УДК 004.056
DOI: 10.18523/2617-3808.2024.7.77-88

Щербина С. С., Бабич Т. А.

ФРЕЙМВОРК УПРАВЛІННЯ ЗАГРОЗАМИ ТА РЕАГУВАННЯ НА ІНЦИДЕНТИ IoT СИСТЕМИ

У статті розглянуто розроблення та впровадження фреймворку для управління загрозами та реагування на інциденти в системах Інтернету речей (IoT). Запропонований фреймворк поєднує елементи розподіленої архітектури, зокрема Nginx як розподільник навантаження, MQTT брокер HiveMQ, сервер авторизації та збірку сервісів ELK Stack. Це рішення забезпечує безпечну комунікацію IoT пристроїв за допомогою протоколу TLS та механізмів шифрування, автентифікації і авторизації. Особливу увагу надано використанню машинного навчання для виявлення аномалій у реальному часі, що дає змогу ефективно реагувати на потенційні загрози в різних доменах IoT. Фреймворк розроблено з урахуванням обмежених обчислювальних ресурсів IoT пристроїв і вимог до їхньої безпеки.

Ключові слова: Інтернет речей, вразливості безпеки, архітектура IoT, розподільник навантаження Nginx, брокер повідомлень MQTT HiveMQ, збірка сервісів ELK Stack, TLS протокол, протокол MQTT, Logstash, Elasticsearch, Kibana.

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 й відкриває дорогу до небачених раніше можливостей та ефективності у сферах розумного будинку, медицини, індустріального виробництва, міського управління тощо [2]. Проте, як і з будь-яким технологічним проривом, IoT (Internet of Things) потребує 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 й передавати інформацію автономно, без людського втручання. Хоча саме це і є головною інновацією, така автономність створює багато вразливостей у безпеці. Наприклад, IoT пристрої часто використовують як приватні, так і публічні мережі, в такий спосіб розширюючи потенційний вектор атаки зловмисника на конфіденційність і цілісність даних. Але це не єдине, про що потрібно турбуватися. Не менш важливими є надійність і доступність критично важливого функціоналу, що його такі прилади надають. Від термостатів і 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.

IoT прилади зазвичай мають мало обчислювальної потужності та оперативної пам'яті. Такі обмеження призводять до того, що часто традиційні підходи до безпеки як складні алгоритми шифрування є неможливими у впровадженні. Зважаючи на це і на вибухове зростання кількості IoT приладів у використанні, також збільшується і потенціал для несанкціонованого втручання в роботу систем. За прогнозом Cybersecurity Ventures, кількість IoT пристроїв у світі зросте до 25,1 мільярда до 2025 р. Кожний із них є потенційною точкою входу для зловмисника. Таке значне поширення створює велику, часто погано захищену мережу взаємопов'язаних приладів, яку можуть використати зловмисники для власних цілей: крадіжки даних, DDoS атак тощо.

Більше того, non-patchable (не підлягають виправленню) і forever-day (вічні) вразливості тільки ускладнюють проблему. Багато IoT пристроїв нечасто отримують оновлення прошивки через логістичні проблеми доставки цих оновлень або ж тому, що виробник про них забув і більше не підтримує. Такі вразливості являють собою постійно відкриті двері для зловмисників і тому потребують інноваційного підходу до поточних рішень.

Інфраструктуру навколо IoT пристроїв можливо розділити на чотири рівні [1]: сприйняття (захоплення вузла, імперсоніфікація вузла, атака відтворення, часова атака, атака позбавлення сну), мережі (підслуховування, DoS/DDoS атака, Main-in-The-Middle, оброблення даних (виснаження ресурсів, експлойти), застосунку (експлойти). Отже, основними проблемами для безпеки в IoT є несанкціонований доступ до пристроїв і даних, підслуховування комунікації, маніпуляція комунікації, виснаження ресурсів систем і пристроїв.

Метою цієї статті є опис запропонованого комплексного рішення для захисту IoT систем, представленого у формі спеціального фреймворка. Існує нагальна потреба в надійних і здатних до масштабування рішень. Наш підхід враховує унікальні характеристики IoT екосистем, а саме різноманітність, обмеженість в обчислювальних ресурсах і фізичну природу приладів.

Неоднорідність і велика різноманітність пристроїв інтернету речей створюють значні проблеми для моніторингу, виявлення та реагування. Це означає, що для різних груп пристроїв потенційно може бути потрібен різний підхід для вищезгаданих процесів. Отримання повного логу операцій пристроїв і відкритого мережевого трафіку є складним завданням, що ускладнює виявлення та розуміння масштабу інцидентів. Обмежена потужність обчислювальних ресурсів і 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.

Усі IoT системи є суттєво залежними від власної доменної області. Наприклад, методи та підходи до управління приладами в системах розумного міста і системах домашнього будинку будуть суттєво відрізнятися. Ця різниця залежатиме 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 внутрішньої комунікації 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 своєю чергою диктує методи та підходи до шифрування [3]. До того ж, останнє, але не за значенням — авторизація та автентифікація [4]. Вони можуть набувати різних форм залежно від обраного підходу до інтеркомунікації та шифрування даних.

Отже, було прийнято рішення розробити систему, що може слугувати фундаментом для конкретних рішень. Запропонований фреймворк надає базовий функціонал, що є легко розширюваним, і відображає наше бачення реалізації безпеки в кінцевій системі.

Архітектура фреймворку

Система складається з таких компонентів (рис. 1): розподільник навантаження Nginx [5], брокер повідомлень MQTT HiveMQ [6], сервер авторизації, сервер збору інформації, збірка сервісів ELK Stack [7].

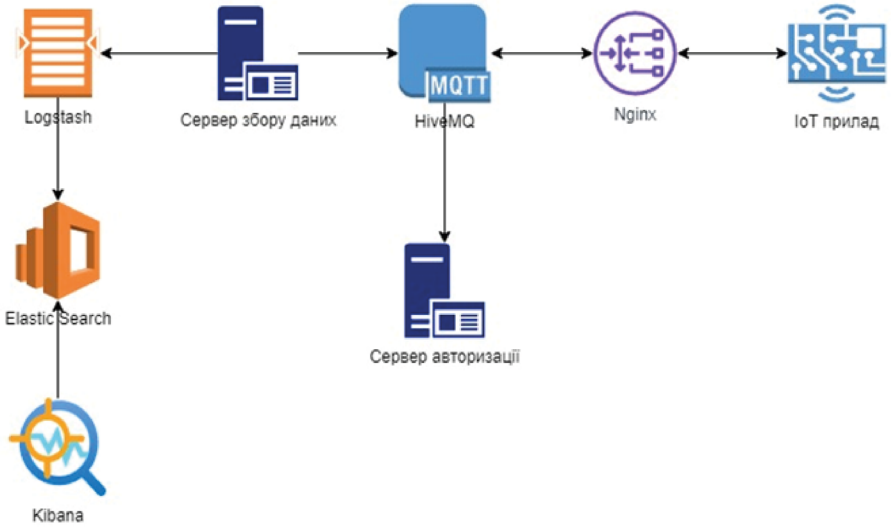

Рис. 1. Архітектура розробленої системи

Nginx

Nginx — це open-source програмне забезпечення для вебсервера, яке використовують для зворотного проксі-сервера, балансування навантаження та кешування. Воно надає можливості HTTPS сервера і розроблене для максимальної продуктивності й стабільності. Nginx 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. Воно використовує мультиплексування та сповіщення про події і призначає окремі завдання окремим процесам.

Nginx не створює новий потік виконання для кожного нового з'єднання. Натомість спеціальний майстер-процес обробляє прийом нових запитів із загального сокету прийому даних і запускає високоефективні цикли виконання всередині окремих робочих процесів. Це дає змогу обробляти тисячі з'єднань у межах одного такого процесу. Спеціальних механізмів розподілу з'єднань між різними робочими 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 роботу буде розподілено між процесами, майстер очікуватиме наступний запит від клієнта, тобто не чекатиме на відповідь від робочого процесу. Після отримання асинхронної відповіді від останнього майстер-процес надішле відповідь клієнту. Також майстер читає і перевіряє конфігурацію сервера шляхом створення, приписування та з'єднання сокетів. Він також обробляє запуск, завершення та підтримку кількості налаштованих робочих процесів. До того ж, він може переналаштовувати робочий процес без переривання оброблення 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 [6].

Сервер авторизації

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

Spring вважають безпечною, недорогою і гнучкою платформою, яка покращує ефективність написання коду та скорочує загальний час розроблення програми за рахунок ефективного використання системних ресурсів. Spring усуває виснажливу роботу з налаштування, щоб розробники могли зосередитися на написанні бізнес-логіки [139]. Крім того, Spring керує інфраструктурою.

Також Spring надає численні слабко зв'язані модулі для покращення функціональності вебдодатка. 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 розроблення.

Розроблена бібліотека виконує такі дві функції: постачання облікових даних від MQTT брокера для IoT пристроїв, автентифікація та авторизація IoT пристроїв на MQTT брокері.

Облікові дані

Запропонований підхід до роботи з обліковими даними має на меті усунути проблему доставки та ротації їх на IoT пристроях. Для досягнення такого результату потрібно, щоби попередньо виконувалися такі умови:

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

Задля виконання першого пункту розробникам потрібно імплементувати та використовувати такий 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 дає змогу розробникам зберігати дані про пристрої в будь-якому зручному для них форматі, наприклад у SQL/NoSQL базах даних тощо.

Дані потрібно зберігати в такому 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<>();
}
```

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

Для IoT пристроїв надають 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;
2) /api/iot/device/certificate – прилад запитує актуальний публічний ключ TLS для Nginx/HiveMQ.

Для HiveMQ теж надають два HTTPS ендпоінти, що є захищеними наперед вибраним паролем:

1) /api/iot/mqtt/authenticate – брокер надсилає логін та пароль, отриманий від IoT пристрою, та очікує на відповідь про дозвіл на автентифікацію;
2) /api/iot/mqtt/authorize – брокер надсилає логін автентифікованого пристрою і топік, із яким пристрій хоче взаємодіяти (писати або читати).

Важливим 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.

У системі реалізується стандартна перевірка сертифіката. У типовому TLS handshake, коли клієнт під'єднується до сервера, сервер представляє клієнту свій цифровий сертифікат. Потім клієнт перевіряє автентичність сертифіката, зокрема те, 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.

Ми пропонуємо і закріплення сертифіката. 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, у мікропрограми пристроїв має бути 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 [1410].

Протокол має два типи об'єктів: брокер повідомлень і клієнт. Брокер MQTT функціонує як сервер, який приймає повідомлення від клієнтів і поширює їх для інших клієнтів. Клієнтом MQTT може бути будь-який пристрій, який використовує бібліотеку MQTT і під'єднується до брокера через мережу.

Інформацію в MQTT структуровано як ієрархію топіків. Коли видавець (publisher) має нові дані, він надсилає повідомлення брокеру на відповідний топік. Далі це повідомлення вичитують підписники (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 надсилають тільки останнє таке повідомлення, всі попередні ігноруються. До того ж, можливо налаштувати повідомлення за замовчуванням, яке буде надіслано підписникам у разі його несподіваного від'єднання від брокера.

MQTT підтримує передавання даних через TCP, але існує також MQTT-SN [11], що працює через інші транспортні протоколи, такі як UDP або Bluetooth. MQTT надсилає облікові дані під'єднання як звичайний текст, що означає відсутність убудованих заходів безпеки або автентифікації. Для гарантування безпеки можна використовувати TLS.

Існує чотирнадцять типів повідомлень MQTT, які відповідають за різні функції, зокрема під'єднання і від'єднання 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 мегабайтів.

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

- щонайбільше один раз – повідомлення надсилається лише один раз, і клієнт і брокер не роблять 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 передавання даних у TCP. Він є надбудовою протоколу MQTT.

Існує чотири основні типи повідомлень в MQTT: CONNECT використовують для надсилання клієнтами запитів на під'єднання до брокера; PUBLISH використовують видавці для надсилання повідомлень брокеру; SUBSCRIBE використовують підписники для отримання повідомлень від брокера; DISCONNECT використовують клієнти 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 і псевдоніми для топіків.

HiveMQ — MQTT брокер і платформа обміну повідомленнями, розроблена мовою Java, яка використовує протокол MQTT для швидкого, надійного та ефективного двонаправленого передання даних до і 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. Є можливість інтегровувати HiveMQ з наявними корпоративними системами завдяки його відкритому API та гнучкому розширенню. Також розробники HiveMQ створили інструмент під назвою MQTT CLI, який надає інтерфейс командного рядка для взаємодії з брокерами MQTT, у нашому випадку з HiveMQ. Така утиліта є корисною, бо дає змогу відкрити декілька інстансів одночасно. А це є зручним для тестування.

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

Кластер MQTT брокерів — це розподілена система, яка діє як єдиний логічний брокер MQTT для під'єднаних клієнтів. Як правило, вузли кластера MQTT встановлюють на окремих фізичних або віртуальних машинах і під'єднують через мережу. Такий підхід гарантує, що розгорнута система не матиме єдиної точки збою. В інфрастуктурі, де доступність є надзвичайно важливою, кластеризація є необхідним компонентом. Якщо один MQTT брокер у кластері більше не доступний, решта вузлів можуть продовжувати обробляти трафік. У HiveMQ реалізовано архітектуру без майстра, що дозволяє мати необмежену кількість нод.

HiveMQ пропонує Extension SDK для розроблення власних розширень, що можуть виконувати такі функції: перехоплення та маніпуляція повідомленнями MQTT, інтеграція з іншими сервісами,

збір статистики, розширення управління доступом. У запропонованій системі увагу приділяють першій і четвертій функціям.

Імплементуючи інтерфейс SimpleAuthenticator, ми реалізуємо метод onConnect (SimpleAuthInput simpleAuthInput, SimpleAuthOutput simpleAuthOutput). У цьому методі замість простої перевірки ACL файлу, що містить дані про доступи, виконується запит до сервера авторизації для отримання актуальної інформації про статус клієнта, що намагається під'єднатися.

Імплементуючи інтерфейс PublishAuthorizer, ми реалізуємо метод authorizePublish(PublishAuthorizerInput input, PublishAuthorizerOutput output). Знову, замість простої перевірки ACL файлу, виконується запит до сервера авторизації для отримання актуальної інформації про доступні топіки для автентифікованого клієнта.

Дані, отримані за обома методами, можливо кешувати для збільшення швидкодії. Це можливо реалізувати як установленням 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 зручного передавання даних до Elastic Search та подальшого їхнього оброблення.

Для забезпечення конфіденційності та цілісності даних використовують TLS.

Сервер збору інформації

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

У поточній реалізації Logstash підіймає TCP сервер і очікує під'єднання від сервера збору інформації. Проте варто зазначити, що таку фунцію 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 три розширення, які мали б дозволити зчитувати повідомлення напряму з MQTT брокера. Проте останнє оновлення найновішого з них датовано 2018 роком. Після багатьох невдалих спроб інтеграції цих розширень стало зрозуміло, що потрібно розробити щось нове. Розширення для Logstash пишуть мовою Ruby. Задля спрощення розроблення і тестування було прийнято рішення не розроблювати нове розширення для Logstash чи намагатися виправити старі версії, а написати власний сервіс на Spring, що виконуватиме ті самі функції.

ELK Stack

Logstash — це потужний інструмент, який відіграє важливу роль в ELK стеку [1512]. Він дає змогу 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 на 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 дають змогу користувачам видаляти небажані дані, перетворювати дані у відповідний формат і агрегувати їх на основі певних критеріїв. Такий 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-от Elasticsearch чи Kibana, чи використання його для надсилання даних до інших місць призначення, як Kafka чи Redis, Logstash забезпечує високий ступінь гнучкості.

У розробленій системі Logstash виконує роль пайплайну до Elasticsearch. Він отримує дані від сервера збору інформації та надсилає їх далі. Також тут можливо скористатися всіма потрібними

плагінами для попереднього оброблення даних, якщо щодо цього є потреба. Використовують такий конфіг:

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

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

filter {

}

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

Elasticsearch є потужною пошуковою системою, яка пропонує масштабовані рішення для повнотекстового пошуку [1512]. Інструмент побудовано на принципах розподілених систем, призначених для оброблення 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, а 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. Тож користувачі можуть виконувати пошук у контексті найновішої інформації, що 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, складними фразами і регулярними виразами. До того ж, існує можливість пошуку за повнотекстовими запитами, які аналізують контекст і семантику термінів. Ще одною великою перевагою використання Elasticsearch є можливість інтеграції з Kibana [1512], що є інструментом візуалізації. Він дає змогу 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, щоби користувачі спершу отримували найрелевантніші результати пошуку.

Elasticsearch постачають разом із 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 приєднання Elasticsearch до розробленої системи є широкі можливості та простота використання інструменту в сфері машинного навчання.

Виявлення аномалій є одним із найпотужніших застосувань машинного навчання в Elasticsearch. Такий механізм дає змогу користувачам знаходити незвичні закономірності або просто незвичності

у своїх даних, які можуть вказувати на потенційні проблеми. Це особливо корисно для виявлення моніторингу безпеки мережі, моніторингу здоров'я життєздатності 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, на їхню думку, має бути виявлення аномалій.

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 з розвитком поведінки користувачів або системних показників ML моделі адаптуються і 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. Вона пропонує набір інструментів, розроблених, щоби допомогти користувачам візуалізувати, досліджувати та взаємодіяти зі своїми даними, що зберігаються в Elasticsearch. Kibana діє як інтерфейс, за допомогою якого величезний набір даних можна перетворити на практичну інформацію, що допомагає компаніям і аналітикам приймати обґрунтовані рішення на основі інформації в реальному часі.

Kibana є потужним інструментом візуалізації даних, який дає можливість користувачам створювати діаграми та графіки, формат яких найкраще відповідає тим чи тим даним (рис. 2). 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-які варіанти використання.

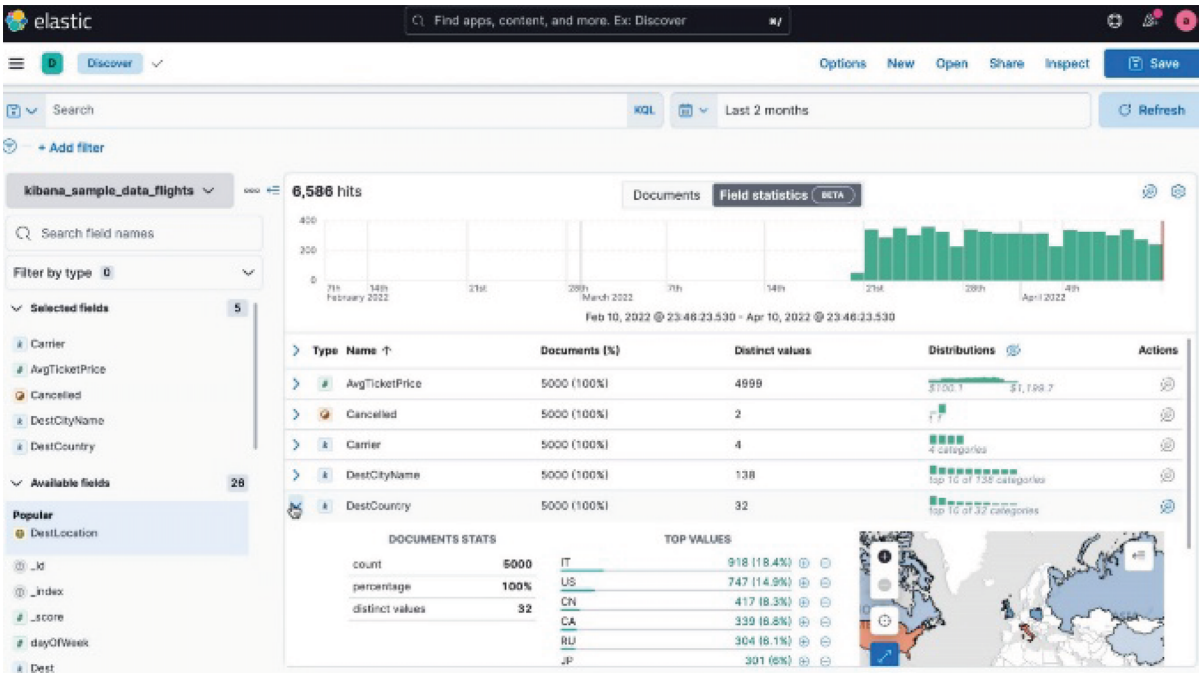

Рис. 2. Приклад сконфігурованого інтерфейсу Kibana

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 ре-

жимі реального часу має вирішальне значення для моніторингу систем, відстеження логів подій або аналізу моделей поведінки користувачів у міру їх виникнення. Більше того, 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 дає змогу 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 згаданого 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.

Наприклад, на рис. 3 наведено екран огляду завдань з виявлення аномалій.

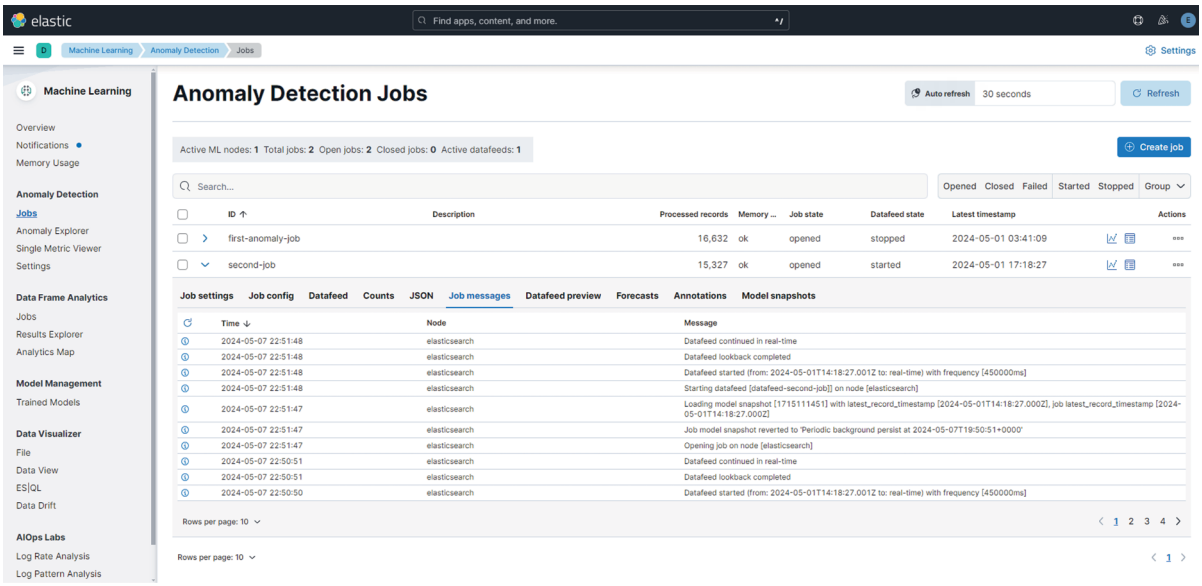

Рис. 3. Екран огляду завдань із виявлення аномалій

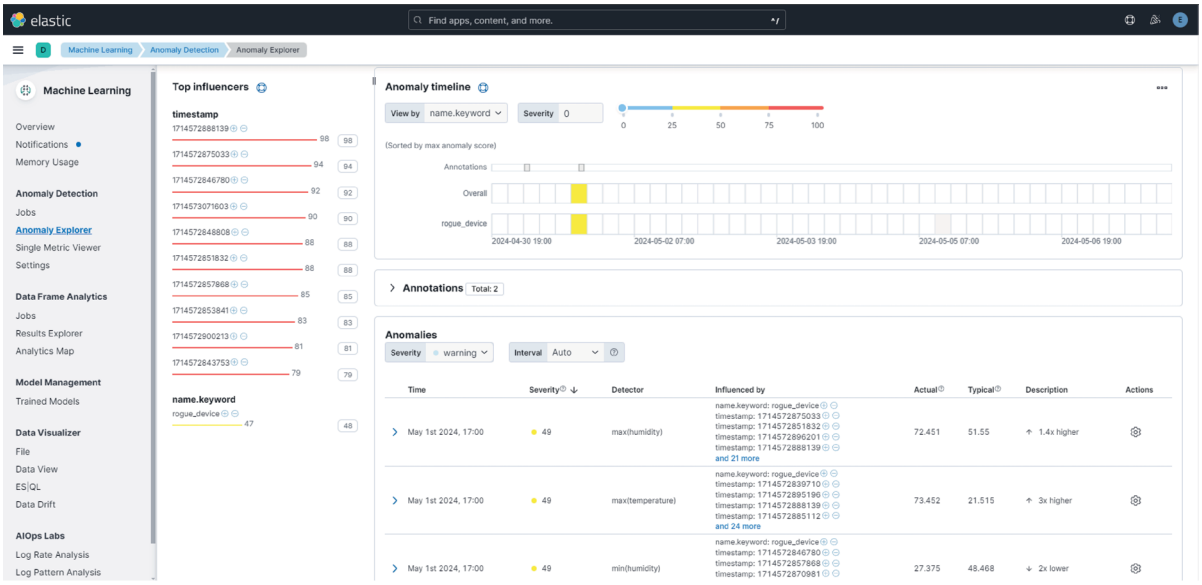

Рис. 4. Екран огляду виявлених аномалій

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

Для демонстрації було налаштовано і натреновано задачу виявлення аномалій (рис. 4). В цьому випадку у роботі брали участь два змодельовані пристрої, які надсилали інформацію зі своїх зчитувачів, а саме: температуру навколишнього середовища та вологість повітря. Спочатку робочий пристрій надіслав 15 000 пакетів з консистентними та очікуваними даними, на яких і було натреновано 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.

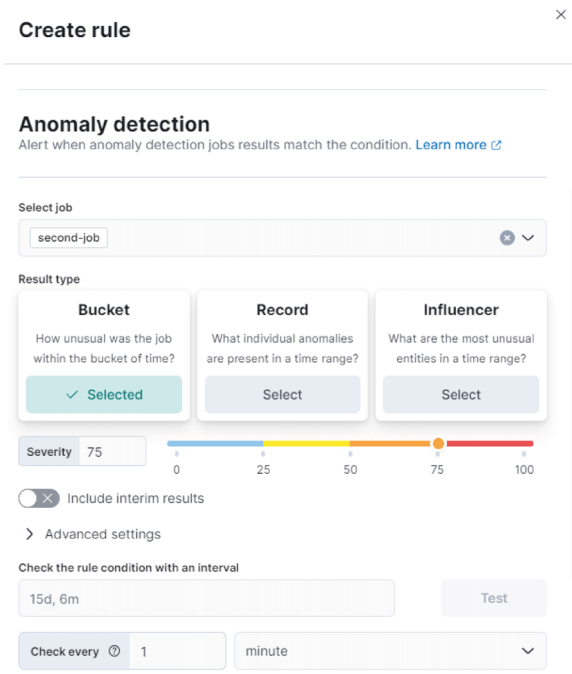

Рис. 5. Встановлення дій, які потрібно виконати після виявлення аномалій

Крім простої візуалізації можливо додати правила сповіщення на випадок виявлення аномалій. У межах нашої системи пропонується налаштовувати вебхуки, які будуть сповіщати про скомпрометовані пристрої відповідний сервер. Той, своєю чергою, робитиме відповідні кроки, наприклад, відмовлятиме приладу в комунікації до з'ясування проблеми тощо.

Висновок

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 пропонує широку гнучкість для вибору конкретних криптографічних бандлів, що найбільше підійдуть для конкретних ситуацій. Такий підхід дає змогу 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 і виявлення аномалій у реальному часі. Система може стати основою для розроблення кінцевих рішень, що враховують унікальні потреби та обмеження кожного домену.

Розроблена система використовує TLS як основний протокол шифрування. Рекомендують застосовувати ECDHE, проте це все ще залишається на розсуд кінцевих розробників, які будуть використовувати систему. На додаток, між приладами та сервером рекомендують використовувати certificate pinning, що дасть змогу убезпечитися від компрометації центрів довіри.

Унікальні характеристики IoT екосистем потребують 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, а й організаційну готовність, зокрема навчання, планування та аналіз після інциденту.

Список літератури

1. Глибовець А. М. Агентно-базовані програмні системи пошуку та аналізу інформації / А. М. Глибовець. — Київ : Видавничий дім «Києво-Могилянська академія», 2019. — 281 с. : іл.
2. Burhan M. IoT Elements, Layered Architectures and Security Issues: A Comprehensive Survey / M. Burhan, R. Rehman, B. Khan, B.-S. Kim // Sensors. — 2018. — Vol. 18, no. 9. — P. 2796. — https://doi.org/10.3390/s18092796.
3. Computationally simple, lightweight replacement for SSL/TLS [Electronic resource]. Information Security Stack Exchange, Apr. 2011. Mode of access: https://security.stackexchange.com/questions/3204/computationally-simple-lightweight-replacement-for-ssl-tls.
4. Difference TLS Vs DTLS protocol TLS [Electronic resource]. The Network DNA, Nov. 2022. Mode of access: https://www.thenetworkdna.com/2022/11/difference-tls-vs-dtls-protocol.html.
5. Echeverría S. Authentication and Authorization for IoT Devices in Disadvantaged Environments / S. Echeverría, G. A. Lewis, D. Klinedinst, L. Seitz // 2019 IEEE 5th World Forum on Internet of Things (WF-IoT). — Limerick, Ireland, 2019. — Pp. 368–373. — https://doi.org/10.1109/WF-IoT.2019.8767192.
6. Grigorik I. 4. Transport Layer Security (TLS) — High Performance Browser Networking [Electronic resource] / I. Grigorik. — Mode of access: https://www.oreilly.com/library/view/high-performance-browser/9781449344757/ch04.html (date of access: May 26, 2024).
7. HiveMQ — Enterprise ready MQTT to move your IoT data [Electronic resource]. — Mode of access: https://www.hivemq.com/.
8. Horovits D. The Complete Guide to the ELK Stack [Electronic resource] / D. Horovits // Logz.io, Feb. 08, 2023. — Mode of access: https://logz.io/learn/complete-guide-elk-stack/#what-elk-stack.
9. Kim H. Authentication and Authorization for the Internet of Things / H. Kim, E. A. Lee // IT Professional. — 2017. — Vol. 19, no. 5. — Pp. 27–33. — https://doi.org/10.1109/mitp.2017.3680960.
10. Roy S. A Lightweight Cellular Automata Based Encryption Technique for IoT Applications / S. Roy, U. Rawat, J. Karjee // IEEE Access. — 2019. — Vol. 7. — Pp. 39782–39793. — https://doi.org/10.1109/ACCESS.2019.2906326.
11. Shahzad M. Continuous Authentication and Authorization for the Internet of Things / M. Shahzad, M. P. Singh // IEEE Internet Computing. — 2017. — Vol. 21, no. 2. — Pp. 86–90. — https://doi.org/10.1109/MIC.2017.33.
12. Spring Framework Overview: Spring Framework [Electronic resource]. — Mode of access: https://docs.spring.io/spring-framework/reference/overview.html.

References

Burhan, M., Rehman, R., Khan, B., & Kim, B.-S. (2018). IoT Elements, Layered Architectures, and Security Issues: A Comprehensive Survey. *Sensors, 18* (9), 2796. https://doi.org/10.3390/s18092796.

Computationally simple lightweight replacement for SSL/TLS. (2011). *Information Security Stack Exchange*. https://security.stackexchange.com/questions/3204/computationally-simple-lightweight-replacement-for-ssl-tls.

Difference TLS Vs DTLS protocol. (2022). *The Network DNA*. https://www.thenetworkdna.com/2022/11/difference-tls-vs-dtls-protocol.html.

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

Grigorik, I. (2013). *Transport Layer Security (TLS) — High Performance Browser Networking*. O'Reilly. https://www.oreilly.com/library/view/high-performance-browser/9781449344757/ch04.html.

HiveMQ — Enterprise ready MQTT to move your IoT data. (n.d.). https://www.hivemq.com/.

Hlybovets, A. M. (2019). *Agent-based software systems for information search and analysis*. Publishing House "Kyiv-Mohyla Academy".

Horovits, D. (2023). The Complete Guide to the ELK Stack. *Logz.io*. https://logz.io/learn/complete-guide-elk-stack/.

Kim, H., & Lee, E. A. (2017). Authentication and Authorization for the Internet of Things. *IT Professional, 19* (5), 27–33. https://doi.org/10.1109/mitp.2017.3680960.

Roy, S., Rawat, U., & Karjee, J. (2019). A Lightweight Cellular Automata Based Encryption Technique for IoT Applications. *IEEE Access, 7*, 39782–39793. https://doi.org/10.1109/ACCESS.2019.2906326.

Shahzad, M., & Singh, M. P. (2017). Continuous Authentication and Authorization for the Internet of Things. *IEEE Internet Computing, 21* (2), 86–90. https://doi.org/10.1109/MIC.2017.33.

Spring Framework Overview: Spring Framework. (n.d.). https://docs.spring.io/spring-framework/reference/overview.html.

S. Shcherbyna, T. Babych

FRAMEWORK FOR THREAT MANAGEMENT AND INCIDENT RESPONSE IN IOT SYSTEMS

The article presents the development and implementation of a framework for managing threats and responding to incidents in Internet of Things (IoT) systems. The proposed framework integrates elements of a distributed architecture, including Nginx as a load balancer, the MQTT HiveMQ broker, an authorization server, and the ELK Stack for data analysis and visualization. This solution ensures secure communication

between IoT devices using the TLS protocol and employs advanced mechanisms for encryption, authentication, and authorization. Particular attention is paid to leveraging machine learning for real-time anomaly detection, which enables effective responses to potential threats in various IoT domains. The framework is designed to accommodate the computational constraints of IoT devices while meeting stringent security requirements.

The importance of IoT lies in its ability to autonomously collect, process, and transmit information without human intervention. However, this autonomy introduces several security vulnerabilities. IoT devices, often operating within public and private networks, increase the attack surface for malicious actors targeting data confidentiality, integrity, and availability. With an estimated 25.1 billion IoT devices expected by 2025, each device represents a potential entry point for cyber threats. Issues like unpatchable vulnerabilities and outdated firmware exacerbate security risks, highlighting the need for innovative solutions.

The proposed framework addresses these challenges by establishing a modular and scalable architecture tailored to the diverse and resource-constrained nature of IoT ecosystems. Components such as Nginx, HiveMQ, and the ELK Stack enable reliable communication and data analysis. Nginx serves as a reverse proxy and an entry point, simplifying TLS certificate management and load balancing. HiveMQ, selected for its extensibility and clustering capabilities, acts as a message broker that facilitates efficient and secure data exchange. The ELK Stack, comprising Logstash, Elasticsearch, and Kibana, provides a comprehensive pipeline for real-time data ingestion, processing, and visualization.

A key feature of the framework is its integration of machine learning models for anomaly detection. These models, trained on historical data, monitor real-time metrics to identify deviations from normal patterns. This capability is crucial for detecting potential security breaches and irregular operations. Moreover, the system employs certificate pinning and other cryptographic measures to protect against Man-in-the-Middle (MITM) attacks and ensure secure device-server interactions.

The framework's modularity allows for customization across specific IoT domains, such as smart cities, healthcare, and industrial IoT. By providing foundational functionality, the framework facilitates the development of domain-specific solutions that address unique challenges while ensuring scalability and security.

In conclusion, the proposed framework represents a comprehensive approach to managing IoT threats and responding to incidents. By integrating secure communication protocols, machine learning-driven anomaly detection, and a modular architecture, it lays the groundwork for reliable and adaptive IoT security solutions.

Keywords: IoT security vulnerabilities, IoT architecture, Nginx load balancer, MQTT HiveMQ broker, ELK Stack services, TLS protocol, MQTT protocol, Logstash, Elasticsearch, Kibana, anomaly detection, threat management, incident response.

Матеріал надійшов 22.09.2024