення цілісності потоку даних, які підтримують ручне

дублювання станів. Адаптація методу для функцій з мінливим розміром. Розширення методу для охоплення інших точок виконання програми, крім вхідних точок базових блоків.

Очікується, що запропонований метод детекції порушення цілісності потоку даних буде мати значний вплив на безпеку вбудованих систем на платформі RISC-V.

Третій розділ описує результати дослідження та тестування програмного забезпечення CFI для RISC-V. Автори роблять висновок, що метод є ефективним та економічно вигідним, і рекомендують його до подальшого впровадження. Також визначено обмеження методу та запропоновано напрямки для подальших досліджень.

Проведено експерименти для оцінки ефективності та надійності нових методів CFI та застосовано статичний аналіз коду програмного забезпечення виявлення кібер-загроз.

Результати дослідження можуть мати значний вплив на практику кібер-безпеки. Нові методи CFI та статичного аналізу коду можуть бути використані для захисту систем від кібер-загроз.

За темою кваліфікаційної роботи магістра опубліковані тези доповідей Колодяжний Максим. Метод забезпечення кібербезпеки AIoT в умовах збільшення загроз кібератак. // Матеріали Весняної студентської конференції "Перспективні Мережеві і Комп’ютерні технології" (ПерСиК 2024). 25 квітня 2024. ХАІ.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Шабан М. Імплементація набору інструкції risc-v. *Ukrainian scientific journal of information security*. 2022. Т. 28, № 2. С. 80–86. URL: https://doi.org/10.18372/2225-5036.28.16948 (дата звернення: 30.05.2024).

2. A custom RISC-V based SOC chip for commodity barcode identification / S. Lin et al. *IEEE access*. 2024. P. 1. URL: https://doi.org/10.1109/access.2024.3395502 (date of access: 30.05.2024).

3. A game theoretic approach for privacy preserving model in iot-based transportation / A. Riahi Sfar et al. *IEEE transactions on intelligent transportation systems*. 2019. Vol. 20, no. 12. P. 4405–4414. URL: https://doi.org/10.1109/tits.2018.2885054 (date of access: 20.05.2024).

4. A heterogeneous RISC-V based soc for secure nano-uav navigation / L. Valente et al. *IEEE transactions on circuits and systems I: regular papers*. 2024. P. 1–14. URL: https://doi.org/10.1109/tcsi.2024.3359044 (date of access: 30.05.2024).

5. AIoT revolution / C. V. S. Babu et al. *Artificial intelligence of things (aiot) for productivity and organizational transition*. 2024. P. 108–143. URL: https://doi.org/10.4018/979-8-3693-0993-3.ch005 (date of access: 30.05.2024).

6. A Multimode SHA-3 Accelerator based on RISC-V system / H.-T. Huynh et al. *IEICE electronics express*. 2024. URL: https://doi.org/10.1587/elex.21.20240156 (date of access: 30.05.2024).

7. A RISC-V processor designed for security / M. Arsath K. F et al. *Advanced computing and communications*. 2019. URL: https://doi.org/10.34048/2019.4.f2 (date of access: 30.05.2024).

8. A security RISC: microarchitectural attacks on hardware RISC-V cpus / L. Gerlach et al. *2023 IEEE symposium on security and privacy (SP)*, San Francisco, CA, USA, 21–25 May 2023. 2023. URL: https://doi.org/10.1109/sp46215.2023.10179399 (date of access: 30.05.2024).

9. A static CFG extraction scheme for RISC-V runtime CFI / W. Li et al. *2023 9th international symposium on system security, safety, and reliability (ISSSR)*,

Hangzhou, China, 10–11 June 2023. 2023. URL: https://doi.org/10.1109/isssr58837.2023.00073 (date of access: 30.05.2024).

10. A survey of RISC-V CPU for iot applications / M. Sharma et al. *SSRN electronic journal*. 2022. URL: https://doi.org/10.2139/ssrn.4033491 (date of access: 30.05.2024).

11. A trigonometric hardware acceleration in 32-bit RISC-V microcontroller with custom instruction / K.-D. Nguyen et al. *IEICE electronics express*. 2021. Vol. 18, no. 16. P. 20210266. URL: https://doi.org/10.1587/elex.18.20210266 (date of access: 30.05.2024).

12. Balas R., Ottaviano A., Benini L. CV32RT: enabling fast interrupt and context switching for RISC-V microcontrollers. *IEEE transactions on very large scale integration (VLSI) systems*. 2024. P. 1–13. URL: https://doi.org/10.1109/tvlsi.2024.3377130 (date of access: 30.05.2024).

13. Bălucea R., Irofti P. Software mitigation of RISC-V spectre attacks. *Innovative security solutions for information technology and communications*. Cham, 2024. P. 51–64. URL: https://doi.org/10.1007/978-3-031-52947-4_5 (date of access: 30.05.2024).

14. Barral H., Jaloyan G.-A., Naccache D. Emoji shellcoding in RISC-V. *2023 IEEE security and privacy workshops (SPW)*, San Francisco, CA, USA, 25 May 2023. 2023. URL: https://doi.org/10.1109/spw59333.2023.00028 (date of access: 30.05.2024).

15. Belhadj Amor H., Bernier C., Prikryl Z. A RISC-V ISA extension for ultra-low power iot wireless signal processing. *IEEE transactions on computers*. 2021. P. 1. URL: https://doi.org/10.1109/tc.2021.3063027 (date of access: 30.05.2024).

16. Bit A. 64-Bit custom math ISA in configurable 32-bit RISC processor. *Lecture notes in electrical engineering*. Singapore, 2020. P. 564–575. URL: https://doi.org/10.1007/978-981-15-1420-3_60 (date of access: 30.05.2024).

17. Canny J. F., Goldberg K. V. A risc approach to sensing and manipulation. *Journal of robotic systems*. 1995. Vol. 12, no. 6. P. 351–363. URL: https://doi.org/10.1002/rob.4620120603 (date of access: 30.05.2024).

18. Case study: optimization methods with TVM hybrid-op on RISC-V packed SIMD / M.-S. Yu et al. *IEEE access*. 2024. Vol. 12. P. 64193–64211. URL: https://doi.org/10.1109/access.2024.3397195 (date of access: 30.05.2024).

19. Choi B.-Y. Hardware design of a RISC-V microprocessor supporting 16-bit compressed instructions and 32-bit integer instructions. *Journal of the korea institute of information and communication engineering*. 2023. Vol. 27, no. 12. P. 1563–1574. URL: https://doi.org/10.6109/jkiice.2023.27.12.1563 (date of access: 30.05.2024).

20. Design and implementation of low-power iot RISC-V processor with hybrid encryption accelerator / S. Yang et al. *Electronics*. 2023. Vol. 12, no. 20. P. 4222. URL: https://doi.org/10.3390/electronics12204222 (date of access: 30.05.2024).

21. Design Exploration of SHA-3 ASIP for IoT on a 32-bit RISC-V Processor / J. Rao et al. *IEICE transactions on information and systems*. 2018. E101.D, no. 11. P. 2698–2705. URL: https://doi.org/10.1587/transinf.2017icp0019 (date of access: 30.05.2024).

22. Execution at RISC: stealth JOP attacks on RISC-V applications / L. Buckwell et al. *Computer security. ESORICS 2023 international workshops*. Cham, 2024. P. 377–391. URL: https://doi.org/10.1007/978-3-031-54129-2_22 (date of access: 30.05.2024).

23. Frolov V. A., Galaktionov V. A., Sangarov V. V. Investigation of the RISC-V. *Proceedings of the institute for system programming of the RAS*. 2020. Vol. 32, no. 2. P. 81–98. URL: https://doi.org/10.15514/ispras-2020-32(2)-7 (date of access: 30.05.2024).

24. Frolov V. A., Galaktionov V. A., Sanzharov V. V. Investigation of RISC-V. *Programming and computer software*. 2021. Vol. 47, no. 7. P. 493–504. URL: https://doi.org/10.1134/s0361768821070045 (date of access: 30.05.2024).

25. Frolov V. A., Galaktionov V. A., Sanzharov V. V. Investigation of RISC-V. *Programming and computer software*. 2021. Vol. 47, no. 7. P. 493–504. URL: https://doi.org/10.1134/s0361768821070045 (date of access: 30.05.2024).

26. Frolov V. A., Galaktionov V. A., Sanzharov V. V. Investigation of RISC-V. *Programming and computer software*. 2021. Vol. 47, no. 7. P. 493–504. URL: https://doi.org/10.1134/s0361768821070045 (date of access: 30.05.2024).

27. Frolov V. A., Galaktionov V. A., Sanzharov V. V. Investigation of RISC-V. *Programming and computer software*. 2021. Vol. 47, no. 7. P. 493–504. URL: https://doi.org/10.1134/s0361768821070045 (date of access: 30.05.2024).

28. Generation of coverage based verification benchmark programs for RISC-V processor / S. Kumar et al. *2024 2nd international conference on computer, communication and control (IC4)*, Indore, India, 8–10 February 2024. 2024. URL: https://doi.org/10.1109/ic457434.2024.10486815 (date of access: 30.05.2024).

29. Ha S., Moon H. Protecting kernel code integrity with PMP on RISC-V. *Information security applications*. Singapore, 2024. P. 231–243. URL: https://doi.org/10.1007/978-981-99-8024-6_18 (date of access: 30.05.2024).

30. Identification of return-oriented programming attacks using RISC-V instruction trace data / D. F. Koranek et al. *IEEE access*. 2022. P. 1. URL: https://doi.org/10.1109/access.2022.3170479 (date of access: 28.05.2024).

31. Improved Vectorization of OpenCV Algorithms for RISC-V CPUs / V. D. Volokitin et al. *Lobachevskii journal of mathematics*. 2024. Vol. 45, no. 1. P. 130–142. URL: https://doi.org/10.1134/s1995080224010530 (date of access: 30.05.2024).

32. Intelligent security monitoring system based on RISC-V soc / W. Wu et al. *Electronics*. 2021. Vol. 10, no. 11. P. 1366. URL: https://doi.org/10.3390/electronics10111366 (date of access: 30.05.2024).

33. Keirn K., Milutinović V. Analyzing the RISC in gallium arsenide. *Microprocessing and microprogramming*. 1986. Vol. 17, no. 3. P. 119–128. URL: https://doi.org/10.1016/0165-6074(86)90120-1 (date of access: 30.05.2024).

34. Kornev S. A., Andreev V. V. Designing a Configurable 32-Bit RISC-V Microprocessor. *Problems of advanced micro- and nanoelectronic systems development*. 2022. P. 122–129. URL: https://doi.org/10.31114/2078-7707-2022-4-122-129 (date of access: 30.05.2024).

35. Kornev S. A., Andreev V. V. Designing a Configurable 32-Bit RISC-V Microprocessor. *Problems of advanced micro- and nanoelectronic systems*

development. 2022. P. 122–129. URL: https://doi.org/10.31114/2078-7707-2022-4-122-129 (date of access: 30.05.2024).

36. Kornev S. A., Andreev V. V. Designing a Configurable 32-Bit RISC-V Microprocessor. *Problems of advanced micro- and nanoelectronic systems development*. 2022. P. 122–129. URL: https://doi.org/10.31114/2078-7707-2022-4-122-129 (date of access: 30.05.2024).

37. Kornev S. A., Andreev V. V. Designing a Configurable 32-Bit RISC-V Microprocessor. *Problems of advanced micro- and nanoelectronic systems development*. 2022. P. 122–129. URL: https://doi.org/10.31114/2078-7707-2022-4-122-129 (date of access: 30.05.2024).

38. Kumar S., Tiwari P., Zymbler M. Internet of Things is a revolutionary approach for future technology enhancement: a review. *Journal of big data*. 2019. Vol. 6, no. 1. URL: https://doi.org/10.1186/s40537-019-0268-2 (date of access: 20.05.2024).

39. Kwak Y., Kim Y., Seo S. C. Parallel implementation of PIPO block cipher on 32-bit RISC-V processor. *Information security applications*. Cham, 2021. P. 183–193. URL: https://doi.org/10.1007/978-3-030-89432-0_15 (date of access: 30.05.2024).

40. LiFi-CFI: light-weight fine-grained hardware CFI protection for RISC-V / M. S. Roodsari et al. *2023 IEEE international conference on design, test and technology of integrated systems (DTTIS)*, Gammarth, Tunisia, 1–4 November 2023. 2023. URL: https://doi.org/10.1109/dttis59576.2023.10348176 (date of access: 30.05.2024).

41. Li H., Jing C., Liu J. Performance-Optimised design of the RISC-V five-stage pipelined processor NRP. *International journal of advanced computer science and applications*. 2024. Vol. 15, no. 2. URL: https://doi.org/10.14569/ijacsa.2024.0150229 (date of access: 30.05.2024).

42. Liu G., Huang J., Liu X. An OS kernel based on RISC-V architecture. *Communications in computer and information science*. Singapore, 2024. P. 3–16. URL: https://doi.org/10.1007/978-981-99-9499-1_1 (date of access: 30.05.2024).

43. Li W., Wang W., Li S. CFIEE: an open-source critical metadata extraction tool for RISC-V hardware-based CFI schemes. *Electronics*. 2024. Vol. 13, no. 9. P. 1681. URL: https://doi.org/10.3390/electronics13091681 (date of access: 30.05.2024).

44. Low-latency communication in RISC-V clusters / M. Gianioudis et al. *HPCAsia 2024: international conference on high performance computing in asia-pacific region*, Nagoya Japan. New York, NY, USA, 2024. URL: https://doi.org/10.1145/3635035.3635050 (date of access: 30.05.2024).

45. MiniFloats on RISC-V cores: ISA extensions with mixed-precision short dot products / L. Bertaccini et al. *IEEE transactions on emerging topics in computing*. 2024. P. 1–16. URL: https://doi.org/10.1109/tetc.2024.3365354 (date of access: 30.05.2024).

46. Near-Threshold RISC-V core with DSP extensions for scalable iot endpoint devices / M. Gautschi et al. *IEEE transactions on very large scale integration (VLSI) systems*. 2017. Vol. 25, no. 10. P. 2700–2713. URL: https://doi.org/10.1109/tvlsi.2017.2654506 (date of access: 30.05.2024).

47. Optimised AES with RISC-V Vector Extensions / M. N. Rizi et al. *2024 27th international symposium on design & diagnostics of electronic circuits & systems (DDECS)*, Kielce, Poland, 3–5 April 2024. 2024. URL: https://doi.org/10.1109/ddecs60919.2024.10508919 (date of access: 30.05.2024).

48. Phanindra K. 32-Bit MIPS RISC Processor. *International journal for research in applied science and engineering technology*. 2017. Vol. V, no. X. P. 1119–1123. URL: https://doi.org/10.22214/ijraset.2017.10162 (date of access: 30.05.2024).

49. Pitcher G. RISC-V powers iot apps processor. *New electronics*. 2018. Vol. 51, no. 4. P. 7. URL: https://doi.org/10.12968/s0047-9624(23)60141-5 (date of access: 30.05.2024).

50. Pitcher G. RISC-V powers iot apps processor. *New electronics*. 2018. Vol. 51, no. 4. P. 7. URL: https://doi.org/10.12968/s0047-9624(23)60141-5 (date of access: 30.05.2024).

51. Pitcher G. RISC-V powers iot apps processor. *New electronics*. 2018. Vol. 51, no. 4. P. 7. URL: https://doi.org/10.12968/s0047-9624(23)60141-5 (date of access: 30.05.2024).

52. Ragini D. K., Jaiswal N. Design of high-performance core micro-architecture based on 32- bit RISC-V instruction set architecture [ISA]. *International journal for research in applied science and engineering technology*. 2023. Vol. 11, no. 7. P. 1025–1033. URL: https://doi.org/10.22214/ijraset.2023.54791 (date of access: 30.05.2024).

53. Rahul R.Balwaik R. R. B. Open-Source 32-Bit RISC Soft-Core Processors. *IOSR journal of VLSI and Signal Processing*. 2013. Vol. 2, no. 4. P. 43–46. URL: https://doi.org/10.9790/4200-0244346 (date of access: 30.05.2024).

54. Rani A., Grover N. Novel design of 32-bit asynchronous (RISC) microprocessor & its implementation on FPGA. *International journal of information engineering and electronic business*. 2018. Vol. 10, no. 1. P. 39–47. URL: https://doi.org/10.5815/ijieeb.2018.01.06 (date of access: 30.05.2024).

55. REALISE-IoT: RISC-V based efficient and lightweight public-key system for iot applications / G. Mao et al. *IEEE internet of things journal*. 2023. P. 1. URL: https://doi.org/10.1109/jiot.2023.3296135 (date of access: 30.05.2024).

56. Realization of Authenticated One-Pass Key Establishment on RISC-V Micro-Controller for IoT Applications / T.-K. Dang et al. *Future internet*. 2024. Vol. 16, no. 5. P. 157. URL: https://doi.org/10.3390/fi16050157 (date of access: 30.05.2024).

57. Reis A. J. D. RISC-V assembly language. Independently Published, 2019.

58. RISC-V custom instructions of elementary functions for iot endpoint devices / Y. Chen et al. *IEEE transactions on computers*. 2023. P. 1–13. URL: https://doi.org/10.1109/tc.2023.3336174 (date of access: 30.05.2024).

59. RISC-Vlim, a RISC-V Framework for Logic-in-Memory Architectures / A. Coluccio et al. *Electronics*. 2022. Vol. 11, no. 19. P. 2990. URL: https://doi.org/10.3390/electronics11192990 (date of access: 30.05.2024).

60. Sá B. V. d. RISC-V lightweight virtualization extensions : master's thesis. 2021. URL: http://hdl.handle.net/1822/76556 (date of access: 30.05.2024).

61. Security and privacy for cloud-based iot: challenges / J. Zhou et al. *IEEE communications magazine*. 2017. Vol. 55, no. 1. P. 26–33. URL: https://doi.org/10.1109/mcom.2017.1600363cm (date of access: 20.05.2024).

62. Self-Powered AIoT Systems / N. N. Chiplunkar et al. New York : Apple Academic Press, 2024. URL: https://doi.org/10.1201/9781032684000 (date of access: 30.05.2024).

63. SERVAS! Secure enclaves via RISC-V authenticryption shield / S. Steinegger et al. *Computer security – ESORICS 2021*. Cham, 2021. P. 370–391. URL: https://doi.org/10.1007/978-3-030-88428-4_19 (date of access: 30.05.2024).

64. Singh S., Suresh Babu K. V. Adaptive smart farming system using Internet of Things (IoT) and artificial intelligence (AI) modeling. *AIoT technologies and applications for smart environments*. 2022. P. 113–126. URL: https://doi.org/10.1049/pbpc057e_ch7 (date of access: 30.05.2024).

65. Smart health care: from iot to aiot / S. Srithar et al. *Self-Powered AIoT Systems*. New York, 2024. P. 63–79. URL: https://doi.org/10.1201/9781032684000-3 (date of access: 30.05.2024).

66. Smart transportation: an overview of technologies and applications / D. Oladimeji et al. *Sensors*. 2023. Vol. 23, no. 8. P. 3880. URL: https://doi.org/10.3390/s23083880 (date of access: 20.05.2024).

67. Smith S. Controlling program flow. *RISC-V assembly language programming*. Berkeley, CA, 2024. P. 67–87. URL: https://doi.org/10.1007/979-8-8688-0137-2_4 (date of access: 30.05.2024).

68. Smith S. Controlling program flow. *RISC-V assembly language programming*. Berkeley, CA, 2024. P. 67–87. URL: https://doi.org/10.1007/979-8-8688-0137-2_4 (date of access: 30.05.2024).

69. Smith S. Floating-Point operations. *RISC-V assembly language programming*. Berkeley, CA, 2024. P. 233–258. URL: https://doi.org/10.1007/979-8-8688-0137-2_11 (date of access: 30.05.2024).

70. Smith S. Floating-Point operations. *RISC-V assembly language programming*. Berkeley, CA, 2024. P. 233–258. URL: https://doi.org/10.1007/979-8-8688-0137-2_11 (date of access: 30.05.2024).

71. Smith S. Getting started. *RISC-V assembly language programming*. Berkeley, CA, 2024. P. 1–24. URL: https://doi.org/10.1007/979-8-8688-0137-2_1 (date of access: 30.05.2024).

72. Smith S. Hacking code. *RISC-V assembly language programming*. Berkeley, CA, 2024. P. 293–314. URL: https://doi.org/10.1007/979-8-8688-0137-2_14 (date of access: 30.05.2024).

73. Smith S. Interacting with C and Python. *RISC-V assembly language programming*. Berkeley, CA, 2024. P. 189–212. URL: https://doi.org/10.1007/979-8-8688-0137-2_9 (date of access: 30.05.2024).

74. Smith S. Interacting with C and Python. *RISC-V assembly language programming*. Berkeley, CA, 2024. P. 189–212. URL: https://doi.org/10.1007/979-8-8688-0137-2_9 (date of access: 30.05.2024).

75. Smith S. Multiply and divide. *RISC-V assembly language programming*. Berkeley, CA, 2024. P. 213–231. URL: https://doi.org/10.1007/979-8-8688-0137-2_10 (date of access: 30.05.2024).

76. Smith S. Optimizing code. *RISC-V assembly language programming*. Berkeley, CA, 2024. P. 259–270. URL: https://doi.org/10.1007/979-8-8688-0137-2_12 (date of access: 30.05.2024).

77. Smith S. Optimizing code. *RISC-V assembly language programming*. Berkeley, CA, 2024. P. 259–270. URL: https://doi.org/10.1007/979-8-8688-0137-2_12 (date of access: 30.05.2024).

78. Smith S. Optimizing code. *RISC-V assembly language programming*. Berkeley, CA, 2024. P. 259–270. URL: https://doi.org/10.1007/979-8-8688-0137-2_12 (date of access: 30.05.2024).

79. Smith S. Programming GPIO Pins. *RISC-V assembly language programming*. Berkeley, CA, 2024. P. 163–187. URL: https://doi.org/10.1007/979-8-8688-0137-2_8 (date of access: 30.05.2024).

80. Smith S. Reading and understanding code. *RISC-V assembly language programming*. Berkeley, CA, 2024. P. 271–292. URL: https://doi.org/10.1007/979-8-8688-0137-2_13 (date of access: 30.05.2024).

81. Smith S. Thanks for the memories. *RISC-V assembly language programming*. Berkeley, CA, 2024. P. 89–115. URL: https://doi.org/10.1007/979-8-8688-0137-2_5 (date of access: 30.05.2024).

82. Smith S. Tooling up. *RISC-V assembly language programming*. Berkeley, CA, 2024. P. 45–66. URL: https://doi.org/10.1007/979-8-8688-0137-2_3 (date of access: 30.05.2024).

83. Suárez D., Almeida F., Blanco V. Comprehensive analysis of energy efficiency and performance of ARM and RISC-V SoCs. *The journal of supercomputing*. 2024. URL: https://doi.org/10.1007/s11227-024-05946-9 (date of access: 30.05.2024).

84. Taheri F., Bayat-Sarmadi S., Hadayeghparast S. RISC-HD: lightweight RISC-V processor for efficient hyperdimensional computing inference. *IEEE internet of things journal*. 2022. P. 1. URL: https://doi.org/10.1109/jiot.2022.3191717 (date of access: 30.05.2024).

85. Trusted hart for mobile RISC-V security / V. Ushakov et al. *2022 IEEE international conference on trust, security and privacy in computing and communications (trustcom)*, Wuhan, China, 9–11 December 2022. 2022. URL: https://doi.org/10.1109/trustcom56396.2022.00228 (date of access: 30.05.2024).

86. Vavro T. Periferie procesoru RISC-V : master's thesis. 2021. URL: http://www.nusl.cz/ntk/nusl-445553 (date of access: 30.05.2024).

87. Vavro T. Periferie procesoru RISC-V : master's thesis. 2021. URL: http://www.nusl.cz/ntk/nusl-445553 (date of access: 30.05.2024).

88. V-Curve25519: efficient implementation of curve25519 on RISC-V architecture / Q. Gao et al. *Information security and cryptology*. Singapore, 2024. P. 130–149. URL: https://doi.org/10.1007/978-981-97-0945-8_8 (date of access: 30.05.2024).

89. V-Curve25519: efficient implementation of curve25519 on RISC-V architecture / Q. Gao et al. *Information security and cryptology*. Singapore, 2024.

P. 130–149. URL: https://doi.org/10.1007/978-981-97-0945-8_8 (date of access: 30.05.2024).

90. Yilmaz Y., Tozlu Y. S., Ors B. Design and implementation of a 32-bit RISC-V core. *2021 13th international conference on electrical and electronics engineering (ELECO)*, Bursa, Turkey, 25–27 November 2021. 2021. URL: https://doi.org/10.23919/eleco54474.2021.9677678 (date of access: 30.05.2024).

91. Zhou H., Montenegro-Marin C. E., Hsu C.-H. Wearable IoT based cloud assisted framework for swimming persons in health monitoring system. *Current psychology*. 2020. URL: https://doi.org/10.1007/s12144-020-00822-0 (date of access: 20.05.2024).

ДОДАТОК А

(обов’язковий)

ЛІСТИНГ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ КОНТРОЛЮ ЦІЛІСНОСТІ ПОТОКУ ДАНИХ

Модуль 1 «Реалізація CFI на RISC-V R32 з 5 функціями та 12 викликами»

```
#include <riscv-v/instruction.h>

// Функція для обчислення факторіала числа
uint32_t factorial(uint32_t n) {
  if (n == 0) {
    return 1;
  } else {
    return n * factorial(n - 1);
  }
}

// Функція для обчислення суми чисел
uint32_t sum(uint32_t a, uint32_t b) {
  return a + b;
}

// Функція для обчислення добутку чисел
uint32_t product(uint32_t a, uint32_t b) {
  return a * b;
}

// Функція для перевірки парності числа
```

```
bool isEven(uint32_t n) {
  return (n % 2) == 0;
}

// Функція для перевірки простоти числа
bool isPrime(uint32_t n) {
  if (n <= 1) {
    return false;
  }
  for (uint32_t i = 2; i * i <= n; i++) {
    if (n % i == 0) {
      return false;
    }
  }
  return true;
}

int main() {
  // Отримання вхідних чисел від користувача
  uint32_t input1;
  uint32_t input2;
  read(input1);
  read(input2);

  // Обчислення факторіала першого вхідного числа
  uint32_t result1 = factorial(input1);

  // Обчислення суми та добутку вхідних чисел
  uint32_t sumResult = sum(input1, input2);
  uint32_t productResult = product(input1, input2);
```

```
  // Перевірка парності та простоти першого вхідного числа
  bool evenResult = isEven(input1);
  bool primeResult = isPrime(input1);

  // Виведення результатів
  write(result1);
  write(sumResult);
  write(productResult);
  write(evenResult);
  write(primeResult);

  return 0;
}
```

Модуль 2 «Адаптація тестової програми до обміну даними по UART з платою RISC-V32-bit MCUAB32VG1 з 5 функціями та 12 викликами»

```
#include <riscv-v/instruction.h>
#include <uart.h>

// Функція для обчислення факторіала числа
uint32_t factorial(uint32_t n) {
  if (n == 0) {
    return 1;
  } else {
    return n * factorial(n - 1);
  }
}

// Функція для обчислення суми чисел
```

```
uint32_t sum(uint32_t a, uint32_t b) {
  return a + b;
}

// Функція для обчислення добутку чисел
uint32_t product(uint32_t a, uint32_t b) {
  return a * b;
}

// Функція для перевірки парності числа
bool isEven(uint32_t n) {
  return (n % 2) == 0;
}

// Функція для перевірки простоти числа
bool isPrime(uint32_t n) {
  if (n <= 1) {
    return false;
  }
  for (uint32_t i = 2; i * i <= n; i++) {
    if (n % i == 0) {
      return false;
    }
  }
  return true;
}

int main() {
  // Ініціалізація UART
  uart_init();

  // Отримання вхідних чисел по UART
  uint32_t input1;
  uint32_t input2;
```

```
  uart_read_int(&input1);
  uart_read_int(&input2);

  // Обчислення факторіала першого вхідного числа
  uint32_t result1 = factorial(input1);

  // Обчислення суми та добутку вхідних чисел
  uint32_t sumResult = sum(input1, input2);
  uint32_t productResult = product(input1, input2);

  // Перевірка парності та простоти першого вхідного числа
  bool evenResult = isEven(input1);
  bool primeResult = isPrime(input1);

  // Виведення результатів по UART
  uart_write_int(result1);
  uart_write_int(sumResult);
  uart_write_int(productResult);
  uart_write_int(evenResult);
  uart_write_int(primeResult);

  return 0;
}……
```

ДОДАТОК Б

(обов’язковий)

ПУБЛІКАЦІЯ

Колодяжний Максим. Метод забезпечення кібербезпеки AIoT в умовах збільшення загроз кібератак. // Матеріали Весняної студентської конференції "Перспективні Мережеві і Комп’ютерні технології" (ПерСиК 2024). 25 квітня 2024. ХАІ.

УДК 004.05:004.71:004.49

МЕТОД ЗАБЕЗПЕЧЕННЯ КІБЕРБЕЗПЕКИ AIOT В УМОВАХ ЗБІЛЬШЕННЯ ЗАГРОЗ КІБЕРАТАК

Колодяжний М. Ю. студент гр. КІ2м-22-1

Науковий керівник: ст.викл. Іштван Є. О.

Національний аерокосмічний університет

ім. М. Є. Жуковського «ХАІ»

Повсякденне життя більшості українців не обходиться без спілкування в Інтернет із використанням різних месенджерів. Люди не лише переписуються, а й спілкуються голосом, надси-лають особисті фото, відео, геолокацію, робочі документи, ме-дичну інформацію тощо. Часто така інформація зберігається на іноземних серверах. Як вона використовується нам невідомо.

Метою даної роботи є порівняльний аналіз централізова-них та децентралізованих систем для спілкування з огляду на безпеку та конфіденційність.

Серед систем спілкування переважають централізовані рішення, такі як Telegram, Signal, WhatsApp, Viber та Skype. Вони засновані на роботі з централізованими серверами, які зберігають та обробляють повідомлення користувачів.

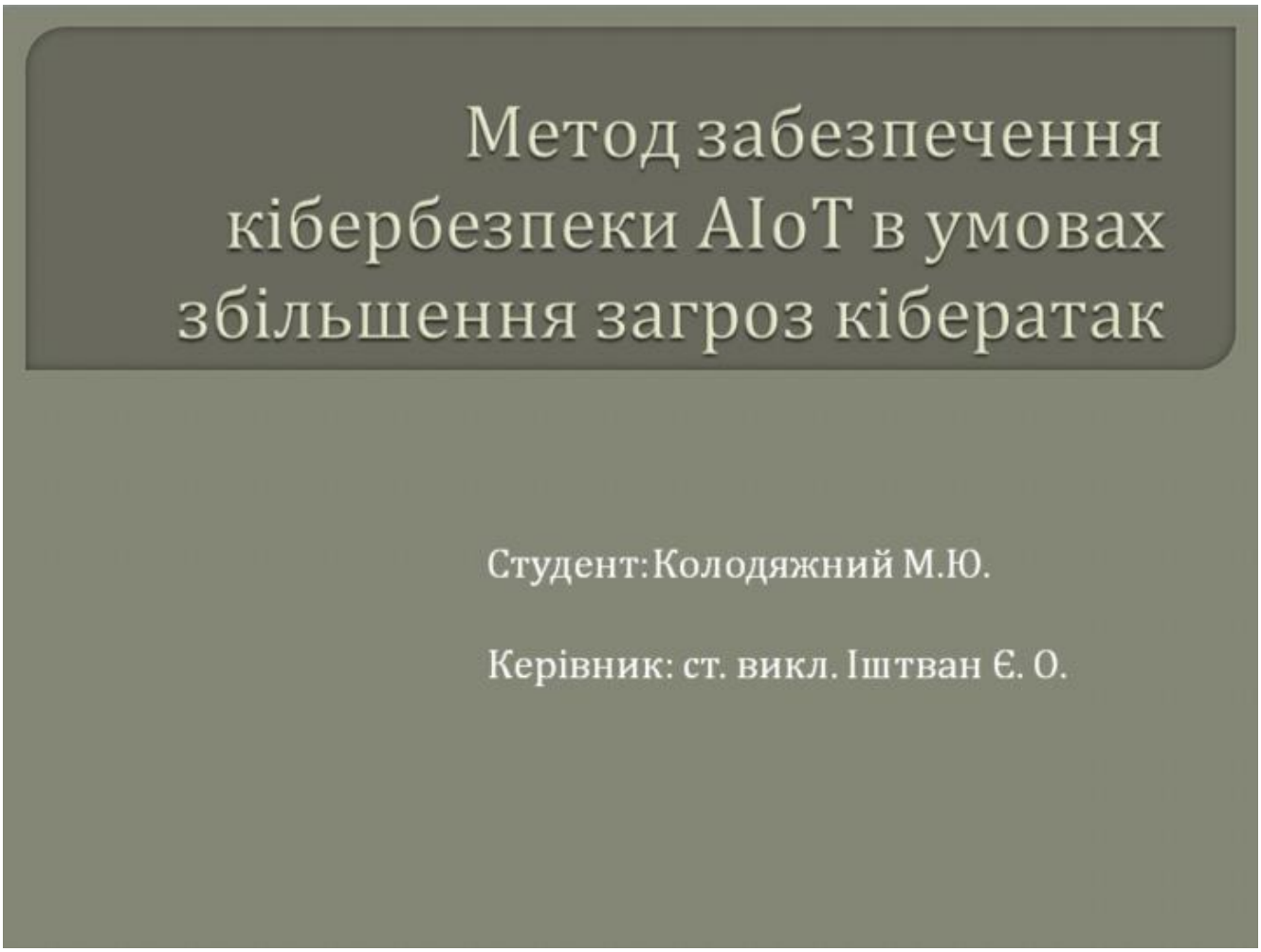
Децентралізовані однорангові P2P системи для спілкуван-ня, такі як Briar та Jami, використовують прямі з'єднання між пристроями. Прикладом федеративної децентралізованої систе-ми є Matrix. Такі системи дозволяють користувачам спілкувати-ся між різними серверами. SimpleX Chat поєднує переваги P2P та федеративних мереж і не надає користувачам ідентифікатори. Такі системи можуть вимагати більшого технічного досвіду та бути менш зручними у використанні, ніж централізовані.

Децентралізовані системи забезпечують вищий рівень конфіденційності та безпеки, ніж централізовані. Будь-яка сис-тема не завжди може гарантувати повну конфіденційність та безпеку, особливо якщо користувач порушує використовує сла-бкі паролі або відкриває доступ несанкціонованим особам.

Для безпечного спілкування раціонально використовувати спеціально розроблений месенджер із підконтрольними серве-рами. Це можна зробити написавши месенджер з нуля, або взяти за основу існуючі децентралізовані федеративні рішення.

ДОДАТОК В

ПРЕЗЕНТАЦІЙНІ МАТЕРІАЛИ



Мета роботи

Метою кваліфакаційної роботи магістра є збільшення стабільності **AIoT** системи з використанням **risk-v** архітектури на рівні кінцевих точок.

Об'єкт дослідження:
Програмні засоби контролю потоку даних для збільшення стабільності **AIoT** системи на рівні **end-point** з **RISC-V** архітектурою.

Предмет дослідження:
Метод оптимізації контролю потоку даних за допомогою алгоритму Галілея.

Наукова новизна

НН 1 — На **Risc-V 32bit** реалізовано метод для пошуку ланцюжків зворотного виклику на базі алгоритму Галілея

НН 2 — Покращено метод детекції порушення цілісності потоку даних **Risc-V 32bit**

НН 3 — Покращено метод детекції порушення цілісності потоку даних **Risc-V 32bit**

Практична значущість отриманих результатів

Висока розповсюдженість **IoT** пристроїв та набування розвиток **AIoT** індустрії створює виклики для стабільності роботи подібних систем. Особливо у медичній сфері, де пристрої використовуються для моніторінгу стану пацієнта чи приймають участь у його діагностиці.

Публікації

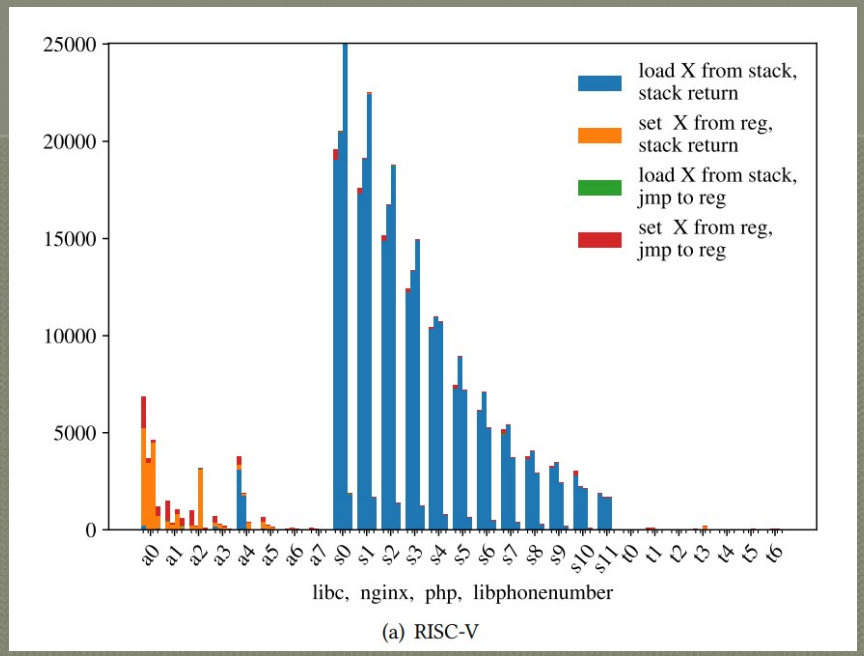
Результати представлені на конференції ПерСиК 2024 (25 квітня 2024) ХАІ.

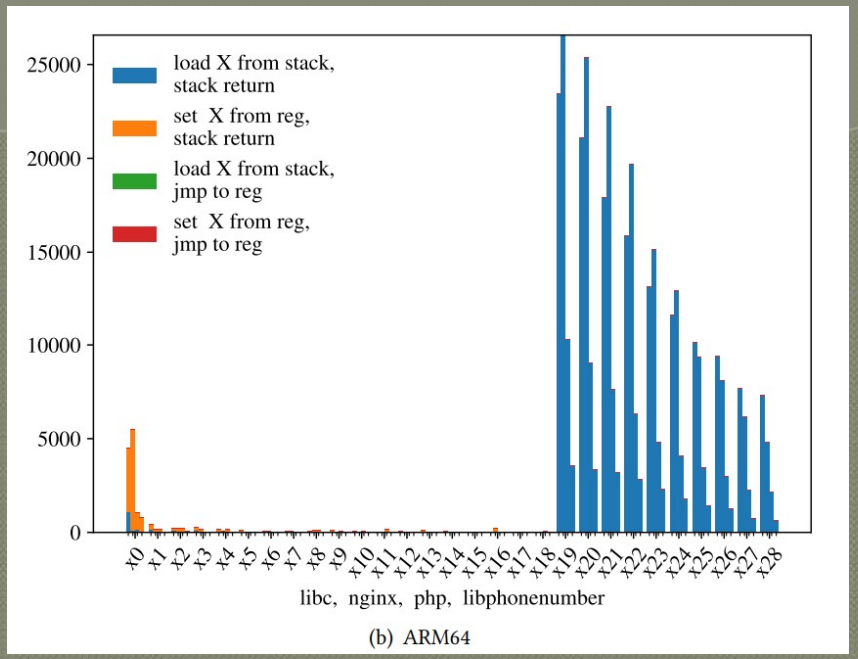
Матеріали якої опублікуються у збірці тез з ISBN.

Актуальність дослідження

- Зростання популярності **AIoT** систем (**Artificial Intelligence of Things**) у різних сферах життя, таких як розумний дім, промисловість, транспорт та охорона здоров'я. Ці системи збирають та обробляють великі обсяги даних, а кількість обчислювальних ресурсів низька, що робить їх вразливими до кібератак.

Актуальність дослідження



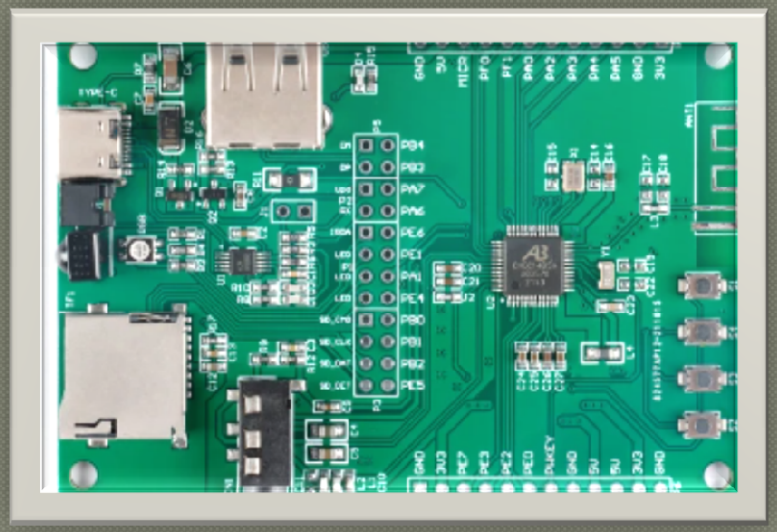


Cloosters, T., Paaßen, D., Wang, J., Draissi, O., Jauernig, P., Stapf, E., Davi, L., & Sadeghi, A.-R. (2022). RiscyROP: Automated return-oriented programming attacks on RISC-V and ARM64. In Proceedings of the 25th International Symposium on Research in Attacks, Intrusions and Defenses (RAID '22) (pp. 30-42). Association for Computing Machinery. https://doi.org/10.1145/3545948.3545997

Постановка задачі

- Інтегрувати у систему з малопотужним **RISC-V** контролером **CFI** (контроль потоку даних).
- Розробити метод низькорівневого моніторингу роботи **AIoT** системи згідно рекомендацій **IBM** та **CISCO** щодо підвищення стабільних **IoT** систем.
- Проведення експериментів з різним із використанням розробленої системи з контролем потоку даних.

Вибір платформи для рівня кінцевих точок (end points)



RISC-V 32-bit MCU AB32VG1

Основні етапи методу Галілео

Вставка міток: На початку кожної функції та перед інструкціями **jalr** та **eret** вставляються мітки. Ці мітки дають **CFI**-системі можливість відстежувати потік даних через програму.

Перевірка міток: **CFI**-система перевіряє мітки перед виконанням інструкцій **jalr** та **eret**. Якщо мітки не відповідають очікуваному порядку, виконання інструкції блокується. Це запобігає зловмисникам зміні потоку даних програми, щоб виконати шкідливий код.

Захист адрес повернення: **CFI**-система захищає адреси повернення функцій (**RA**) від зміни. Це робиться шляхом зберігання **RA** в спеціальному регістрі, до якого зловмисники не мають доступу.

Відновлення стану: **CFI**-система відновлює стан програми після виконання інструкцій **jalr** та **eret**. Це гарантує, що програма повертається до очікуваного стану після виклику функції.

Експериментальна частина

На прикладі програми CFI на RISC-V r32 з 5 функціями та 12 викликами для різних вхідних даних проведено розрахунки:

-розміру стека і глибини
-розміру таблиці GAM (Global Address Mapping)
-розміру таблиці LAM (Local Address Mapping)
-розміру таблиці TT (Thread Tracking)

Експериментальна частина

На прикладі програми CFI на RISC-V r32 з 5 функціями та 12 викликами для різних вхідних даних проведено розрахунки:

-розміру стека і глибини
-розміру таблиці GAM (Global Address Mapping)
-розміру таблиці LAM (Local Address Mapping)
-розміру таблиці TT (Thread Tracking)

Експериментальна частина

На прикладі програми CFI на RISC-V r32 з 5 функціями та 12 викликами для різних вхідних даних проведено розрахунки:

-розміру стека і глибини
-розміру таблиці GAM (Global Address Mapping)
-розміру таблиці LAM (Local Address Mapping)
-розміру таблиці TT (Thread Tracking)

Експериментальна частина

Показник	CFI статичного аналізу	CFI динамічного моніторингу	CFI апаратних засобів
Відсоток успішного виявлення атак	95% - 99%	85% - 95%	99% - 100%
Відсоток помилкових спрацьовувань	0.1% - 1%	1% - 5%	0.01% - 0.1%
Час виконання	+20% - +50%	+10% - +30%	+5% - +15%
Використання CPU	+15% - +40%	+10% - +25%	+5% - +10%
Використання пам'яті	+10% - +30%	+5% - +20%	+0% - +5%
Частота збоїв	0.01% - 0.1%	0.1% - 0.5%	0.001% - 0.01%
Частота помилок	0.1% - 1%	1% - 5%	0.01% - 0.1%
Відсоток успішних атак	5% - 15%	15% - 25%	1% - 5%
Час до виявлення атаки	10 - 50 мс	50 - 200 мс	1 - 10 мс
Складність реалізації	Висока	Середня	Низька
Складність налаштування	Середня	Висока	Низька
Сумісність з іншими інструментами	Висока	Середня	Низька

Висновки

У першому розділі було проведено огляд літератури, описано штучний інтелект речей (AIoT) як динамічно розвиваючуся область, що поєднує в собі штучний інтелект (AI) з Інтернетом речей (IoT)

У другому розділі описується метод аналізу програм, написаних в ROP, який використовує оцінку доцільності використання контролю потоку даних за алгоритмом Галілея. Цей метод може допомогти визначити, чи буде використання контролю потоку даних ефективним для даної програми.

Висновки

У третьому розділі описані результати дослідження та тестування програмного забезпечення CFI для RISC-V. Автори роблять висновок, що метод є ефективним та економічно вигідним, і рекомендують його до подальшого впровадження. Також визначено обмеження методу.

У четвертому розділі проведено експерименти для оцінки ефективності та надійності нових методів CFI та застосовано статичний аналіз коду програмного забезпечення виявлення кібер-загроз.

UNICHECK
by Turnitin

Ім'я користувача:
Кафедра КІ

ID перевірки:
1016298169

Дата перевірки:
30.05.2024 10:42:43 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
30.05.2024 17:30:21 EEST

ID користувача:
100005591

Назва документа: **Колодяжний_Метод забезпечення кібербезпеки AIoT в умовах збільшення загроз кібератак**

Кількість сторінок: **77** Кількість слів: **14874** Кількість символів: **111177** Розмір файлу: **1,002.24 KB** ID файлу: **1016093333**

1.66%
Схожість

Найбільша схожість: **0.62% з джерелом з Бібліотеки** (ID файлу: **1014495205**)

1.42% Джерела з Інтернету	69	Сторінка 79
0.98% Джерела з Бібліотеки	80	Сторінка 80

0% Цитат

Цитати	1	Сторінка 81
Посилання	1	Сторінка 81

0%
Вилучень

Немає вилучених джерел

Модифікації

Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.

Замінені символи	13

Anti-Plagiarism v-15.257

Максимальне співпадіння з одним документом 0.0%

Словники перевірки: en_US, ru_RU, ua_UA. **Помилок в документах: 13%**

	Документ		Сумарний збіг по Базі Даних	
ID: 127786 Назва: МКР Метод забезпечення кібербезпеки AIoT в умовах збільшення загроз кібератак Додано в БД: 2024-05-30 Автора: Колодяжний М.Ю. Керівники: Іштван Є.О. Консультанти: Опоненти:	Символи	Лексеми	Символи	Лексеми
	92802	821	1149 (1%)	23 (3%)

Джерело плагіату

ID	Опис	Наявність плагіату в документі	
		Символи	Лексеми

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА ДИПЛОМНУ РОБОТУ

Дипломник: Колодяжний Максим Юрійович

Тема: Метод забезпечення кібербезпекиAIoT в умовах збільшення загроз кібератак

Спеціальність**:** 123 «Комп'ютерна інженерія»

Обсяг дипломної роботи:

Кількість листів креслень _–__; кількість сторінок записки _72__

1. Короткий зміст роботи та прийнятих рішень _У роботі запропоновано архітектуру AIoT системи з системою виявлення нестабільностей _____

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 першому розділі проведено огляд технологій Інтернету речей та архітектурних особливостей AIOT систем. Досліджено відомі рішення та засоби в цій сфері. У другому розділі Забезпечення стійкості IoT та AIoT. У третьому розділі запропоновано метод Модель програмного контролю цілісності даних в системі AIoT. У четвертому розділі запропоновано інформаційну технологію виявлення кібер-загроз._____

4. Позитивні сторони роботи: ___Запропонована система вичвлення нестабільності AIoT системи дозволяє проводити моніторинг, досліджувати вразливості та використовується у підвищенні надійності системи.

5. Негативні сторони роботи: ___В роботі присутні певні логічні помилки щодо опису використання алгоритму Галілея у CFI системі_____

6. Оцінка графічного оформлення та пояснювальної записки роботи: __–____

7. Відгук про роботу в цілому: В загальному робота виконана на задовільному рівні.

8. Інші зауваження: –

9. Оцінка дипломної роботи:

Розглянувши позитивні та негативні сторони представленої дипломної роботи вважаю, що робота заслуговує оцінки «задовільно» 3.00 (E)

Рецензент (прізвище, ім'я, по батькові, посада, місце роботи) ____д.т.н.. професор, Мартинюк В.В., завідувач кафедри АКІТтаР

"__20 травня__"_____2023р.

Завідувачу кафедри КІІС
д-р.техн.наук, проф. Говорущенко Т. О.

Колодяжний М. Ю.
ПІБ здобувача вищої освіти

ФІТ, 2 курсу, групи КІ2м-22-1

ЗАЯВА

З правилами чинного Положення «Про дотримання академічної доброчесності в Хмельницькому національному університеті» від 26.09.2020 (зі змінами від 26.11.2020), згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування заходів дисциплінарної та академічної відповідальності, ознайомлений (а). Про використання програмно-технічних засобів для перевірки кваліфікаційних робіт здобувачів вищої освіти на плагіатоповіщений (а) та надаю свою згоду на обробку та збереження університетом моєї роботи в інституційному репозитарії університету.

Також надаю університету право на передачу моєї роботи для обробки та збереження в базах даних програмно-технічних засобів (Unicheck та Anti-Plagiarism) та використання роботи для виявлення плагіату в інших роботах, які перевіряються програмно-технічними засобами та користувачами, що мають доступ до цих програмно-технічних засобів, виключно в обмежених цілях для виявлення плагіату в текстах робіт.

Робота для перевірки університетом надається в друкованому та електронному варіанті. Електронна версія моєї роботи збігається (ідентична) з друкованою.

18.05.24р
дата

підпис

РІШЕННЯ ЕКСПЕРНОЇ КОМІСІЇ

КАФЕДРИ КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ ТА ІНФОМАЦІЙНИХ СИСТЕМ

ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Підтверджуємо ознайомлення з результатом звіту подібності щодо роботи, генерованого системою виявлення текстових збігів/ідентичності/схожості:

Назва: Метод забезпечення кібербезпеки AIoT в умовах збільшення загроз кібератак

Автор: Колодяжний Максим Юрійович

Спеціальність: 123 – Компютерна інженерія

Освітня програма: освітньо-наукова

Науковий керівник: Іштван Є.О. ст.викл.

Після аналізу звіту подібності зроблено такий висновок:

№	Висновок	Позначка про відповідність
1	Запозичення, виявлені в роботі, є законними і не є плагіатом. Робота приймається до захисту.	відповідає
2	Виявлені запозичення не є плагіатом, розміщені в розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи Робота приймається до захисту, але має бути відкоригована. Відкоригований варіант має бути поданий на кафедру за 2 дні до захисту, разом із заявою щодо самостійності виконання письмової роботи та ідентичності друкованої та електронної версії роботи	
3	Виявлені запозичення не є плагіатом, але частково розміщені в розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. В зв'язку з цим мета роботи та поставлені завдання не були досягнені. Робота може бути допущена до захисту (наступного року) після того як буде відкоригована та допрацьована і успішно пройде повторну перевірку на академічний плагіат.	
4	Робота містить навмисні текстові спотворення, передбачувані спроби укриття запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту.	

Підтвердження:

Запозичення, виявлені в роботі, є законними і не є плагіатом, оскільки:

1) окремі виявлені збіги відносяться до шаблону документу або є загальновживаними фразами або виразами, про що свідчить посилання системи на збіг з 69-80 джерелами на один фрагмент речення;
2) всі зафіксовані системою ознаки модифікації тексту відносяться до комбінування латинських символів зі україномовними скороченнями, індексів в формулах, що не є модифікацією тексту.

Сумарний обсяг всіх запозичень, визначений системою виявлення збігів/ ідентичності/схожості, складає 1.66% і адресується до 149 першоджерел, що, з урахуванням наведених обґрунтувань, відповідає характеру наукового дослідження і свідчить на користь кваліфікаційної роботи.

Керівник роботи _____ Є.О. Іштван

Гарант ОП _____ О. С. Савенко

Завідувач кафедри КІПСч _____ Т. О. Говорущенко