

Міністерство освіти і науки України
Державний університет «Житомирська політехніка»
Факультет інформаційно-комп'ютерних технологій
Кафедра комп'ютерної інженерії та кібербезпеки

ПОЯСНЮВАЛЬНА ЗАПИСКА

до випускної кваліфікаційної роботи магістра
на тему: «Методи та технології забезпечення кібербезпеки у системі моніторингу IoT-мережі на базі ESP32»

Тема роботи: «Методи та технології забезпечення кібербезпеки у системі моніторингу IoT-мережі на базі ESP32»

затверджена Наказом університету від 6 жовтня 2025 р. №566/с

Термін здачі студентом закінченої роботи 16 грудня 2025 р.

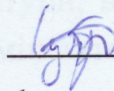
Вихідні дані роботи:

1. Система моніторингу IoT-мережі на базі мікроконтролерної платформи ESP32.

2. Методи та технології забезпечення кібербезпеки у системі моніторингу з урахуванням обмежень об'єкта дослідження.

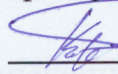
3. Показники сенсори температури та вологості, датчик руху (PIR), результати тестування механізму безпеки мережевих атак.

Виконала: студент 2-го курсу групи КБм-24-1 спеціальності 125 «Кібербезпека та захист інформації»



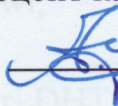
Катерина СУПРУНЕНКО

Керівник: Професор кафедри КІ та КБ, д.т.н., с.н.с.



Сергій КОВБАСЮК

Рецензент: Доцент кафедри КІ та КБ, к.т.н.



Олексій ШЕЛУХА

Житомир – 2025

Державний університет «Житомирська політехніка»
Факультет інформаційно-комп'ютерних технологій
Кафедра комп'ютерної інженерії та кібербезпеки
Спеціальність 125 «Кібербезпека та захист інформації»
Освітня програма «Кібербезпека»

ЗАТВЕРДЖУЮ

Завідувач

кафедри комп'ютерної
інженерії та кібербезпеки

 Андрій ЄФІМЕНКО

3 листопада 2025 р.

ЗАВДАННЯ

на випускню кваліфікаційну роботу магістра

Студент: Супруненко Катерина Олександрівна

Тема роботи: «Методи та технології забезпечення кібербезпеки у системі моніторингу IoT-мережі на базі ESP32»

затверджена Наказом університету від 6 жовтня 2025 р. №566/с

Термін здачі студентом закінченої роботи 16 грудня 2025 р.

Вихідні дані роботи:

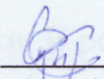
1. Система моніторингу IoT-мережі на базі мікроконтролерної платформи ESP32.
2. Методи та засоби забезпечення кібербезпеки IoT-систем моніторингу з урахуванням обмежених обчислювальних ресурсів.
3. Показники сенсорів температури, вологості та руху, журнали подій системи, результати тестування механізмів автентифікації, авторизації та захисту від мережових атак.

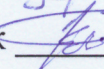
Обладнання та програмне забезпечення:

- середовище моделювання Wokwi;
- мікроконтролер ESP32;
- сенсор температури та вологості DHT, датчик руху (PIR);
- мова програмування C/C++ (Arduino framework);

Консультанти з випускної кваліфікаційної роботи із зазначенням розділів, що їх стосуються

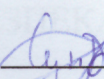
Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
1			
2			
3			

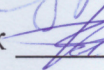
Студент  Катерина СУПРУНЕНКО

Керівник  Сергій КОВБАСЮК

Календарний план

№ з/п	Назви етапів випускної кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1	Постановка завдання	06.10.2025	Виконано
2	Аналіз проблем безпеки в IoT-мережах	13.11.25	Виконано
3	Формулювання технічного завдання	27.11.25	Виконано
4	Опрацювання літературних джерел	10.12.25	Виконано
5	Проектування	12.12.25	Виконано
6	Налагодження захисту	14.12.25	Виконано
7	Перевірка працездатності	17.12.25	Виконано
8	Написання пояснювальної записки	18.12.25	Виконано
9	Захист		

Студент  Катерина СУПРУНЕНКО

Керівник  Сергій КОВБАСЮК

АНОТАЦІЯ

Пояснювальна записка до кваліфікаційної роботи на тему «Методи та технології забезпечення кібербезпеки у системі моніторингу IoT-мережі на базі ESP32» складається з вступу, трьох розділів, висновків, списку використаної літератури та додатків. Текстова частина викладена на 64 сторінках друкованого тексту. Пояснювальна записка містить 3 додатки на 10 сторінках. Список використаних джерел містить 27 найменувань і займає 2 сторінки. В роботі наведено 21 рисунок. Загальний обсяг роботи – 74 сторінки.

Кваліфікаційна робота присвячена дослідженню та реалізації методів і технологій забезпечення кібербезпеки у системі моніторингу IoT-мережі з використанням мікроконтролерної платформи ESP32. Основною метою роботи є розробка захищеної системи моніторингу, яка забезпечує безпечний збір, передачу та обробку даних з IoT-пристроїв в умовах обмежених обчислювальних ресурсів.

У рамках роботи досліджено можливості використання контролера ESP32 для побудови систем моніторингу IoT-мереж з урахуванням його технічних характеристик, зокрема наявності вбудованих модулів бездротового зв'язку та підтримки мережевих протоколів. Реалізовано апаратну частину системи, що включає сенсори температури, вологості та руху, а також програмне забезпечення для збору, обробки та передачі даних.

Особливу увагу в роботі приділено питанням кібербезпеки. Реалізовано механізми автентифікації та авторизації користувачів, криптографічного захисту даних, а також засоби протидії типовим мережевим атакам, зокрема DoS/DDoS, brute force та атакам типу “людина посередині”. Проведене тестування підтвердило ефективність застосованих методів і стабільну роботу системи моніторингу.

Ключові слова: IoT, ESP32, КІБЕРБЕЗПЕКА, СИСТЕМА МОНІТОРИНГУ, ЗАХИСТ ВІД АТАК.

ANNOTATION

The explanatory note to the diploma project entitled «Methods and Technologies for Ensuring Cybersecurity in an IoT Network Monitoring System Based on ESP32» consists of an introduction, three chapters, conclusions, a list of references, and appendices. The main text is presented on 64 pages of printed text. The explanatory note contains 3 appendices totaling 10 pages. The list of references includes 27 sources and occupies 2 pages. The work contains 21 figures. The total volume of the work is 74 pages.

The qualification work is devoted to the study and implementation of methods and technologies for ensuring cybersecurity in an IoT network monitoring system using the ESP32 microcontroller platform. The main objective of the work is to develop a secure monitoring system that ensures safe data collection, transmission, and processing from IoT devices under conditions of limited computational resources.

Within the scope of the work, the capabilities of the ESP32 controller for building IoT network monitoring systems were investigated, taking into account its technical characteristics, including the availability of integrated wireless communication modules and support for network protocols. The hardware part of the system was implemented, including temperature, humidity, and motion sensors, as well as software for data collection, processing, and transmission.

Special attention is paid to cybersecurity issues in the work. User authentication and authorization mechanisms, cryptographic data protection, and countermeasures against common network attacks, including DoS/DDoS, brute force, and man-in-the-middle attacks, were implemented. The conducted testing confirmed the effectiveness of the applied methods and the stable operation of the monitoring system.

Keywords: IoT, ESP32, CYBERSECURITY, MONITORING SYSTEM, ATTACK PROTECTION.

ЗМІСТ

ВСТУП.....	8
РОЗДІЛ 1. АНАЛІЗ ЗАГРОЗ ТА СУЧАСНИХ ПІДХОДІВ ДО ЗАБЕЗПЕЧЕННЯ КІБЕРБЕЗПЕКИ ІoT-СИСТЕМ	10
1.1. Особливості функціонування ІoT-мереж у контексті кібербезпеки	10
1.2. Аналіз типових загроз і вразливостей у ІoT-мережах	18
1.3. Огляд сучасних методів захисту ІoT-пристроїв та мереж.....	26
1.4. Постановка задачі дослідження та вимоги до системи кібербезпеки	34
Висновок до першого розділу	35
РОЗДІЛ 2. ПРОЕКТУВАННЯ ТА ЗАБЕЗПЕЧЕННЯ КІБЕРБЕЗПЕКИ СИСТЕМИ МОНІТОРИНГУ ІoT-МЕРЕЖІ НА БАЗІ ESP32.....	36
2.1. Функціональні та нефункціональні вимоги до системи моніторингу	36
2.2. Архітектура та реалізація захищеної системи моніторингу ІoT-мережі на базі ESP32	38
2.3. Забезпечення безпеки доступу та захист даних	43
Висновки до другого розділу.....	48
РОЗДІЛ 3. РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ СИСТЕМИ МОНІТОРИНГУ ІoT ТА АНАЛІЗ ОТРИМАНИХ РЕЗУЛЬТАТІВ.....	49
3.1. Реалізація системи моніторингу ІoT мережі.....	49
3.2. Тестування системи моніторингу ІoT мережі.....	54
3.3. Перспективні технології кіберзахисту ІoT-систем.....	59
Висновки до третього розділу	61
ВИСНОВОК.....	62
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	63
ДОДАТКИ.....	65

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ABAC – Модель керування доступом на основі атрибутів

ADC – Аналого-цифровий перетворювач

BLE – Бездротовий енергозберігаючий інтерфейс

CPU – Центральний процесор

DAC – Цифро-аналоговий перетворювач

DoS – Кібератака, що перевантажує систему, щоб зробити її недоступною для законних користувачів

EEPROM – Електрично стираєма програмована читаєма пам'ять

ESP32 – Контролер ESP32

FTP – Протокол передачі файлів

GPIO – Загальні входи/виходи

I2C – Інтерфейс міжінтегрального зв'язку

IDE – Середовище розробки

IoT – Інтернет речей

IT – Інформаційні технології

MCU – Мікроконтролер

MiTM – Кібератака, коли зломисник таємно втручається в комунікацію між двома сторонами, перехоплюючи та часто змінюючи дані

OT – Операційні технології

ПЗ – Програмне забезпечення

PWM – Широтно-імпульсна модуляція

RAM – Оперативна пам'ять

RBAC – Модель керування доступом на основі ролей

SPI – Послідовний периферійний інтерфейс

UART – Універсальний асинхронний приймач-передавач

Wi-Fi – Бездротова локальна мережа

ВСТУП

Стрімкий розвиток технологій IoT зумовив широке впровадження мереж з великою кількістю вбудованих пристроїв у різних галузях, зокрема в промисловості, енергетиці, “розумних” будівлях, транспорті та системах моніторингу. Водночас обмежені апаратні ресурси IoT-пристроїв, зокрема мікроконтролерних платформ, створюють додаткові виклики щодо забезпечення належного рівня кібербезпеки. Багато IoT-рішень залишаються вразливими до несанкціонованого доступу, перехоплення даних, підміни інформації та атак типу відмова в обслуговуванні.

Особливої актуальності набуває проблема захисту IoT-мереж на базі платформи ESP32, яка широко використовується завдяки низькій вартості, енергоефективності та вбудованим бездротовим інтерфейсам. Водночас використання таких платформ потребує впровадження комплексних методів і технологій кібербезпеки, здатних забезпечити конфіденційність, цілісність і доступність даних без суттєвого зниження продуктивності системи. У зв’язку з цим розробка та дослідження захищеної системи моніторингу IoT-мережі з використанням сучасних методів кіберзахисту є актуальним науково-практичним завданням.

Метою кваліфікаційної роботи є підвищення рівня кібербезпеки системи моніторингу IoT-мережі на базі платформи ESP32 шляхом розробки та впровадження методів і технологій захисту даних, пристроїв і каналів зв’язку.

Об’єктом дослідження є процеси функціонування та моніторингу IoT-мереж із використанням вбудованих мікроконтролерних пристроїв.

Предметом дослідження є методи, засоби та технології забезпечення кібербезпеки в системі моніторингу IoT-мережі на базі мікроконтролерної платформи ESP32, зокрема механізми автентифікації, шифрування даних, захисту каналів зв’язку та контролю доступу.

Визначена мета обумовлює наступні завдання:

- проаналізувати сучасні архітектури IoT-мереж та типові загрози їх інформаційній безпеці;
- дослідити існуючі методи та технології забезпечення кібербезпеки в IoT-системах;

- розробити архітектуру захищеної системи моніторингу IoT-мережі на базі ESP32;
- розробити механізми захисту даних і каналів зв'язку в системі;
- провести дослідження ефективності запропонованих методів кібербезпеки та оцінити їх вплив на продуктивність системи.

У процесі виконання кваліфікаційної роботи було розроблено проект захищеної системи моніторингу IoT-мережі на базі мікроконтролерної платформи ESP32, яка забезпечує безпечний збір, передачу та обробку даних. Проект передбачає використання сучасних криптографічних методів, механізмів автентифікації пристроїв і захисту каналів зв'язку, що дозволяє підвищити загальний рівень кібербезпеки IoT-системи.

РОЗДІЛ 1. АНАЛІЗ ЗАГРОЗ ТА СУЧАСНИХ ПІДХОДІВ ДО ЗАБЕЗПЕЧЕННЯ КІБЕРБЕЗПЕКИ ІоТ-СИСТЕМ

1.1. Особливості функціонування ІоТ-мереж у контексті кібербезпеки

Пристрої ІоТ є результатом поєднання сфер ІТ та ОТ. Багато ІоТ-пристроїв виникли внаслідок конвергенції хмарних обчислень, мобільних обчислень, вбудованих систем, великих даних, доступного за ціною апаратного забезпечення та інших технологічних досягнень. ІоТ-пристрої можуть забезпечувати обчислювальні можливості, зберігання даних і мережеве з'єднання для обладнання, яке раніше не мало таких функцій, відкриваючи нові рівні ефективності та технологічні можливості, зокрема віддалений доступ для моніторингу, налаштування та усунення несправностей. Крім того, ІоТ надає можливість аналізувати дані про фізичний світ і використовувати результати такого аналізу для більш обґрунтованого прийняття рішень, зміни фізичного середовища та прогнозування майбутніх подій [1].

Пристрої Інтернету речей зазвичай стикаються з тими ж типами ризиків для кібербезпеки та конфіденційності, що й звичайні ІТ-пристрої, хоча поширеність та серйозність таких ризиків часто відрізняються. Наприклад, ризики безпеки даних майже завжди є суттєвою проблемою для звичайних ІТ-пристроїв, але для деяких пристроїв Інтернету речей ризиків безпеки даних може не бути, оскільки вони не містять жодних даних, які потребують захисту [1].

Взаємодія з фізичним світом, яку забезпечують ІоТ-пристрої, може по різному впливати на ризики кібербезпеки та конфіденційності. Розглянемо декілька прикладів.

Дані ІоТ-сенсорів, що відображають вимірювання фізичного світу, завжди містять певні невизначеності. Ефективне управління даними ІоТ-сенсорів, зокрема розуміння рівнів невизначеності, є необхідним для оцінювання якості та змісту даних, щоб організація могла приймати обґрунтовані рішення щодо їх використання та уникати створення нових ризиків. Без цього рівні похибок можуть залишатися невідомими для різних контекстів використання ІоТ-пристрою. Ефективне управління даними сенсорів також є важливим для протидії фізичним атакам на сенсорні

технології, наприклад атакам, здійсненим за допомогою бездротових сигналів, які можуть змусити сенсори генерувати хибні результати [1].

Повсюдне розміщення IoT-сенсорів у публічному та приватному середовищах може сприяти агрегації та аналізу величезних обсягів даних про окремих осіб. Такі дії можуть використовуватися для впливу на поведінку або прийняття рішень людьми способами, яких вони не усвідомлюють, або призводити до розкриття інформації, яку особи не бажали розкривати, зокрема до повторної ідентифікації раніше знеособлених персональних даних. Крім того, ці процеси можуть виходити за межі початково запланованої сфери функціонування IoT-пристрою [1].

IoT-пристрої з виконавчими механізмами (актуаторами) мають здатність змінювати фізичні системи й таким чином впливати на фізичний світ. Потенційний вплив таких змін має бути чітко усвідомлений і врахований з позицій кібербезпеки та конфіденційності. У найгіршому сценарії компрометація може дозволити зловмиснику використати IoT-пристрій для загрози безпеці людей, пошкодження або знищення обладнання та об'єктів інфраструктури чи спричинення серйозних операційних збоїв. Питання конфіденційності та пов'язані з ними питання громадянських свобод можуть виникати і внаслідок санкціонованих змін фізичних систем, які впливають на фізичну автономію або поведінку людей в особистих чи публічних просторах. Наприклад, системи фізичного контролю доступу, такі як автоматизовані дверні замки, можуть обмежувати доступ до приміщень або будівель із людьми всередині, а системи керування середовищем, зокрема освітленням чи температурою, можуть використовуватися для впливу на переміщення людей у будівлях [1].

Мережеві інтерфейси IoT часто забезпечують віддалений доступ до фізичних систем, які раніше могли бути доступні лише локально. Виробники, постачальники та інші треті сторони можуть використовувати віддалений доступ до IoT-пристроїв для управління, моніторингу, обслуговування та усунення несправностей. Це може значно підвищувати ризик компрометації фізичних систем, до яких надається доступ через IoT-пристрої. Крім того, такі децентралізовані функції обробки даних мо-

жуть посилювати деякі ризики для конфіденційності, ускладнюючи для людей розуміння принципів роботи IoT-системи, що, своєю чергою, заважає їм приймати обґрунтовані рішення щодо обробки їхньої інформації та взаємодії з IoT-системою [1].

Практики організації щодо зниження ризиків у сфері кібербезпеки та конфіденційності можуть потребувати суттєвих змін через велику кількість IoT-пристроїв і значну різноманітність їх типів. Традиційні IT-пристрої в межах одного типу, як правило, мають подібні можливості. Наприклад, більшість ноутбуків характеризуються схожими можливостями зберігання та обробки даних, інтерфейсами взаємодії з користувачем і мережевими інтерфейсами, а також допоміжними функціями, такими як централізоване управління. Це дозволяє організаціям визначати підходи до управління ризиками для кожного з десятків типів традиційних IT-пристроїв із певними налаштуваннями для окремих пристроїв і моделей, і зазвичай організації звикли до такого рівня зусиль [1].

Натомість більшість організацій можуть мати значно більше типів IoT-пристроїв, ніж традиційних IT-пристроїв, через вузькоспеціалізований характер більшості IoT-рішень. Організації може знадобитися визначати підходи до управління ризиками для сотень або навіть тисяч типів IoT-пристроїв (Рис. 1.1).



Рисунок 1.1 – Типів IoT-пристроїв [2]

Можливості таких пристроїв можуть істотно відрізнятись, так один тип може не мати засобів зберігання даних і централізованого управління, тоді як інший

може містити численні сенсори та актуатори, використовувати локальні й віддалені можливості зберігання та обробки даних і одночасно бути підключеним до кількох внутрішніх і зовнішніх мереж. Така варіативність можливостей зумовлює аналогічну різноманітність ризиків у сфері кібербезпеки та конфіденційності для кожного типу IoT-пристроїв, а також відмінності в способах зниження цих ризиків.

Крім того, організації може знадобитися керувати ризиками не лише за типами пристроїв, а й з урахуванням сценаріїв їх використання. Спосіб використання пристрою може свідчити про те, що одна ціль безпеки, наприклад цілісність, є важливішою за іншу, наприклад конфіденційність, що, у свою чергу, може вимагати застосування різних механізмів зниження ризиків. Аналогічно, пристрій може використовуватися таким чином, що деякі його можливості не є необхідними й можуть бути вимкнені, що дозволить зменшити рівень ризику, пов'язаного з цим пристроєм.

З огляду на постійне розширення поверхні атаки IoT, безпека IoT також має бути комплексною та охоплювати широкий спектр методів захисту (рис.1.2).



Рисунок 1.2 – Основні методи забезпечення кібербезпеки IoT

Ці методи повинні бути здатними захищати IoT-пристрої та мережі від різноманітних загроз - як нових, так і вже відомих. До ключових методів забезпечення безпеки IoT можна віднести:

- Безпека API. Інтерфейс прикладного програмування (API) є програмним посередником, який дозволяє додаткам безпечно обмінюватися інформацією шляхом перевірки та контролю передавання даних. Оскільки IoT-пристрої обмінюються інформацією з серверними системами, організації можуть застосовувати механізми безпеки API, щоб гарантувати, що лише авторизовані пристрої та користувачі взаємодіють з API [3].

- Безпека шлюзів. шлюзи безпеки IoT створюють додатковий рівень захисту між пристроями та мережами. Такі шлюзи мають можливість застосовувати захисні механізми (наприклад, міжмережеві екрани), щоб запобігти переміщенню кіберзлочинців із скомпрометованих пристроїв до інших підключених систем [3].

- Контроль доступу до мережі. Контроль доступу до мережі формує інвентар усіх IoT-пристроїв, підключених до мережі, надаючи організаціям надійну основу для моніторингу пристроїв і реагування на аномалії [3].

- Мережева безпека. Оскільки метою багатьох IoT-атак є отримання доступу до приватної мережі, мережева безпека є критично важливою для будь-якої організації, що використовує IoT. Вона охоплює як цифрові, так і фізичні компоненти та включає антивірусний захист, безпеку портів, виявлення та запобігання вторгненням, управління IP-адресами, тощо [3].

- Управління оновленнями. Однією з переваг підключених пристроїв є можливість усунення їхніх вразливостей. Регулярне встановлення оновлень і патчів для IoT-пристроїв (автоматизовано або через мережу) допомагає забезпечити врахування відомих вразливостей [3].

- Аутентифікація на основі PKI. Інфраструктура відкритих ключів використовує асиметричну криптосистему з двома ключами та цифрові сертифікати для підвищення ефективності шифрування і дешифрування даних IoT. PKI є важливим компонентом електронної комерції [3].

- Сегментація. Сегментація - це підхід до безпеки IoT, який передбачає ізоляцію IoT-пристроїв, підключених до Інтернету, в окремі мережі з обмеженим доступом до основної мережі. Це дозволяє організаціям ретельніше контролювати

IoT-мережі та виявляти підозрілу активність до того, як вона призведе до серйозних наслідків [3].

Хоча існує багато різних методологій, кожна з яких відіграє певну роль у забезпеченні безпеки IoT, існують також проблеми, що ускладнюють захист IoT-пристроїв. Нижче наведено деякі з викликів, які можуть завадити організації досягти повноцінного рівня безпеки IoT:

- Тіньові IT. Оскільки IoT-пристрої стають дедалі поширенішими, їх також стає складніше обліковувати. Користувачі часто самостійно підключають власні IoT-пристрої до корпоративних мереж без відома IT-відділу, що унеможливорює ефективний моніторинг цих нових точок доступу до мережі [3].

- Масштабна поверхня атаки. З кожним новим фітнес-трекером, цифровим асистентом або “розумною” лампочкою, які з’являються в організації, зростає кількість потенційно вразливих точок. Чим більша поверхня атаки, тим більше витрат і зусиль необхідно для її захисту [3].

- Складність інтеграції. Більшість компаній використовують різноманітні інструменти безпеки. З появою нових IoT-пристроїв інтеграція сторонніх пристроїв із наявними рішеннями може бути проблематичною [3].

- Використання програмного забезпечення з відкритим кодом. Багато IoT-пристроїв містять програмне забезпечення з відкритим вихідним кодом, який є загальнодоступним. Зловмисники можуть легко отримати інформацію про такий код і виявити вразливості, придатні для експлуатації [3].

- Некеровані обсяги даних. Коли йдеться про дані, їх може бути занадто багато навіть за умови користі. Постійне зростання кількості IoT-пристроїв може призводити до перевантаження даними, що ускладнює їх управління та захист [3].

- Слабкі облікові дані користувачів. Багато IoT-пристроїв призначені для швидкого впровадження “з коробки” та містять стандартні паролі, які користувачам рекомендується змінити. На жаль, дуже часто ці паролі за замовчуванням так і не змінюються, надаючи зловмисникам прямий доступ до пристроїв [3].

- Недостатня обчислювальна потужність. Більшість IoT-пристроїв мають невеликі розміри й можуть не мати достатніх обчислювальних ресурсів для інтеграції важливого програмного забезпечення безпеки [3].

Інтернет речей є важливою складовою сучасних інформаційних та операційних систем, однак його широке впровадження суттєво підвищує рівень кіберризиків і загроз конфіденційності. Значна кількість IoT-пристроїв, різноманітність їх типів, обмежені обчислювальні ресурси та складність централізованого управління ускладнюють забезпечення належного рівня безпеки. Вразливості IoT можуть призводити не лише до витоку даних, а й до негативного впливу на фізичні системи та безпеку людей.

Ефективний захист IoT-інфраструктури потребує комплексного підходу, що включає безпечну мережеву архітектуру, надійну автентифікацію, використання РКІ, сегментацію мереж, постійний моніторинг, аналітику безпеки та своєчасне оновлення пристроїв. Реалізація таких заходів дозволяє знизити ризики кібератак, забезпечити цілісність і достовірність даних, а також підвищити загальний рівень кібербезпеки IoT-систем.

Вибір мікроконтролерної платформи безпосередньо впливає на можливість реалізації криптографічних алгоритмів, механізмів автентифікації, захисту прошивки та стійкості до мережевих атак. У зв'язку з цим у сучасних IoT-рішеннях все більшої уваги приділяють не лише обчислювальним характеристикам контролерів, а й наявності вбудованих апаратно-програмних механізмів безпеки.

На практиці для побудови IoT-систем широко використовуються такі мікроконтролерні платформи, як Arduino, STM32, RP2040, а також спеціалізовані IoT-контролери сімейства ESP. Класичні плати Arduino, попри простоту використання та велику спільноту, мають обмежені обчислювальні ресурси та практично не містять апаратних засобів криптографічного захисту. Платформи STM32 забезпечують високу продуктивність і гнучкість, однак у більшості моделей потребують зовнішніх модулів для бездротового зв'язку та додаткової конфігурації механізмів безпеки, що ускладнює розробку компактних IoT-рішень.

У цьому контексті особливе місце займає платформа ESP32, розроблена компанією Espressif Systems. Вона була створена як універсальний контролер для IoT-застосувань і поєднує в одному чіпі обчислювальні ядра, модулі бездротового зв'язку та апаратні засоби безпеки. ESP32 підтримує Wi-Fi та Bluetooth, що дозволяє реалізовувати повноцінні мережеві IoT-системи без використання зовнішніх комунікаційних модулів. Це суттєво знижує складність апаратної архітектури та зменшує поверхню атаки системи [4].

Важливою особливістю ESP32 є наявність вбудованих апаратних криптографічних модулів, які забезпечують підтримку сучасних алгоритмів шифрування та хешування, зокрема AES, SHA-2, RSA та ECC. Завдяки цьому стає можливим виконання криптографічних операцій без значного навантаження на центральний процесор, що є критично важливим для ресурсно-обмежених IoT-пристроїв. Окрім цього, ESP32 оснащений апаратним генератором випадкових чисел, який використовується для формування криптографічно стійких ключів і ініціалізаційних векторів. Ще одним суттєвим фактором є підтримка механізмів апаратного захисту прошивки та даних. Контролери ESP32 містять незмінювану пам'ять eFuse, яка використовується для зберігання критичних параметрів безпеки, зокрема хешів ключів або конфігураційних бітів. На основі eFuse реалізуються механізми Secure Boot та Flash Encryption, що дозволяють забезпечити контроль цілісності програмного коду та захист даних у флеш-пам'яті навіть у випадку фізичного доступу до пристрою. Такий підхід формує апаратний корінь довіри, що є базовим елементом сучасних захищених IoT-систем [4].

З точки зору кібербезпеки ESP32 є показовим прикладом платформи, у якій апаратний та програмний рівні захисту тісно інтегровані. Це дозволяє реалізовувати принципи “secure by design” та “defense in depth” без залучення зовнішніх спеціалізованих модулів безпеки. Завдяки цьому ESP32 широко використовується не лише в аматорських або навчальних проєктах, але й у промислових і комерційних IoT-рішеннях, де важливими є конфіденційність, цілісність і доступність даних.

1.2. Аналіз типових загроз і вразливостей у IoT-мережах

Технологія Інтернету речей стикається з великою кількістю кіберзагроз, рівень складності яких з часом постійно зростає. Багато пристроїв підключаються до Wi-Fi-мереж через власні незахищені мережеві інтерфейси, що значно спрощує доступ для зловмисників. Деякі пристрої можуть бути скомпрометовані через використання стандартних паролів виробника, що надає хакерам можливість отримати доступ до всієї мережі. Після проникнення в систему зловмисник може реалізувати різні типи атак [5].

Шкідливе програмне забезпечення для IoT - це зловмисне ПЗ, спеціально розроблене для ураження пристроїв Інтернету речей, підключених до мережі. Таке програмне забезпечення може експлуатувати вразливості та слабкі місця в системах безпеки IoT-пристроїв, надаючи зловмисникам несанкціонований доступ, можливість викрадення даних або виконання інших шкідливих дій [6].

IoT-шкідливе ПЗ може поширюватися різними шляхами, зокрема через заражені мобільні додатки, вкладення електронної пошти, методи соціальної інженерії та незахищені мережеві з'єднання. Після зараження хоча б одного пристрою в мережі шкідливе програмне забезпечення може швидко поширитися на інші вразливі пристрої в межах тієї ж мережі, що дозволяє зловмиснику отримати контроль над усією системою [6].

Є декілька видів шкідливого програмного забезпечення, які можуть становити загрозу для інформаційних систем та IoT-пристроїв. Нижче наведено деякі з найбільш поширених із них.

Вірус

Вірус - це самореплікований шкідливий програмний код, який зазвичай приховується в коді програми-носія. Під час “зараження” комп'ютера вірус копіює себе та вбудовує свій код в інші програми. Віруси є найстарішою формою шкідливого ПЗ. Теоретичні основи комп'ютерних вірусів були описані ще у 1949 році, а перший відомий вірус Creeper було зафіксовано в мережі ARPANET у 1971 році. Його метою було виведення повідомлення: “I'm the creeper, catch me if you can!” (Рис. 1.3.) [7].



Рисунок 1.3 – Повідомлення вірусу Creeper

Скреер використовував FTP для копіювання себе з одного комп'ютера на інший. Після перенесення на нову систему програма видаляла себе з оригінальної машини, тобто не створювала кілька копій на кожній системі. Цей процес дозволяв їй “перестрибувати” між системами, не залишаючись на одному місці, функціонуючи як рання форма мобільного застосунку, який міг передавати себе між пристроями [8].

Черв'яки (Worms)

Подібно до вірусів, черв'яки здатні до саморозмноження, однак вони поширюються мережею без участі користувача. Черв'яки використовують вразливості в системах для пошуку нових хостів і автоматичного поширення між пристроями. Відомий черв'як Морріса (1988 р.) став одним із перших масштабних мережеских черв'яків, інфікувавши близько 6 000 комп'ютерів - приблизно 10% усіх пристроїв, підключених до Інтернету на той час [7].

Черв'як Морріса був таємно запущений з комп'ютера Массачусетського технологічного інституту аспірант Роберт Таппан Морріс з Корнельського університету та поширив на підключені до Інтернету комп'ютери, на яких працювала версія BSD UNIX. Черв'як був розроблений таким чином, щоб його неможливо було виявити, але недолік у конструкції призвів до створення набагато більшої кількості своїх копій, ніж очікував Морріс, і спричинив різке перевантаження всіх комп'ютерів, на яких він був встановлений. Це, у свою чергу, дозволило негайно виявити його та виправити недоліки, які він використовував [9].

Троян (Trojan Horses)

Троян - це шкідлива програма, що маскується під легітимне програмне забезпечення. Як і відомий дерев'яний кінь із давньогрецького міфу, троян приховує свій справжній намір. Користувач може відкрити вкладення з фішингового листа, після чого троян встановлюється та активує шкідливе навантаження. Багато троянів створюють приховані канали доступу. Сучасні трояни часто поширюються через легальні платформи на кшталт GitHub або сайтів розповсюдження ПЗ, що робить їх особливо небезпечними для організацій без розвинених механізмів виявлення загроз [7].

У 1990-х роках з'явився сумнозвісний NetBus. Він поширювався, маскуючись під гру Whack-A-Mole, і надавав засоби віддаленого доступу, а також набір “жартівливих” функцій, зокрема відкривання лотка CD-приводу. У 2000 році через масову спам-кампанію поширився шкідливий ILOVEYOU. Він являв собою зловмисний макрос, який маскувався під любовного листа для отримувача. Наприкінці 2005 року троян Zlob розповсюджувався під виглядом нібито необхідного відеокодека у формі компонента ActiveX. Насправді ж він виконував функції бекдору та надавав зловмисникам прихований доступ до системи [10].

Програми-вимагачі (Ransomware)

Ransomware є одним із найпопулярніших інструментів кіберзлочинців. Таке ПЗ блокує доступ користувача до системи, мережі або даних до моменту сплати викупу, зазвичай у криптовалюті. Після оплати зловмисники обіцяють надати ключ для розшифрування, однак його працездатність не гарантується. Кількість атак із використанням програм-вимагачів постійно зростає, а відомі інциденти, зокрема атака Petya (Рис. 1.4.), спричинили значні збої в роботі організацій у всьому світі.

Програма-вимагач Petya поширюється переважно через фішингові електронні листи, що містять шкідливі вкладення. До них можуть належати електронні листи до відділів кадрів із фальшивими заявками на роботу. [7].



Рисунок 1.4 – Повідомлення вірусу Petya

Ризики, пов’язані з IoT-шкідливим ПЗ, можуть бути вкрай серйозними та включати викрадення даних, фінансові збитки й репутаційні втрати. Зловмисники можуть використовувати IoT-шкідливе ПЗ для здійснення DDoS-атак, викрадення конфіденційної інформації, контролю критичної інфраструктури або проведення атак із використанням програм-вимагачів, а також для інших зловмисних дій. Крім того, таке ПЗ часто складно виявити та видалити, що може призводити до тривалих і масштабних наслідків для інформаційної системи.

DDoS-атаки - атаки, спрямовані на порушення нормальної роботи мережі шляхом перевантаження її великою кількістю запитів, що надсилаються за допомогою ботнетів. Унаслідок цього система не здатна обробляти легітимні запити користувачів [7].

Інтернет речей може бути джерелом атак, коли цільова система отримує величезну кількість запитів з кількох джерел - зазвичай це скомпрометовані пристрої (Рис.1.5.). Кіберзлочинці все більше покладаються на “розумні” пристрої для запуску DDoS-атак, оскільки більшість цих пристроїв легко зламати, і їх багато по всьому світу.

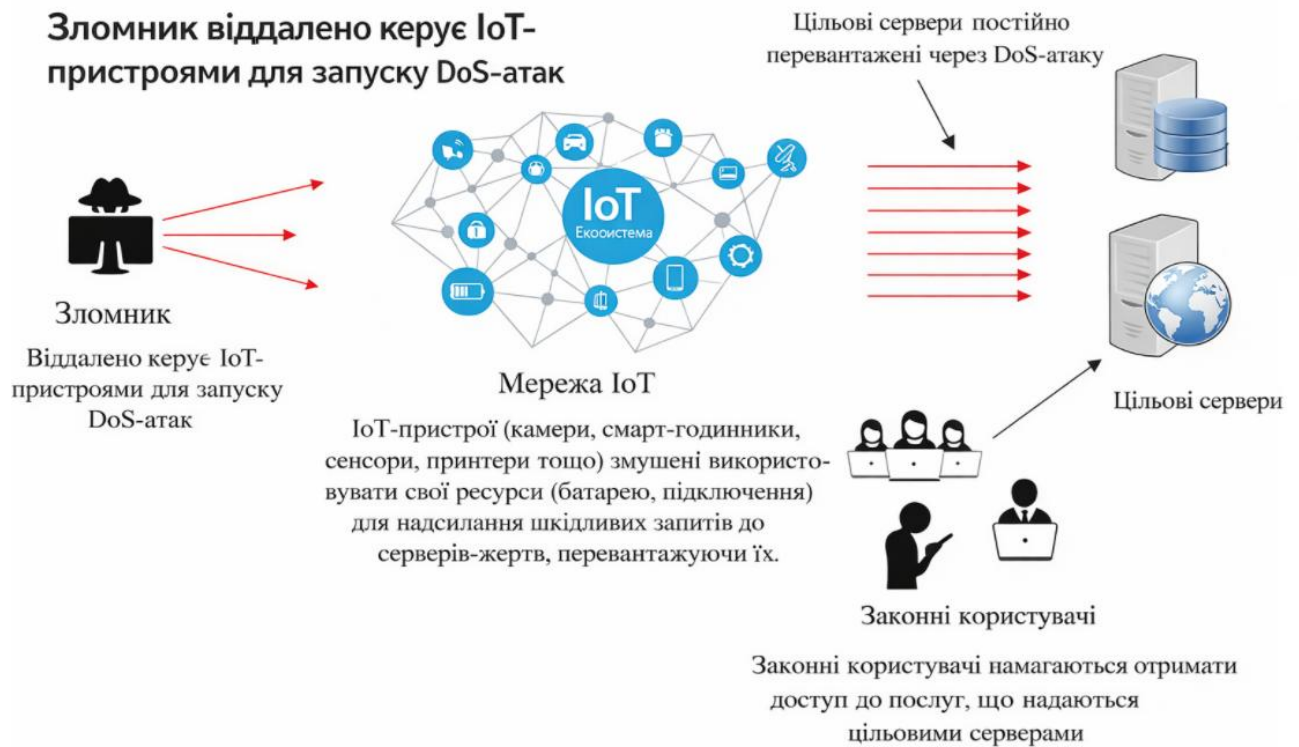


Рисунок 1.5 – DoS атака на IoT

Ботнет Mirai є одним із найпоширеніших векторів атак для здійснення DDoS-атак. Він відомий короткочасними, але надзвичайно інтенсивними атаками. Шкідливе програмне забезпечення Mirai націлене на IoT-пристрої, що мають бекдори або слабкі механізми захисту, які використовуються для проникнення в систему. Після інфікування пристрою шкідливий код самостійно поширюється - сканує мережу, знаходить інші вразливі пристрої, заражає їх і долучає до ботнету [11].

Назва Mirai, що в перекладі з японської означає “майбутнє”, вперше з’явилася у 2016 році. Її створення пов’язують із Парасом Джа (Paras Jha) та Джосайєю Вайтом (Josiah White), які розробили це шкідливе ПЗ для атак на сервери гри Minecraft. Початковою метою було здійснення DDoS-атак за допомогою Mirai для виведення з ладу Minecraft-серверів, а згодом - продаж послуг із захисту від DDoS-атак для цих самих серверів. Коли на ринку з’явилися конкурентні сервіси з протидії DDoS, автори також почали атакувати їх за допомогою Mirai [11].

Хоча спочатку Mirai використовувався як частина шахрайської схеми з метою наживи на популярності Minecraft, уже за тиждень після першого застосування

код було оприлюднено. Це призвело до стрімкого поширення шкідливого ПЗ. Навіть зловмисники з мінімальними технічними знаннями змогли адаптувати вихідний код і створювати власні IoT-ботнети. У результаті Mirai швидко став інструментом масових атак на численні хостинг-провайдери, інтернет-провайдерів, вебсайти, мережі та інші елементи цифрової інфраструктури [11].

Mirai заражає IoT-пристрої, перетворюючи їх на мережу віддалено керованих ботів або “зомбі”. Така мережа називається ботнетом і дозволяє здійснювати потужні, об’ємні DDoS-атаки. Незважаючи на те, що Mirai було створено майже десять років тому, воно та його численні модифікації й досі залишаються серйозною та поширеною загрозою. Mirai цілеспрямовано атакує вразливі IoT-пристрої з відкритими портами або стандартними іменами користувачів і паролями. Багато користувачів не усвідомлюють важливості створення надійних облікових даних для IoT-пристроїв, що робить їх легкою мішенню для атак [11].

Коли ботнет Mirai виявляє вразливий пристрій, він копіює себе, інфікує цей пристрій і переходить до пошуку наступної цілі. З огляду на те, що користувачі поступово стають більш обізнаними у питаннях безпеки та частіше оновлюють паролі, сучасні версії Mirai та його модифікації також активно використовують відомі вразливості програмного забезпечення для проникнення в IoT-пристрої [11].

Атака типу “людина посередині” - це кібератака, під час якої хакер викрадає конфіденційну інформацію, підслуховуючи зв’язок між двома онлайн-цілями, такими як користувач та вебдодаток [12].

Після непомітного втручання в двосторонній зв’язок зловмисники MITM перехоплюють конфіденційні дані, такі як номери кредитних карток, інформацію про облікові записи та облікові дані для входу. Потім хакери використовують цю інформацію для скоєння інших кіберзлочинів, таких як здійснення несанкціонованих покупок, викрадання фінансових рахунків та крадіжка особистих даних. Окрім обміну даними між користувачем та додатком, зловмисник MITM також може підслуховувати приватне спілкування між двома людьми. У цьому сценарії зловмисник перенаправляє та передає повідомлення між двома людьми, іноді змінюючи або замінюючи їх для контролю розмови [12].

Атаки Sniffer на IoT

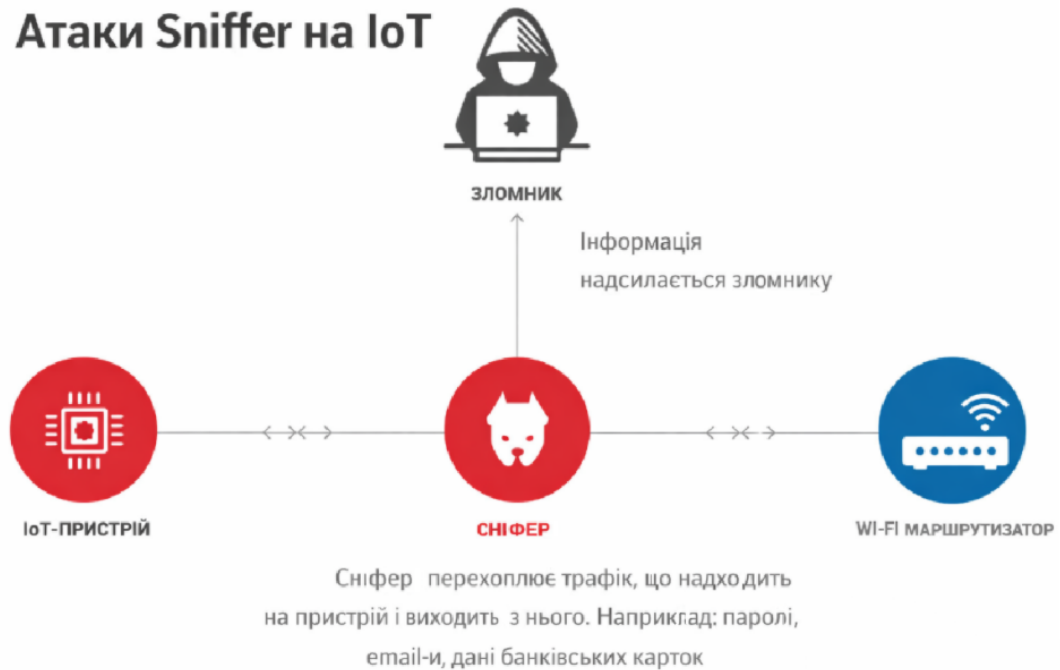


Рисунок 1.6 – MITM/Sniffer атака на IoT

Наприкінці 2022 року споживчий бренд Eufy потрапив під шквал критики, оскільки вони не змогли виправити масові вразливості безпеки своїх камер та брехали про заяви про локальне зберігання відео. Було виявлено, що хакери могли перехоплювати відеопотік та отримувати доступ до систем з привілеями, використовуючи переповнення буфера або атаку MitM (Рис. 1.6.). У березні 2021 року на системи камер безпеки Verkada було здійснено атаку MitM. Зловмисникам, будучи частиною хакерського колективу APT-69420, вдалося отримати облікові дані, що зберігалися на сервері Jenkins, що дозволило їм проникнути в камери клієнтів. Інцидент з витоком даних Equifax у 2017 році також є ідеальним прикладом атаки MitM. Кредитна компанія не поновила відкритий ключ, який вони використовували у своїх системах безпеки, що врешті-решт було виявлено хакерами, які скористалися цим і створили кілька веб-оболонок, заманивши багатьох клієнтів Equifax на фальшиві веб-сайти [13].

Атака методом перебору - це тип кібератаки, під час якої хакери намагаються отримати несанкціонований доступ до облікового запису або зашифрованих даних

методом спроб і помилок, пробуючи кілька облікових даних для входу або ключів шифрування, доки не знайдуть правильний пароль. На відміну від інших кібератак, які використовують вразливості програмного забезпечення, атаки методом перебору використовують обчислювальну потужність та автоматизацію для підбору паролів або ключів. Базові спроби перебору використовують автоматизовані скрипти або ботів для перевірки тисяч комбінацій паролів на хвилину - подібно до того, як злодій пробує всі можливі комбінації на замку, поки той не відкриється. Слабкі або прості паролі полегшують роботу, тоді як сильні можуть зробити цей тип атаки надзвичайно трудомістким або непрактичним. Однак постійно розробляються більш просунуті методи перебору [14].

Попри те, що атаки типу brute force є, ймовірно, найстарішим і найменш складним видом кібератак, вони й досі залишаються надзвичайно поширеними та ефективними. Складається враження, що більшість виробників IoT-пристроїв не відчувають відповідальності за безпеку створюваних ними пристроїв або за захист своїх користувачів. Більшість кінцевих користувачів не усвідомлюють проблем безпеки таких продуктів і навіть у разі зміни стандартного пароля (що роблять далеко не всі) пристрої часто містять приховані або жорстко закодовані облікові записи. У разі їх витоку або виявлення вони фактично стають “відкритими дверима” для будь-кого. Оновлення безпеки для усунення таких проблем або взагалі відсутні, або застосовуються нерегулярно [15].

У результаті мільйони вкрай уразливих пристроїв підключені до Інтернету й становлять легку мішень для атак. За наявності такої кількості пристроїв, якими можна легко керувати, питання полягає лише в часі, коли вони будуть атаковані та залучені до ботнетів для здійснення DDoS-атак або майнінгу криптовалют. Крім того, ці пристрої можуть використовуватися для фішингових атак, вимагання викупу, надання проксі-сервісів або навіть повного виведення пристроїв з ладу [15].

Існує понад 25 000 різних комбінацій облікових даних, які використовуються виключно для brute force-атак на SSH і Telnet. Водночас приблизно 100 із них за-

стосовуються значно частіше за інші. Близько 50% найпопулярніших облікових даних жорстко закодовані у вихідному коді різних варіантів Mirai, а решту дуже легко знайти шляхом звичайного пошуку в Інтернеті [15].

Аналіз цих облікових даних показує наявність двох основних груп. До першої групи належать універсальні облікові дані, такі як “admin:admin”, “root:1234” або “root:root”, які використовуються великою кількістю різних пристроїв. Ця група є загальною та становить близько 27% усіх атак. Друга група складається з облікових даних, специфічних для конкретних пристроїв, зокрема маршрутизаторів, контролерів, принтерів і серверів. Водночас найбільш “готовими до зламу” пристроями, які містять найбільшу кількість таких облікових даних у другій групі, є IP-камери - вони також становлять близько 27% [15].

Інтернет речей значно розширює функціональні можливості сучасних інформаційних систем, проте водночас створює суттєві загрози кібербезпеці. Масове використання IoT-пристроїв, слабкі механізми автентифікації, стандартні або жорстко закодовані облікові дані, відсутність регулярних оновлень і низька обізнаність користувачів призводять до появи великої кількості вразливих вузлів у мережі. Це робить IoT-пристрої легкою ціллю для шкідливого програмного забезпечення, ботнетів і DDoS-атак, зокрема таких як Mirai.

1.3. Огляд сучасних методів захисту IoT-пристроїв та мереж

Одним із базових напрямів захисту IoT-систем є забезпечення безпеки передавання даних. Для цього застосовуються криптографічні механізми, зокрема симетричне та асиметричне шифрування, протоколи TLS, DTLS, а також захищені варіанти MQTT і HTTPS. Шифрування дозволяє гарантувати конфіденційність та цілісність даних, що передаються між пристроями, шлюзами та серверними компонентами. Проте на практиці реалізація криптографічного захисту часто ускладнюється апаратними обмеженнями IoT-пристроїв, що призводить до використання спрощених або застарілих алгоритмів, а інколи - до повної відсутності шифрування [16].

Важливим елементом сучасного захисту є механізми автентифікації та ідентифікації пристроїв. Замість стандартних логінів і паролів дедалі частіше застосовуються унікальні криптографічні ключі, цифрові сертифікати, токени доступу та апаратні ідентифікатори. Такий підхід зменшує ризик несанкціонованого підключення до мережі, однак потребує складнішого управління ключами та інфраструктури довіри, що не завжди є економічно виправданим для невеликих або домашніх IoT-мереж [16].

Ще одним важливим методом захисту є сегментація мережі та ізоляція IoT-пристроїв від критично важливих інформаційних ресурсів. Логічне або фізичне відокремлення IoT-сегмента дозволяє обмежити поширення атаки у разі компрометації одного з пристроїв. У поєднанні з міжмережевими екранами та системами контролю доступу це значно підвищує загальний рівень безпеки. Проте сегментація потребує грамотного проєктування мережі та постійного адміністрування, що ускладнює її впровадження в динамічних або масштабованих середовищах [16].

Окрему категорію сучасних методів становлять системи виявлення вторгнень і аномалій у IoT-мережах. Такі системи аналізують мережевий трафік, поведінку пристроїв і відхилення від нормальних шаблонів роботи. На відміну від класичних IDS/IPS, IoT-орієнтовані рішення часто використовують поведінкові моделі та машинне навчання, що дозволяє виявляти навіть невідомі атаки. Однак ефективність таких підходів значною мірою залежить від якості навчальних даних та коректності базових профілів поведінки [16].

У цьому контексті значну увагу привертають комплексні комерційні платформи безпеки, зокрема рішення класу IoT Security Platform. Одним із таких рішень є **Microsoft Defender for IoT**, який позиціонується як універсальний інструмент для моніторингу, аналізу та захисту IoT і OT-середовищ. Дана платформа забезпечує автоматичне виявлення пристроїв у мережі, аналіз їх поведінки, ідентифікацію вразливостей та кореляцію подій безпеки з іншими компонентами екосистеми Microsoft [17].

Основною перевагою Microsoft Defender for IoT є можливість agentless-моніторингу, тобто аналізу мережевого трафіку без встановлення програмного забезпечення безпосередньо на пристрої. Це є суттєвим плюсом для промислових або застарілих IoT-пристроїв, де інсталяція агентів є неможливою або небезпечною. Крім того, платформа забезпечує централізований огляд стану безпеки та інтеграцію з центрами моніторингу безпеки, що підвищує ефективність реагування на інциденти [17].

Разом із тим аналіз документації та практики використання Microsoft Defender for IoT свідчить про низку суттєвих обмежень. Насамперед, це висока вартість впровадження та експлуатації рішення, зумовлена необхідністю використання хмарної інфраструктури Azure, ліцензування та залучення кваліфікованих фахівців. Для невеликих організацій, навчальних проєктів або домашніх IoT-мереж такі витрати є економічно необґрунтованими. Крім цього, платформа має виражену залежність від екосистеми Microsoft, що знижує гнучкість її використання в гетерогенних середовищах або у випадках, коли організація використовує альтернативні хмарні або локальні рішення. У сценаріях, де критично важливою є автономність мережі або відсутність постійного інтернет-з'єднання, застосування хмарних механізмів аналізу може бути технічно складним або неприйнятним з точки зору політик безпеки [17].

Таблиця 1.1

Порівняльна таблиця переваг і недоліків Microsoft Defender for IoT

Критерій	Переваги	Недоліки / обмеження
Архітектура захисту	Підтримує agentless-моніторинг, що дозволяє аналізувати трафік без встановлення ПЗ на пристрої	Agentless-підхід не забезпечує глибокого контролю внутрішнього стану пристрою (firmware, пам'ять, процеси)
Виявлення пристроїв	Автоматичне виявлення IoT та OT-пристроїв у мережі без попередньої конфігурації	Можливі хибні спрацювання або некоректна ідентифікація нестандартних або саморобних IoT-пристроїв
Виявлення загроз	Підтримка поведінкового аналізу та виявлення невідомих атак	Поведінкові моделі потребують часу для "навчання"; Реакція здебільшого пасивна (alerting), а не активна (device-level blocking)

Масштабованість	Добре масштабується для великих корпоративних і промислових мереж	Надмірно складне та ресурсоємне для малих IoT-мереж або навчальних проєктів
Хмарна аналітика	Потужні аналітичні можливості завдяки хмарним сервісам Azure	Залежність від інтернет-з'єднання та хмарної інфраструктури
Керування ризиками	Надає оцінку рівня ризику пристроїв та рекомендації з безпеки	Рекомендації часто є загальними та не завжди застосовні до обмежених IoT-платформ
Вартість впровадження	Виправдана для великих підприємств з розвинутою SOC-інфраструктурою	Висока вартість ліцензій, хмарних ресурсів та супроводу
Придатність для ESP32 / DIY IoT	Може аналізувати трафік від таких пристроїв на мережевому рівні	Не адаптоване до мікроконтролерів із мінімальними ресурсами

Оскільки значна частина атак на IoT-системи спрямована на підміну прошивки, модифікацію програмного коду або запуск несанкціонованого програмного забезпечення, особливого значення набувають апаратно-програмні засоби контролю цілісності та автентичності коду. Такі механізми дозволяють забезпечити довіру до пристрою ще на етапі його завантаження, незалежно від стану мережі або наявності зовнішніх систем моніторингу. Одним із ключових елементів такого підходу є технологія **Secure Boot**, яка реалізує концепцію довіреного ланцюга завантаження та є базовою складовою сучасної безпеки мікроконтролерних IoT-платформ.

Secure Boot у контролерах ESP реалізує механізм перевірки цифрового підпису прошивки під час кожного завантаження пристрою. Його основна ідея полягає в тому, що виконання коду дозволяється лише у випадку, якщо прошивка була підписана довіреним криптографічним ключем розробника. Під час першого завантаження мікроконтролера публічний ключ або хеш ключа зберігається в незмінюваній пам'яті eFuse, після чого будь-яка спроба запуску модифікованої або неавторизованої прошивки буде автоматично заблокована. Такий підхід унеможливорює підміну програмного забезпечення навіть у випадку фізичного доступу до пристрою або компрометації каналу оновлення. Важливою особливістю Secure Boot в ESP є

його тісна інтеграція з апаратними засобами захисту, що робить атаку шляхом обходу програмних перевірок практично неможливою. Разом із тим використання Secure Boot накладає певні обмеження на процес розробки та оновлення прошивки, оскільки втрата приватного ключа або помилки у процесі підписування можуть призвести до неможливості подальшого оновлення пристрою. Незважаючи на це, Secure Boot є одним із найбільш ефективних і доцільних механізмів захисту IoT-пристроїв, особливо у проектах, де критично важливо гарантувати цілісність програмного коду та запобігти виконанню несанкціонованого програмного забезпечення [18].

Важливим аспектом кібербезпеки залишається стійкість системи до мережових атак. Навіть повністю захищений на апаратному рівні пристрій може бути виведений з ладу шляхом перевантаження каналів зв'язку або обчислювальних ресурсів. Особливо небезпечними в цьому контексті є атаки типу Denial of Service (DoS) та Distributed Denial of Service (DDoS), які спрямовані не на компрометацію даних, а на порушення доступності сервісів. Для IoT-мереж такі атаки становлять підвищену загрозу через обмежені ресурси пристроїв, масовість підключень і часто відсутність повноцінних механізмів мережевого захисту.

Одним із базових методів захисту IoT-мереж від DoS-атак є контроль і обмеження мережевого трафіку. Застосування механізмів **rate limiting** дозволяє обмежити кількість запитів, які пристрій або сервіс обробляє за одиницю часу. Це зменшує ймовірність перевантаження, однак має і суттєві недоліки. Надмірно жорсткі обмеження можуть негативно впливати на легітимну роботу системи, особливо у сценаріях з піковими навантаженнями або реального часу. Крім того, для простих IoT-пристроїв реалізація гнучкого контролю трафіку може бути складною через обмежені ресурси [19].

Одним з поширених аналогів є **фільтрація трафіку за правилами**. Даний підхід передбачає блокування або обмеження доступу на основі IP-адрес, портів або протоколів. Його основною перевагою є простота реалізації, однак у контексті DoS-атак він є малоефективним. Атаки часто здійснюються з динамічних або підмінних IP-адрес, а в разі DDoS - з тисяч джерел одночасно. На відміну від rate

limiting, фільтрація не обмежує інтенсивність трафіку, а лише реагує на заздалегідь відомі параметри, що робить її реактивною та легко обхідною. Іншим підходом є **повна автентифікація кожного запиту із використанням криптографічних механізмів**. Теоретично це дозволяє відсіяти неавторизовані запити, однак на практиці такий метод створює додаткове обчислювальне навантаження на IoT-пристрої. Під час DoS-атаки навіть обробка невдалих спроб автентифікації може призвести до виснаження ресурсів, що фактично перетворює сам механізм захисту на фактор відмови в обслуговуванні. Ще одним аналогом є **черги та пріоритизація трафіку**, де запити обробляються відповідно до заданих класів пріоритету. Хоча QoS дозволяє забезпечити обслуговування критичних повідомлень, він не запобігає самому перевантаженню системи. При масовій атаці черги швидко заповнюються, а затримки зростають до неприйняттого рівня.

Таблиця 1.2

Порівняльна характеристика методів захисту IoT-систем від DoS-атак

Метод захисту	Принцип роботи	Переваги	Недоліки	Оцінка доцільності для IoT
Rate limiting	Обмеження кількості запитів/пакетів за одиницю часу	Простота реалізації; мінімальні вимоги до ресурсів; ефективний проти різних типів DoS; запобігає виснаженню ресурсів	Потребує коректного підбору порогів	Найбільш доцільний та універсальний метод
Фільтрація трафіку (ACL, firewall)	Блокування за IP, портами, протоколами	Легка реалізація; низьке навантаження	Не захищає від масових атак з різних джерел; легко обходиться	Низька ефективність
Повна автентифікація кожного запиту	Перевірка легітимності кожного з'єднання	Відсіювання неавторизованих клієнтів	Високе навантаження на CPU; може спричинити DoS сама по собі	Малопридатна
QoS та пріоритизація трафіку	Обробка запитів за пріоритетами	Забезпечує доставку критичних повідомлень	Не зменшує обсяг трафіку; черги переповнюються	Обмежена ефективність

Особливу загрозу становлять атаки типу MITM, які є одними з найпоширеніших у бездротових та розподілених IoT-мережах.

Однією з основних причин вразливості IoT до MITM-атак є недостатній рівень захисту мережевих протоколів. Використання незашифрованих або слабко захищених протоколів передавання даних дозволяє зловмиснику легко здійснювати перехоплення трафіку. Навіть у випадку застосування шифрування, відсутність перевірки автентичності сторін може призвести до успішної реалізації атаки шляхом підміни ключів або сертифікатів. Дослідження показують, що значна кількість IoT-пристроїв або взагалі не використовує криптографічний захист, або застосовує його некоректно, що суттєво підвищує ризик MITM-атак.

Одним із ключових методів захисту від MITM є **використання криптографічно захищених каналів зв'язку**. Застосування протоколів TLS або DTLS забезпечує шифрування даних і захист від їх модифікації під час передавання. Важливою умовою ефективності такого підходу є коректна реалізація перевірки сертифікатів і автентифікації сторін. Без цього шифрування саме по собі не гарантує захисту від атак із підміною вузлів зв'язку. Не менш важливим є використання механізмів взаємної автентифікації, коли не лише клієнт підтверджує легітимність сервера, а й сервер перевіряє справжність клієнта. У контексті IoT це може реалізовуватися за допомогою цифрових сертифікатів, попередньо згенерованих криптографічних ключів або апаратних ідентифікаторів пристроїв. Такий підхід значно ускладнює можливість підміни однієї зі сторін з'єднання, однак вимагає наявності інфраструктури управління ключами та сертифікатами, що може бути складним для маломасштабних систем [20].

Важливу роль у протидії MITM-атакам відіграє захист на рівні завантаження та прошивки пристрою. Використання механізмів Secure Boot гарантує, що пристрій виконує лише довірений програмний код, унеможливаючи впровадження шкідливих компонентів, здатних перехоплювати або модифікувати мережевий трафік. У поєднанні з шифруванням флеш-пам'яті це дозволяє знизити ризик внутрішніх MITM-атак, реалізованих безпосередньо на самому IoT-пристрої.

Одними з найпоширеніших і водночас найпростіших у реалізації є атаки типу brute force, які активно застосовуються проти IoT-пристроїв у глобальних мережах.

Одним із базових методів захисту від brute-force атак є відмова від використання стандартних облікових даних і впровадження унікальних ідентифікаційних даних для кожного пристрою. Застосування унікальних паролів або криптографічних ключів значно ускладнює автоматизований перебір, однак потребує налагодженої системи управління обліковими даними, що не завжди реалізується в масових IoT-рішеннях.

Ефективним засобом протидії brute-force атакам є обмеження кількості спроб автентифікації за певний проміжок часу. Механізми тимчасового блокування або затримки між спробами входу зменшують швидкість перебору і роблять атаку практично неефективною. У порівнянні з повною блокадою доступу, використання прогресивних затримок дозволяє зберегти працездатність системи для легітимних користувачів, що є важливим для IoT-пристроїв з обмеженими інтерфейсами керування. Ще одним важливим підходом є використання альтернативних методів автентифікації, які не залежать від введення паролів. Застосування криптографічних ключів, сертифікатів або токенів дозволяє усунути саму можливість перебору облікових даних. Для IoT-платформ такий підхід є більш надійним, проте вимагає наявності апаратних або програмних засобів безпечного зберігання ключів і коректної реалізації криптографічних операцій [21].

Реалізація механізмів довіреного завантаження, зокрема Secure Boot, дозволяє гарантувати цілісність програмного коду та унеможливити виконання несанкціонованого програмного забезпечення на рівні пристрою. Водночас мережеві методи захисту, такі як rate limiting та захищені канали зв'язку, забезпечують стійкість системи до атак, спрямованих на порушення доступності та перехоплення даних.

Порівняльний аналіз показав, що прості та превентивні механізми є більш доцільними для малоресурсних IoT-платформ, ніж складні та ресурсомісні рішення корпоративного рівня. Таким чином, вибір методів захисту IoT-систем має базуватися на балансі між рівнем безпеки, складністю реалізації та доступними ресурсами, що є особливо важливим при проєктуванні систем моніторингу IoT-мереж на базі мікроконтролерів.

1.4. Постановка задачі дослідження та вимоги до системи кібербезпеки

Аналіз сучасного стану розвитку IoT як платформи для моніторингу, а також дослідження основних кіберзагроз і методів кіберзахисту в IoT-середовищі дозволяють сформулювати вимоги до розробки системи моніторингу з підвищеним рівнем безпеки. Така система повинна забезпечувати не лише ефективний збір і обробку даних, але й відповідати сучасним вимогам інформаційної безпеки з урахуванням обмежених ресурсів IoT-пристроїв.

Метою розробки є створення захищеної системи моніторингу IoT-мережі на базі мікроконтролерної платформи ESP32, яка забезпечує безпечний збір, передачу, обробку та зберігання даних у реальному часі. Система призначена для використання у розподілених IoT-середовищах, де важливими є надійність роботи, масштабованість і стійкість до кіберзагроз.

Розроблювана система моніторингу повинна забезпечувати виконання таких основних функцій:

- збір даних з датчиків, підключених до IoT-пристроїв на базі ESP32;
- попередню обробку даних на рівні кінцевого пристрою;
- передачу даних до серверної або хмарної частини системи;
- підтримку віддаленого керування та адміністрування IoT-пристроїв.

З урахуванням специфіки IoT-середовища система повинна відповідати таким вимогам кібербезпеки:

- захист каналів зв'язку шляхом використання криптографічних методів шифрування;
- забезпечення цілісності та конфіденційності даних моніторингу;
- реалізація механізмів контролю доступу до функцій системи;
- захист від несанкціонованого доступу та підміни даних;
- підтримка безпечного оновлення програмного забезпечення IoT-пристроїв.

Як апаратну основу системи передбачається використання мікроконтролерної платформи ESP32, яка забезпечує необхідні обчислювальні можливості, підтримку бездротових інтерфейсів та апаратні механізми безпеки. Програмна частина

системи повинна бути реалізована з використанням сучасних IoT-протоколів та бібліотек, що підтримують криптографічний захист і безпечну передачу даних.

Програмне забезпечення повинно бути модульним, масштабованим та адаптованим до обмежених ресурсів IoT-пристроїв, а також забезпечувати можливість подальшого розширення функціоналу системи.

У результаті виконання завдання має бути розроблена система моніторингу IoT-мережі з підвищеним рівнем кібербезпеки, яка демонструє ефективність запропонованих методів захисту та може бути використана як основа для подальшого розвитку захищених IoT-рішень.

Висновок до першого розділу

Аналіз особливостей IoT-мереж показав, що ключовими факторами, які ускладнюють забезпечення належного рівня безпеки, є обмежені обчислювальні ресурси пристроїв, різноманітність апаратних і програмних платформ, відсутність централізованого управління та часте використання стандартних або слабких механізмів автентифікації.

У межах аналізу типових загроз було встановлено, що IoT-мережі є вразливими до широкого спектра атак, серед яких особливу небезпеку становлять шкідливе програмне забезпечення, атаки типу DoS/DDoS, Man-in-the-Middle та brute force. Наведені приклади реальних інцидентів свідчать про те, що більшість успішних атак на IoT-інфраструктуру зумовлені поєднанням технічних вразливостей, відсутності оновлень безпеки та низького рівня захисту облікових даних. Масове використання IoT-пристроїв у складі ботнетів, зокрема для здійснення DDoS-атак, підтверджує критичну необхідність впровадження превентивних механізмів захисту вже на етапі проектування систем.

Огляд сучасних методів і технологій кіберзахисту IoT-пристроїв та мереж показав, що ефективне забезпечення безпеки можливе лише за умови комплексного підходу, який поєднує апаратно-програмні механізми контролю цілісності, криптографічний захист даних, надійну автентифікацію та мережеві методи протидії атакам.

РОЗДІЛ 2. ПРОЕКТУВАННЯ ТА ЗАБЕЗПЕЧЕННЯ КІБЕРБЕЗПЕКИ СИСТЕМИ МОНІТОРИНГУ IoT-МЕРЕЖІ НА БАЗІ ESP32

2.1. Функціональні та нефункціональні вимоги до системи моніторингу

Перед початком проектування та реалізації системи моніторингу IoT-мережі необхідно чітко визначити вимоги, яким вона повинна відповідати. Формулювання функціональних та нефункціональних вимог є ключовим етапом, оскільки саме вони задають межі та напрям подальшої розробки, визначають архітектурні рішення, вибір апаратної платформи, програмних компонентів і механізмів кіберзахисту.

У контексті IoT-систем особливу роль відіграє той факт, що такі рішення функціонують в умовах обмежених обчислювальних ресурсів, нестабільних мережових з'єднань та підвищеного рівня кіберзагроз. Це унеможливорює пряме застосування багатьох класичних підходів, характерних для корпоративних інформаційних систем, і вимагає пошуку компромісу між рівнем безпеки, продуктивністю та складністю реалізації.

Функціональні вимоги визначають, що саме система повинна робити: які дані збирати, як їх обробляти, передавати, зберігати та відображати користувачеві. Нефункціональні вимоги, у свою чергу, описують як система повинна це робити, тобто задають вимоги до надійності, безпеки, продуктивності, масштабованості та зручності використання.

Функціональні вимоги до системи

Система моніторингу IoT-мережі повинна забезпечувати виконання таких основних функцій:

- Збір даних з сенсорів, підключених до IoT-пристроїв на базі платформи ESP32, у режимі реального часу або з заданою періодичністю.
- Попередню обробку даних на рівні кінцевого пристрою, включаючи фільтрацію, нормалізацію та перевірку коректності отриманих вимірювань.
- Безпечну передачу даних від IoT-пристроїв до серверної або хмарної частини системи з використанням захищених протоколів зв'язку.

- Зберігання даних моніторингу з можливістю ведення історії вимірювань та подальшого аналізу.
- Візуалізацію даних у зручному для користувача вигляді через вебінтерфейс або панель керування.
- Управління IoT-пристроями, зокрема можливість дистанційного налаштування параметрів та керування виконавчими пристроями.
- Автентифікацію та авторизацію користувачів і пристроїв з розмежуванням прав доступу.
- Моніторинг стану системи, включаючи контроль доступності пристроїв, журналювання подій та сповіщення про збої або підозрілу активність.

Нефункціональні вимоги до системи

Нефункціональні вимоги визначають якісні характеристики системи та обмеження її функціонування. До основних нефункціональних вимог належать:

- Система повинна забезпечувати конфіденційність, цілісність і доступність даних, а також бути стійкою до типових атак на IoT-середовище, зокрема несанкціонованого доступу, перехоплення трафіку та атак типу відмова в обслуговуванні.
- Система повинна забезпечувати обробку та передачу даних із мінімальними затримками, достатніми для реалізації моніторингу у режимі реального часу.
- Система повинна коректно функціонувати в умовах нестабільного мережевого з'єднання, забезпечувати повторну передачу даних та відновлення роботи після збоїв.
- Система повинна підтримувати стандартні IoT-протоколи та бути сумісною з різними типами сенсорів і серверних платформ.
- Вебінтерфейс і засоби адміністрування повинні бути інтуїтивно зрозумілими та не потребувати спеціальної підготовки користувачів.

Ключовими з точки зору кібербезпеки є вимоги до автентифікації, контролю доступу, захисту даних і стійкості до типових мережевих атак, тоді як надмірно

складні або ресурсоємні механізми не є доцільними для мікроконтролерних платформ.

Зосередження на критично важливих вимогах дозволяє забезпечити баланс між рівнем захищеності та працездатністю системи. Визначені вимоги створюють основу для проєктування архітектури захищеної системи моніторингу та обґрунтовують вибір конкретних технологій і методів кіберзахисту, що розглядаються у наступних підрозділах роботи.

2.2. Архітектура та реалізація захищеної системи моніторингу IoT-мережі на базі ESP32

Розробка захищеної системи моніторингу IoT-мережі потребує комплексного підходу, який охоплює проєктування архітектури системи, інтеграцію апаратних компонентів та створення програмного забезпечення з урахуванням вимог кібербезпеки.

DOIT Esp32 DevKit v1 (Рис. 2.1.) - це одна з плат розробки, створена компанією DOIT для оцінки модуля ESP-WROOM-32. Вона побудована на основі мікроконтролера ESP32, який підтримує WiFi, Bluetooth, Ethernet і функцію низького енергоспоживання в одному чіпі [21].



Рисунок 2.1 – Плата DOIT Esp32 DevKit v1

Флеш-пам'ять модуля ESP32 організована у вигляді однієї області з сторінками по 4096 байт кожна. Флеш-пам'ять починається з адреси 0x00000, хоча багато

областей зарезервовано для Esp32 IDF SDK та Zerynth VM. Існує два різних макети в залежності від підтримки Bluetooth Low Energy [21].

Контролер ESP32 є однією з найбільш поширених апаратних платформ для побудови IoT-систем завдяки поєднанню низької вартості, високої продуктивності та широких комунікаційних можливостей. Водночас ESP32 вирізняється наявністю розвинених апаратно-програмних механізмів безпеки, що дозволяє використовувати його у проєктах, де критично важливими є цілісність програмного коду, захист даних та протидія несанкціонованому доступу. Архітектура безпеки ESP32 була спроектована з урахуванням типових загроз IoT-середовища та реалізує концепцію багаторівневого захисту [21].

Основою безпеки ESP32 є апаратний корінь довіри, який формується ще на етапі виробництва мікроконтролера. У пам'яті eFuse зберігаються незмінювані бітові поля, що використовуються для збереження криптографічних ключів, хешів або конфігураційних прапорців безпеки. Після програмування ці значення не можуть бути змінені або прочитані у відкритому вигляді, що забезпечує високий рівень захисту навіть у випадку фізичного доступу до пристрою. Саме eFuse виступає фундаментом для реалізації таких механізмів, як Secure Boot і Flash Encryption [21].

Важливим компонентом архітектури безпеки ESP32 є механізм Secure Boot, який забезпечує перевірку автентичності та цілісності програмного коду на кожному етапі завантаження. Процес стартує з ROM-завантажувача, який є апаратно зашитим і не може бути змінений. Під час завантаження прошивки виконується криптографічна перевірка цифрового підпису, і лише у випадку успішної валідації код допускається до виконання. Таким чином унеможлиблюється запуск модифікованої або шкідливої прошивки, навіть якщо злоумисник отримав доступ до флеш-пам'яті пристрою. Secure Boot формує безперервний ланцюг довіри, який гарантує, що кожен наступний компонент системи є легітимним [21].

Доповненням до Secure Boot є механізм шифрування флеш-пам'яті, відомий як Flash Encryption. Його призначення полягає у захисті даних і програмного коду, збережених у зовнішній пам'яті SPI Flash, від несанкціонованого зчитування або

аналізу. Усі дані автоматично шифруються при записі та дешифруються під час читання за допомогою апаратного AES-блоку, що працює прозоро для програмного забезпечення. Ключі шифрування зберігаються в eFuse і ніколи не з'являються у відкритому вигляді в оперативній пам'яті, що значно ускладнює атаки типу reverse engineering або фізичне клонування пристроїв [21].

ESP32 також оснащений апаратними криптографічними прискорювачами, які забезпечують виконання складних криптографічних операцій без значного навантаження на центральний процесор. Підтримка алгоритмів AES, SHA, RSA, ECC та генератора апаратних випадкових чисел дозволяє реалізовувати захищені канали зв'язку, цифрові підписи та механізми автентифікації навіть на пристроях з обмеженими ресурсами. Використання апаратних прискорювачів не лише підвищує продуктивність, але й зменшує ризик атак, пов'язаних із побічними каналами, оскільки критичні операції виконуються у спеціалізованих модулях [21].

У сукупності механізми безпеки ESP32 формують цілісну платформу для створення захищених IoT-систем, у яких апаратний рівень захисту тісно інтегрований із програмними засобами. На відміну від багатьох інших мікроконтролерів, де безпека реалізується переважно програмними методами, ESP32 надає вбудовані апаратні інструменти, що значно підвищують стійкість до широкого спектра атак. Це робить ESP32 придатним для використання у системах моніторингу, керування та збору даних, де безпека є одним із ключових вимог [21].

Проект роботи було виконано у Wokwi - сучасному онлайн-симуляторі вбудованих систем та IoT-проектів, що дозволяє користувачам моделювати роботу мікроконтролерів і електронних схем без фізичного апаратного забезпечення.

Платформа підтримує емуляцію широкого спектра плат, зокрема ESP32, STM32, Arduino та Raspberry Pi Pico, а також дозволяє підключати до них різні датчики, дисплеї, модулі зв'язку та інші периферійні компоненти у віртуальному середовищі. Симуляція здійснюється безпосередньо в браузері, що усуває необхідність інсталяції спеціального програмного забезпечення та надає можливість швидкого прототипування та тестування коду у реальному часі. Крім стандартних можливостей емуляції, Wokwi підтримує розробку проектів на різних мовах (C/C++,

MicroPython, Rust), використання захищених каналів зв'язку, а також інтеграцію з середовищем розробки Visual Studio Code за допомогою відповідних розширень. Такий функціональний набір робить Wokwi зручним інструментом для навчання, розробки та відлагодження IoT-систем, що особливо корисно при тестуванні програмних рішень до їхньої реалізації на реальному апаратному рівні [23].

На відміну від більшості альтернатив, Wokwi орієнтований не лише на логіку програмного коду, а й на реалістичну поведінку периферії, таймерів, GPIO та мережевих інтерфейсів, що робить результати симуляції максимально наближеними до реального пристрою.

Додатковою перевагою Wokwi є його доступність: симулятор працює без встановлення складного програмного забезпечення, підтримує інтеграцію з популярними середовищами розробки та дозволяє швидко відтворювати типові сценарії роботи IoT-систем. Це робить його особливо зручним для навчальних і наукових проєктів, де важливими є відтворюваність експериментів і мінімальні апаратні витрати.

Запропонована архітектура системи моніторингу (Рис.2.2.) включає такі основні рівні:

- рівень сенсорів і виконавчих пристроїв, які здійснюють вимірювання параметрів навколишнього середовища або виконання керуючих дій;
- рівень IoT-пристрою на базі ESP32, який забезпечує збір, попередню обробку та безпечну передачу даних;
- серверний рівень, призначений для зберігання, обробки та аналізу отриманих даних;
- рівень вебінтерфейсу, що забезпечує доступ користувачів до результатів моніторингу та керування системою.

Такий підхід дозволяє розмежувати функції між рівнями системи, підвищити її масштабованість та забезпечити впровадження механізмів кіберзахисту на кожному етапі обробки даних.

Інтеграція сенсорів і виконавчих пристроїв у систему моніторингу здійснюється з урахуванням вимог безпеки та надійності. ESP32 виступає центральним елементом, що забезпечує взаємодію між апаратними компонентами та серверною частиною системи. Підключення сенсорів реалізується через стандартні інтерфейси (GPIO, I2C, SPI, UART), що дозволяє забезпечити сумісність з різними типами пристроїв.

Розроблену схему системи моніторингу відображено на рисунку 2.2.

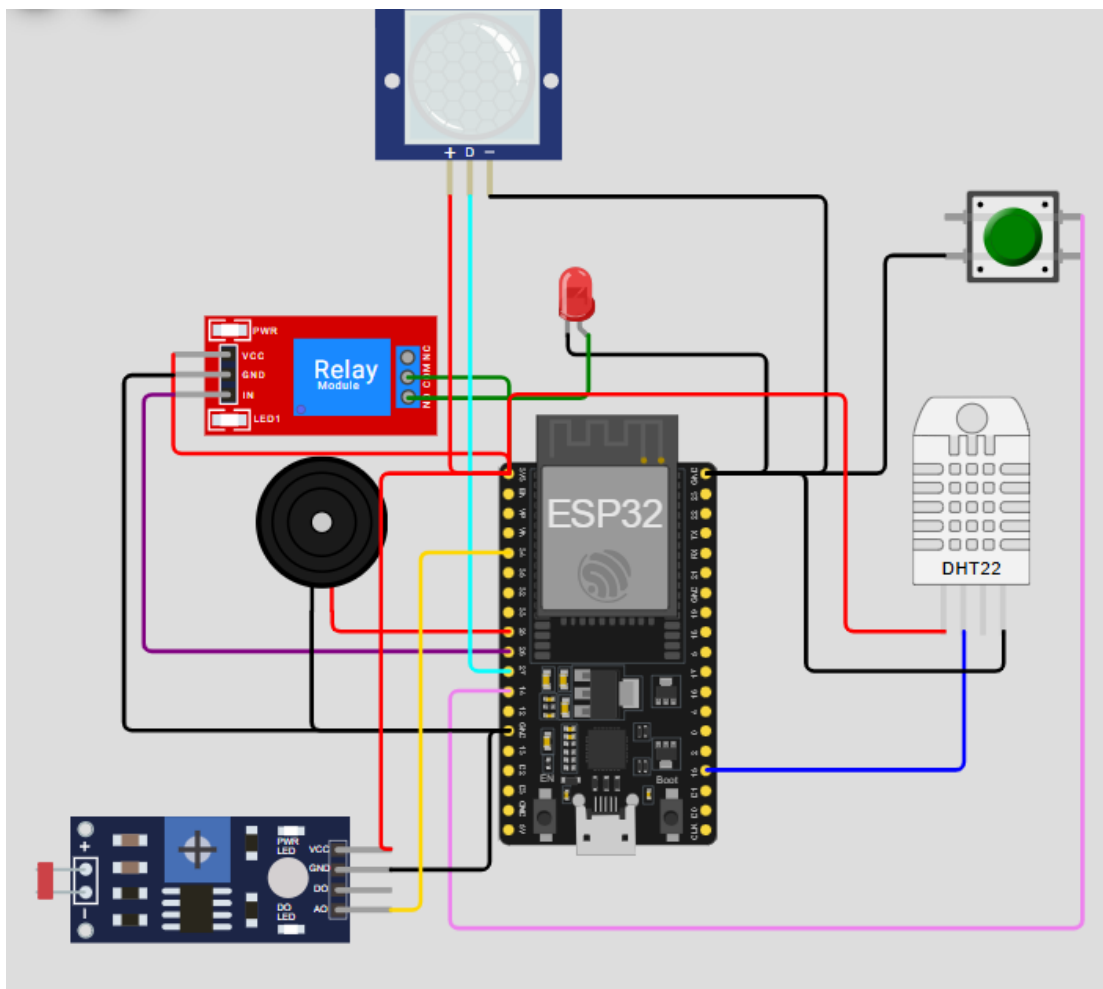


Рисунок 2.2 – Схема системи моніторингу розроблена в симуляторі Wokwi

Запропонована архітектура системи моніторингу IoT-мережі визначає структурну організацію взаємодії між апаратними компонентами, програмним забезпеченням мікроконтролера ESP32 та клієнтською частиною системи. Архітектура охоплює рівень сенсорів, рівень IoT-пристрою та рівень взаємодії з користувачем і

враховує вимоги до безпеки, продуктивності та масштабованості. Обрана архітектурна модель створює основу для подальшої реалізації програмних механізмів захисту, зокрема автентифікації, контролю доступу та протидії мережевим атакам, а також забезпечує цілісність і узгодженість усіх компонентів системи в межах єдиного проєкту.

2.3. Забезпечення безпеки доступу та захист даних

Програмне забезпечення для ESP32 розроблялося з урахуванням принципів “secure by design” та “defense in depth”. Прошивка мікроконтролера відповідає за ініціалізацію апаратних компонентів, обробку даних сенсорів, управління виконавчими пристроями та комунікацію із серверною частиною системи.

Таблиця 2.1

Реалізація “secure by design”

Принцип	Реалізація
Мінімальний доступ	Ролі viewer / operator / admin
Ролі viewer / operator / admin	Усі API-запити вимагають автентифікації
Перевірка	Кожен запит проходить перевірку безпеки
Fail-safe	У разі помилки - доступ забороняється

У розробленій системі контроль доступу реалізований з використанням криптографічних ключів. Основним призначенням ключів є підтвердження автентичності джерела запиту та забезпечення цілісності переданих даних.

Криптографічний ключ зберігається безпосередньо на мікроконтролері ESP32 та використовується виключно для формування і перевірки криптографічних підписів запитів. Ключ не передається мережею та не зберігається у відкритому вигляді на стороні клієнта або у вебінтерфейсі, що значно знижує ризик його компрометації.

Кожен HTTP-запит до системи містить:

- токен доступу користувача;
- лічильник запитів (counter);
- криптографічний підпис, сформований за алгоритмом HMAC-SHA256.

На стороні клієнта формується підпис на основі комбінації параметрів запиту та секретного ключа. На стороні ESP32 з використанням локально збереженого ключа обчислюється власний HMAC-підпис, який порівнюється з підписом, отриманим у запиті. У разі неспівпадіння підписів запит вважається неавтентичним і відхиляється (Рис. 2.3.).

Навіть у разі часткового або фізичного доступу до мікроконтролера ESP32 злоумисник не може отримати доступ до функціоналу системи без коректно сформованого криптографічного підпису. Ключ використовується лише всередині пристрою для перевірки автентичності запитів і не доступний через мережеві або прикладні інтерфейси.

Автентифікація реалізована на основі токенів доступу з криптографічною перевіркою автентичності кожного запиту. Такий підхід дозволяє підтвердити легітимність джерела запиту без передачі секретного ключа мережею. Для запобігання повторному використанню повідомлень застосовується механізм захисту від replay-атак, що ґрунтується на перевірці монотонно зростаючого лічильника запитів (Рис. 2.3.).

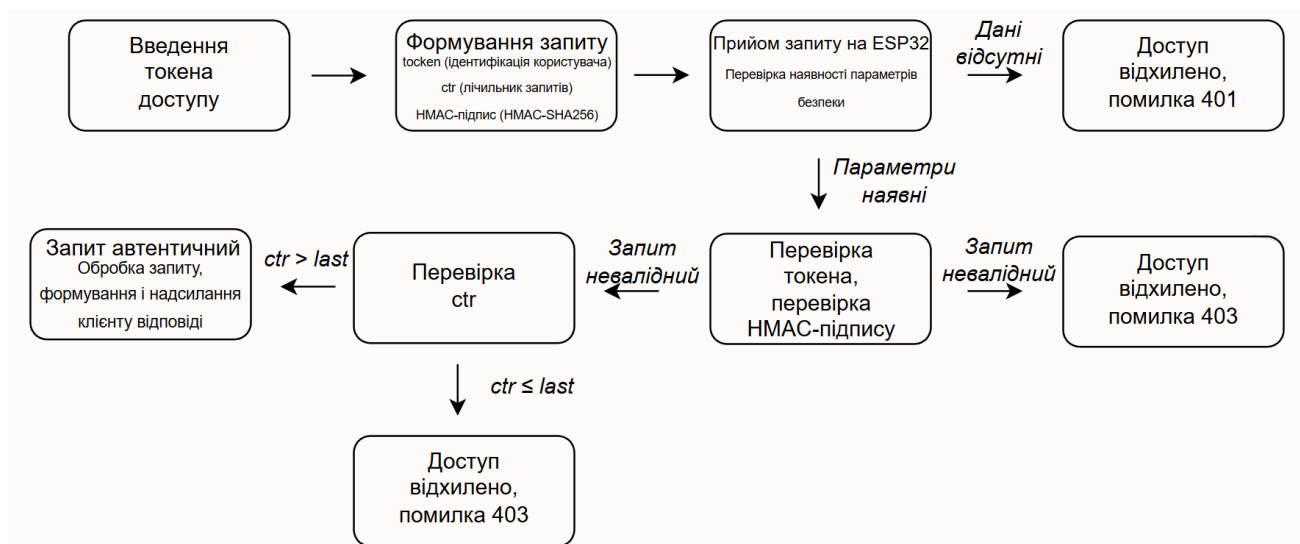


Рисунок 2.3 – Блок-схема автентифікації запитів

Авторизація у системі реалізована за принципом рольового доступу (RBAC). Кожному автентифікованому користувачу призначається роль, яка визначає перелік дозволених дій у системі. У межах реалізації визначено ролі viewer, operator та

admin. Користувачі з роллю viewer мають доступ виключно до перегляду даних моніторингу, тоді як operator та admin отримують можливість керування виконавчими пристроями. Перевірка ролі виконується на стороні ESP32 перед виконанням будь-якої критичної операції.

Таблиця 2.2

Порівняння ролей користувачів

Функціональність	Роль		
	Viewer	Operator	Admin
Аутентифікація в системі	+	+	+
Перегляд температури та вологості	+	+	+
Перегляд стану датчика руху	+	+	+
Перегляд історії параметрів середовища	+	+	+
Перегляд історії руху	-	+	+
Оновлення даних (Refresh)	+	+	+
Керування виконавчими пристроями	-	+	+
Додавання пристроїв	-	-	+
Додавання користувачів	-	-	+

Функціонування систем моніторингу IoT-мережі в умовах відкритих і бездротових середовищ обумовлює підвищені ризики виникнення кіберзагроз. Особливістю IoT-систем є використання великої кількості ресурсно-обмежених пристроїв, що ускладнює застосування традиційних засобів захисту. Тому виявлення та протидія кіберзагрозам повинні базуватися на поєднанні профілактичних заходів, механізмів моніторингу та реагування на інциденти безпеки.

Атаки типу “відмова в обслуговуванні” є однією з найбільш поширених загроз для IoT-систем, оскільки можуть призводити до повної або часткової недоступності сервісів моніторингу. У таких атаках зловмисники намагаються перевантажити IoT-пристрої або серверну частину системи великою кількістю запитів.

Для запобігання перевантаженню мікроконтролера ESP32 у системі реалізовано механізм обмеження частоти звернень до вбудованого вебсервера.

У поточній реалізації застосовано такі параметри:

- часове вікно: 1 секунда;
- максимальна кількість запитів: 5 HTTP-запитів у межах одного вікна;
- реакція на перевищення ліміту: автоматичне відхилення подальших запитів без виконання прикладної логіки.

Кожен вхідний HTTP-запит фіксується серверною частиною системи, при цьому ведеться підрахунок кількості запитів у межах заданого часовго інтервалу. Порогове значення кількості запитів є програмно заданим і може бути змінене залежно від умов експлуатації та апаратних обмежень пристрою.

У разі перевищення допустимого порогового значення кількості запитів:

- подальші запити автоматично відхиляються;
- прикладна логіка системи не виконується;
- ESP32 повертає відповідь про відмову в обробці запиту.

Механізми протидії DoS/DDoS-атакам у системі тісно інтегровані з механізмами автентифікації. Запити, що не містять коректного токена або криптографічного підпису, не лише вважаються неавторизованими, але й одразу виключаються з подальшої обробки. Це дозволяє ефективно відокремити легітимний трафік від потенційно шкідливого.

Атаки типу “людина посередині” спрямовані на перехоплення та модифікацію даних під час їх передачі між компонентами IoT-системи. Такі атаки є особливо небезпечними для систем моніторингу, оскільки можуть призвести до спотворення даних або несанкціонованого керування пристроями.

Для забезпечення взаємної довіри між клієнтом і ESP32 кожен запит проходить обов’язкову автентифікацію. Клієнт доводить свою легітимність шляхом формування криптографічного підпису, тоді як IoT-пристрій перевіряє коректність цього підпису перед обробкою запиту. Контроль цілісності реалізовано шляхом використання HMAC-підпису, який обчислюється на основі вмісту запиту, ідентифікатора користувача та лічильника запитів. Будь-яка спроба модифікації даних під час передачі призводить до порушення підпису та відхилення запиту. Навіть у разі

перехоплення коректного запиту його повторне використання неможливе завдяки механізму захисту від replay-атак.

На цьому етапі перевіряється наявність обов'язкових параметрів безпеки (рис. 2.3.), зокрема:

- токена доступу;
- лічильника запитів;
- криптографічного підпису (HMAC).

Запити, що не містять зазначених параметрів або мають некоректну структуру, відхиляються негайно, що дозволяє зменшити навантаження на процесор і пам'ять ESP32.

Атаки типу brute force спрямовані на підбір облікових даних користувачів або ключів доступу до IoT-пристроїв. У разі успішної реалізації таких атак зломисник може отримати повний контроль над системою моніторингу.

Для запобігання багаторазовим спробам підбору токенів у системі використовується лічильник невдалих спроб автентифікації. Для кожного токена відстежується кількість некоректних запитів, після перевищення якої доступ автоматично обмежується. Додатково, у системі реалізовано глобальний механізм реагування на підозрілу активність. У разі великої кількості некоректних запитів у короткий проміжок часу активується тимчасовий режим обмеження доступу для всієї системи.

Основні параметри реалізації:

- ліміт невдалих спроб для одного токена: 3 послідовні некоректні запити;
- реакція на перевищення ліміту: тимчасове блокування обробки запитів з відповідним токеном;
- тривалість блокування: до завершення поточного сеансу або до зниження рівня підозрілої активності.

Такий підхід унеможливорює ефективний підбір токенів шляхом автоматизованого перебору.

Висновки до другого розділу

У другому розділі було розглянуто питання проєктування та забезпечення кібербезпеки системи моніторингу IoT-мережі на базі мікроконтролерної платформи ESP32. Основну увагу приділено формуванню вимог до системи, розробці її архітектури, інтеграції апаратних компонентів, реалізації програмного забезпечення та впровадженню механізмів кіберзахисту, адаптованих до особливостей IoT-середовища.

В межах розділу було визначено функціональні та нефункціональні вимоги, запропоновано багаторівневу архітектуру захищеної системи моніторингу, а також розглянуто особливості інтеграції сенсорів, виконавчих пристроїв та принципи їх безпечної взаємодії в IoT-середовищі.

Значну увагу приділено розробці програмного забезпечення для ESP32 з урахуванням вимог кібербезпеки. Проаналізовано використання механізмів secure boot, безпечного віддаленого оновлення програмного забезпечення та захисту криптографічних ключів, що дозволяє забезпечити цілісність і автентичність програмного коду IoT-пристроїв.

У розділі також детально розглянуто питання автентифікації та авторизації користувачів і пристроїв, криптографічного захисту даних під час передачі та зберігання, захисту серверної частини і вебінтерфейсу системи та виявленню та протидії основним кіберзагрозам у IoT-середовищі, зокрема атакам типу DoS/DDoS, Man-in-the-Middle та brute force.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ СИСТЕМИ МОНІТОРИНГУ IoT ТА АНАЛІЗ ОТРИМАНИХ РЕЗУЛЬТАТІВ

3.1. Реалізація системи моніторингу IoT мережі

Реалізація системи моніторингу IoT-мережі виконувалася з урахуванням раніше сформованих функціональних і нефункціональних вимог, а також вимог до кібербезпеки. Основною метою даного етапу було створення працездатного прототипу системи, який поєднує апаратні компоненти, програмне забезпечення мікроконтролера ESP32 та клієнтський вебінтерфейс із вбудованими механізмами захисту.

Система реалізована у вигляді багаторівневої архітектури, що включає рівень сенсорів, рівень IoT-пристрою на базі ESP32 та рівень взаємодії з користувачем через вебінтерфейс. Такий підхід дозволяє чітко розмежувати функції між компонентами системи та спростити подальше тестування й аналіз її роботи.

Апаратна реалізація системи моніторингу IoT-мережі виконувалася у симуляційному середовищі Wokwi, яке дозволяє моделювати роботу мікроконтролерних платформ (Рис. 3.1.), зокрема ESP32 (Рис. 3.2.), разом із підключеними периферійними пристроями. Використання даного середовища дало змогу відтворити схему підключення сенсорів, виконавчих пристроїв і мережеву взаємодію без необхідності використання фізичного обладнання.

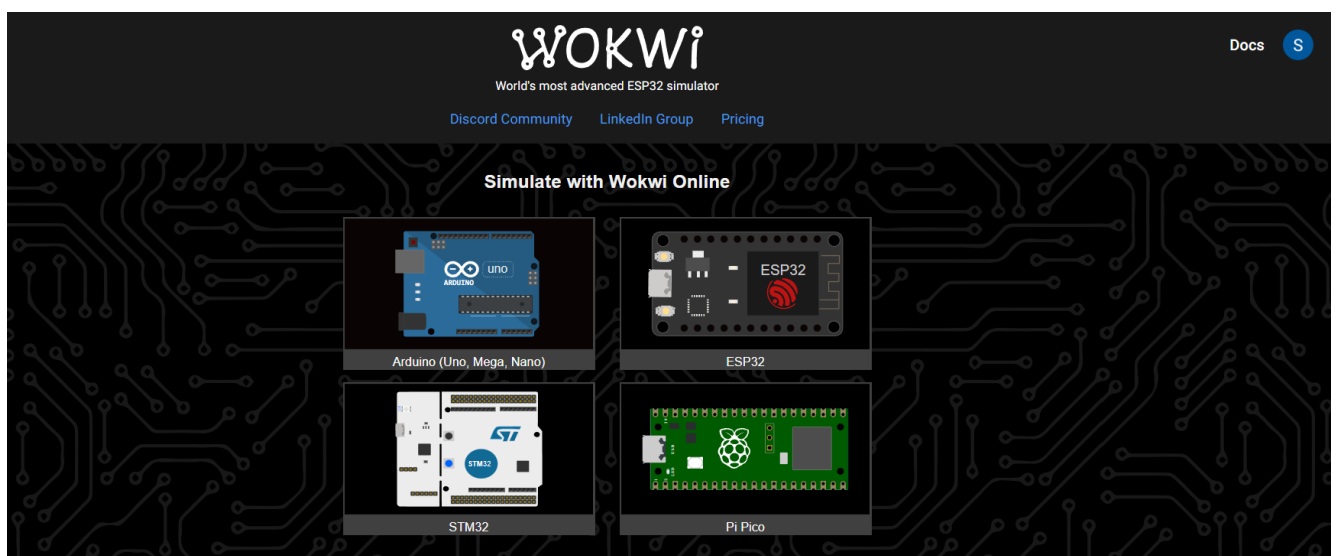


Рисунок 3.1 – Різноманітність плат у Wokwi

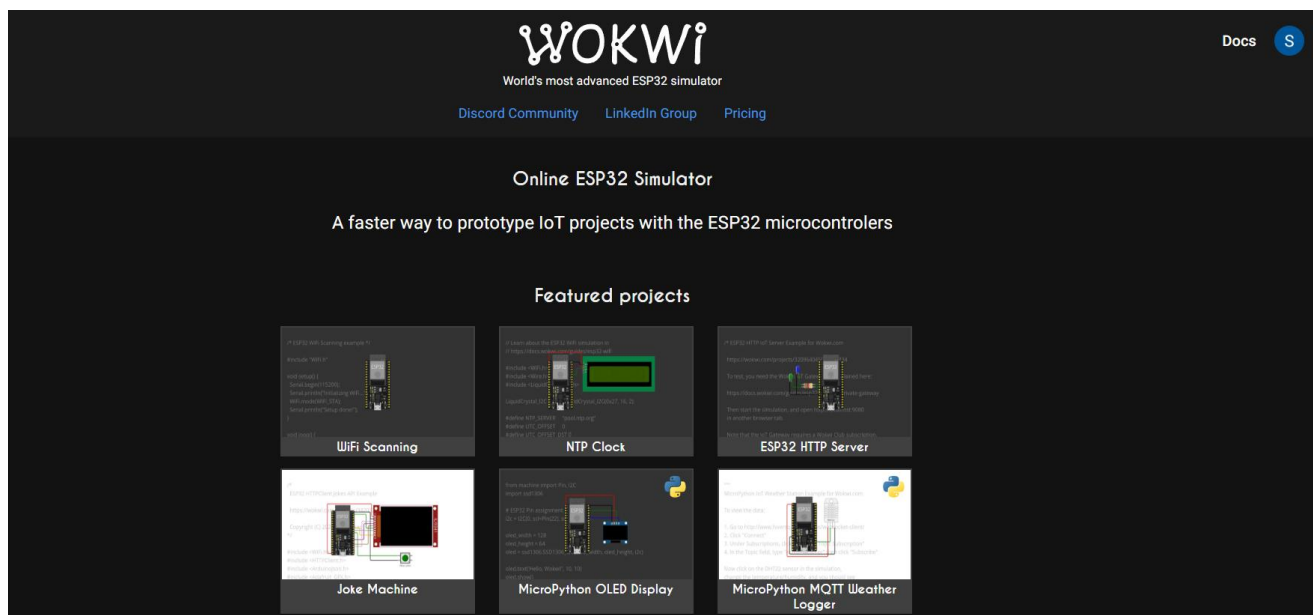


Рисунок 3.2 – Вибір доступних плат ESP32 у Wokwi

Платформа Wokwi забезпечує коректну емуляцію роботи GPIO-пінів, таймерів, переривань і мережевих інтерфейсів, що дозволяє проводити тестування логіки системи, механізмів безпеки та обробки подій у наближених до реальних умовах. Це зробило можливим проведення експериментів з різними сценаріями навантаження та атаками без ризику пошкодження апаратних компонентів.

Як апаратну основу системи використано мікроконтролерну платформу ESP32 (DOIT ESP32 DevKit v1). Вибір даної платформи обумовлений її високою продуктивністю, наявністю вбудованих бездротових інтерфейсів Wi-Fi, а також апаратною підтримкою криптографічних алгоритмів і механізмів безпеки.

До ESP32 підключено такі периферійні компоненти:

- датчик температури та вологості DHT22, який використовується для моніторингу параметрів навколишнього середовища;
- датчик руху PIR, призначений для фіксації наявності руху в зоні спостереження;
- виконавчий пристрій (реле), який використовується для демонстрації функцій керування.

Підключення сенсорів здійснюється через стандартні GPIO-піни мікроконтролера, що забезпечує простоту апаратної конфігурації та сумісність із широким

спектром пристроїв. ESP32 виконує роль центрального вузла, який здійснює зчитування показників сенсорів, їх попередню обробку та передачу результатів у захищеному вигляді.

Програмне забезпечення системи моніторингу реалізовано у вигляді прошивки для мікроконтролера ESP32 з використанням мови програмування C/C++ та Arduino Framework. Обрана платформа забезпечує доступ до мережеских бібліотек, засобів роботи з периферійними пристроями та криптографічних примітивів, що є необхідними для побудови захищеної IoT-системи.

Програмна логіка мікроконтролера умовно поділяється на декілька функціональних модулів: ініціалізація системи, робота з мережею, обробка сенсорів, реалізація вебсервера, автентифікація та захист від атак.

Ініціалізація системи та апаратних компонентів

На етапі запуску виконується ініціалізація мікроконтролера, мережевого з'єднання та периферійних пристроїв. Це забезпечує коректну підготовку системи до подальшої роботи.

Лістинг:

```
void setup() {  
  Serial.begin(115200);  
  pinMode(LED_PIN, OUTPUT);  
  pinMode(PIR_PIN, INPUT);  
  dht.begin();  
  setupWiFi();  
  setupWebServer();  
}
```

Даний фрагмент коду відповідає за запуск послідовного порту для налагодження, ініціалізацію GPIO-пінів, підключення датчика температури та вологості та підготовку мережевого з'єднання та вебсервера.

Підключення до мережі Wi-Fi

ESP32 підключається до бездротової мережі для забезпечення взаємодії з клієнтською частиною системи.

Лістинг:

```
void setupWiFi() {  
  WiFi.begin("Wokwi-GUEST");
```

```
while (WiFi.status() != WL_CONNECTED) {  
    delay(500); }}
```

Реалізація вбудованого HTTP-вебсервера

Для взаємодії з користувачем використовується вбудований вебсервер ESP32, який обробляє HTTP-запити до API.

Лістинг:

```
WebServer server(80);  
void setupWebServer() {  
    server.on("/api/status", HTTP_GET, handleStatus);  
    server.on("/api/control", HTTP_POST, handleControl);  
    server.begin();  
}
```

Кожен маршрут відповідає за окрему функцію системи, наприклад отримання стану сенсорів або керування виконавчими пристроями.

Автентифікація та перевірка цілісності запитів

Для забезпечення безпеки кожен запит до API проходить перевірку автентичності та цілісності. Використовується механізм HMAC-SHA256 із секретним ключем.

Лістинг:

```
bool verifyHMAC(String data, String receivedHmac) {  
    String calculated = hmacSHA256(secretKey, data);  
    return calculated == receivedHmac;  
}
```

Перед обробкою запиту ESP32 перевіряє наявність токена доступу, коректність HMAC-підпису та значення лічильника запитів. У разі невідповідності запит автоматично відхиляється.

Захист від replay-атак

Для запобігання повторному використанню перехоплених запитів використовується монотонно зростаючий лічильник. Цей механізм гарантує, що кожен запит може бути оброблений лише один раз.

Лістинг:

```
bool checkCounter(uint32_t counter) {
    if (counter <= lastCounter) {
        return false;
    }
    lastCounter = counter;
    return true;}

```

Захист від DoS/DDoS-атак

Для запобігання перевантаженню мікроконтролера реалізовано обмеження кількості запитів у часовому вікні. У разі перевищення ліміту подальші запити відхиляються без виконання прикладної логіки.

Лістинг:

```
bool rateLimitExceeded() {
    if (millis() - windowStart > 1000) {
        windowStart = millis();
        requestCount = 0;
    }
    requestCount++;
    return requestCount > 5;
}

```

Захист від brute force-атак

Система відстежує кількість невдалих спроб автентифікації для кожного токена. Після перевищення допустимого порогу доступ для відповідного токена тимчасово обмежується.

Лістинг:

```
void registerFailedAttempt(String token) {
    failedAttempts[token]++;
}

```

Таким чином, програмне забезпечення системи реалізує повний цикл обробки запитів - від прийому та перевірки безпеки до виконання прикладної логіки. Кожен функціональний модуль відповідає за окремий аспект роботи системи, що забезпечує модульність, зручність тестування та відповідність принципам *secure by design*.

Фрагменти коду наведені у тексті для ілюстрації ключових механізмів, тоді як повний лістинг програмного забезпечення наведено у Додатку А.

3.2. Тестування системи моніторингу IoT мережі

Після запуску системи відбувається ініціалізація апаратних та програмних компонентів IoT-пристрою на базі ESP32. На початковому етапі мікроконтролер налаштовує мережеве з'єднання, ініціалізує сенсори температури та вологості, датчик руху, а також виконавчі пристрої. Паралельно запускається вбудований веб-сервер, який забезпечує взаємодію з клієнтською частиною системи.

Користувач отримує доступ до системи через вебінтерфейс, який відкривається у браузері (Рис. 3.2.). Після завантаження інтерфейсу користувач вводить токен доступу, що ініціює процес автентифікації. У разі успішної перевірки токена та криптографічного підпису користувач отримує доступ до функціоналу системи відповідно до своєї ролі (Рис. 3.3. - Рис. 3.5.).

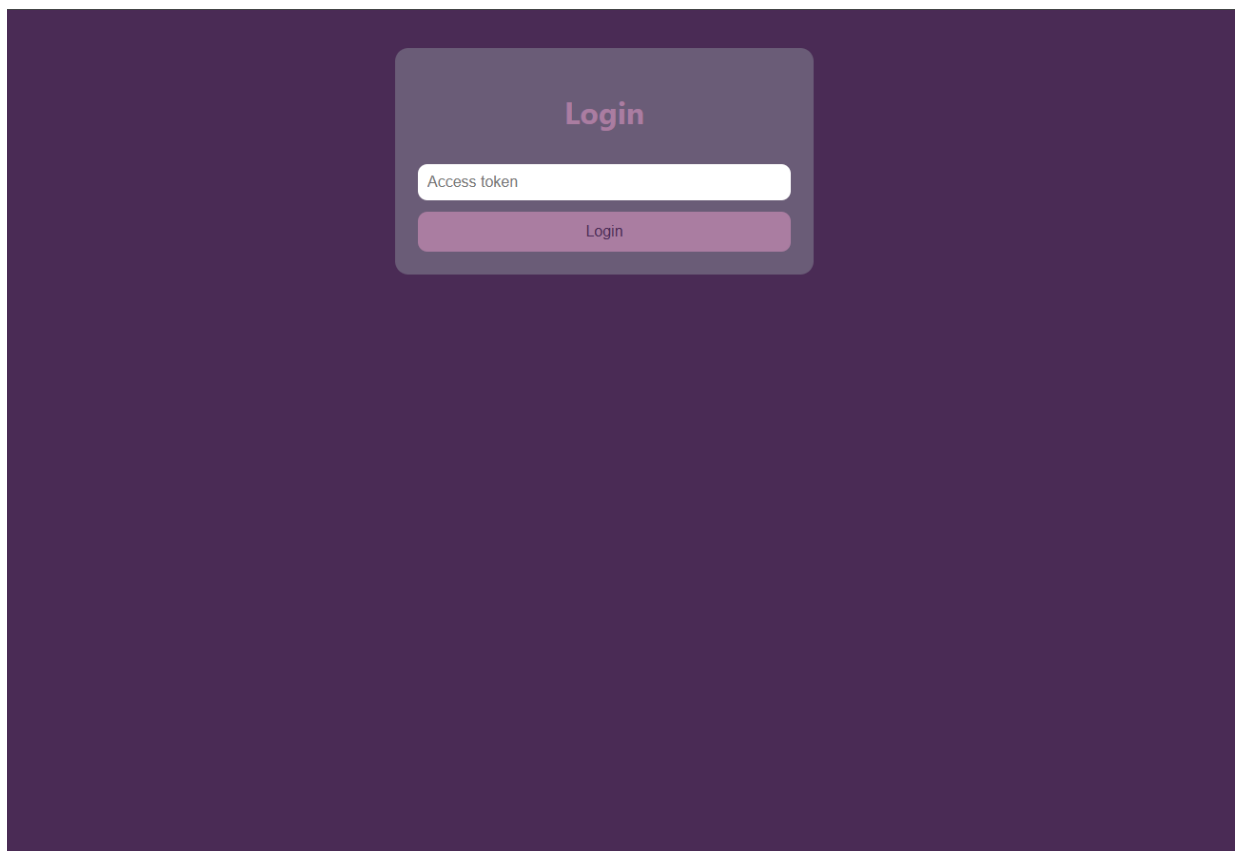


Рисунок 3.2 – Вікно логіну

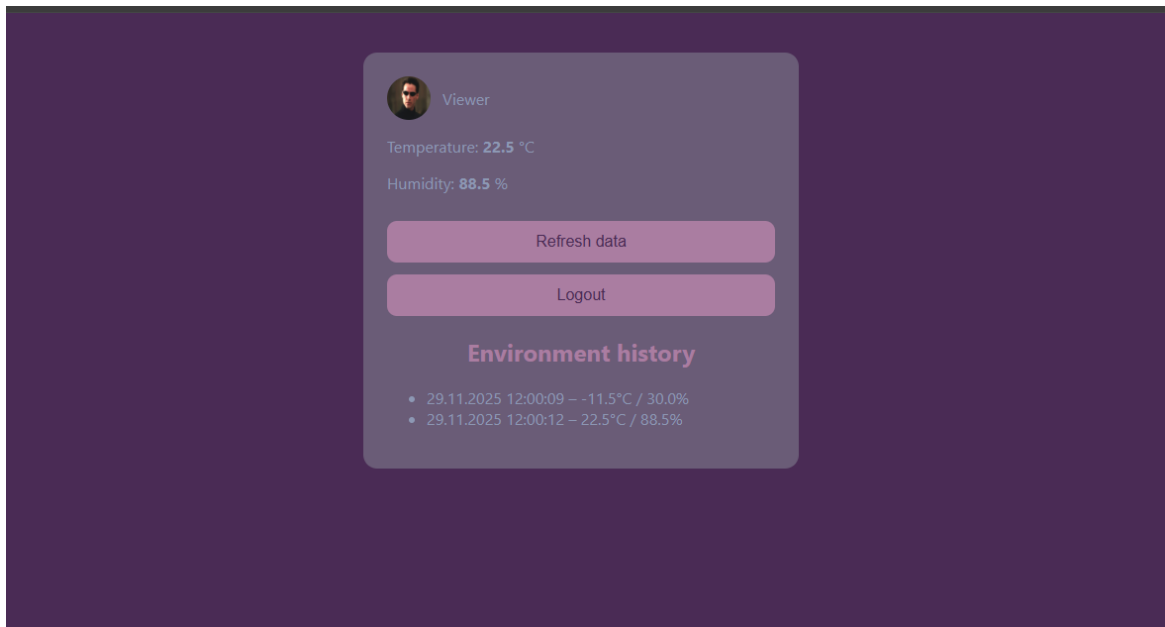


Рисунок 3.3 – Користувач Viewer та його доступ

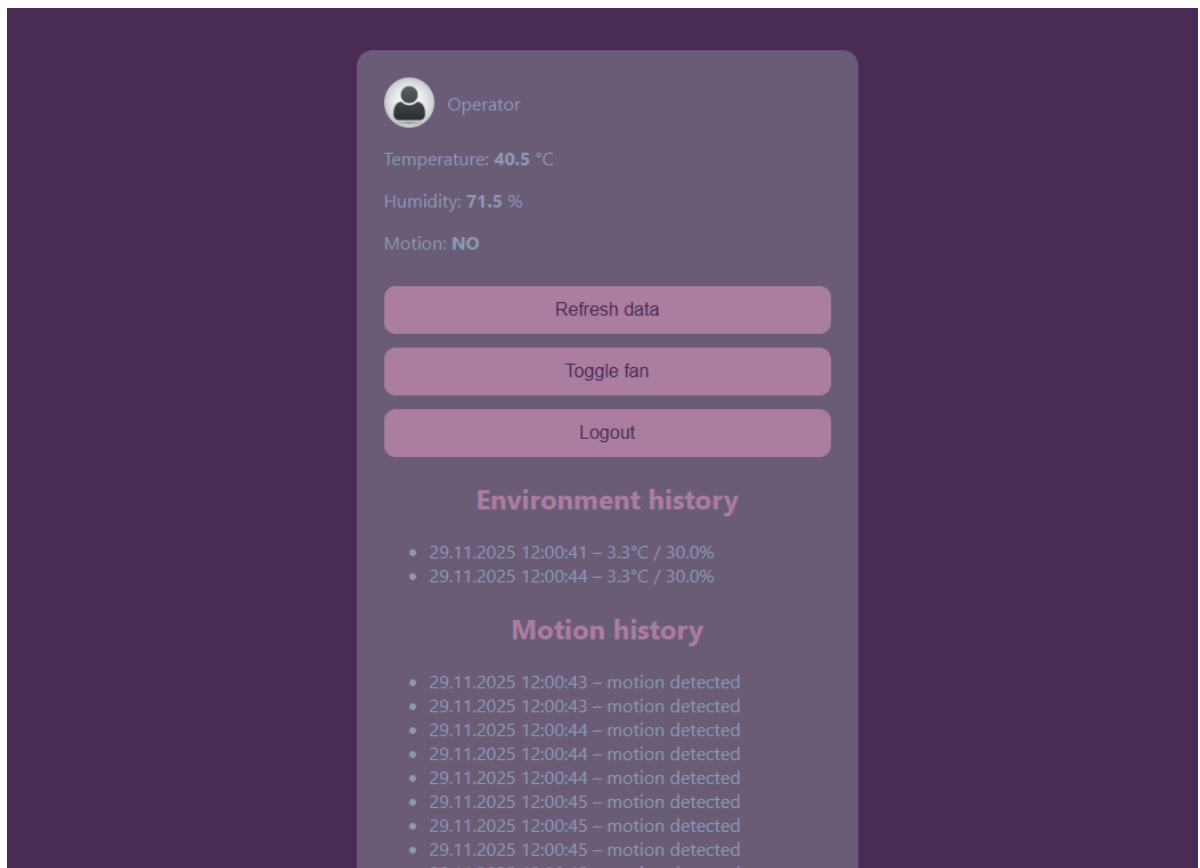


Рисунок 3.4 – Користувач Operator та його доступ

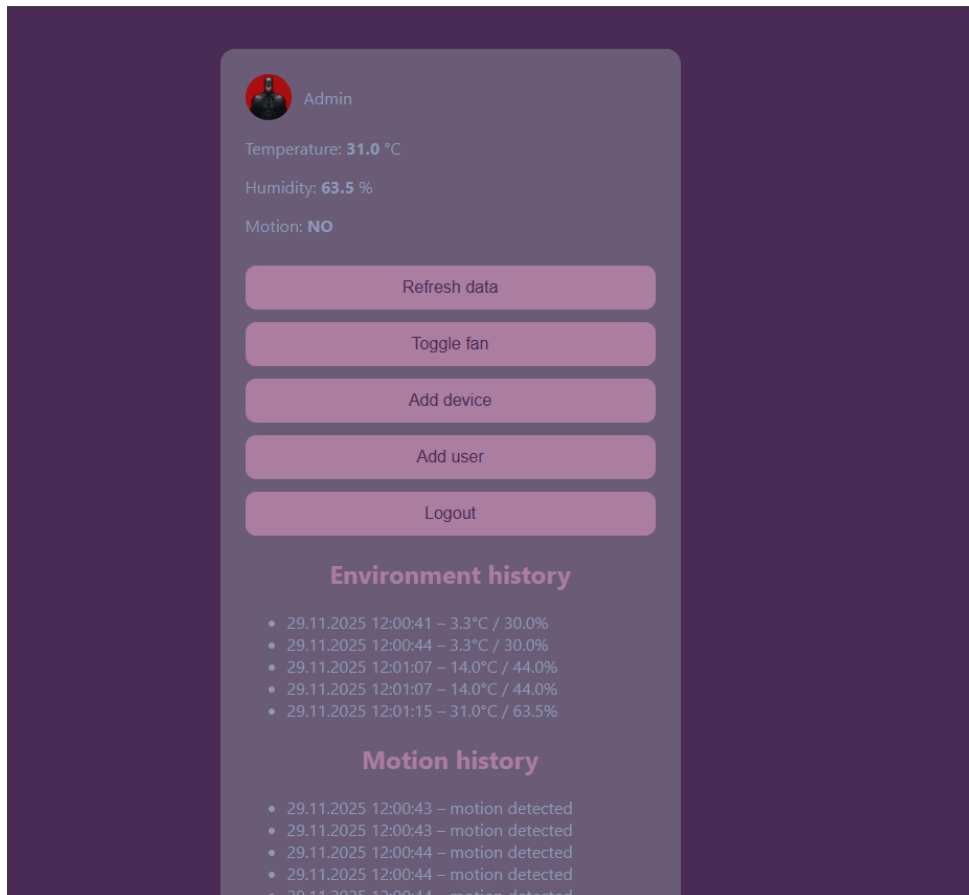


Рисунок 3.5 – Користувач Admin та його доступ

Якщо вводиться неіснуючий або пустий токен - доступ відсутній та відображається вікно з відповідним повідомленням (Рис. 3.6. – Рис. 3.7.).

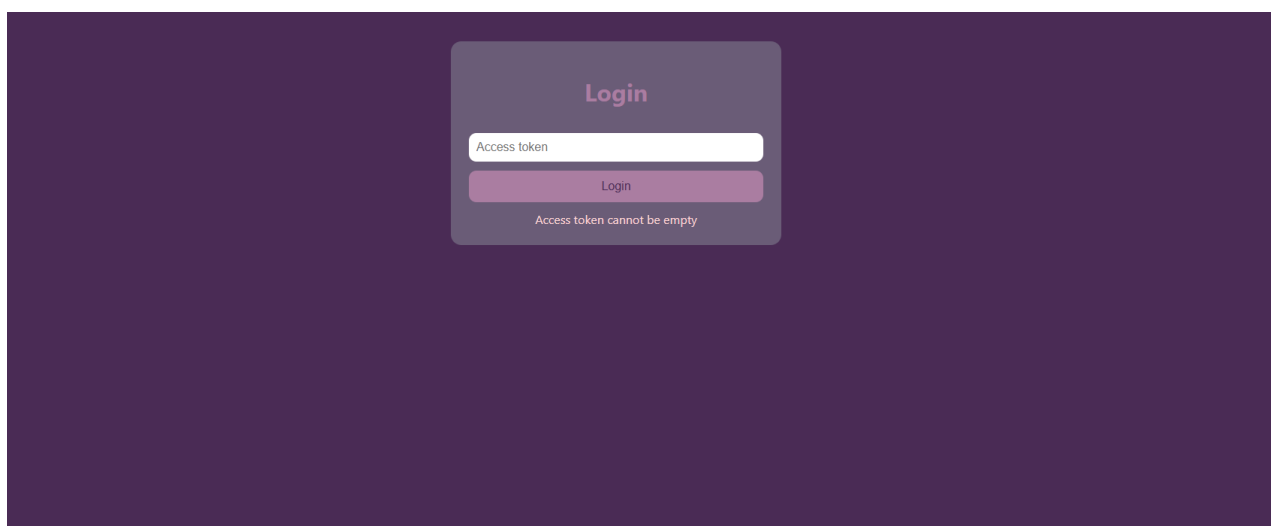


Рисунок 3.6 – Спроба логіну з пустим значенням токена

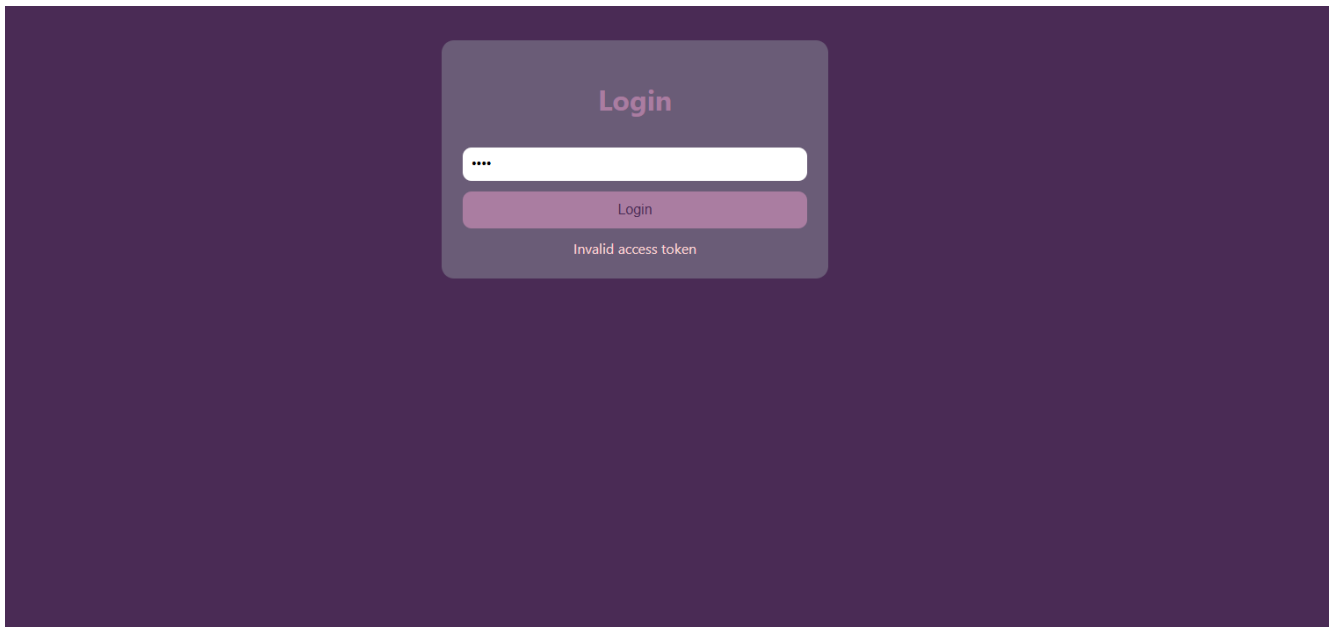


Рисунок 3.7 – Спроба логіну з невірним токеном

Після автентифікації вебінтерфейс відображає актуальні дані з сенсорів, зокрема температуру, вологість та стан датчика руху. Оновлення інформації виконується за запитом користувача, що дозволяє контролювати навантаження на систему під час тестування (див. Рис. 3.3. - Рис. 3.5.).

Було виконано серію контрольованих експериментів, спрямованих на перевірку стійкості системи до типових загроз, зокрема DoS/DDoS (Рис. 3.8.), MITM (Рис. 3.9.) та brute force-атак (Рис. 3.10.).

```
> async function ddosTest() {  
  for (let i = 0; i < 20; i++) {  
    const r = await fetch('/status?token=ADMIN789_SECURE');  
    console.log(i, r.status);  
    await new Promise(res => setTimeout(res, 300));  
  }  
}  
ddosTest();  
< ▶ Promise {<pending>}  
  
0 200 VM93:4  
1 200 VM93:4  
2 200 VM93:4  
3 200 VM93:4  
4 200 VM93:4
```

Рисунок 3.8 – Перевірка Rate Limit (перевірка від DoS/DDoS атак)

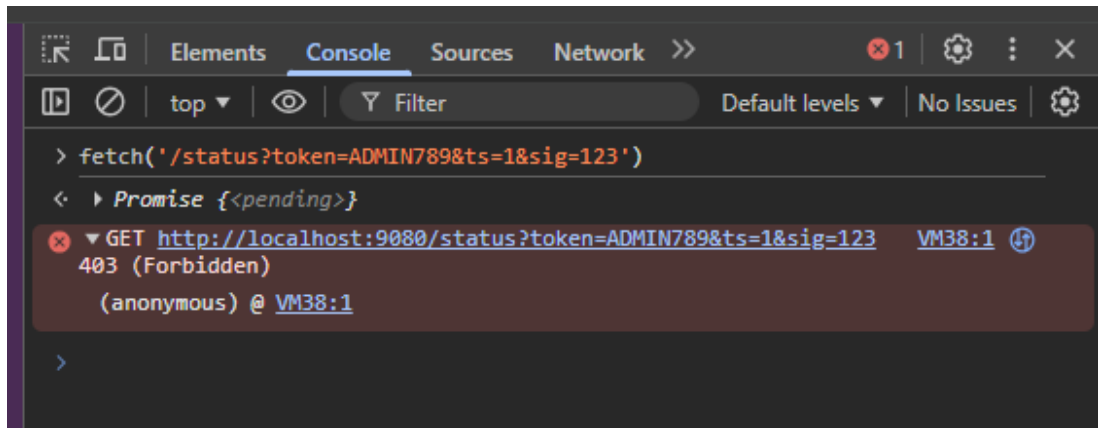


Рисунок 3.9 – Перевірка захисту від MITM

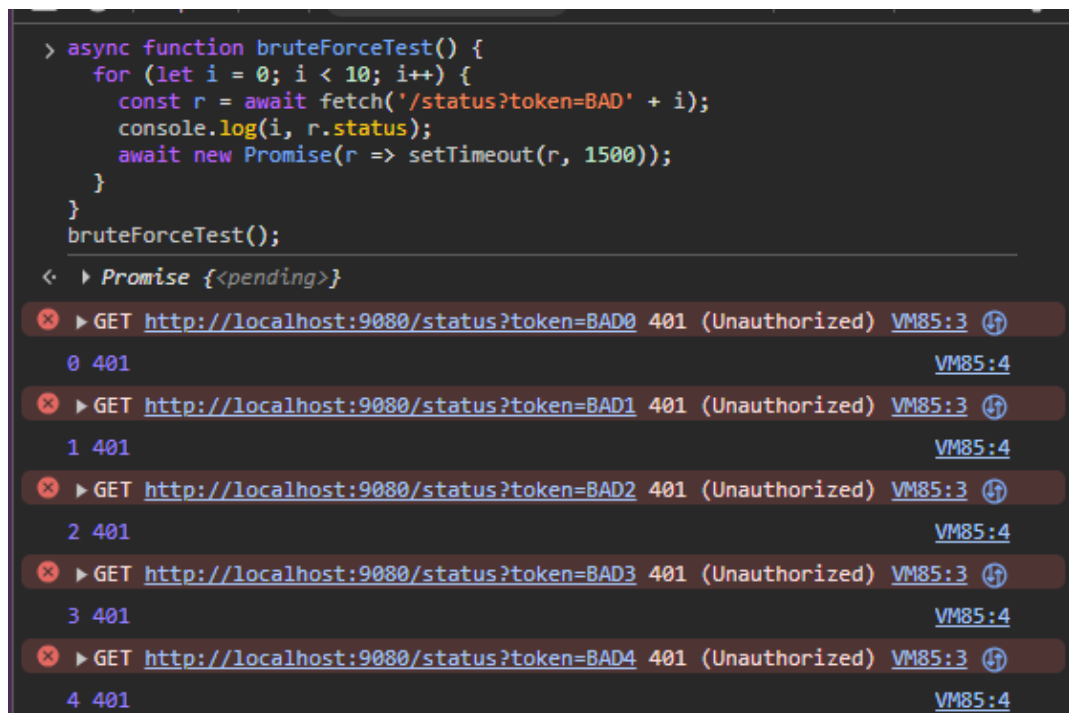


Рисунок 3.10 – Перевірка захисту від Brute Force

Для кожного сценарію перевірялося, чи відбувається коректне відхилення не-санкціонованих запитів, а також чи зберігається стабільність роботи системи для легітимних користувачів. Додатково, було перевірено працездатність системи під час використання кнопки Refresh багато разів - система все ще працює коректно і без перебоїв.

У розробленій системі реалізовано механізм симетричного шифрування з використанням алгоритму AES-256 у режимі CBC. Даний алгоритм обрано з огляду

на його високий рівень криптографічної стійкості та оптимальну продуктивність для ресурсно-обмежених мікроконтролерних платформ, зокрема ESP32.

Під час формування захищеної відповіді система генерує унікальний ініціалізаційний вектор, який використовується у процесі шифрування. IV забезпечує криптографічну випадковість та унеможливорює повторюваність зашифрованих даних навіть у разі передачі ідентичних повідомлень. Згенерований IV передається разом із зашифрованими даними та не містить конфіденційної інформації.

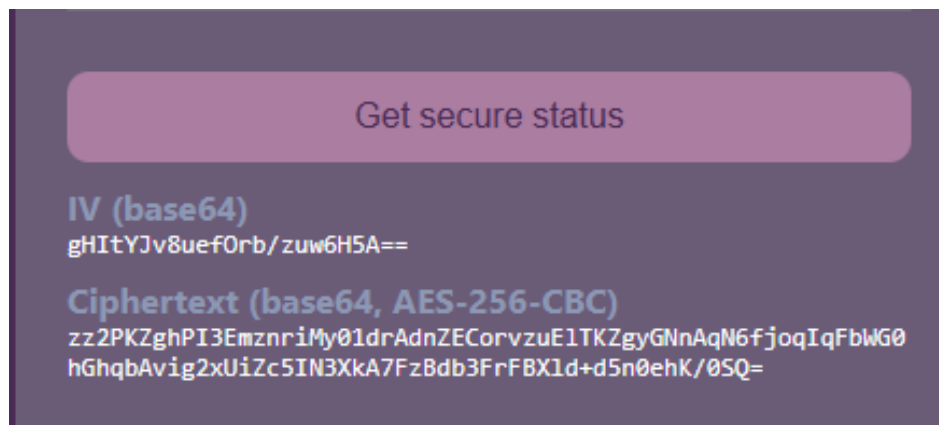


Рисунок 3.11 – Перевірка шифрування

Дані стану системи, які можуть містити чутливу інформацію про роботу IoT-пристроїв, шифруються із застосуванням алгоритму AES-256-CBC з використанням спільного секретного ключа (Рис. 3.11.). Результатом цього процесу є зашифрований текст, який передається клієнтській частині у форматі base64. Таке представлення використовується для зручності обробки та передачі бінарних даних у межах HTTP-взаємодії. Проведене тестування показало, що у процесі обміну даними між клієнтом і ESP32 жодна чутлива інформація не передається у відкритому вигляді.

3.3. Перспективні технології кіберзахисту IoT-систем

Подальший розвиток IoT-систем та зростання кількості підключених пристроїв зумовлюють необхідність впровадження нових підходів до забезпечення кібербезпеки. Традиційні методи захисту, орієнтовані на периметрову безпеку, стають недостатньо ефективними в умовах розподілених та динамічних IoT-

середовищ. У зв'язку з цим значну увагу приділяють перспективним технологіям кіберзахисту, серед яких ключовими є концепція Zero Trust, застосування методів машинного навчання та використання блокчейн-технологій.

Zero Trust є сучасною моделлю безпеки, основним принципом якої є відмова від довіри за замовчуванням. У межах цієї концепції кожен користувач, пристрій або сервіс повинен проходити перевірку незалежно від його розташування в мережі. Для IoT-систем такий підхід є особливо актуальним через велику кількість розподілених і часто фізично незахищених пристроїв [24].

Застосування Zero Trust у IoT передбачає постійну автентифікацію пристроїв, мінімізацію прав доступу та сегментацію мережі. Це дозволяє обмежити можливості зловмисника навіть у разі компрометації окремого компонента системи. Реалізація Zero Trust у системах моніторингу сприяє підвищенню загальної кіберстійкості та зниженню ризику масштабних атак [25].

Машинне навчання є перспективним напрямом у сфері кібербезпеки, оскільки дозволяє автоматизувати процеси виявлення загроз та адаптуватися до нових типів атак. У IoT-системах методи машинного навчання можуть використовуватися для аналізу мережевого трафіку, поведінки пристроїв та виявлення аномалій, що свідчать про потенційні атаки.

Алгоритми машинного навчання здатні виявляти нетипові патерни поведінки, які важко зафіксувати традиційними правилами. Це дозволяє підвищити ефективність систем виявлення вторгнень та зменшити кількість хибних спрацьовувань. Використання машинного навчання у поєднанні з класичними методами захисту створює передумови для побудови адаптивних та самонавчальних систем кібербезпеки [26].

Блокчейн-технології розглядаються як перспективний інструмент для підвищення рівня довіри та прозорості в IoT-середовищі. Децентралізована природа блокчейну дозволяє зменшити залежність від центральних вузлів управління та знизити ризик компрометації даних. У контексті IoT блокчейн може використовуватися для безпечного зберігання журналів подій, управління ідентифікацією

пристроїв та контролю доступу. Завдяки незмінності записів у блокчейні забезпечується цілісність даних, що є особливо важливим для систем моніторингу. Водночас впровадження блокчейну в IoT-системи потребує врахування обмежень, пов'язаних із продуктивністю та енергоспоживанням пристроїв [27].

Висновки до третього розділу

Система стабільно ініціалізується після запуску, коректно підключається до мережі та забезпечує безперервну взаємодію з користувачем через вебінтерфейс.

Результати тестування показали, що зчитування та обробка даних із сенсорів температури, вологості та руху виконуються без суттєвих затримок, а відображення інформації у вебінтерфейсі відповідає фактичному стану середовища. Керування виконавчими пристроями здійснюється лише після успішного проходження автентифікації та авторизації.

Проведені експерименти засвідчили ефективність реалізованих заходів захисту від DoS/DDoS, brute force та MITM-атак. Несанкціоновані або шкідливі запити своєчасно відхилялися, при цьому стабільність роботи системи для легітимних користувачів зберігалася.

Загалом результати тестування підтверджують, що розроблена система відповідає поставленим вимогам, демонструє стійкість до типових загроз для IoT-середовищ та може бути використана як основа для побудови безпечних систем моніторингу з урахуванням обмежених ресурсів мікроконтролерних платформ.

ВИСНОВОК

У межах даної роботи було досягнуто поставленої мети - розроблено та реалізовано захищену систему моніторингу IoT-мережі на базі мікроконтролерної платформи ESP32 з урахуванням сучасних вимог до кібербезпеки та обмежених обчислювальних ресурсів IoT-пристроїв.

Було проаналізовано основні загрози безпеці, характерні для IoT-систем, зокрема атаки типу DoS/DDoS, brute force та “людина посередині”. На основі проведеного аналізу сформульовано функціональні та нефункціональні вимоги до системи, що дозволило обґрунтувати вибір архітектурних рішень і механізмів захисту.

Реалізовано апаратно-програмний прототип системи моніторингу, який забезпечує зчитування параметрів навколишнього середовища, передачу даних та керування виконавчими пристроями. Програмна реалізація базується на використанні вбудованого вебсервера ESP32 та клієнтського вебінтерфейсу, що дозволяє здійснювати взаємодію з системою у режимі реального часу.

У системі впроваджено автентифікацію та авторизацію користувачів на основі токенів доступу, криптографічний контроль цілісності даних із використанням HMAC-SHA256, а також захист від повторного використання запитів за допомогою монотонно зростаючого лічильника. Для забезпечення конфіденційності даних реалізовано симетричне шифрування з використанням алгоритму AES-256.

З метою підвищення стійкості системи до мережових атак реалізовано механізми обмеження частоти HTTP-запитів, ранньої фільтрації некоректного трафіку та підрахунку невдалих спроб автентифікації. Зазначені механізми дозволяють ефективно протидіяти атакам типу DoS/DDoS і brute force без істотного впливу на продуктивність системи.

Проведене тестування у симуляційному середовищі підтвердило працездатність розробленої системи, коректність реалізованих механізмів безпеки та її стабільну роботу за умов нормального та підвищеного навантаження. Результати свідчать про можливість практичного використання оглянутих рішень у IoT-мереж.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Considerations for Managing Internet of Things (IoT) Cybersecurity and Privacy Risks. URL: <https://nvlpubs.nist.gov/nistpubs/ir/2019/NIST.IR.8228.pdf>.
2. What is IoT: A Connected World Explained. URL: <https://relevant.software/blog/what-is-iot/>.
3. What is IoT security?. URL: <https://www.servicenow.com/fr-ca/products/security-operations/what-is-iot-security.html>.
4. How to Ensure Cybersecurity in the Age of IoT. URL: <https://appinventiv.com/blog/how-to-ensure-cybersecurity-in-iot/>.
5. What is the Internet of Things (IoT)? Definition and Critical Risks. URL: <https://www.upguard.com/blog/internet-of-things-iot>.
6. What is IoT Malware?. URL: <https://cyberpedia.reasonlabs.com/EN/iot%20malware.html>.
7. 25 Common Types of Malware & How To Identify Them. URL: <https://securityscorecard.com/blog/common-types-of-malware/>.
8. The Creeper virus: The beginning of malware. URL: <https://moxso.com/blog/glossary/creeper-virus>.
9. The Morris Worm. URL: <https://limn.press/article/the-morris-worm/>.
10. Trojans. URL: <https://www.malwarebytes.com/blog/threats/trojans>.
11. What is the Mirai?. URL: <https://www.corero.com/what-is-the-mirai-botnet/>.
12. What is a man-in-the-middle attack?. URL: <https://www.ibm.com/think/topics/man-in-the-middle>.
13. IoT and Man-in-the-Middle Attacks. URL: <https://arxiv.org/pdf/2308.02479>.
14. What is a brute force attack?. URL: <https://www.ibm.com/think/topics/brute-force-attack>.
15. Brute Force Attacks on IoT. URL: <https://www.allot.com/blog/brute-force-attacks-iot/>.
16. ЗАХИСТ В ТЕХНОЛОГІЯХ IOT. URL: https://csecurity.kubg.edu.ua/index.php/journal/article/view/705?utm_source=chatgpt.com.

17. Microsoft Defender for IoT. URL: <https://learn.microsoft.com/en-us/azure/defender-for-iot/>.
18. Security Overview. URL: <https://docs.espressif.com/projects/espidf/en/latest/esp32/security/security.html>.
19. Essential Strategies for Protection. URL: <https://deviceauthority.com/mitigating-iot-denial-of-service-attacks-essential-strategies-for-protection/>.
20. Impact of MITM attacks on IoT. URL: <https://archive.interconf.center/index.php/conference-proceeding/article/view/4846>.
21. Brute force attacks. URL: <https://www.okta.com/en-sg/identity-101/brute-force/>.
22. Understanding ESP32's Security Features. URL: <https://medium.com/the-esp-journal/understanding-esp32s-security-features-14483e465724>.
23. Advanced ESP32 simulator. URL: https://wokwi.com/?utm_source=chatgpt.com.
24. What is zero trust?. URL: <https://www.ibm.com/think/topics/zero-trust>.
25. Securing IoT Devices with Zero Trust URL: <https://medium.com/@okanyildiz1994/securing-iot-devices-with-zero-trust-architecture-an-exhaustive-guide-c77ab3173640>.
26. Demystifying Data Science and Machine Learning. URL: <https://www.iotforall.com/demystifying-data-science-and-machine-learning-in-iot>.
27. Blockchain in IoT. URL: <https://tektelic.com/what-it-is/blockchain-in-iot/>.

ДОДАТКИ

Лістинг програми

```

#include <WiFi.h>
#include <WebServer.h>
#include <DHT.h>
#include "mbedtls/md.h"
/* ===== HW
===== */
#define DHT_PIN 15
#define DHTTYPE DHT22
#define FAN_PIN 26 // your "fan" =
LED/relay control pin
#define PIR_PIN 27 // optional: motion
sensor (PIR). If you don't have PIR, leave it
unconnected; it will read LOW.

DHT dht(DHT_PIN, DHTTYPE);
/* ===== WIFI
===== */
const char* ssid = "Wokwi-GUEST";
WebServer server(80);
/* ===== USERS / ROLES
===== */
struct User {
    const char* token;
    const char* role; // viewer/operator/admin
};
User users[] = {
    {"VIEW123_SECURE", "viewer"},
    {"OPER456_SECURE", "operator"},
    {"ADMIN789_SECURE", "admin"}
};

/* ===== SECURITY:
MITM (HMAC + anti-replay)
===== */
/*
    We sign each request with:
    sig = HMAC_SHA256( SECRET, token +
    "|" + ctr + "|" + uri )
    and send token, ctr, sig as query params.
    anti-replay: ctr must be strictly increasing per
    token.
*/
const char* HMAC_SECRET =
"demo_secret_change_me_please";
/* per-token replay tracker */
struct ReplayState {
    String token;
    uint32_t lastCtr;
};
ReplayState replayStates[10];
/* ===== SECURITY:
Brute force / auth lockout
===== */
/*
    This lockout is aimed at brute force of tokens
    against the API.
    Because attacker may randomize tokens, we
    do "global invalid attempts" window too.
*/
#define MAX_FAILS_PER_TOKEN 5
#define TOKEN_LOCK_MS 60000UL

struct AuthState {
    String token;
    uint8_t fails;
    unsigned long lockUntil;
};
AuthState authStates[10];
/* global invalid attempts protection (anti
brute-force scanning / low-grade DoS) */
#define BAD_WINDOW_MS 10000UL
#define BAD_MAX_IN_WINDOW 10
unsigned long badWindowStart = 0;
uint16_t badCountInWindow = 0;
unsigned long globalCooldownUntil = 0;
#define GLOBAL_COOLDOWN_MS
15000UL
/* ===== SECURITY:
DDoS / DoS rate limiting
===== */
/*
    Keep it gentle (ESP32 WebServer is small).
    This prevents overload without dropping TCP.
*/
#define RL_WINDOW_MS 2000UL
#define RL_MAX_REQ_IN_WINDOW 10
unsigned long rlWindowStart = 0;
uint16_t rlCount = 0;

/* ===== APP STATE
===== */
float temperature = 0;
float humidity = 0;
bool motion = false;
bool fanState = false;

```

```

/* logs update ONLY when user presses
Refresh in UI */
#define MAX_ENV_LOG 60
String envLog[MAX_ENV_LOG];
int envCount = 0;
#define MAX_MOTION_LOG 60
String motionLog[MAX_MOTION_LOG];
int motionCount = 0;

/* pseudo datetime (fixed date, moving time) */
unsigned long startMillis;
String nowDateTime() {
    unsigned long t = (millis() - startMillis) /
1000;
    char buf[32];
    sprintf(buf,
"29.11.2025 %02lu:%02lu:%02lu",
        (12 + (t / 3600)) % 24,
        (t / 60) % 60,
        t % 60);
    return String(buf);
}
/* ===== UTIL: HMAC-
SHA256 hex ===== */
String hmacSha256Hex(const String& msg) {
    byte out[32];
    mbedtls_md_context_t ctx;
    mbedtls_md_init(&ctx);
    mbedtls_md_setup(&ctx,
mbedtls_md_info_from_type(MBEDTLS_MD
_SHA256), 1);

    mbedtls_md_hmac_starts(&ctx, (const
unsigned char*)HMAC_SECRET,
strlen(HMAC_SECRET));
    mbedtls_md_hmac_update(&ctx, (const
unsigned char*)msg.c_str(), msg.length());
    mbedtls_md_hmac_finish(&ctx, out);
    mbedtls_md_free(&ctx);

    char hexbuf[65];
    for (int i = 0; i < 32; i++) sprintf(hexbuf + i *
2, "%02x", out[i]);
    hexbuf[64] = 0;
    return String(hexbuf);
}
/* ===== RATE LIMIT
===== */
bool allowRate() {
    unsigned long now = millis();
    if (now - rlWindowStart >
RL_WINDOW_MS) {
        rlWindowStart = now;

        rlCount = 1;
        return true;
    }
    rlCount++;
    if (rlCount >
RL_MAX_REQ_IN_WINDOW) {
        server.send(429, "text/plain", "Too Many
Requests");
        delay(5);
        return false;
    }
    return true;
}

/* ===== Auth state
helpers ===== */
AuthState* getAuthState(const String& token)
{
    for (auto &a : authStates) {
        if (a.token == token) return &a;
        if (a.token == "") {
            a.token = token;
            a.fails = 0;
            a.lockUntil = 0;
            return &a;
        }
    }
    return nullptr;
}
ReplayState* getReplayState(const String&
token) {
    for (auto &r : replayStates) {
        if (r.token == token) return &r;
        if (r.token == "") {
            r.token = token;
            r.lastCtr = 0;
            return &r;
        }
    }
    return nullptr;
}
/* ===== SECURITY:
verify signature + anti-replay
===== */
bool verifyMitm(User*& uOut, String&
roleOut) {
    if (!server.hasArg("token"))
|| !server.hasArg("ctr") || !server.hasArg("sig"))
{
        server.send(401, "text/plain",
"Unauthorized");
        delay(5);
        return false;
    }
}

```

```

    }
    String token = server.arg("token");
    String ctrS = server.arg("ctr");
    String sig = server.arg("sig");
    String uri = server.uri(); // include path in
signature
    // global cooldown (anti brute-force scanning)
    if (globalCooldownUntil > millis()) {
        server.send(423, "text/plain", "Locked
(cooldown)");
        delay(5);
        return false;
    }
    // DDoS rate limit (gentle)
    if (!allowRate()) return false;

    // find user by token
    User* found = nullptr;
    for (auto &user : users) {
        if (token == user.token) { found = &user;
break; }
    }
    // brute force / invalid token tracking
    if (!found) {
        // count invalid attempts globally in window
        unsigned long now = millis();
        if (now - badWindowStart >
BAD_WINDOW_MS) {
            badWindowStart = now;
            badCountInWindow = 1;
        } else {
            badCountInWindow++;
        }
        if (badCountInWindow >=
BAD_MAX_IN_WINDOW) {
            globalCooldownUntil = millis() +
GLOBAL_COOLDOWN_MS;
            server.send(423, "text/plain", "Locked (too
many invalid attempts)");
            delay(5);
            return false;
        }
        server.send(401, "text/plain",
"Unauthorized");
        delay(5);
        return false;
    }
    // per-token lockout (brute force on known
token)
    AuthState* st = getAuthState(token);
    if (st && st->lockUntil > millis()) {
        server.send(423, "text/plain", "Locked
(token)");

```

```

        delay(5);
        return false;
    }
    // anti-replay ctr check
    uint32_t ctr = (uint32_t)ctrS.toInt();
    ReplayState* rs = getReplayState(token);
    if (rs && ctr <= rs->lastCtr) {
        server.send(403, "text/plain", "Replay
detected");
        delay(5);
        return false;
    }
    // signature check
    String msg = token + "|" + ctrS + "|" + uri;
    String expected = hmacSha256Hex(msg);
    if (!expected.equalsIgnoreCase(sig)) {
        // count as failed auth for that token
        if (st) {
            st->fails++;
            if (st->fails >=
MAX_FAILS_PER_TOKEN) {
                st->lockUntil = millis() +
TOKEN_LOCK_MS;
            }
        }
        server.send(403, "text/plain", "Invalid
signature");
        delay(5);
        return false;
    }
    // success: reset fails, update ctr
    if (st) st->fails = 0;
    if (rs) rs->lastCtr = ctr;
    uOut = found;
    roleOut = String(found->role);
    return true;
}

/* ===== Logging helpers
===== */
void pushEnvLog(float t, float h) {
    String line = nowDateTime() + " — " +
String(t, 1) + "°C / " + String(h, 1) + "%";
    if (envCount < MAX_ENV_LOG) {
        envLog[envCount++] = line;
    } else {
        // shift left
        for (int i = 1; i < MAX_ENV_LOG; i++)
envLog[i - 1] = envLog[i];
        envLog[MAX_ENV_LOG - 1] = line;
    }
}
void pushMotionLog(bool m) {

```

```

    if (!m) return;
    String line = nowDateTime() + " — motion
detected";
    if (motionCount < MAX_MOTION_LOG) {
        motionLog[motionCount++] = line;
    } else {
        for (int i = 1; i < MAX_MOTION_LOG;
i++) motionLog[i - 1] = motionLog[i];
        motionLog[MAX_MOTION_LOG - 1] =
line;
    }
}
/* ===== WEB UI
===== */
String webPage() {
    // NOTE: For demo, HMAC secret is in JS. In
real systems keep secret server-side and use
TLS/pinning.
    return R"rawliteral(
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Secure Smart Home</title>
<style>
*{box-sizing:border-box}
body{
    margin:0;
    font-family:system-ui,Segoe UI,Arial;
    background:#4a2b55;
    color:#8d98b4;
}
.container{max-
width:640px;margin:auto;padding:40px}
.card{
    background:#6a5c77;
    border-radius:18px;
    padding:28px;
}
h2{margin:0 0 12px;color:#aa7da1}
input,button{
    width:100%;
    padding:12px 14px;
    margin-top:12px;
    border-radius:12px;
    border:none;
    font-size:16px;
}
input{background:#f4f2f6}
button{background:#aa7da1;color:#4a2b55;cur-
sor:pointer}
.row{display:flex;gap:12px}
.row > button{flex:1}

```

```

.notice{color:#ffd6d6;margin-top:10px}
.small{font-size:13px;opacity:.9}
.hidden{display:none}
hr{border:none;border-top:1px solid
rgba(255,255,255,.12);margin:18px 0}
ul{margin:8px 0 0;padding-left:18px}
</style>
</head>
<body>
<div class="container">
    <div class="card" id="loginCard">
        <h2>Login</h2>
        <div class="small">Use demo tokens:
VIEW123_SECURE / OPER456_SECURE /
ADMIN789_SECURE</div>
        <input type="password" id="tokenInput"
placeholder="Access token">
        <button
onclick="doLogin()">Login</button>
        <div class="notice" id="err"></div>
    </div>
    <div class="card hidden" id="appCard">
        <h2 id="roleTitle">Dashboard</h2>
        <div class="small">Security: HMAC
integrity + anti-replay, brute force lockout,
DDoS rate limiting.</div>
        <hr>
        <div>Temperature: <b id="t">-
</b> °C</div>
        <div>Humidity: <b id="h">-</b> %</div>
        <div>Motion: <b id="m">-</b></div>
        <div class="row" style="margin-top:12px">
            <button
onclick="refreshAll()">Refresh</button>
            <button id="fanBtn"
onclick="toggleFan()">Toggle fan</button>
        </div>
        <button style="margin-top:12px"
onclick="logout()">Logout</button>

        <hr>
        <h2 style="font-size:18px">Environment
history</h2>
        <ul id="env"></ul>
        <hr>
        <h2 style="font-size:18px">Motion
history</h2>
        <ul id="mot"></ul>
    </div>
</div>
<script>
/* ===== demo secret (for report) ===== */

```

```

const SECRET =
"demo_secret_change_me_please";
/* ===== local session ===== */
let token = localStorage.getItem('token') || "";
let ctr = parseInt(localStorage.getItem('ctr') ||
"0", 10);
/* ===== HMAC SHA-256 in browser =====
*/
function hex(buf){
  return [...new
Uint8Array(buf)].map(b=>b.toString(16).padSt
art(2,'0')).join("");
}
async function hmac(message){
  const enc = new TextEncoder();
  const key = await crypto.subtle.importKey(
    "raw", enc.encode(SECRET),
    { name: "HMAC", hash: "SHA-256" },
    false, ["sign"]
  );
  const sig = await crypto.subtle.sign("HMAC",
key, enc.encode(message));
  return hex(sig);
}
/* ===== signed fetch helper ===== */
async function signedFetch(path){
  ctr += 1;
  localStorage.setItem('ctr', String(ctr));
  const msg = `${token}|${ctr}|${path}`;
  const sig = await hmac(msg);
  const url =
`${path}?token=${encodeURIComponent(toke
n)}&ctr=${ctr}&sig=${sig}`;
  return fetch(url);
}
/* ===== UI ===== */
const loginCard =
document.getElementById('loginCard');
const appCard =
document.getElementById('appCard');
const err =
document.getElementById('err');
async function doLogin(){
  const t =
document.getElementById('tokenInput').value.t
rim();
  if(!t){ err.innerText="Token is empty";
return; }
  token = t;
  localStorage.setItem('token', token);
  // reset counter per login to avoid replay issues
after logout/login
  ctr = 0;

```

```

localStorage.setItem('ctr', "0");
// try status call
const r = await signedFetch('/api/status');
if(r.status === 429){ err.innerText="Too
many requests (rate limit). Try again.";
return; }
if(r.status === 423){ err.innerText="Locked
(cooldown/lockout). Wait a bit and retry.";
return; }
if(!r.ok){
  const tx = await r.text();
  err.innerText = "Login failed: " + tx;
  return;
}
err.innerText = "";
loginCard.classList.add('hidden');
appCard.classList.remove('hidden');
await refreshAll();
}

function logout(){
  localStorage.removeItem('token');
  localStorage.removeItem('ctr');
  location.reload();
}
async function refreshAll(){
  const r = await signedFetch('/api/refresh'); //
refresh also appends logs
  if(r.status === 429){ alert("Too many requests
(DDoS protection)."); return; }
  if(r.status === 423){ alert("Locked. Please
wait and retry."); return; }
  if(!r.ok){ alert(await r.text()); return; }
  const data = await r.json();
  document.getElementById('roleTitle').innerTex
t = data.role.toUpperCase();
  document.getElementById('t').innerText =
data.temperature.toFixed(1);
  document.getElementById('h').innerText =
data.humidity.toFixed(1);
  document.getElementById('m').innerText =
data.motion ? "YES" : "NO";
  // role-based UI
  if(data.role === "viewer"){
document.getElementById('fanBtn').style.displ
ay = "none";
  } else {
document.getElementById('fanBtn').style.displ
ay = "block";
  }
  // logs
  document.getElementById('env').innerHTML
= data.env.map(x => `<li>${x}</li>`).join("");

```

```

    document.getElementById('mot').innerHTML
= data.motionLog.map(x =>
`<li>${x}</li>`).join("");
}
// operator/admin only
async function toggleFan(){
    const r = await signedFetch('/api/fan');
    if(r.status === 429){ alert("Too many requests
(DDoS protection)."); return; }
    if(r.status === 423){ alert("Locked. Please
wait and retry."); return; }
    if(r.status === 403){ alert("Forbidden");
return; }
    if(!r.ok){ alert(await r.text()); return; }
    await refreshAll();
}
// auto open if token exists
if(token){
    (async)=>{
        loginCard.classList.add('hidden');
        appCard.classList.remove('hidden');
        try{
            await refreshAll();
        }catch(e){
            // if fails, show login again
            loginCard.classList.remove('hidden');
            appCard.classList.add('hidden');
        }
    })();
}
</script>
</body>
</html>
)rawliteral";
}
/* ===== API handlers
===== */
void sendJsonStatus(const String& role) {
    // motion read
    motion = (digitalRead(PIR_PIN) == HIGH);
    String json = "{";
    json += "\"temperature\":" +
String(temperature, 1) + ",";
    json += "\"humidity\":" + String(humidity, 1)
+ ",";
    json += "\"motion\":" + String(motion ?
"true" : "false") + ",";
    json += "\"role\":" + role + ",";
    // env history
    json += "\"env\":";
    for (int i = 0; i < envCount; i++) {
        json += "\"" + envLog[i] + "\"";
        if (i < envCount - 1) json += ",";
    }
}

```

```

    }
    json += ",";
    // motion history
    json += "\"motionLog\":";
    for (int i = 0; i < motionCount; i++) {
        json += "\"" + motionLog[i] + "\"";
        if (i < motionCount - 1) json += ",";
    }
    json += "]";
    json += "}";
    server.send(200, "application/json", json);
}
void handleStatus() {
    User* u = nullptr;
    String role;
    if (!verifyMitm(u, role)) return;
    // read sensors (no logging here)
    temperature = dht.readTemperature();
    humidity = dht.readHumidity();
    if (isnan(temperature)) temperature = 0;
    if (isnan(humidity)) humidity = 0;
    sendJsonStatus(role);
}
void handleRefresh() {
    User* u = nullptr;
    String role;
    if (!verifyMitm(u, role)) return;
    // read sensors
    temperature = dht.readTemperature();
    humidity = dht.readHumidity();
    if (isnan(temperature)) temperature = 0;
    if (isnan(humidity)) humidity = 0;

    // append logs ONLY on refresh (your
requirement)
    pushEnvLog(temperature, humidity);
    motion = (digitalRead(PIR_PIN) == HIGH);
    pushMotionLog(motion);
    sendJsonStatus(role);
}
void handleFan() {
    User* u = nullptr;
    String role;
    if (!verifyMitm(u, role)) return;

    if (role == "viewer") {
        server.send(403, "text/plain", "Forbidden");
        delay(5);
        return;
    }
    fanState = !fanState;
    digitalWrite(FAN_PIN, fanState ? HIGH :
LOW);
}

```

```

    server.send(200, "text/plain", "OK");
}
/* ===== SETUP / LOOP
===== */
void setup() {
    pinMode(FAN_PIN, OUTPUT);
    pinMode(PIR_PIN, INPUT);
    dht.begin();
    startMillis = millis();
    WiFi.begin(ssid);
    while (WiFi.status() != WL_CONNECTED)
        delay(300);
    // UI

```

```

    server.on("/", [](){ server.send(200,
"text/html", webPage()); });
// API
server.on("/api/status", handleStatus);
server.on("/api/refresh", handleRefresh);
server.on("/api/fan", handleFan);
server.begin();
}
void loop() {
    server.handleClient();
}

```

Порівняльна таблиця ESP32 та альтернативних MCU

Критерій	ESP32	STM32	RP2040	Arduino Uno
Архітектура CPU	Dual-core Xtensa / RISC-V	ARM Cortex-M	Dual-core ARM Cortex-M0+	8-bit AVR
Тактова частота	До 240 МГц	До 400+ МГц (залежить від моделі)	133 МГц	16 МГц
Wi-Fi / Bluetooth	Вбудовані	Відсутні (потрібні модулі)	Відсутні	Відсутні
Secure Boot	Вбудований (апаратний)	Частково (залежить від серії)	Обмежений	Відсутній
Flash Encryption	Так	Частково	Ні	Ні
Апаратні криптомодулі	AES, SHA, RSA, ECC, RNG	Частково	Мінімальні	Відсутні
Захист від reverse engineering	Високий (eFuse, encryption)	Середній	Низький	Дуже низький
Орієнтація на IoT	Висока	Загального призначення	Навчальна / DIY	Навчальна
Простота розробки	Висока	Середня	Висока	Дуже висока
Вартість	Низька	Середня	Низька	Низька

Порівняльна таблиця симуляторів для IoT і мікроконтролерів

Критерій	Wokwi	Proteus	Tinkercad	QEMU
Підтримка ESP32	Повна	Обмежена / відсутня	Відсутня	Часткова
Робота в браузері	Так	Ні	Так	Ні
Реалістичність периферії	Висока	Середня	Низька	Висока (але складна)
Простота використання	Дуже висока	Середня	Дуже висока	Низька
Налаштування мережевих сценаріїв	Так	Обмежено	Ні	Так
Не потребує фізичного обладнання	Так	Так	Так	Так
Орієнтація на IoT	Так	Ні	Навчальна	Ні
Поріг входу	Низький	Середній	Низький	Високий