DEVSECOPS

Una estrategia integral para el IoT



Mariana G. Sánchez

Director de tesina: Lic. Ricardo Brea

Instituto de Tecnología e Ingeniería
Universidad Nacional de Hurlingham (UNAHUR)

Esta tesina se presenta para el título de
Licenciatura en Informática

Universidad Nacional de Hurlingham Diciembre 2025

A mis hijos Celeste y Demián, que día a día, durante este largo camino fueron pacientes y comprensivos, entregándome su amor y su tiempo para lograr este objetivo, quienes fueron mi motor para no bajar nunca los brazos y por quienes en cada jornada, me esfuerzo para ser una mejor persona y una buena profesional.

A mi hermana Vanesa, fuente infinita de cariño, quien siempre confió en mis capacidades y me apoyó de todas las maneras que existen para perseguir este sueño.

A mis sobrinos Vero, Fer, Eli, Nicky y Luz, quienes me iluminan la vida a lo largo de las suyas. Son mi cable a tierra.

A mi cuñado Edgar, que más que un cuñado es un hermano, y a toda su familia por ser la mía. Especialmente a su mamá Isabel, quien llenó el vacío que dejó la sangre y me devolvió el amor.

Dedico este trabajo a mi familia querida por su bondad, por su amor, por el apoyo, por la confianza, por inculcarme la dignidad del trabajo y los valores éticos que me han hecho ser la persona que soy. Sin ustedes, nada hubiera sido igual, son mi mayor triunfo.

Agradecimientos

A mi director de tesis, Ricardo Brea, a quien admiro profundamente, como persona, como profesional y como docente, quien ha compartido con generosidad sus conocimientos, y ha sido mi guía con su ejemplar dedicación, compromiso y paciencia.

A mis compañeros y docentes que fueron parte de este proceso de aprendizaje.

A Fernando Puricelli, director del Instituto de Tecnología e Ingeniería, por su enorme esfuerzo, paciencia y dedicación a lo largo de estos años, trabajador incansable por la educación, la ciencia, la investigación y las oportunidades para toda la comunidad.

A la Universidad Nacional de Hurlingham, a sus autoridades y personal nodocente, por construir un espacio que cambia realidades y abre futuros. Donde mi camino académico fue posible gracias a las políticas de género e inclusión, que me brindaron el acceso a la beca mujeres, con la cual fue posible la continuidad y finalización de mis estudios.

Gracias infinitas por acompañarme y ser claridad en los momentos difíciles.

Abstract

In the present day, IoT has been consolidated as a component both in people's daily lives and in the productive processes of various industries. This advancement has driven an ecosystem whose technological heterogeneity and volume introduce significant challenges in the incorporation of effective measures to ensure its management, monitoring, updating, supervision, and intercommunication. Current infrastructures pose significant challenges for the implementation of security measures, such as the lack of resources, the absence of secure update systems, the lack of procedures, insufficient physical barriers, and/or weak passwords. Each of these factors makes devices susceptible to threats either individually or in combination. Among the fastest-growing threats is the *supply-chain attack*. This type of attack is capable of compromising the security of devices and systems at different stages of their lifecycle, deployment, and operation. This may involve data theft, the introduction of malware, or any other malicious activity. Another threat that can be observed is the phenomenon of *bricking* (malicious blocking). This blocking involves the manipulation of devices in such a way that they become inoperable, often through the removal of firmware or critical configurations, which allows attackers to force an unauthorized or uncontrolled shutdown of a device, potentially causing data loss or malfunction.

This thesis focuses on the implementation of DevSecOps in IoT to mitigate multiple security threats. DevSecOps is a comprehensive strategy that incorporates development, security, and operations from the earliest stages. By diligently integrating the latest security updates, it guarantees a proactive defensive posture against emerging threats and optimizes system resilience. The passionate adoption of these security challenges not only reflects a commitment to technical excellence but is also a strategic imperative to safeguard reliability and security in today's dynamic technological environments, additionally proposing, through infrastructure control, the possibility of customizing the environment and prioritizing autonomy.

Keywords: DevSecOps, IoT, MQTT, Supply-Chain Attack, Bricking, Botnet

Resumen

En la actualidad, el IoT se ha consolidado como un componente tanto en la vida de las personas como en los procesos productivos de diversas industrias. Este avance ha impulsado un ecosistema cuya heterogeneidad tecnológica y su volumen introducen desafíos importantes en la incorporación de medidas efectivas para garantizar su administración, seguimiento, actualización, supervisión e intercomunicación. Las infraestructuras actuales plantean retos significativos para la implementación de medidas de seguridad, tales como: la falta de recursos, la ausencia de sistemas de actualización seguros, la falta de procedimientos, la insuficiencia de barreras físicas y/o las contraseñas débiles. Cada uno de estos factores hace a los dispositivos susceptibles a amenazas ya sea por sí solos o combinados. Entre las amenazas de mayor crecimiento se encuentra *supply-chain attack* (ataque a la cadena de suministro). Este tipo de ataque es capaz de comprometer la seguridad de los dispositivos y sistemas en distintas etapas del ciclo de vida, despliegue y operación. Esto puede implicar el robo de datos, la introducción de malware o cualquier otra actividad maliciosa. Otra de las amenazas que podemos notar es el fenómeno del *bricking* (bloqueo malicioso). Este bloqueo involucra la manipulación de dispositivos de manera que se vuelvan inoperables, a menudo mediante la eliminación del firmware o de configuraciones críticas, lo que permite a los atacantes forzar un apagado no autorizado o incontrolado de un dispositivo, potencialmente causando la pérdida de datos o el mal funcionamiento.

Esta tesina se enfoca en la implementación de DevSecOps en IoT para mitigar múltiples amenazas a la seguridad. DevSecOps es una estrategia integral que incorpora desde las primeras etapas el desarrollo, la seguridad y las operaciones. Al integrar de manera diligente las últimas actualizaciones de seguridad se garantiza una postura defensiva proactiva contra las amenazas emergentes y se optimiza la resiliencia del sistema. La adopción apasionada de estos retos de seguridad no solo refleja un compromiso con la excelencia técnica, sino que también es un imperativo estratégico para salvaguardar la confiabilidad y la seguridad en los ambientes tecnológicos dinámicos de la actualidad, proponiendo además, mediante el control de la infraestructura, la posibilidad de personalizar el entorno y priorizar la autonomía.

Palabras clave: DevSecOps, IoT, MQTT, Supply-Chain Attack, Bricking, Botnet

Índice general

1. Introducción **1**
1.1. Descripción del problema . . . 1
1.2. Importancia del problema . . . 3
1.3. Objetivo del trabajo . . . 10
1.3.1. Objetivos específicos . . . 10

2. Metodología **13**
2.1. Enfoque metodológico . . . 13
2.2. Fuentes de información y alcance temporal . . . 13
2.3. Estrategia de búsqueda y criterios de inclusión . . . 14
2.4. Consideraciones éticas y de validez . . . 14

3. Marco teórico **15**
3.1. IoT . . . 15
3.1.1. Clasificación por capacidad y limitaciones de IoT . . . 15
3.2. Firmware . . . 16
3.2.1. Arquitectura del firmware . . . 16
3.3. Protocolos de red y comunicación . . . 18
3.4. Modelo OSI . . . 18
3.5. Capa 7 - Aplicación . . . 19
3.5.1. HTTP . . . 19
3.5.2. MQTT . . . 19
3.6. Capa 5-4 - Sesión/Transporte . . . 22
3.6.1. TLS . . . 22
3.7. Capa 4 - Transporte . . . 23
3.7.1. Transmission Control Protocol(TCP) . . . 23
3.8. Capa 3 - Red . . . 24
3.8.1. Internet Protocol versión 4 (IPv4) . . . 24
3.9. Capa 1/2 - Física/Enlace . . . 24
3.9.1. IEEE 802.11 (Wi-Fi) . . . 24

3.10. Edge Computing/Fog Computing . . . 25
3.10.1. Gateway . . . 27
3.11. Vulnerabilidades críticas en IoT . . . 28
3.11.1. Supply-chain - Supply-chain attack . . . 28
3.11.2. SBOM como mecanismo de mitigación para *supply-chain attack* . . . 29
3.11.3. Bricking . . . 30
3.12. DevOps . . . 30
3.12.1. Diferencias entre Integración Continua(CI), Entrega Continua(CD) y Despliegue Continuo(CD) . . . 32
3.13. DevSecOps . . . 33
3.13.1. Cultura DevSecOps . . . 34
3.13.2. Componentes de DevSecOps . . . 35
3.13.3. Herramientas de análisis DevSecOps . . . 36
3.13.4. Diferencias entre análisis . . . 36
3.13.5. Comparativa DevOps-DevSecOps . . . 37
3.13.6. Comparativa beneficios DevSecOps . . . 37

4. Desarrollo **39**
4.1. Alcance del proyecto . . . 39
4.2. Alcance específico del proyecto . . . 39
4.3. Limitaciones del proyecto . . . 40

5. Materiales y métodos **43**
5.1. Materiales . . . 43
5.1.1. Hardware . . . 43
5.1.2. Software . . . 44
5.1.3. Configuración del entorno de pruebas . . . 45
5.1.4. Entornos de despliegue y automatización . . . 46
5.2. Métodos . . . 47
5.2.1. Control de versiones - Branching . . . 47
5.2.2. Procedimientos de análisis de seguridad . . . 47
5.2.3. Configuración de pipelines CI/CD con seguridad integrada . . . 48
5.2.4. Estrategia DevSecOps . . . 48
5.2.5. Registro y reporte de hallazgos . . . 48
5.3. Análisis del dispositivo ESP8266 . . . 49
5.4. Análisis de riesgos del dispositivo . . . 49
5.5. Análisis del dispositivo ESP32 . . . 50
5.6. Análisis de riesgos del dispositivo . . . 51
5.7. Criterios de evaluación de riesgos . . . 53

5.7.1. Mapa de calor de probabilidad por impacto . . . 53
5.7.2. Clasificación de riesgo según el producto P×I . . . 54
5.7.3. Esquema de identificación y clasificación de riesgos . . . 54
5.8. Matriz de riesgos del dispositivo IoT . . . 54
5.8.1. Puntaje de vulnerabilidades IoT . . . 57

6. Arquitectura de la solución **59**
6.1. Diagrama de arquitectura . . . 59
6.2. Integración SonarCloud . . . 60
6.3. Integración GitHub actions . . . 62
6.3.1. Manejo de Tokens . . . 63
6.4. Integración DefectDojo . . . 64
6.5. Integración en ThingsBoard . . . 66
6.6. Integración de PI-Hole . . . 71
6.7. Detección y monitoreo . . . 72
6.7.1. Seguimiento y métricas . . . 73
6.7.2. Telemetría y registros . . . 73
6.8. Conectividad como vector de ataque y segmentación . . . 75
6.8.1. Características de red por defecto . . . 75
6.8.2. Matriz de riesgos de la red . . . 76
6.8.3. Características de red con seguridad . . . 77
6.8.4. Comparación entre topologías . . . 77

7. Modelo de amenazas **79**
7.1. Activos críticos . . . 79
7.1.1. Roles y actores fuente de amenazas . . . 80
7.1.2. Vectores y puntos de entrada principales . . . 81
7.1.3. Mapeo *STRIDE* . . . 81
7.2. Árboles de ataque . . . 82

8. Casos prácticos **85**
8.1. Caso A — Escaneo de superficie y descubrimiento . . . 85
8.2. Caso B — Captura y análisis de tráfico *(confidentiality/MITM)* . . . 85
8.3. Caso C — Prueba OTA maliciosa / *fallback recovery (bricking)* . . . 86
8.4. Caso D — Compromiso del *pipeline* (*supply chain*/repositorio) . . . 86
8.5. Caso E — Fuerza bruta / panel administrativo . . . 87
8.6. Caso F — DDoS sobre el dispositivo . . . 87
8.7. Caso G — Acceso físico (UART/JTAG) . . . 87
8.8. *Mapping* entre la matriz de riesgos y los casos de prueba . . . 88
8.9. Mitigación completa . . . 88

8.10. Inferencia 92
8.10.1. Integración y mitigación continua DevSecOps 92

9. Resultados de los experimentos **93**
9.1. Resultados cuantitativos 93
9.1.1. Riesgos residuales de red 93
9.1.2. Riesgos residuales del dispositivo 93

10. Conclusiones **97**

11. Problemas encontrados **99**

12. Futuras líneas de investigación y mejora **101**

Bibliografía **103**

Apéndice A. Terminología **107**

Apéndice B. Evidencias **115**
B.1. Imágenes de pruebas y configuraciones 116
B.1.1. Configuraciones de red 116
B.2. Casos prácticos 118
B.2.1. Caso A - Escaneo de superficie y descubrimiento 118
B.2.2. Caso B — Captura y análisis de tráfico (confidentiality/MITM) 122
B.2.3. Caso C — Prueba OTA maliciosa / fallback recovery (bricking) 125
B.2.4. Caso D — Compromiso del pipeline (supply chain/repositorio) 125
B.2.5. Caso E — Fuerza bruta / panel administrativo 129
B.2.6. Caso F — DDoS sobre el dispositivo 132

Apéndice C. Procedimiento de buenas prácticas para IoT **137**
C.1. Accesos lógicos no autorizados 137
C.1.1. Gestión de cuentas y credenciales 137
C.1.2. Monitoreo y correlación de eventos 138
C.2. Actualizaciones y ciclo de vida seguro 138
C.2.1. Política de actualizaciones y parches 138
C.2.2. Verificación post-actualización 138
C.3. Conectividad a internet y gateway (Zero Trust Networking) 138
C.3.1. *Firewalls*, filtrado y segmentación 138
C.3.2. VPN y canales cifrados 138
C.4. Sistema operativo y hardening 139
C.4.1. Selección segura 139

C.4.2. Hardening del sistema . . . 139
C.5. Contenedores . . . 139
C.5.1. Configuración segura de contenedores . . . 139
C.5.2. Ciclo de vida de contenedores . . . 139
C.6. Tipo y seguridad de los dispositivos IoT . . . 139
C.6.1. Evaluación y requisitos de seguridad . . . 139
C.7. Particionado de memoria y protección en dispositivos . . . 140
C.7.1. Política de particionado . . . 140
C.7.2. Optimización y seguridad . . . 140
C.8. Revisiones, auditorías y mejora continua . . . 140
C.8.1. Revisiones periódicas . . . 140
C.8.2. Auditorías internas y externas . . . 140
C.9. Acceso físico no autorizado . . . 140
C.9.1. Control de acceso físico . . . 140
C.9.2. Registro de accesos . . . 140
C.10. Controles técnicos recomendados . . . 141
C.11. Checklist de verificación y criterios de aceptación . . . 141
C.12. Plantilla para reportar hallazgos en DefectDojo . . . 142

Índice de figuras

1.1. Cronología del IoT 1
1.2. Superposiciones de IoT con otros campos de investigación 3
1.3. Brechas existentes antes de la pandemia 5
1.4. Eras de la evolución 6
1.5. Mapa ETSI 8
1.6. Informe anual ENISA 9
1.7. OWASP IoT Top10 9

3.1. Arquitectura de firmware 17
3.2. Configuración de conexión RFC 9431 20
3.3. Escenario modelo MQTT 21
3.4. Escenario Edge Computing 26
3.5. Comunicación de dispositivo a Gateway 27
3.6. Supply Chain típica IoT 28
3.7. Proceso continuo de implementación DevOps 31
3.8. Continuos Integration, Continuos Delivery, Continuos Deployment 33
3.9. Proceso continuo de implementación DevSecOps 34
3.10. Diferencias entre análisis IAST, DAST Y SAST 36
3.11. Comparación entre DevOps y DevSecOps 37
3.12. Beneficios DevSecOps 37

5.1. Dispositivos de IoT ESP32 y ESP8266 43

6.1. Diagrama de arquitectura 59
6.2. Construcción de pipeline SAST. 60
6.3. Ingreso a SonarCloud 61
6.4. Vista principal del proyecto en SonarCloud. 61
6.5. Análisis superado en SonarCloud. 62
6.6. Release exitoso en *GitHub Actions* tras cumplir los *quality gates*. 63
6.7. Configuración de tokens utilizados en la integración CI. 64

6.8. Configuración de tokens para ThingsBoard 64
6.9. Inicio de sesión de DefectDojo. 65
6.10. Dashboard DefectDojo. 65
6.11. Integración activa entre SonarCloud y DefectDojo. 66
6.12. Diagrama de estados de IoTWebConf 67
6.13. Login ThingsBoard 68
6.14. ThingsBoard MFA 68
6.15. Diagrama de secuencia ThingsBoard 69
6.16. Diagrama de estados de conexión 70
6.17. Pantalla de autenticación de Pi-hole. 71
6.18. Dashboard de Pi-hole 72
6.19. *E-mail* enviado automáticamente desde *SonarCloud* ante detección de actividad. . . . 72
6.20. Dashboard DefectDojo con *findings* importados 73
6.21. Plataforma *ThingsBoard* 73
6.22. ThingsBoard vista de actividad 74
6.23. Envío de datos 74
6.24. Actualizaciones OTA 75

7.1. Jerarquía descendente de actores de amenaza. 80

8.1. Diagrama de secuencia con aplicación DevSecOps 89

B.1. Configuración de red 116
B.2. Configuración de servicios 117
B.3. Configuraciones de IP 118
B.4. Escaneo de puertos 118
B.5. Escaneo de rutas 119
B.6. Configuración de ESP32 119
B.7. Vista de tecnologías 120
B.8. Vista de tecnologías 121
B.9. Vista de dominios 121
B.10. Vista de IP dominios 122
B.11. Captura de tráfico en Wireshark generada por el dispositivo. 122
B.12. Credenciales observadas en texto plano dentro del tráfico capturado. 123
B.13. Capturas de tráfico 123
B.14. Capturas de tráfico 124
B.15. Implementación de TLS para proteger el tráfico MQTT. 124
B.16. Actualizaciones OTA 125
B.17. Resultados de análisis estático con Cppcheck. 126
B.18. Problemas detectados en SonarCloud. 126

B.19. Detección de contraseñas codificadas por SonarCloud. 127
B.20. Mantenibilidad del código en SonarCloud. 127
B.21. Release bloqueado en GitHub Actions debido a errores previos. 128
B.22. Pipeline de build. 128
B.23. PR corregido luego de resolver issues detectados. 129
B.24. Ejecución del ataque de fuerza bruta sobre el servicio del dispositivo. 130
B.25. Conjunto de experimentos realizados durante la prueba de fuerza bruta. 130
B.26. Iteración avanzada del ataque de fuerza bruta. 131
B.27. Iteración: 200 request 132
B.28. Iteración; 300 request 132
B.29. Iteración: 500 request 132
B.30. Slowhttptest 133
B.31. Slowhttptest. 134
B.32. Slowhttptest Informe HTML 134
B.33. DDOS completado 135

Índice de tablas

1.1. IoT Provisiones del mercado mundial - [4] 4
1.2. Verificación de contraseñas - [5] 4
1.3. Promedio de ataques IoT 2023 vs 2022 - [6] 4
1.4. Ataques IoT relevantes por Sectigo 7

2.1. Criterios de selección de fuentes documentales. 14

3.1. Clases de dispositivos restringidos (KiB = 1024 bytes) 16
3.2. Pila de protocolos utilizada según el modelo OSI 19
3.3. Niveles de Calidad de Servicio (QoS) en MQTT 20
3.4. Propiedades fundamentales del protocolo TLS 23
3.5. Edge Computing y Fog Computing según organizaciones relevantes 27
3.6. Tipos de operaciones en la red de suministro de dispositivos 29
3.7. Causas comunes de *bricking* en dispositivos 30
3.8. Descripción de los conceptos DevSecOps 34
3.9. Descripción de los conceptos SAST, IAST y DAST 36

5.1. Control de versiones 47
5.2. Especificaciones del módulo ESP8266 50
5.3. Especificaciones del módulo ESP32-S3 parte 1 51
5.4. Especificaciones del módulo ESP32-S3 parte 2 52
5.5. Escala de probabilidad 53
5.6. Escala de impacto 53
5.7. Mapa de calor de probabilidad vs. impacto 53
5.8. Clasificación de riesgo según el producto P×I 54
5.9. Esquema de identificación de riesgos 54
5.10. Matriz de riesgos ESP8266 Parte 1 54
5.11. Matriz de riesgos ESP8266 Parte 2 55
5.12. Matriz de riesgos ESP8266 Parte 3 56
5.13. Puntajes de vulnerabilidades 57

6.1. Descripción de los estados de actividad del dispositivo 71
6.2. Matriz de riesgos de la topología de red por defecto y mitigaciones 76
6.3. Puntajes de vulnerabilidades de la topología de red 77
6.4. Comparación entre topología por defecto y red mínima de seguridad propuesta . . . 78

7.1. Activos críticos dentro del ciclo de vida seguro del *firmware* de IoT 79
7.2. Superficies de ataque identificadas en el ecosistema 81

9.1. Comparación de riesgo inicial vs residual tras aplicar la red mínima segura 93
9.2. Comparación de riesgo inicial vs residual tras aplicar mitigaciones en dispositivo . . 94
9.3. Comparación de riesgo inicial vs residual tras aplicar controles DevSecOps 95

Capítulo 1

Introducción

1.1. Descripción del problema

El IoT (Internet of Things) se refiere a la red de objetos físicos que se conectan a Internet mediante software, sensores y otras tecnologías. Esto permite que los dispositivos “inteligentes” recopilen e intercambien datos, lo que facilita la comunicación y la automatización. El IoT se extiende más allá de las computadoras y los teléfonos inteligentes para abarcar objetos cotidianos como electrodomésticos, vehículos e incluso ropa. Esta interconexión permite el monitoreo, control y análisis remotos, lo que impulsa la eficiencia, la conveniencia y la innovación en varios sectores, desde hogares inteligentes y atención médica, hasta fabricación y transporte.

El ecosistema del IoT es inherentemente variado. En correlación con el aumento de los dispositivos y su complejidad, también aumentan las vulnerabilidades y ataques informáticos. Dada su popularidad actual, propongo hacer un recorrido para entender sus comienzos y así comprender su futuro.

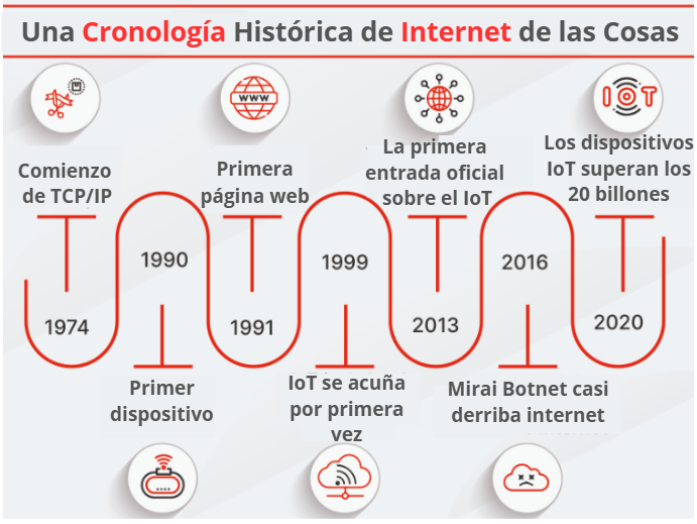


Figura 1.1 Cronología del IoT

1974: El comienzo de TCP/IP

En 1974, Robert E. Kahn y Vinton Cerf inventaron el Protocolo de Internet (IP) y el Protocolo de Control de Transmisión (TCP). La IP se asegura de que los datos pasen de un lugar a otro sin perder ninguno a lo largo del camino. El TCP ayuda a reensamblar los datos en lo que era antes de comenzar su trayectoria.

1990: El primer dispositivo IoT

En 1990, John Romkey creó un tostador que podía encenderse y apagarse a través de Internet. Lo hizo y lo presentó en la conferencia INTEROP de octubre.

1991: La primera página web

Tim Berners-Lee creó la primera página web en 1991 y tenía la dirección `info.cern.ch`.

1999: El IoT se acuña por primera vez

Kevin Ashton acuñó por primera vez el término "IoT" en 1999 mientras daba una presentación en PowerPoint para Procter and Gamble. Propuso dispositivos que podrían detectar su entorno utilizando sensores y software.

2013: La primera entrada oficial de IoT

Aunque el término ya había existido durante 14 años, el Diccionario de Oxford agregó "IoT" a sus páginas en 2013. En aquel momento, Helen Duce escribió: "Tenemos una visión clara, crear un mundo donde cada objeto, desde los jets jumbo hasta las agujas de costura, esté vinculado a Internet".

2016: La botnet Mirai casi derriba Internet

La botnet Mirai infectó más de 600,000 dispositivos con malware en 2016 y luego ejecutó un enorme ataque de denegación de servicio distribuido (DDoS) utilizando los dispositivos.

2020: Los dispositivos de IoT superan los 20 mil millones

Hay más de 20 mil millones de dispositivos conectados a Internet y ese número continuará creciendo durante la próxima década. Esto se debe principalmente al valor que muchas empresas ponen en la producción de productos y servicios que dependen o aprovechan en gran medida la conectividad a Internet. Pronto, un dispositivo eléctrico podrá parecer obsoleto si al menos no permite una conexión a Internet.[1]

El documento de la IEEE *"Hacia una definición del Internet de las cosas (IoT)"* describe el IoT como un paradigma global en el cual objetos físicos, equipados con sensores, actuadores o capacidades de computación, quedan conectados entre sí y con redes más amplias mediante protocolos de red estándares. Esta interconexión transforma objetos cotidianos en entidades inteligentes, capaces de recolectar datos del entorno, comunicarse, y participar en flujos de información en tiempo real — extendiendo la Internet tradicional hacia un entramado mucho más diverso. El informe indica que IoT no es una tecnología aislada, sino un espacio de convergencia: integra redes, dispositivos, servicios en la nube, sistemas embebidos y entornos físicos, superponiéndose con áreas como redes de sensores, computación ubicua, sistemas cibernéticos y ecosistemas en la nube (véase la Figura 1.2).[2]

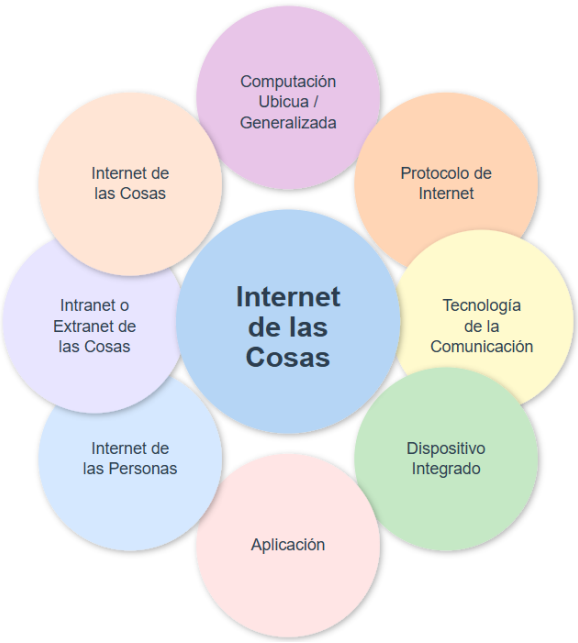


Figura 1.2 Superposiciones de IoT con otros campos de investigación

El crecimiento exponencial no es un fenómeno exclusivo del IoT. Otras tecnologías han pasado por una transformación digital similar. Un ejemplo es la transformación de sistemas de arquitecturas monolíticas en arquitecturas distribuidas. Esta transformación es similar al IoT porque incorpora dos factores determinantes: cantidad y heterogeneidad. Tomando esta experiencia como ejemplo es relevante incorporar prácticas similares para evitar tener los mismos problemas. La técnica de seguridad más relevante al momento es DevSecOps.[3]

DevSecOps es un marco de trabajo, que integra pruebas de seguridad en cada etapa del proceso de desarrollo de software. Incluye herramientas y procesos que fomentan la colaboración entre desarrolladores, especialistas en seguridad y equipos operativos para crear software que sea a la vez eficiente y seguro. DevSecOps aporta una transformación cultural que hace que la seguridad sea una responsabilidad compartida para todos los que crean el software.

1.2. Importancia del problema

Durante los últimos años se han observado aumentos en la adopción de las tecnologías de IoT tanto por parte de usuarios finales, como de industrias. La información proporcionada en los cuadros 1.1 y 1.2, brinda herramientas para comprender el crecimiento en el uso y adopción de las mismas, así como también los comportamientos de los consumidores en relación con la seguridad del IoT.

La inclusión de estos informes pretende proporcionar un marco contextual que resalte la importancia de la seguridad en el desarrollo y adopción de IoT, ofreciendo una perspectiva basada en datos empíricos recolectados directamente de los usuarios finales.

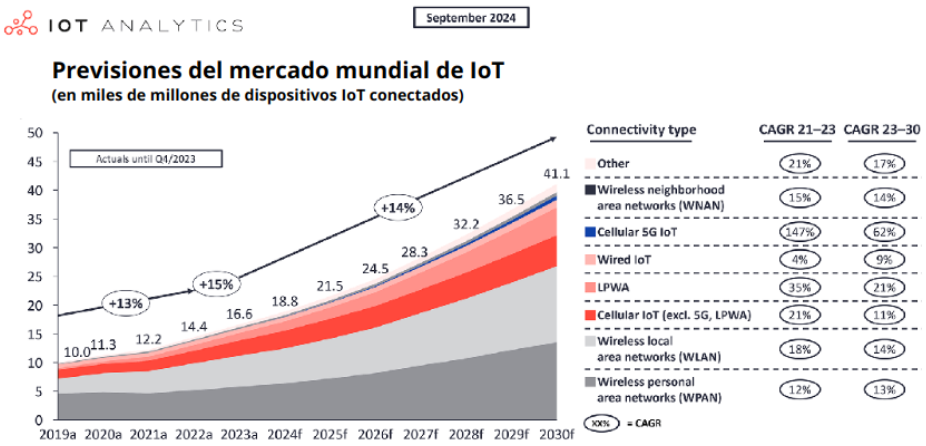


Tabla 1.1 IoT Provisiones del mercado mundial - [4]



Tabla 1.2 Verificación de contraseñas - [5]

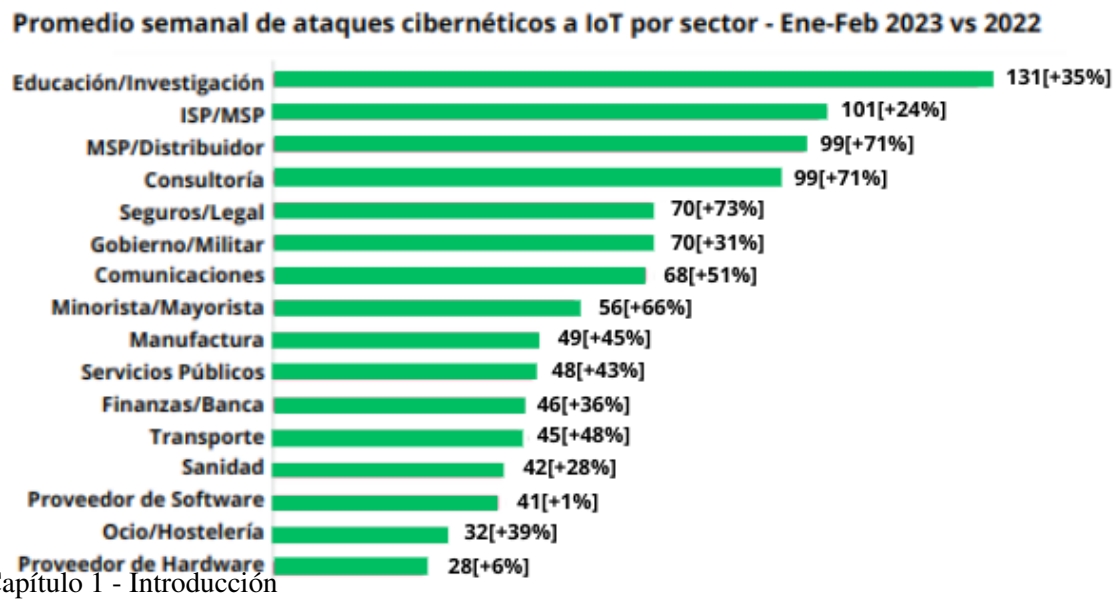


Tabla 1.3 Promedio de ataques IoT 2023 vs 2022 - [6]

El incremento en la adopción de dispositivos IoT fue más notorio desde la pandemia COVID-19 y podría creerse que las vulnerabilidades en las tecnologías comenzaron allí, pero esto no es cierto.

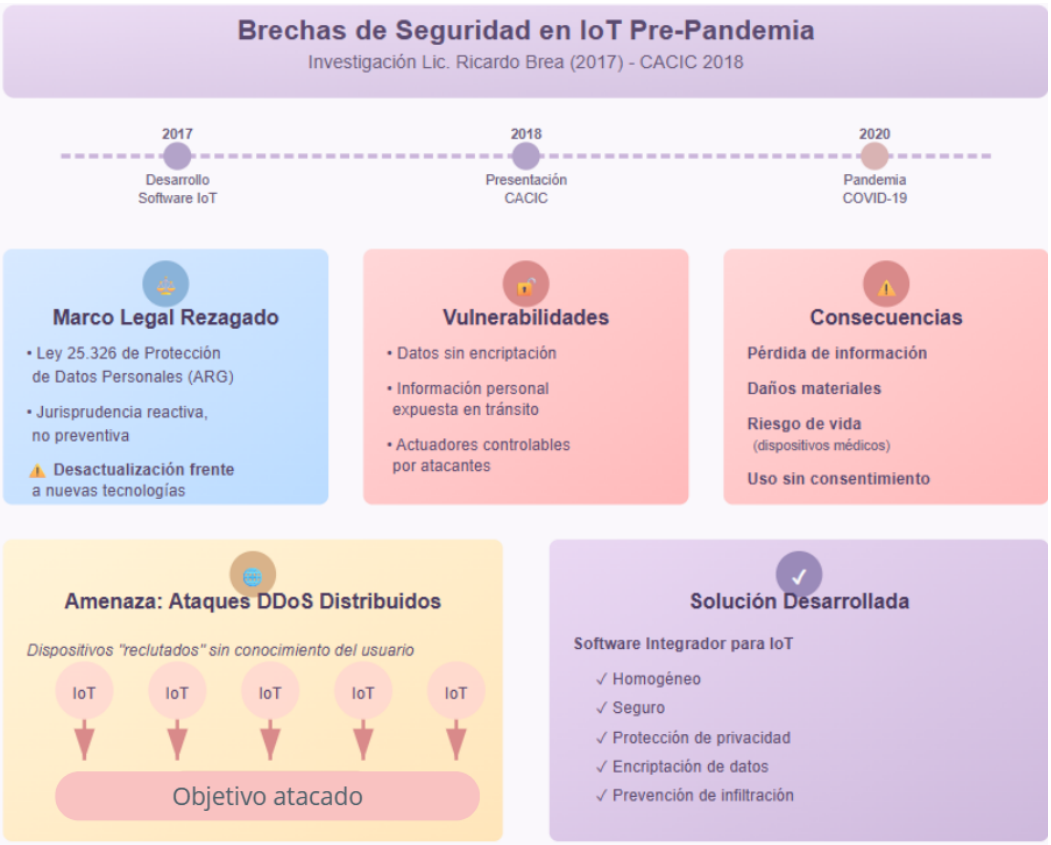


Figura 1.3 Brechas existentes antes de la pandemia

Las brechas de seguridad son anteriores a la pandemia, por este motivo, resulta pertinente citar el estudio de campo que realizó el Lic. Ricardo Brea en el cual ha desarrollado un software integrador homogéneo y seguro para IoT, culminado en 2017. Dentro de su investigación se encuentran las brechas acerca de privacidad, contemplando que en Argentina el uso de los datos está regulado por la ley de protección de datos personales (Ley 25.326), pero la jurisprudencia no se adelanta a la tecnología, sino que actúa como consecuencia de ella, y por lo tanto queda desactualizada frente a nuevas tecnologías. Destaca que los dispositivos IoT son susceptibles a ataques cibernéticos al no enviar la información encriptada, lo cual es un riesgo grave teniendo en cuenta que recolectan información de la vida de las personas y algunos poseen actuadores que encienden diversos elementos. Esto significa que un ataque cibernético a un dispositivo IoT puede causar no solo la pérdida de información, sino también una pérdida material, o en el peor de los casos causar la muerte si el dispositivo atacado es un elemento que monitorea los signos vitales de una persona enferma. Un caso especial en la seguridad de IoT sucede cuando un dispositivo es infiltrado y utilizado sin el conocimiento del usuario, para atacar a un tercero. IoT se convierte en un blanco ideal de ataques

informáticos por la gran cantidad de dispositivos que pueden ser “reclutados” a fin de infiltrarlos para un ataque de denegación de servicio. Este trabajo fue presentado en el XXIV Congreso Argentino de Ciencias de la Computación (CACIC 2018).[7]

Las soluciones que existían en 2017/2018 para el uso de IoT no se encontraban integradas, eran inseguras o no eran transparentes en el manejo de la información. En la actualidad transitando el 2024, estas problemáticas siguen vigentes.

Dentro de la presente investigación, se examinó el artículo ”Regulación mediante bricking: ordenamiento privado en IoT”[8] , el cual se refiere al deterioro o destrucción deliberada del software con la intención de afectar negativamente la funcionalidad del producto. El artículo argumenta y sostiene que se está empleando bricking a través de la regulación post-compra de bienes conectados a Internet, y que las empresas del “Internet de las Cosas” tienen una capacidad injusta para imponer sus políticas preferidas de manera unilateral, automática y remota. Induciendo que el control sobre el software permite por tanto, el control sobre el hardware.

Las brechas de seguridad van dejando el paso abierto a que se consoliden diferentes vulnerabilidades. La empresa proveedora de soluciones de seguridad Sectigo, indica que la evolución de los ataques a dispositivos IoT se puede dividir en tres eras (véase la Figura 1.4). [9]



Figura 1.4 Eras de la evolución

Sectigo informa sobre la batalla entre los ciberataques y las tecnologías de seguridad, evidenciando el incremento y recurrencia de las *Botnets* (véase la Tabla 1.4).

Primera Era – *The Age of Exploration* (2005–2009)	
Stuxnet Worm	Malware que manipulaba PLCs industriales de Siemens en Irán.
Puerto Rico Power Meter Hack	Fraude en medidores eléctricos modificando el *firmware*.
Mexican Oil Platform Hack	Intrusión remota en sistemas de control de plataformas petroleras.
Michigan Traffic Lights Hack	Control de semáforos mediante comunicaciones de radio inseguras.
Segunda Era – *The Age of Exploitation* (2011–2019)	
Illinois Water Utility Hack	Acceso remoto a SCADA; quemaron una bomba de agua.
Baby Monitor Hack	Accesos no autorizados a monitores IP.
Ukrainian Power Grid Attack	Apagones mediante control remoto de infraestructura eléctrica.
Lenovo Firmware Vulnerability	*Backdoor* persistente que reinstalaba malware.
German Steel Mill Hack	Daño físico en alto horno por intrusión en planta.
Tesla Model S Hack	Control remoto del vehículo.
Jeep Cherokee Hack	Control remoto por vulnerabilidad *Uconnect*.
Smart Refrigerator Botnet	Heladeras infectadas enviando *spam*.
Mirai Botnet	*Botnet* masiva de IoT por contraseñas débiles.
BrickerBot	Destrucción permanente de dispositivos IoT (*PDoS*).
Reaper Botnet	*Botnet* explotando vulnerabilidades reales.
VPNFilter Botnet	Infección de *routers*/NAS con espionaje y destrucción.
BlueBorne	Ejecución remota por *Bluetooth* sin emparejar.
Amazon Ring Hack	Acceso a cámaras por credenciales filtradas.
Casino Fish Tank Thermometer Hack	Termómetro IoT usado como punto de entrada para robo de datos.
Google Home / Chromecast Leak	Filtración que permitía ubicar físicamente al usuario.
Two Million IoT Device Takeover	Toma masiva de IoT formando gran *botnet*.
Tercera Era – *The Age of Protection* (2020→)	
Building Control / Chemical Plant Attack	Manipulación de sistemas industriales y de climatización.
MedJack	Compromiso de dispositivos médicos con robo de datos.
Blue Link Car Remote Attack	Acceso remoto al vehículo por API débil.
Smart Meter Exploits	Manipulación de contadores eléctricos.
Swarmbot / Swarm Attacks	Enjambres de IoT coordinados para ataques avanzados.

Tabla 1.4 Ataques IoT relevantes por Sectigo

Al avanzar en el desarrollo de este proyecto surgió la necesidad de identificar las principales normativas, regulaciones y entidades que intervienen en la protección de los datos y la seguridad en dispositivos IoT. A nivel internacional existen diferentes organizaciones gubernamentales, de estandarización y fundaciones especializadas que establecen marcos regulatorios, definen buenas prácticas y publican lineamientos para mejorar la ciberseguridad. Entre las más relevantes se encuentran el GDPR[10], ETSI [11], ENISA[12] y OWASP[13], descritos a continuación junto con sus respectivos aportes. [14]

- **GDPR (General Data Protection Regulation).** Pertenece al Consejo de la Unión Europea y es reconocido como el marco regulatorio de privacidad y seguridad más estricto a nivel internacional. Actualizó la directiva de protección de datos en 2016 y entró en vigencia el 25

de mayo de 2018. Define los derechos fundamentales de los individuos, las obligaciones de quienes procesan datos, los mecanismos de cumplimiento y las sanciones. Si bien constituye una base sólida, no cubre todas las particularidades asociadas al IoT.

- **ETSI (European Telecommunications Standards Institute).** Es el organismo regional encargado de la normalización en telecomunicaciones, radiodifusión y servicios electrónicos. Reúne más de 850 organizaciones miembros provenientes de más de 60 países y cinco continentes (véase la Figura 1.5). Desde 2020 proporciona un estándar específico para dispositivos conectados, cuyo objetivo es establecer disposiciones de alto nivel sobre seguridad y protección de datos en el entorno IoT. Este estándar incluye directrices concretas para organizaciones dedicadas al diseño o fabricación de dispositivos IoT de consumo.

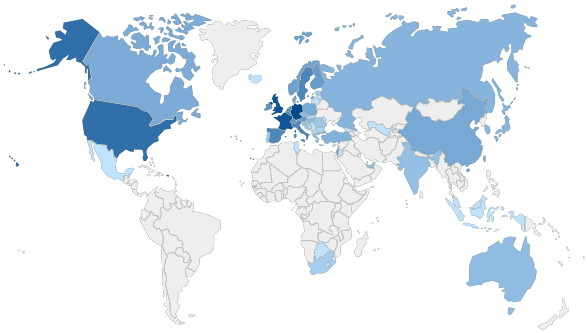


Figura 1.5 Mapa ETSI

- **ENISA (European Union Agency for Cybersecurity).** Esta agencia trabaja en mejorar la ciberseguridad en los Estados miembros de la Unión Europea y fortalecer la capacidad de respuesta ante incidentes. Su informe anual sobre el panorama de amenazas identifica tendencias, actores maliciosos, técnicas de ataque, impacto y recomendaciones de mitigación (véase la Figura 1.6).



Figura 1.6 Informe anual ENISA

- **OWASP (Open Worldwide Application Security Project).** Es una fundación sin fines de lucro dedicada a mejorar la seguridad del software. Publica el reconocido ranking de las 10 vulnerabilidades más frecuentes (véase la Figura 1.7), utilizado como referencia en auditorías y análisis de seguridad, manteniendo un listado específico de vulnerabilidades comunes en dispositivos IoT, brinda descripciones, riesgos asociados y recomendaciones de mitigación.



Figura 1.7 OWASP IoT Top10

Investigando acerca de la actualidad para conocer si han surgido nuevas soluciones a las problemáticas de IoT. Existe un trabajo de investigación que analiza los desafíos que enfrentan los Security Operations Centers (SOC) al incorporar dispositivos IoT en sus procesos de monitoreo, detección y respuesta. Mediante una investigación cualitativa basada en entrevistas con equipos SOC de distintos sectores, el autor identifica dificultades como la heterogeneidad tecnológica, la falta de estandarización, la ausencia de visibilidad del inventario IoT y la complejidad para integrar telemetría relevante a sistemas SIEM. Este trabajo constituye un antecedente directo para la presente investigación, dado que también aborda la seguridad en entornos IoT, la necesidad de mecanismos de monitoreo adecuados y la importancia de comprender las vulnerabilidades y riesgos asociados a dispositivos distribuidos.[15].

Por último, pero no menos importante, se encuentra un trabajo de campo vinculado con la aplicación de metodologías ágiles y automatización en entornos IoT. El estudio *Ïmplementación de prácticas DevOps para la integración y el despliegue continuos de software en sistemas IoT"* (2025), analiza la incorporación de principios DevOps en proyectos basados en dispositivos embebidos y arquitecturas distribuidas. Este trabajo propone herramientas de integración y entrega continua (CI/CD), control de versiones, y análisis de calidad de código para optimizar el ciclo de vida del desarrollo de firmware en dispositivos IoT. Plantea la relevancia de la automatización del despliegue, la gestión de la configuración y la observabilidad del sistema, destacando la necesidad de sinergia entre los equipos de desarrollo y operaciones, demostrando la viabilidad de trasladar las prácticas DevOps a entornos con recursos limitados, integrados con servicios, protocolos de comunicación MQTT y con actualizaciones mediante OTA(Over The Air)[16]. Este trabajo será tomado como punto de partida, en conjunto con los conceptos teóricos desarrollados en el presente capítulo, para la aplicación de un enfoque DevSecOps orientado a ecosistemas IoT.

1.3. Objetivo del trabajo

La escasez de soluciones y mitigaciones tempranas a las vulnerabilidades de las tecnologías IoT, que avanzan a mayor velocidad que las medidas adecuadas de contención en temas referentes a la seguridad en los dispositivos de comercialización, se convierten en la motivación necesaria para realizar el presente trabajo.

Esta tesina propone una solución a las problemáticas actuales, mediante la aplicación de la metodología de trabajo DevSecOps mitigando así las brechas de seguridad existentes y emergentes.

1.3.1. Objetivos específicos

1. Relevar y sistematizar las principales vulnerabilidades de seguridad presentes en dispositivos y ecosistemas IoT, considerando tanto configuraciones por defecto, protocolos de comunicación y gestión de credenciales.

2. Analizar metodologías y frameworks de seguridad aplicables al ciclo de vida del software en entornos IoT, con foco en *DevSecOps*, integración continua (CI), entrega continua (CD) e implementación continua (CD).

3. Diseñar e implementar *pipelines* de DevSecOps adaptado a entornos IoT, incorporando herramientas de análisis estático, análisis dinámico, gestión de dependencias, escaneos y pruebas de seguridad automatizadas.

4. Ejecutar escenarios de validación en entornos reales, que permitan medir de manera objetiva las brechas de seguridad detectadas en dispositivos IoT antes y después de aplicar la metodología propuesto(Exceptuando limitaciones).

5. Cuantificar las mejoras obtenidas en términos de reducción de vulnerabilidades, tiempos de detección, y nivel de exposición a ciberataques, comparando métricas de seguridad iniciales y finales.

6. Proponer un modelo de buenas prácticas y lineamientos de seguridad para organizaciones que desarrollen, integren o administren dispositivos IoT, a partir de la experiencia obtenida en la demostración práctica.

Capítulo 2

Metodología

2.1. Enfoque metodológico

La presente investigación adopta un enfoque documental, orientado a identificar, analizar y categorizar la literatura científica, técnica y normativa relevante sobre *dispositivos IoT, seguridad de firmware, protocolos de comunicación, ataques, vulnerabilidades y prácticas DevSecOps*.

El objetivo principal de este enfoque es establecer el estado del arte y las tendencias emergentes en torno a la integración de la seguridad en el ciclo de vida del desarrollo de dispositivos conectados.

2.2. Fuentes de información y alcance temporal

El corpus documental se compone de fuentes publicadas entre **2015 y 2025**, incluyendo:

- **Artículos científicos y conferencias** indexadas en *IEEE Xplore*, *ScienceDirect* y *ACM Digital Library*.
- **Informes técnicos y whitepapers** emitidos por organismos de estandarización y agencias internacionales como *IETF*, *ISO*, *ETSI*, *ENISA*, *IEEE*, *IoT Security Foundation (IoTSF)* y la *Connectivity Standards Alliance (CSA)*.
- **Informes de la industria y estudios de mercado** de entidades como *Fortinet*, *Check Point*, *AWS*, *Microsoft*, *AVG*, *RedHat* e *IBM*, que aportan evidencia empírica sobre vulnerabilidades, amenazas y adopción tecnológica.
- **Literatura académica aplicada**, incluyendo tesis de grado y maestría relevantes (p. ej., R. Brea, 2017; Jørgensen, 2023; M. Acuña, 2025).
- **Documentación técnica y especificaciones de protocolos** (RFCs, ISO/IEC, IEEE 802.11, MQTT v5.0).

El rango temporal 2015–2025 se seleccionó por incluir los principales hitos en la madurez del ecosistema IoT, desde la publicación de *"Towards a Definition of the Internet of Things"* (IEEE, 2015) hasta la *"Implementación de prácticas DevOps para la integración y el despliegue continuos de software en sistemas IoT"* (M. Acuña, 2025), y los informes de amenazas más recientes.

2.3. Estrategia de búsqueda y criterios de inclusión

La búsqueda se realizó mediante motores académicos y técnicos (*Google Scholar, IEEE Xplore, ScienceDirect, GitHub, repositorios institucionales y documentación de fabricantes*), empleando operadores booleanos y técnicas de búsqueda avanzada, tales como:

```
("Internet of Things" OR IoT) AND ("firmware security" OR "secure boot")
("DevSecOps" OR "software supply chain" OR "bricking" OR "botnet") AND IoT
(site:ietf.org OR site:etsi.org)
("MQTT" OR "CoAP" OR "Thread") AND (security OR encryption) AND (protocol OR RFC)
```

Los criterios de selección se definieron de acuerdo con estándares de calidad científica y técnica:

Criterio	Descripción
Autoridad	Preferencia por publicaciones emitidas por organismos de estandarización (IEEE, ETSI, IETF), agencias de ciberseguridad (ENISA), o instituciones académicas reconocidas.
Actualidad	Documentos publicados o actualizados entre 2015 y 2025.
Relevancia temática	Alineación directa con IoT, Seguridad, DevSecOps.
Verificabilidad	Fuentes verificables mediante DOI, ISBN o URL de sitios oficiales.
Diversidad	Inclusión de fuentes académicas, normativas, industriales y de divulgación técnica.

Tabla 2.1 Criterios de selección de fuentes documentales.

2.4. Consideraciones éticas y de validez

Se respetaron los principios de integridad académica, citando todas las fuentes conforme a las normas *APA 7ª edición*. La validez de la información recolectada se garantizó mediante corroboración de datos y contraste con estándares internacionales.

Capítulo 3

Marco teórico

3.1. IoT

La IETF *(Internet Engineering Task Force)* define el IoT *(Internet of Things)* como una red de dispositivos físicos (objetos, sensores, dispositivos electrónicos, software, etc.) que están conectados a Internet y son capaces de recopilar, intercambiar y procesar datos a través de la red sin intervención humana. Estos dispositivos pueden interactuar entre sí, actuar automáticamente y ser controlados de manera remota mediante la infraestructura de internet. Un documento clave de la IETF sobre IoT es el RFC 7452[17], que describe los requisitos y desafíos técnicos relacionados con la conectividad, la seguridad y la interoperabilidad de los dispositivos IoT. Se enfatiza la importancia de estándares abiertos, seguridad robusta y escalabilidad para manejar la vasta cantidad de dispositivos que estarán conectados en el futuro. El IoT según la IETF, se refiere a un ecosistema de objetos conectados y autónomos que interactúan a través de internet para realizar tareas inteligentes y automatizadas.

3.1.1. Clasificación por capacidad y limitaciones de IoT

La IETF, a través del RFC 7228,[18], clasifica los dispositivos de IoT en tres clases principales basadas en sus limitaciones de recursos:

- **Clase 0:** Dispositivos extremadamente restringidos. Tienen capacidades de procesamiento y memoria muy limitadas, lo que les impide manejar comunicaciones seguras directamente con Internet. Generalmente, dependen de puertas de enlace o servidores proxy para la comunicación.
- **Clase 1:** Estos dispositivos son un poco más potentes que los de C0, pero aún limitados. Pueden manejar protocolos ligeros como CoAP y DTLS para la comunicación segura, pero no son capaces de utilizar protocolos más complejos como HTTP o TLS sin asistencia de una puerta de enlace.
- **Clase 2:** Dispositivos con muchas menos restricciones, similares a los teléfonos móviles o portátiles en términos de capacidades de procesamiento. Son capaces de manejar protocolos

más robustos y completos sin necesidad de intermediarios, y tienen un mayor potencial para interoperabilidad y soporte de aplicaciones complejas.

Nombre	Tamaño de datos (Ej. RAM)	Tamaño de código (Ej. Flash)
Clase 0 (C0)	$\ll$ 10 KiB	$\ll$ 100 KiB
Clase 1 (C1)	$\sim$ 10 KiB	$\sim$ 100 KiB
Clase 2 (C2)	$\sim$ 50 KiB	$\sim$ 250 KiB

Tabla 3.1 Clases de dispositivos restringidos (KiB = 1024 bytes)

3.2. Firmware

El firmware es un tipo de software que proporciona instrucciones de máquina a los componentes de hardware de un dispositivo, lo que le permite funcionar a un nivel básico. Dado que es instalado por el fabricante y normalmente no se puede eliminar, a veces se denomina software integrado. El firmware se encarga de la comunicación entre el sistema operativo y el hardware y le indica a un dispositivo cómo debe funcionar a un nivel muy básico.[19]

Existen consideraciones de la IETF en cuanto a la implementación de firmware. Para esto, se debe utilizar la autenticación y la protección de la integridad de las imágenes de firmware, pero la protección confidencial es opcional. En el caso de las actualizaciones de firmware, se realizan no solo para corregir errores, sino también para agregar nuevas funcionalidades y reconfigurar el dispositivo para que funcione en nuevos entornos o se comporte de manera diferente en un contexto ya implementado.[20]

3.2.1. Arquitectura del firmware

Para visualizar de qué manera operan e interactúan los componentes y la arquitectura se ha tomado el modelo de la IETF, RFC9019 (véase la Figura 3.1).

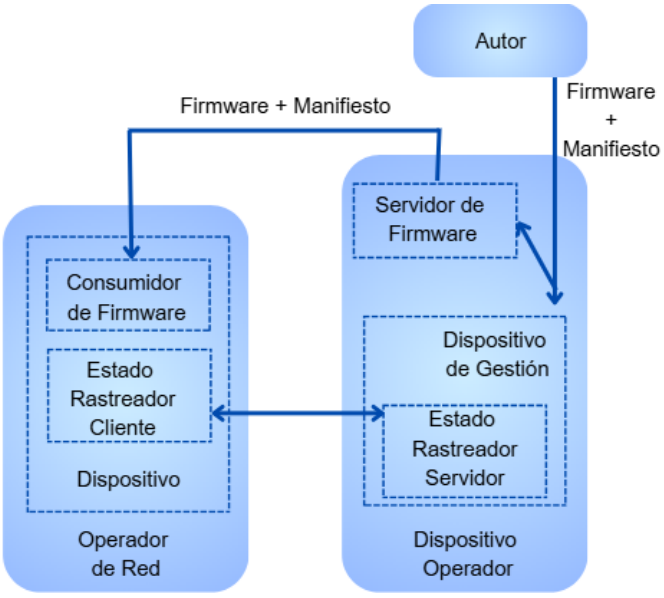


Figura 3.1 Arquitectura de firmware

- **Imagen de firmware:** Es un binario que puede contener el software completo de un dispositivo o un subconjunto del mismo. La imagen de firmware puede constar de varias imágenes si el dispositivo contiene más de un microcontrolador. Una imagen de firmware de aplicación, como indica su nombre, contiene el programa de aplicación, que a menudo incluye todo el código necesario para ejecutarlo (como pilas de protocolos y un sistema operativo (OS) integrado).
- **Manifiesto:** El manifiesto contiene metadatos sobre la imagen del firmware. El manifiesto está protegido contra modificaciones y proporciona información sobre el autor.
- **Autor:** Es la entidad que crea la imagen del firmware.
- **Operador del dispositivo:** Es responsable del funcionamiento diario de una flota de dispositivos IoT.
- **Operador de red:** Es responsable del funcionamiento de una red a la que se conectan los dispositivos IoT.
- **Rastreador de estado:** El rastreador de estado tiene un componente cliente y un componente servidor y realiza tres tareas:

  1. Comunica la disponibilidad de una nueva versión de firmware. Esta información fluirá del servidor al cliente.
  2. Transmite información sobre las características del software y hardware del dispositivo. El flujo de información va del cliente al servidor.
  3. Puede activar de forma remota el proceso de actualización del firmware. El flujo de información va del servidor al cliente.

- **Consumidor de firmware:** Es el destinatario de la imagen de firmware y del manifiesto. Es responsable de analizar y verificar el manifiesto recibido y de almacenar la imagen de firmware obtenida. Interactúa con el servidor de firmware y el cliente de seguimiento de estado (localmente).
- **Servidor de firmware:** Almacena imágenes de firmware y manifiesto. Distribuye a los dispositivos IoT.
- **Gestor de arranque:** Un gestor de arranque es un programa que se ejecuta una vez que se reinicia un microcontrolador. Es el encargado de decidir qué código ejecutar.
- **Dispositivo (IoT):** Se refiere a todo el producto de IoT, que consta de uno o varios microcontroladores, sensores y/o actuadores. Muchos de los dispositivos de IoT que se venden hoy en día contienen varios microcontroladores; por lo tanto, un solo dispositivo puede necesitar obtener más de una imagen y manifiesto de firmware para realizar una actualización correctamente.

3.3. Protocolos de red y comunicación

Un protocolo de comunicación es un conjunto de reglas y convenciones que permiten la transmisión de datos entre dispositivos en una red. Estos protocolos definen aspectos como el formato de los mensajes, el control de flujo, la corrección de errores y la sincronización de la comunicación.[21]

3.4. Modelo OSI

El término OSI (Interconexión de Sistemas Abiertos, por sus siglas en inglés Open System Interconnection) califica normas para el intercambio de información entre sistemas que son “abiertos” entre sí. Este modelo ISO de referencia, proporciona una base común para la coordinación del desarrollo de normas con el fin de interconectar sistemas, permitiendo a la vez que las normas existentes se sitúen en perspectiva dentro del Modelo de Referencia global. El hecho de que un sistema sea abierto no implica ninguna implementación particular, tecnología o medio de interconexión, sino que se refiere al reconocimiento y soporte mutuo de las normas aplicables. [22]

En el contexto del presente trabajo, el modelo OSI puede relacionarse directamente con la pila de protocolos utilizada por el microcontrolador en las comunicaciones seguras basadas en MQTT sobre TLS. La tabla 3.2 muestra la correspondencia entre cada capa del modelo y los protocolos o tecnologías empleados.

N°	Capa (Modelo OSI)	Protocolo / Tecnología utilizada
7	Aplicación	MQTT, HTTP
5-4	Sesión-Transporte	TLS, mTLS (seguridad y cifrado)
4	Transporte	TCP
3	Red	IP (IPv4)
2	Enlace de datos	Wi-Fi (IEEE 802.11 b/g/n)
1	Física	Radiofrecuencia 2.4 GHz

Tabla 3.2 Pila de protocolos utilizada según el modelo OSI

3.5. Capa 7 - Aplicación

3.5.1. HTTP

HTTP (Hypertext Transfer Protocol) es un protocolo utilizado para la transferencia de información en la web. Permite la comunicación entre un navegador web y un servidor, facilitando la carga de páginas web y la transmisión de datos, actúa en la capa 7 del modelo OSI y utiliza el puerto 80 de manera predeterminada.

3.5.2. MQTT

MQTT es un protocolo de transporte de mensajes de publicación/suscripción de cliente-servidor. Es liviano, abierto, simple y está diseñado para ser fácil de implementar. Estas características lo hacen ideal para su uso en muchas situaciones, incluidos entornos restringidos, como la comunicación en contextos de máquina a máquina (M2M) e internet de las cosas (IoT) donde se requiere una pequeña huella de código y/o el ancho de banda de la red es limitado.

El protocolo se ejecuta sobre TCP/IP o sobre otros protocolos de red que proporcionan conexiones ordenadas, bidireccionales y sin pérdidas. Entre sus características se incluyen:

Quality of Service	Descripción
QoS 0	Como máximo una vez. Los mensajes se envían según los mejores esfuerzos del entorno operativo, pudiendo producirse pérdidas. Se utiliza en situaciones donde la pérdida de una lectura no afecta al sistema, como sensores ambientales que generan datos continuamente.
QoS 1	Al menos una vez. Se garantiza que los mensajes llegarán al receptor, pero pueden aparecer duplicados debido a reintentos. Es adecuado cuando la entrega es importante, pero el receptor puede manejar duplicados.
QoS 2	Exactamente una vez. Garantiza que cada mensaje sea recibido una única vez, sin pérdidas ni duplicados. Se emplea en sistemas donde la integridad del dato es crítica, como facturación o transacciones que no toleran errores.

Tabla 3.3 Niveles de Calidad de Servicio (QoS) en MQTT

El documento RFC 9431 describe los intercambios de mensajes como interacciones del protocolo MQTT. Los clientes son clientes MQTT, que se conectan al Broker para publicar y suscribirse a los mensajes de la aplicación (que están etiquetados con sus temas).[23]

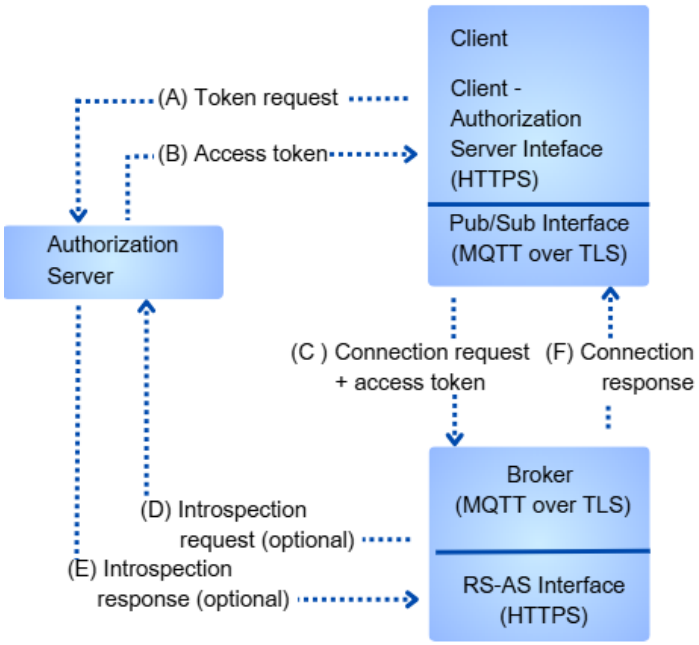


Figura 3.2 Configuración de conexión RFC 9431

La Figura 3.3 da muestra del funcionamiento del protocolo MQTT en un modelo típico. Este modelo está adaptado de la investigación *"Un nuevo modelo DevSecOps para una seguridad robusta en un Internet de las cosas MQTT"*.[24]

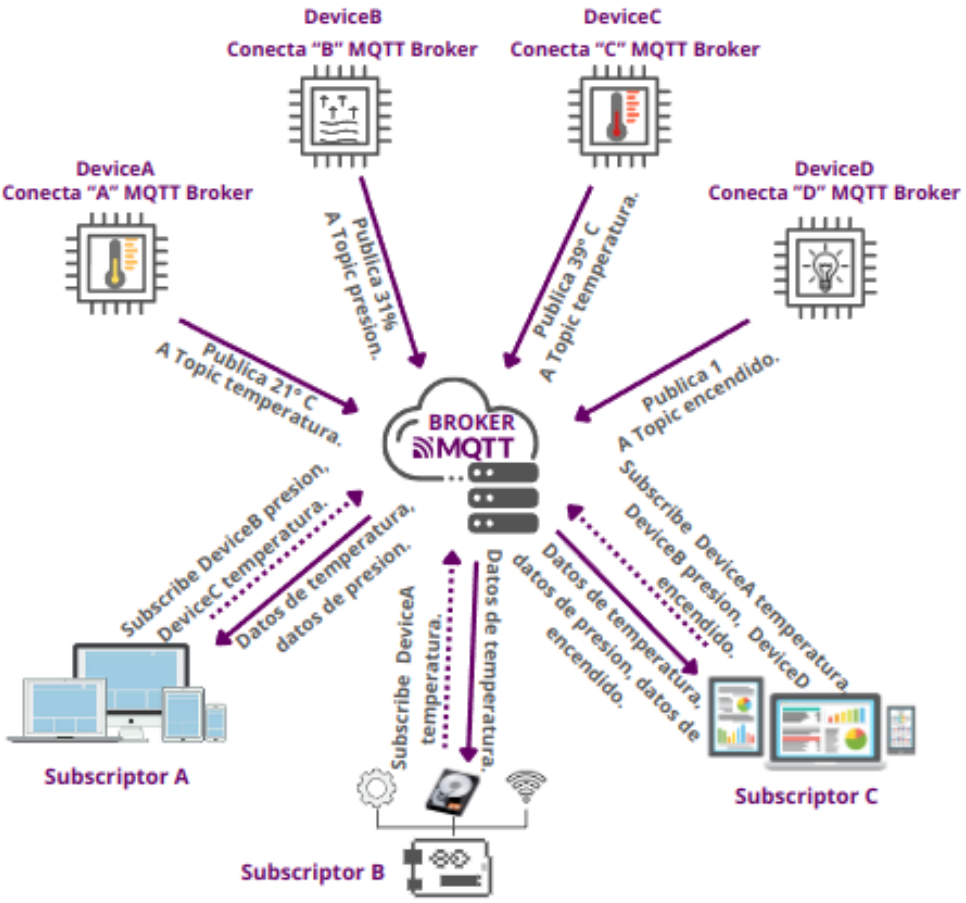


Figura 3.3 Escenario modelo MQTT

Conceptos del protocolo MQTT:

- **Broker:** El broker MQTT actúa como intermediario entre los clientes que publican mensajes de aplicación y los clientes que realizaron suscripciones. El intermediario actúa como servidor de recursos para los clientes.
- **Cliente:** Un dispositivo o programa que utiliza MQTT.
- **Conexión de red:** Una construcción proporcionada por el protocolo de transporte subyacente que utiliza MQTT. Conecta al cliente con el servidor. Proporciona los medios para enviar un flujo de bytes ordenado y sin pérdidas en ambas direcciones. Este documento utiliza TLS como protocolo de transporte.

- **Sesión:** Una interacción con estado entre un cliente y un agente. Algunas sesiones duran solo lo que dura la conexión de red; otras pueden abarcar varias conexiones de red.
- **Mensaje de la aplicación:** Los datos que se transmiten mediante el protocolo MQTT tienen un nivel de calidad de servicio (QoS) y un nombre de tema asociados.
- **Paquete de control MQTT:** El protocolo MQTT funciona intercambiando una serie de paquetes de control MQTT. Cada paquete se compone de un encabezado fijo, un encabezado variable (según el tipo de paquete de control) y una carga útil.
- **Topic Name:** La etiqueta adjunta a un mensaje de aplicación, que coincide con una suscripción.
- **Suscripción:** Una suscripción incluye un filtro de temas y una calidad de servicio máxima. Una suscripción está asociada a una única sesión.

En el protocolo MQTT de publicación-suscripción, después de conectarse al MQTT server (Broker), un cliente puede publicar y suscribirse a múltiples temas. El broker actúa como servidor de recursos (RS), se encarga de distribuir los mensajes publicados por los editores a sus suscriptores. Los mensajes se publican bajo un "Topic Name", y los suscriptores se suscriben a los topic names para recibir los mensajes correspondientes. El broker utiliza el nombre de tema de un mensaje publicado para determinar qué suscriptores deben recibir los mensajes. Si el cliente tiene recursos limitados o no admite HTTPS, un servidor de autorización de cliente independiente puede llevar a cabo la solicitud de token en nombre del cliente y posteriormente, incorporar el token al cliente (véase la Figura 3.2).

3.6. Capa 5-4 - Sesión/Transporte

3.6.1. TLS

El TLS (*Transport Layer Security*) es un protocolo criptográfico diseñado para proporcionar comunicaciones seguras a través de redes no confiables. Su objetivo principal es establecer un canal seguro entre dos entidades, requiriendo únicamente un transporte subyacente confiable y ordenado. Este canal debe garantizar tres propiedades fundamentales:

Propiedad	Descripción
Autenticación	El servidor está siempre autenticado, y el cliente puede estarlo opcionalmente. La autenticación se realiza mediante criptografía asimétrica (RSA, ECDSA, EdDSA) o mediante claves simétricas precompartidas (PSK).
Confidencialidad	Garantiza que los datos transmitidos solo sean accesibles para los extremos autorizados mediante cifrado. Aunque el contenido está protegido, la longitud de los mensajes permanece visible.
Integridad	Evita que los datos sean modificados sin ser detectados, incluso ante un atacante con control completo de la red. Se basa en códigos de autenticación de mensajes y funciones criptográficas robustas.

Tabla 3.4 Propiedades fundamentales del protocolo TLS

TLS consta de dos protocolos principales:

- **Handshake:** Autentica a las partes, negocia parámetros criptográficos y establece claves compartidas resistentes a manipulación.
- **Record:** Protege el tráfico dividiéndolo en registros cifrados y verificados.

El estándar es independiente del protocolo de aplicación, permitiendo que niveles superiores lo utilicen de forma transparente. La versión 1.3 reemplaza a versiones anteriores (incluida TLS 1.2), introduciendo mejoras en la derivación de claves, el manejo de tickets y la validación de certificados.[25]

MQTT con TLS

MQTT se basa en el protocolo de transporte TCP. De forma predeterminada, las conexiones TCP no utilizan comunicación cifrada. Para cifrar toda la comunicación MQTT, muchos agentes MQTT permiten el uso de TLS en lugar de TCP simple. El puerto 8883 está reservado exclusivamente para MQTT sobre TLS.[26]

Para proporcionar confidencialidad de comunicación y autenticación de broker a los clientes MQTT, se utiliza TLS y se recomienda puntualmente TLS 1.3.[25]

3.7. Capa 4 - Transporte

3.7.1. Transmission Control Protocol(TCP)

El Protocolo de Control de Transmisión (TCP) está diseñado para usarse como un protocolo host a host altamente confiable entre hosts en redes de comunicación informática con conmutación de

paquetes y en sistemas interconectados de dichas redes. El datagrama de internet proporciona un medio para direccionar los TCP de origen y destino en diferentes redes. El protocolo de internet también se encarga de cualquier fragmentación o reensamblaje de los segmentos TCP necesarios para lograr el transporte y la entrega a través de múltiples redes y puertas de enlace interconectadas. El protocolo de internet también contiene información sobre la precedencia, la clasificación de seguridad y la compartimentación de los segmentos TCP, de modo que esta información pueda comunicarse de extremo a extremo a través de múltiples redes.[27]

3.8. Capa 3 - Red

3.8.1. Internet Protocol versión 4 (IPv4)

Este protocolo de internet actúa como mecanismo principal de direccionamiento y enrutamiento. Esta capa permite que los paquetes generados por las capas superiores (por ejemplo, TCP/TLS/MQTT) sean encapsulados y enviados a través de diferentes redes hasta alcanzar su destino final.[28]

IPv4 asigna a cada dispositivo una dirección lógica única dentro de la red, lo que permite su identificación y comunicación con otros nodos o servicios en Internet. El módulo Wi-Fi integrado del ESP8266 y del ESP32 implementan el *stack* TCP/IP, permitiendo establecer y mantener conexiones con servidores remotos sin necesidad de un procesador adicional.

Entre las funciones principales de esta capa se encuentran:

- **Direccionamiento:** asignación de una dirección IP al dispositivo, ya sea de forma estática o mediante DHCP.
- **Encapsulación y fragmentación:** división y reensamblado de los paquetes para su transporte eficiente.
- **Enrutamiento:** determinación del camino que siguen los datos a través de la red hasta llegar al destino.

3.9. Capa 1/2 - Física/Enlace

3.9.1. IEEE 802.11 (Wi-Fi)

Este estándar define las especificaciones de la capa física (PHY) y la subcapa de control de acceso al medio (MAC) para la conectividad inalámbrica de baja velocidad de datos con dispositivos fijos, portátiles y móviles sin batería o con un consumo de batería muy limitado. Además, el estándar proporciona modos que permiten una medición precisa de distancias. Las capas físicas (PHY) se definen para dispositivos que operan en diversas regiones geográficas. [29]

Entre las principales características de la interfaz Wi-Fi del caso de estudio de este proyecto se incluyen:

- Soporte para modos de operación *Station*, *Soft-AP* y *Station + Soft-AP*.
- Velocidades de transmisión de hasta 72.2 Mbps en 802.11n.
- Mecanismos de seguridad WEP, WPA/WPA2 y autenticación mediante contraseñas o certificados.
- Utilización de radiofrecuencia en la banda de 2.4 GHz, con 13 canales disponibles (dependiendo de la región).

Esta capa constituye la base sobre la cual se establecen las comunicaciones TCP/IP seguras utilizadas por el protocolo MQTT sobre TLS.

3.10. Edge Computing/Fog Computing

El Edge Computing, o computación en el borde, es un nuevo paradigma de computación en el que los datos del IoT son procesados en la periferia de la red (cloud edge), en la misma fuente que los genera o tan cerca de ella como sea posible. Podemos comparar el Edge Computing con los gobiernos regionales establecidos en un país para evitar que toda la gestión pase por el gobierno central o, para cerrar el símil, el Cloud Computing.[30]

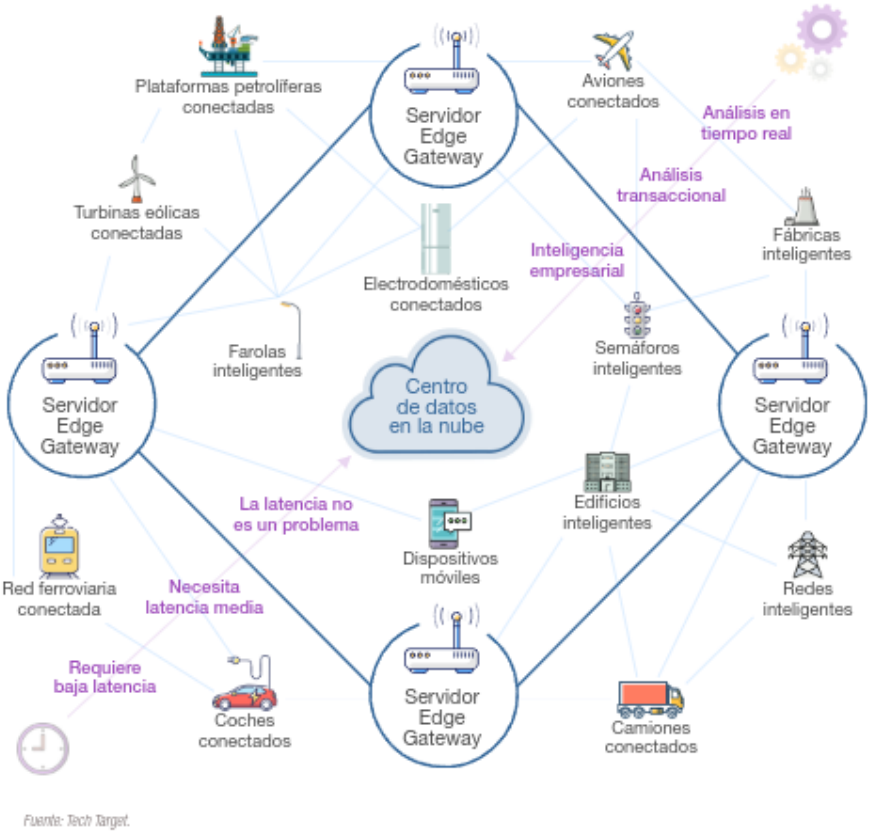


Figura 3.4 Escenario Edge Computing

El Edge Computing, también conocido como “Fog Computing” en algunos entornos, es un nuevo paradigma en el que se colocan recursos de computación y almacenamiento sustanciales en el borde de internet, cerca de dispositivos móviles, sensores, actuadores o máquinas. Procesa tanto datos descendentes (que se originan en servicios en la nube) como datos ascendentes (que se originan en dispositivos finales o elementos de red). El término "Fog Computing", generalmente representa la noción de computación de borde de múltiples niveles, es decir, varias capas de infraestructura informática entre dispositivos finales y servicios en la nube.

Un dispositivo de borde es cualquier recurso informático o de red que reside entre las fuentes de datos de los dispositivos finales y los centros de datos basados en la nube. En Edge Computing, los dispositivos finales consumen y producen datos.

Varias organizaciones de desarrollo de normas (Several Standards Developing Organizations) y foros de la industria han proporcionado definiciones de Edge Computing y de Fog Computing:

A partir de estas definiciones, podemos resumir una filosofía general de Edge Computing como distribución de las funciones requeridas cerca de los usuarios y datos, mientras que la diferencia con los sistemas locales clásicos es el uso de funciones de gestión y orquestación adoptadas desde el Cloud Computing. [31]

Organización	Definición
ISO	Define la computación de borde como una forma de computación distribuida en la que el procesamiento y almacenamiento de datos significativos se realiza en nodos ubicados en el borde de la red.
ETSI	Define la computación de borde de acceso múltiple como un sistema que proporciona un entorno de servicios de TI y computación en la nube, con capacidades ubicadas en el borde de una red de acceso que puede contener distintos tipos de tecnologías, en estrecha proximidad a los usuarios.
IIC (Industrial Internet Consortium)	Define el Fog Computing como un sistema horizontal de arquitectura por niveles que distribuye la computación, el almacenamiento y el control de funciones de red más cerca de los usuarios a lo largo de un continuo hacia la nube.

Tabla 3.5 Edge Computing y Fog Computing según organizaciones relevantes

3.10.1. Gateway

El Gateway actúa como intermediario entre los dispositivos de IoT y los sistemas centralizados, lo que permite el procesamiento local y reduce la dependencia de los servicios en la nube para tareas urgentes o que requieren muchos datos. En dispositivos con una capacidad poca capacidad de proceso, el Gateway, agrega una capa adicional de seguridad para protegerlos de ataques.[31]

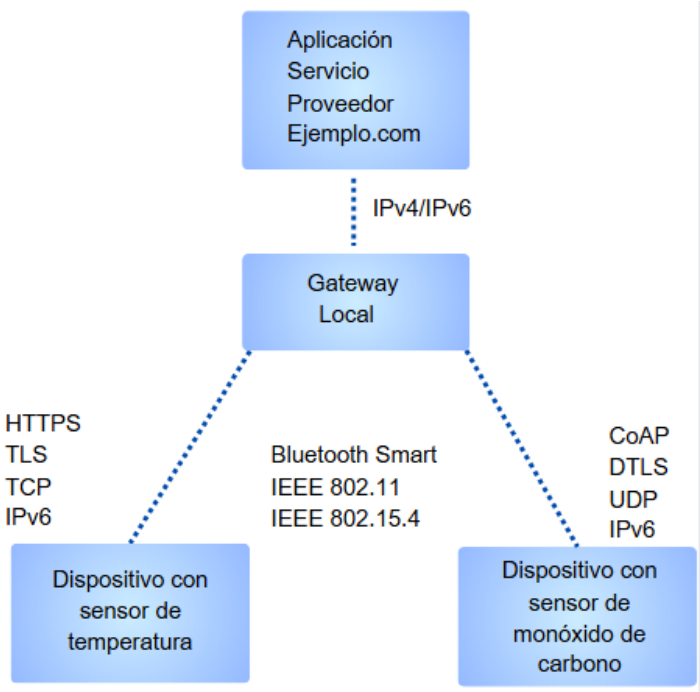


Figura 3.5 Comunicación de dispositivo a Gateway

Un modelo para aplicar Edge Computing en IoT es con la utilización de un elemento llamado Gateway o pasarela IoT.[17]

3.11. Vulnerabilidades críticas en IoT

3.11.1. Supply-chain - Supply-chain attack

La cadena de suministro es la secuencia de procesos involucrados en la producción de componentes, software y piezas que juntos forman un sistema, y su integración abarcando muchas organizaciones, incluidos proveedores, vendedores y múltiples niveles de subcontratación. Es un sistema complejo, geográficamente diverso y distribuido globalmente de redes interconectadas, que cubre una amplia superficie.[32]

Las cadenas de suministro de IoT son sistemas de IoT que pueden verse comprometidos por ciberataques en su cadena de suministro. Los componentes comprometidos en la cadena de suministro y luego implementados en entornos operativos abren el camino a una variedad de ataques. Los ataques a la cadena de suministro son extremadamente rentables desde el punto de vista de los atacantes. Los usuarios posteriores suelen confiar implícitamente en los activos de TI que provienen de entornos de desarrollo, fabricación y distribución, a pesar de los controles de seguridad débiles o desconocidos en esos entornos. Además, un compromiso exitoso de un solo proveedor de TI bien elegido puede extenderse a toda la base de clientes del proveedor, a medida que se implementan productos y actualizaciones de software. No es coincidencia que muchos de los ciberataques más notorios hayan sido ataques a la cadena de suministro.

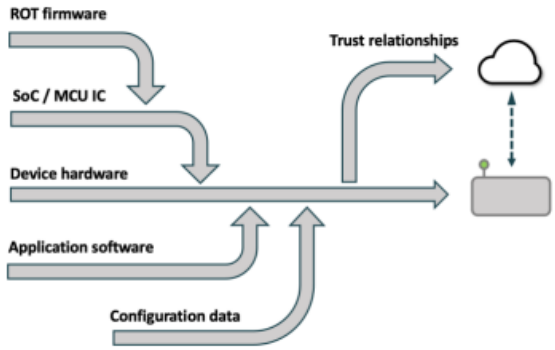


Figura 3.6 Supply Chain típica IoT

Esta red de suministro consta de seis tipos básicos de operaciones, como se muestra en la siguiente tabla:

Tipo de operación	Descripción
Diseño	Reúne los requisitos y especifica un sistema capaz de satisfacerlos.
Desarrollo	Crea y pone en funcionamiento activos de producción funcionales.
Ensamblaje de hardware	Integra progresivamente componentes y subconjuntos en dispositivos completos.
Programación	Instala activos lógicos en dispositivos físicos. Forma parte del proceso de aprovisionamiento.
Personalización	Genera una identidad única para cada dispositivo. Forma parte del proceso de aprovisionamiento.
Integración	Coloca los dispositivos en relaciones de confianza con otros sistemas. Forma parte del proceso de aprovisionamiento.

Tabla 3.6 Tipos de operaciones en la red de suministro de dispositivos

Un ciberataque podría lanzarse en cualquier etapa de la red de suministro de un dispositivo inteligente. El diseño y el desarrollo son actividades de tipo proyecto que tienen mucho en común con los proyectos de diseño y desarrollo de software y hardware en todas partes. Durante estas fases, la seguridad es idéntica a la calidad, porque puede ser imposible distinguir entre un error y un sabotaje. Afortunadamente, existe una amplia literatura sobre ciclos de vida de desarrollo de software (SDLC) seguros y buenas prácticas de TI para proteger los entornos de desarrollo.[33]

3.11.2. SBOM como mecanismo de mitigación para *supply-chain attack*

Los ataques a la cadena de suministro suelen explotar la falta de visibilidad sobre los componentes que integran un sistema o dispositivo IoT. La complejidad creciente del software moderno —basado en múltiples librerías, dependencias transitivas, firmware de terceros y módulos obtenidos desde repositorios externos— dificulta verificar su origen, integridad y estado de seguridad. Esta opacidad facilita que código vulnerable o malicioso se incorpore inadvertidamente durante las fases de diseño, desarrollo, ensamblaje o programación del dispositivo.[33]

El Software Bill of Materials (SBOM) se presenta como un mecanismo para mejorar la transparencia, trazabilidad y verificabilidad del software embebido. Un SBOM es una “lista formal y estructurada de los componentes que conforman un software”, incluyendo dependencias, versiones, licencias y metadatos de procedencia.[**NIST_SBOM_2021**] El Departamento de Comercio de los Estados Unidos, a través de la NTIA, remarca que un SBOM permite a los equipos de desarrollo, operaciones y seguridad identificar rápidamente componentes vulnerables, verificar su integridad y evaluar la exposición ante fallas críticas detectadas en la cadena de suministro.[**NTIA2021**]

3.11.3. Bricking

El término 'bricking' en ciberseguridad y tecnología se refiere a un evento donde un dispositivo electrónico, como un teléfono inteligente, computadora, tableta, o cualquier dispositivo IoT, se vuelve completamente inoperable o significativamente dañado, haciéndolo tan útil como un ladrillo. Esta condición, a menudo irreversible, hace que el dispositivo no pueda realizar sus funciones primarias o encenderse, requiriendo reparaciones potencialmente costosas o reemplazos completos. El 'bricking' puede ser el resultado de actividades maliciosas deliberadas, errores graves de software o firmware, o daños accidentales durante actualizaciones o modificaciones de sistema fallidas.

El 'bricking' puede ocurrir de diversas maneras, cada una con sus mecanismos e implicaciones:

Causa	Descripción
Software malicioso	Malware o ataques dirigidos pueden reescribir el firmware del dispositivo o configuraciones críticas del sistema, dejando el dispositivo inoperativo.
Actualizaciones fallidas	Actualizaciones defectuosas o interrumpidas del firmware o sistema operativo del dispositivo pueden corromper archivos esenciales, llevando al 'bricking'.
Modificaciones no oficiales	Realizar 'jailbreaking' o 'rooting' para eludir restricciones del fabricante puede exponer los dispositivos a riesgos de 'bricking', debido a modificaciones incompatibles o inestables.
Fallos de hardware	Aunque menos comunes, daños físicos o defectos de fabricación que afectan componentes cruciales también pueden resultar en un estado de 'bricking'.
Ataques remotos	Los dispositivos conectados a internet pueden ser vulnerables a ataques remotos que explotan agujeros de seguridad para inducir 'bricking' intencionalmente, comúnmente llamados ataques de Ejecución Remota de Código (RCE).

Tabla 3.7 Causas comunes de *bricking* en dispositivos

3.12. DevOps

DevOps es una evolución de la metodología ágil de desarrollo de software, que surgió como una alternativa a la metodología en cascada. En el enfoque en cascada, los equipos de desarrollo de software dedicaron meses a desarrollar grandes cuerpos de código, que luego se sometieron a meses de pruebas antes del lanzamiento. Por el contrario, el desarrollo ágil adopta un enfoque iterativo para el ciclo de vida de la entrega de software. DevOps agrega nuevos procesos y herramientas a la metodología ágil, en particular la automatización de gran parte del pipeline de CI/CD.[34]

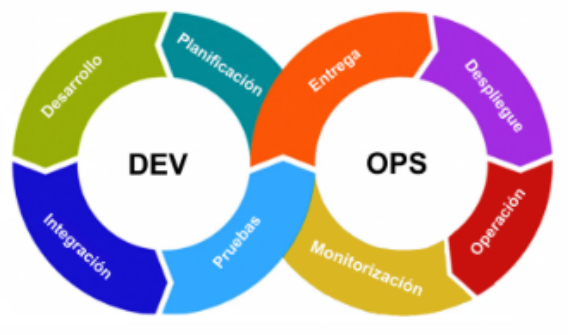


Figura 3.7 Proceso continuo de implementación DevOps

Al adoptar una cultura de DevOps junto con prácticas y herramientas de DevOps, los equipos adquieren la capacidad de responder mejor a las necesidades de los clientes, aumentar la confianza en las aplicaciones que crean y alcanzar los objetivos empresariales en menos tiempo.([35])

Procesos:

- **Planificación:** En la fase de planeamiento, los equipos de DevOps conciben, definen y describen las características y la funcionalidad de las aplicaciones y los sistemas que planean crear. Los equipos utilizan las siguientes prácticas de DevOps para establecer planes con agilidad y visibilidad:
  - Creación de trabajo pendiente.
  - Seguimiento de errores.
  - Administración del desarrollo de software Agile con Scrum.
  - Uso de paneles Kanban.
  - Visualización del progreso con paneles.
- **Desarrollo:** La fase de desarrollo comprende todos los aspectos del desarrollo de código de software. En esta fase, los equipos de DevOps llevan a cabo las tareas a continuación:
  - Selección de un entorno de desarrollo.
  - Escritura, prueba, revisión e integración del código.
  - Compilación del código en artefactos para implementarlo en diferentes entornos.
  - Uso de control de versiones, normalmente Git, para colaborar en el código y trabajar en paralelo.

  Para innovar con rapidez sin sacrificar la calidad, la estabilidad ni la productividad, los equipos de DevOps:
  - Utilizan herramientas altamente productivas.
  - Automatizan los pasos cotidianos y manuales.

  - Realizan iteraciones en incrementos pequeños mediante pruebas automatizadas e integración continua (CI).

- **Integración continua y entrega continua (CI/CD):** La integración continua (CI) es la práctica que utilizan los equipos de desarrollo para la automatización, combinación y prueba de código. La integración continua ayuda a detectar errores en etapas tempranas del ciclo de desarrollo, lo que hace que sean menos costosos de corregir. Los sistemas de CI generan artefactos y los alimentan para liberar procesos a fin de impulsar implementaciones frecuentes. La entrega continua (CD) es un proceso por el que el código se compila, prueba e implementa en uno o varios entornos de prueba y producción. La implementación y las pruebas en varios entornos aumentan la calidad. Los procesos de versión automatizados consumen estos artefactos para publicar versiones nuevas y correcciones en los sistemas existentes.

- **Despliegue:** El despliegue es el proceso de implementación de aplicaciones en entornos de producción. En esta fase, los equipos de DevOps:

  - Establecen un proceso de administración de versiones con etapas de aprobación manual claras.
  - Configuran puertas automatizadas para mover aplicaciones entre fases hasta la versión final a los clientes.
  - Automatizan los procesos de entrega para que sean escalables, repetibles, controlados y probados correctamente.

- **Operaciones:** La fase de operaciones implica el mantenimiento, la supervisión y la solución de problemas de aplicaciones en entornos de producción, que incluyen nubes híbridas o públicas. Los objetivos de los equipos de DevOps son la confiabilidad del sistema, la alta disponibilidad, la seguridad sólida y el tiempo de inactividad cero. La entrega automatizada y las prácticas de implementación segura ayudan a los equipos a identificar y paliar los problemas rápidamente cuando se producen. El mantenimiento de la vigilancia requiere una telemetría muy completa, alertas que permitan tomar medidas y visibilidad total de las aplicaciones y de los sistemas subyacentes.

- **Supervisión continua:** Supone tener una visibilidad completa y en tiempo real del rendimiento y el estado de la totalidad de las aplicaciones. La visibilidad abarca desde la infraestructura de base que ejecuta la aplicación hasta los componentes de software de nivel superior.[36]

3.12.1. Diferencias entre Integración Continua(CI), Entrega Continua(CD) y Despliegue Continuo(CD)

La integración continua forma parte tanto de la entrega continua como del despliegue continuo. Y el despliegue continuo es como la entrega continua, excepto que las versiones se realizan automáticamente.[37]

Integración continua (CI)	Entrega continua (CD)	Despliegue continuo (CD)
Los desarrolladores integran sus cambios en la rama principal con alta frecuencia. Las integraciones se validan con compilaciones y pruebas automatizadas, evitando problemas acumulados y asegurando que el código no se rompa con nuevas confirmaciones.	Extiende la integración continua. Después de compilar y probar automáticamente, los cambios se despliegan en un entorno de prueba o preproducción. El despliegue a producción es manual, pero rápido y confiable gracias al pipeline automatizado. Permite liberar en cualquier momento, idealmente con despliegues frecuentes y pequeños.	Va un paso más allá de la entrega continua. Si el cambio supera todas las etapas del pipeline, se despliega automáticamente a producción sin intervención humana. Solo una prueba fallida detiene el proceso. Acelera el feedback de los usuarios y elimina la presión del "día de lanzamiento".

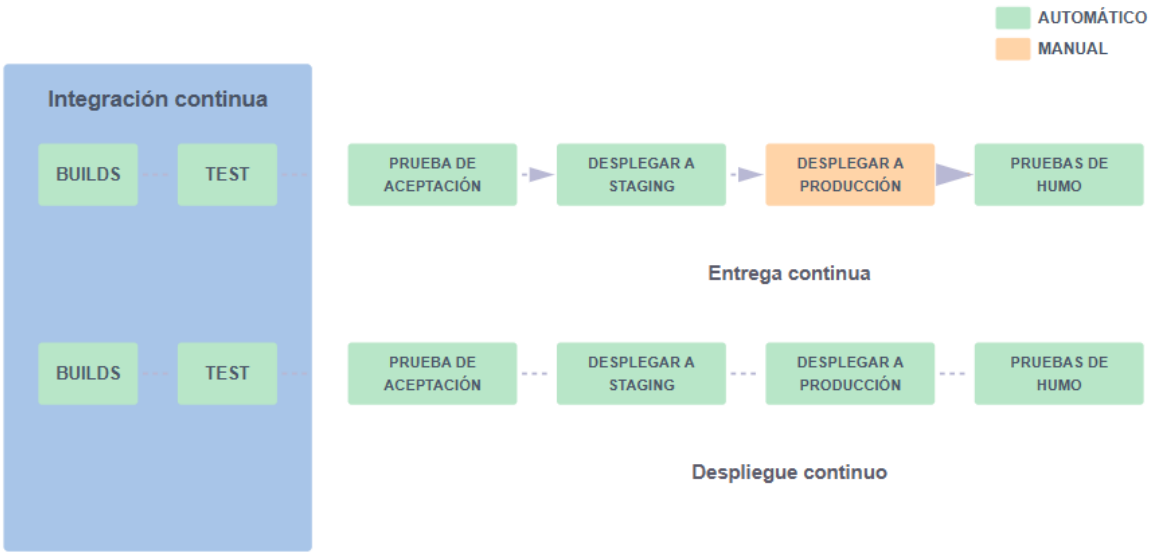


Figura 3.8 Continuos Integration, Continuos Delivery, Continuos Deployment

En entornos IoT, la publicación de una nueva versión de firmware puede activar automáticamente el proceso de actualización OTA (Over-The-Air), lo que permite desplegar la actualización en los dispositivos sin intervención manual.

3.13. DevSecOps

DevSecOps significa desarrollo, seguridad y operaciones. Se trata de un enfoque que aborda la cultura, la automatización y el diseño de plataformas, e integra la seguridad como una responsabilidad compartida durante todo el ciclo de vida de la TI.[3]



Figura 3.9 Proceso continuo de implementación DevSecOps

Concepto	Abreviatura	Descripción
Desarrollo	**Dev**	Es el proceso de planificación, codificación y prueba de la aplicación.
Seguridad	**Sec**	Significa introducir elementos de seguridad en una etapa temprana del desarrollo.
Operaciones	**Ops**	Implementa, supervisa y corrige cualquier problema con el software o los dispositivos.

Tabla 3.8 Descripción de los conceptos DevSecOps

3.13.1. Cultura DevSecOps

Comunicación: Las empresas implementan DevSecOps por un cambio cultural que comienza por los altos directivos. Los líderes sénior explican la importancia y los beneficios de adoptar prácticas de seguridad al equipo de DevOps. Los desarrolladores de software y los equipos de operaciones necesitan las herramientas, los sistemas y la motivación adecuados para adoptar las prácticas de DevSecOps.

Personas: DevSecOps conduce a una transformación cultural que involucra a los equipos de software. Los desarrolladores de software ya no se aferran a los roles convencionales de crear, probar y desplegar el código. Al utilizar DevSecOps, los desarrolladores de software y los equipos de operaciones trabajan en estrecha colaboración con los expertos en seguridad para mejorar la seguridad en todo el proceso de desarrollo.

Tecnologías: Los equipos de software utilizan la tecnología para realizar pruebas de seguridad automatizadas durante el desarrollo. Los equipos de DevOps la utilizan para verificar la aplicación en busca de defectos de seguridad sin comprometer el plazo de entrega.

Procesos/Procedimientos: Al adoptar DevSecOps, los equipos de software realizan pruebas y evaluaciones de seguridad a lo largo de todas las fases de desarrollo. Los desarrolladores de software buscan defectos de seguridad al escribir el código. Posteriormente, un equipo de seguridad prueba la aplicación previa al lanzamiento en busca de vulnerabilidades de seguridad.

3.13.2. Componentes de DevSecOps

- **Análisis de código:** El análisis del código se refiere al proceso de investigar el código fuente de una aplicación en busca de vulnerabilidades y garantizar que se ajusta a las prácticas recomendadas de seguridad.
- **Administración de cambios:** Los equipos de software utilizan herramientas de administración de cambios para realizar un seguimiento de los cambios relacionados con el software o los requisitos, así como para administrarlos e informar sobre estos, evitando vulnerabilidades de seguridad involuntarias ocasionadas por los cambios en el software.
- **Administración del cumplimiento:** Los equipos de software se encargan de garantizar que el software cumpla con los requisitos regulatorios.
- **Modelado de amenazas:** Los equipos de DevSecOps investigan los problemas de seguridad que puedan surgir antes y después de desplegar la aplicación. Corrigen cualquier problema conocido y lanzan una versión actualizada de la aplicación.
- **Formación en seguridad:** La formación en materia de seguridad implica capacitar a los desarrolladores de software y a los equipos de operaciones en cuanto a las directrices de seguridad más recientes. De este modo, los equipos de desarrollo y de operaciones pueden tomar decisiones de seguridad independientes a la hora de crear y desplegar la aplicación.[38]

3.13.3. Herramientas de análisis DevSecOps

Concepto	Descripción
SAST	Las herramientas de pruebas de seguridad de aplicaciones estáticas (Static Application Security Testing) analizan y encuentran vulnerabilidades en el código fuente propio.
IAST	Los equipos de DevSecOps utilizan herramientas de pruebas de seguridad de aplicaciones interactivas (Interactive Application Security Testing) para evaluar las posibles vulnerabilidades de una aplicación en el entorno de producción. IAST comprende monitores de seguridad especiales que se ejecutan desde el interior de la aplicación.
DAST	Las herramientas de pruebas de seguridad de aplicaciones dinámicas (Dynamic Application Security Testing) imitan a los hackers al probar la seguridad de la aplicación desde fuera de la red.

Tabla 3.9 Descripción de los conceptos SAST, IAST y DAST

3.13.4. Diferencias entre análisis

La seguridad de aplicaciones consiste en proteger las aplicaciones de software contra amenazas y vulnerabilidades de seguridad. [39]



Figura 3.10 Diferencias entre análisis IAST, DAST Y SAST

3.13.5. Comparativa DevOps-DevSecOps

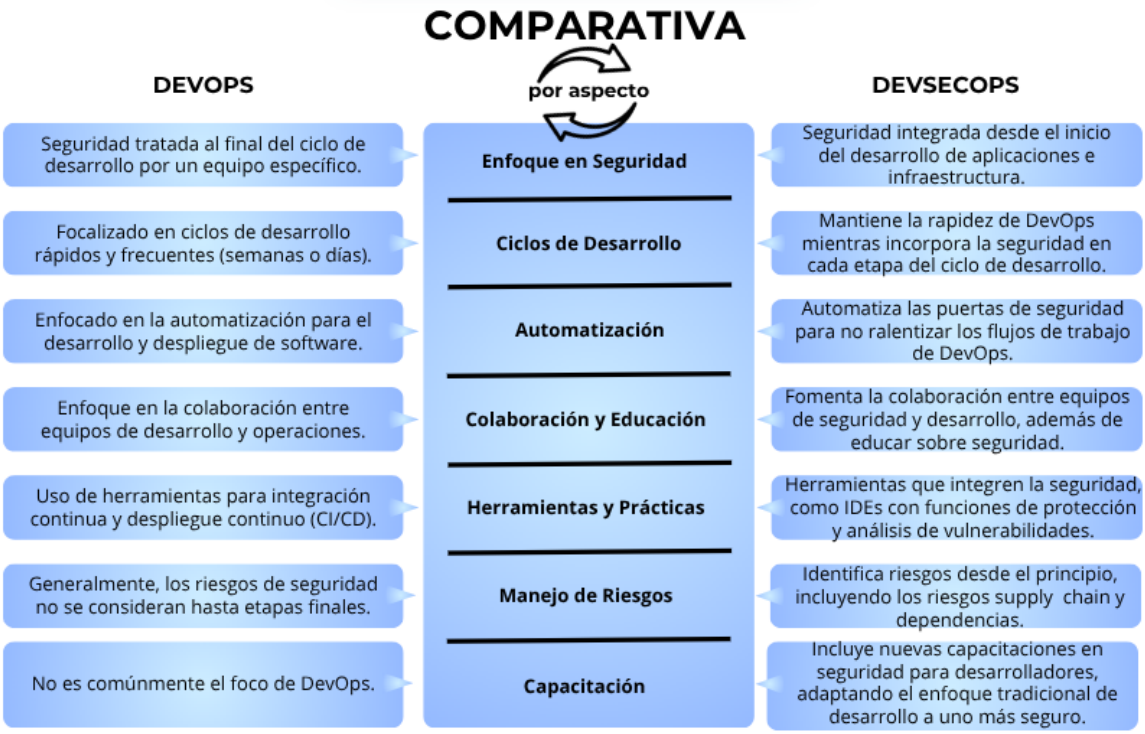


Figura 3.11 Comparación entre DevOps y DevSecOps

3.13.6. Comparativa beneficios DevSecOps



Figura 3.12 Beneficios DevSecOps

Capítulo 4

Desarrollo

4.1. Alcance del proyecto

El presente trabajo se centra en la aplicación de la metodología DevSecOps al ciclo de vida de dispositivos IoT de bajo consumo y recursos limitados. Como caso de estudio se seleccionaron dos dispositivos, ESP8266 y ESP32. El protocolo elegido para la comunicación es MQTT, debido a su ligereza y eficiencia en entornos IoT. Este proyecto permite demostrar cómo se integran seguridad, confiabilidad y escalabilidad en una arquitectura que combina dispositivos físicos, firmware y servicios desplegados en contenedores mediante DevSecOps.

4.2. Alcance específico del proyecto

El alcance específico del proyecto incluye:

- **Análisis de riesgos y vulnerabilidades:** Evaluación de los dispositivos ESP8266 y ESP32, considerando configuraciones por defecto, protocolos de comunicación y gestión de credenciales.
- **Conectividad:** Evaluación de redes considerando configuraciones por defecto, análisis de riesgos, y propuestas de mitigación de riesgos.
- **Diseño de la arquitectura de la solución:** Aplicación de buenas prácticas de seguridad y diseño de infraestructura, integrando pipelines DevSecOps, los cuales permiten la entrega continua y segura de software y firmware.
- **Infraestructura de nube y servicios:** Implementación de un broker MQTT seguro, almacenamiento de datos, dashboards de análisis, monitoreo y alertas por correo electrónico ante desviaciones. Se asegura la trazabilidad y la resiliencia mediante pipelines de CI/CD que permiten pruebas automatizadas antes de desplegar actualizaciones.
- **Firmware y código:** Uso de desarrollo de firmware con implementación de actualizaciones OTA (Over-The-Air)[16], se aplicarán técnicas en él pipeline DevSecOps, con validación de

integridad, cifrado de credenciales y comunicaciones, y mitigación de riesgos como *bricking* o *supply-chain attack*.

- **Desarrollo de casos de prueba prácticos:** Evaluación de escenarios críticos de seguridad, incluyendo *bricking* del dispositivo, *supply-chain attack*, interceptación de comunicaciones (MITM), fuerza bruta y ataques DDoS.
- **Implementación de mecanismos de detección y monitoreo:** Supervisión continua de eventos, alertas ante desviaciones y seguridad.
- **Establecimiento de controles técnicos y procesales:** Consolidar el pipeline de seguridad automatizado con GitHub Actions, integrando SonarCloud para análisis estático y cobertura de código, y DefectDojo como plataforma centralizada de gestión de vulnerabilidades. El flujo incluye auditorías de dependencias, compilación controlada y ejecución de escaneos en fases de *pull request* y *release*. Este proceso garantiza la detección proactiva de defectos, la trazabilidad de riesgos y la generación automática de reportes de seguridad bajo criterios de aceptación definidos.

4.3. Limitaciones del proyecto

Las principales limitaciones del proyecto son:

- El proyecto se centra únicamente en los **dispositivos de referencia (ESP8266 y ESP32)**. No aborda la totalidad de dispositivos IoT existentes.
- Dependencia de un entorno de pruebas controlado. Los ataques, casos de prueba y las validaciones de seguridad se realizan en un ambiente controlado para evitar el compromiso de terceros. Esto implica que, factores reales como interferencias electromagnéticas, congestión de red, latencia, ruido en canales inalámbricos o condiciones adversas de energía no se reproducen completamente, lo cual limita la extrapolación a entornos industriales o redes geográficamente distribuidas.
- La configuración específica del mecanismo OTA (Over-The-Air), así como el diseño detallado de las particiones de memoria necesarias para soportar dicho proceso, queda fuera del alcance de este proyecto, ya que forma parte del trabajo complementario descrito en el documento "Implementación de prácticas DevOps para la integración y el despliegue continuos de software en sistemas IoT" [16]. El uso de este proyecto genera varias limitaciones adicionales:
    - **Entornos de conectividad inestables:** La eficacia de las actualizaciones OTA depende de la estabilidad de la red. En escenarios reales con Wi-Fi débil, dispositivos energéticamente restringidos o nodos con conectividad esporádica, las actualizaciones pueden fallar o interrumpirse, generando riesgos de firmware incompleto.

  - **Riesgo de fallos críticos durante la actualización:** Sin un mecanismo operativo de particiones A/B, verificación criptográfica y *rollback* automático, no es posible validar experimentalmente el riesgo de *bricking*, especialmente en microcontroladores con memoria flash limitada.

- Restricciones de memoria, procesamiento y desgaste del hardware, tanto el ESP8266 como el ESP32 presentan limitaciones inherentes a dispositivos IoT de bajo consumo:

  - Memoria RAM reducida, que limita la ejecución de telemetría avanzada o procesos concurrentes del pipeline.
  - Memoria flash cuya vida útil es limitada, lo que restringe la frecuencia de despliegues y pruebas que impliquen escrituras repetitivas.
  - No se mide el impacto energético real de procesos continuos, quedando fuera del alcance evaluar su consumo en dispositivos alimentados por batería.

- Aunque el proyecto contempla amenazas asociadas al acceso físico mediante UART/JTAG, no es posible validar experimentalmente dichos controles debido a que:

  - El prototipo no cuenta con un gabinete definitivo.
  - No se dispone de resina epóxica, blindaje ni carcasa antivandálica.
  - No se realizaron pruebas de extracción de firmware ni protección frente a *glitching* o ataques de hardware.

  En consecuencia, la efectividad de los mecanismos de protección física no puede ser demostrada en esta etapa.

- Debido a restricciones presupuestarias y a la disponibilidad limitada de hardware, el proyecto no contempla la ejecución de pruebas concurrentes sobre múltiples dispositivos ESP8266/ESP32. La ausencia de redundancia impide evaluar escenarios de actualización coordinada, balanceo de carga, propagación de fallos, o comportamientos divergentes entre unidades nominalmente idénticas. La imposibilidad de contar con múltiples dispositivos en paralelo restringe el análisis de variabilidad en el rendimiento, estabilidad y tolerancia a fallos del pipeline DevSecOps en condiciones de operación distribuida.

- No se incluyen análisis de costos ni escalabilidad masiva de la infraestructura, dado que el proyecto busca principalmente un enfoque académico y de demostración.

- Aunque el proyecto cuenta con la capacidad técnica para registrar, clasificar y gestionar hallazgos de seguridad mediante la integración con DefectDojo —incluyendo la categorización por criticidad, asociación de evidencia con el pipeline o commit correspondiente, uso de plantillas

estandarizadas para documentar causa raíz, impacto y recomendaciones, así como el seguimiento de métricas como MTTR— estas funcionalidades no se utilizan de manera completa en esta etapa. La razón es que el manejo operativo de cargas de datos, la configuración avanzada de flujos de trabajo y la alimentación continua del sistema requieren un nivel de tiempo y dedicación que excede el alcance central de esta solución.

- La arquitectura de red propuesta corresponde a una configuración mínima en cascada, diseñada para ser replicable con equipamiento doméstico y de bajo costo económico. Si bien esta aproximación permite demostrar los principios fundamentales de segmentación y aislamiento, no constituye un diseño exhaustivo. Quedan fuera del alcance configuraciones más avanzadas como la implementación de VLANs, segmentación dinámica, listas de control de acceso complejas o *firewalls* de nueva generación, las cuales requieren infraestructura más sofisticada que excede los objetivos y recursos del proyecto.

Capítulo 5

Materiales y métodos

5.1. Materiales

5.1.1. Hardware

El entorno físico del proyecto está conformado por:

- **PC-IoT:** Corresponde al modelo **Gigabyte Technology Ltd. Z270-Gaming K3**, equipada con procesador **Intel Core i7-7700**, **16 GB de memoria RAM**, gráficos integrados **Intel HD 630 (KBL GT2)** y una unidad de almacenamiento **de 500 GB**. La versión de firmware del sistema base es **F4**, lo que asegura la compatibilidad con módulos de virtualización y gestión de dispositivos USB requeridos para la programación de los microcontroladores. Sistema operativo es **Ubuntu 24.10** ***(Oracular)*** (64-bit). Este equipo constituye la base del entorno de desarrollo, compilación, orquestación y monitoreo de la infraestructura DevSecOps.
- **ESP8266:** Microcontrolador Wi-Fi de bajo costo, empleado como nodo de sensado y transmisión de datos.
- **ESP32:** Microcontrolador de versión más avanzada con doble núcleo, mayor capacidad de memoria, soporte para Wi-Fi/Bluetooth, utilizada para pruebas comparativas y de firmware.

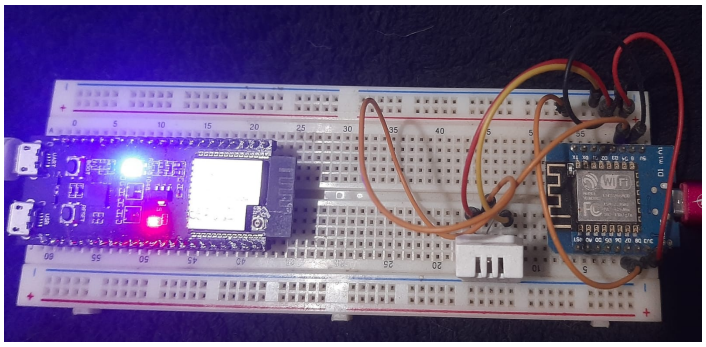


Figura 5.1 Dispositivos de IoT ESP32 y ESP8266

- **Fuente de alimentación USB 5V** y **cables Dupont** para conexión de los módulos.
- **Raspberry Pi 3 Modelo B+:** Microcomputadora de bajo costo utilizada como gateway de borde. Cuenta con un procesador de 4 núcleos y 1 GB de RAM. En este dispositivo se desplegó Docker, con Portainer y Pi-Hole.
- **Router principal (MitraStar GPT-2541GNAC):** Equipo provisto por el Internet Service Provider, con conectividad Wi-Fi de doble banda (2.4 GHz y 5 GHz) y cuatro puertos Ethernet Gigabit. En este proyecto actúa como puerta de enlace hacia Internet y punto de conexión para el router secundario, estableciendo la topología en cascada utilizada para el entorno de pruebas.
- **Router secundario (TP-Link DD-WRT):** Dispositivo empleado para generar una subred aislada destinada a las pruebas IoT y servicios de backend. Opera con dirección IP `192.168.8.1` y actúa como servidor DHCP de la red `192.168.8.0/24`, brindando conectividad controlada y capacidad de monitoreo sin afectar la red principal.

5.1.2. Software

El entorno de software integra herramientas de desarrollo, orquestación, seguridad y monitoreo que permiten aplicar la metodología **DevSecOps** a todo el ciclo de vida del firmware y de la infraestructura IoT:

- **Visual Studio Code:** Entorno de desarrollo integrado (IDE) utilizado para la edición, depuración y control de versiones del código fuente.
- **PlatformIO:** Entorno de desarrollo y compilación de firmware para los microcontroladores ESP8266 y ESP32.
- **GitHub Actions:** Plataforma de CI/CD donde se definen los pipelines de construcción, auditoría, análisis estático y despliegue seguro.
- **Helm:** Gestor de paquetes para Kubernetes empleado en la instalación de componentes complejos como DefectDojo, Redis, PostgreSQL y ThingsBoard.
- **Kubernetes:** Plataforma de orquestación de contenedores utilizada para desplegar y gestionar la infraestructura DevSecOps.
- **Operator Lifecycle Manager (OLM):** Framework que gestiona la instalación y actualización de operadores en Kubernetes, como Cert-Manager.
- **Cert-Manager:** Encargado de la gestión automática de certificados TLS para asegurar las comunicaciones internas del clúster.
- **DefectDojo:** Sistema de gestión centralizada de vulnerabilidades, desplegado mediante Helm en el namespace `devsecops`.

- **ThingsBoard:** Plataforma de gestión IoT para la visualización y análisis de datos provenientes de los dispositivos.
- **SonarCloud:** Herramienta SaaS utilizada para el análisis estático del código, medición de cobertura y detección de vulnerabilidades.
- **pip-audit:** Utilidad para auditoría de dependencias Python en los pipelines CI/CD.
- **Nginx Ingress Controller:** Componente de Kubernetes que gestiona el enrutamiento de tráfico externo hacia los servicios internos de la infraestructura.

5.1.3. Configuración del entorno de pruebas

El entorno de pruebas se diseñó con el objetivo de simular un ecosistema IoT seguro y automatizado, aplicando DevSecOps. La infraestructura principal se implementó sobre **PC-IoT**, garantizando compatibilidad con los módulos de virtualización y la gestión de dispositivos USB necesarios. Sobre esta estación se instalaron las herramientas principales de desarrollo y automatización: **Visual Studio Code** (con extensiones de Git y PlatformIO), **Docker Engine** y **Minikube** como entorno local de gestión de contenedores. La orquestación de componentes se automatizó mediante un **Makefile**, que inicializa los elementos del clúster DevSecOps: Nginx Ingress Controller, Operator Lifecycle Manager (OLM), Cert-Manager, ThingsBoard, PostgreSQL, Redis, DefectDojo y GitHub Actions Runner Controller (ARC). Este último permite la ejecución de pipelines de integración y entrega continua (CI/CD) en entornos controlados.

El proceso de monitoreo se estructuró de forma que los resultados del análisis estático de código obtenidos desde **SonarCloud** se transfieren automáticamente a **DefectDojo**, donde se centralizan los hallazgos, vulnerabilidades y métricas de seguridad. Esta integración facilita la trazabilidad de riesgos y la priorización de acciones correctivas dentro del ciclo de desarrollo.

La red de pruebas se configuró mediante un router principal doméstico y un router secundario TP-Link con firmware DD-WRT, el cual actúa como gateway de seguridad y punto de monitoreo de tráfico. Ambos dispositivos permiten la segmentación de la red en zonas de confianza diferenciadas, simulando un entorno de borde (*edge network*) donde se distribuyen los nodos IoT y los servidores auxiliares.

Se incluyó una **Raspberry Pi 3 Model B** como nodo *edge* dentro de la red local. Este dispositivo ejecuta el sistema operativo Raspberry Pi OS (Raspbian) y fue configurado como servidor auxiliar para resolución de nombres y control de tráfico. Sobre la Raspberry se desplegó Docker, Portainer y dentro de ella se instaló Pi-Hole, desempeñando el rol de servidor DNS, controlador de dominio y filtro de tráfico, lo cual permitió controlar las peticiones salientes de los dispositivos IoT y mitigar la exposición a dominios maliciosos. Esta configuración favoreció el aislamiento del entorno experimental y la observabilidad de las comunicaciones entre los microcontroladores y los servicios externos.

Los microcontroladores **ESP8266** y **ESP32** se programaron con un firmware inicial (versión 1.0) con soporte para conexión Wi-Fi, publicación de datos mediante protocolo MQTT, comunicación

con la plataforma ThingsBoard y capacidad de actualización OTA (*Over-The-Air*). Este flujo de comunicación permitió validar la transmisión de telemetría, la integridad del canal cifrado (TLS) y la correcta integración con el sistema de monitoreo central.

5.1.4. Entornos de despliegue y automatización

A diferencia de un entorno basado únicamente en Docker, la infraestructura del proyecto se construyó sobre Kubernetes utilizando un **Makefile** automatizado que orquesta la instalación de todos los componentes.

El archivo `Makefile` define tareas modulares que emplean Helm, kubectl y envsubst para desplegar operadores, controladores y servicios en el namespace `devsecops`. Las principales tareas incluyen:

- **install-nginx-ingress:** Instala el controlador de entrada Nginx mediante Helm para exponer servicios dentro del clúster.
- **install-olm:** Despliega el Operator Lifecycle Manager, requerido para la instalación de operadores como Cert-Manager.
- **install-cert-manager:** Configura el gestor de certificados TLS utilizando los recursos provistos por OperatorHub.
- **install-postgres** y **install-redis:** Implementan bases de datos de soporte para DefectDojo y ThingsBoard.
- **install-dd:** Despliega DefectDojo desde su Helm chart oficial, aplicando los valores personalizados definidos en `resources/defectdojo-values.yaml`.
- **install-arc:** Configura el controlador de runners de GitHub Actions (*Actions Runner Controller*) dentro del clúster, lo que permite ejecutar pipelines CI/CD desde Kubernetes.

Este enfoque modular facilita la reproducibilidad del entorno, el versionado de infraestructura y la integración con flujos de trabajo GitOps.

5.2. Métodos

5.2.1. Control de versiones - Branching

Decisión	Descripción
Estrategia adoptada	**Trunk-Based Development**: Integración continua sobre una única rama principal, segregando accesos y roles, promoviendo aprobaciones, evitando acumulación de cambios y reduciendo la complejidad del repositorio.
Organización de ramas	**main**: Contiene la versión estable y actual del proyecto. **Short-lived branches**: Ramas creadas desde `main`, de vida corta (1–2 días), destinadas a funciones pequeñas o correcciones.
Uso de Pull Requests	Se emplean PR para todo cambio. Cada PR debe superar verificaciones CI/CD: pruebas, SAST, auditoría de dependencias y controles de seguridad. Requiere al menos una revisión humana.
Reglas sobre `main`	No se permiten commits directos. Rama protegida: todo merge debe pasar por CI/CD. El estado debe ser estable; PRs con fallas no pueden integrarse.
Flujo de trabajo	1 - Crear rama corta desde `main`. 2 - Realizar cambios pequeños y frecuentes. 3 - Subir la rama y ejecutar CI/CD. 4 - Abrir PR hacia `main`. 5 - Aprobación + validaciones exitosas. 6 - Merge y eliminación de rama.
Ventajas	Integración continua y veloz. Menor riesgo de conflictos extensos. Código siempre liberable. Adecuado para DevSecOps y pipelines automatizados.

Tabla 5.1 Control de versiones

5.2.2. Procedimientos de análisis de seguridad

El proceso de seguridad se ejecuta en múltiples niveles y fases, integrando análisis estáticos (SAST), validaciones dinámicas (DAST) y gestión centralizada de hallazgos:

1. **Análisis estático:** Implementado con SonarCloud que analiza cada *pull request* y *release* para detectar vulnerabilidades de código, malas prácticas y métricas de calidad.

2. **Auditoría de dependencias:** Mediante Cppcheck se revisan librerías con vulnerabilidades conocidas (CVE), generando reportes automáticos por cada ejecución de pipeline.

3. **Gestión de hallazgos:** Los resultados de SonarCloud se consolidan en DefectDojo, donde se clasifican por severidad y se documenta su mitigación.

5.2.3. Configuración de pipelines CI/CD con seguridad integrada

Los pipelines fueron definidos en GitHub Actions y se dividen en tres flujos principales:

- **PR Security Checks:** Se ejecuta ante cada *pull request*. Compila el firmware mediante PlatformIO, aplica el *build wrapper* de SonarCloud y ejecuta análisis de código y auditoría de dependencias.
- **Release Security Build:** Se activa al publicar una versión. Repite las validaciones anteriores y genera artefactos listos para despliegue, asegurando que solo versiones aprobadas lleguen a producción.
- **Workflow Dispatch:** Permite ejecutar manualmente un build o auditoría de seguridad desde la interfaz de GitHub Actions, sin necesidad de cambios en el código.

Cada pipeline usa el token de GitHub y el secreto `SONAR_TOKEN` para integrarse con SonarCloud y registra resultados que posteriormente son enviados a DefectDojo.

5.2.4. Estrategia DevSecOps

La estrategia adoptada busca incorporar seguridad como una práctica transversal y automatizada en todas las etapas del desarrollo:

- **Shift-left Security:** Los análisis de código y dependencias se ejecutan desde las fases iniciales, permitiendo detectar errores antes de su integración.
- **Infraestructura declarativa:** Todo el despliegue se gestiona mediante Helm Charts y Makefiles versionados, garantizando consistencia entre entornos.
- **Integración continua de seguridad:** GitHub Actions y ARC Runner dentro del clúster permiten ejecutar los pipelines en un entorno controlado, con auditoría completa.
- **Visibilidad y trazabilidad:** DefectDojo centraliza los resultados de seguridad, asociándolos con commits, builds y versiones.

5.2.5. Registro y reporte de hallazgos

Los hallazgos detectados durante los análisis se integran automáticamente a DefectDojo, donde se gestionan según su criticidad:

- **Clasificación:** Los hallazgos se categorizan como críticos, altos, medios o bajos.

- **Asociación de evidencia:** Cada hallazgo incluye referencias al pipeline, commit o release que lo originó.
- **Documentación:** El proyecto tiene la capacidad de emplear plantillas estandarizadas para registrar la causa raíz, impacto, recomendación y responsable de mitigación.
- **Seguimiento y métricas:** El proyecto tiene la capacidad de monitorear indicadores como el tiempo medio de resolución (MTTR) y la cantidad de vulnerabilidades por release.

5.3. Análisis del dispositivo ESP8266

El dispositivo ESP8266 es indicado por su fabricante para los siguientes usos:

- Electrodomésticos
- Domótica
- Enchufes y luces inteligentes
- Control inalámbrico industrial
- Monitores para bebés
- Cámaras IP
- Redes de sensores
- Dispositivos electrónicos portátiles
- Dispositivos con geolocalización Wi-Fi
- Etiquetas de identificación de seguridad
- Balizas de sistema de posicionamiento Wi-Fi

5.4. Análisis de riesgos del dispositivo

Al analizar los riesgos asociados al dispositivo que empleo en este proyecto, es valioso comprender en profundidad las especificaciones técnicas detalladas en su datasheet.[40] Este documento no solo describe las capacidades del hardware y las limitaciones operativas del dispositivo, sino que también permite delimitar la superficie de ataque y anticipar las posibles vulnerabilidades que podrían ser aprovechadas por actores maliciosos.

Tabla 5.2 Especificaciones del módulo ESP8266

Categoría	Elemento	Parámetros
Wi-Fi	Certificación	Wi-Fi Alliance
	Protocolos	802.11 b/g/n (HT20)
	Rango de frecuencia	2.4G ∼ 2.5G (2400M ∼ 2483.5M)
	Potencia de transmisión (TX Power)	802.11 b: +20 dBm 802.11 g: +17 dBm 802.11 n: +14 dBm
	Sensibilidad de recepción (Rx Sensitivity)	802.11 b: –91 dBm (11 Mbps) 802.11 g: –75 dBm (54 Mbps) 802.11 n: –72 dBm (MCS7)
	Antena	Trazo PCB, Externa, Conector IPEX, Chip cerámico
Hardware	CPU	Procesador Tensilica L106 de 32 bits
	Interfaz periférica	UART / SDIO / SPI / I2C / I2S / Control remoto IR GPIO / ADC / PWM / Luz LED y botón
	Rango de temperatura de operación	–40°C ∼ 125°C
	Tamaño del encapsulado	QFN de 32 pines (5 mm x 5 mm)
Interfaz Externa		-
Software	Modo Wi-Fi	Estación / SoftAP / SoftAP + Estación
	Seguridad	WPA / WPA2
	Cifrado	WEP / TKIP / AES
	Actualización de firmware	Descarga UART / OTA (vía red)
	Desarrollo de software	Soporte para desarrollo en servidor en la nube / Firmware y SDK para programación rápida en chip
	Protocolos de red	IPv4, TCP/UDP/HTTP
	Configuración del usuario	Conjunto de instrucciones AT, Servidor en la nube, Aplicaciones Android/iOS

5.5. Análisis del dispositivo ESP32

El dispositivo ESP32 es indicado por su fabricante para los siguientes usos:

- Hogar inteligente
- Automatización industrial

- Sanidad
- Electrónica de consumo
- Agricultura inteligente
- Terminales TPV
- Robots de servicio
- Dispositivos de audio
- Concentradores de sensores IoT genéricos de bajo consumo
- Registradores de datos IoT genéricos de bajo consumo
- Cámaras para transmisión de vídeo
- Dispositivos USB
- Reconocimiento de voz
- Reconocimiento de imágenes
- Tarjeta de red Wi-Fi + Bluetooth
- Sensores táctiles y de proximidad

5.6. Análisis de riesgos del dispositivo

Al analizar los riesgos asociados al dispositivo que empleo en este proyecto, es valioso comprender en profundidad las especificaciones técnicas detalladas en su datasheet.[41] Este documento no solo describe las capacidades del hardware y las limitaciones operativas del dispositivo, sino que también permite delimitar la superficie de ataque y anticipar las posibles vulnerabilidades que podrían ser aprovechadas por actores maliciosos.

Tabla 5.3 Especificaciones del módulo ESP32-S3 parte 1

Categoría	**Elemento**	**Parámetros**
Wi-Fi	Protocolos	IEEE 802.11 b/g/n, 20/40 MHz en 2.4 GHz
	Velocidad máxima	Hasta 150 Mbps (1T1R)
	Modos Wi-Fi	Station / SoftAP / Station + SoftAP (4 interfaces virtuales)
	Antena	Soporte para antena única con mecanismo de coexistencia Wi-Fi / BT

Tabla 5.4 Especificaciones del módulo ESP32-S3 parte 2

Categoría	Elemento	Parámetros
Hardware	CPU	Xtensa LX7 dual-core, 32 bits, hasta 240 MHz (1329.92 CoreMark)
	Memoria interna	ROM: 384 KB, SRAM: 512 KB, SRAM RTC: 16 KB
	Periféricos digitales	45 GPIOs, SPI (hasta Octal/OPI), 3× UART, 2× I2C, 2× I2S, USB OTG FS, Cámara DVP, LCD RGB/I8080/MOTO6800, RMT, LED PWM (8 canales), TWAI (CAN 2.0), SD/MMC, MCPWM, GDMA
	Periféricos analógicos	2× ADC SAR de 12 bits (hasta 20 canales) Sensor de temperatura 14 canales táctiles
Interfaz Externa		USB OTG / SPI QSPI / OPI para flash y PS-RAM externas
Software	Actualización de firmware	UART / OTA
	Desarrollo	SDK ESP-IDF, soporte para FreeRTOS
	Protocolos de red	TCP/UDP/HTTP/HTTPS, IPv4
	Configuración	Conjunto de comandos AT, soporte en IDEs y herramientas Espressif
Seguridad	Funciones de seguridad	Secure Boot Flash Encryption
	Criptografía HW	AES-128/256, SHA, RSA, RNG, HMAC, Firma Digital
	OTP	4 Kbit, con 1792 bits para el usuario
Low Power	Modos de energía	Active / Modem-sleep / Light-sleep / Deep-sleep (7 µA)
	Retención RTC	RTC Memory activa en Deep-sleep

5.7. Criterios de evaluación de riesgos

De acuerdo con las normas ISO 31000:2018 e ISO/IEC 27005:2022, los riesgos se evalúan en función de su **probabilidad de ocurrencia** y su **impacto potencial**. Los valores asignados se muestran en las Tablas 5.5 y 5.6.

Valor	Nivel	Descripción
1	**Muy baja**	Requiere acceso físico o conocimiento profundo del hardware.
2	**Baja**	Necesita tiempo, herramientas especializadas o errores de configuración.
3	**Media**	Posible con técnicas conocidas o exploits públicos.
4	**Alta**	Existen vectores de ataque comunes y fácilmente explotables.
5	**Muy alta**	Vulnerabilidad conocida y activamente explotada.

Tabla 5.5 Escala de probabilidad

Valor	Nivel	Descripción
1	**Muy bajo**	Afecta mínimamente la operación o confidencialidad.
2	**Bajo**	Pérdida de datos no críticos o breve interrupción del servicio.
3	**Medio**	Interrupción parcial o exposición de datos limitados.
4	**Alto**	Interrupción prolongada o pérdida significativa de datos.
5	**Muy alto**	Compromiso total del dispositivo o daño físico.

Tabla 5.6 Escala de impacto

5.7.1. Mapa de calor de probabilidad por impacto

	Impacto				
Probabilidad	**1 (Muy bajo)**	**2 (Bajo)**	**3 (Medio)**	**4 (Alto)**	**5 (Muy alto)**
5 (Muy alta)	5	10	15	20	25
4 (Alta)	4	8	12	16	20
3 (Media)	3	6	9	12	15
2 (Baja)	2	4	6	8	10
1 (Muy baja)	1	2	3	4	5

Tabla 5.7 Mapa de calor de probabilidad vs. impacto

5.7.2. Clasificación de riesgo según el producto P×I

Puntaje 1–4:	**Verde (bajo)**
Puntaje 5–9:	**Amarillo (medio-bajo)**
Puntaje 10–14:	**Naranja (alto)**
Puntaje 15–25:	**Rojo (crítico)**

Tabla 5.8 Clasificación de riesgo según el producto P×I

5.7.3. Esquema de identificación y clasificación de riesgos

Para facilitar la identificación y las categorías de los riesgos, se define como estrategia de etiquetado, el siguiente esquema:

ID	Categoría de riesgo
NET-01	Riesgos de red
FMW-01	Riesgos de firmware
IOT-01	Riesgos del dispositivo IoT
CI-01	Riesgos del pipeline CI/CD
SEC-01	Riesgos generales de seguridad

Tabla 5.9 Esquema de identificación de riesgos

5.8. Matriz de riesgos del dispositivo IoT

ID	Vulnerabilidad	Riesgo asociado	Probabilidad	Impacto	Medidas de mitigación
IOT-01	Puertos UART y depuración sin protección	Extracción de credenciales, flasheo no autorizado, acceso físico malicioso	Alta	Alto	Deshabilitar o sellar puertos, implementar autenticación de firmware, proteger físicamente el hardware.

Tabla 5.10 Matriz de riesgos ESP8266 Parte 1

ID	Vulnerabilidad	Riesgo asociado	Probabilidad	Impacto	Medidas de mitigación
FMW-01	Firmware OTA sin firma o sin canal seguro	Inyección de firmware malicioso, secuestro del dispositivo (bricking)	Media	Alto	Implementar OTA con firma digital y cifrado TLS, verificación de integridad post-flasheo.
SEC-01	Credenciales por defecto o hardcodeadas	Acceso remoto no autorizado, escalamiento de privilegios	Alta	Alto	Cambio de contraseñas, rotación de claves, uso de gestión segura de secretos.
NET-01	Protocolos inseguros (WEP, HTTP)	Espionaje, inyección de tráfico, suplantación de dispositivo	Media	Alto	Uso obligatorio de TLS 1.2+, cifrado WPA2/WPA3, validación de certificados.
IOT-02	Limitaciones de CPU y memoria	Imposibilidad de aplicar cifrado fuerte o autenticación compleja, degradación de rendimiento	Media	Medio	Uso de algoritmos ligeros (AES-128, ECC), optimización de firmware y priorización de funciones críticas.
FMW-02	Supply Chain Attack (ataque a la cadena de suministro)	Inserción de componentes o firmware comprometido antes de la entrega o en actualizaciones	Media	Muy alto	Verificación de procedencia del hardware, hashes de firmware, control de versiones y auditorías de proveedores.

Tabla 5.11 Matriz de riesgos ESP8266 Parte 2

ID	Vulnerabilidad	Riesgo asociado	Probabilidad	Impacto	Medidas de mitigación
FMW-03	Bricking intencional del dispositivo	Inutilización total del equipo, pérdida de servicio y disponibilidad	Media	Alto	Implementar recuperación segura, respaldo de firmware, bloqueo de escritura no autorizada.
SEC-02	Ataque de denegación de servicio (DoS)	Saturación de red o del microcontrolador, caída del sistema	Media	Medio	Filtrado de tráfico, limitación de solicitudes, detección de anomalías y monitoreo.
NET-02	Configuraciones de red inseguras o exposición pública del broker MQTT	Intercepción o manipulación de datos del sensor, secuestro de sesiones MQTT	Alta	Alto	Segmentación de red, autenticación TLS, uso de topics con permisos granulares.
FMW-04	Ausencia de arranque seguro (Secure Boot)	Ejecución de firmware alterado o comprometido al inicio	Media	Muy alto	Implementar verificación de firma al arranque, bloqueo de flasheo sin autorización.

Tabla 5.12 Matriz de riesgos ESP8266 Parte 3

5.8.1. Puntaje de vulnerabilidades IoT

Uso una escala 1–5 para Probabilidad *P* e Impacto *I*.

Vulnerabilidad	P (1–5)	I (1–5)	Puntaje
Puertos UART sin protección	4	4	16
Firmware OTA sin firma	3	5	15
Credenciales por defecto	4	5	20
Protocolos inseguros	3	4	12
Limitaciones CPU/mem	3	3	9
Supply Chain Attack	3	5	15
Bricking intencional	3	4	12
DoS	3	3	9
Broker MQTT expuesto	4	5	20
Ausencia Secure Boot	3	5	15

Tabla 5.13 Puntajes de vulnerabilidades

Capítulo 6

Arquitectura de la solución

6.1. Diagrama de arquitectura

La solución se estructura en tres niveles principales:

- **Dispositivo IoT:** ESP32 con firmware basado en PlatformIO (*Arduino Framework*) encargado de la transmisión de datos mediante MQTT.
- **Plataforma intermedia:** Un *broker* MQTT que recibe los mensajes publicados por el dispositivo y los distribuye hacia las plataformas de gestión.
- **Ecosistema DevSecOps en Kubernetes:** Conjunto de servicios e infraestructura para análisis de seguridad, gestión de vulnerabilidades, monitoreo de dispositivos y ejecución de pipelines CI/CD.

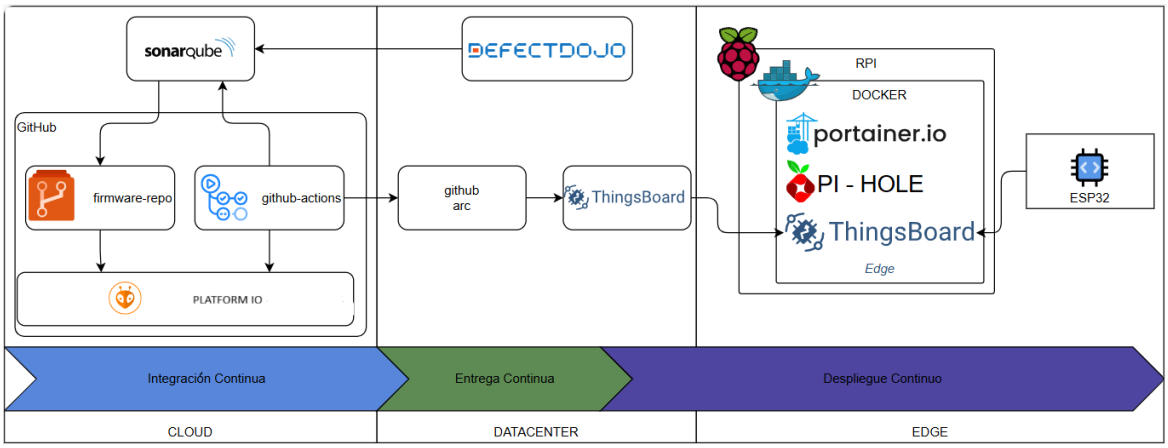


Figura 6.1 Diagrama de arquitectura

6.2. Integración SonarCloud

En esta sección se presentan las evidencias obtenidas durante la implementación del flujo de CI/CD con integración de análisis estático (SonarCloud) y generación de artefactos reproducibles. El pipeline validó cada *Pull Request* asegurando que el proyecto compilara correctamente, que no se detectaran fallas mediante el análisis estático, que no existieran dependencias desactualizadas y que se ejecutara el análisis con SonarCloud para identificar vulnerabilidades y problemas de calidad. La Figura 6.2 resume este flujo de CI/CD, desde la construcción y verificación del código hasta la liberación del artefacto firmado, garantizando que solo se integren cambios confiables en la rama *main*.

```yaml
name: PR security checks
on:
  pull_request:
    branches: [ main ]
  workflow_dispatch:
jobs:
  security-check:
    name: Verificación PR (Compila + Audita)
    runs-on: ubuntu-latest
    env:
      BUILD_WRAPPER_OUT_DIR: build-out
    steps:
      - name: Checkout repo
        uses: actions/checkout@v4
        with:
          fetch-depth: 0
      - name: Set up Python
        uses: actions/setup-python@v4
        with:
          python-version: '3.13'
      - name: Install PlatformIO
        run: |
          python3 -m pip install --upgrade pip
          pip3 install -U platformio setuptools
      - name: Comprobar dependencias desactualizadas
        run: pio pkg outdated
      - name: Instalar Build Wrapper
        uses: SonarSource/sonarqube-scan-action/install-build-wrapper@v5
      - name: Analisis estatico PIO (CPPCHECK)
        run: pio check
      - name: Analisis de dependencias
        run: pio pkg outdated
      - name: Compilar con PlatformIO
        run: |
          build-wrapper-linux-x86-64 --out-dir ${{env.BUILD_WRAPPER_OUT_DIR}} platformio run || true
      - name: Análisis SonarQube
        uses: SonarSource/sonarqube-scan-action@v6
        env:
          GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}
          SONAR_TOKEN: ${{ secrets.SONAR_TOKEN }}
        with:
          args: >
            -Dsonar.cfamily.compile-commands="${{env.BUILD_WRAPPER_OUT_DIR}}/compile_commands.json"
```

Figura 6.2 Construcción de pipeline SAST.

SonarCloud implementa un modelo de autenticación federada que permite el acceso mediante proveedores de identidad como GitHub, Bitbucket, GitLab, Azure DevOps o SSO. Estos métodos mejoran la seguridad al evitar la gestión de credenciales propias y aprovechar los controles nativos de cada servicio, como la autenticación multifactor.

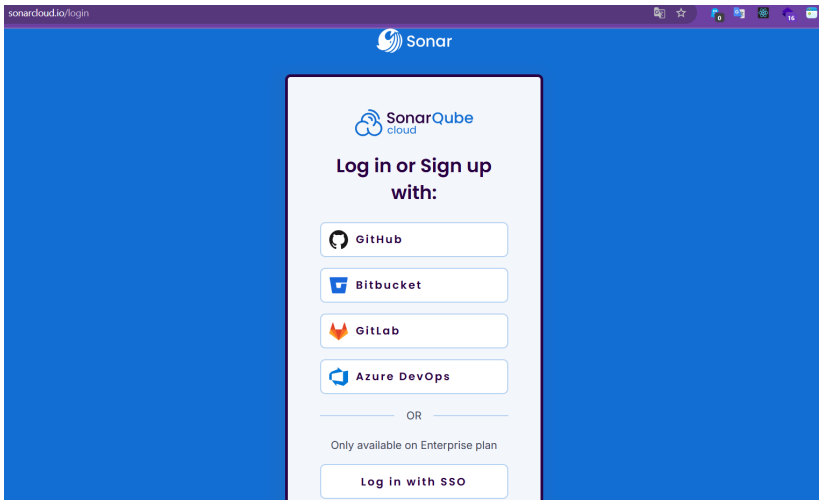


Figura 6.3 Ingreso a SonarCloud

La Figura 6.4 presenta el tablero principal de SonarCloud donde se visualiza el estado del proyecto, métricas globales de calidad y el nivel de riesgo detectado.

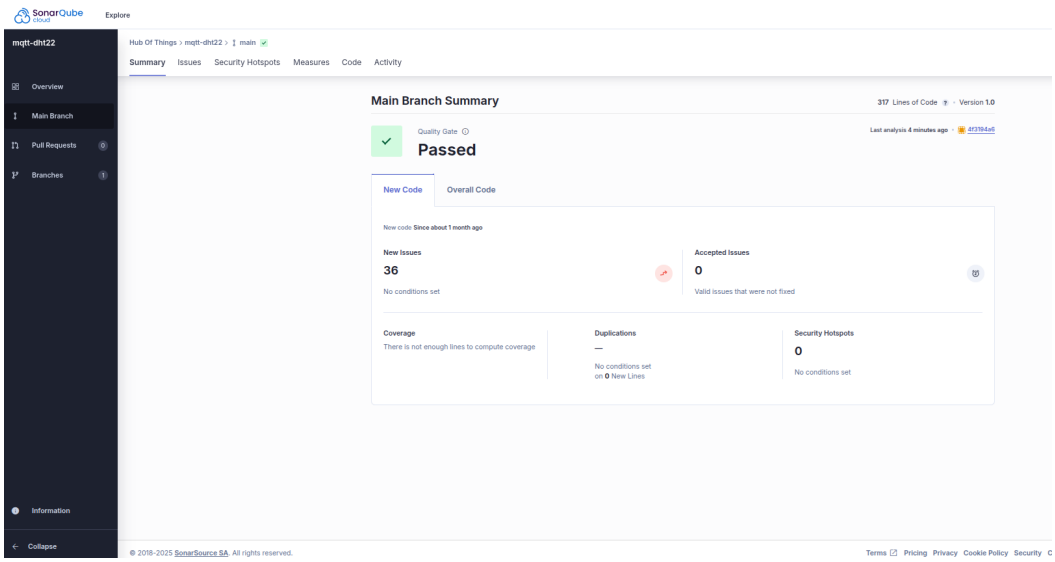


Figura 6.4 Vista principal del proyecto en SonarCloud.

En la figura 6.5 se observa el resultado de un análisis exitoso en SonarCloud donde el proyecto ha superado satisfactoriamente los criterios de calidad establecidos. El panel muestra que no se detectaron vulnerabilidades, errores críticos ni problemas de mantenibilidad, lo que indica que el código cumple con los estándares definidos y no presenta riesgos relevantes para su despliegue o evolución.

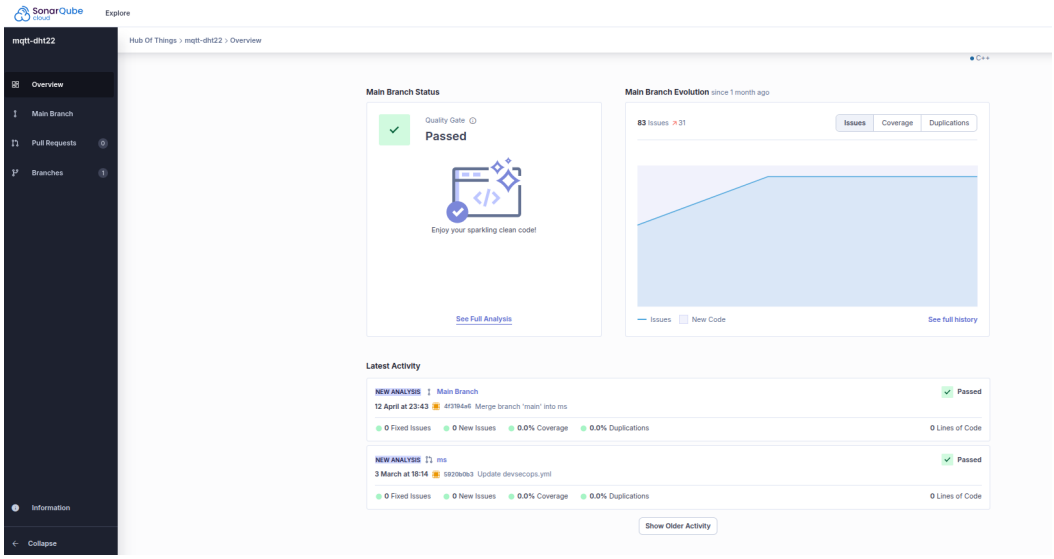


Figura 6.5 Análisis superado en SonarCloud.

6.3. Integración GitHub actions

GitHub Actions es una plataforma de automatización integrada en GitHub que permite ejecutar flujos de trabajo (*workflows*) en respuesta a eventos del repositorio, como commits, pull requests o publicaciones de versiones. Estos flujos pueden realizar tareas de integración continua (CI) y entrega continua (CD), incluyendo compilación, análisis estático de código, ejecución de pruebas y validación de seguridad antes de permitir la integración de cambios en ramas principales. En la imagen 6.6, se evidencia precisamente esta integración: el pull request generado activa automáticamente varios checks configurados mediante GitHub Actions, entre ellos procesos de compilación y auditoría del código. Uno de estos controles se encuentra aún en ejecución, mientras otro — *PR security checks / Verificación PR (Compila + Audita)*— aparece validado correctamente. Gracias a esta automatización GitHub determina si el pull request puede fusionarse de manera segura, asegurando que los cambios cumplan los requisitos de calidad y seguridad definidos por el proyecto antes de permitir su integración a la rama main.

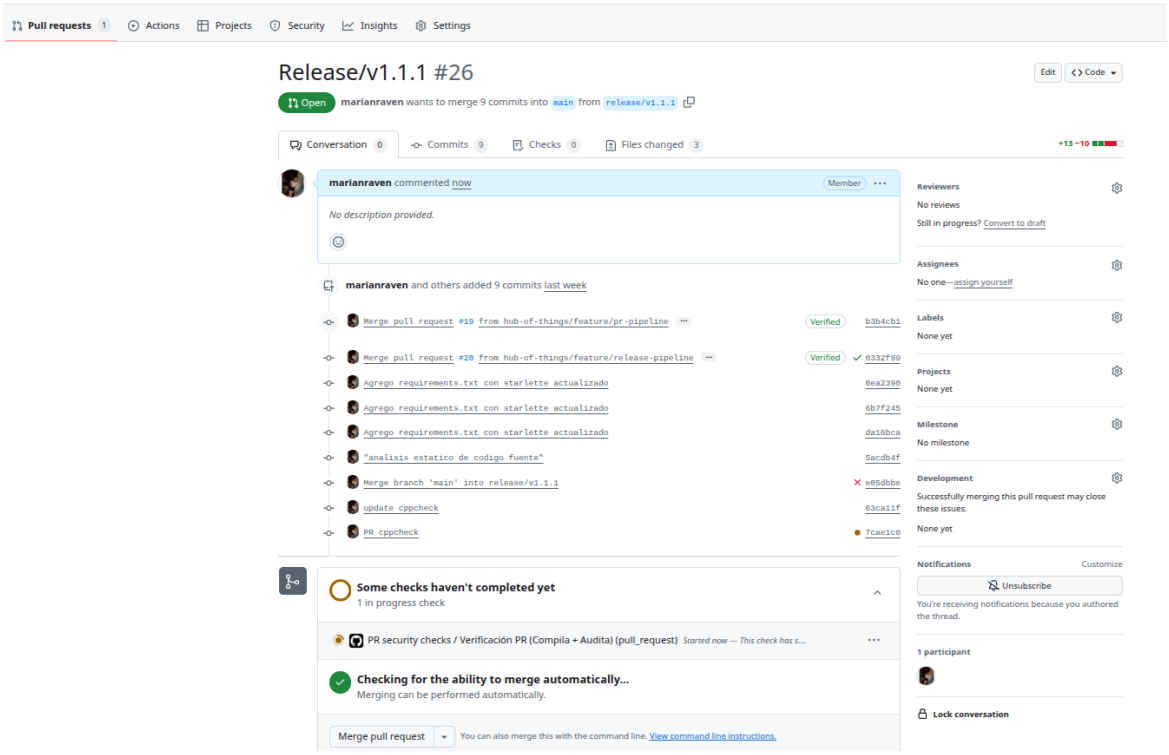


Figura 6.6 Release exitoso en *GitHub Actions* tras cumplir los *quality gates*.

6.3.1. Manejo de Tokens

Para garantizar la integridad del proceso de integración y evitar compromisos en la cadena CI, el flujo entre *GitHub Actions, SonarCloud y DefectDojo* se implementó con manejo seguro de credenciales:

- Los tokens se almacenan exclusivamente en *GitHub Secrets*.
- La comunicación con DefectDojo se realiza mediante *API Key* con permisos mínimos (principio de menor privilegio).
- No se exponen credenciales en los logs del pipeline.

La Figura 6.7 presenta la evidencia del flujo de tokens y la configuración aplicada.

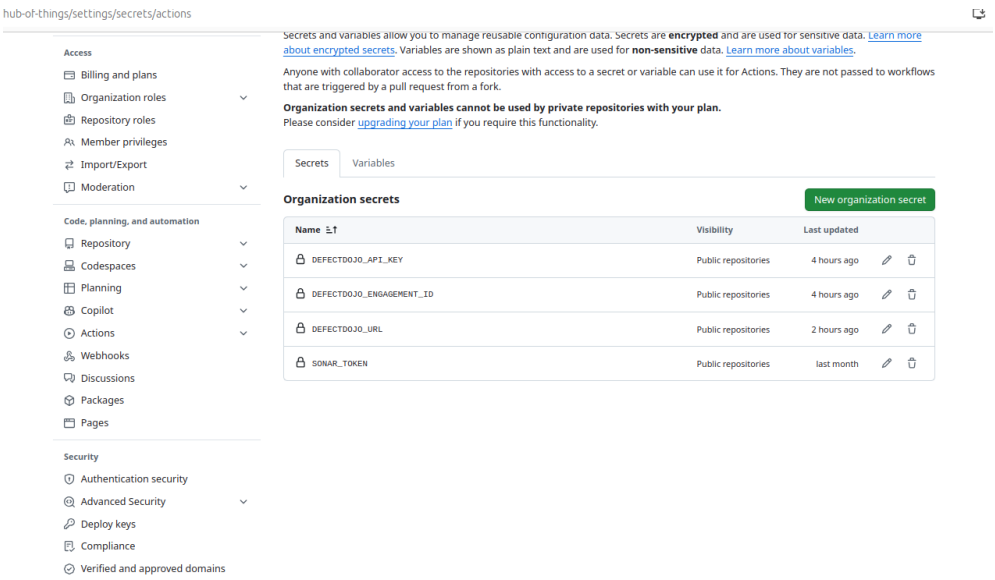


Figura 6.7 Configuración de tokens utilizados en la integración CI.

Por su parte, la Figura 6.8 muestra la integración de manera segura los dispositivos de IoT a la plataforma *ThingsBoard*.

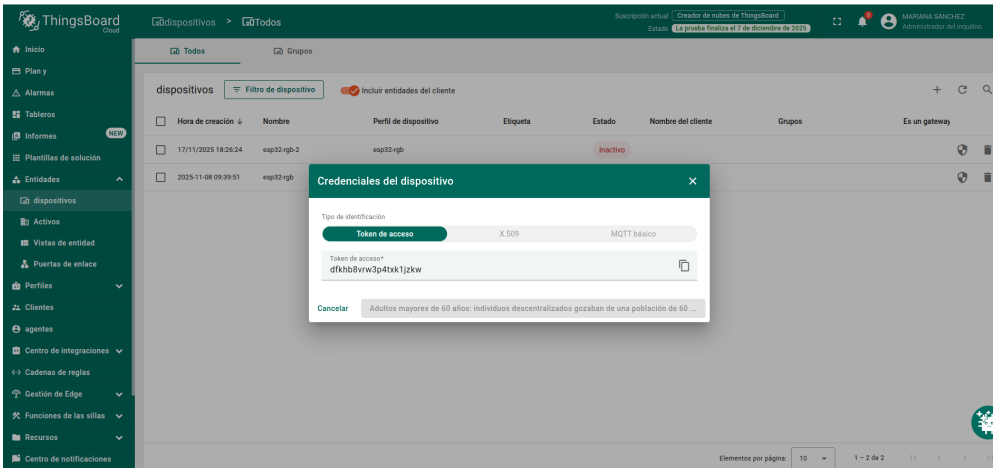


Figura 6.8 Configuración de tokens para ThingsBoard

6.4. Integración DefectDojo

DefectDojo es una plataforma diseñada para centralizar y gestionar hallazgos de seguridad provenientes de múltiples herramientas de análisis, permitiendo el seguimiento del ciclo de vida de las vulnerabilidades. Antes de interactuar con sus funcionalidades, es necesario validar el acceso mediante su sistema de autenticación.

La Figura 6.9 muestra la pantalla de autenticación del sistema.

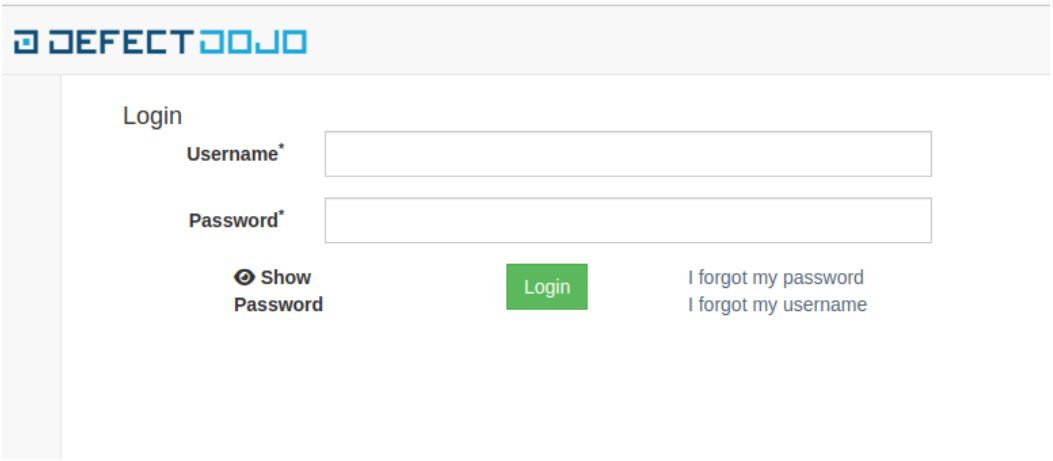


Figura 6.9 Inicio de sesión de DefectDojo.

Como punto de partida se presenta el dashboard base de DefectDojo sin datos cargados, previo al proceso de integración. Luego, desde DefectDojo, se realiza la recolección de datos de SonarCloud mediante una API de manera automatizada.

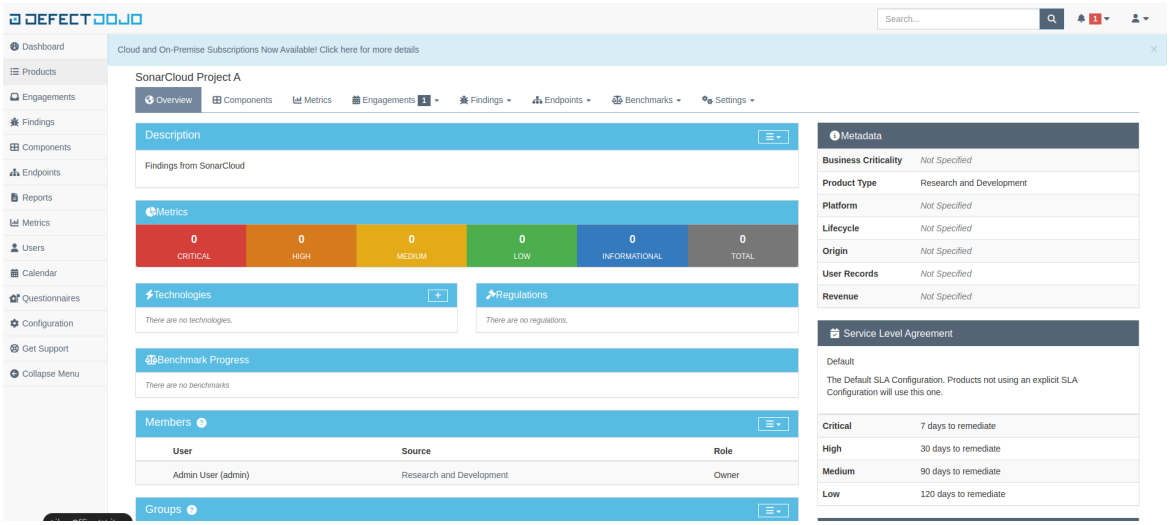


Figura 6.10 Dashboard DefectDojo.

En la Figura 6.11 se muestra la configuración de la integración funcionando correctamente.

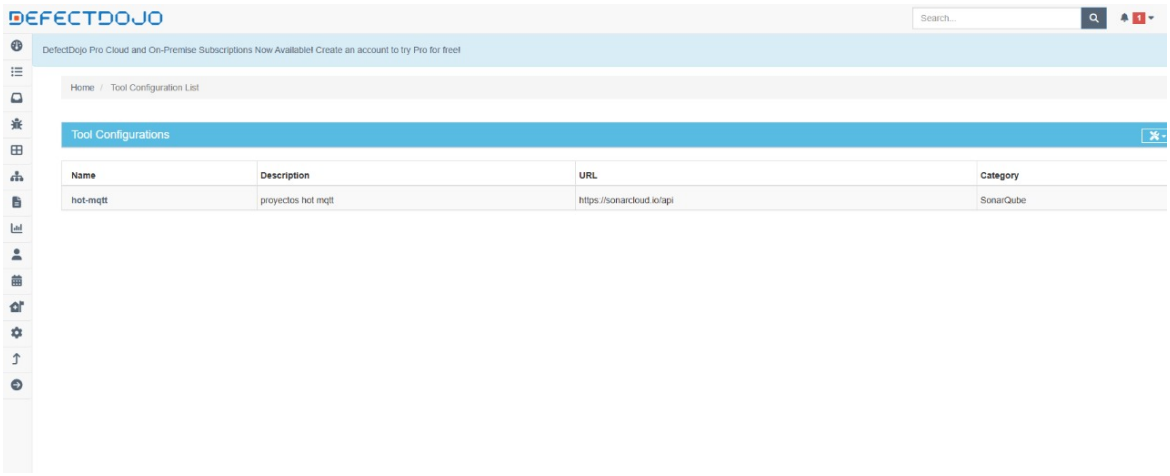


Figura 6.11 Integración activa entre SonarCloud y DefectDojo.

6.5. Integración en ThingsBoard

ThingsBoard es una plataforma de gestión IoT de código abierto que permite la recolección, procesamiento, visualización y control remoto de datos provenientes de dispositivos conectados. Su arquitectura basada en protocolos estándar —como MQTT y HTTP— facilita la integración de sensores, actuadores y aplicaciones, ofreciendo además capacidades avanzadas de telemetría, reglas de negocio y actualización de firmware.

En el dispositivo de IoT se implementa la biblioteca IotWebConf la cual proporciona un mecanismo para configurar parámetros mediante un portal cautivo, tales como las credenciales de red Wi-Fi y los datos necesarios para la conexión con ThingsBoard, esto evita la necesidad de almacenar información sensible de manera fija en el firmware y facilita la parametrización por parte del usuario final. La comunicación entre el dispositivo y ThingsBoard se establece mediante el uso de WiFiClientSecure, una implementación que proporciona soporte para conexiones TLS. Para garantizar la validez del canal seguro, es necesario incorporar el certificado público de la autoridad certificadora correspondiente.

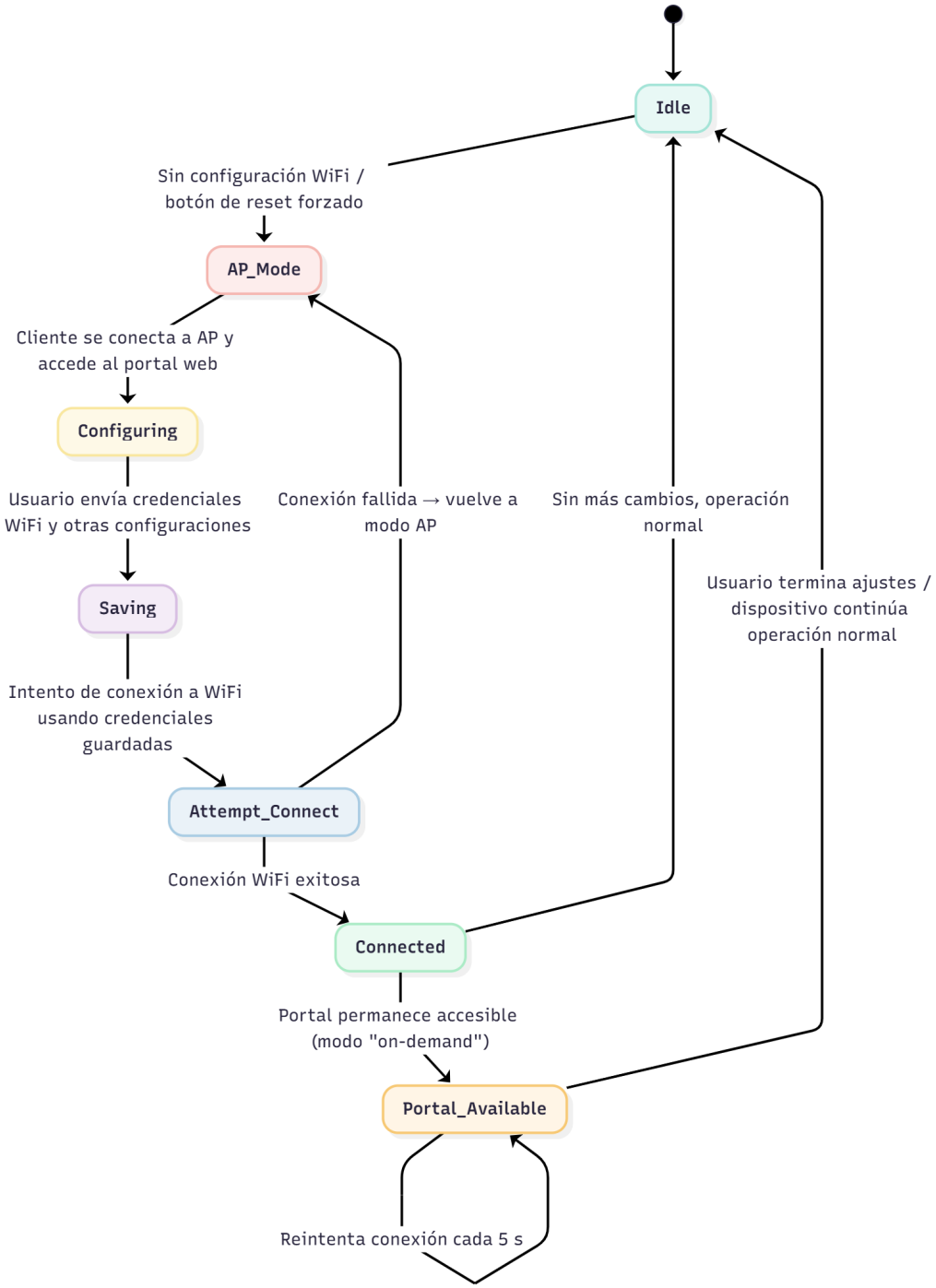


Figura 6.12 Diagrama de estados de IoTWebConf

La pantalla de acceso de ThingsBoard ofrece distintos métodos de autenticación orientados a reforzar la seguridad y facilitar el acceso del usuario. Los botones de inicio de sesión mediante Google, Facebook, GitHub y Apple implementan identidad federada, lo que permite delegar la autenticación

a proveedores externos que aplican controles avanzados, como autenticación multifactor, gestión robusta de contraseñas y detección de actividad anómala.

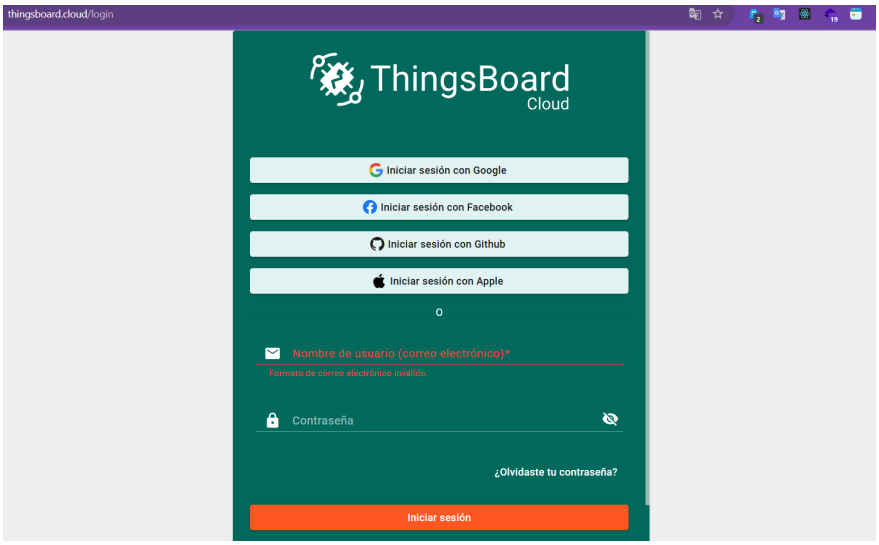


Figura 6.13 Login ThingsBoard

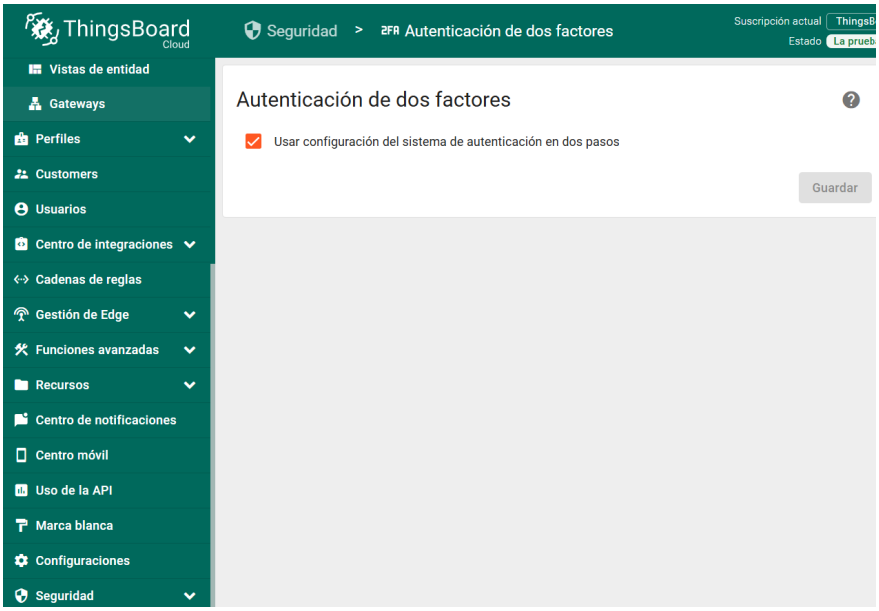


Figura 6.14 ThingsBoard MFA

A continuación, el diagrama de secuencia de la figura 6.15 ilustra el flujo de comunicación entre los elementos involucrados desde la perspectiva de ThingsBoard.

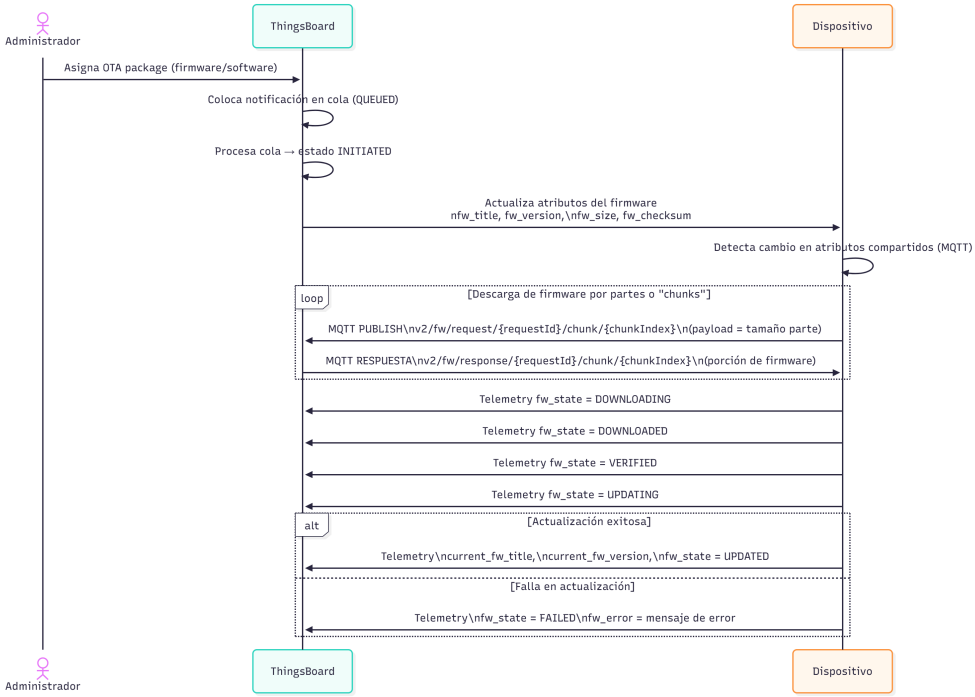


Figura 6.15 Diagrama de secuencia ThingsBoard

Descripción del diagrama de secuencia

El flujo inicia cuando el administrador carga en ThingsBoard un paquete OTA que contiene el firmware o software a instalar, con sus metadatos asociados. Una vez asignado el paquete al dispositivo, ThingsBoard coloca la notificación correspondiente en una cola interna y cambia el estado del proceso a la condición denominada *QUEUED*. Luego, al procesar dicha cola, ThingsBoard cambia el estado a *INITIATED*, momento en el cual actualiza los atributos compartidos del dispositivo con la información del firmware, incluyendo su título, versión, tamaño total y *checksum*. El dispositivo permanece suscripto a los atributos compartidos mediante MQTT y, al detectar la modificación correspondiente a un nuevo paquete OTA, inicia el procedimiento de descarga. El intercambio se realiza mediante un mecanismo basado en fragmentos o porciones del firmware, denominados *chunks*. El dispositivo solicita cada fragmento a través de un tópico MQTT específico, que incluye el identificador de la petición y el índice del fragmento requerido. ThingsBoard responde enviando la porción del firmware correspondiente, y el dispositivo repite este procedimiento en un ciclo hasta completar la descarga de todos los fragmentos. A lo largo del proceso, el dispositivo reporta de manera continua su estado a través de telemetría, transitando las etapas *DOWNLOADING*, *DOWNLOADED*, *VERIFIED* y *UPDATING*. Una vez finalizada la descarga, el dispositivo verifica la integridad del firmware mediante la comparación del *checksum* provisto por ThingsBoard. Si la verificación es exitosa, procede a actualizar el firmware y posteriormente reporta el estado *UPDATED*, junto con la información de versión vigente. En caso de producirse un error en cualquiera de las etapas, el dispositivo comunica el estado *FAILED* e informa además el mensaje de error asociado. Esto permite

que el administrador identifique la causa del fallo y evalúe la necesidad de reiniciar el proceso de actualización o realizar acciones correctivas. El diagrama representa de manera secuencial la interacción entre el administrador, ThingsBoard y el dispositivo para llevar a cabo una actualización OTA segura y controlada, detallando el intercambio de atributos, la descarga fragmentada mediante MQTT y los mecanismos de notificación del estado del proceso.

Diagrama de estados de conexión

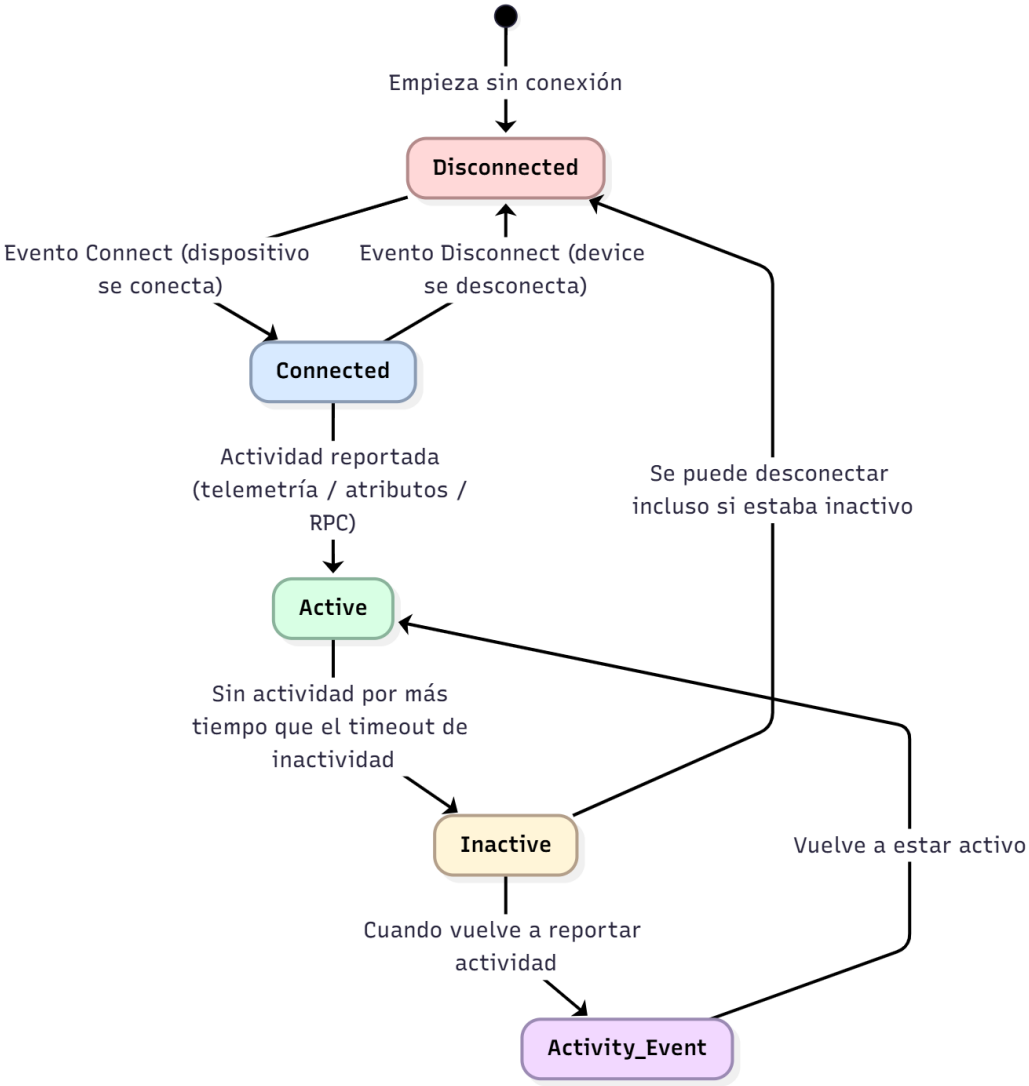


Figura 6.16 Diagrama de estados de conexión

Estado	Descripción
Disconnected	El dispositivo no está conectado o acaba de desconectarse.
Connected	Ha ocurrido un evento de conexión (*Connect event*).
Active	El dispositivo ha reportado actividad recientemente (telemetría, atributo, RPC, etc.).
Inactive	No ha habido actividad por un tiempo definido (timeout de inactividad).
Activity_Event	Estado intermedio o de transición cuando el dispositivo pasa de inactivo a activo nuevamente.

Tabla 6.1 Descripción de los estados de actividad del dispositivo

6.6. Integración de PI-Hole

Pi-hole funciona como un servidor DNS local que filtra y analiza todas las solicitudes de los dispositivos, permitiendo aplicar listas de bloqueo, reglas personalizadas y monitoreo en tiempo real. La Figura 6.17 muestra la pantalla de autenticación utilizada para acceder al panel administrativo de Pi-hole.

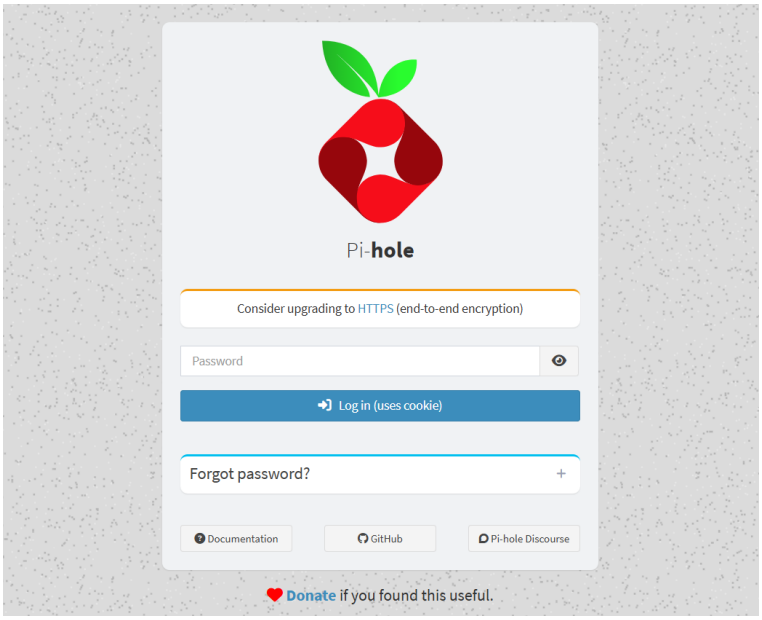


Figura 6.17 Pantalla de autenticación de Pi-hole.

Una vez autenticado, el panel principal (Figura 6.18) ofrece métricas detalladas acerca del tráfico DNS, consultas permitidas y bloqueadas, así como estadísticas de comportamiento que permiten identificar patrones anómalos o intentos de comunicación con dominios no confiables.

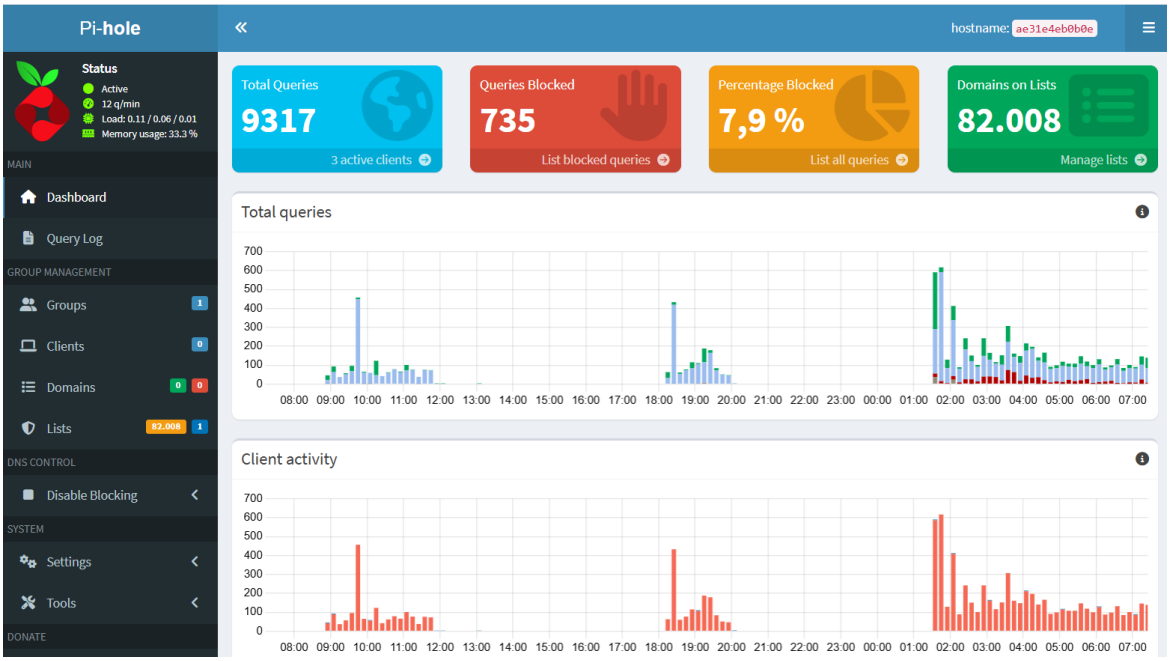


Figura 6.18 Dashboard de Pi-hole

6.7. Detección y monitoreo

El proceso de detección se lleva adelante desde DefectDojo a SonarCloud. En medio de este proceso cualquier hallazgo identificado por SonarCloud genera una alerta inmediata enviada por e-mail. La Figura 6.19 muestra el correo de notificación automática recibido inmediatamente después de un análisis fallido.

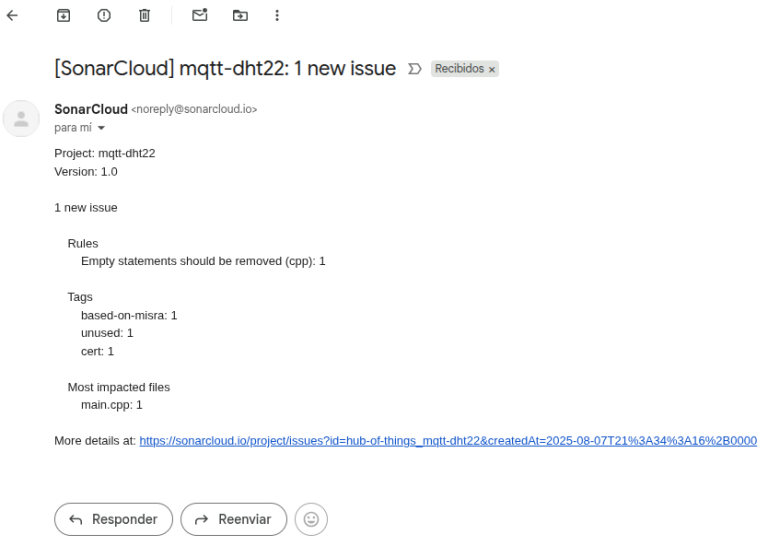


Figura 6.19 *E-mail* enviado automáticamente desde *SonarCloud* ante detección de actividad.

6.7.1. Seguimiento y métricas

En la Figura 6.20 se observa el dashboard de *DefectDojo* mostrando los *findings* cargados de manera automatizada desde *SonarCloud*.

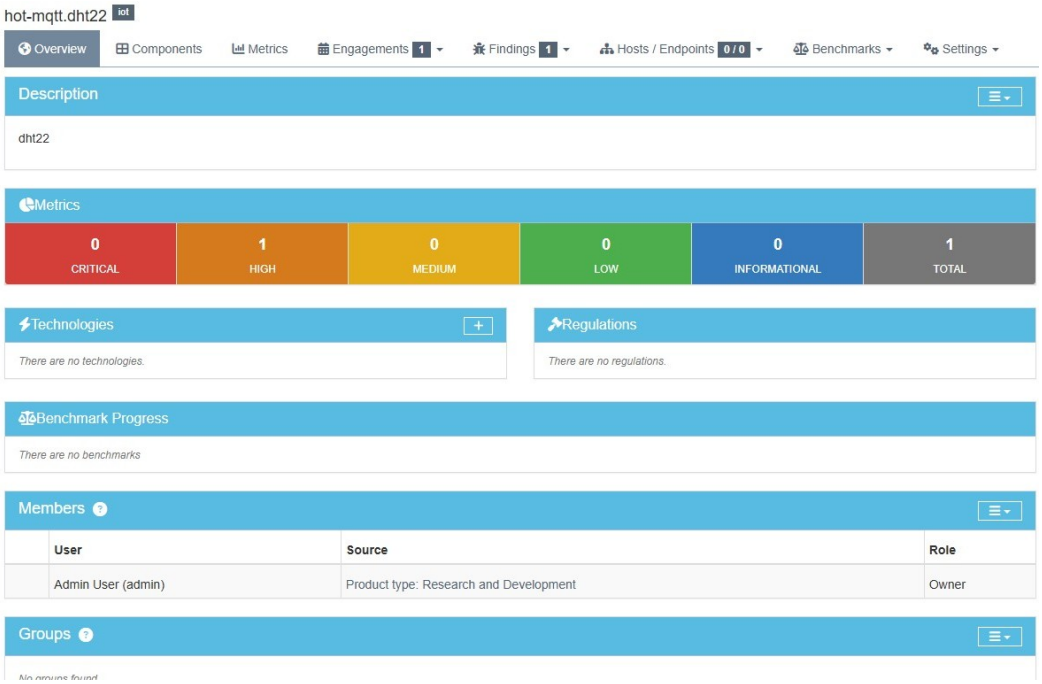


Figura 6.20 Dashboard DefectDojo con *findings* importados

6.7.2. Telemetría y registros

El despliegue incluye un módulo de monitoreo en *ThingsBoard* que registra actividades, y métricas del *firmware* (versión, hash de integridad y *lastSeen*) como parte del flujo operativo de la plataforma.

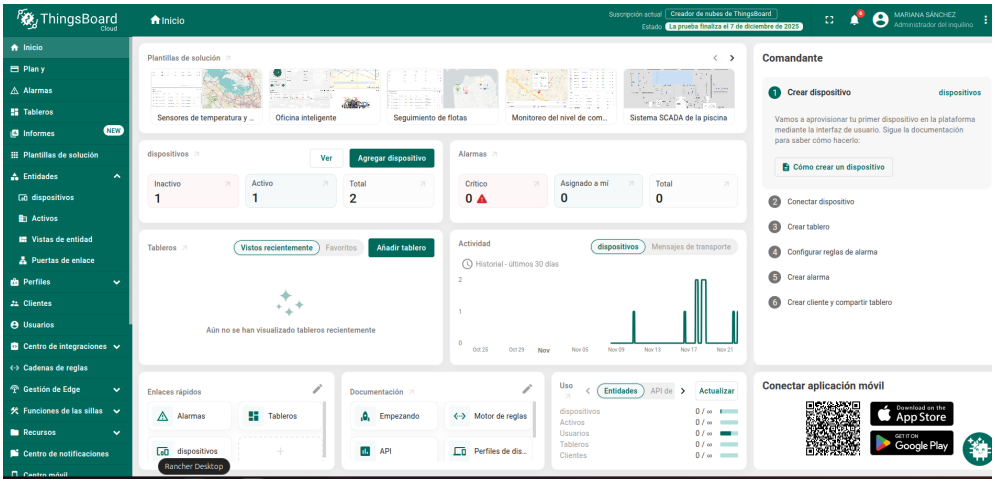


Figura 6.21 Plataforma *ThingsBoard*



Figura 6.22 ThingsBoard vista de actividad

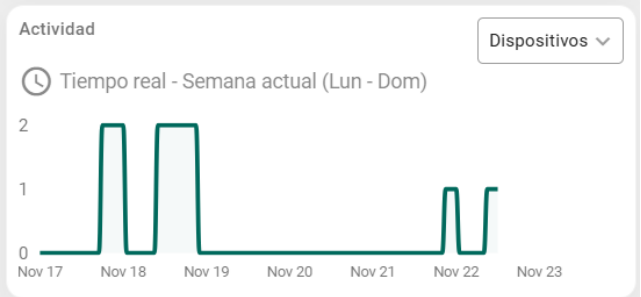


Figura 6.23 Envío de datos

La plataforma ThingsBoard incorpora mecanismos de actualización de firmware mediante OTA como parte de la operación del sistema. Este proceso permite que los dispositivos reciban nuevas versiones del software, en donde para cada actualización, la plataforma almacena información como el número de versión, el identificador hash, el estado de distribución y la fecha en que fue puesta a disposición. De esta manera, es posible reconstruir la secuencia completa de actualizaciones aplicadas y verificar la integridad de cada una. Desde el punto de vista operativo, cuando un nuevo firmware es cargado en el repositorio, la plataforma lo registra como una versión disponible y establece las condiciones para su distribución. Los dispositivos consultan periódicamente la existencia de nuevas versiones a través del canal de comunicación previamente configurado. En caso de que se detecte una versión superior a la instalada, el dispositivo podría iniciar el proceso de descarga y verificación. Solo después de comprobar la integridad del archivo recibido, se procede a su escritura en la partición correspondiente y al reinicio controlado del sistema.

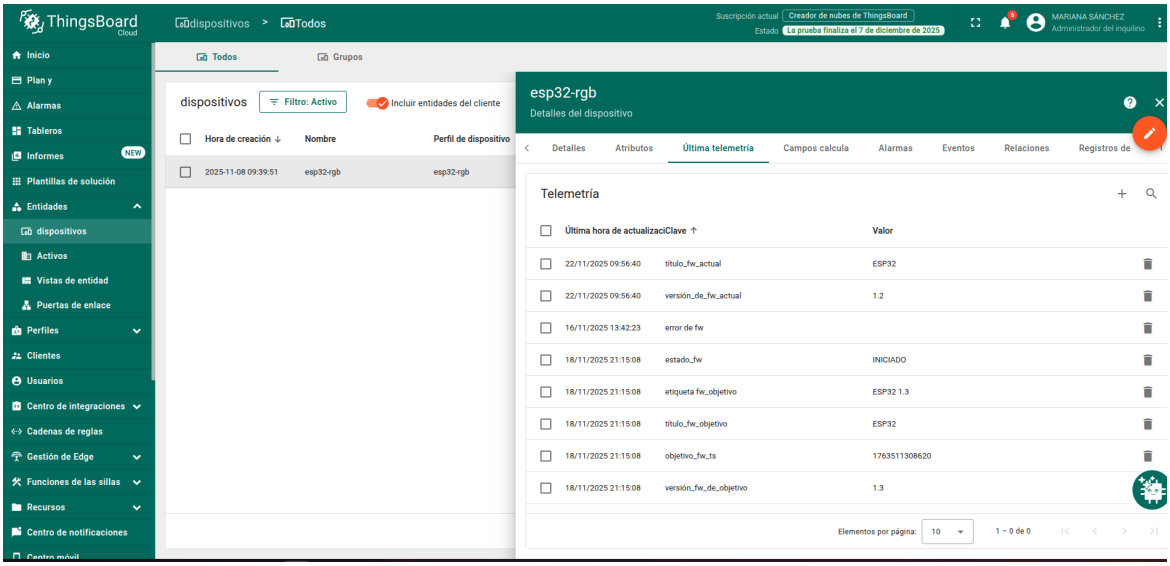


Figura 6.24 Actualizaciones OTA

6.8. Conectividad como vector de ataque y segmentación

6.8.1. Características de red por defecto

Resumen de la topología por defecto

Descripción general: Red básica donde todos los dispositivos se conectan a un único router. Es simple y con facilidad de implementación, ya que basta con un único router para proporcionar conectividad a todos los dispositivos.
Router principal: IP `192.168.1.1`, conectado a WAN, funciona como gateway, NAT y DHCP para `192.168.1.0/24`.
Dispositivos: Todos comparten el mismo dominio de difusión, lo que expone el tráfico interno.
Riesgos principales:

- No hay segmentación ni aislamiento.
- Un compromiso impacta a todos los equipos.
- No existen controles avanzados de tráfico o registros.

Adecuación: Útil solo para entornos domésticos. No apta para producción o sistemas sensibles.
Hacia la mitigación: Este modelo sirve como base para proponer una red mínima segura con segmentación y un segundo nivel de enrutamiento.

6.8.2. Matriz de riesgos de la red

La siguiente matriz resume los riesgos asociados a la topología de red por defecto y las mejoras que aporta la red mínima segura propuesta.

ID	Vulnerabilidad o debilidad	Riesgo asociado	Probabilidad	Impacto	Medidas de mitigación / mejora
NET-03	**Segmento único sin aislamiento**	Movimiento lateral entre dispositivos, posible propagación de malware o acceso no autorizado.	Alta	Alto	Implementar subredes aisladas mediante router secundario con firmware DD-WRT; crear VLAN o zonas DMZ separadas para IoT y red principal.
NET-04	**Router sin cortafuegos avanzado ni filtrado de tráfico**	Tráfico no controlado, exposición de servicios internos y falta de registros de actividad.	Media	Alto	Configurar reglas de *firewall*, NAT y monitoreo en DD-WRT; registrar y limitar conexiones sospechosas.
NET-05	**Credenciales predeterminadas o débiles en equipos de red**	Compromiso total del router o manipulación de la configuración de red.	Muy alta	Alto	Cambiar contraseñas por defecto, habilitar autenticación fuerte y deshabilitar acceso remoto innecesario.
NET-06	**Actualizaciones de firmware no aplicadas**	Exposición a vulnerabilidades conocidas en routers o puntos de acceso.	Alta	Alto	Mantener firmware actualizado, usando DD-WRT estable y realizar copias de seguridad de la configuración.

Tabla 6.2 Matriz de riesgos de la topología de red por defecto y mitigaciones

Vulnerabilidad	P (1–5)	I (1–5)	Score
Segmento único sin aislamiento	4	4	16
Router sin cortafuegos avanzado	3	4	12
Credenciales débiles en equipos de red	5	4	20
Actualizaciones de firmware no aplicadas	4	4	16

Tabla 6.3 Puntajes de vulnerabilidades de la topología de red

6.8.3. Características de red con seguridad

Resumen de la topología propuesta

Descripción general: La red implementa un esquema en **cascada** mediante un router principal conectado a WAN y un router secundario con DD-WRT que crea una subred aislada para dispositivos de laboratorio.
Router principal: IP `192.168.1.1`, conectado a WAN. Funciona como **puerta de enlace predeterminada** para la red interna y enlaza con el router secundario para generar segmentación.
Router secundario (DD-WRT): IP `192.168.8.1`, crea una **subred independiente** (`192.168.8.0/24`). Proporciona **DHCP** y funciona como capa adicional de aislamiento respecto de la red principal.
Dispositivos en la subred:

- **ESP8266 (`192.168.8.145`):** Puertos `8883 (MQTT)`, `80 (HTTP)`. Microcontrolador Wi-Fi conectado a sensores y al broker IoT.
- **Raspberry Pi (`192.168.8.139`):** Puertos `22 (SSH)`, `53 (DNS)`, `8883 (MQTT)`, `443 (HTTPS)`. Nodo multifunción para administración, servicios de red y pruebas web.

Aislamiento y control: El router DD-WRT opera como **firewall**, limitando el alcance de los dispositivos IoT y evitando exposición innecesaria hacia la red principal.
Adecuación: Configuración aplicando segmentación y defensa en profundidad sin afectar la red doméstica o productiva.
Hacia la mitigación: Esta arquitectura permite evaluar vulnerabilidades, servicios y configuraciones de manera controlada, reduciendo riesgos para la infraestructura general.

6.8.4. Comparación entre topologías

A continuación se presenta un cuadro comparativo entre una topología de red por defecto y la topología mínima de seguridad propuesta.

El objetivo es evidenciar las mejoras obtenidas en términos de aislamiento, control de tráfico y reducción de riesgos.

Aspecto	Topología por defecto	Red mínima propuesta
Estructura general	Un único segmento de red donde todos los dispositivos comparten el mismo dominio de difusión.	Dos segmentos de red diferenciados: un nivel superior general y una subred segmentada para entornos aislados.
Aislamiento de dispositivos	Inexistente; los equipos pueden comunicarse entre sí sin restricciones.	Elevado; el router DD-WRT actúa como frontera lógica y de seguridad entre las redes.
Superficie de ataque	Amplia; un dispositivo comprometido puede afectar al resto de los equipos.	Reducida; los dispositivos experimentales quedan contenidos dentro de una subred aislada.
Control de tráfico	Limitado a las funciones básicas del router principal.	Avanzado; DD-WRT permite reglas de cortafuegos, NAT, monitoreo y segmentación del tráfico.
Gestión y monitoreo	Escaso o inexistente; no se dispone de control granular ni registros detallados.	Centralizado en DD-WRT; posibilita registro de actividad, control de ancho de banda y acceso remoto seguro.
Seguridad global	Dependiente de configuraciones básicas de enrutamiento y contraseñas.	Basada en el principio de defensa en profundidad, con segmentación, autenticación reforzada y control de acceso selectivo.

Tabla 6.4 Comparación entre topología por defecto y red mínima de seguridad propuesta

Capítulo 7

Modelo de amenazas

En esta solución propongo un modelo de amenazas orientado a evaluar y validar los controles del ciclo de vida seguro del *firmware* de IoT (repositorio *GitHub, GitHub Actions, SonarCloud, DefectDojo y clúster Kubernetes*) con enfoque en los mencionados *supply-chain attack, bricking*, ataques de fuerza bruta y ataques *DDoS*.

7.1. Activos críticos

Categoría	Descripción del activo
Código fuente	Repositorio en *GitHub* con el código del *firmware* en C++, configuraciones y dependencias platformio.ini.
Artefactos generados	Binarios compilados (`.bin`), hashes de integridad y versiones publicadas en releases.
Gestión de secretos	Claves para firma OTA, certificados TLS, *tokens de GitHub Actions y secrets de Kubernetes*.
Pipeline CI/CD	*Runners*, imágenes y contenedores utilizados por *GitHub Actions* para la automatización de compilaciones y auditorías.
Dispositivo final	Hardware ESP8266/ESP32 y su bootloader, incluyendo mecanismos de arranque seguro.
Plataforma IoT	*Broker MQTT y ThingsBoard*, responsables del intercambio y visualización de telemetría.
Infraestructura	Componentes de *Kubernetes: Nginx Ingress, Cert-Manager, PostgreSQL, Redis y Actions Runner Controller*.
Monitoreo y auditoría	Registro e informe de *SonarCloud, DefectDojo*.

Tabla 7.1 Activos críticos dentro del ciclo de vida seguro del *firmware* de IoT

7.1.1. Roles y actores fuente de amenazas

Se consideran los siguientes actores como fuentes de amenaza relevantes, ordenados desde el entorno físico hasta los más remotos:

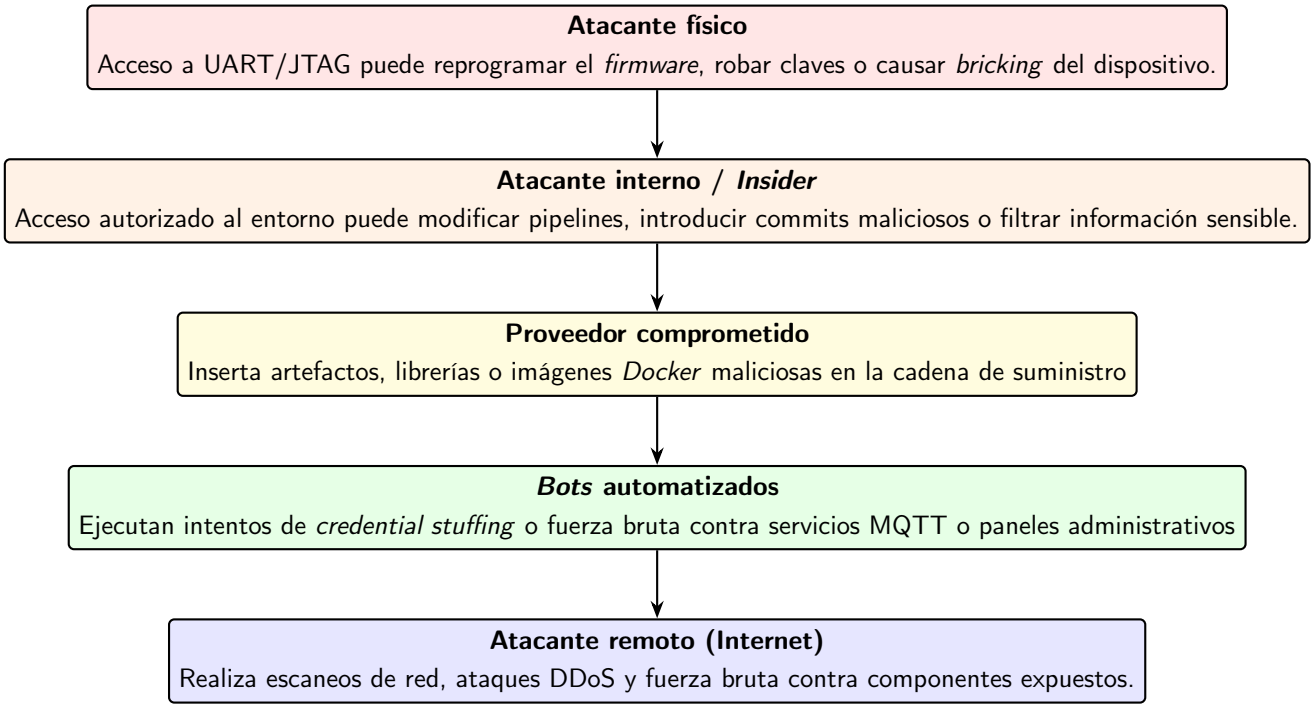


Figura 7.1 Jerarquía descendente de actores de amenaza.

7.1.2. Vectores y puntos de entrada principales

Los vectores y puntos de entrada:

Superficie de ataque	**Descripción del vector**	**Posibles amenazas asociadas**
Puertos físicos (UART / JTAG)	Interfaces de depuración expuestas en la placa que permiten acceso directo al bootloader y a la consola.	Tampering por acceso físico, extracción parcial de información sensible, manipulación del arranque o carga no autorizada de firmware.
Puertos expuestos en red	Servicios de comunicación como MQTT sin cifrado o Ingress Controller mal configurado en Kubernetes.	Ataques de denegación de servicio (DoS), interceptación de datos, secuestro de sesión o *man-in-the-middle*.
Endpoint OTA	Servidor HTTPS o repositorio donde se alojan las actualizaciones de firmware (*Over-The-Air*).	Inyección de firmware malicioso, ataque de *bricking*, interceptación de tráfico si no se usa TLS.
Dependencias de terceros	Paquetes de PlatformIO, módulos `pip` y librerías C++ externas utilizadas en la compilación del firmware.	Inclusión de código comprometido, vulnerabilidades en componentes, pérdida de integridad del firmware.
Push / Pull Request en GitHub	Interacción con el repositorio mediante commits o PRs; ejecución de *workflows* en GitHub Actions.	Inyección de código malicioso, manipulación del pipeline CI/CD, *Supply Chain Attack*.
Runners de Actions mal configurados	Ejecución de *self-hosted runners* o instancias del Actions Runner Controller (ARC) sin aislamiento adecuado.	Escalada de privilegios, acceso no autorizado a secretos, ejecución arbitraria de comandos.

Tabla 7.2 Superficies de ataque identificadas en el ecosistema

7.1.3. Mapeo *STRIDE*

STRIDE es un modelo de modelado de amenazas que clasifica los posibles ataques en seis categorías: Suplantación de identidad (*Spoofing*), Alteración de datos (*Tampering*), Repudio (*Repudiation*), Divulgación de información (*Information Disclosure*), Denegación de servicio (Denial of Service) y Elevación de privilegios (*Elevation of Privilege*). Desarrollado por *Microsoft*, este acrónimo es una herramienta mnemotécnica para ayudar a los equipos de desarrollo a identificar y mitigar riesgos de seguridad en software y sistemas.

Aplicación de *STRIDE* para identificar amenazas:

- ***Spoofing:*** Credenciales por defecto o claves en texto claro.
- ***Tampering:*** Acceso físico al dispositivo. OTA sin firma o modificación de artefactos.
- ***Repudiation:*** Falta de registro de *builds o builds* no reproducibles.
- ***Information disclosure*:** MQTT / HTTP sin cifrado, secretos expuestos.
- ***Denial of Service:*** DDoS sobre el dispositivo o al *broker*.
- ***Elevation of Privilege:*** Interfaces de depuración sin protección.

7.2. Árboles de ataque

Presento los árboles de ataque a alto nivel:

***Supply-Chain Attack* (alto nivel)**

- **Objetivo:** Entregar *firmware* malicioso al dispositivo.
- Vectores:
  - Comprometer el repositorio GitHub e inyectar un commit que pase pipelines.
  - Comprometer un *runner* de *GitHub Actions (self-hosted)* para ejecutar *builds* con artefactos maliciosos.
  - Comprometer una dependencia (PlatformIO / pip) e introducir código malicioso en el *build*.
  - Comprometer el *storage* / servidor OTA y sustituir el binario.
- **Resultado posible:** *Bricking* del equipo, instalación de backdoor, exfiltración de datos.

***Bricking* intencional**

- **Objetivo:** Dejar el dispositivo inutilizable.
- Vectores:
  - Enviar una OTA corrupta sin mecanismo de fallback.
  - *Flasheo* físico sin posibilidad de recuperación (sobrescribir bootloader).
  - Ataque al bootloader si no hay verificación de firma.
- **Resultado posible:** Pérdida de servicio y garantía.

Fuerza bruta en management / MQTT

- **Objetivo:** Obtener acceso administrativo o secuestrar sesiones.
- Técnicas: *Credential stuffing*, intentos de autenticación masivos contra el *broker* o panel administrativo.
- **Resultado posible:** Acceso a *topics* sensibles, reconfiguración remota del dispositivo.

DDoS

- **Objetivo:** Agotar CPU, recursos del SO o saturar la red.
- Técnicas: *Flood* sobre *endpoint* OTA, *broker* MQTT o enviar conexiones masivas al ESP.
- **Resultado posible:** Denegación de servicio del dispositivo o servicios asociados.

Capítulo 8

Casos prácticos

8.1. Caso A — Escaneo de superficie y descubrimiento

Objetivo: Descubrir puertos abiertos, servicios y versiones.
Precondición: Dispositivo en red de pruebas.
Herramientas: *Nmap, Gobuster,* herramienta de inspección de navegador web *Wappalizer, Shodan*.

Comandos:

```
nmap -sS -sV -p- --script vuln  192.168.8.140
gobuster dir -u http://192.168.8.140 -w /usr/share/seclists/Discovery/Web-Content/directory
Panel de detección de tecnologías \textit{Wappalizer}
Panel de detección de puertos y servicios \textit{Shodan}
```

Criterio *FAIL*: Puertos de depuración expuestos, versiones vulnerables.
Evidencia: Salida de *nmap*, salida de *gobuster*, capturas de pantalla, Wappalizer, así como el registro temporal de la consulta en Shodan.

> **Nota**
>
> Ver Apéndice - Evidencias

8.2. Caso B — Captura y análisis de tráfico *(confidentiality/MITM)*

Objetivo: Comprobar si el tráfico viaja en texto plano y si posee TLS .
Herramientas: *Wireshark*.

Comando:

```
Filtrar en\textit{ Wireshark:  http.request || tls.handshake}
```

Criterio *FAIL*: Credenciales o payloads en texto plano; ausencia de verificación de certificado.
Evidencia: Filtros aplicados y capturas. Aplicación de certificados.

> Nota
>
> Ver Apéndice - Evidencias

8.3. Caso C — Prueba OTA maliciosa / *fallback recovery (bricking)*

Objetivo: Validar que la OTA requiere firma y que existe mecanismo de *rollback/fallback*.
Precondición: Entorno de pruebas con copia de seguridad.

Pasos (laboratorio controlado):

1. Generar *firmware* legítimo firmado (firma A).
2. Generar *firmware* corrupto o firmado con una llave distinta.
3. Intentar la OTA con el *firmware* corrupto.

Criterio *OK*: El dispositivo rechaza binarios sin firma válida; si falla el flasheo, existe *rollback* a la versión anterior.
Evidencia: Logs de la consola serie del dispositivo, logs del servidor OTA, hash del *firmware* recibido.

> Nota sobre el alcance de la prueba
>
> Ver Apéndice - Evidencias

8.4. Caso D — Compromiso del *pipeline* (*supply chain*/repositorio)

Objetivo: Comprobar la resiliencia ante *PR/commit* malicioso o *runner* comprometido.
Precondición: *Pipeline de GitHub Actions* con secretos, ARC *self-hosted runner* desplegado en Kubernetes.

Pruebas: Simulación de un *pull request* que introduce código defectuoso o potencialmente malicioso.

Detección esperada: Validación mediante logs de *GitHub Actions*, análisis de integridad de artefactos, y verificación de firmas o hashes.

Criterio *FAIL:* Que el *pipeline* produzca un artefacto sin revisión adecuada, acepte cambios inseguros o exponga secretos durante la ejecución.

Evidencia: Logs de ejecución, *diffs de commits/PR,* fallas detectadas por SAST, verificación de *hashes*, bloqueos de releases y reportes de análisis estático.

Nota

Ver Apéndice - Evidencias

8.5. Caso E — Fuerza bruta / panel administrativo

Objetivo: Comprobar mecanismos de autenticación y detección de intentos repetidos.
Herramientas: *Hydra*.
Ejemplo:

```
hydra -L users.txt -P rockyou.txt -s 80 -f 192.168.8.145
```

Criterio *FAIL*: Credenciales débiles aceptadas.
Evidencia: *Logs del broker*, ausencia de alertas y captura de intentos.
Mitigación: Bloqueo por intentos, *rate limiting*, mTLS, contraseñas fuertes y rotación de credenciales.

Nota

Ver Apéndice - Evidencias

8.6. Caso F — DDoS sobre el dispositivo

Objetivo: Probar límites y vulnerabilidades (*rate limiting*, filtrado en *Ingress*).
Herramientas: `wrk`, `slowhttptest`.
Criterio *FAIL*: El dispositivo deja de responder, el *broker* supera el número máximo de conexiones y no existen alertas/*autoscaling*.
Evidencia: Capturas del ataque y la caída del servicio.
Mitigación: QoS, *rate limiting*, filtrado en Ingress.

Nota

Ver Apéndice - Evidencias

8.7. Caso G — Acceso físico (UART/JTAG)

Objetivo: Validar que los puertos físicos no permiten extracción de credenciales ni flasheo sin autorización.
Pasos:

1. Conectar a UART y revisar la consola y logs del bootloader. Intentar interrumpir el arranque y obtener consola de recuperación.

Criterio *FAIL*: Es posible extraer claves o flashear sin autenticación.
Mitigación (*tampering* / acceso físico): Dado que el entorno no cuenta con mecanismos de seguridad adicionales (eFuses, Secure Boot, cifrado de flash), la mitigación aplicable es la **protección física del dispositivo** para prevenir el tampering. Esto debería incluir:

- Encapsular o sellar la placa para impedir el acceso directo a pines UART/JTAG.
- Ubicar los equipos en gabinetes cerrados con control de acceso.
- Emplear etiquetas de tamper-evident o precintos cuando aplique.

Nota sobre el alcance de la prueba

Se presenta este caso debido a su importancia y criticidad, aunque la ejecución no resulta posible dentro del entorno experimental definido. Ver limitaciones especificadas.

8.8. *Mapping* entre la matriz de riesgos y los casos de prueba

Vulnerabilidad (tabla)	Caso de prueba (referencia)
UART sin protección	Caso G (acceso físico)
OTA sin firma	Caso C (OTA maliciosa + *rollback*)
Credenciales por defecto	Caso E *(brute force)* + *secret scanning*
Protocolos inseguros	Caso B *(Wireshark)* + *nmap*
Limitaciones CPU/mem	Caso F *(DDoS resilience)*
Supply-chain attack	Caso D (compromiso *pipeline* + *checks SBOM*)
Bricking	Caso C + verificación de *fallback*
DoS	Caso F
Broker MQTT expuesto	Caso A/B/E
Ausencia de *Secure Boot*	Caso C/G

8.9. Mitigación completa

La mitigación de los riesgos identificados se sustenta con la presente aplicación DevSecOps que incorpora controles técnicos, verificaciones automatizadas y validaciones manuales a lo largo de todo el ciclo de vida del firmware. La figura 8.1 del diagrama de secuencia ilustra cómo cada etapa del pipeline interviene para reducir la superficie de ataque, prevenir introducción de vulnerabilidades y garantizar que únicamente artefactos verificados alcancen el dispositivo IoT en producción.

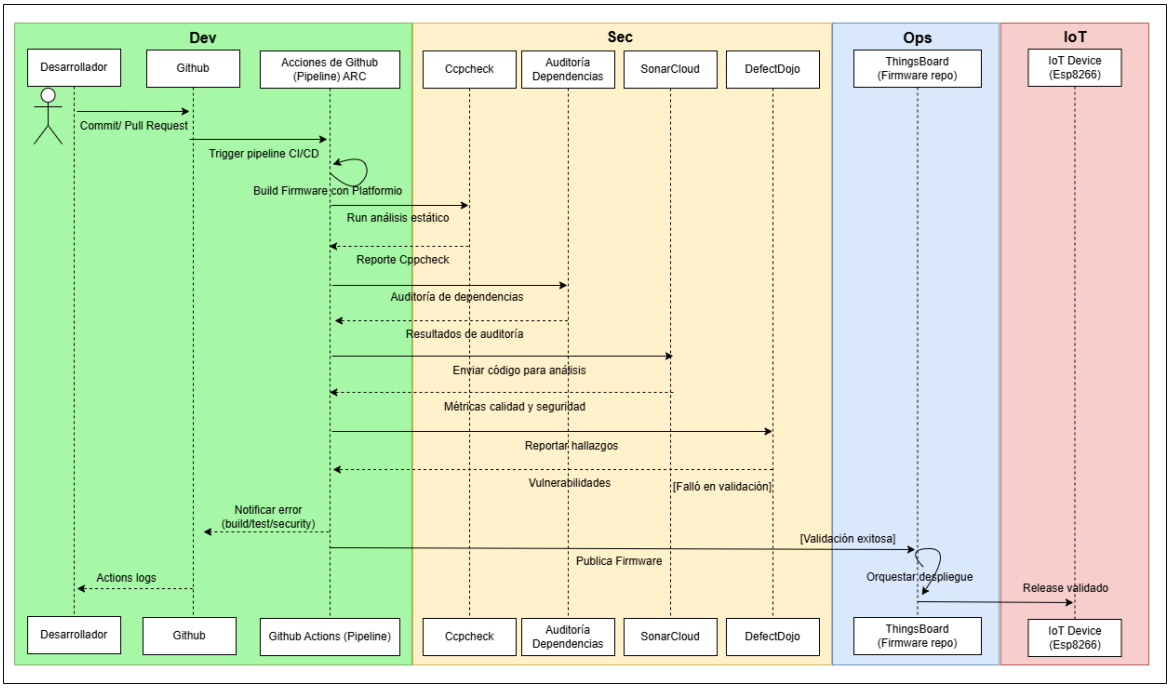


Figura 8.1 Diagrama de secuencia con aplicación DevSecOps

Explicación de los componentes del diagrama

- **Dev:** El desarrollador realiza un *commit o pull request en GitHub*, lo que activa el pipeline CI/CD mediante GitHub Actions y ARC en Kubernetes. Durante esta fase se ejecuta la compilación del firmware con PlatformIO y se inicia el análisis estático. De manera paralela, se disparan también las auditorías de dependencias, cuyos resultados se devuelven automáticamente al *pipeline*.

- **Sec:** Las herramientas de seguridad —Cppcheck, auditoría de dependencias, *SonarCloud y DefectDojo*— reciben el código para su evaluación, generando métricas de calidad, seguridad y mantenibilidad. Cuando alguno de estos sistemas detecta fallos críticos, malas prácticas o vulnerabilidades, el *pipeline* se bloquea automáticamente y se notifica al desarrollador. Una vez corregidos los hallazgos, la validación continúa y el *pipeline* avanza.

- **Ops:** Si todas las verificaciones anteriores son satisfactorias, la fase de operaciones procede a orquestar el despliegue utilizando el repositorio de *firmware* de *ThingsBoard*. Solo se autorizan artefactos que hayan superado exitosamente las etapas de análisis, pruebas y seguridad.

- **IoT:** El dispositivo IoT únicamente recibe releases validados y publicados mediante el flujo seguro, garantizando que cada *firmware* instalado ha sido auditado, construido y supervisado en todo el ciclo DevSecOps.

Este proceso integrado asegura que cada nueva versión del *firmware* atraviese controles rigurosos de calidad, seguridad y trazabilidad antes de llegar al dispositivo final, fortaleciendo la postura de seguridad del ecosistema completo y reduciendo riesgos tanto en desarrollo como en operación.

1. Mitigaciones en la superficie de ataque y servicios expuestos (Casos A y B)

Los experimentos de descubrimiento, escaneo de puertos y captura de tráfico expusieron servicios sin cifrado, protocolos en texto plano y puertos innecesarios accesibles. Para reducir esta superficie de exposición se implementan los siguientes controles:

- **Cifrado extremo a extremo:** Uso obligatorio de mTLS en MQTT, incluyendo verificación estricta de certificados y rechazo de conexiones inseguras desde *ThingsBoard*.
- **Política de puertos mínimos:** Solo se habilitan aquellos estrictamente necesarios para operación.
- **Fortificación de credenciales:** Eliminación de credenciales por defecto y almacenamiento seguro.
- **Validación automatizada en CI/CD:** El *pipeline* ejecuta detecciones de protocolos inseguros (análisis *SonarCloud*, inspección automatizada), bloqueando *releases* con hallazgos críticos.

Estas medidas reducen el riesgo de exposición accidental, ataques MITM, fuga de credenciales y movimientos laterales en la red.

2. Mitigaciones en integridad del *firmware* y actualización OTA (Caso C)

Las pruebas de actualización maliciosa [16] demuestran la importancia de verificar la autenticidad e integridad del *firmware*, así como la disponibilidad de mecanismos de recuperación ante fallos. Las mitigaciones recomendadas son:

- **Firmado criptográfico obligatorio:** Cada binario debe ser firmado en la etapa de *build* del pipeline, usando claves protegidas en *GitHub Actions* y auditables.
- **Verificación de firma en el dispositivo:** El *bootloader* del ESP8266 debe validar la firma antes de aplicar la actualización y descartar binarios corruptos o no autorizados.
- ***Rollback* automático:** Implementación de *fallback* hacia la última versión estable cuando la OTA falle o se detecte integridad comprometida.
- **Almacenamiento seguro del *bootloader* y claves:** Restricción de flasheo no autorizado y resguardo de la cadena de confianza.

Estas medidas evitan el *bricking*, la introducción de *backdoors* y ataques de manipulación de *firmware*.

3. Mitigaciones frente a *supply-chain attack* (Caso D)

La simulación de commits maliciosos, runners comprometidos y alteración de dependencias evidenció la necesidad de controles dentro del *pipeline*. Las mitigaciones incluyen:

- **Validación de integridad de artefactos:** Cálculo y verificación de hashes, firma de binarios y verificación cruzada en *DefectDojo*.
- **Control de dependencias (SBOM):** Auditoría automatizada de librerías (seguridad de PlatformIO, pip-audit, Cppcheck, análisis de CVEs).
- ***Runners* aislados:** Uso de *Actions Runner Controller* dentro de *Kubernetes* para segregar contextos y reducir superficie de compromiso.
- **Revisión obligatoria de PRs:** Ningún cambio impacta en producción sin revisión de seguridad y aprobación por pares.
- **Alertas y auditoría:** Análisis de logs de *GitHub Actions* y bloqueo automático ante modificaciones no autorizadas en archivos sensibles.

Estas medidas mitigan *supply-chain attacks*, inyección de código y vulnerabilidades introducidas de forma inadvertida o maliciosa.

4. Mitigación de ataques de fuerza bruta y autenticación débil (Caso E)

Los experimentos de fuerza bruta confirmaron ausencia de controles de protección en la autenticación. Para reducir este riesgo:

- Contraseñas robustas.
- Autenticación con certificados (mTLS): Sustituir credenciales débiles por mecanismos basados en claves.
- Segmentación de redes, aplicación de reglas de *firewalls*, filtrado web.

Estas medidas evitan accesos no autorizados mediante automatización y fortalecen la seguridad del *broker*.

5. Resiliencia ante condiciones de sobrecarga y *DDoS* (Caso F)

Las pruebas con `slowhttptest` y `wrk` confirmaron que el dispositivo y el broker pueden saturarse bajo carga anómala. Propuestas:

- Limitación de tasa (*rate limiting*) en *Ingress y broker MQTT*.

- QoS y control de mensajes duplicados.
- Restricción de conexiones simultáneas.
- Monitoreo en *Kubernetes*. Autoescalado, alertas de uso de CPU/memoria y políticas de recuperación.
- Bloqueos en *firewall*, reglas de `iptables` y control de tráfico en el borde de red.

8.10. Inferencia

8.10.1. Integración y mitigación continua DevSecOps

Todas las medidas anteriores se integran dentro del flujo automatizado representado en el diagrama 8.1, donde cada etapa participa en la validación del *firmware* y en la reducción de riesgos desde el origen del pipeline.

Capítulo 9

Resultados de los experimentos

9.1. Resultados cuantitativos

9.1.1. Riesgos residuales de red

Se presentan los resultados cuantitativos del análisis de riesgos, incluyendo los valores de probabilidad, impacto, puntajes iniciales y puntajes residuales obtenidos tras aplicar las medidas de mitigación.

ID	Riesgo	P Inicial	I Inicial	Puntaje Inicial	Puntaje Residual
NET-03	Segmento único sin aislamiento que facilita el movimiento lateral.	4	4	16	6
NET-04	Router sin *firewall* avanzado ni registro de tráfico.	3	4	12	4
NET-05	Credenciales débiles o predeterminadas en equipos de red.	5	4	20	6
NET-06	*Firmware* sin actualizar, expuesto a vulnerabilidades (CVE) conocidas.	4	4	16	6

Tabla 9.1 Comparación de riesgo inicial vs residual tras aplicar la red mínima segura

9.1.2. Riesgos residuales del dispositivo

Se presentan los resultados cuantitativos del análisis de riesgos, incluyendo los valores de probabilidad, impacto, puntajes iniciales y puntajes residuales obtenidos tras aplicar las medidas de mitigación.

ID	Riesgo	P Inicial	I Inicial	Puntaje Inicial	Puntaje Residual
IOT-01	Extracción de credenciales, reprogramación no autorizada, acceso físico malicioso	4	4	16	16
FMW-01	Inyección de firmware malicioso, bloqueo irreversible del dispositivo *(bricking)*	3	5	15	6
SEC-01	Acceso remoto no autorizado, escalamiento de privilegios	4	5	20	6
NET-01	Espionaje, inyección de tráfico, suplantación de dispositivo	3	4	12	4
IOT-02	Degradación o fallo por limitación de CPU/memoria	3	3	9	4
FMW-02	Inserción de firmware o hardware comprometido *(supply-chain attack)*	3	5	15	6
FMW-03	*Bricking* intencional y pérdida total de disponibilidad	3	4	12	6
SEC-02	Caída del sistema por ataques de denegación de servicio	3	3	9	4
NET-02	Intercepción o secuestro de sesiones MQTT por mala configuración	4	4	20	6
FMW-04	Ejecución de *firmware* alterado por la ausencia de Secure Boot	3	5	15	15

Tabla 9.2 Comparación de riesgo inicial vs residual tras aplicar mitigaciones en dispositivo

ID	Riesgo	P Inicial	I Inicial	Puntaje Inicial	Puntaje Residual
FMW-05	Dependencias externas no verificadas (PlatformIO/pip) — riesgo de *supply-chain*.	3	5	15	4
CI-01	PRs maliciosos o commits comprometidos que manipulan el repositorio.	5	5	25	6
CI-02	*Runners self-hosted* sin aislamiento, exposición de secretos o ejecución arbitraria.	3	5	15	4
SEC-03	Credenciales débiles, tokens expuestos o secretos sin rotación.	5	4	20	6
SEC-04	Falta de registros de auditoría, builds no trazables o falta de evidencia.	2	3	6	2

Tabla 9.3 Comparación de riesgo inicial vs residual tras aplicar controles DevSecOps

Capítulo 10

Conclusiones

El presente trabajo permitió demostrar que la integración de prácticas *DevSecOps* en el ciclo de vida de dispositivos IoT de recursos limitados —como los ESP8266 y ESP32 utilizados en el estudio— constituye una estrategia viable y certera para reducir brechas de seguridad, mejorar la trazabilidad del software y fortalecer la resiliencia frente a ataques emergentes. A partir del relevamiento realizado, se identificaron vulnerabilidades recurrentes asociadas a configuraciones por defecto, protocolos no cifrados, gestión débil de credenciales, dependencias de terceros, ausencia de mecanismos de actualización seguros y riesgos en la cadena de suministro. Los análisis de riesgos y el modelo de amenazas desarrollado —incluyendo actores, superficies de ataque, mapeo *STRIDE* y árboles de ataque— permitió estructurar estas debilidades y priorizarlas según su impacto potencial. Esta base analítica fue importante para diseñar los escenarios prácticos de prueba, que evidenciaron la factibilidad real de ataques como *MITM*, fuerza bruta, *DDoS*, *bricking* y *supply chain attack*. La implementación de *pipelines DevSecOps* adaptados al contexto IoT, incluyendo análisis estático con *SonarCloud*, auditorías de dependencias, escaneos automáticos, control de integridad, gestión centralizada de vulnerabilidades con *DefectDojo* y validación continua de artefactos, permitió cuantificar mejoras sustanciales en la detección temprana de fallas y en la reducción de la superficie de ataque. Estas prácticas aportaron trazabilidad, predecibilidad y control sobre la cadena de construcción del *firmware*, mitigando riesgos críticos. Los casos prácticos ejecutados en entorno controlado validaron empíricamente las amenazas identificadas, confirmando que la aplicación de controles de seguridad —incluyendo cifrado *TLS*, autenticación robusta, aislamiento de *runners*, monitoreo continuo, segmentación de redes y mecanismos de *fallback* para *OTA*— reduce significativamente la exposición del dispositivo. Aunque ciertas limitaciones del proyecto, como la ausencia de blindaje físico restringen el alcance experimental, los resultados obtenidos aportan evidencia sólida del impacto positivo de integrar seguridad de manera diligente y automatizada.

Capítulo 11

Problemas encontrados

Durante el desarrollo de esta investigación se presentaron diversos desafíos técnicos y limitaciones de recursos que condicionaron el alcance experimental del trabajo:

1. **Imposibilidad de implementar medidas de protección física:** La ausencia de mecanismos contra accesos no autorizados a los dispositivos ESP32 y ESP8266 constituyó una limitación importante, dado que soluciones como encapsulados antimanipulación o sellado epóxico requerían recursos económicos adicionales no disponibles.

2. **Restricciones de hardware en la infraestructura de orquestación:** Inicialmente se implementó *Rancher* para la gestión de clústeres *Kubernetes*, pero el consumo elevado de memoria RAM causó inestabilidad en el equipo de pruebas. Se evaluó posteriormente *Portainer* como alternativa, presentando limitaciones similares. Finalmente, se adoptó *Minikube* una solución más liviana que permitió desplegar todos los *pods* necesarios, aunque sacrificando funcionalidades avanzadas de administración visual y monitoreo centralizado.

3. **Curva de aprendizaje pronunciada:** La investigación exigió adquirir conocimientos profundos sobre arquitecturas IoT, protocolos de comunicación como *MQTT* y prácticas *DevSecOps*, áreas en las cuales no se contaba con experiencia previa sustancial.

4. **Restricciones en la selección de protocolos:** Durante la evaluación de protocolos se consideró la adopción de alternativas más robustas y modernas como *Zigbee* o *Matter* con *Thread*. Sin embargo, estos protocolos requieren hardware especializado —como *coordinadores*, *border routers* y chips compatibles con IEEE 802.15.4— que implicaban costos adicionales y estaban orientados a casos de uso específicos como domótica avanzada, redes *mesh* de bajo consumo energético y ecosistemas de hogar inteligente interoperables, que excedían el propósito general de este estudio centrado en dispositivos Wi-Fi de bajo costo.

Capítulo 12

Futuras líneas de investigación y mejora

- **Extensión de la automatización del pipeline:** Incorporar análisis de otras integraciones como *Semgrep* (detección de patrones inseguros específicos en código C/C++), *GitGuardian*, o *Gitleaks* (detección de secretos y credenciales expuestas en repositorios), *Grype* (análisis de vulnerabilidades en binarios compilados), y *OWASP Dependency-Check* (identificación de bibliotecas con *CVEs* conocidos), u optimizar las existentes para mejorar la cobertura de seguridad y eficiencia del proceso *DevSecOps*.

- **Integración de arquitecturas Matter y Thread mediante gateway:** Evaluar la incorporación de un *gateway* que permita interoperar el ecosistema actual basado en *MQTT* con dispositivos compatibles con *Matter* y *Thread*. Esta integración habilita el aprovechamiento de las ventajas técnicas propias de estos estándares, como una conectividad más estable, topologías malladas (*mesh*) con autocuración y un menor consumo energético gracias al uso de *Thread* como red de baja potencia. Asimismo, *Matter* proporciona un marco de comunicación seguro por diseño, con autenticación nativa, actualización confiable y gestión unificada de dispositivos, reduciendo la complejidad de interoperabilidad entre fabricantes. Estratégicamente, la adopción de Matter y Thread está experimentando un crecimiento sostenido en el mercado global, impulsado por fabricantes de dispositivos de hogar inteligente y por la tendencia hacia ecosistemas interoperables. Integrar estos estándares mediante un *gateway* permitiría ampliar la compatibilidad del sistema, mejorar la eficiencia operativa y evaluar su impacto real en el diseño, operación y ciclo de vida del *firmware*, contribuyendo al fortalecimiento del modelo de seguridad y resiliencia del dispositivo IoT.

- **Integración de métricas de sostenibilidad y eficiencia energética:** Evaluar el impacto energético de los controles de seguridad integrados, especialmente en dispositivos con recursos extremadamente limitados. Esto permitiría estudiar la relación entre robustez criptográfica, nivel de protección alcanzado y aumento del consumo energético, orientando decisiones de diseño que busquen el equilibrio entre seguridad y eficiencia.

- **Aplicación de inteligencia artificial para auto-adaptación:** Implementar técnicas de IA para la evolución autónoma del *firmware* y del *pipeline* de seguridad. Mediante modelos generativos y sistemas de aprendizaje continuo sería posible crear mecanismos de autoendurecimiento, donde el *firmware* se actualice automáticamente tras detectar nuevas amenazas o patrones de explotación. A nivel de *pipeline*, la IA podría analizar miles de ejecuciones históricas del proceso *DevSecOps* para optimizar tiempos, predecir fallos de compilación, ajustar parámetros de análisis y priorizar vulnerabilidades según su impacto real. Estas capacidades darían lugar a un ecosistema IoT con mayor autonomía, resiliencia y capacidad de respuesta frente a amenazas en constante evolución.

Bibliografía

[1] Fortinet. *Definición y explicación del Internet de las Cosas (IoT)*. Inf. téc. Fortinet, 2025. URL: https://www.fortinet.com/lat/resources/cyberglossary/iot.

[2] IEEE Internet of Things Initiative. *Towards a Definition of the Internet of Things*. Inf. téc. IEEE, 2015. URL: https://iot.ieee.org/images/files/pdf/IEEE_IoT_Towards_Definition_Internet_of_Things_Revision1_27MAY15.pdf.

[3] Red Hat. *¿Qué es DevSecOps?* Actualizado 15 de marzo de 2023. 2023. URL: https://www.redhat.com/es/topics/devops/what-is-devsecops?sc_cid=7015Y0000048RsyQAE&gad_source=1&gclid=CjwKCAjwiaa2BhAiEiwAQBgyHtX6qv7vvPwfcKX68EeYjgyWIxdjufzGS4_eEoqpoEoGyrcbRhEc0BoCGTEQAvD_BwE.

[4] Satyajit Sinha. *State of IoT 2023: Number of connected IoT devices growing 16 to 16.7 billion globally*. Consultado el 24 de agosto de 2024. 2023. URL: https://iot-analytics.com/number-connected-iot-devices/.

[5] Ipsos MORI. *Consumer attitudes towards IoT security: Research report*. Inf. téc. Department for Digital, Culture, Media y Sport (DCMS), 2021. URL: https://assets.publishing.service.gov.uk/government/uploads/system/uploads/attachment_data/file/978685/Consumer_Attitudes_Towards_IoT_Security_-_Research_Report.pdf.

[6] Checkpoint. *The tipping point: Exploring the surge in IoT cyberattacks plaguing the education sector*. Accedido: 23 de Junio de 2024. 2023. URL: https://blog.checkpoint.com/security/the-tipping-point-exploring-the-surge-in-iot-cyberattacks-plaguing-the-education-sector/.

[7] Ricardo Brea. «Hub Of Things: concentrador para internet de las cosas». En: *XXIV Congreso Argentino de Ciencias de la Computación (CACIC 2018)*. La Plata, Argentina: Red de Universidades con Carreras en Informática (RedUNCI), oct. de 2018, págs. 673-682. ISBN: 978-950-658-472-6. URL: http://sedici.unlp.edu.ar/handle/10915/73296.

[8] Natasha Tusikov. «Regulation through "bricking": private ordering in the "Internet of Things"». En: *Internet Policy Review* 8.2 (2019). DOI: 10.14763/2019.2.1405. URL: http://policyreview.info/articles/analysis/regulation-through-bricking-private-ordering-internet-things.

[9] Sectigo. *Evolution of IoT Attacks Report*. Consultado el 24 de enero de 2024. 2024. URL: https://www.sectigo.com/download-content?target=resource-library/sectigo-evolution-of-iot-attacks-report&utm_campaign=iot-attacks-study.

[10] Council of the European Union. *General Data Protection Regulation (GDPR)*. Accessed: 2024-08-16. 2016. URL: https://www.consilium.europa.eu/en/policies/data-protection/data-protection-regulation/.

[11] European Telecommunications Standards Institute (ETSI). *Cyber Security for Consumer Internet of Things: Baseline Requirements*. Inf. téc. EN 303 645 V2.1.1. Accessed: 2024-08-16. Sophia Antipolis, France: ETSI, 2020. URL: https://www.etsi.org/deliver/etsi_en/303600_303699/303645/02.01.01_60/en_303645v020101p.pdf.

[12] European Union Agency for Cybersecurity (ENISA). *ENISA Threat Landscape 2023*. ENISA, 2023. ISBN: 978-92-9204-588-3. DOI: 10.2824/782573.

[13] OWASP. *OWASP Internet of Things Project*. Consultado el 13 de febrero de 2024. 2024. URL: https://wiki.owasp.org/index.php/OWASP_Internet_of_Things_Project#tab=IoT_Vulnerabilities.

[14] OWASP. *OWASP Internet of Things Project*. Consultado el 13 de febrero de 2024. 2018. URL: https://wiki.owasp.org/index.php/OWASP_Internet_of_Things_Project#tab=OWASP_IoT_Top_10_2018_Mapping_Project.

[15] Per-Arne Jørgensen. «Security Operations (SecOps) and the Internet of Things (IoT): Introducing IoT Security Monitoring». Master in Applied Computer Science, Faculty of Computer Science, Engineering and Economics. Master's thesis. Halden, Norway: Østfold University College, jun. de 2023. URL: https://hiof.brage.unit.no/hiof-xmlui/bitstream/handle/11250/3089708/J%c3%b8rgensen_%20Per-Arne.PDF?sequence=1&isAllowed=y.

[16] Martin Acuña. *Implementación de prácticas DevOps para la integración y el despliegue continuos de software en sistemas IoT*. Inf. téc. Recuperado como referencia conceptual sobre prácticas DevOps aplicadas a entornos IoT. Documento técnico interno, Universidad Nacional de Hurlingham, 2025.

[17] B. Trammell y E. Kuhne. *Architectural Considerations in Smart Object Networking*. Internet Requests for Comments. RFC. Mar. de 2015. URL: https://datatracker.ietf.org/doc/html/rfc7452.

[18] C. Bormann, M. Ersue y A. Keranen. *Terminology for Constrained-Node Networks*. Request for Comments 7228, Internet Engineering Task Force. ISSN: 2070-1721. 2014. URL: https://datatracker.ietf.org/doc/html/rfc7228.

[19] AVG. *Firmware de seguridad para dispositivos*. Accedido el 25 de agosto de 2024. 2024. URL: https://www.avg.com/es/signal/firmware.

[20] C. Perkins y col. *Multicast Listener Report Message for IPv6 with MLDv2 Extensions*. Internet Requests for Comments. RFC. Abr. de 2021. URL: https://datatracker.ietf.org/doc/html/rfc9019.

[21] Chris M. Finen. *Communications and Protocol Basics*. Accedido: 2025-09-10. 2019. URL: https://ewh.ieee.org/r3/nashville/events/2019/IEEE%5C%20Meeting%5C%20Presentation%5C%20-%5C%20Communications%5C%20Basics%5C%202019-02-05.pdf.

[22] International Organization for Standardization. *ISO/IEC 7498-1:1994 - Information technology – Open Systems Interconnection – Basic Reference Model – Part 1: Overview*. Accedido: 2025-09-10. 1994. URL: https://www.iso.org/standard/20269.html.

[23] A. Farrel, D. King y S. Turner. *A Common YANG Data Model for Layer 2 and Layer 3 VPNs*. RFC 9431. 2023. URL: https://datatracker.ietf.org/doc/rfc9431/.

[24] M. Manasa Ekoramaradhya y C. Christina Thorpe. «A novel DevSecOps model for robust security in an MQTT Internet of Things». En: *Proceedings of the 17th International Conference on Cyber Warfare and Security*. 2022, págs. 63-71. DOI: 10.34190/iccws.17.1.31. URL: https://doi.org/10.34190/iccws.17.1.31.

[25] Eric Rescorla. *The Transport Layer Security (TLS) Protocol Version 1.3*. RFC 8446, Internet Engineering Task Force. Ago. de 2018. URL: %7Bhttps://datatracker.ietf.org/doc/html/rfc8446%7D.

[26] HiveMQ Team. *MQTT Security Fundamentals: TLS/SSL*. Accessed: 2024-10-04. 2024. URL: %7Bhttps://www.hivemq.com/blog/mqtt-security-fundamentals-tls-ssl/%7D.

[27] Jon Postel. *Transmission Control Protocol*. RFC 793. Internet Engineering Task Force (IETF), 1981. URL: https://datatracker.ietf.org/doc/html/rfc793.

[28] V. Cerf y J. Postel. *Internet Protocol*. RFC 791. Internet Engineering Task Force (IETF), 1981. URL: https://datatracker.ietf.org/doc/html/rfc791.

[29] *IEEE Standard for Low-Rate Wireless Networks*. Código ICS: 35.110 - Redes. Jul. de 2020. DOI: 10.1109/IEEESTD.2020.9144691. URL: https://ieeexplore.ieee.org/servlet/opac?punumber=9144689.

[30] Iberdrola. *Edge Computing*. Actualizado 24 de agosto de 2024. 2024. URL: https://www.iberdrola.com/innovacion/que-es-edge-computing#:~:text=QU%C3%89%20ES%20EL%20'EDGE%20COMPUTING,de%20ella%20como%20sea%20posible.

[31] J. Hong y col. *Desafíos y funciones del Internet de las cosas (IoT) Edge*. Request for Comments 9556. RFC 9556. abril de 2024. URL: https://datatracker.ietf.org/doc/html/rfc9556.

[32] Tope Omitola y Gary Wills. «Towards Mapping the Security Challenges of the Internet of Things (IoT) Supply Chain». En: *Procedia Computer Science* 126 (2018), págs. 441-450. DOI: 10.1016/j.procs.2018.07.278. URL: https://doi.org/10.1016/j.procs.2018.07.278.

[33] IoT Security Foundation. *Supply Chain Integrity Programme (SCIP): Guidance for IoT Supply Chain Security*. Inf. téc. London, United Kingdom: IoT Security Foundation, nov. de 2022. URL: https://www.iotsecurityfoundation.org.

[34] IBM. *¿Qué es DevOps?* IBM Think. Revisado 3 de octubre. s. f. URL: https://www.ibm.com/mx-es/think/topics/devops.

[35] Microsoft Azure. *What is DevOps?* Último acceso: 4 de octubre de 2024. 2024. URL: https://azure.microsoft.com/es-es/resources/cloud-computing-dictionary/what-is-devops/.

[36] Microsoft Learn. *What is DevOps?* Último acceso: 4 de octubre de 2024. 2024. URL: https://learn.microsoft.com/es-es/devops/what-is-devops#implement-devops-practices.

[37] Sten Pittet. *Continuous integration vs. delivery vs. deployment*. 2025. URL: https://www.atlassian.com/continuous-delivery/principles/continuous-integration-vs-delivery-vs-deployment.

[38] Amazon Web Services, Inc. *¿Qué es DevSecOps?* Accessed: 2024-08-24. 2024. URL: https://aws.amazon.com/es/what-is/devsecops/.

[39] Ranjan Singh. *Ensuring Robust Application Security: Exploring SAST, DAST, and IAST for Comprehensive Protection*. Accedido: 28 de agosto de 2025. 2023. URL: https://ranjaniitian.medium.com/ensuring-robust-application-security-exploring-sast-dast-and-iast-for-comprehensive-protection-d68b4cf5cf79.

[40] Ltd. Espressif Systems (Shanghai) Co. *ESP8266 Datasheet*. https://www.alldatasheet.es/datasheet-pdf/view/1148031/ESPRESSIF/ESP8266.html. Consultado el 26 de octubre de 2025. 2020.

[41] Ltd. Espressif Systems (Shanghai) Co. *ESP8266 Datasheet*. https://documentation.espressif.com/esp32-s3_datasheet_en.pdf. Consultado el 26 de octubre de 2025. 2020.

Apéndice A

Terminología

Acceso físico (UART/JTAG): Escenario donde el atacante interactúa directamente con pines de depuración del dispositivo, pudiendo leer la salida del bootloader, extraer información o manipular el *firmware*.

Actor de amenaza: Individuo, entidad o proceso que puede explotar una vulnerabilidad. Puede ser físico, interno, proveedor, bot automatizado o atacante remoto.

Activo crítico: Elemento del sistema cuyo compromiso afecta directamente la disponibilidad, integridad o confidencialidad de la solución (código fuente, artefactos, secretos, infraestructura, hardware).

Administración de cambios: Procedimiento para registrar, controlar y supervisar las modificaciones realizadas en el software, con el fin de prevenir vulnerabilidades derivadas de cambios no controlados.

Administración del cumplimiento: Conjunto de actividades destinadas a asegurar que el software cumpla con los estándares normativos, legales y de seguridad aplicables.

Aislamiento de runners: Práctica consistente en ejecutar los *runners* de CI/CD en entornos segregados con permisos mínimos, sin acceso directo a secretos persistentes ni a sistemas críticos, para reducir el impacto si un runner queda comprometido.

Amenaza: Evento, acción o actor con la capacidad de causar un impacto negativo sobre un activo.

Análisis de código: Proceso de revisión del código fuente para detectar vulnerabilidades, errores o desviaciones de las buenas prácticas de seguridad.

Análisis de riesgos: Proceso sistemático que identifica activos críticos, amenazas y vulnerabilidades, evalúa la probabilidad de ocurrencia y el impacto potencial, y prioriza medidas de mitigación según criterios de riesgo aceptados por la organización.

Artefactos: Productos generados por el proceso de construcción (binarios, librerías, imágenes) que se despliegan en dispositivos o entornos. Su integridad y procedencia deben ser verificadas antes del despliegue.

Ataques MITM (Man-in-the-Middle): Técnicas en las que un atacante intercepta o altera la comunicación entre dos partes, pudiendo espiar, modificar o redirigir mensajes si no existen mecanismos criptográficos y de autenticación adecuados.

Auditoría de dependencias: Evaluación sistemática de las bibliotecas y paquetes de terceros para detectar vulnerabilidades conocidas, licencias problemáticas o paquetes maliciosos, y mantener un inventario controlado (SBOM).

Autenticación robusta: Conjunto de mecanismos para verificar la identidad de usuarios o dispositivos más allá de una contraseña simple, incluyendo factores múltiples (algo que sabe, algo que tiene, algo que es) y métodos basados en claves o certificados que reducen el riesgo de suplantación.

Autocuración (self-healing): Capacidad del sistema para detectar anomalías o fallos y aplicar acciones correctivas automáticas (reinicios, conmutación a rutas alternativas, restauración de configuraciones) para recuperar la funcionalidad sin intervención humana.

Bootloader: Programa inicial que se ejecuta al encender el dispositivo y se encarga de verificar, cargar y transferir el control al firmware principal. Puede incluir funciones de verificación criptográfica, actualización segura y recuperación ante fallos.

Blindaje físico: Medidas de protección hardware (carcasas selladas, resinas, precintos, encapsulados) destinadas a dificultar o evidenciar el acceso físico y la manipulación de componentes sensibles del dispositivo.

Bricking: Estado en el cual el dispositivo queda inutilizable debido a firmware corrupto, bootloader sobrescrito o fallas en el arranque.

Broker MQTT: Componente responsable del intercambio de mensajes IoT. Ataques incluyen fuerza bruta, secuestro de sesión, MITM o saturación.

CI/CD (Continuous Integration / Continuous Deployment): Práctica de ingeniería de software que automatiza la integración de código y su despliegue, reduciendo errores humanos y acelerando la entrega de nuevas versiones.

Cifrado de flash: Proceso mediante el cual el contenido almacenado en la memoria flash del dispositivo es encriptado utilizando claves internas del microcontrolador. Protege firmware, configuraciones y secretos frente a extracción mediante acceso físico o ingeniería inversa.

Cifrado TLS: Protocolo criptográfico que proporciona confidencialidad, integridad y autenticación en las comunicaciones entre cliente y servidor. Utilizado para proteger conexiones (por ejemplo HTTPS, MQTT sobre TLS), impide la interceptación y manipulación de datos en tránsito.

Consumo energético: Parámetro que mide la energía requerida por un dispositivo para operar; relevante en IoT por su impacto en autonomía, dimensionamiento de baterías y en la viabilidad de ciertas técnicas criptográficas o protocolos.

Control de integridad: Mecanismos y procesos que garantizan que los artefactos y configuraciones no han sido alterados, típicamente mediante hashes, firmas digitales y comprobaciones automáticas en el pipeline y en el dispositivo.

Cppcheck: Herramienta de análisis estático que verifica la calidad del código C/C++ y detecta errores potenciales o desviaciones de estándares.

Cultura DevSecOps: Cambio organizacional que promueve la colaboración entre desarrolladores, operadores y especialistas en seguridad. Fomenta la comunicación, la responsabilidad compartida y la automatización de los procesos de seguridad.

DAST (Dynamic Application Security Testing): Herramientas que evalúan la seguridad de una aplicación en ejecución, simulando ataques externos para identificar posibles fallos de seguridad.

DefectDojo: Herramienta de gestión de vulnerabilidades que centraliza los resultados de análisis de seguridad y facilita el seguimiento de remediaciones.

Denegación de Servicio (DoS/DDoS): Ataques que saturan recursos de red o CPU para impedir el funcionamiento del dispositivo, broker MQTT o servicios de la infraestructura.

Dependencias externas: Librerías o paquetes utilizados durante la compilación (PlatformIO, `pip`, módulos C++). Su compromiso impacta directamente en la integridad del firmware.

Dev: Fase del proceso enfocada en la planificación, codificación y prueba de la aplicación.

DevSecOps: Integración de las prácticas de desarrollo (**Dev**), seguridad (**Sec**) y operaciones (**Ops**) en un ciclo de vida continuo. Su objetivo es incorporar la seguridad desde las etapas tempranas del desarrollo, manteniendo la eficiencia y la entrega continua del software.

Dual-bank: Mecanismo de diseño en dispositivos embebidos que utiliza dos particiones o bancos de firmware independientes (activo y de respaldo). Permite almacenar simultáneamente la versión actual del firmware y una versión alternativa o actualizada. Durante un proceso de actualización, el nuevo firmware se escribe en el banco inactivo y, una vez verificado, el sistema conmuta hacia él. Si ocurre un fallo en la actualización o el arranque, el dispositivo puede revertir automáticamente al banco anterior, garantizando alta disponibilidad, tolerancia a fallos y recuperación segura.

eFuses: Bits programables físicamente dentro del microcontrolador que permiten almacenar configuraciones de seguridad irreversibles. Se utilizan para habilitar Secure Boot, restringir depuración, fijar claves criptográficas o bloquear modos inseguros de arranque.

ESP32: Microcontrolador de la familia Espressif con conectividad Wi-Fi y Bluetooth integrada, mayor capacidad de CPU, memoria y periféricos que el ESP8266, y soporte para mecanismos de seguridad como Secure Boot y cifrado en hardware.

ESP8266: Microcontrolador de bajo consumo y costo, con conectividad Wi-Fi integrada, utilizado como plataforma base en proyectos IoT.

Formación en seguridad: Capacitación continua del personal técnico en prácticas seguras de desarrollo, despliegue y operación de software.

Fuerza bruta/Credential stuffing: Ataques que buscan obtener acceso mediante múltiples intentos de autenticación, utilizando contraseñas comunes o combinaciones filtradas.

GitHub Actions: Servicio de automatización que permite ejecutar flujos de trabajo (*pipelines*) para compilación, prueba y despliegue continuo.

Gates: Puntos de control obligatorios dentro del ciclo de desarrollo o del flujo de CI/CD, donde se evalúan criterios previamente definidos (seguridad, calidad, pruebas, cumplimiento, dependencias, diseño, entre otros) antes de permitir el avance a la siguiente etapa. Su función es prevenir que artefactos con errores, vulnerabilidades o requisitos incompletos progresen hacia fases críticas, garantizando un proceso de liberación más seguro y controlado.

Gateway: Nodo intermediario que conecta redes o protocolos distintos (por ejemplo Thread/Matter con MQTT o redes IP), realizando traducción de protocolos, control de acceso y funciones de seguridad perimetral.

Gestión centralizada de vulnerabilidades: Plataforma o proceso que consolida hallazgos de diversas herramientas, prioriza vulnerabilidades según riesgo y coordina el ciclo de remediación y seguimiento.

IAST (Interactive Application Security Testing): Método que combina elementos de SAST y DAST. Evalúa vulnerabilidades mientras la aplicación se ejecuta, utilizando agentes internos para monitorear su comportamiento.

Integración progresiva: Estrategia de adopción gradual de nuevas tecnologías o estándares mediante pruebas controladas y coexistencia con la infraestructura existente, minimizando riesgos operativos.

Integridad del software: Propiedad que garantiza que el código, binarios y configuraciones no han sido alterados sin autorización. Se asegura mediante firmas digitales, hashes y verificaciones en el pipeline y en tiempo de arranque.

Interoperabilidad: Capacidad de diferentes dispositivos, estándares y plataformas para comunicarse e integrarse de manera coherente, permitiendo que componentes de distintos fabricantes funcionen conjuntamente.

IoT (Internet of Things): Paradigma tecnológico que conecta dispositivos físicos a internet para recopilar, intercambiar y procesar datos.

JTAG: Interfaz de depuración a bajo nivel que permite lectura y escritura directa de memoria y control del procesador. Su acceso sin restricciones implica control total del dispositivo.

Kubernetes: Plataforma de orquestación de contenedores que automatiza el despliegue, escalado y gestión de aplicaciones.

Matter: Estándar de conectividad orientado a interoperabilidad en entornos domésticos e IoT, diseñado con mecanismos de seguridad nativos, gestión de dispositivos y funcionalidades que facilitan la integración entre fabricantes.

Mecanismos de actualización seguros: Conjunto de controles aplicados al proceso OTA y al pipeline de distribución: firmas digitales de binarios, verificaciones de integridad en el dispositivo, canales cifrados, protección de endpoints y políticas de rollback ante fallos.

Modelado de amenazas: Técnica empleada para identificar, analizar y priorizar posibles amenazas que podrían afectar a un sistema antes y después de su despliegue.

mTLS (mutual TLS): Extensión de TLS donde tanto el cliente como el servidor se autentican mutuamente mediante certificados digitales. Aumenta la seguridad al garantizar que ambas partes son legítimas, comúnmente usado en comunicaciones entre servicios, APIs o dispositivos IoT.

MQTT (Message Queuing Telemetry Transport): Protocolo de mensajería ligero basado en el modelo *publicador/suscriptor*, diseñado para entornos con recursos limitados, ideal para comunicación entre dispositivos IoT.

Over-The-Air (OTA): Mecanismo que permite actualizar firmware de forma remota. La integridad y autenticidad del binario dependen de la protección del endpoint OTA y del pipeline de construcción.

Ops: Conjunto de tareas orientadas a la implementación, supervisión, mantenimiento y corrección de los servicios o aplicaciones en producción.

Pipeline CI/CD: Conjunto de procesos automatizados para construir, analizar y desplegar software. Incluye runners, contenedores, workflows y secretos asociados.

Protocolos no cifrados: Protocolos que transmiten datos en texto claro sin confidencialidad ni autenticación, lo que facilita la inspección, modificación e inyección de mensajes por parte de atacantes.

QoS (Quality of Service): Mecanismo del protocolo MQTT que define el nivel de garantía en la entrega de mensajes (0, 1 o 2). Permite balancear fiabilidad, latencia y consumo de recursos según las necesidades del sistema IoT.

Rate limiting: Técnica que restringe la cantidad de solicitudes o mensajes que un cliente puede enviar en un intervalo de tiempo. Reduce el riesgo de DoS, abuso de recursos y picos de tráfico inesperados.

Recursos limitados: Condición de hardware en la que el dispositivo dispone de restricción de CPU, memoria, almacenamiento o energía, lo que condiciona la elección y la implementación de controles de seguridad.

Red mallada / mesh: Arquitectura de red en la cual los nodos cooperan para retransmitir y encaminar mensajes, generando múltiples rutas entre emisores y receptores y aumentando la disponibilidad y resistencia frente a fallos puntuales.

Resiliencia: Propiedad del sistema para soportar incidentes o fallos y mantener o recuperar su operatividad mediante redundancia, aislamiento y mecanismos de recuperación.

Riesgo: Combinación de la probabilidad de explotación de una vulnerabilidad y el impacto resultante para la organización o el sistema.

Riesgos en la cadena de suministro: Amenazas asociadas a componentes, dependencias, servicios o procesos externos que intervienen en la creación y entrega del software. Su explotación permite introducir vulnerabilidades o código malicioso sin atacar directamente el dispositivo final.

Rollback/Fallback: Mecanismo que permite revertir a una versión previa y estable del firmware cuando la actualización actual falla, no supera las verificaciones de integridad o impide el arranque correcto. Asegura continuidad operativa y evita que el dispositivo quede inutilizado tras una actualización defectuosa.

Runners de GitHub Actions: Máquinas (hosted o self-hosted) que ejecutan pipelines. Si se comprometen, permiten modificar artefactos, obtener secretos o inyectar código malicioso.

SAST (Static Application Security Testing): Herramientas de análisis de seguridad de aplicaciones estáticas que examinan el código fuente para detectar vulnerabilidades antes de la ejecución.

SBOM (Software Bill of Materials): Inventario estructurado que lista todos los componentes, librerías, dependencias y versiones que forman parte de un software.

Sec: Incorporación de mecanismos, prácticas y herramientas de seguridad durante todas las etapas del desarrollo y despliegue.

Secure Boot: Mecanismo de seguridad que garantiza que el dispositivo solo ejecute firmware autenticado y no modificado. Verifica criptográficamente la firma del bootloader y/o del firmware durante el proceso de arranque para evitar la ejecución de código malicioso o no autorizado.

SonarCloud: Plataforma de análisis continuo de código fuente que evalúa métricas de seguridad, mantenibilidad y calidad.

STRIDE: Modelo de clasificación de amenazas desarrollado por Microsoft que agrupa riesgos en seis categorías:

Superficie de ataque: Conjunto de puntos de entrada, interfaces o componentes del sistema que pueden ser aprovechados por un atacante para causar daño o comprometer activos.

Supply-Chain Attack: Ataque orientado a comprometer fases del ciclo de vida del software: repositorios, dependencias, pipelines o servidores OTA con el objetivo de introducir código malicioso o artefactos adulterados.

Tampering: Manipulación no autorizada de firmware, artefactos o configuraciones, ya sea por acceso físico (UART/JTAG) o mediante modificaciones en el pipeline CI/CD.

Tamper-resistant: Significa resistente a manipulaciones. Se utiliza para describir dispositivos, sistemas o mecanismos diseñados para dificultar, detectar o retrasar cualquier intento de alteración, sabotaje o acceso no autorizado.

ThingsBoard: Plataforma de código abierto para la gestión y visualización de datos IoT, integrable con protocolos como MQTT.

Thread: Protocolo de red mallada (mesh) de baja potencia basado en IPv6, optimizado para dispositivos con recursos limitados, que proporciona rutas redundantes, autocuración y baja latencia en redes locales de sensores y actuadores.

TLS 1.3: Versión moderna del protocolo de cifrado TLS que ofrece mayor seguridad y mejor rendimiento, reduciendo el número de pasos en el handshake y eliminando algoritmos inseguros. Protege la confidencialidad e integridad de las comunicaciones entre cliente y servidor.

Topologías distribuidas: Modelos de arquitectura donde funciones clave (comunicación, procesamiento, almacenamiento) se reparten entre múltiples nodos en lugar de concentrarse en un único punto, reduciendo puntos únicos de fallo.

Trazabilidad: Capacidad de reconstruir el historial de cambios, orígenes y transformaciones de artefactos y configuraciones durante la cadena de construcción, facilitando auditorías y análisis forense.

UART: Interfaz serial asíncrona utilizada para depuración. Puede exponer consola o mensajes del bootloader si no está protegida contra acceso físico.

Vulnerabilidad: Debilidad o configuración insegura que puede ser explotada por una amenaza.

Zero-touch provisioning: Proceso de aprovisionamiento inicial de dispositivos que permite su configuración y enrolamiento automático y seguro sin intervención manual, empleando autenticación y provisión asistida por servidores de confianza.

Apéndice B

Evidencias

B.1. Imágenes de pruebas y configuraciones

B.1.1. Configuraciones de red

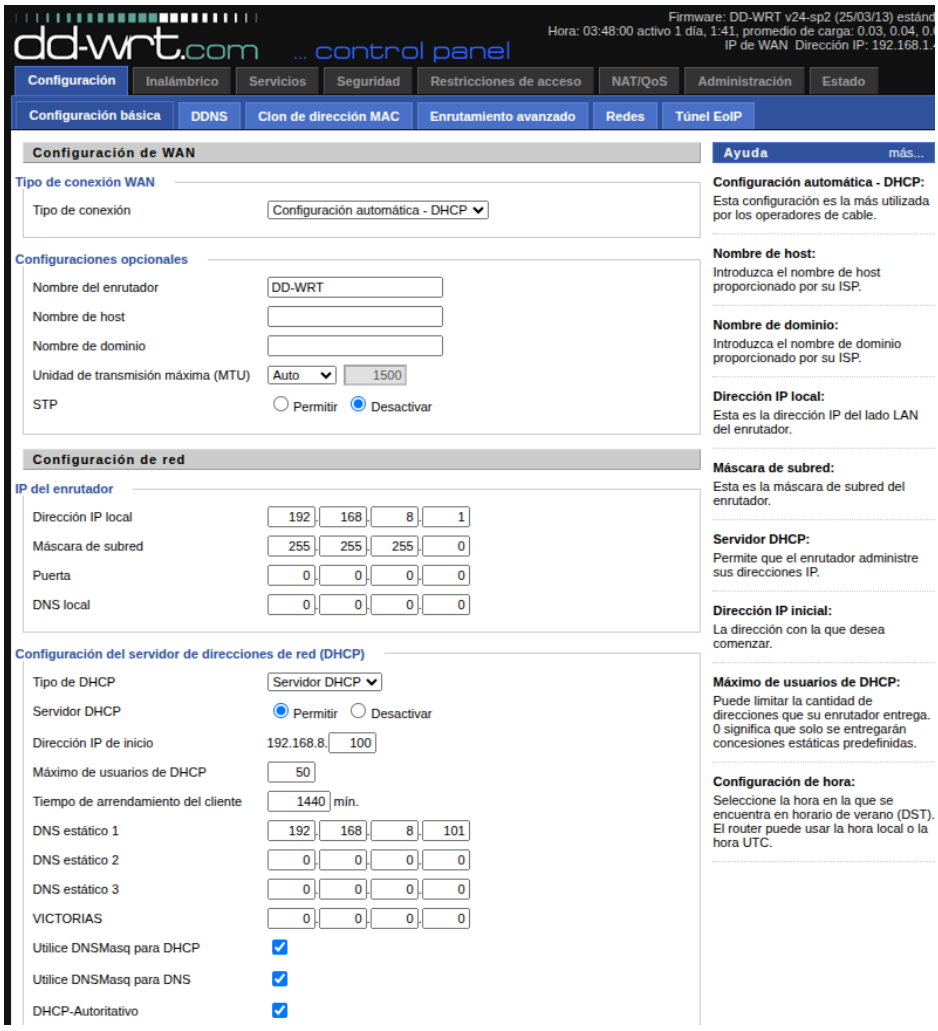


Figura B.1 Configuración de red

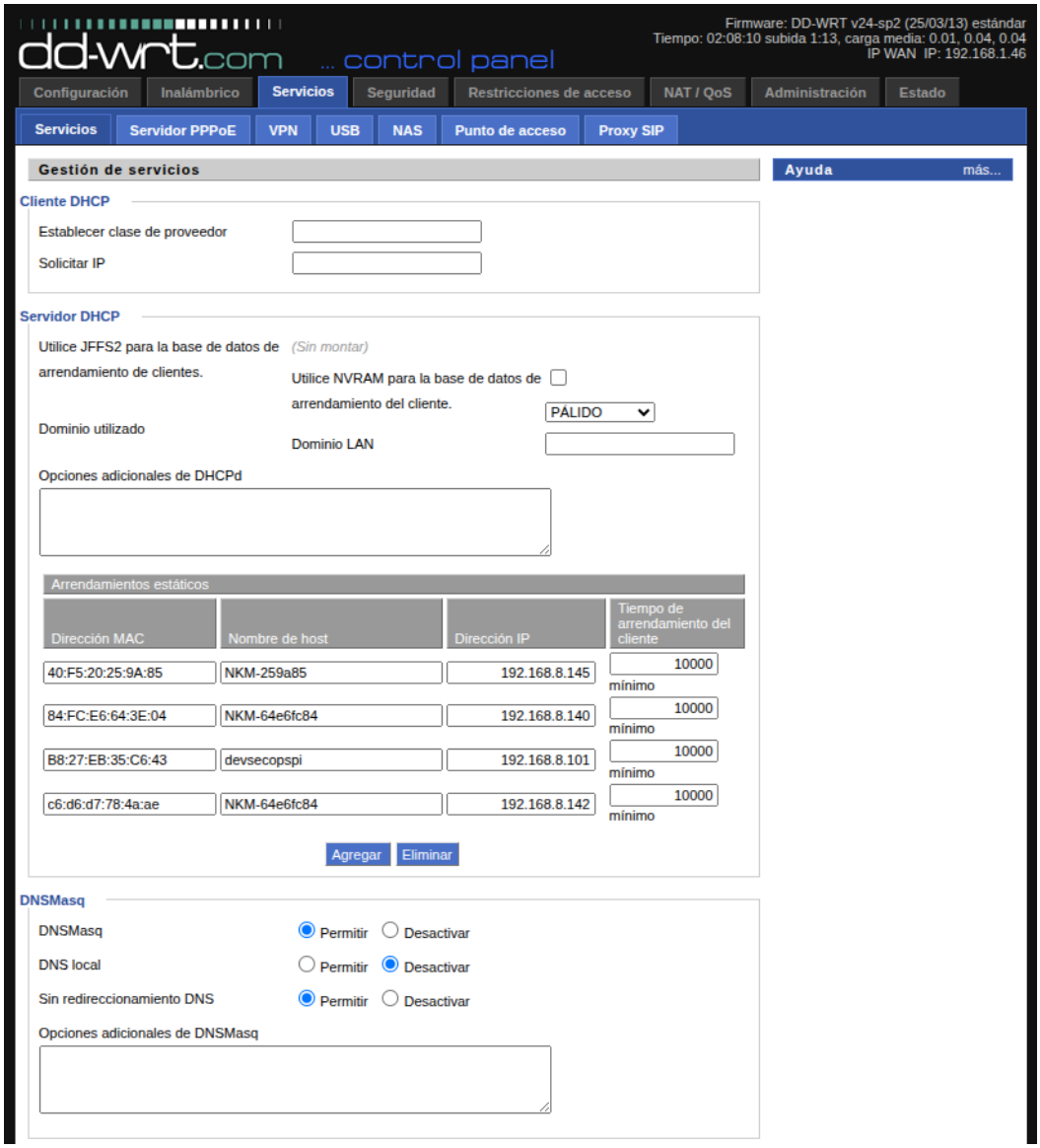


Figura B.2 Configuración de servicios

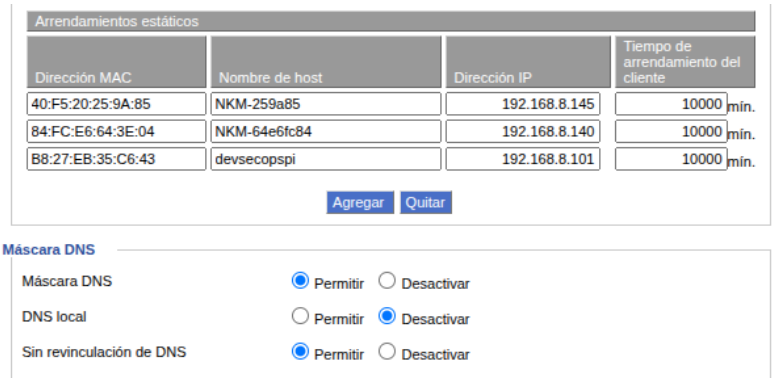


Figura B.3 Configuraciones de IP

B.2. Casos prácticos

B.2.1. Caso A - Escaneo de superficie y descubrimiento

El primer paso consistió en identificar el dispositivo objetivo dentro del segmento de red. A partir de un escaneo activo fue posible detectar, enumerar puertos abiertos y observar los servicios expuestos sin autenticación.

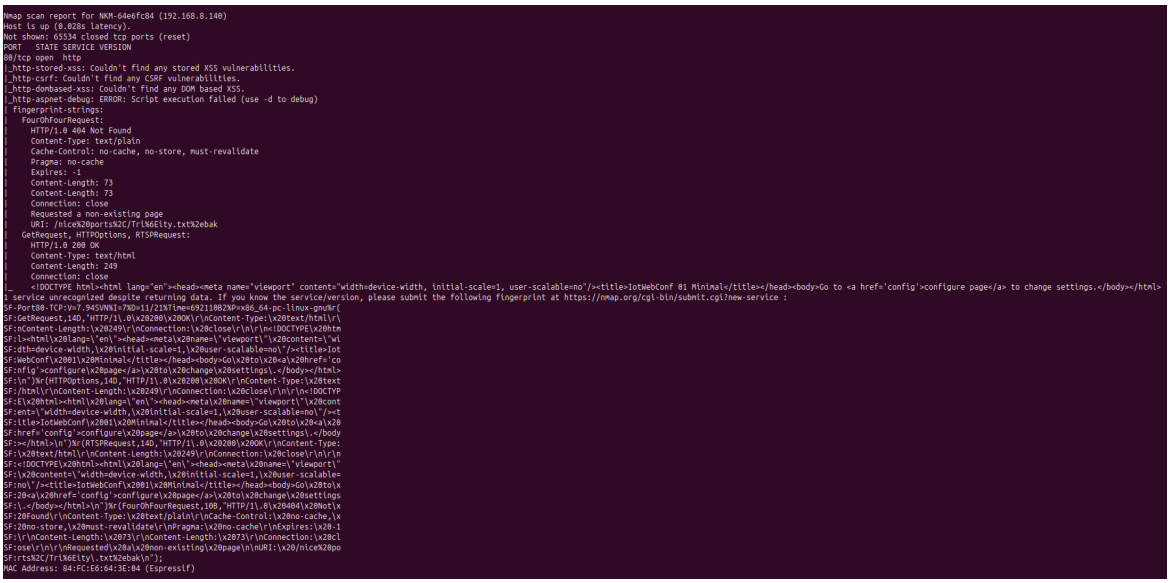


Figura B.4 Escaneo de puertos

Una vez identificados el puerto y las tecnologías, se evidencia que existe la enumeración de rutas accesibles mediante Gobuster.

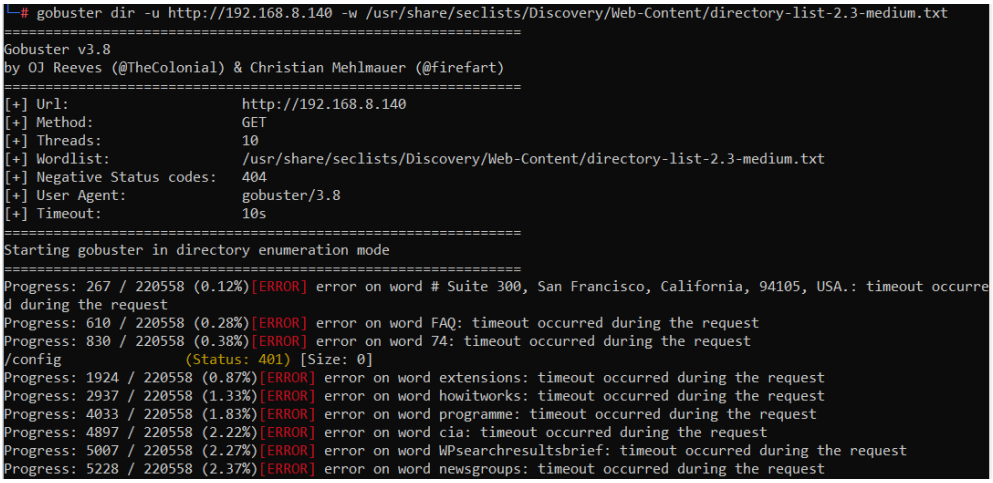


Figura B.5 Escaneo de rutas

En la siguiente imagen se puede ver la pantalla de configuración del dispositivo.

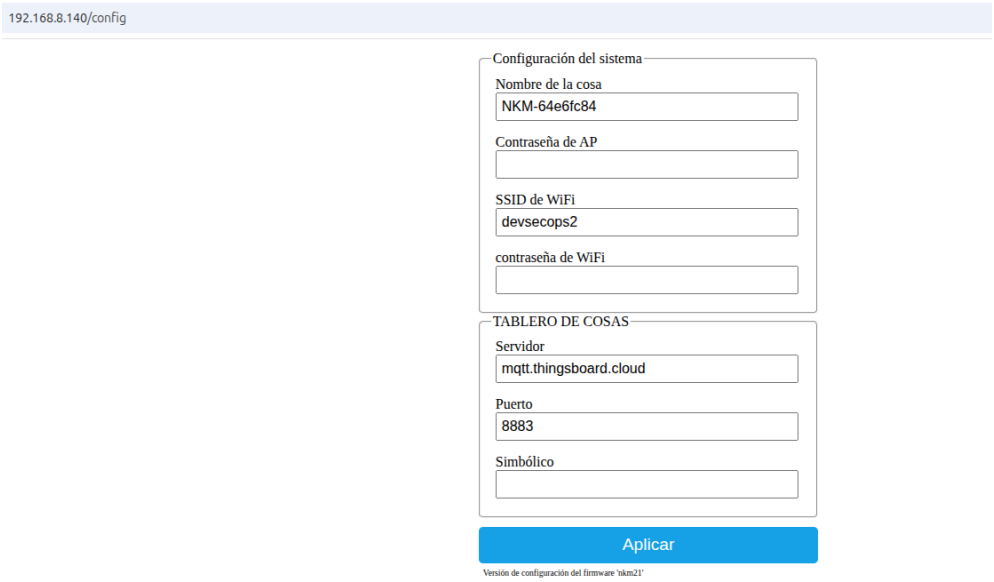


Figura B.6 Configuración de ESP32

Mitigación

Las vulnerabilidades identificadas en el dispositivo, especialmente aquellas asociadas al uso de HTTP sobre el puerto 80 para tareas de configuración —lo que implica la transmisión de datos sin cifrado y la posibilidad de interceptación o manipulación— pueden ser mitigadas mediante la aplicación de controles de segmentación de red.
La implementación de una arquitectura en cascada o el uso de VLANs dedicadas permite aislar el dispositivo dentro de un dominio de menor exposición, restringiendo el acceso únicamente a los sistemas autorizados y evitando que el tráfico de gestión quede disponible en la red general.

Búsqueda de vulnerabilidades y tecnologías en ThingBoardCloud

Dado que la conexión del dispositivo en este laboratorio se llevó a cabo a través de la integración del ESP32 a la plataforma de ThingBoardCloud, se realiza el descubrimiento de tecnologías, versiones y vulnerabilidades que pudieran existir en dicha plataforma.

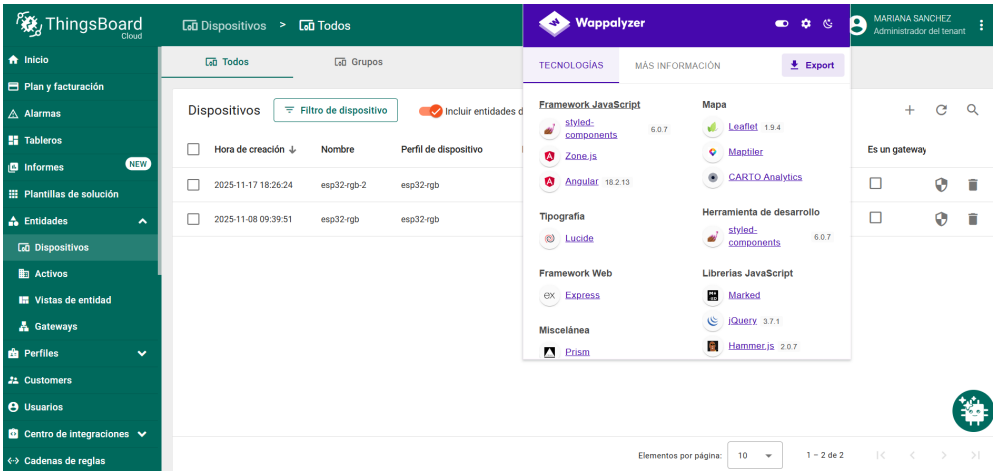


Figura B.7 Vista de tecnologías

La plataforma web no es accesible sin las credenciales adecuadas, los dispositivos se conectan únicamente mediante tokens. Ante esto, se evalúan las versiones de las tecnologías y, como se puede ver en la imagen, se encuentran en la última versión estable para cada una al momento del análisis (23/11/2025).

Por otra parte, de esta misma plataforma se examinan mediante Shodan puertos abiertos, certificados y dominio.

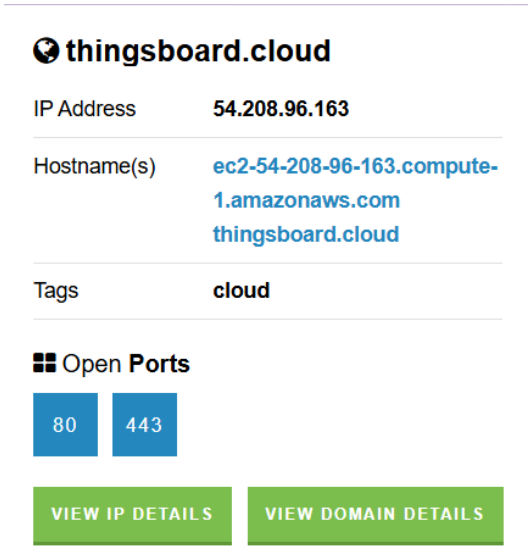


Figura B.8 Vista de tecnologías

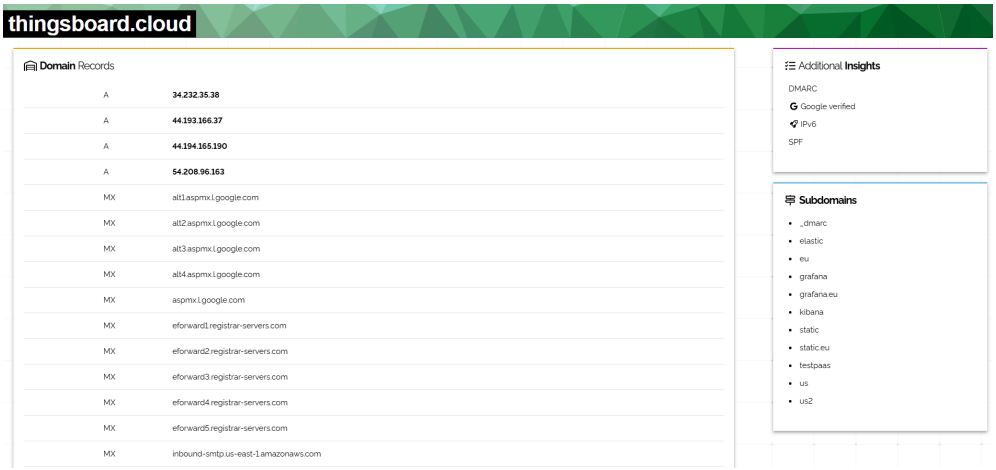


Figura B.9 Vista de dominios

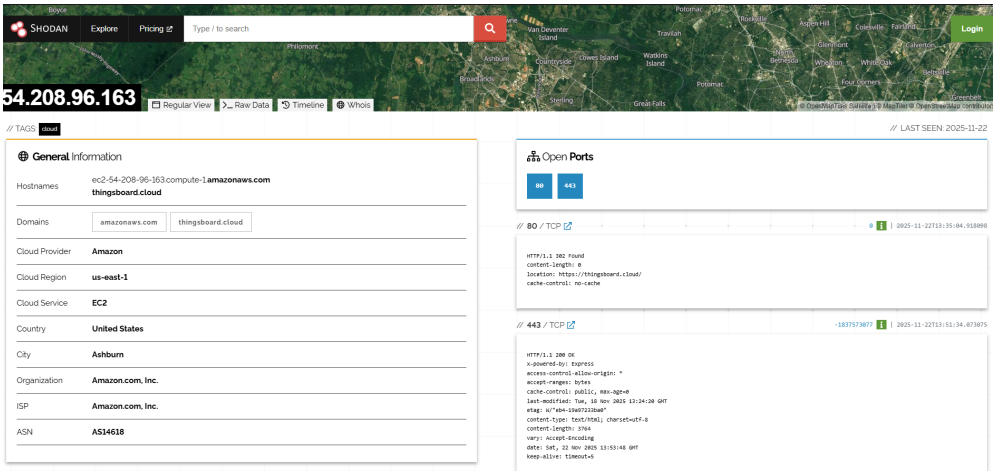


Figura B.10 Vista de IP dominios

Los resultados permiten concluir que la exposición principal recae sobre el dispositivo local —mitigable mediante segmentación de red— mientras que la plataforma asociada mantiene un nivel de seguridad adecuado para su función en el ecosistema IoT.

B.2.2. Caso B — Captura y análisis de tráfico (confidentiality/MITM)

El análisis pasivo del tráfico mediante Wireshark permitió observar la transmisión de datos sensibles sin cifrado. Durante la captura se identificó tráfico en texto plano, incluyendo credenciales transmitidas sin protección.

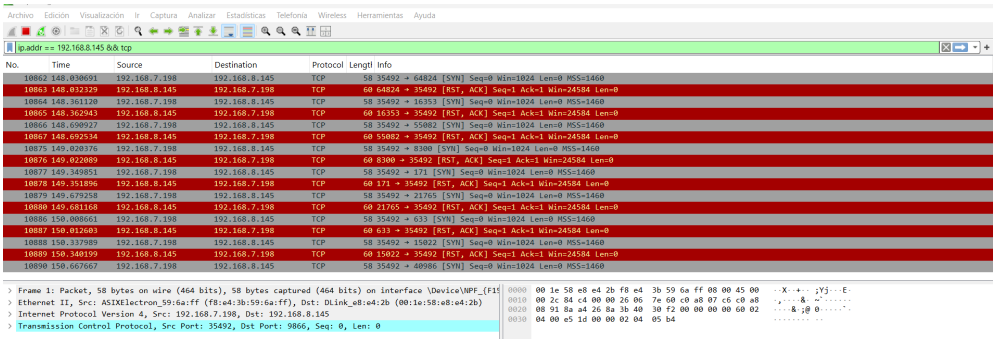


Figura B.11 Captura de tráfico en Wireshark generada por el dispositivo.

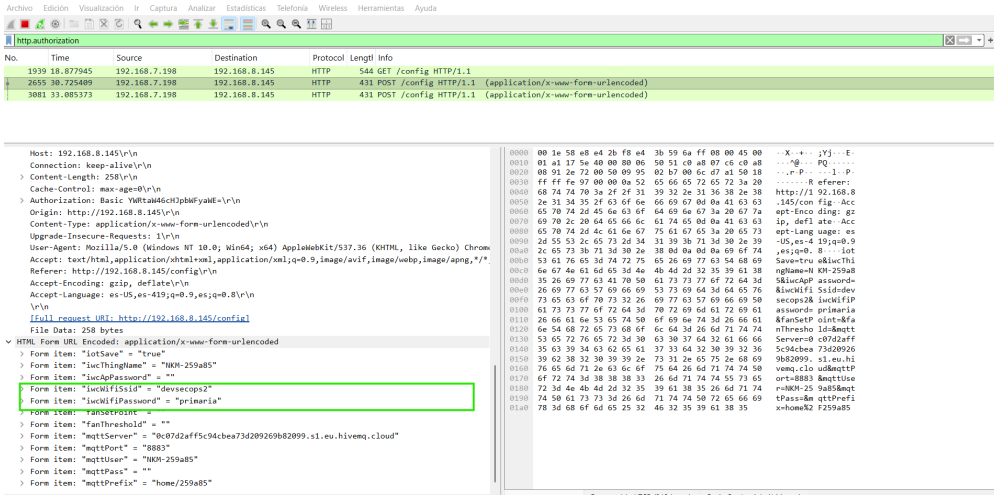


Figura B.12 Credenciales observadas en texto plano dentro del tráfico capturado.

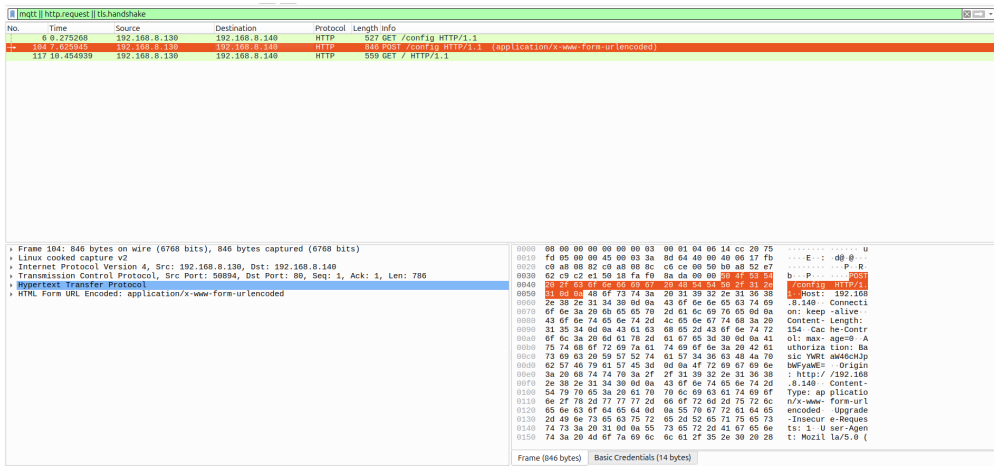


Figura B.13 Capturas de tráfico

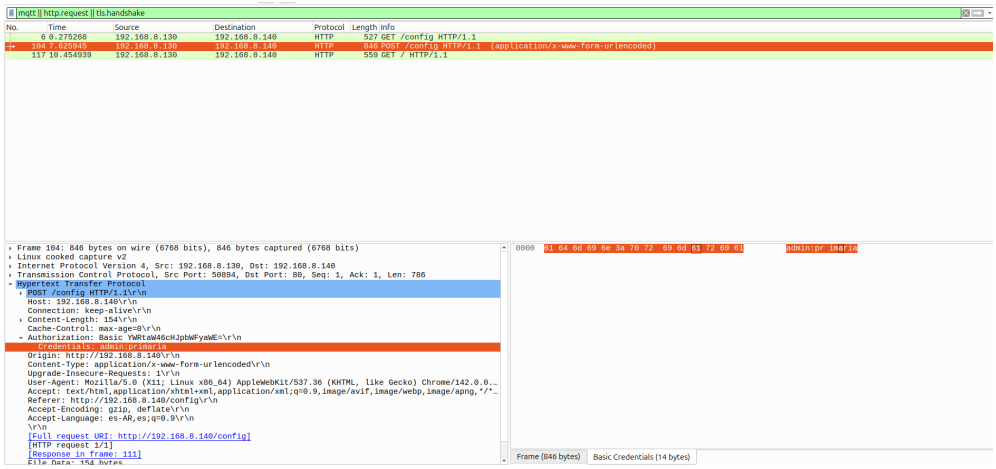


Figura B.14 Capturas de tráfico

Configuración de MQTT-TLS

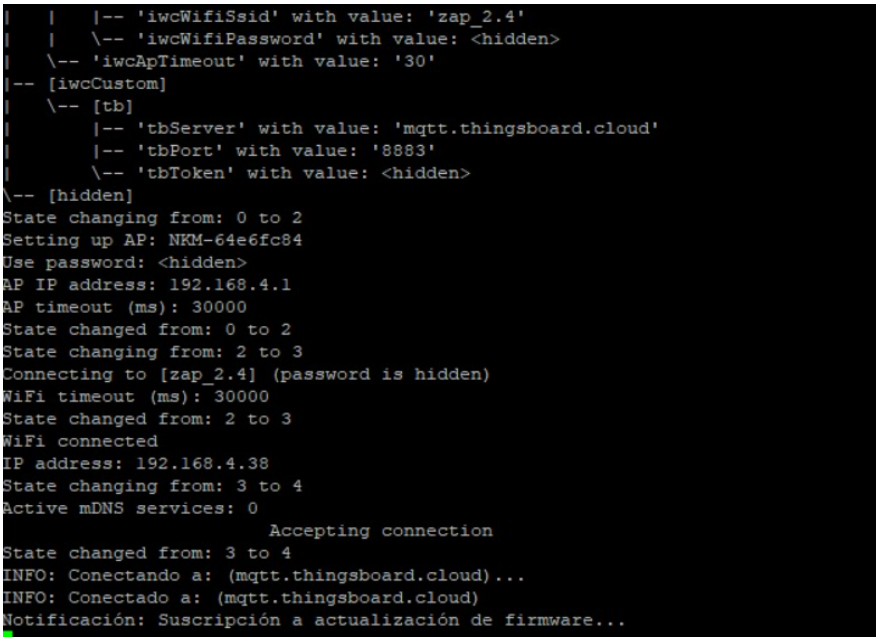


Figura B.15 Implementación de TLS para proteger el tráfico MQTT.

Mitigación

La implementación de una arquitectura en cascada o el uso de VLANs dedicadas permite aislar el dispositivo dentro de un dominio de menor exposición, restringiendo el acceso únicamente a los sistemas autorizados y evitando que el tráfico de gestión quede disponible en la red general. Para la remediación de la transmisión de datos del protocolo MQTT, se implementa MQTT-TLS.

B.2.3. Caso C — Prueba OTA maliciosa / fallback recovery (bricking)

Dado que la realización de esta prueba implica particionado de memoria, redundancia de dispositivos para realizar la sobreescritura necesaria, se prevé como evidencia únicamente la evidencia del uso seguro del código y su posterior vista desde la plataforma de administración de dispositivos prevista.

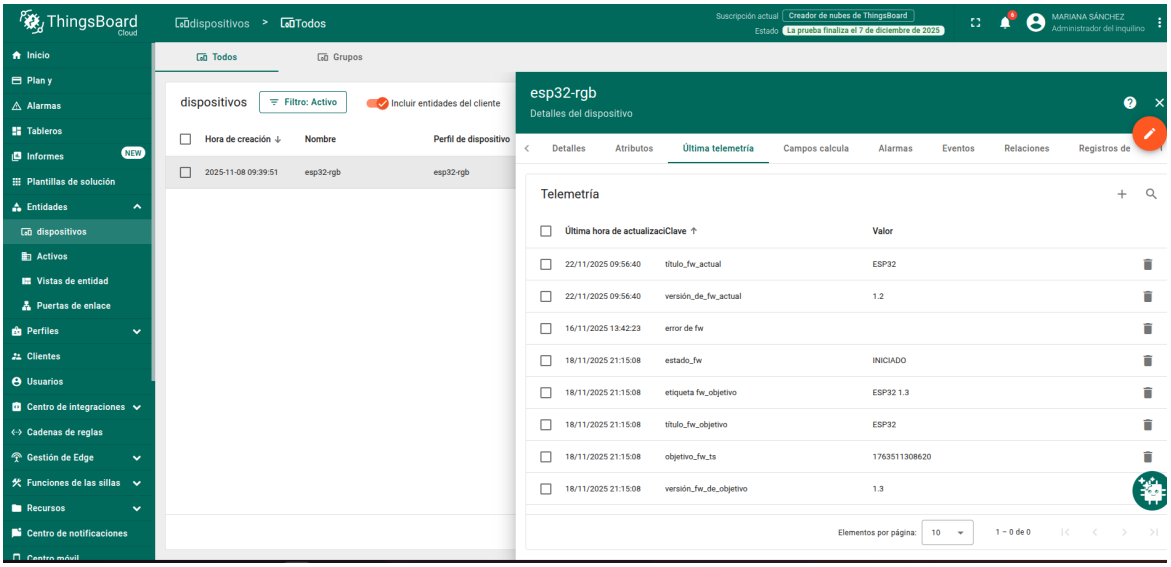


Figura B.16 Actualizaciones OTA

B.2.4. Caso D — Compromiso del pipeline (supply chain/repositorio)

Para evaluar la capacidad de detección temprana frente a cambios potencialmente maliciosos, se introdujeron modificaciones controladas al repositorio con el fin de observar el comportamiento de las herramientas SAST integradas en el pipeline.

La Figura B.17 muestra un análisis fallido generado por Cppcheck, evidenciando defectos en el código fuente que hubieran permitido la propagación de fallas si el pipeline no hubiese detenido la ejecución.

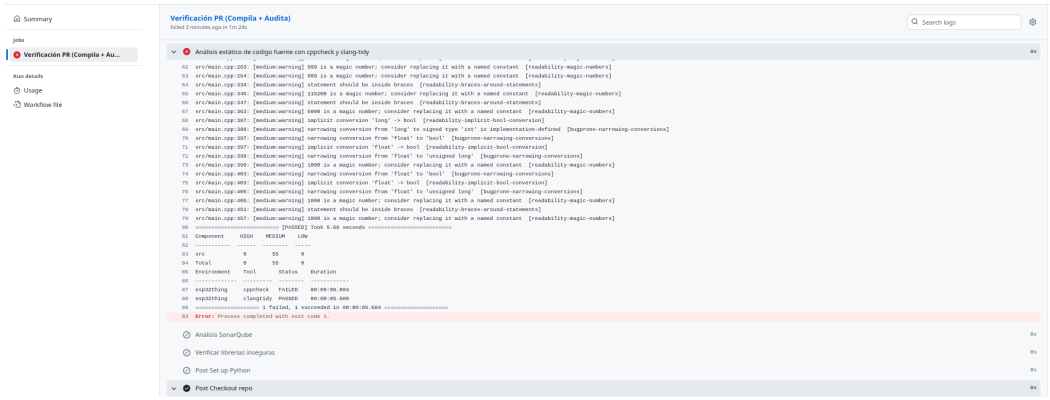


Figura B.17 Resultados de análisis estático con Cppcheck.

Posteriormente, SonarCloud complementó el análisis con mayor profundidad. Las Figuras B.18 y B.19 presentan hallazgos críticos, incluyendo malas prácticas como credenciales codificadas en texto plano y problemas de calidad que representan riesgos directos para la integridad del pipeline.

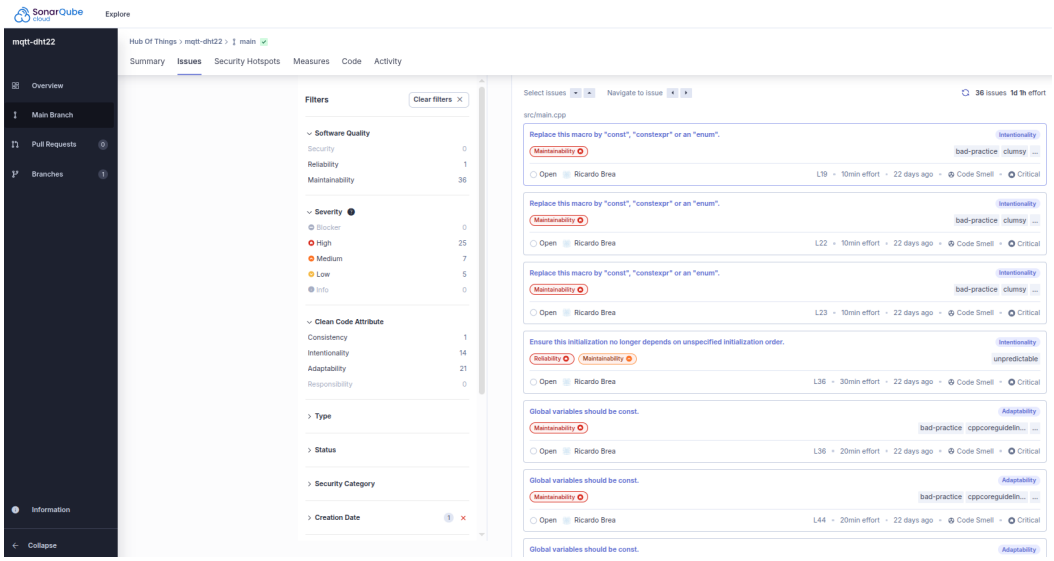


Figura B.18 Problemas detectados en SonarCloud.

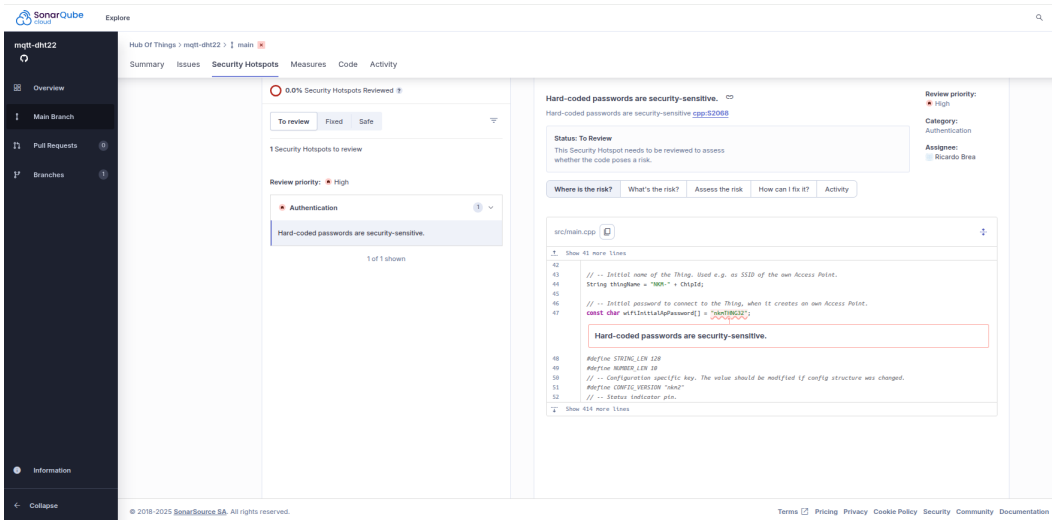


Figura B.19 Detección de contraseñas codificadas por SonarCloud.

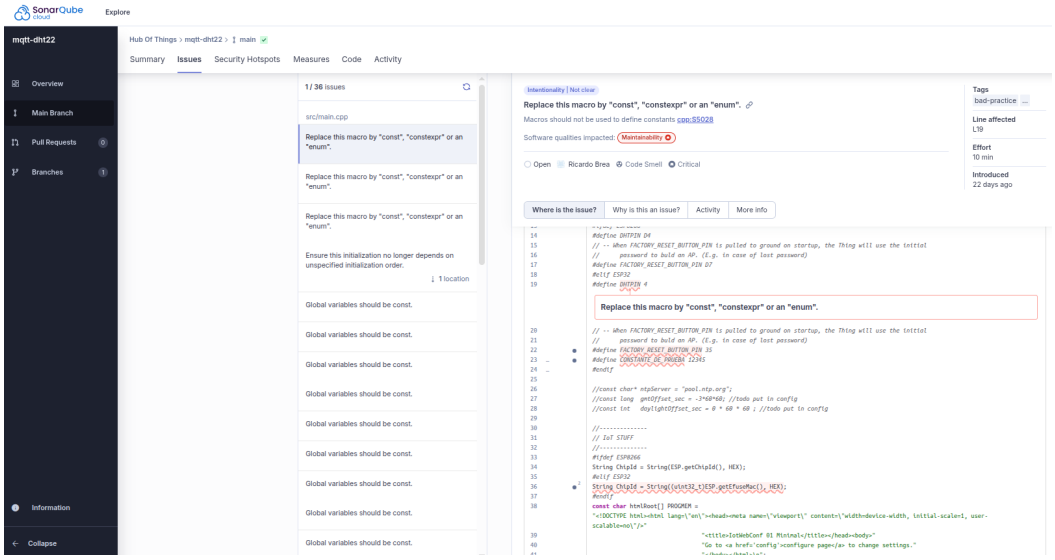


Figura B.20 Mantenibilidad del código en SonarCloud.

Para complementar este análisis, se evaluaron los indicadores de mantenibilidad, como se observa en la Figura B.20. Estos indicadores permiten medir el impacto acumulado de defectos y dificultades para mantener el código, factores clave en escenarios de supply chain attacks donde los atacantes suelen introducir cambios que pasan desapercibidos.

El pipeline de GitHub Actions respondió conforme lo esperado. En la Figura B.21 se observa un release bloqueado debido a fallas detectadas previamente, lo que evidencia la efectividad de los mecanismos de control que impiden que un runner comprometido o un PR malicioso avance a la fase de despliegue.

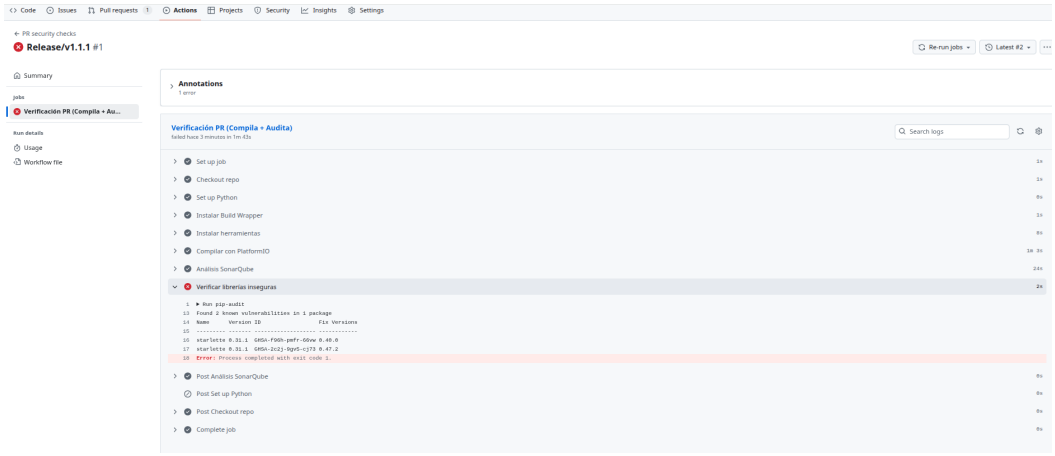


Figura B.21 Release bloqueado en GitHub Actions debido a errores previos.

La Figura B.22 muestra el pipeline ejecutándose sobre un pull request, integrando los resultados de SonarCloud y demás herramientas SAST antes de permitir la fusión del código. Esto garantiza que cualquier cambio, aún si proviene de un atacante intentando insertar código malicioso, será evaluado bajo los mismos estándares de calidad y seguridad.

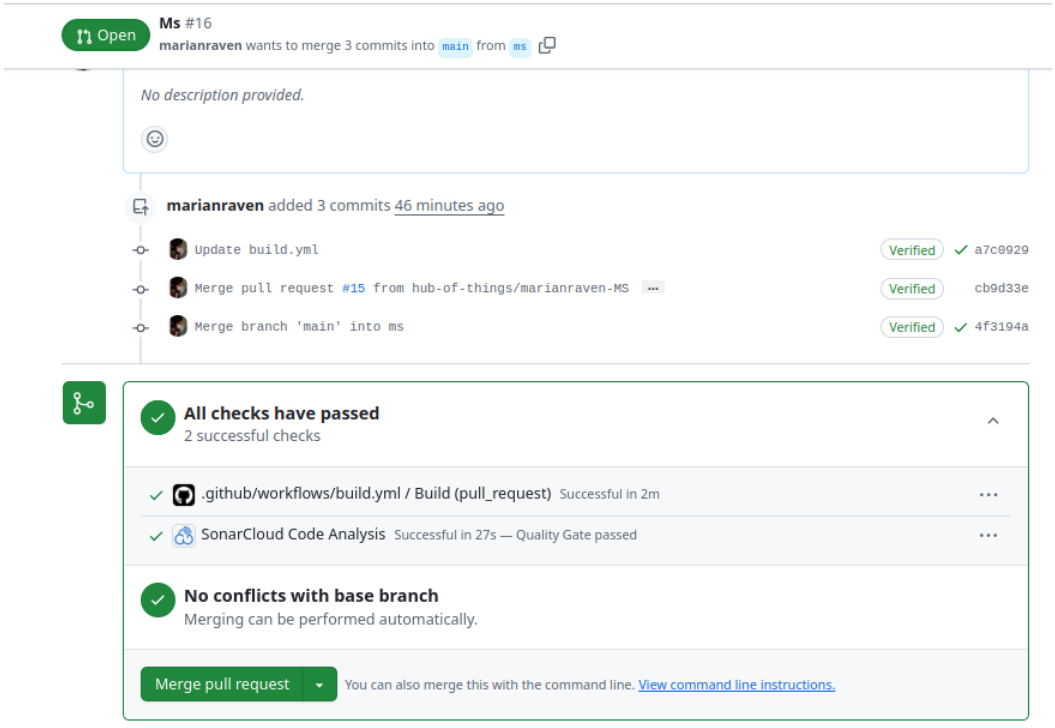


Figura B.22 Pipeline de build.

Finalmente, en la Figura B.23 se presenta un ejemplo de pull request corregido. Luego de las detecciones del pipeline, el desarrollador introduce los cambios necesarios, se reejecutan las

verificaciones SAST y se observa cómo el PR queda en estado aprobado. Esto evidencia que la integridad del pipeline se mantiene incluso frente a intentos de inyectar fallas, demostrando un nivel adecuado de resiliencia.

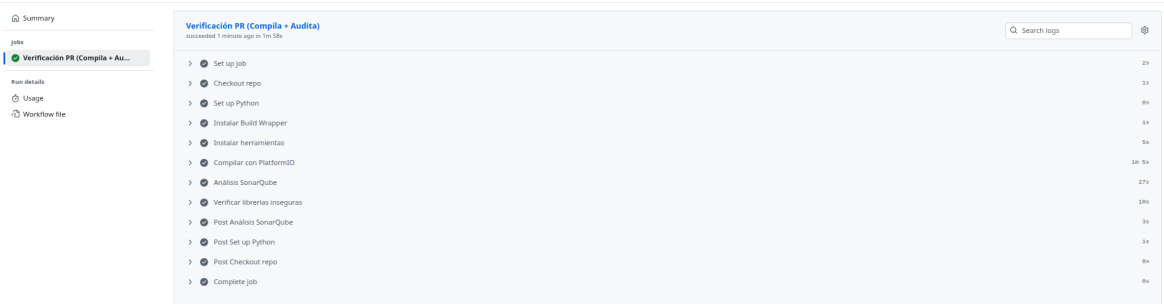


Figura B.23 PR corregido luego de resolver issues detectados.

B.2.5. Caso E — Fuerza bruta / panel administrativo

Para este caso se evaluó la capacidad del dispositivo y de los servicios asociados para resistir intentos reiterados de autenticación no autorizada. El objetivo principal fue validar si existían mecanismos de protección frente a ataques de fuerza bruta, tales como limitación de intentos, bloqueos temporales, alertas, o endurecimiento de credenciales.

Durante los experimentos se ejecutaron ataques con listas de contraseñas reconocidas, como el diccionario rockyou, así como variaciones incrementales y combinaciones de mayor complejidad. El objetivo fue determinar si el servicio reaccionaba ante múltiples intentos fallidos o permitía la autenticación sin restricciones.

Las Figuras B.24, B.25 y B.26 muestran ejemplos representativos del proceso de fuerza bruta llevado a cabo, incluyendo iteraciones, variantes del ataque y ejecución mediante diccionarios comunes.

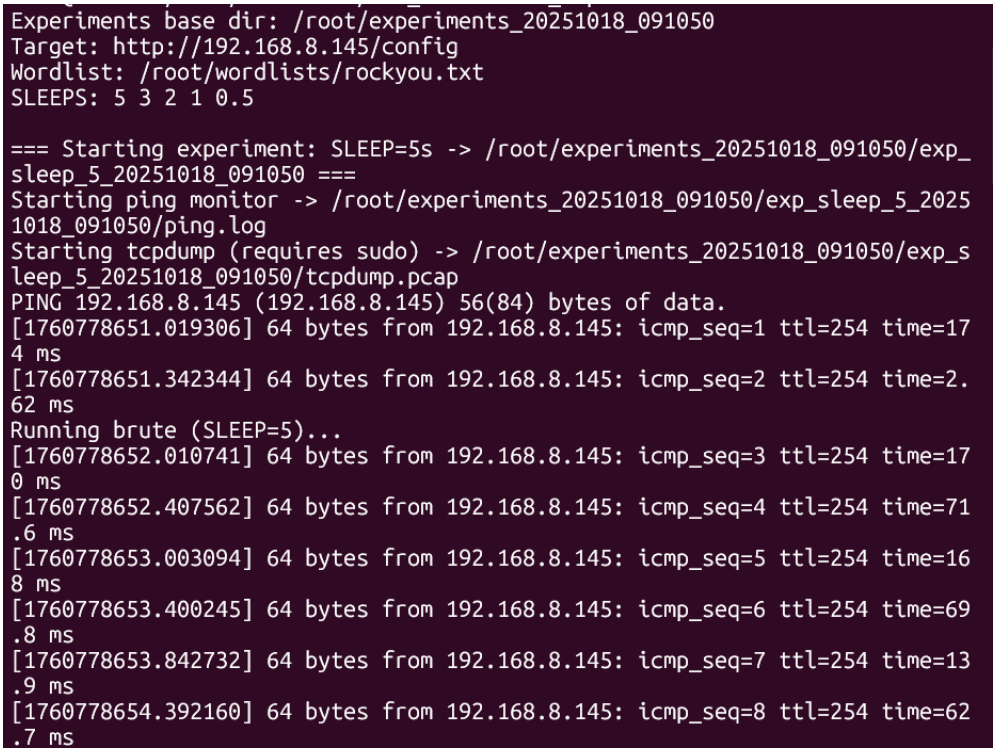


Figura B.24 Ejecución del ataque de fuerza bruta sobre el servicio del dispositivo.

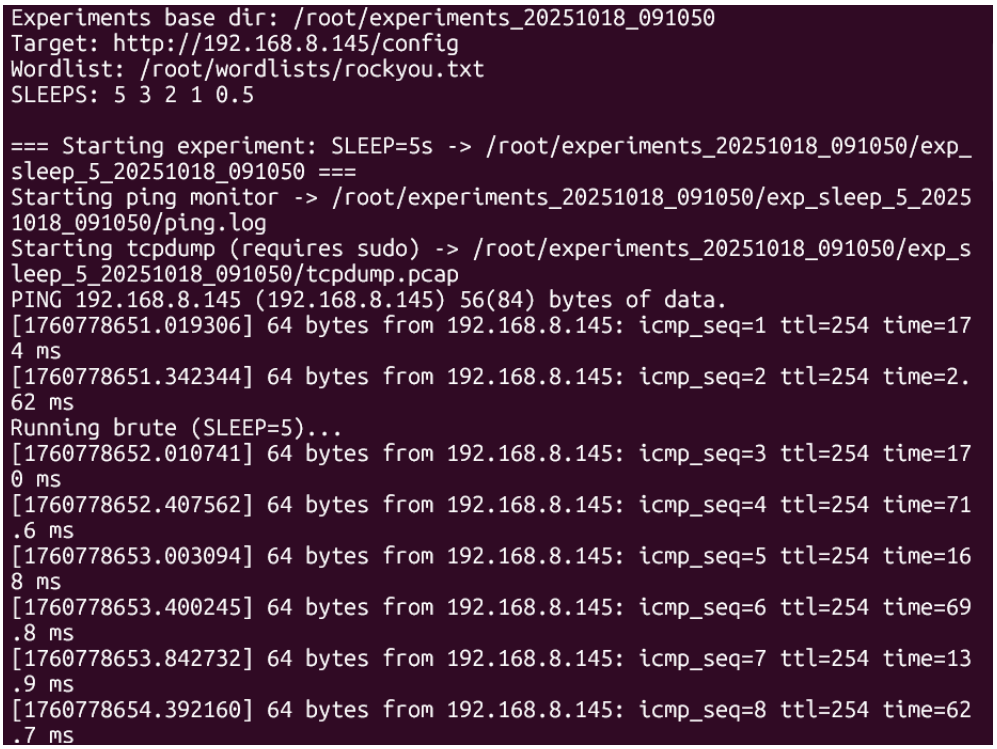


Figura B.25 Conjunto de experimentos realizados durante la prueba de fuerza bruta.

```
Experiment SLEEP=5 finished. Results in /root/experiments_20251018_091050/e
xp_sleep_5_20251018_091050
---- Summary ----
PING_PID=6974
TCPDUMP_PID=6975
start_epoch=1760778651
BRUTE_PID=6981
end_epoch=1760778736
elapsed_seconds=85
Stopped ping monitor (pid 6974)
Stopped tcpdump (pid 6975)
attempts_logged=12
found=yes
FOUND admin:primaria attempt=11 at 1760778735.531
Computing ping stats...
ping_count=163
min_ms=1.660
avg_ms=3.896
max_ms=71.600
std_ms=8.944
gaps=0
```

Figura B.26 Iteración avanzada del ataque de fuerza bruta.

Los resultados mostraron que el dispositivo no implementa mecanismos de protección contra fuerza bruta, ya que durante todas las pruebas, no se activó ningún bloqueo temporal, no se generaron alertas visibles, no se evidenció limitación de velocidad (rate limiting), las contraseñas débiles fueron aceptadas en determinados escenarios, el servicio por defecto no aplica controles adicionales como mTLS o verificación reforzada.

Este comportamiento incrementa significativamente el riesgo de compromiso del dispositivo mediante ataques automatizados, especialmente cuando se utilizan credenciales por defecto o configuraciones iniciales.

Mitigación recomendada

Para reducir la superficie de ataque asociada a intentos de fuerza bruta, se recomienda implementar mecanismos como:

- Bloqueo temporal ante múltiples fallos consecutivos.
- Limitación de velocidad (rate limiting) para conexiones repetidas.
- Autenticación basada en certificados (mTLS) en el broker MQTT.
- Políticas de contraseñas fuertes.
- Alertas y registro centralizado de intentos sospechosos.

Estas medidas previenen accesos no autorizados y reducen significativamente el riesgo de explotación por automatización.

B.2.6. Caso F — DDoS sobre el dispositivo

Se ejecutaron iteraciones progresivas de 200, 300 y 500 solicitudes concurrentes para evaluar el impacto sobre el servicio expuesto por el dispositivo.

En la Figura B.27 se observa que ante 200 solicitudes, el dispositivo aún mantiene respuesta estable, con tiempos elevados pero sin cortes.

```
└─# wrk -t4 -c200 -d30s http://192.168.8.140
Running 30s test @ http://192.168.8.140
  4 threads and 200 connections
  Thread Stats   Avg      Stdev     Max   +/- Stdev
    Latency   109.41ms    6.01ms 142.62ms   89.76%
    Req/Sec    12.95      7.14    40.00     84.28%
  1046 requests in 27.08s, 340.24KB read
  Socket errors: connect 0, read 0, write 0, timeout 11
Requests/sec:     38.62
Transfer/sec:     12.56KB
```

Figura B.27 Iteración: 200 request

Al aumentar la presión a 300 solicitudes (Figura B.28), se empieza a notar un incremento marcado en la latencia, junto con los primeros timeouts intermitentes.

```
└─# wrk -t4 -c300 -d30s http://192.168.8.140
Running 30s test @ http://192.168.8.140
  4 threads and 300 connections
  Thread Stats   Avg      Stdev     Max   +/- Stdev
    Latency   109.26ms    7.14ms 152.16ms   90.43%
    Req/Sec    13.66      7.48    40.00     82.53%
  1045 requests in 27.05s, 339.83KB read
  Socket errors: connect 0, read 0, write 0, timeout 11
Requests/sec:     38.63
Transfer/sec:     12.56KB
```

Figura B.28 Iteración; 300 request

Finalmente, con 500 solicitudes (Figura B.29), el dispositivo presenta fallas sostenidas, ausencia de respuestas y degradación total del servicio, lo que indica que no dispone de mecanismos de rate limiting ni protección contra sobrecarga.

```
└─# wrk -t4 -c500 -d30s http://192.168.8.140
Running 30s test @ http://192.168.8.140
  4 threads and 500 connections
  Thread Stats   Avg      Stdev     Max   +/- Stdev
    Latency   112.54ms   15.80ms 309.92ms   93.98%
    Req/Sec    11.87      6.49    40.00     59.27%
  1026 requests in 27.09s, 333.65KB read
  Socket errors: connect 0, read 0, write 0, timeout 12
Requests/sec:     37.87
Transfer/sec:     12.32KB
```

Figura B.29 Iteración: 500 request

Pruebas con `slowhttptest` — Slow-rate / agotamiento de conexiones

Para complementar la evaluación volumétrica, se aplicaron ataques de tipo *Slow Headers* utilizando `slowhttptest`. Este tipo de ataque mantiene cientos de conexiones simultáneas abiertas enviando datos de manera extremadamente lenta, ocupando recursos del dispositivo sin generar tráfico significativo.

Las Figuras B.30 y B.31 muestran los parámetros utilizados (1000 conexiones, intervalos de 10 segundos, pruebas de 240 segundos), así como el comportamiento observado durante la ejecución.

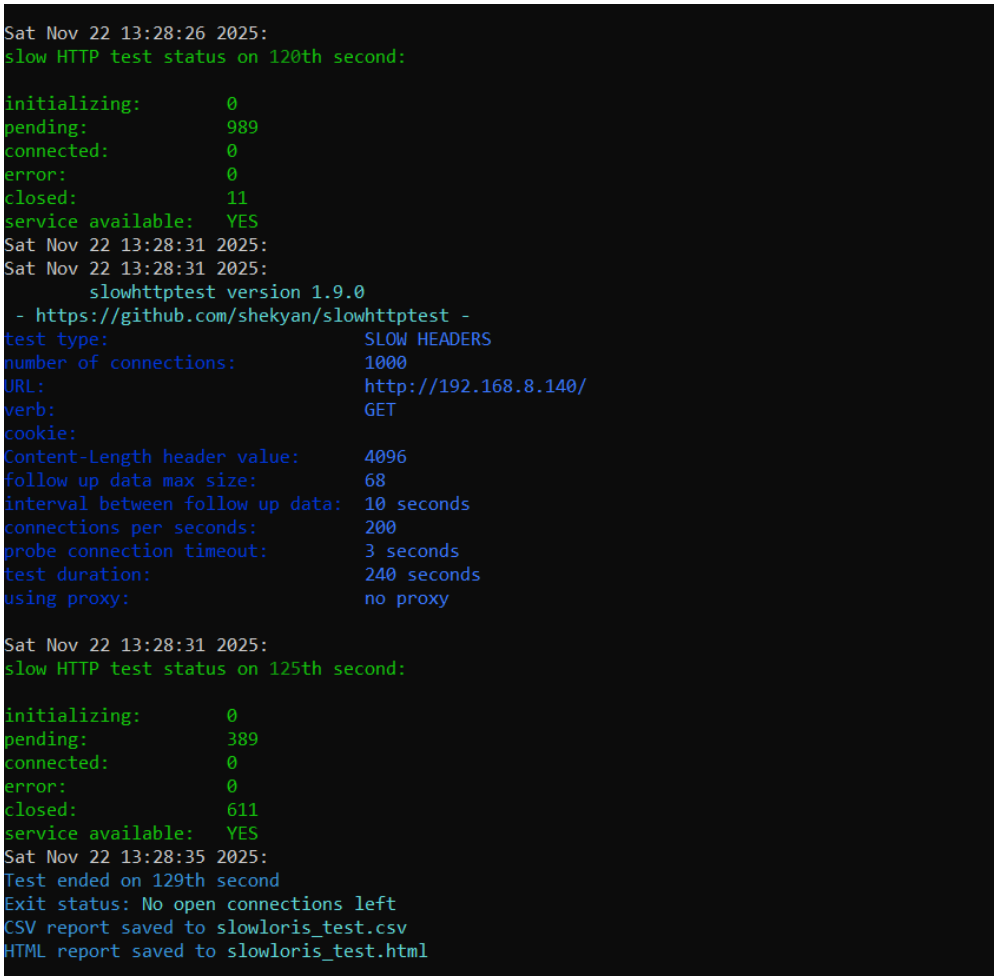


Figura B.30 Slowhttptest

```
└─# slowhttptest -c 1000 -H -g -o slowloris_test -i 10 -r 200 -t GET -u http://192.168.8.140/ -p 3
Sat Nov 22 13:26:26 2025:
Sat Nov 22 13:26:26 2025:
        slowhttptest version 1.9.0
 - https://github.com/shekyan/slowhttptest -
test type:                        SLOW HEADERS
number of connections:            1000
URL:                              http://192.168.8.140/
verb:                             GET
cookie:
Content-Length header value:      4096
follow up data max size:          68
interval between follow up data:  10 seconds
connections per seconds:          200
probe connection timeout:         3 seconds
test duration:                    240 seconds
using proxy:                      no proxy

Sat Nov 22 13:26:26 2025:
slow HTTP test status on 0th second:

initializing:        0
pending:             1
connected:           0
error:               0
closed:              0
service available:   YES
Sat Nov 22 13:26:31 2025:
Sat Nov 22 13:26:31 2025:
```

Figura B.31 Slowhttptest.

El informe HTML generado (Figura B.32) evidencia cómo la mayoría de las conexiones permanecieron abiertas hasta agotar la capacidad del dispositivo.

Test parameters	
Test type	SLOW HEADERS
Number of connections	1000
Verb	GET
Content-Length header value	4096
Cookie	
Extra data max length	68
Interval between follow up data	10 seconds
Connections per seconds	200
Timeout for probe connection	3
Target test duration	240 seconds
Using proxy	no proxy

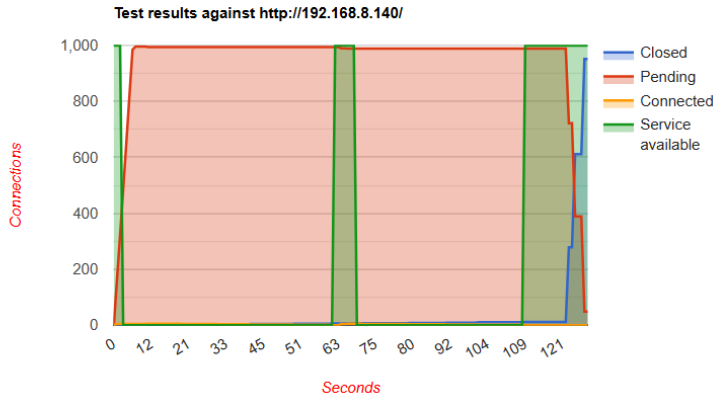


Figura B.32 Slowhttptest Informe HTML

Finalmente, la Figura B.33 muestra la caída completa del servicio tras la saturación, confirmando la ausencia de defensas como:

- límites de conexiones por cliente,
- tiempo máximo de sesión,
- protección contra ataques de tipo slowloris,
- mecanismos de priorización de tráfico o colas de backpressure.

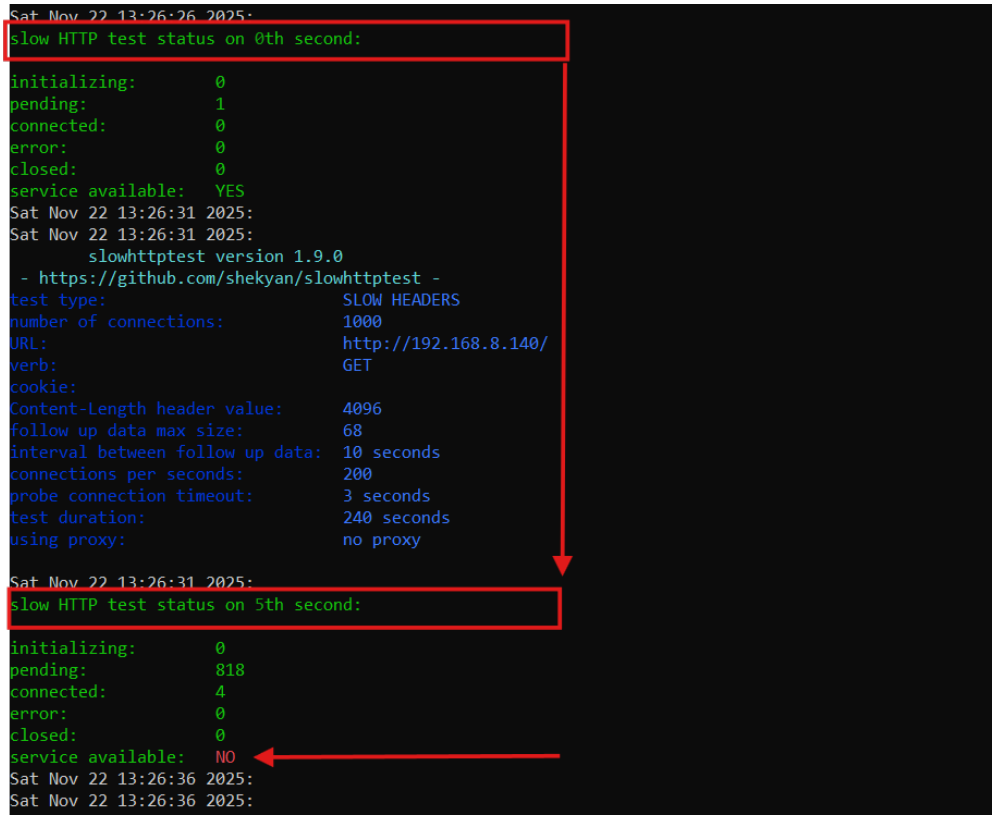


Figura B.33 DDOS completado

Los experimentos demuestran que el dispositivo y el broker presentan vulnerabilidades significativas ante ataques de denegación de servicio, tanto volumétricos como de agotamiento lento de conexiones. No se observaron controles de mitigación nativos, lo que implica que el dispositivo puede quedar inoperable con un volumen de tráfico malicioso.

Apéndice C

Procedimiento de buenas prácticas para IoT

Objetivo

Establecer un procedimiento integral que asegure el desarrollo, despliegue, operación y mantenimiento del software y de los dispositivos IoT asociados, considerando amenazas de acceso físico y lógico no autorizado, gestión de actualizaciones, riesgos inherentes a la conectividad, seguridad en el sistema operativo, dockerización, tipos de dispositivos utilizados y el adecuado particionado de memoria.

Responsables

- Desarrolladores
- Operaciones
- QA
- Seguridad informática

C.1. Accesos lógicos no autorizados

C.1.1. Gestión de cuentas y credenciales

- Definir y aplicar políticas de creación, administración y revocación de cuentas.
- Aplicar el principio de mínimo privilegio y *role-based access control* (RBAC).
- Implementar autenticación multifactor (MFA) en todos los accesos críticos.
- Utilizar gestores de secretos y rotación automática (KMS, Vault, OIDC).

C.1.2. Monitoreo y correlación de eventos

- Registrar eventos relevantes y centralizar los logs en un SIEM.
- Implementar detección de anomalías y alertas tempranas.
- Establecer retención segura de logs según política de cumplimiento.

C.2. Actualizaciones y ciclo de vida seguro

C.2.1. Política de actualizaciones y parches

- Mantener un inventario de componentes y dependencias (SBOM).
- Aplicar parches críticos en plazos establecidos según severidad.
- Asegurar builds reproducibles y aprobados antes de cualquier release.

C.2.2. Verificación post-actualización

- Realizar pruebas funcionales y de seguridad (DAST/SAST/IAST).
- Validar que no se introducen regresiones de seguridad.
- Monitorizar telemetría para identificar fallos en producción.

C.3. Conectividad a internet y gateway (Zero Trust Networking)

C.3.1. *Firewalls*, filtrado y segmentación

- Aplicar segmentación de red y microsegmentación para el IoT.
- Configurar *firewalls* con reglas de tráfico mínimo necesario.
- Implementar rate limiting e inspección TLS en gateways.

C.3.2. VPN y canales cifrados

- Utilizar VPN corporativa o túneles seguros para administración.
- Asegurar que todos los protocolos utilicen mTLS.

C.4. Sistema operativo y hardening

C.4.1. Selección segura

- Seleccionar sistemas operativos con soporte activo y comunidad madura.
- Evaluar distribuciones con mecanismos de seguridad avanzados (AppArmor, SELinux).

C.4.2. Hardening del sistema

- Deshabilitar servicios innecesarios y puertos no utilizados.
- Aplicar restricciones mediante MAC, namespaces y controles CIS.

C.5. Contenedores

C.5.1. Configuración segura de contenedores

- Aplicar aislamiento de recursos (cgroups, seccomp, capabilities).
- Construir imágenes minimalistas y firmadas (SLSA Level 2 o superior).
- Ejecutar contenedores sin privilegios y con usuarios no root.

C.5.2. Ciclo de vida de contenedores

- Escanear imágenes en cada build (Trivy, Grype).
- Usar runners efímeros en CI/CD con permisos mínimos.

C.6. Tipo y seguridad de los dispositivos IoT

C.6.1. Evaluación y requisitos de seguridad

- Verificar la autenticidad del hardware y sus componentes.
- Asegurar la protección física de interfaces de depuración (UART/JTAG).
- Implementar protección de datos locales mediante cifrado.
- Asegurar mecanismos de recuperación ante fallos (*fallback* o dual-bank).

C.7. Particionado de memoria y protección en dispositivos

C.7.1. Política de particionado

- Separar áreas críticas: bootloader, *firmware*, logs y configuración.
- Evitar ejecución en regiones de memoria no destinadas a código.

C.7.2. Optimización y seguridad

- Configurar memoria con políticas WX (Write XOR Execute).
- Proteger zonas de configuración frente a escritura no autorizada.

C.8. Revisiones, auditorías y mejora continua

C.8.1. Revisiones periódicas

- Revisar y actualizar este procedimiento cada 12 meses o tras incidentes.
- Validar la alineación con estándares actualizados y nuevas amenazas.

C.8.2. Auditorías internas y externas

- Verificar cumplimiento mediante métricas DevSecOps.
- Validar calidad de código, procesos y evidencias.

C.9. Acceso físico no autorizado

C.9.1. Control de acceso físico

- Restringir acceso a laboratorios, servidores y estaciones de desarrollo.
- Utilizar cámaras, cerraduras inteligentes y tarjetas de ingreso.

C.9.2. Registro de accesos

- Mantener registros inalterables (*tamper-resistant*).
- Revisar periódicamente eventos sospechosos.

C.10. Controles técnicos recomendados

- **Firma de *firmware* y *Secure Boot***: comprobación de integridad en arranque.
- ***SBOM* y *builds* reproducibles**: verificación de *hashes* y trazabilidad.
- **Protección de secretos**: uso de OIDC, GitHub Secrets y KMS.
- ***Hardening de runners***: *runners* efímeros, permisos mínimos.
- **Revisión y gates**: PR con aprobaciones, SAST, SCA y Sonar Quality Gates.
- **Seguridad física**: sellado de UART/JTAG y eFuses.
- **Red**: mTLS para MQTT, segmentación robusta, *firewall* con *rate limiting*.
- **Proveedor**: auditoría contractual, validación de firmas y SBOM.
- **Recuperación**: *dual-bank* y *rollback* automático en caso de OTA fallida.
- **Rotación de claves**: automatizada en CI/CD.

C.11. Checklist de verificación y criterios de aceptación

- Cada release incluye SBOM, firma criptográfica y evidencia de build reproducible.
- GitHub Actions no registra secretos en logs.
- Sonar Quality Gate aprobado para permitir merge.
- Dispositivos validan firma y ejecutan rollback seguro.
- Ausencia de credenciales embebidas (`secret scanning` / `cppcheck`).
- MQTT implementa TLS/mTLS; sin exposición pública del broker.
- Puertos UART/JTAG protegidos o sellados.
- Monitoreo de *firmware*.
- Actions Runner Controller (ARC) con runners efímeros.

C.12. Plantilla para reportar hallazgos en DefectDojo

Título: `OTA` *`firmware accepted without signature`*

Descripción: Resumen, pasos para reproducir, impacto, recomendación (bloquear OTA sin firma; habilitar rollback) y evidencias (pcap, logs, hash).

Severity: asignada según mapping Score → Severity.

References: SBOM, commit ID, GitHub Action run ID.