Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра інформатики факультету інформатики

Розробка комплексного підходу до захисту систем інтернету речей, що базується на ранньому виявленні загроз і гранульованому контролі доступу

Текстова частина магістерської роботи

за спеціальністю “Інженерія Програмного Забезпечення” 121

Керівник: Шабінська М.О.,

ст. викладач, канд. техн. наук

“____” _____ 2024 р.

Виконав: студент Щербина С.С.

“____” _____ 2024 р.

Київ 2024
Міністерство освіти і науки України

Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра інформатики факультету інформатики

ЗАТВЕРДЖУЮ
Зав.кафедри інформатики,
к.ф.-м.н., доц. С. С. Гороховський
_____
“____”_____2024 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ
на магістерську роботу

Студенту 2 р.н. магістерської програми Інженерія програмного забезпечення Щербині Сергію Сергійовичу

Завдання Розробити комплексний підхід до захисту систем інтернету речей, що базується на ранньому виявленні загроз і гранульованому контролі доступу.

Зміст текстової частини до магістерської роботи:

Зміст
Вступ
РОЗДІЛ 1. Типи загроз
РОЗДІЛ 2. Проблеми автентифікації та авторизації
РОЗДІЛ 3. Шифрування
РОЗДІЛ 4. Реагування на інциденти безпеки
РОЗДІЛ 5. Розробка комплексного підходу
Висновки
Список використаних джерел
Додатки

Дата видачі “___” _____ 2024 р.

Керівник
Шабінська М.О. кандидат техн. наук, ст. викладач _____

Завдання отримав
Щербина С.С. _____

Зміст

Календарний план виконання роботи

№	Назва етапу	Термін виконання	Примітки
1	Формування завдання на дипломну роботу	31.10.2023	
2	Реалізація прототипу практичної частини	27.01.2024	
3	Реалізація MVP практичної частини	27.02.2024	
4	Реалізація всієї запланованої практичної частини	31.03.2024	
5	Написання текстової частини	12.05.2024	
6	Правки практичної частини за результатами попереднього захисту	23.05.2024	
7	Правки текстової частини відповідно до змін практичної частини	25.05.2024	
8	Кінцеві правки усієї роботи	27.05.2024	
9	Створення презентації для захисту	01.06.2024	

Вступ

Інтернет речей розпочав трансформацію нашого цифрового простору, в якому повсякденні об’єкти є взаємопов’язаними та здатні до комунікації між собою. Ця трансформація не тільки спрощує нам життя, а ще й відкриває дорогу до небачених раніше можливостей та ефективності у сферах розумного будинку, медицини, індустріального виробництва, міського управління тощо. Проте, як і з будь-яким технологічним проривом, IoT(Internet of Things) вимагає зваженого та обережного впровадження. Насамперед, масштабне та неконтрольоване впровадження IoT пристроїв провокує багато проблем з безпекою, які необхідно вирішувати.

Основою IoT пристроїв є здатність збирати, оброблювати та передавати інформацію автономно, без людського втручання. Хоча саме це і є головною іновацією, така автономність створює багато вразливостей у безпеці. Наприклад, IoT пристрої часто використовують як приватні, так і публічні мережі, таким чином розширюючи потенційний вектор атаки зловмисника на конфіденційність та цілісність даних. Але це не єдине, про що потрібно турбуватися. Не менш важливими є надійність та доступність критично важливого функціоналу, що такі прилади надають. Від термостатів та годинників до автономних транспортних засобів та інфраструктури міста, вплив скомпрометованих IoT пристроїв може варіюватися від простої незручності до катастрофічних збоїв, що вплинуть на життя мільйонів людей. Прикладом слугує інцидент, що стався 23 грудня 2015 року [1]. Російські хакери влаштували масштабну кібер-атаку та отримали доступ до системи управління компанії «Прикарпаттяобленерго». Внаслідок цієї атаки, виконаної за допомогою троянської програми BlackEnergy, приблизно 230 тис. мешканців залишились без світла. «Чернівціобленерго» та «Київобленерго» також зазнали синхронних атак, які мали менш значні наслідки.

IoT прилади зазвичай мають мало обчислювальної потужності та оперативної пам’яті. Такі обмеження призводять до того, що часто такі традиційні підходи до безпеки як складні алгоритми шифрування є неможливими у впровадженні. До того

ж, неоднорідність IoT екосистем, що складається з різних пристроїв, часто пропрієтарних протоколів та стандартів, ускладнює впровадження єдиного рішення в плані безпеки.

Зважаючи на вищезгадані проблеми та вибухове зростання кількості IoT приладів у використанні, також зростає і потенціал для несанкціонованого втручання в роботу систем. За прогнозом Cybersecurity Ventures, кількість IoT пристроїв у світі зросте до 25.1 мільярда до 2025 року. Кожний з них є потенційною точкою входу для зловмисника. Таке широке прийняття створює велику, часто погано захищену мережу взаємопов'язаних приладів, яка може бути використана зловмисником для власних цілей: крадіжки даних, DDoS атак тощо.

Більше того, "non-patchable"(не підлягають виправленню) та "forever-day"(вічні) вразливості тільки ускладнюють проблему. Багато IoT пристроїв нечасто отримують оновлення прошивки в зв'язку з логістичними проблемами доставки цих оновлень, або ж тому що вони були забуті виробником і більше не підтримуються. Такі вразливості презентують вічно відкриті двері для зловмисників і тому вимагають іноватівного підходу, що простягається далі поточних рішень.

Метою даної роботи є розробка комплексного рішення для захисту IoT систем, а також дослідження відомих та новітніх підходів до захисту IoT систем, включаючи методи шифрування, способи авторизації та автентифікації, застосування машинного навчання. Існує нагальна потреба в надійних та здатних до масштабування рішень. Вони повинні враховувати унікальні характеристики IoT екосистем, а саме різноманітність, обмеженість в обчислювальних ресурсах і фізичну природу приладів.

Розділ 1. Типи загроз

Інфраструктуру навколо IoT пристроїв можливо розділити на 4 рівні [2].

Рівень сприйняття

На цьому рівні IoT пристрої збирають дані про навколишній світ.

Захоплення вузла

Зловмисник отримує доступ до однієї з ключових нод, таких як шлюз. Шлюзом слугує IoT пристрій, який об’єднує декілька сенсорів чи інших пристроїв. Втрата контролю над шлюзом може призвести до витоку всієї інформації, включаючи зв'язок між відправником і одержувачем, ключ, який використовується для забезпечення безпечного зв'язку, і інформацію, що зберігається в пам'яті пристроїв.

Імперсоніфікація вузла

Атака, під час якої зловмисник додає вузол до системи та вводить підроблені дані. Такі дії спрямовані на припинення передачі реальної інформації. Вузол, доданий зловмисником, відтягує на себе ресурси реальних вузлів у спробі перешкодити роботі системи.

Атака відтворення

Зловмисник підслуховує передачу даних між відправником і одержувачем, отримуючи масив справжньою комунікації. Далі зловмисник надсилає ту ж оригінальну інформацію повторно. Повідомлення знаходиться в зашифрованому вигляді, тому одержувач сприйме його як коректний запит і виконає дії бажані зловмисником.

Часова атака

Зазвичай спрямована на пристрої з слабкими обчислювальними ресурсами. Дозволяє зловмиснику виявляти вразливі місця та витягувати секрети чи інші дані. Атака націлена на визначення часу, потрібного системі, щоб відповісти на різні запити, введення або криптографічні алгоритми.

Атака позбавлення сну

Сенсорні вузли на батареях зазвичай мають режим сну з низьким енергоспоживанням, що використовується для економії енергії. Основною метою атаки позбавлення сну є вичерпання ресурсу батареї цільових вузлів шляхом максимізації їхнього енергоспоживання, таким чином скорочуючи термін їхньої роботи та потенційно виводячи з ладу частини системи.

Рівень мережі

Рівень мережі також відомий як транспортний рівень. Він діє як міст між рівнем сприйняття та рівнем обробки даних. Він передає інформацію, зібрану через датчики та IoT пристрої.

Підслуховування

Зловмисник підслуховує комунікацію в реальному часі через відкриту або погано зашифровану мережу у спробі викрасти конфіденційну інформацію.

DoS/DDoS атака

Атака, спрямована на те, аби запобігти доступу справжніх користувачів до пристроїв або інших мережевих ресурсів. Зазвичай це досягається заповненням цільових пристроїв або мережевих ресурсів надлишковими запитами, щоб унеможливити або ускладнити їх використання справжніми користувачами.

Main-in-The-Middle

Зловмисник перехоплює та видозмінює комунікацію між відправником і одержувачем без їхнього відома. Оскільки зловмисник контролює комунікацію, він може змінювати повідомлення відповідно до своїх потреб. Це створює серйозну загрозу безпеці, оскільки успішна атака дає зловмиснику можливість перехоплювати та маніпулювати інформацією в режимі реального часу.

Рівень обробки даних

Рівень обробки є також відомим як проміжний рівень. Він відповідає за збір даних, які надсилаються з транспортного рівню. Далі виконується обробка цих даних. Рівень несе відповідальність за видалення надлишкових даних, задля отримання тільки корисної інформації.

Виснаження ресурсів

Виникає як наслідок DoS/DDoS атак, під час яких зловмисник надсилає жертві чи системним вузлам багато запитів, з метою зробити сервіс недоступним для користувачів.

Експлойти

Зазвичай цей рівень в силу своєї природи є добре захищеним від різного виду експлойтів. Проте в залежності від конкретних ролей, які виконують вузли на цьому рівні, ноди можуть бути вразливими до тих же типів атак, що і на рівні застосунку.

Рівень застосунку

Рівень, що напряму використовується користувачем.

Експлойти

Цей рівень не є унікальним тільки для IoT і є вразливим до всіх стандартних методів атак, а саме: SQL ін’єкцій, Cross-Site Scripting (XSS), Cross-Site Request Forgery (CSRF), Local File Inclusion (LFI) та Remote File Inclusion (RFI), DDoS, атака через некоректну десеріалізацію даних, XML External Entities (XXE), Remote Code Execution (RCE), експлойти в авторизації чи автентифікації тощо.

Висновок до розділу 1

Безпека IoT інфраструктури є критичним аспектом, що потребує комплексного підходу. Різноманітність атак та вразливостей вимагає впровадження

багаторівневого підходу до безпеки, що включає як технічні, так і організаційні рішення.

Основними проблемами для безпеки в IoT є:

- несанкціонований доступ до пристроїв та даних
- підслуховування комунікації
- маніпуляція комунікації
- виснаження ресурсів систем та пристроїв

Також необхідно враховувати загрози, що є специфічними для кожного окремо рівня інфраструктури, та впроваджувати відповідні заходи безпеки, наприклад: використання надійних методів шифрування, регулярне оновлення програмного забезпечення, моніторинг мережевої активності, впровадження горизонтальної архітектури для систем, фізичний захист IoT пристроїв тощо.

Розділ 2. Проблеми автентифікації та авторизації

Вразливі облікові дані

Наявні у використанні IoT прилади мають багато вразливостей безпеки. Серед них можливо виокремити використання слабких паролів, паролів за замовчуванням та hardcoded (вписаних у прошивку пристроїв) паролів. Ця вразливість виділяється поміж інших через свою поширеність та простоту, з якою нею можна скористатися. Такі паролі створюють значні ризики як для звичайних користувачів, так і для організацій. Потенційно це може призвести до несанкціонованих доступів, витоку даних і масштабних збоях у мережах.

Слабкі паролі – паролі, які легко вгадати, або ті, які створені з використанням простих шаблонів чи поширених слів. Наприклад, «12345789», “pass”, “password”, “admin” часто використовують через їхню простоту для запам'ятовування.

Паролі за замовчуванням – заводські паролі, що поставляються виробником пристрою для ініціалізації пристрою та першого доступу. Часто вони є загальними для великої кількість приладів. Наприклад, зв'язка логіну та паролю admin/admin.

Hardcoded паролі – теж заводські паролі, проте ці є закодованими напряму в прошивку приладу, тому користувачі не можуть їх змінити взагалі. Така практика іноді виправдовується необхідністю мати резервний метод автентифікації, проте створює суттєві ризики для безпеки.

Причина використання вразливих паролів

Причиною є поєднання декількох факторів.

- Простота використання: виробники вважають, що користувачі знаходять процес зміни паролю морошливим чи надлишковим. Тому пріорітезується зручність для користувача над його безпекою.

- Недостатня обізнаність: користувачі можуть не знати про ризики пов'язані з стандартними паролями, або ж недооцінюють цінність своїх приладів для зловмисників.
- Технічні обмеження: у випадку, коли виробник використав hardcoded паролі, процес зміни не передбачений, навіть якщо того бажає користувач.
- Бізнес обмеження: конкуренція на ринку змушує виробників зменшувати витрати на розробку. В цьому випадку, жертвуючи розробкою надійного, а іноді і базового, функціоналу безпеки.

Стратегії мітигації

Для виробників

- Уникати використання стандартних паролів: прилади мають вимагати встановлення нового унікального паролю від користувача під час першого налаштування.
- Уникати використання hardcoded паролів: необхідно розроблювати прилади так, аби паролі не були зашитими в прошивку, та існувала можливість для користувача встановити свій власний пароль.
- Інформування користувачів: потрібно надавати прості та зрозумілі інструкції по зміні паролю та заохочувати користувачів виконати цю дію.

Для користувачів

- Під час першого доступу та ініціалізації приладу завжди необхідно змінювати паролі за замовченням на складні та унікальні паролі.
- Хорошою практикою є використання перевірених менеджерів паролів. Вони полегшать процес обліку та дозволять використовувати складні та згенеровані паролі.
- Потрібно регулярно оновлювати свої паролі, особливо якщо існує підозра, що прилад було скомпрометовано.

Стандарти та правові норми

Державні установи та міжнародні агенства також можуть суттєво впливати на вирішення цієї проблеми. Треба вводити стандарти, що регулювали б формат та логіку роботи з паролями на IoT приладах, не виключаючи заборону на стандартні та hard-coded паролі. Наприклад, Сполучене Королівство(UK) має документ рекомендаційного характеру - “Code of Practice for Consumer IoT Security”. В ньому чітко прописана рекомендація не використовувати стандартні паролі.

Випадки взломів

Одним з відомих випадків взлому внаслідок використання вразливих паролів є Mirai ботнет. У 2016 зловмисне ПЗ Mirai сканувало інтернет в пошуках IoT приладів, на яких використовувалися заводські налаштування логіну та паролю. Всього ПЗ мало дані про 61 комбінацію, які використовувало методом перебору для отримання доступу та подальшого інфікації пристроїв. Отримавши доступ до достатньої кількості приладів, зловмисники запустили масштабну DDOS атаку націлену на ряд великих веб-сервісів, поміж яких: X(Twitter), Netflix, CNN тощо. Дослідження пізніше показали, що більшість скомпрометованих пристроїв були вироблені компаніями XiongMai Technologies та Dahua. На піку процесу інфікації в інтернеті можливо було знайти 560 000 приладів, що були вразливими до атак подібного типу.

Окремо слід зазначити постійні атаки на IP камери. За інформацією пошукової системи Shodan, у світі нараховується щонайменше 8373 камери, що використовують протокол потокової передачі даних в реальному часі(RTSP) і стандартну пару логіну та паролю. Деякі камери навіть можна знайти в google, шукаючи за стандартними URL, які використовує виробник.

Унікальність IoT систем

З вищезгаданого добре видно, що автентифікація та авторизація є критично важливими для забезпечення безпеки IoT приладів. Автентифікація відповідає за те, аби тільки дозволені прилади та користувачі мали змогу отримати доступ до сервісу. Авторизація ж – контролює дії, які може робити авторизована сутність. В

контексті IoT, традиційних підходів може бути недостатньо, з огляду на унікальний стан IoT екосистем: обмеження в обчислювальних ресурсах та різноманіття приладів і вузлів.

Розподілені центри довіри

Стаття за авторством Kim, H., та Lee, E. A. [3] пропонує використовувати локально централізовану, але глобально розподілену систему управління довірою, замість глобальних централізованих центрів довіри (CA). Така пропозиція обґрунтувана тим, що у випадку скомпрометованого CA, скомпрометованою стає вся система. Прикладом слугує інцидент WoSign, який стався в 2016 році. Китайський CA, WoSign, помилково видав сертифікат не тим сутностям. Запропонована система передбачає розгортання локальних об’єктів автентифікації та авторизації на периферійних пристроях, наприклад, телефонах.

Документ пропонує зміну парадигми від традиційних централізованих систем до більш розподіленої системи. Завдяки децентралізації процесів автентифікації та авторизації та розміщенню їх ближче до місця, де генеруються та обробляються дані, IoT може досягти вищого рівня безпеки. Цей метод вирішує проблеми масштабу та різноманітності в системах IoT, зменшує затримку, підвищує ефективність і потенційно пропонує більшу стійкість проти атак, спрямованих на централізовані системи.

Нові інтерфейси автентифікації та авторизації

Стаття за авторством Shahzad, M., та Singh, M. P. [4] розглядає IoT пристрої, у яких немає традиційних інтерфейсів користувача, таких як клавіатура та сенсорний екран. Як приклад вразливостей, пов’язаних з нетрадиційними способами автентифікації та авторизації наводиться Apple Watch, з його функціоналом розблокування за допомогою розблокування зв'язаного з ним телефону. Якщо такий годинник було втрачено, власник може нічого не підозрюючи розблокувати свій телефон, а з ним і годинник. Далі зловмисник може отримати доступ до Mac

жертви за допомогою годинника. Таким чином, автори стверджують про необхідність так званої постійної автентифікації та авторизації.

Динамічний характер середовищ IoT, де пристрої можуть не постійно перебувати під контролем користувача, вимагає переходу від одноразових механізмів автентифікації до рішень, які можуть постійно перевіряти особу користувача. Автори наголошують, що середовища IoT, незважаючи на свої обмеження, пропонують нові шляхи для інноваційних підходів до автентифікації та авторизації. Зокрема, вони обговорюють WiFiU, гіпотетичну систему на основі Wi-Fi для аутентифікації людини. Вона демонструє, як навколишні сигнали, в цьому випадку радіохвилі, можна використовувати для безперервної перевірки користувача.

Також розглядаються варіанти з використанням унікальних людині біологічних властивостей, таких як ритм серцебиття, кількість світла, що відбиває людське світло, радіохвилі тощо. За допомогою машинного навчання, гіпотетично, це можливо перетворити в системи автентифікації та авторизації.

Авторизація та автентифікація в несприятливих умовах

Стаття за авторством Echeverria, S., Lewis, G. A., Klinedinst, D., та Seitz, L. [5] пропонує вирішення проблеми захисту IoT пристроїв у середовищах, де традиційні заходи безпеки можуть бути неможливими, наприклад у зонах з обмеженими каналами підключення і ресурсами. Такі зони є звичайними для служб швидкого реагування, військових, бойових медиків тощо. Основна увага їхніх досліджень зосереджена на забезпеченні ефективного методу як для автентифікації, так і для авторизації пристроїв IoT у таких несприятливих умовах.

Документ зосереджується на трьох проблемах: отримання облікових даних(ключів) IoT пристроями та ресурс сервером, підтримання безпечної комунікації та оперування в несприятливих середовищах із переривчастими або низькоякісним з’єднанням.

Для отримання облікових даних пропонується 4 варіанти:

1. Запис унікальних облікових даних у прошивку на стадії виробництва. Такий метод є серйозною перепоною для імперсоніфікації пристрою чи підслуховування комунікації, проте вимагає більше зусиль на стадії виробництва та підтримки.
2. Облікові дані пристроїв зберігаються у вигляді фізичних QR кодів та додаються до серверу авторизації перед тим, як бути відданими користувачам. QR код залишається в безпечному місці. В цьому і подальшому варіантах використовується так звана «довіра через фізичну близькість», а QR код використовується для процесу ACE обміну.
3. Облікові дані пристроїв зберігаються у вигляді фізичних QR кодів та додаються до серверу авторизації напряму перед використанням. У цьому варіанті існує проблема втрати QR коду та подальшої компрометації. Також, якщо QR було скомпрометовано, його неможливо змінити без зміни прошивки.
4. Випадково згенерований QR код, що надано з пристроєм, використовується для початкового шифрування та отримання нового ключа. Цей процес можливо виконати тільки після фізичного перезавантаження пристрою, аби уникнути використання «довіри через фізичну близькість» як вразливість.

Для забезпечення безпечної комунікації використовується CoAP через IPv6 із DTLS. Також, важливою частиною всієї системи є можливість відкликати токени авторизації(token revocation). Пристрої та ресурс сервер постійно запитують у сервера авторизації інформацію про актуальність токенів та доступів.

Висновок до розділу 2

Проблему, яку представляють вразливі облікові дані в IoT приладах неможливо переоцінити. Усунення цієї вразливості потребує зусиль від виробників, користувачів та навіть державних установ.

Забезпечення захисту IoT пристроїв та мереж, в яких вони оперують, потребує імплементації надійних механізмів автентифікації та авторизації.

Враховуючи часто конфіденційний характер даних і потенційні наслідки дір у безпеці, розробка та впровадження таких механізмів має першочергове значення. Рішення мають бути ефективними та масштабованими, здатними адаптуватися до унікальних вимог IoT екосистем.

Розділ 3. Шифрування

У величезному та взаємопов'язаному світі IoT шифрування даних є гарантом конфіденційності та цілісності. Оскільки пристрої IoT збирають, передають і обробляють постійно зростаючий обсяг конфіденційних даних, відсутність надійного шифрування даних стає явною вразливістю. Ця відсутність не тільки наражає користувачів на ризик витоку даних і конфіденційності, але й підриває довіру, необхідну для більш широкого впровадження IoT. Поєднуючи це разом з унікальною природою IoT приладів, а саме – часту обмеженість в ресурсах, виникає потреба в обчислювально легких та достатньо надійних методах та парадигмах шифрування. Цей розділ має на меті представити як добре відомі і широко використовувані підходи та методи, так і нові, що можуть слугувати альтернативою чи покращенням.

TLS

Протокол TLS можна вважати стандартним вибором для впровадження захисту комунікації. Наприклад, TLS використовується в інфраструктурах, що містить брокер повідомлень MQTT. TLS(transport layer security) в своїй основі має протокол SSL(Secure Sockets Layer), що був розроблений компанією Netscape [6]. Протокол було реалізовано над транспортним рівнем, в цьому випадку – над TCP. Свого часу, правильно сконфігурований SSL надавав хороші гарантії безпеки. Потенційний зловмисник міг дізнатися лише параметри з'єднання, такі як конкретний тип шифрування, частоту обміну та кількість даних.

В 1999 IETF стандартизував протокол SSL, і його було перейменовано в TLS. В загальному, назви протоколів є взаємозамінними, тому в літературі часто можна помітити назву "SSL/TLS". Проте варто зауважити, що існує декілька версій цього протоколу. Перша версія мала назву SSL 2.0, що невдовзі була замінена версією SSL 3.0. Інша назва цієї версії – TSL 1.0. Розширенням та покращенням слугувала версія TSL 1.1, а вже за нею у 2008 році було розроблено версію TSL 1.2, що наразі є найбільш актуальною.

Протокол TLS виконує наступні функції: шифрування, автентифікація та гарантія цілісності даних. Задля встановлення безпечного зв’язку, учасники мають обрати методи шифрування та ключі. В протоколі цей процес має назву TLS Handshake. Протокол використовує парадигму криптографії з відкритим ключем. Це означає, що клієнти можуть встановити безпечний зв'язок без попередніх знань один про одного, проте це залежить від обраного криптографічного набору(наприклад, TLS-SRP вимагає попереднього знання про спільний секрет). Також, в процедурі TLS Handshake підтверджується автентичність презентованих сторін, зазвичай клієнт перевіряє сервер, проте існує можливість взаємної перевірки, відома як mTLS.

Кожне надіслане повідомлення містить MAC(Message Authentication Code) код, що створюється за допомогою односторонньої криптографічної функції хешування та виконує функцію контрольної суми. Цю контрольну суму також генерує і отримувач для визначення цілісності інформації.

TLS Handshake

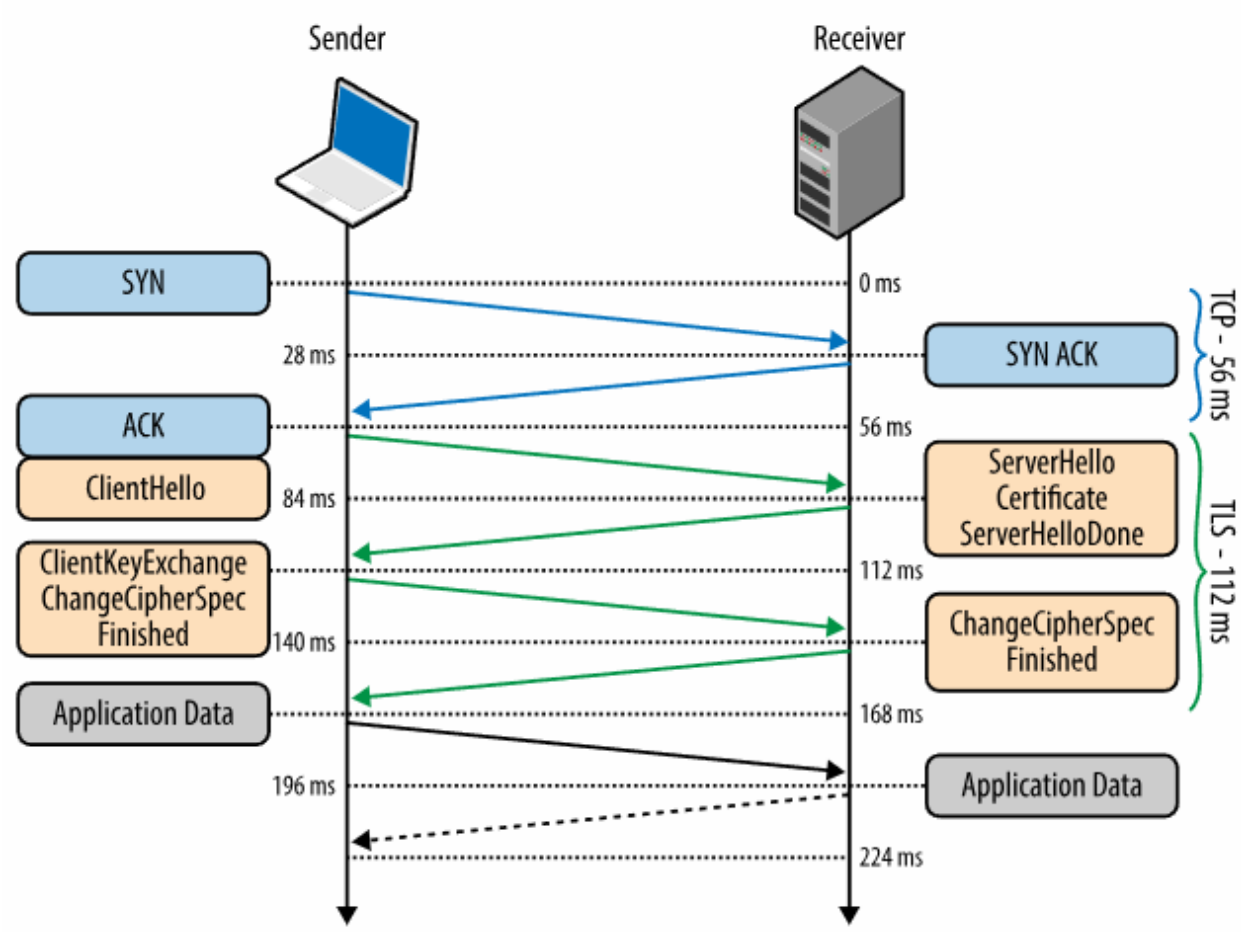

Рис. 1. Процес TLS Handshake

1. Спочатку між клієнтом і сервером встановлюється TCP-з'єднання.
2. Клієнт посилає на сервер специфікацію у вигляді відкритого тексту (а саме версію протоколу, яку він хоче використовувати, методи шифрування, що підтримуються тощо).
3. Сервер затверджує версію протоколу, що використовується, обирає спосіб шифрування з наданого списку, прикріплює свій сертифікат і відправляє відповідь клієнту. У випадку використання mTLS клієнт ще запитує сертифікат клієнта.
4. На цьому етапі версія протоколу та спосіб шифрування вважаються затвердженими. Клієнт перевіряє надісланий сертифікат та, залежно від встановлених параметрів, ініціює або RSA, або процес обміну ключами Діффі-Геллмана(D-H).

5. Сервер обробляє надіслане клієнтом повідомлення, звіряє MAC, і відправляє клієнту заключне (Finished) повідомлення в зашифрованому вигляді.
6. Клієнт розшифровує отримане повідомлення та звіряє MAC. З'єднання вважається встановленим і починається обмін даними.

Процес TLS Handshake є довгим та складним. В протоколі прописано декілька оптимізацій, що мають на меті скоротити та спростити цю операцію. А саме: “abbreviated handshake” та “false start”.

Abbreviated Handshake

Ще у версії SSL 2.0 було перебачено скорочений процес TLS Handshake, що має назву “abbreviated handshake”. В пакеті “ServerHello” сервер може надіслати 32 байтний ідентифікатор сесії. Цей ідентифікатор генерується на сервері та зберігається в кеш для подальшого використання. Клієнт, отримавши ідентифікатор сесії, також зберігає його в себе. Тепер, якщо з’єднання було розірвано, клієнт може в пакеті “ClientHello” надіслати вищезгаданий ідентифікатор. Якщо учасники з'єднання мають однакові ідентифікатори, процес TLS Handshake буде скорочено. Якщо ні – виконається повна версія.

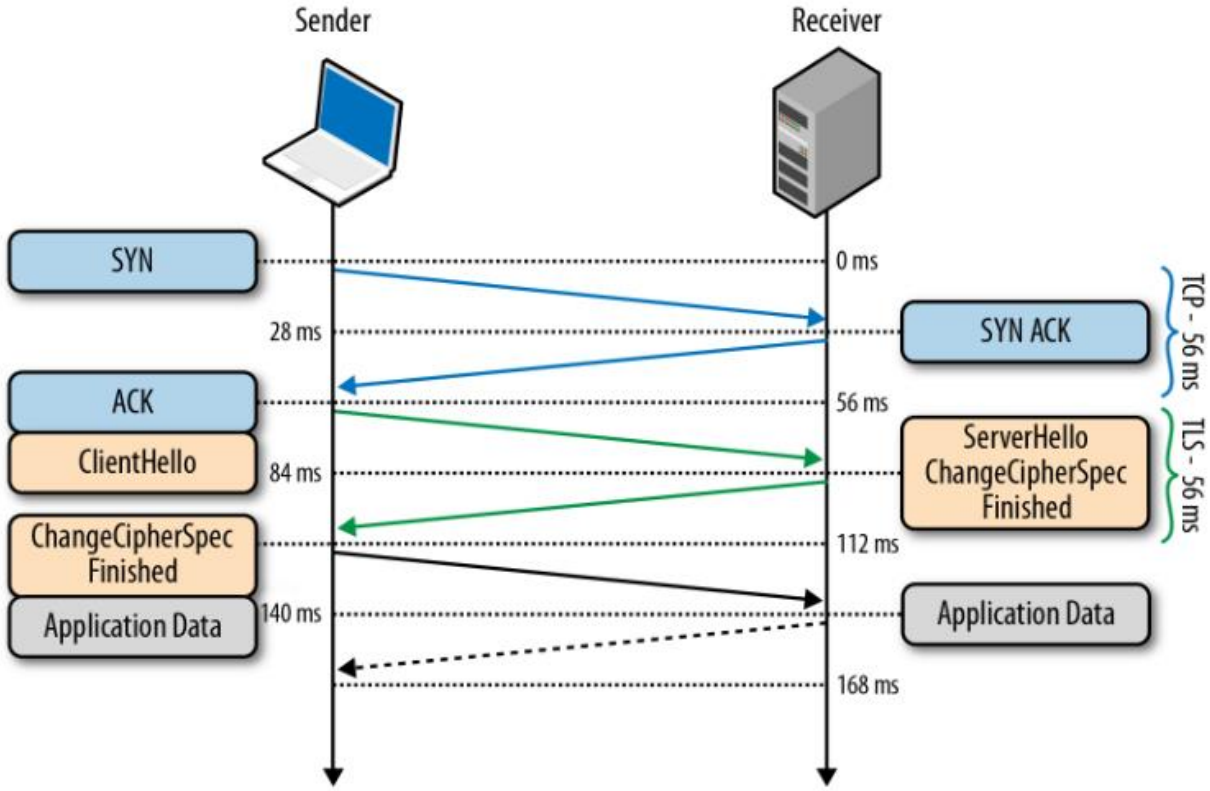

Рис. 2. Процес TLS Abbreviated Handshake

Abbreviated handshake дозволяє пропустити генерацію симетричного ключа, що суттєво скорочує час встановлення з’єднання. Проте, такий підхід вимагає сервер зберігати дані про всі минулі сесії, що на великих обсягах може впливати на його швидкодію. Для вирішення цієї проблеми існує механізм “Session Ticket”. Якщо клієнт підтримує такий механізм, то під час першого з’єднання замість ідентифікатора сесії сервер надішле “Session Ticket” – параметри сесії, зашифровані приватним ключем серверу. В такому варіанті, при відновленні з’єднання клієнт замість ідентифікатора сесії надсилає вищезгаданий Ticket, що дозволяє серверу не зберігати дані про сесію.

TLS False Start

Механізм Abbreviated Handshake пришвидшує процес відновлення сесії, проте він ніяк не впливає ні на найперший етап встановлення зв'язку, TLS Hadnshake, ні на випадок, коли сесія вже не є дійсною. Аби пришвидшити роботу і тут, існує

опціональне розширення TLS False Start. Воно дозволяє надсилати дані ще до завершення процесу TLS Hadnshake.

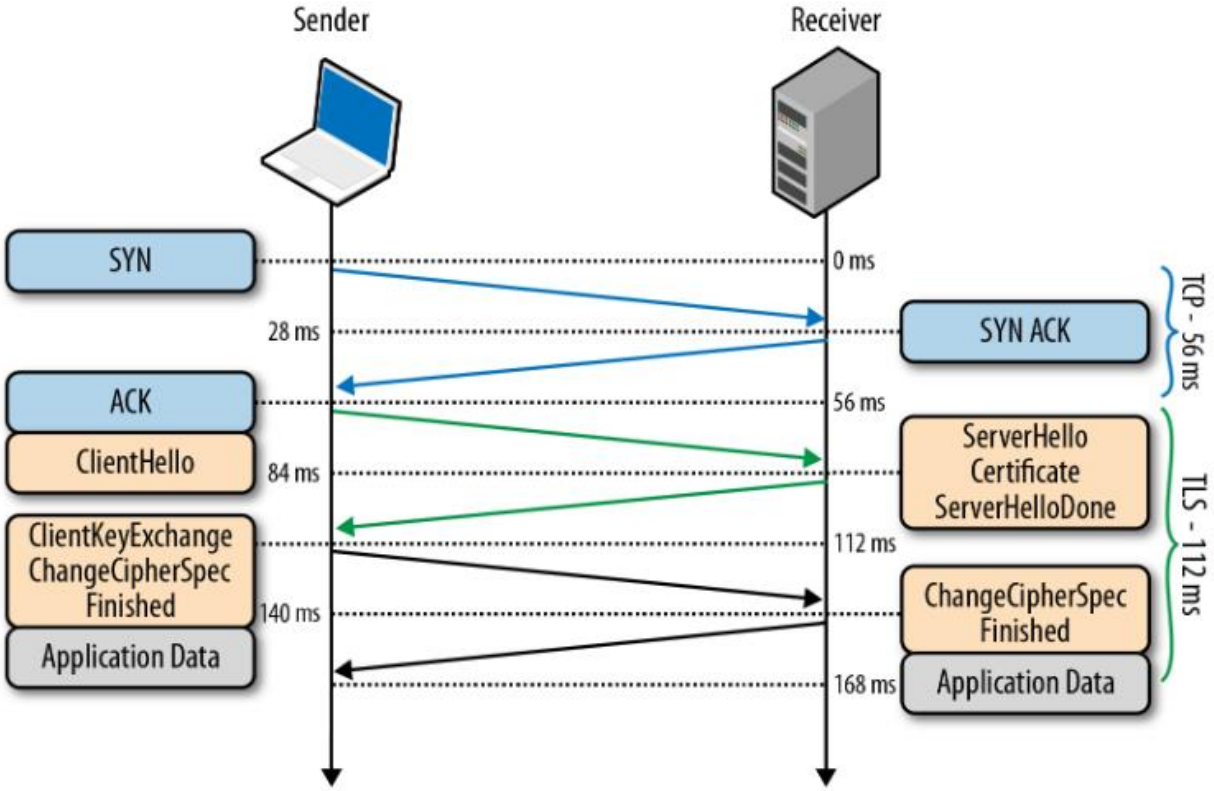

Рис. 3. Процес TLS False Start

Варто зауважити, що TLS False Start не впливає на роботу TLS Handshake. Він використовує припущення, що на момент, коли учасники дізналися про параметри з'єднання і потрібний симетричний ключ, вони вже можуть обмінюватися даними. Всі інші перевірки відбудуться паралельно. В результаті використання такого механізму стає на одну ітерацію обміну повідомленнями менше.

Обмін ключами

В силу різних причин раніше частіше за все обмін ключами відбувався за допомогою алгоритму RSA. У ньому клієнт генерує симетричний ключ, шифрує його за допомогою публічного ключа серверу та надсилає. Далі сервер розшифровує симетричний ключ за допомогою власного приватного ключа. На

цьому етапі обмін ключами вважається завершеним. Цей алгоритм має один недолік – у випадку, якщо зловмисник заволодіє приватним ключем серверу, він зможе розшифровувати всі сеанси зв'язку з цим сервером. Навіть гірше, зловмисник може роками записувати шифрований трафік, маючи на меті розшифрувати дані, коли він отримає приватний ключ сервера. Таким чином, втрата одного єдиного приватного ключа скомпрометовує не тільки поточну і майбутню комунікацію, а ще й історичну.

На відміну від RSA, процес обміну ключами Діффі-Геллмана(D-H) є більш захищеним, бо обраний симетричний ключ шифрування ніколи не виходить за межі ні клієнта, ні серверу, і тому не може бути перехоплений по мережі. DH зменшує ризики пов’язані з компрометації попередніх сеансів зв’язку, оскільки для кожного нового сеансу створюється новий тимчасовий симетричний ключ. В найгіршому випадку, коли зловмисник заволодів приватним ключем серверу, він зможе розшифровувати поточну і майбутню комунікацію, але не історичну.

Варто ще раз зауважити, що шифрування з публічним ключем використовується тільки в процесі TLS Handshake. Після встановлення зв’язку дані шифруються і розшифровуються за допомогою симетричної криптографії. Такий підхід обрано для збільшення швидкодії, оскільки шифрування з публічним ключем є набагато більше вимогливим у розрізі обчислювальних ресурсів.

Chain of trust

В протоколі TLS важливим концептом є ланцюг довіри. Коли клієнт звертається до серверу, він має мати змогу підтвердити автентичність наданого публічного ключа/сертифікату. Зробити це він може декількома способами, а саме:

- Certiticate/public key pinning – клієнт має збережений сертифікат чи публічний ключ серверу і порівнює його з наданим
- Перевірка наданого сертифікату за допомогою сертифікату центру довіри(CA certificate)

Ланцюг довіри засновано на сертифікатах автентичності, що надаються спеціальними установами – центрами сертифікації (CA – certificate authorities). З виданих сертифікатів складається наступний ланцюг.

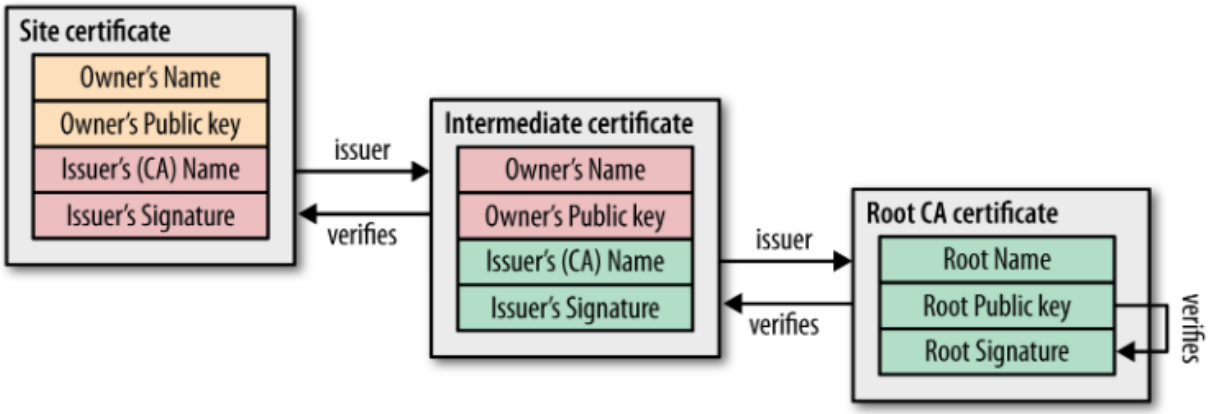

Рис. 4. Ланцюг довіри в TLS

Корінем цього ланцюгу є “Root CA certificate” – сертифікат, підписаний великим центром, якому довіряють всі. Він, в свою чергу, підписує проміжні сертифікати авентичності. Далі цей ланцюг продовжується і доходить вже до site/server/leaf сертифікатів. Причиною використання ланцюга є розподілення ризиків безпеки.

Очевидно, виникають випадки коли сертифікати будь-якого рівню є скомпрометованими та вимагають ануляції. Сертифікати містять в собі інструкції по перевірці їхньої актуальності, тому при перевірці сертифікату потрібно перевіряти весь ланцюг довіри.

В основі процедури перевірки знаходиться “Certificate Revocation List”(CRL) – список відкликаних сертифікатів. Кожний центр сертифікації має такий список. Якщо сертифікат там знаходиться, то це означає що його було відкликано і з'єднання встановлювати не можна. Проте запит на отримання всього списку відкликаних сертифікатів для перевірки одного є нераціональним з точки зору ресурсів. До того ж, такі списки публікуються періодично, а не постійно. Тому було розроблено механізм Online Certificate Status Protocol(OCSP). Він дозволяє

виконувати перевірку статусу сертифікату динамічно та селективно. Це зменшує навантаження на мережу та кінцевих користувачів, проте такоє створює низьку проблем:

- Центри сертифікації мають справлятися з навантаженням, що приносить з собою перевірка в реальному часі
- Центри сертифікації мають гарантувати постійну доступність до своїх ресурсів
- Центри довіри отримують доступ до інформації кінцевих клієнів, а саме – які сайти відвідує клієнт.

Сучасні браузери зазвичай підтримують обидва механізми, проте часто вони є вимкненими за замовчуванням з вищенаведених недоліків.

Використання в IoT

Найбільшою проблемою використання TLS для деяких приладів буде обмеженість обчислювальних ресурсів. Проте, існує велика кількість полегшених бібліотек та імплементацій [7], які залишають лише найнеобхідніші компоненти протоколу, позбавляючись всього іншого. Наприклад, можливо створити TLS бібліотеку, що буде займати всього 20кб оперативної пам’яті під час роботи. Більшість складнощів при розробці таких рішень полягає в великій кількості підтримуваних шифрувальних комлектів. Якщо взяти за основу TLS 1.2 комплект TLS_RSA_WITH_AES_128_CBC_SHA256, то доведеться імплементувати тільки RSA, AES та SHA-256. Також, процес Handshake можливо зробити синхронним та позбавитися всіх додаткових механізмів, як False Start та Abbreviated Handshake. Найскладнішою частиною всієї імплементації є робота з X.509 сертифікатами, але і тут можливо спросити, використавши механізм certificate pinning. Інший варіант, використати бандл TLS_SRP_SHA_WITH_AES_128_CBC_SHA, але тоді треба мати наперед відомий спільний секрет. Варто ще раз зауважити, що процес обміну RSA не надає forward secrecy захисту, тому все таки, рекомендується використовувати ECDHE шифр.

DTLS

Datagram Transport Layer Security (DTLS) – протокол побудований над UDP, що загалом надає ті ж самі гарантії безпеки і має схожу структуру, що і TLS. Використовується в таких протоколах, як CoAP. Через використання UDP як транспортного протоколу, DTLS має як плюси, так і мінуси в порівнянні з TLS. Найчастіше використовується в системах з великим обсягом стрімінгових даних. DTLS дозволяє уникати повільної комунікації, що присутня TLS. Проте, в той же час, він не змінює порядок пакетів та не гаратує їхню унікальність(non-replayability).

Відмінності з TLS

Field	TLS	DTLS
Function	It must function over a dependable TCP-based transport channel. Unreliable datagram traffic cannot be secured with it.	It is used to build "TCL over datagram" structures.
RFC	RFC 4346(V1.1), RFC 5246 (V1.2), RFC 8446(V1.3)	RFC 6347(V1.2)
Protocol	TLS covers both securities for TCP and UDP transport types.	DTLS is implied if the transport type is UDP.
Message type	1,3	1,3
Message Sequence Number	N/A	2
Fragment Offset	N/A	3
Fragment Length	N/A	3
Role	Client only	Server and Client

Рис. 5. Порівняльна таблиця відмінності TLS та DTLS

Всі відмінності між TLS та DTLS пов’язані з використанням протоколу UDP [8].

Ретрансляція

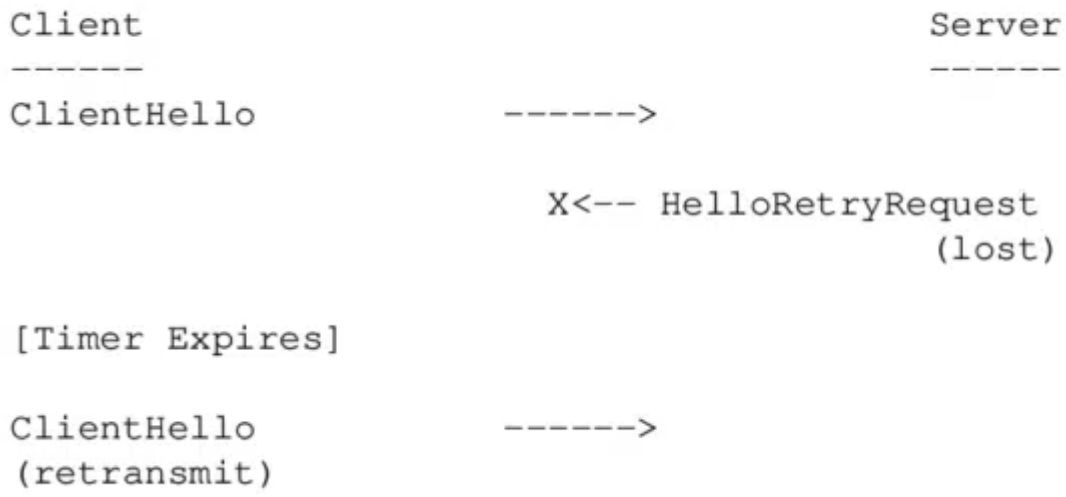

Рис. 6. Процес ретрансляції повідомлень в DTLS

Крім шифрування, DTLS також використовується для запобігання втрати або надходженню пакетів даних у неправильному порядку. DTLS використовує для цього простий таймер повторної передачі, при цьому кожний учасник зв'язку продовжує повторно передавати своє останнє повідомлення, доки не буде отримано відповідь.

Явні записи

TLS ділить довгу послідовність даних на кілька фрагментів. Протокол є відповідальним за розбір цих фрагментів на кінці отримувача, і тому клієнтський код не потребує ніяких додаткових модифікацій. На відміну, DTLS використовує записи, які можна надсилати повністю чи ні. Цими записами повинні керувати самі програми, тобто протокол не є відповідальним за це.

Толерування змін в записах

Датаграми можуть бути втрачені, або надійти до отримувача пошкодженими чи не в тому порядку. Таким чином, і клієнтські, і серверні частини можуть дозволяти нерегуляризовані комунікації. Наприклад, порядок записів може бути змінений з

будь-якої кількості причин (наприклад, затримка). Проте дублікати викличуть попередження, а записи з великою кількістю аномалій будуть проігноровані.

Відсутність явного завершення комунікації

Так само, як і UDP, у DTLS немає сигналу завершення передачі. В DTLS передача даних просто припиняється. Це означає, що клієнт, який отримує дані від сервера, не знає, чи було доставлено всі дані, чи сталася помилка під час зв'язку. На відміну, TLS сигналізує про це сповіщенням.

Захист від підміни IP-адреси

DTLS використовує cookie для запобігання підміни IP-адреси. Це дозволяє спілкуватися із сервером, не розкриваючи клієнтську IP-адресу. З іншого боку, TLS здійснює зв'язок лише після встановлення TCP-handshake, що ускладнює підміну IP-адреси.

Proxy Re-Encryption

У дослідженні за авторством SuHyun Kim та ImYeong Lee [9] пропонується використовувати Proxy Re-Encryption (PRE) - перешифрування на проксі. Цей спосіб шифрування вирішує критичну проблему захисту передачі даних в екосистемах IoT, що характеризується безліччю взаємопов'язаних пристроїв з обмеженими обчислювальними можливостями.

PRE — це криптографічна парадигма, яка дозволяє делегувати права на дешифрування. Вона надає проксі ноді можливість перетворювати дані, шифровані ключем A, у дані, які можливо розшифрувати ключем B. Під час цієї операції дані у відкритому вигляді не з'являються.

Цей механізм особливо підходить для середовища IoT з кількох причин:

- Обмежені ресурси: пристрої IoT часто мають обмежену обчислювальну потужність і пам'ять, що в цих випадках робить традиційні схеми

шифрування непрактичними. PRE пропонує полегшену альтернативу, яка враховує ці обмеження.

- Обмін даними: інфраструктура IoT часто вимагає безпечного обміну даними між кількома пристроями або користувачами. PRE полегшує це, дозволяючи безпечно та безпосередньо ділитися зашифрованими даними без необхідності розшифровки та повторного шифрування, заощаджуючи таким чином обчислювальні ресурси.
- Динамічне середовище: IoT-ландшафт дуже динамічний, де пристрої часто приєднуються до мережі та виходять з неї. PRE підтримує цей динамізм, забезпечуючи гнучке та безпечне керування ключами та механізми контролю доступу до даних.

У контексті IoT PRE можна використовувати наступним чином:

1. Шифрування – кожний пристрій генерує собі пару ключів: публічного(pk) і приватного(sk).
2. Кожен пристрій генерує ключ перешифрування аби обмінюватися даними з іншими пристроями. Аби пристрій A міг поділитися даними з пристроєм B, йому потрібно згенерувати ключ повторного шифрування rk(A→B). Він генерується за допомогою sk(A) та pk(B) і далі надсилається на проксі ноду. У цьому сценарії кожен пристрій створює ключ перешифрування для всіх пристроїв, крім себе.
3. Проксі нода перешифровує дані від пристрою A за допомогою ключа rk(A→B) та надсилає на пристрій B.
4. Пристрій B розшифровує дані за допомогою sk(B).

Novel Tiny Symmetric Encryption Algorithm

У статті за авторством S. Rajesh, V. Paul, Varun G. Menon та M. Khosravi [10] пропонується новий підхід до TEA шифрування. Цей метод спрямований на

підвищення безпеки та ефективності передачі текстових файлів між пристроями IoT, що відповідає критичній потребі в легких криптографічних рішеннях.

NTSA розроблено для подолання обмежень алгоритму Tiny Encryption Algorithm (TEA) та його варіантів. Вони, незважачюси на невелике використання пам’яті та простоту імплементації, страждають від вразливостей безпеки через використання постійного ключа протягом усього процесу шифрування. NTSA вводить динамічну плутанину ключів(Dynamic Key Confusions) для кожного раунду шифрування, значно покращуючи безпеку зашифрованих даних. Цей підхід особливо корисний для середовищ Інтернету речей, де пристрої часто працюють із суворими обмеженнями ресурсів.

Ключові особливості

- Динамічна плутанина ключів: на відміну від TEA, який використовує той же ключ для всіх раундів шифрування, NTSA динамічно генерує додаткові ключі для кожного раунду. Динамічні ключі генеруються на основі основного ключа та даних, що планується зашифрувати. Цей метод значно підвищує безпеку процесу шифрування, збільшуючи складність ключа та роблячи його більш стійким до атак.
- Оптимізація для текстових файлів: NTSA спеціально оптимізовано для безпечної передачі текстових файлів у системах IoT.
- Ефективне використання обчислювальних ресурсів: хоча NTSA і пропонує розширені гарантії безпеки, цей метод шифрування залишається легким і ефективним. Це робить його придатним для використання у пристроях IoT, де кількість обчислювальних можливостей є обмеженою.

Приклад

Розглянемо сценарій в IoT системі в сфері сільського господарства, де різні датчики збирають дані про вологість ґрунту, температуру та стан рослин, які потім передаються на централізований сервер для аналізу. Дані, зібрані кожним датчиком будуть зашифровані перед передачею. Динамічний характер ключів шифрування

гарантує, що навіть у випадку перехоплення даних, розшифровка неавторизованими особами стає надзвичайно складною справою.

Наприклад, датчик температури шифрує свої показання за допомогою NTSA перед тим, як відправити їх на сервер. Якщо зловмисник перехопить ці дані, але без доступу до конкретних ключів, які використовувалися під час раундів шифрування, розшифровка даних буде нераціональною з точки зору обчислень та потенційної вигоди, тим самим захищаючи цілісність і конфіденційність переданої інформації.

Lightweight CA Cipher (LCC)

У статті за авторством Roy, S., Rawat, U., та Karjee, J. [11] представляє шифрування за назвою Lightweight Cellular Automata Based Encryption Technique – полегшене шифрування на основі клітинних автоматів.

LCC використовує принципи клітинного автомата — математичної моделі, яка складається з сітки комірок, кожна з яких перебуває в одному зі скінченної кількості станів, наприклад увімкнено або вимкнено. Стан комірки на наступному кроці часу визначається набором правил на основі поточного стану комірки та станів сусідніх комірок. LCC використовує простоту та ефективність CA, щоб забезпечити надійний, але легкий метод шифрування, який підходить для обмежених середовищ пристроїв IoT. Цей метод шифрування є симетричним.

Ключові особливості

- Ефективність і простота: алгоритм LCC розроблено таким чином, щоб бути ефективним як з точки зору обчислювальних вимог, так і з точки зору споживання енергії. Це, свою чергу, робить його особливо корисним для використання в IoT пристроях, бо вони часто живляться від батарей і мають обмежені ресурси для обробки даних.
- Безпека: дослідження показало, що LCC проходить тести на випадковість, призначені Національним інститутом стандартів і технологій (NIST), і всі

тести DIEHARD. Таким чином, незважаючи на свою полегшену імплементацію, LCC пропонує високий рівень безпеки.

- Масштабованість: хоча LCC розроблено для середовищ з обмеженими ресурсами, його також можна масштабувати для використання в більших мережах сенсорних вузлів, що робить його універсальним для широкого спектру програм в IoT.

Як це працює

У методі LCC ключем шифрування слугує наперед обраний набір правил RVList512 та кількість ітерацій(1-8). Вводом слугує 512 біт початкових даних.

Правила клітинних автоматів можуть бути згенеровані залежно від радіуса та кількості можливих значень, які клітина може мати як значення стану. У випадку одновимірного клітинного автомату з радіусом r = 1 (три можливі сусіди(лівий, центральний, правий), можна створити 256 можливих правил CA, але усі ці комбінації не утворюють правила GCA(груповий клітинний автомат). Для генерації GCA правил використовується алгоритм, який випадковим чином бере вісім правил CA(RV8) з доступних 256 правил CA, а потім генерує GCA вектор(RVList512) шляхом конкатенації. Такий вектор , що містить 512 правил CA, можна використовувати в шифруванні. Зловмисник має витратити майже експоненційний час, аби вгадати 512 правил, що зводить нанівець можливість брутфорс атаки.

Приклад

Практичне застосування LCC може бути в інфраструктурі розумного міста, де численні датчики збирають дані про транспортні потоки, умови навколишнього середовища та комунальні послуги. Кожен датчик шифрує свої дані за допомогою методу LCC перед передачею. Це гарантує, що навіть якщо дані перехоплені, вони залишаються в безпеці завдяки шифруванню. На приймальному кінці пристрої-шлюзи, які служать центральними точками для збору та аналізу даних, розшифровують дані за допомогою відповідного процесу дешифрування.

Наприклад, мережа датчиків якості повітря, розміщена по всьому місту, збирає дані про забруднюючі речовини. Перед надсиланням цих даних на центральну станцію моніторингу кожен датчик застосовує LCC для шифрування інформації. Потім зашифровані дані передаються через загальнодоступну мережу на центральну станцію, де вони розшифровуються для аналізу. Це налаштування гарантує, що конфіденційні екологічні дані залишаються конфіденційними та захищеними від несанкціонованого доступу під час передачі.

Functional Encryption (FE)

FE є узагальненням існуючих технології шифрування за допомогою відкритого ключа, такі як шифрування на основі ідентифікації (IBE), на основі атрибутів (ABE), гомоморфне шифрування (HE), шифрування предикатів (PE), шифрування з можливістю пошуку (SE) тощо [12].

Функціональне шифрування (FE) представляє зміну парадигми технології шифрування, що дозволяє більш детально контролювати доступ до зашифрованих даних. У системі FE ключі дешифрування видаються на основі певних функцій; таким чином, власники цих ключів можуть обчислювати лише певні функції над зашифрованими даними, не вивчаючи нічого іншого про дані. У застосуванні в IoT, особливо в додатках електронної охорони здоров’я, FE пропонує новий підхід до захисту конфіденційних даних пацієнтів, водночас дозволяючи постачальникам медичних послуг виконувати необхідну аналітику та обробку даних.

Ключові особливості

- Гранульований контроль доступу: функціональне шифрування дозволяє точно контролювати, які дані можна розшифрувати та обробити, залежно від дозволів, пов’язаних із ключем розшифрування. Це особливо корисно в програмах електронної охорони здоров’я, де різним організаціям може знадобитися доступ до певних частин даних про здоров’я пацієнта.
- Конфіденційність даних: функціональне шифрування гарантує, що конкретні актори можуть отримати доступ лише до тих даних, на які вони мають право.

Таким чином FE підтримує конфіденційність даних пацієнтів, що є ключовим аспектом у сфері охорони здоров’я.

- Ефективне використання даних: шифрування на основі функцій забезпечує ефективне використання даних у централізованих системах електронної охорони здоров’я з використанням IoT пристроїв.

Як це працює

У випадку системи електронної охорони здоров’я з використанням IoT пристроїв, функціональне шифрування працюватиме наступним чином:

1. Шифрування даних: дані пацієнтів, зібрані медичними пристроями Інтернету речей (наприклад, портативними моніторами здоров’я), шифруються за допомогою FE. Процес шифрування вбудовує в дані певні функції, які визначають, що можна робити з даними після їх розшифрування.
2. Розподіл ключів: ключі дешифрування видаються постачальникам медичних послуг на основі функцій, які вони повинні виконувати. Наприклад, лікар загальної практики може отримати ключ, який дає йому доступ до загальних показників здоров’я. В свою чергу інший лікар, що має вузько направлену спеціальність, отримає ключ, що надасть доступ до іншої чи більш детальної інформації про стан здоров’я пацієнта.
3. Обробка та аналіз даних: постачальники медичних послуг використовують свої ключі для розшифровки та обробки зашифрованих даних. Функціональне шифрування гарантує, що вони можуть розшифрувати лише дані, необхідні для їхніх конкретних, наперед визначених, цілей, захищаючи іншу конфіденційну інформацію.

Приклад

Розглянемо сценарій, де пацієнт носить розумний монітор здоров’я, який відстежує різні показники здоров’я, такі як частота серцебиття, рівень глюкози в крові, режим сну тощо. Цей пристрій шифрує дані за допомогою функціонального шифрування перед надсиланням їх до централізованої бази даних охорони здоров’я.

- Лікар загальної практики може мати ключ, який дозволить йому розшифровувати та переглядати загальні тенденції здоров’я та основні показники для надання загальних порад щодо здоров’я.
- В свою чергу кардіолог може мати ключ, що буде дозволяти розшифрувати детальні дані про частоту та інші параметри серцевих скорочень для більш спеціалізованої допомоги.
- Спеціаліст зі сну може отримати доступ до детальних даних про режим сну, задля діагностики розладів сну та призначення лікування.

Кожен фахівець може отримати доступ лише до даних, що стосуються його галузі, забезпечуючи конфіденційність пацієнта, а також цілеспрямовану та ефективну медичну допомогу.

Висновок до розділу 3

У світі IoT шифрування даних відіграє ключову роль у захисті конфіденційності та цілісності інформації. Через зростаючий обсяг чутливих даних, які збирають і обробляють пристрої IoT, надійне шифрування є критично важливим для запобігання витокам даних та порушенням конфіденційності. Це особливо актуально у зв'язку з обмеженістю ресурсів IoT пристроїв, що підвищує необхідність у легких та водночас ефективних методах шифрування. Розділ розглянув як перевірені часом, так і нові підходи, які можуть бути використані для підвищення безпеки IoT систем. Викладені методи та парадигми забезпечують міцну основу для створення надійних, ефективних та безпечних систем, здатних адаптуватися до різних вимог.

Розділ 4. Реагування на інциденти безпеки

Стратегії реагування

Враховуючи складність і різноманітність середовищ Інтернету речей, традиційні стратегії реагування на інциденти часто виявляються неефективними. Для ефективної та правильної реакції на випадки компрометації систем інтернету речей необхідний свій спеціальний підхід.

- Підготовка: ця фаза передбачає створення спеціального плану реагування на інциденти безпеки. Вона включає в себе визначення критичних активів, протоколів комунікації, а також створення спеціалізованих груп реагування. Задля підвищення готовності реагування на вразливості в безпеці, організації регулярно проводити аудити своїх систем та проводити навчання для своїх співробітників.
- Виявлення та аналіз: повинні використовуватися системи моніторингу для виявлення аномалій, що будуть сповіщати про інциденти порушення безпеки. Це може включати моніторинг мереж на незвичні шаблони в трафіку або аналіз поведінки пристроїв. Інструменти та методи повинні враховувати величезні обсяги даних, що генеруються IoT пристроями. Також потрібно закладати у такі інструменти розуміння про можливі помилкові спрацьовування (false positives).
- Containment, Eradication, and Recovery - стримування, знищення та відновлення. Після виявлення інциденту необхідно негайно вжити заходів для локалізації порушення. Це може включати ізоляцію уражених пристроїв або сегментів мережі. Усунення загрози включає видалення зловмисного програмного забезпечення, зміну скомпрометованих облікових даних і виправлення вразливостей. Відновлення зосереджено на безпечному відновленні служб і пристроїв до нормального робочого стану.
- Після інциденту дуже важливо провести ретельний аналіз, щоб зрозуміти, що сталося, чому існуючі засоби захисту не спрацювали та як можна запобігти

подібним інцидентам у майбутньому. Це має призвести до оновлення плану реагування на інциденти та заходів безпеки.

Проблеми реагування на інциденти в IoT системах

- Неоднорідність і велика різноманітність пристроїв інтернету речей створюють значні проблеми для моніторингу, виявлення та реагування. Це означає, що різні групи пристроїв потенційно можуть вимагати різного підходу для вищезгаданих процесів.
- Отримання повного логу операцій пристроїв і відкритого мережевого трафіку є складним завданням, що ускладнює виявлення та розуміння масштабу інцидентів.
- Дослідження інцидентів: обмежена потужність обчислювальних ресурсів та кількість постійної пам’яті багатьох пристроїв інтернету речей обмежують обсяг даних, які можна зібрати після інциденту. Крім того, потенційна фізична недоступність деяких пристроїв ускладнює аналіз, якщо є підозра на фізичний характер інциденту порушення безпеки.

Висновок до розділу 4

Унікальні характеристики IoT екосистем вимагають спеціального підходу до управління загрозами та реагування на інциденти. Розуміючи конкретні загрози, з якими стикаються пристрої та мережі IoT, і запроваджуючи систему реагування на інциденти, адаптовану до цих проблем, організації можуть підвищити свою стійкість проти кібератак. Це передбачає не лише технічні заходи, але й організаційну готовність, включаючи навчання, планування та аналіз після інциденту. Завдяки старанним зусиллям і стратегічному плануванню можна значно посилити безпеку всієї інфраструктури, захищаючи як пристрої, так і дані, які оброблюються пристроями.

Розділ 5. Розробка власної комплексної IoT системи

Всі IoT системи є суттєво залежними від власної доменної області. Наприклад, методи та підходи до управління приладами в системах розумного міста та системах домашнього будинку будуть суттєво відрізнятися. Ця різниця буде залежати від багатьох причин. Спершу, обчислювальна здатність приладів, кількість оперативної та постійної пам’яті прямо диктують обмеження для всіх інших частин системи. Від цих параметрів залежить як спосіб інтеркомунікації між приладами, так і спосіб комунікації з централізованими серверами. Чи кожний прилад спілкується напряму один з одним та з сервером, чи існують проміжні вузли, чи прилади покладаються на сервер для комунікації один з одним. А це вже в свою чергу диктує методи та підходи до шифрування, деякі варіанти з яких викладені у вищенаведених розділах. До того ж, останнє, але не по значенню – авторизація та автентифікація. Вони можуть приймати різні вигляди в залежності від обраного підходу до інтеркомунікації та шифрування даних.

Таким чином, було прийнято рішення розробити систему, що може слугувати фундаментом для конкретних рішень. Запропонований фреймворк надає базовий функціонал, що є легко розширюваним, та бачення безпеки в кінцевій системі. Система складається з наступних компонентів: розподільник навантаження Nginx, брокер повідомлень MQTT HiveMQ, сервер авторизації, сервер збору інформації, збірка сервісів ELK Stack.

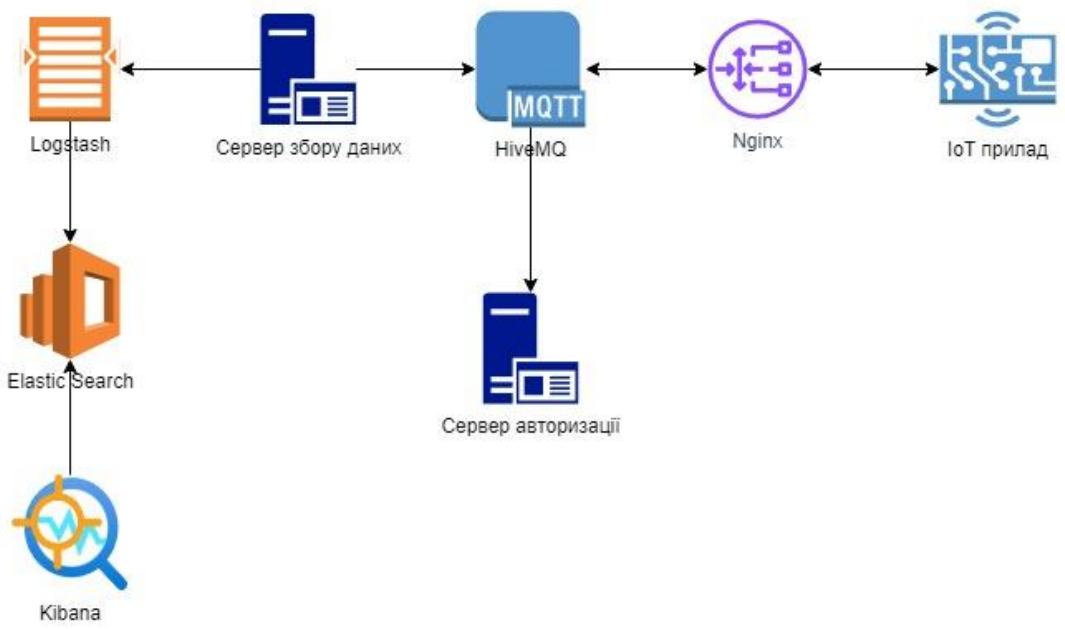

Рис. 7. Архітектура розробленої системи

Nginx

Nginx — це open-source програмне забезпечення для веб-сервера, яке, в загальному, використовується для зворотного проксі-сервера, балансування навантаження та кешування. Воно надає можливості HTTPS сервера і розроблене для максимальної продуктивності та стабільності. Воно також функціонує як проксі-сервер для протоколів електронної пошти, таких як IMAP, POP3 і SMTP.

Nginx використовує master-slave архітектуру, шляхом імплементації асинхронного, неблокуючого та керованого подіями підходу. ПЗ використовує мультиплексування та сповіщення про події та присвячує окремі завдання окремим процесам. Наприклад, якщо існує 20 завдань, 20 різних процесів оброблятимуть кожне з них.

- Nginx не створює новий потік виконання для кожного нового з’єднання. Натомість спеціальний майстр процес обробляє прийом нових запитів із загального сокету прийому даних та запускає високоефективні цикли

виконання всередині окремих worker процесів. Це дозволяє обробляти тисячі з’єднань в межах одного такого процесу. Спеціальних механізмів розподілу з’єднань між різними worker процесами в Nginx не існує, ця робота виконується на рівні ядра ОС.

- Майстер процес розподіляє запити для робочих процесів відповідно до вимог клієнта. Після того, як робота буде розподілена між робітниками, майстер очікуватиме наступний запит від клієнта, тобто не не буде чекати на відповідь від робітників. Після отримання асинхронної відповіді від робітника майстер процес надішле відповідь клієнту. Також майстер читає та перевіряє конфігурацію серверу шляхом створення, приписування та з’єднання сокетів. Він також обробляє запуск, завершення та підтримку кількості налаштованих робочих процесів. До того ж, він може переналаштувати робочий процес без переривання обробки запитів.
- Проксі-кеші — це спеціальні процеси. Вони мають завантажувач кешу та менеджер. Завантажувач кешу перевіряє елемент дискового кешу та заповнює in-memory базу даних метаданими кешу. Він готує Nginx до роботи з файлами, які вже зберігаються на диску. Менеджер кешу відповідальний за час життя елементів та інвалідацію елементів у кеші.

У розробленій системі Nginx виконує роль реверс-проксі та слугує точкою входу для IoT приладів до MQTT брокера та точкою входу користувачам до інших частин системи. Такий підхід спрощує роботу з та підтримку TLS сертифікатів.

Сервер авторизації

В запропонованій системі цей сервер виконує функції авторизації та автентифікації IoT пристроїв. Сервер розроблено у вигляді модуля мовою Java для Spring фреймворку.

Spring вважається безпечною, недорогою та гнучкою платформою, яка покращує ефективність написання коду та скорочує загальний час розробки програми за рахунок ефективного використання системних ресурсів. Spring усуває

виснажливу роботу з налаштування, щоб розробники могли зосередитися на написанні бізнес-логіки [13].

Крім того, Spring керує інфраструктурою, щоб розробники могли зосередитися на розробці високоякісних, корисних і масштабованих корпоративних програм, не турбуючись про середовища розгортання та інші інфраструктурні проблеми.

Також Spring надає численні слабко зв’язані модулі для покращення функціональності веб-додатку. Ці модулі, такі як Spring AOP, Spring Security, Spring Session, Spring MVC, Spring Data та інші можна використовувати окремо або разом, забезпечуючи більшу гнучкість під час розробки.

Розроблена бібліотека виконує наступні дві функції:

1. Постачання облікових даних від MQTT брокера для IoT пристроїв
2. Автентифікація та авторизація IoT пристроїв на MQTT брокері

Облікові дані

У розділі «Проблеми автентифікації та авторизації» вже згадувалося наскільки вразливі облікові дані впливають на безпеку всієї системи. Запропонований підхід має на меті усунути проблему доставки та ротації облікових даних на IoT пристроях. Для досягнення такого результату потрібно аби попередньо виконувалися наступні умови:

1. Сервер авторизації має інформацію про серійний номер всіх випущених IoT пристроїв, які потрібно підтримувати
2. Всі IoT пристрої мають вбудованими в прошивку домен та публічні TLS ключі серверу

Задля виконання першого пункту розробникам потрібно імплементувати та використовувати наступний інтерфейс.

```
public interface IoTDeviceCrudService {

    boolean save(IoTDevice device);
    boolean deleteBySerialNumber(String serialNumber);
    Optional<IoTDevice> findBySerialNumber(String serialNumber);
    Optional<IoTDevice> findByMqttLogin(String mqttLogin);
}
```

Надається стандартна імплементація, що зберігає дані в оперативній пам’яті. Це можливо використовувати для тестування. Такий підхід дозволяє розробникам зберігати дані про пристрої в будь-якому для них зручному форматі, наприклад в SQL/NoSQL базах даних тощо.

Дані потрібно зберігати в наступному форматі.

```
public class IoTDevice {

    private String serialNumber;
    private String mqttLogin;
    private String mqttPassword;
    private Date mqttCredentialsLastRotation;
    private boolean blacklisted = false;
    private Set<String> topics = new HashSet<>();

}
```

Вважається, що сутність IoTDevice з’явиться в системі до продажі пристроїв клієнтам. Спосіб генерації логіну та паролю віддається на розсуд розробників. Ознака blacklisted відповідає за те, чи дозволено тому чи іншому пристрою отримувати нові облікові дані. Список topics відповідає за дозволені для зчитування топіки цього пристрою.

Для IoT пристроїв надається два HTTPS ендпоінти:

1. /api/iot/device/credentials – прилад надсилає свій серійний номер, і якщо всі перевірки проходять успішно, сервер відповідає актуальними обліковими даними для цього пристрою.

```
@GetMapping(value = "/credentials")
public Response<DeviceAuthorizeResponse>
getCredentials(@RequestParam("serial_number") String serialNumber) {
    return Response.of(iotDeviceService.getCredentials(serialNumber));
}
```

Абстрактний сервіс IoTDeviceService відповідає за знаходження актуальних даних.

```
public DeviceAuthorizeResponse getCredentials(String serialNumber) {
    if (canReturnCredentials(serialNumber)) throw new ForbiddenException();
    var device =
iotDeviceCrudService.findBySerialNumber(serialNumber).orElseThrow(()->new
NotFoundException());
    if (!device.hasCredentials()) {
        device = createCredentials(device);
    }
    return new DeviceAuthorizeResponse(mqttCredentialsConfig.getDomain(),
device.getMqttLogin(), device.getMqttPassword());
}


private IoTDevice createCredentials(IoTDevice device){
    device.setMqttLogin(UUID.randomUUID().toString());
    device.setMqttPassword(UUID.randomUUID().toString());
    device.setMqttCredentialsLastRotation(new Date());
    return iotDeviceCrudService.save(device);
}

public abstract boolean canReturnCredentials(String serialNumber);
```

Абстрактний метод canReturnCredentials вимагає імлпементації, задля визначення конкретної бізнес логіки, на основі якої і приймається рішення, чи надавати актуальні облікові дані. Метод createCredentials встановить нові облікові дані. Нарешті, метод getCredentials знайде відповідний прилад, перевірить його наявність та доступ, за необхідності викличе createCredentials та поверне облікові дані.

2. /api/iot/device/certificate – прилад запитує актуальний публічний ключ TLS для Nginx/HiveMQ

```
@GetMapping(value = "/certificate")
public Response<String> getCredentials() {
    return Response.of(nginxConfig.getCertificatePubKey());
}
```

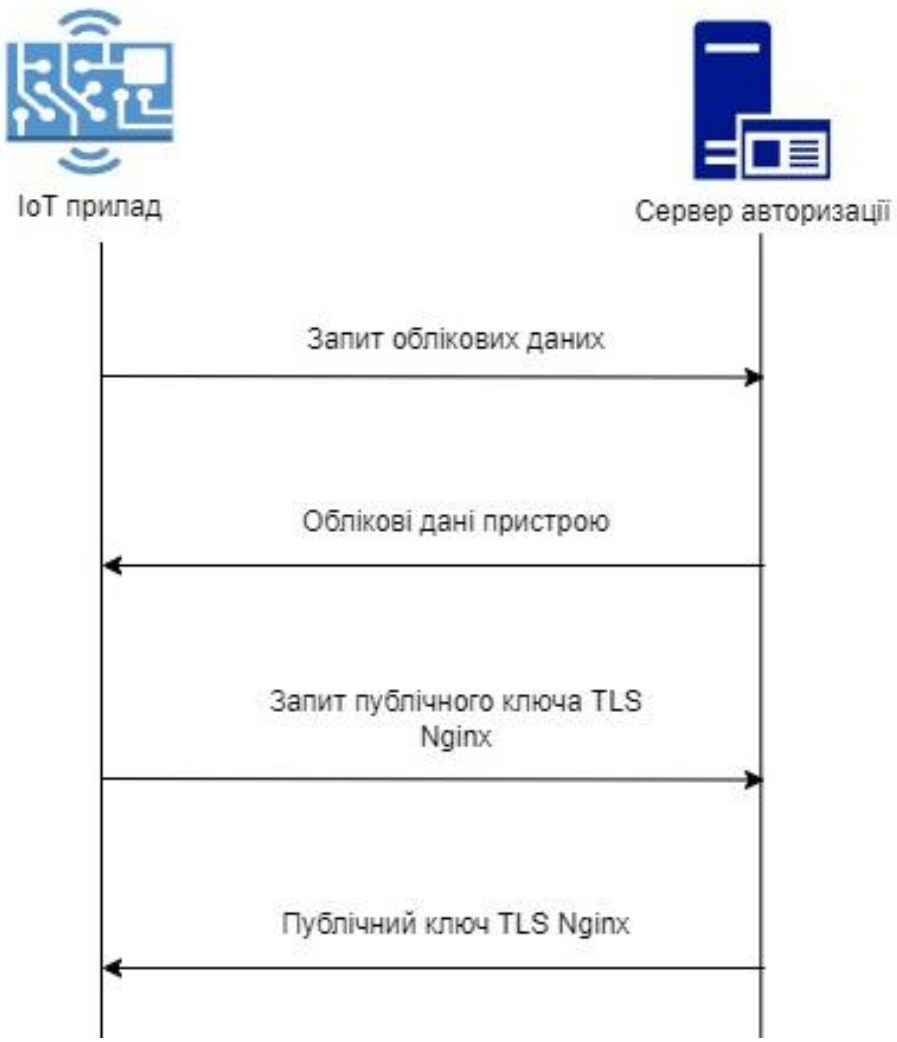

Рис. 8. Процес отримання облікових даних приладом

Для HiveMQ теж надається два HTTPS ендпоінти, що є захищеними наперед обраним паролем:

1. /api/iot/mqtt/authenticate – брокер надсилає логін та пароль, отриманий від IoT пристрою, та очікує на відповідь про дозвіл на автентифікацію

```
@PostMapping(value = "/authenticate")
public Boolean authenticate(@RequestBody AuthenticateRequest request) {
    if
(!SecurityContextHolder.getContext().getAuthentication().getAuthorities().con
tains(new HiveMQAuthority()))
        throw new ForbiddenException();

    return hiveMQAuthentificationService.authenticate(request.login(),
request.password());
}
```

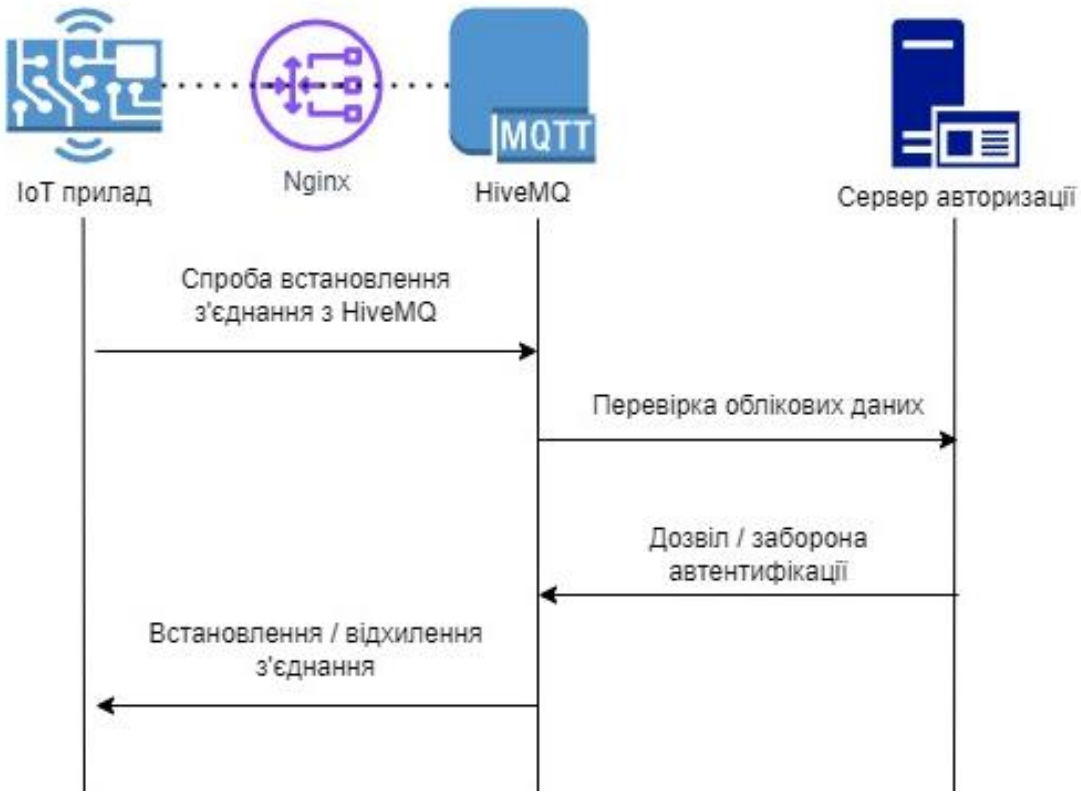

Рис. 9. Процес автентифікації приладу

2. /api/iot/mqtt/authorize – брокер надсилає логін автентифікованого пристрою та топік, з яким пристрій хоче взаємодіяти (писати чи читати)

```
@PostMapping(value = "/authorize")
public Boolean authorize(@RequestBody AuthorizeRequest request) {
    if
(!SecurityContextHolder.getContext().getAuthentication().getAuthorities().con
tains(new HiveMQAuthority()))
        throw new ForbiddenException();

    return hiveMQAuthentificationService.authorize(request.login(),
request.topic());
}
```

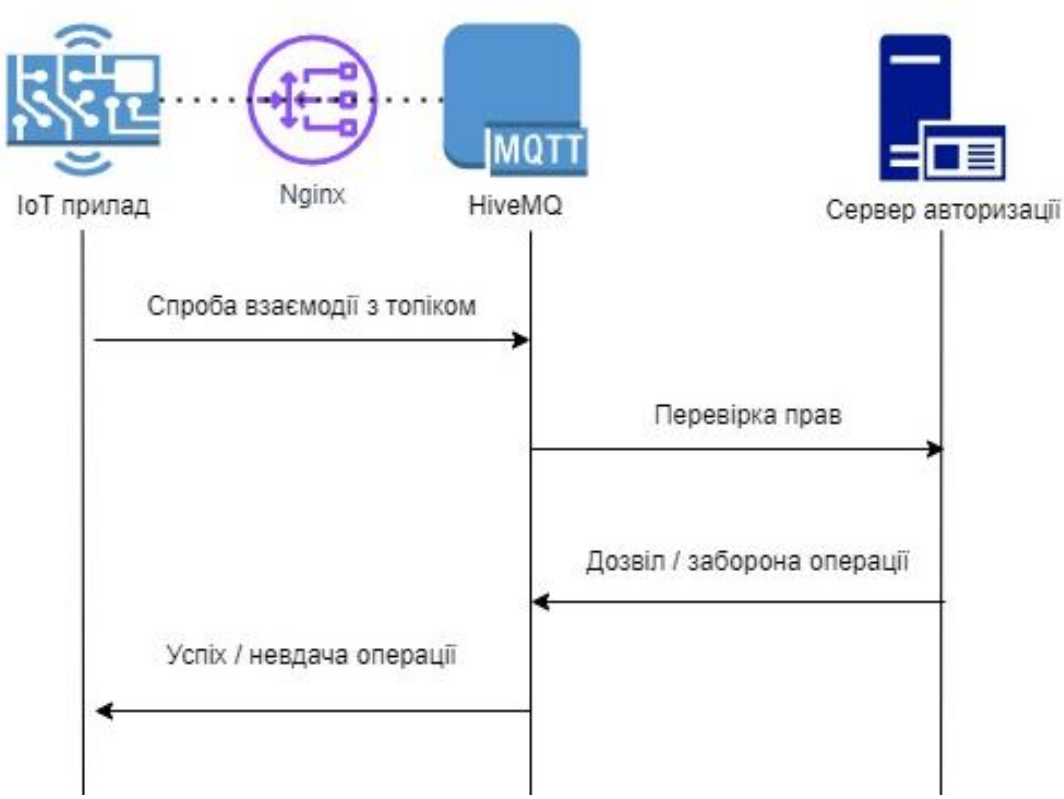

Рис. 10. Процес авторизації приладу

Важливими компонентом безпеки в запропонованій системі є механізм certificate(public key) pinning. Його основна мета — підвищити безпеку з'єднання шляхом зниження ризику атак типу «людина посередині» (MITM) і забезпечення зв'язку клієнта лише з нашим автентичним сервером.

Стандартна перевірка сертифіката: у типовому TLS handshake, коли клієнт підключається до сервера, сервер представляє клієнту свій цифровий сертифікат. Потім клієнт перевіряє автентичність сертифіката, перевіряючи, що він підписаний

довіреним центром сертифікації (CA) і що його термін дії не закінчився або він не був відкликаний. Якщо перевірки проходять успішно, клієнт продовжує безпечне з’єднання.

Закріплення сертифікату: цей механізм покращує процес встановлення довіри. Замість того, щоб покладатися виключно на систему CA, клієнт має попередньо налаштований список відкритих ключів або сертифікатів, яким він явно довіряє.

Таким чином, мікропрограма пристроїв має мати перелік попередньо згенерованих публічних ключів серверу авторизації для запобігання атаки MITM. Публічний ключ точки входу до MQTT брокера буде отримано вже від серверу авторизації.

HiveMQ

HiveMQ є брокером повідомлень протоколу MQTT [14].

Протокол MQTT

Протокол має два типи об'єктів: брокер повідомлень і клієнт. Брокер MQTT функціонує як сервер, який приймає повідомлення від клієнтів і розповсюджує їх іншим клієнтам. Клієнтом MQTT може бути будь-який пристрій, який використовує бібліотеку MQTT та підключається до брокера через мережу.

Інформація в MQTT структурована у вигляді ієрархії топіків. Коли видавець(publisher) має нові дані, він надсилає повідомлення брокеру на відповідний топік. Далі це повідомленн вичитується підписниками(subscriber) цієї теми. Видавцеві не потрібно знати, хто саме підписаний на його повідомлення, а підписникам не потрібно знати, звідки приходять повідомлення.

Якщо брокер отримує повідомлення на тему без підписників, він його відхиляє. Однак, видавець може позначити повідомлення як збережене, щоб брокер зберіг його для майбутніх підписників. Нові підписники отримують збережене повідомлення відразу після підписки. Збережене повідомлення містить спеціальний прапорець, що вказує про його статус. Підписникам надсилається

тільки останнє таке повідомлення, всі попередні ігноруються. До того ж, можливо налаштувати повідомлення за замовчуванням, яке буде надіслано підписникам у разі його несподіваного відключення від брокера.

MQTT підтримує передачу даних через TCP, але існує також MQTT-SN, що працює через інші транспортні протоколи, такі як UDP або Bluetooth. MQTT надсилає облікові дані підключення у вигляді звичайного тексту, що означає відсутність вбудованих заходів безпеки або автентифікації. Для забезпечення безпеки можна використовувати TLS.

Існує чотирнадцять типів повідомлень MQTT, які відповідають за різні функції, включаючи підключення та відключення клієнтів, публікацію та підтвердження отримання даних, а також управління з'єднанням між клієнтом і сервером. Обсяг даних в одному повідомленні може варіюватися від двох байтів до 256 мегабайт.

Кожний клієнт має вказати QoS – Quality of Service(якість з’єднання). Цей показник класифікуєься в порядку збільшення накладних витрат:

- Щонайбільше один раз – повідомлення надсилається лише один раз, і клієнт і брокер не вживають жодних додаткових кроків для підтвердження доставки
- Принаймні один раз – відправник повторює повідомлення кілька разів, доки не буде отримано підтвердження
- Рівно один раз – відправник і одержувач беруть участь у дворівневому рукостисканні, щоб забезпечити отримання лише однієї копії повідомлення

QoS не впливає на порядок передачі даних в TCP. Він є надбудовою протоколу MQTT.

Існує 4 основні типи повідомлень в MQTT:

- CONNECT: використовується для надсилання клієнтами запитів на підключення до брокера
- PUBLISH: використовується видавцями для надсилання повідомлень брокеру

- SUBSCRIBE: використовується підписниками для отримання повідомлень від брокера
- DISCONNECT: використовується клієнтами для припинення комунікації.

Протокол MQTT має декілька версій, найновішою з яких є MQTT 5.0. Вона вводить декілька нових змін, найбільші з яких: ACK повідомлення повертають коди помилок, об'єднані підписки на топіки, час життя для недоставлених повідомлень та псевдоніми для топіків.

Брокер HiveMQ

HiveMQ — MQTT брокер і платформа обміну повідомленнями розроблена мовою Java, яка використовує протокол MQTT для швидкої, надійної та ефективної двонаправленої передачі даних до та з пристроїв IoT. HiveMQ надає власну клієнтську бібліотеку, але її можна використовувати з будь-якою клієнтською бібліотекою, сумісною з протоколом MQTT. Він може бути розгорнутий у приватній, гібридній або публічній хмарі. Можливо інтегровувати HiveMQ з існуючими корпоративними системами завдяки його відкритому API та гнучкому розширенню. Також від розробників HiveMQ існує інструмент під назвою MQTT CLI, який надає інтерфейс командного рядка для взаємодії з брокерами MQTT, в нашому випадку з HiveMQ. Така утиліта є корисною, бо дозволяє відкрити декілька інстансів одночасно. А це є зручним для тестування.

HiveMQ було обрано через наявність кластеризації та зручні можливості для розширення функціоналу брокеру шляхом написання плагінів.

Кластер MQTT брокерів — це розподілена система, яка діє як єдиний логічний брокер MQTT для підключених клієнтів. Як правило, вузли кластера MQTT встановлюються на окремих фізичних або віртуальних машинах і підключаються через мережу. Такий підхід гарантує, що розгорнута система не матиме єдиної точки збою. В інфрастуктурі, де доступність є надзвичайно важливою, кластеризація є необхідним компонентом. У випадку коли один MQTT брокер у кластері більше не доступний, решта вузлів можуть продовжувати

обробляти трафік. В HiveMQ реалізовано архітектуру без майстра, що дозволяє мати необмежену кількість нод.

Розширення для HiveMQ

HiveMQ пропонує Extension SDK для розробки власних розширень, що можуть виконувати такі функції як:

- Перехоплення та маніпуляція повідомленнями MQTT
- Інтеграція з іншими сервісами
- Збір статистики
- Розширення управління доступом
- Тощо

В запропонованій системі увага приділяється першій та четвертій функціям.

Імплементуючи інтерфейс SimpleAuthenticator ми реалізуємо метод onConnect (SimpleAuthInput simpleAuthInput, SimpleAuthOutput simpleAuthOutput). В цьому методі замість простої перевірки ACL файлу, що порядково містить дані про доступи, виконується запит до серверу авторизації для отримання актуальної інформації про статус клієнта, що намагається здійснити підключення.

```
@Override
public void onConnect(@NotNull SimpleAuthInput simpleAuthInput, @NotNull
SimpleAuthOutput simpleAuthOutput) {
    String clientId = simpleAuthInput.getConnectPacket().getClientId();
    var passwordBytes = simpleAuthInput.getConnectPacket().getPassword();
    if (passwordBytes.isEmpty()) simpleAuthOutput.failAuthentication();

    CharBuffer charBuffer =
StandardCharsets.US_ASCII.decode(passwordBytes.get());
    String password = charBuffer.toString();

    if (checkAuthentication(clientId, password)) {
        simpleAuthOutput.authenticateSuccessfully();
    } else {
        simpleAuthOutput.failAuthentication();
    }
}
```

Імплементуючи інтерфейс PublishAuthorizer ми реалізуємо метод authorizePublish(PublishAuthorizerInput input, PublishAuthorizerOutput output). Знову, замість простої перевірки ACL файлу, виконується запит до серверу

авторизації для отримання актуальної інформації про доступні топіки для автентифіковагого клієнта.

```
@Override
public void authorizePublish(@NotNull PublishAuthorizerInput input, @NotNull
PublishAuthorizerOutput output) {
    PublishPacket publishPacket = input.getPublishPacket();
    String topicName = publishPacket.getTopic();
    boolean isAuthorized =
checkAuthorization(input.getClientInformation().getClientId(), topicName);

    if (isAuthorized) {
        output.authorizeSuccessfully();
    } else {
        output.failAuthorization();
    }
}
```

Дані отримані в обидвох методах можливо кешувати для збільшення швидкодії. Це можливо реалізувати як і встановленням часу життя на отримані дані, так і більш складними підходами, такими як інвалідація кешу, що буде ініційована сервером авторизації.

Додатково ми імплементуємо інтерфейс PublishInboundInterceptor для дублювання трафіку у визначений топік. Це потрібно для зручної передачі даних до Elastic Search та подальшої їхньої обробки.

```
@Override
public void onInboundPublish(final PublishInboundInput publishInboundInput,
                             final PublishInboundOutput publishInboundOutput)
{
    final ModifiablePublishPacket publishPacket =
publishInboundOutput.getPublishPacket();
    final PublishService publishService = Services.publishService();

    // Duplicate the message only if it's not already to "traffic/data"
    if (!publishPacket.getTopic().equals(CONFIG.logstashTopic()) &&
publishPacket.getPayload().isPresent()) {
        publishService.publish(Builders.publish()
                                       .topic(CONFIG.logstashTopic())

.payload(publishPacket.getPayload().get())
                                       .qos(publishPacket.getQos())
                                       .retain(publishPacket.getRetain())
                                       .build());
    }
}
```

Для забезпечення конфіденційності та цілісності даних використовується TLS.

Сервер збору інформації

Сервер написано мовою Java на платформі Spring з використанням бібліотеки spring-integration-mqtt. Основною задачою сервісу є вичитка даних з MQTT брокеру та надсилання їх до Logstash. Додатково, тут можливо робити всі необхідні трансформації та фільтрування.

В поточний реалізації, Logstash підіймає TCP сервер та очікує підключення від серверу збору інформації. Проте, варто зазначити, що таку фукнцію можливо було б реалізувати інакше. Logstash, як і HiveMQ, надає широкий інструментарій для розширення системи. А саме як інформація надходить, як оброблюється і як надсилається далі. Більше того, вже існує 3 розширення, які мали б дозволити зчитувати повідомлення напряму з MQTT брокера. Проте, останнє оновлення найновішого з них датується 2018 роком. Після багатьох невдалих спроб інтеграції цих розширень, стало зрозуміло, що потрібно розробити щось нове. Розширення для Logstash пишуться мовою Ruby. Задля спрощення розробки і тестування було прийнято рішення не розроблювати нове розширення для Logstash чи намагатися виправити старі версії, а написати власний сервіс на Spring, що виконуватиме тіж функції.

Сервер вичитує дані з MQTT за допомогою класів IntegrationFlow та MessageProducerSupport.

```
@Bean
public IntegrationFlow mqttInFlow(MessageProducerSupport mqttInbound) {
    return IntegrationFlow.from(mqttInbound)
            .transform(p -> p)
            .handle(message ->
messageConsumer.consume(message.getPayload().toString()))
            .get();
}

@Bean
public MessageProducerSupport mqttInbound(MqttPahoClientFactory factory,
                                          @Value("${mqtt.topic}") String
topic,
                                          @Value("${mqtt.clientId}") String
clientId) {
    MqttPahoMessageDrivenChannelAdapter adapter = new
MqttPahoMessageDrivenChannelAdapter(clientId, factory, topic);
    adapter.setCompletionTimeout(5000);
    adapter.setConverter(new DefaultPahoMessageConverter());
    adapter.setQos(1);
    return adapter;
}
```

Далі сервіс MqttLogstashMessageConsumer надсилає дані на Logstash.

```
@Override
public void consume(String s) {
    restoreConnection();
    try {
        DataOutputStream os = new DataOutputStream(new
BufferedOutputStream(socket.getOutputStream()));
        os.writeBytes(s + "\n");
        os.flush();
    } catch (Exception e){
        log.error(e.getMessage());
    }
}
```

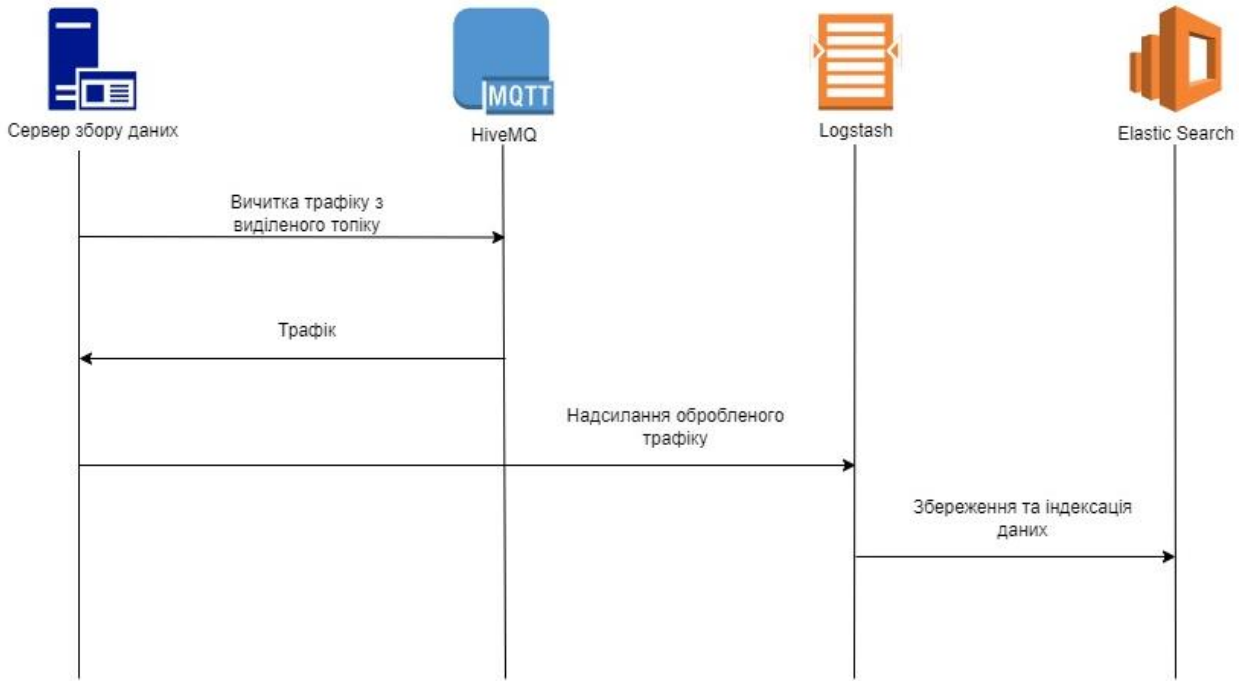

Рис. 11. Процес збору інформації

ELK Stack

Logstash

Logstash — це потужний інструмент, який відіграє важливу роль в ELK стеку [15]. Він дозволяє користувачам збирати, обробляти та аналізувати дані з різних джерел. За своєю суттю Logstash — це пайплайн для даних, який допомагає організаціям отримувати цінну інформацію зі своїх логів шляхом перетворення необроблених даних у формат, який можна легко використовувати та аналізувати. Сильною стороною Logstash є його здатність збирати дані з різноманітних джерел та в різних довільних форматах: файли, журнали подій додатків чи пристроїв, інші джерела даних. Така універсальність дозволяє організаціям збирати всі свої логи чи інші дані в одному місці, що є корисним для визначення тенденцій, виявлення аномалій та розбору проблем.

Logstash також надає зручний механізм фільтрації та конвертації, які дозволяють користувачам видаляти небажані дані, перетворювати дані у відповідний формат і агрегувати їх на основі певних критеріїв. Така функціонал

надає можливість розробникам знаходити корисну інформацію серед великої кількості даних: виявляти вузькі місця чи загрози безпеці, тощо.

Гнучкість Logstash також є однією з його найбільших переваг. Завдяки широкому спектру плагінів введення, фільтрування та виводу користувачі можуть налаштувати інструмент відповідно до своїх потреб. Незалежно від того, чи йдеться про інтеграцію Logstash з іншими інструментами стеку ELK, як Elasticsearch чи Kibana, чи використання його для надсилання даних до інших місць призначення, як Kafka чи Redis, Logstash забезпечує високий ступінь гнучкості.

В розробленій системі Logstash виконує роль пайплайну до Elasticsearch. Він отримує дані від серверу збору інформації та надсилає їх далі. Також тут можливо скористатися всіма потрібними плагінами для попередньої обробки даних, якщо на це є потреба. Використовується наступний конфіг:

```
input {
        beats {
                port => 5044
        }

        tcp {
                port => 50000
                codec => json
        }
}

fiter {

}

output {
        elasticsearch {
                index => "iot-index"
                hosts => "elasticsearch:9200"
                user => "logstash_internal"
                password => "${LOGSTASH_INTERNAL_PASSWORD}"
        }
}
```

Elasticsearch

Elasticsearch є потужною пошуковою системою, яка пропонує масштабовані рішення для повнотекстового пошуку [15]. Інструмент побудовано на принципах розподілених систем, призначених для обробки великих обсягів даних на кількох вузлах у кластері. Це досягається за допомогою механізму шардингу, коли

проіндексовані дані поділяються на менші, більш керовані частини, кожна з яких зберігається на різних вузлах. Це не тільки забезпечує масштабованість, але й підвищує відмовостійкість і доступність завдяки реплікації цих сегментів на різних вузлах у кластері.

Однією з найважливіших особливостей Elasticsearch є його здатність пошуку в реальному часі. На відміну від традиційних систем пошуку, які індексують дані пакетами, Elasticsearch може обробляти потоки даних у міру їх надходження. Це дозволяє користувачам виконувати пошук у контексті найновішої інформації і є особливо корисним для таких застосувань, як аналіз журналів подій, де актуальність даних є важливою для негайного аналізу та прийняття рішень. Мова запитів Elasticsearch одночасно потужна та гнучка. Інструмент надає можливість робити запити у багатьох форматах: пошук за ключовими словами, пошук по складних фразах і регулярних виразах. До того ж, існує можливість пошуку по повнотекстових запитах, які аналізують контекст і семантику термінів. Ще одною великою перевагою використання Elasticsearch є можливість інтеграції з Kibana, що є інструментом візуалізації. Він дозволяє аналізувати отримані дані від пошукової системи в зручному візуальному форматі.

Система підрахунку релевантності є ще одним важливим компонентом. Вона використовує частоту термінів, зворотну частоту документа та довжину поля для ранжування документів. Це робиться для того, аби користувачі спершу отримували найрелевантніші результати пошуку.

Elasticsearch поставляється разом з широким набором API, що дозволяє інтегруватися з будь-якими іншими системами. Це робить його зручним для розробників, які працюють у різних технологічних стеках.

Машинне навчання

Однією з найголовніших причин для включення Elasticsearch в розроблену систему є широкі можливості та простота використання інструменту в сфері машинного навчання.

Виявлення аномалій є одним із найпотужніших застосувань машинного навчання в Elasticsearch. Такий механізм дозволяє користувачам знаходити незвичні закономірності або просто незвичності в своїх даних, які можуть вказувати на потенційні проблеми. Це особливо корисно для виявлення моніторингу безпеки мережі, моніторингу здоров'я вузлів чи пристроїв.

Система виявлення аномалій в Elasticsearch працює шляхом постійного аналізу вхідних даних у порівнянні з історичними закономірностями. Задачі машинного навчання використовують методи неконтрольованого навчання на історичних даних, щоб створити базову статистичну модель (z-оцінки / XGBoost) того, що вважається «нормальним». Коли модель навчена, вона може відстежувати нові дані в режимі реального часу, щоб виявляти відхилення. Користувачі можуть налаштувати чутливість системи виявлення, тобто наскільки суворим чи м'яким вони хочуть, щоб виявлення аномалій було.

Найважливішим компонентом виявлення аномалій в Elastisearch є навчання в режимі реального часу, тобто здатність ML моделей поступово оновлювати свою базу знань у міру появи нових даних без необхідності перенавчання з нуля. Це означає, що в міру розвитку поведінки користувачів або системних показників ML моделі адаптуються та продовжують забезпечувати точне виявлення аномалій. Навчання в режимі реального часу має вирішальне значення для підтримки актуальності та ефективності системи виявлення аномалій, особливо в динамічних середовищах, де тенденції можуть швидко змінюватися.

Kibana

Kibana є важливою частиною ELK стеку [15]. Вона пропонує набір інструментів, розроблених, аби допомогти користувачам візуалізувати, досліджувати та взаємодіяти зі своїми даними, що зберігаються в Elasticsearch. Вона діє як інтерфейс, за допомогою величезний набір даних можна перетворити на практичну інформацію, що дозволяє компаніям і аналітикам приймати обґрунтовані рішення на основі інформації в реальному часі.

Kibana є потужним інструментом візуалізації даних, який дозволяє користувачам створювати діаграми та графіки, формат яких найкраще відповідає тим чи іншим даним. Вона підтримує розширені візуалізації, такі як криві часових рядів, діаграми розсіювання, теплові та геопросторові карти. Візуалізації не статичні, а динамічні та інтерактивні. Поєднуючи все вищезгадане, Kibana може задовольнити будь-яким варіантам використання.

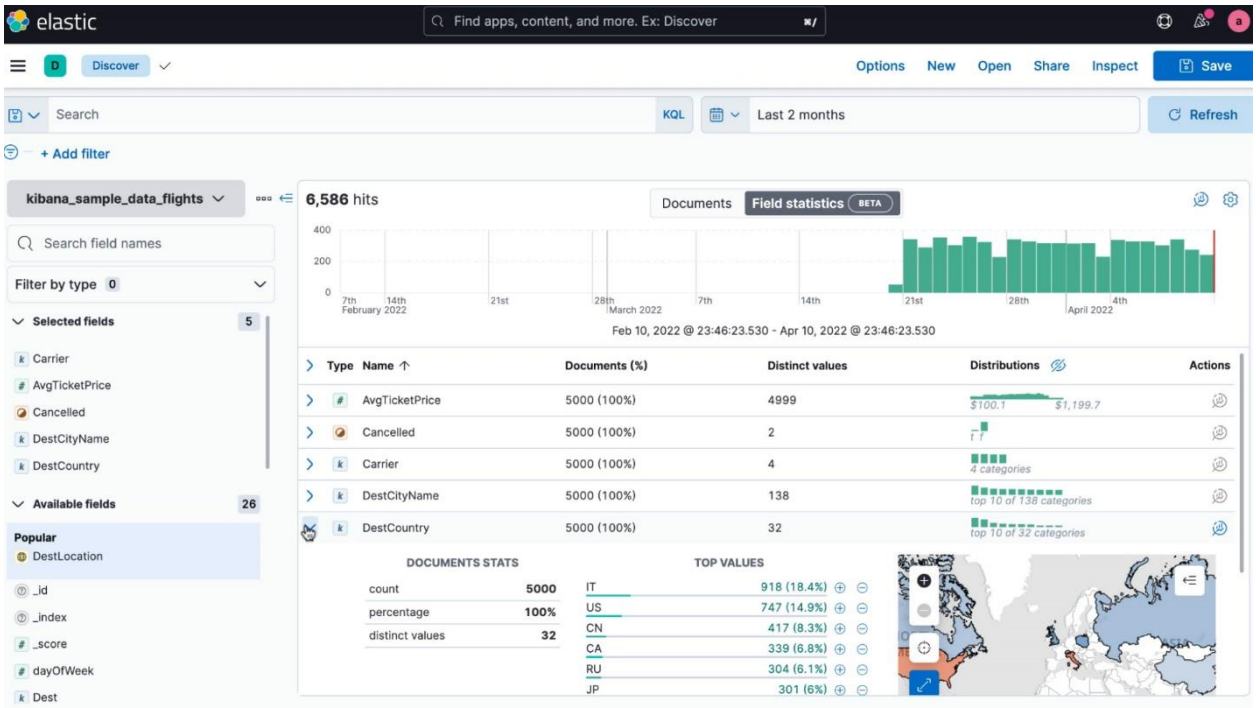

Рис. 12. Приклад сконфігурованого інтерфейсу Kibana

Однією з видатних особливостей Kibana є її здатність подавати складні дані інтуїтивно зрозумілим способом. Використовуючи пошукові можливості Elasticsearch, Kibana може миттєво фільтрувати мільйони записів для відображення найбільш релевантних результатів. Ця інтерактивність у режимі реального часу має вирішальне значення для моніторингу систем, відстеження логів подій або аналізу моделей поведінки користувачів у міру їх виникнення. Більше, Kibana має бездоганну інтеграцію з Elasticsearch. Науковці та дата-аналітики можуть використовувати Kibana для дослідження індексованих даних безпосередньо з Elasticsearch, застосовуючи фільтри, запити та агрегації, не виходячи з інтерфейсу візуалізації. Такий тісний зв’язок гарантує, що користувачі завжди працюють з

актуальною інформацією, оскільки запити Kibana виконуються з живими даними в режимі реального часу.

Гнучкість платформи є ще одним ключовим активом. Kibana дозволяє користувачам налаштовувати свої інформаційні панелі, а саме: обирати візуалізації та впорядковувати їх у макеті. Такий підхід дозволяє користувачам підлаштовувати інструмент під свою доменну область та робочий процес. Цими інформаційними панелями можна ділитися з членами команди або взагалі будь з ким, якщо на це є потреба.

На додаток до всього вище, Kibana надає набір інструментів для керування кластерами Elasticsearch. Користувачі можуть контролювати працездатність систем, керувати кластерами та оптимізувати розподіл ресурсів. Цей аспект Kibana особливо цінний для DevOps команд, яким необхідно забезпечити безперебійну роботу своєї інфраструктури.

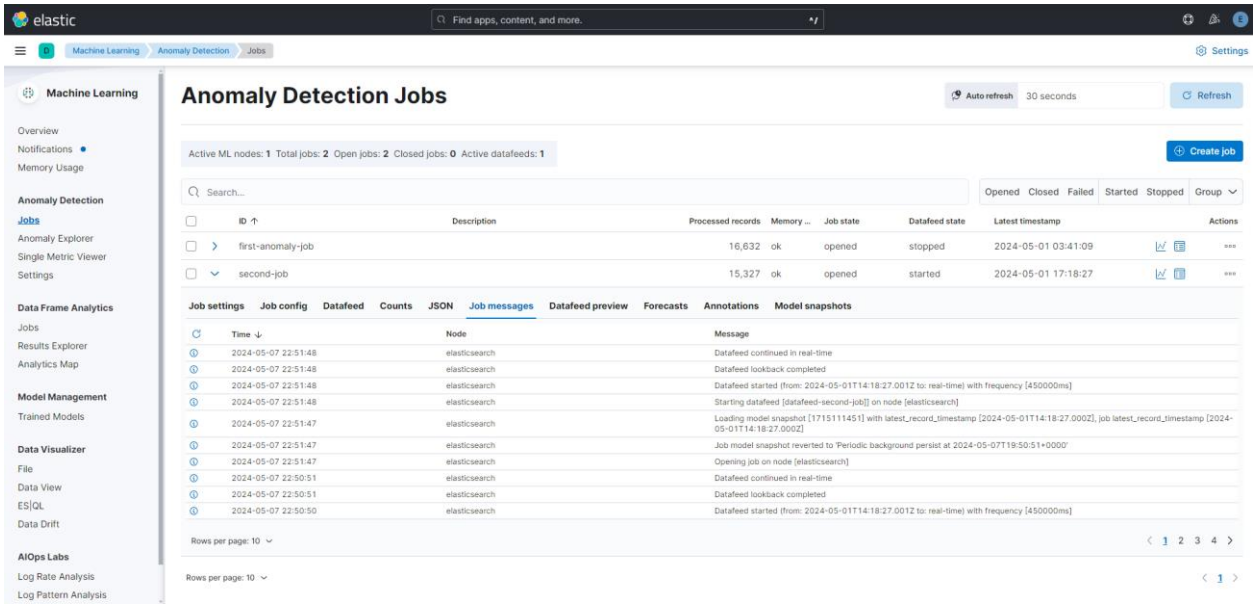

Рис. 13. Екран огляду завдань по виявленню аномалій

Як було вже вище згадано, Kibana пропонує зручний інтерфейс для взаємодії з Elastisearch, в тому числі роботу з ML моделями.

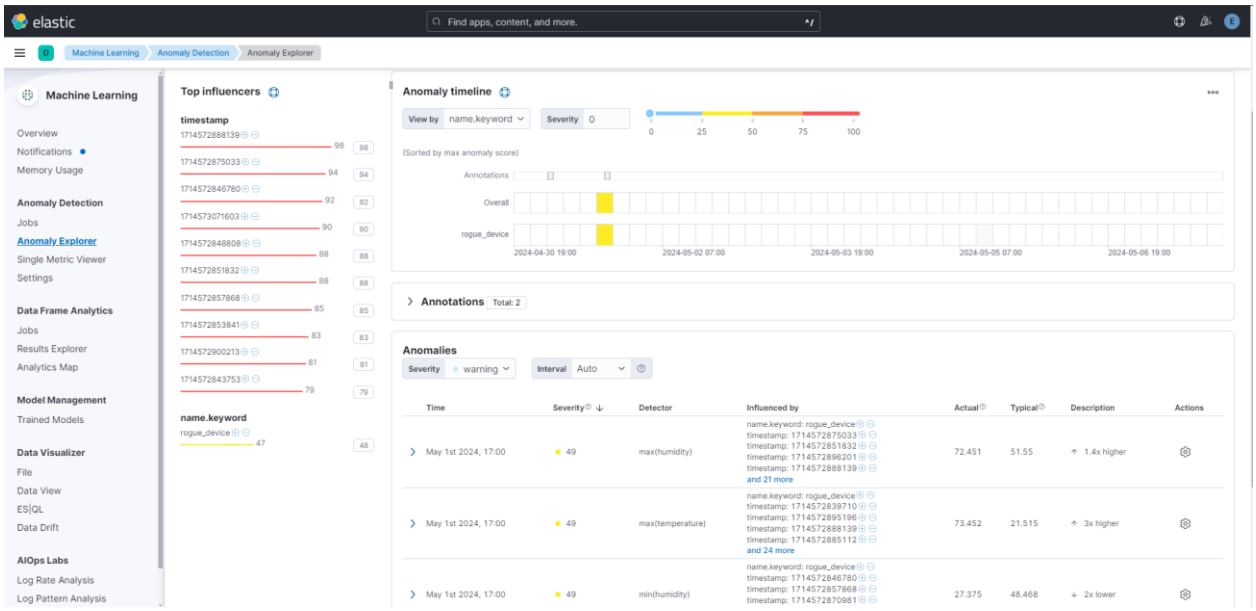

Рис. 14. Екран огляду виявлених аномалій

Для демонстрації було налаштовано і натреновано задачу виявлення аномалій. В цьому випадку у роботі приймали участь два змодельованих пристрої, які надсилали інформацію зі своїх зчитувачів, а саме: температуру навколишнього середовища та вологість повітря. Спочатку, робочий пристрій надіслав 15000 пакетів з консистентними та очікуваними даними, на яких і була натренована модель. Далі, скомпрометований пристрій почав надсилати дані, які не відповідали заданому тренду, а саме набагато завищені показними температури та вологості. Elasticsearch відразу позначив такі пакети як аномальні.

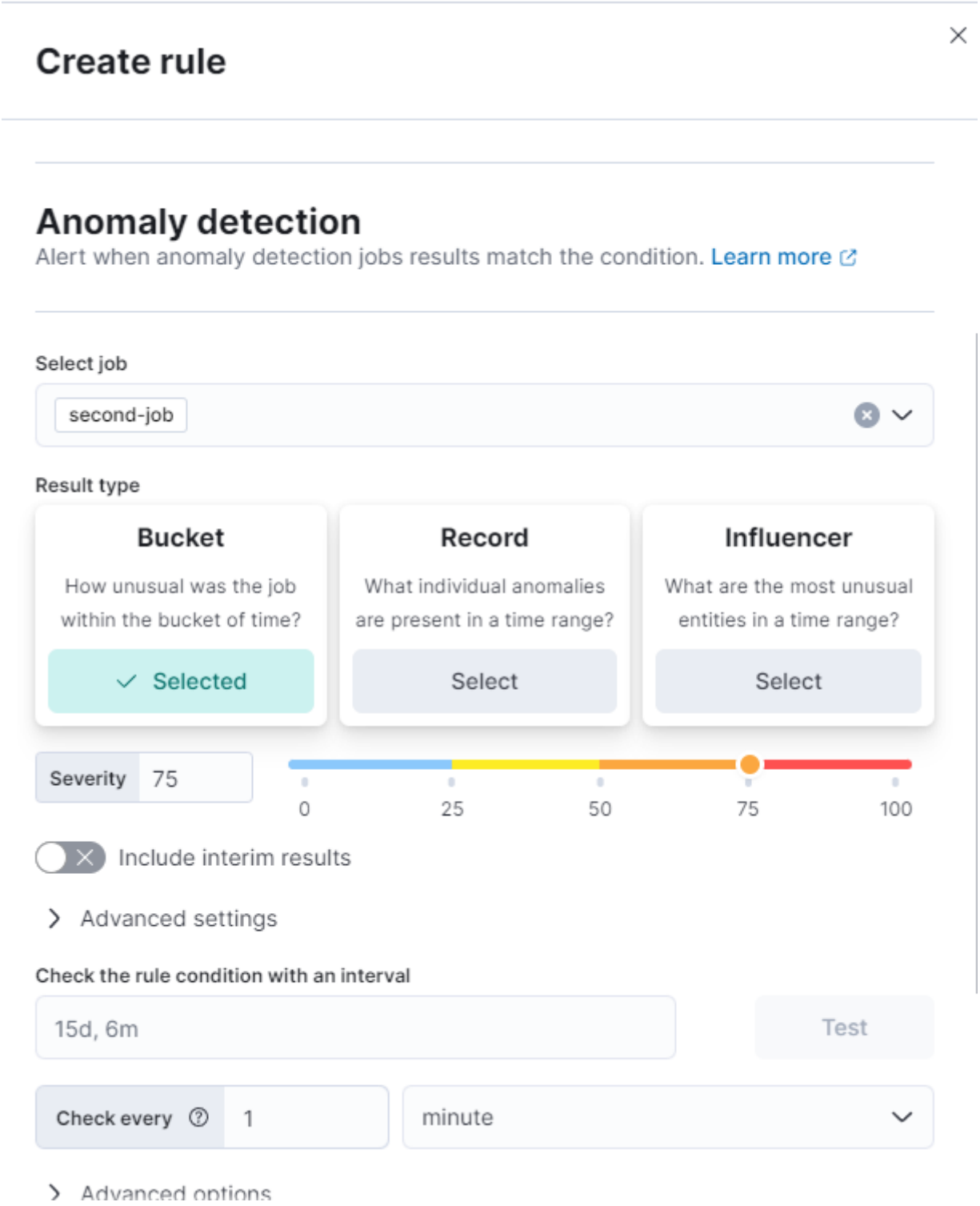

Рис. 15. Встановлення дій, що потрібно виконати після виявлення аномалій

Крім простої візуалізації можливо додати правила сповіщення на випадок виявлення аномалій. В межах нашої системи пропонується налаштовувати вебхуки, які будуть сповіщати про скомпрометовані пристрої відповідний сервер. Той в

свою чергу, буде приймати відповідні кроки, наприклад, відмову приладу в комунікації до з'ясування проблеми тощо.

Висновок до розділу 5

Запропонована система забезпечує фундамент для побудови різноманітних рішень в сфері IoT, враховуючи унікальність IoT систем та їхніх доменних областей. Розроблений фреймворк пропонує базовий функціонал, який можна легко розширювати для задоволення конкретних вимог. Основні компоненти системи – розподільник навантаження Nginx, брокер повідомлень MQTT HiveMQ, сервер авторизації, сервер збору інформації та збірка сервісів ELK Stack. Разом вони формують надійну та масштабовану архітектуру. Ці елементи об'єднуються для забезпечення ефективної та безпечної комунікації. TLS протокол дозволяє широку гнучкість для вибору конкретних криптографічних бандлів, що найбільше підійдуть для конкретних ситуацій. Такий підхід дозволяє пристроям IoT безпечно взаємодіяти між собою та із сервером, використовуючи ефективні методи шифрування, автентифікації та авторизації. Додатково, в системі використовується машинне навчання для навчання моделей та виявлення аномалій в реальному часі. Система може стати основою для розробки кінцевих рішень, що враховують унікальні потреби та обмеження кожного домену.

Висновок

Метою даної роботи була розробка комплексного рішення для захисту IoT систем, а також дослідження відомих та новітніх рішень у цій сфері.

В розділі 1 було окреслено шарове представлення архітектури IoT систем, а саме рівні сприйняття, мережі, обробки даних та застосунку. Кожному з цих рівнів притаманні як спільні вразливості, так і унікальні. Визначаються критичні точки, які можуть бути використанні зловмисниками для отримання доступу до систем чи перешкоджання роботи тих чи інших вузлів.

Розділ 2 фокусується на проблемах автентифікації та авторизації. Стандартні облікові дані визначаються як найпоширеніша та найпростіша для скористання зловмисниками вразливість. Ця вразливість існує в великому переліку систем і вже не один раз призводила до широких успішних кібер-атак, як то Mirai ботнет. Далі розглядаються наукові роботи присвячені менш розповсюдженим проблемам в сфері контролю доступу: проблема централізованих центрів довіри в TLS протоколі і пропозиція переходу на розподілені центри; випадки застосування IoT пристроїв без традиційних способів контролю доступу чи з потребою в продовжуваному контролі доступу і спекуляції на тему використання унікальних біологічних даних індивіда в поєднанні з машинним навчанням; використання IoT в несприятливих умовах (зона бойових дій), де існує ризик фізичної втрати пристроїв чи серверів і компрометації систем, та способи захисту в таких сценаріях.

Розділ 3 присвячено шифруванню. Тут досліджуються такі шифрувальні протоколи та методи як TLS, DTLS, Novel Tiny Symmetric Encryption Algorithm, Lightweight CA Cipher (LCC) та Functional Encryption (FE), а також їхнє застосування в IoT.

Розділ 4 окреслює стратегії реагування на інциденти безпеки, і визначає ключові елементи для ефективних відповідей на атаки.

Останній розділ описує процес розробки та імпелементації власної системи для підтримання безпеки в IoT оточеннях. Розроблена система використовує TLS як основний протокол шифрування. Рекомендується використовувати ECDHE, проте це все ще залишається на розсуд кінцевих розробників, що будуть використовувати систему. На додаток, між приладами та сервером рекомендується використовувати certificate pinning, що дозволить убезпечитися від компрометації центрів довіри. Система складається з 5 компонентів:

1. Nginx – використовується в ролі reverse-proxy.
2. HiveMQ – MQTT брокер для інтер- та інтра-комунікації між приладами та сервером.
3. Сервер контролю доступу – зберігає облікові дані та загальну інформацію про пристрої. Розроблені розширення для HiveMQ використовують його для авторизації та автентифікації приладів.
4. ELK Stack (Elasticsearch, Logstash, Kibana) – збірка сервісів ELK використовується для аналізу даних та знаходження аномалій в реальному часі за допомогою машинного навчання з можливістю відповіді на знайдені проблеми.
5. Сервер збору інформації – відповідає за трансфер даних з MQTT брокера до Logstash.

Дана робота дослідила різні підходи до захисту IoT систем, а також представила розробку власного підходу, чим і досягла поставленої мети.

Джерела

[1] “Кібервійна росії проти України. Як українські спецслужби та хактивісти виборюють незалежність держави,” speka.media. https://speka.media/kiberviina-rosiyi-proti-ukrayini-9qy4ok

[2] M. Burhan, R. Rehman, B. Khan, and B.-S. Kim, “IoT Elements, Layered Architectures and Security Issues: A Comprehensive Survey,” Sensors, vol. 18, no. 9, p. 2796, Aug. 2018, doi: https://doi.org/10.3390/s18092796.

[3] H. Kim and E. A. Lee, “Authentication and Authorization for the Internet of Things,” IT Professional, vol. 19, no. 5, pp. 27–33, 2017, doi: https://doi.org/10.1109/mitp.2017.3680960.

[4] M. Shahzad and M. P. Singh, "Continuous Authentication and Authorization for the Internet of Things," in IEEE Internet Computing, vol. 21, no. 2, pp. 86-90, Mar.-Apr. 2017, doi: 10.1109/MIC.2017.33.

[5] S. Echeverría, G. A. Lewis, D. Klinedinst and L. Seitz, "Authentication and Authorization for IoT Devices in Disadvantaged Environments," 2019 IEEE 5th World Forum on Internet of Things (WF-IoT), Limerick, Ireland, 2019, pp. 368-373, doi: 10.1109/WF-IoT.2019.8767192.

[6] I. Grigorik, “4. Transport Layer Security (TLS) - High Performance Browser Networking [Book],” www.oreilly.com, Sep. 2013. https://www.oreilly.com/library/view/high-performance-browser/9781449344757/ch04.html (accessed May 26, 2024).

[7] “Computationally simple, lightweight replacement for SSL/TLS,” Information Security Stack Exchange, Apr. 2011. https://security.stackexchange.com/questions/3204/computationally-simple-lightweight-replacement-for-ssl-tls

[8] “Difference TLS Vs DTLS protocol,” The Network DNA, Nov. 2022. https://www.thenetworkdna.com/2022/11/difference-tls-vs-dtls-protocol.html

[9] S. Kim and I. Lee, “IoT device security based on proxy re-encryption,” Journal of Ambient Intelligence and Humanized Computing, vol. 9, no. 4, pp. 1267–1273, Nov. 2017, doi: https://doi.org/10.1007/s12652-017-0602-5.

[10] S. Rajesh, V. Paul, V. Menon, and M. Khosravi, “A Secure and Efficient Lightweight Symmetric Encryption Scheme for Transfer of Text Files between Embedded IoT Devices,” Symmetry, vol. 11, no. 2, p. 293, Feb. 2019, doi: https://doi.org/10.3390/sym11020293.

[11] S. Roy, U. Rawat and J. Karjee, "A Lightweight Cellular Automata Based Encryption Technique for IoT Applications," in IEEE Access, vol. 7, pp. 39782-39793, 2019, doi: 10.1109/ACCESS.2019.2906326.

[12] D. Sharma and Devesh Jinwala, “Functional Encryption in IoT E-Health Care System,” Lecture notes in computer science, pp. 345–363, Jan. 2015, doi: https://doi.org/10.1007/978-3-319-26961-0_21.

[13] “Spring Framework Overview :: Spring Framework,” docs.spring.io. https://docs.spring.io/spring-framework/reference/overview.html

[14] “HiveMQ - Enterprise ready MQTT to move your IoT data,” www.hivemq.com. https://www.hivemq.com/

[15] D. Horovits, “The Complete Guide to the ELK Stack,” Logz.io, Feb. 08, 2023. https://logz.io/learn/complete-guide-elk-stack/#what-elk-stack

Додатки

1. Розширення для HiveMQ
   https://github.com/EinErste/spring-hivemq-auth-extentsion
2. Сервер автентифікації та авторизації
   https://github.com/EinErste/spring-hivemq-auth-starter
3. Сервер збору інформації
   https://github.com/EinErste/spring-hivemq-collector